

CRAY

RESEARCH, INC.

CRAY PROPRIETARY

CRAY® COMPUTER SYSTEMS

**CRAY X-MP/1 SYSTEM
PROGRAMMER REFERENCE MANUAL**

CSM0111000

Copyright© 1987 by CRAY RESEARCH, INC. This manual or parts thereof may not be reproduced in any form without permission of CRAY RESEARCH, INC.

Each time this manual is revised and reprinted, all changes issued against the previous version are incorporated into the new version and the new version is assigned an alphabetic level.

Every page changed by a reprint with revision has the revision level in the lower righthand corner. Changes to part of a page are noted by a change bar in the margin directly opposite the change. A change bar in the margin opposite the page number indicates that the entire page is new. If the manual is rewritten, the revision level changes but the manual does not contain change bars.

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

CRAY RESEARCH, INC.
ATTENTION: Publications
890 Industrial Blvd.
Technical Operations Building
Chippewa Falls, WI 54729

<u>Revision</u>	<u>Description</u>
-----------------	--------------------

July 1984 - Original printing.

- | | |
|---|--|
| A | November 1985 - Reprint with revision. Memory addressing formats and Exchange Package changes for 64K-chip systems and 8-million-word memories were added. The Gather/scatter, compressed index, and the 24-bit immediate constant load instructions were also added. Numerous editorial changes and corrections were also made. This revision obsoletes all previous printings. |
| B | August 1986 - Reprint with revision. Corrections were made to the SECEDED maintenance mode function statement documentation and the underflow detection statement. Additions were made to the scalar reference hold issue condition and the vector load information. The clock period (CP) time on serial number 326 and above has been upgraded from 9.5-ns to 8.5-ns. Numerous technical and editorial changes and corrections were also made. This revision obsoletes all previous printings. |

June 1987 - This manual has been converted to a system programmer reference manual to protect Cray propriety information. When ordering this manual, use publication number CSM0111000.

CRAY, CRAY-1, and SSD are registered trademarks and APLM, CFT, CFT77, CFT2, COS, CRAY-2, CRAY X-MP, CSIM, IOS, SEGLDR, SID, SUPERLINK/ISP, and UNICOS are trademarks of Cray Research, Inc. The UNICOS system is derived from the AT&T UNIX system; UNIX is a registered trademark of AT&T.

PREFACE

This manual describes the functions of CRAY X-MP series single-processor 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 equipment. It also describes the operation of the Central Processing Unit (CPU) that executes instructions, provides memory protection, and reports hardware exceptions within the computer systems.

The following publications give details of the I/O Subsystem (IOS), the disk storage units (DSUs), and the SSD solid-state storage device:

- HR-0030 I/O Subsystem Hardware Reference Manual
- HR-0031 Solid-state Storage Device Hardware Reference Manual
- HR-0630 Mass Storage Subsystem Hardware Reference Manual
- HR-0077 Disk Systems 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. Operation 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

<u>PREFACE</u>	iii
1. <u>SYSTEM DESCRIPTION</u>	1-1
CONVENTIONS	1-4
Italics	1-4
Register conventions	1-4
Number conventions	1-4
Clock period	1-4
SYSTEM COMPONENTS	1-5
Central Processing Unit	1-5
Interfaces	1-7
I/O Subsystem	1-8
Disk storage units	1-10
Solid-state Storage Device	1-11
Condensing units	1-12
Power distribution units	1-13
Motor-generator units	1-14
SYSTEM CONFIGURATION	1-15
2. <u>CPU RESOURCES</u>	2-1
CENTRAL MEMORY	2-1
Memory organization	2-2
Memory addressing	2-3
Memory addressing for 16-bank, 16K-chip, 1- and 2-million-word system	2-3
Memory addressing for 32-bank, 16K-chip, 4-million- word system	2-3
Memory addressing for 16-bank, 64K-chip, 4-million- word system	2-3
Memory addressing for 32-bank, 64K-chip, 8-million- word system	2-3
Memory access	2-5
Conflict resolution	2-7
Memory access priorities	2-7
16-bank phasing	2-7
Memory error correction	2-8

2. CPU RESOURCES (continued)

INTER-CPU COMMUNICATION SECTION	2-10
Real-time clock	2-10
Inter-CPU communication and control	2-11
Shared Address and Shared Scalar registers	2-12
Semaphore registers	2-12
CPU INPUT/OUTPUT SECTION (Maximum Configuration)	2-13
Data transfer for Solid-state Storage Device	2-14
Data transfer for I/O Subsystem	2-14
6 Mbyte per second channels	2-15
6 Mbyte per second channel operation	2-15
Input channel programming	2-17
Input channel error conditions	2-18
Output channel programming	2-18
Programmed Master Clear to external device	2-19
Memory access	2-19
I/O lockout	2-22
Memory bank conflicts	2-22
I/O memory conflicts	2-22
I/O memory request conditions	2-23
I/O memory addressing	2-23

3. CPU CONTROL SECTION 3-1

INSTRUCTION ISSUE AND CONTROL	3-1
Program Address register	3-2
Next Instruction Parcel register	3-2
Current Instruction Parcel register	3-2
Lower Instruction Parcel register	3-3
Instruction buffers	3-3
EXCHANGE MECHANISM	3-5
Exchange Package	3-5
Processor number	3-7
Memory error data	3-7
Program Address register	3-8
Memory field registers	3-8
Mode register	3-8
Vector not used (VNU)	3-9
Enable second vector logical (ESVL)	3-10
Flag register	3-11
Exchange Address register	3-12
Enhanced addressing mode (EAM)	3-12
Data Base Address register	3-13
Program State register	3-13
Cluster Number register	3-13
Data Limit Address register	3-13
A registers	3-13
S registers	3-13
Active Exchange Package	3-13

EXCHANGE MECHANISM (continued)	
Exchange sequence	3-14
Exchange initiated by deadstart sequence	3-14
Exchange initiated by Interrupt flag set	3-14
Exchange initiated by program exit	3-14
Exchange sequence issue conditions	3-15
Exchange Package management	3-15
MEMORY FIELD PROTECTION	3-16
Instruction Base Address register	3-17
Instruction Limit Address register	3-17
Data Base Address register	3-17
Data Limit Address register	3-18
Program range error	3-19
Operand range error	3-19
PROGRAMMABLE CLOCK	3-19
Instructions	3-19
Interrupt Interval register	3-20
Interrupt Countdown counter	3-20
Clear programmable clock interrupt request	3-21
PERFORMANCE MONITOR	3-21
DEADSTART SEQUENCE	3-21

4. CPU COMPUTATION SECTION 4-1

OPERATING REGISTERS	4-3
ADDRESS REGISTERS	4-3
A registers	4-3
B registers	4-6
SCALAR REGISTERS	4-6
S registers	4-7
T registers	4-9
VECTOR REGISTERS	4-9
V registers	4-9
V register reservations and chaining	4-12
Vector control registers	4-13
Vector Length register	4-13
Vector Mask register	4-13
FUNCTIONAL UNITS	4-14
Address functional units	4-14
Address Add functional unit	4-15
Address Multiply functional unit	4-15
Scalar functional units	4-15
Scalar Add functional unit	4-15
Scalar Shift functional unit	4-16
Scalar Logical functional unit	4-16
Scalar Population/Parity/Leading Zero functional unit	4-16
Vector functional units	4-16
Vector functional unit reservation	4-17

FUNCTIONAL UNITS (continued)	
Vector Add functional unit	4-17
Vector Shift functional unit	4-17
Full Vector Logical functional unit	4-18
Second Vector Logical functional unit	4-18
Vector Population/Parity functional unit	4-19
Floating-point functional units	4-19
Floating-point Add functional unit	4-20
Floating-point Multiply functional unit	4-20
Reciprocal Approximation functional unit	4-21
ARITHMETIC OPERATIONS	4-21
Integer arithmetic	4-21
Floating-point arithmetic	4-22
Normalized floating-point numbers	4-24
Floating-point range errors	4-24
Double-precision numbers	4-28
Addition algorithm	4-28
Multiplication algorithm	4-29
Division algorithm	4-31
Derivation of the division algorithm	4-31
LOGICAL OPERATIONS	4-36

5. <u>CPU INSTRUCTIONS</u>	5-1
INSTRUCTION FORMAT	5-1
1-parcel instruction format with discrete <i>j</i> and <i>k</i> fields .	5-1
1-parcel instruction format with combined <i>j</i> and <i>k</i> fields .	5-2
2-parcel instruction format with combined <i>j</i> , <i>k</i> ,	
and <i>m</i> fields	5-3
2-parcel instruction format with combined <i>i</i> , <i>j</i> , <i>k</i> ,	
and <i>m</i> fields	5-4
SPECIAL REGISTER VALUES	5-5
INSTRUCTION ISSUE	5-5
INSTRUCTION DESCRIPTIONS	5-6

APPENDIX SECTION

A. <u>INSTRUCTION SUMMARY FOR CRAY X-MP SINGLE-PROCESSOR COMPUTER SYSTEMS</u>	A-1
B. <u>6 MBYTE PER SECOND CHANNEL DESCRIPTIONS</u>	B-1
6 MBYTE PER SECOND INPUT CHANNEL SIGNAL SEQUENCE	B-1
Data bits 2^0 through 2^{15}	B-1
Parity bits 0 through 3	B-2
Ready signal	B-3
Resume signal.	B-3
Disconnect signal	B-3

B. 6 MBYTE PER SECOND CHANNEL DESCRIPTIONS (continued)

6 MBYTE PER SECOND OUTPUT CHANNEL SIGNAL SEQUENCE	B-3
Data bits 2^0 through 2^{15}	B-4
Parity bits 0 through 3	B-5
Ready signal	B-5
Resume signal	B-5
Disconnect signal	B-5

C. PERFORMANCE MONITOR C-1

SELECTING PERFORMANCE EVENTS	C-1
READING PERFORMANCE RESULTS	C-3
TESTING PERFORMANCE COUNTERS	C-4

D. SECDED MAINTENANCE FUNCTIONS D-1

VERIFICATION OF CHECK BIT STORAGE	D-1
VERIFICATION OF CHECK BIT GENERATION	D-2
VERIFICATION OF ERROR DETECTION AND CORRECTION	D-2
CLEARING MAINTENANCE MODE FUNCTIONS	D-3

FIGURES

1-1	CRAY X-MP Model 11, 12, 14, or 18 with a Cray I/O Subsystem .	1-2
1-2	Basic Organization of the Single-processor System	1-5
1-3	Control and Datapaths for the CPU	1-6
1-4	Typical Interface Cabinet	1-8
1-5	I/O Subsystem Chassis	1-9
1-6	Solid-state Storage Device Chassis	1-11
1-7	Condensing Unit	1-12
1-8	Power Distribution Units	1-13
1-9	Motor-generator Equipment	1-14
1-10	Block Diagram of a Typical CRAY X-MP Single-processor System with Full Disk Capacity	1-15
1-11	Block Diagram of a Typical CRAY X-MP Single-processor System with Block Multiplexer Channels	1-16
2-1	Central Memory Organization for a Single-processor System . .	2-2
2-2	Memory Datapath with SECDED	2-8
2-3	Error Correction Matrix	2-9
2-4	Shared Registers and Real-time Clock	2-11
2-5	Basic I/O Program Flowchart	2-17
2-6	Channel I/O Control	2-20
2-7	Input/Output Datapaths	2-21
3-1	Instruction Issue and Control Elements	3-1
3-2	Instruction Buffers	3-3
3-3	Exchange Package for a Four-processor System	3-6
4-1	Address Registers and Functional Units	4-4

FIGURES (continued)

4-2	Scalar Registers and Functional Units	4-7
4-3	Vector Registers and Functional Units	4-10
4-4	Integer Data Formats	4-22
4-5	Floating-point Data Format	4-22
4-6	Internal Representation of Floating-point Number (Octal) . .	4-23
4-7	Exponent Matrix for Floating-point Multiply Unit	4-26
4-8	Integer Multiply in Floating-point Multiply Functional Unit	4-28
4-9	49-bit Floating-point Addition	4-28
4-10	Floating-point Multiply Partial-product Sums Pyramid	4-30
4-11	Newton's Method	4-32
5-1	General Form for Instructions	5-1
5-2	1-parcel Instruction Format with Discrete <i>j</i> and <i>k</i> Fields . .	5-2
5-3	1-parcel Instruction Format with Combined <i>j</i> and <i>k</i> Fields . .	5-2
5-4	2-parcel Instruction Format with Combined <i>j</i> , <i>k</i> , and <i>m</i> Fields	5-3
5-5	2-parcel Instruction Format with Combined <i>i</i> , <i>j</i> , <i>k</i> , and <i>m</i> Fields	5-4
5-6	2-parcel Instruction Format for a 24-bit Immediate Constant with Combined <i>i</i> , <i>j</i> , <i>k</i> , and <i>m</i> Fields	5-4
5-7	Vector Left Double Shift, First Element, VL Greater than 1 .	5-72
5-8	Vector Left Double Shift, Second Element, VL Greater than 2 .	5-72
5-9	Vector Left Double Shift, Last Element	5-73
5-10	Vector Right Double Shift, First Element	5-74
5-11	Vector Right Double Shift, Second Element, VL Greater than 1	5-75
5-12	Vector Right Double Shift, Last Operation	5-75

TABLES

1-1	CRAY X-MP Single-processor System Characteristics	1-3
2-1	Memory Addressing Formats	2-4
2-2	Channel Word Assembly/Disassembly	2-16
5-1	Special Register Values	5-5
B-1	Input Channel Signal Exchange	B-2
B-2	Output Channel Signal Exchange	B-4
C-1	Performance Counter Group Descriptions	C-2

INDEX

CRAY X-MP single-processor computer systems are powerful, general purpose single-processor computer systems. They are able to achieve extremely high processing rates by efficiently using the scalar and vector capabilities of the Central Processing Unit (CPU) combined with the systems' solid-state, random-access memory (RAM), and registers.

Vector processing is the performance of iterative operations on sets of ordered data. When two or more vector operations are chained together, two or more operations can be executing simultaneously; therefore, the computational rates for vector processing greatly exceed the computational rates of conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adaptable to vector techniques.

Equipment options allow the systems to be configured for a particular use (refer to table 1-1). Central Memory of the single-processor mainframe can be 1 million (model 11), 2 million (model 12), 4 million (model 14), or 8 million (model 18) 64-bit words. The system is compatible with all existing models of the Cray I/O Subsystem (IOS), which matches the mainframe's processing rates with high input/output (I/O) transfer rates for communication with mass storage units, other peripheral devices, and a wide variety of host computers.

In addition to the mainframe and IOS, a Cray Research, Inc. (CRI) SSD Solid-state Storage Device can be configured with the system. An SSD provides significantly improved throughput of programs that access large data files repetitively. Figure 1-1 shows the mainframe configured with a Cray IOS and an SSD.

This section describes system components and configurations. Table 1-1 gives overall system characteristics.

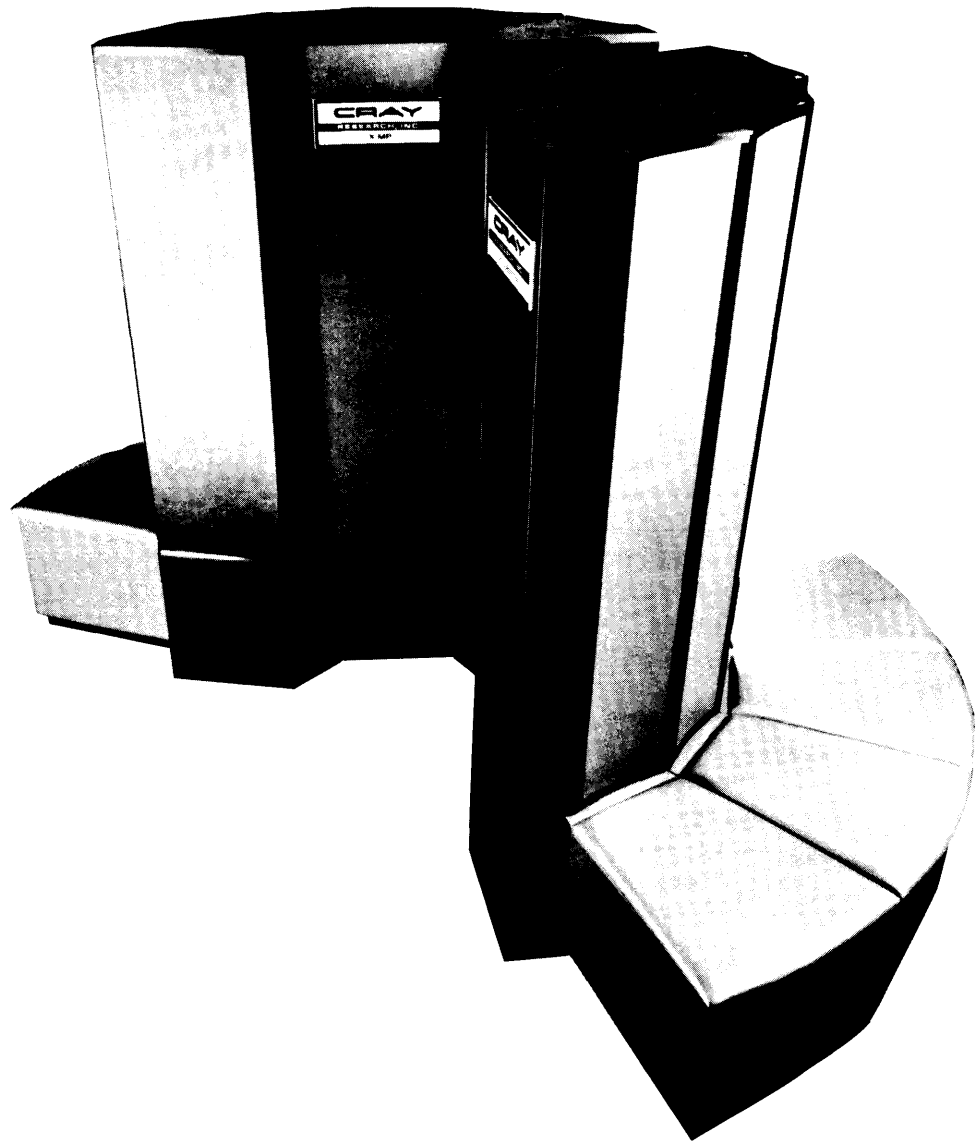


Figure 1-1. CRAY X-MP Model 11, 12, 14 or 18 with a Cray I/O Subsystem

Table 1-1. CRAY X-MP Single-processor System Characteristics

Configuration	<ul style="list-style-type: none"> • One CPU • IOS with 2, 3, or 4 I/O Processors (IOPs) • Optional SSD
8.5-ns clock CPU speed	<ul style="list-style-type: none"> • 8.5-ns CPU CP • 117 million floating-point additions per second • 117 million floating-point multiplications per second • 117 million half-precision, floating-point divisions per second • 37 million full-precision, floating-point divisions per second • Simultaneous floating-point addition, multiplication, and reciprocal approximation
9.5-ns clock CPU speed	<ul style="list-style-type: none"> • 9.5-ns CPU CP • 105 million floating-point additions per second per CPU • 105 million floating-point multiplications per second per CPU • 105 million half-precision, floating-point divisions per second per CPU • 33 million full-precision, floating-point divisions per second per CPU • Simultaneous floating-point addition, multiplication, and reciprocal approximation within each CPU
Memories	<ul style="list-style-type: none"> • Mainframe has 1 million (model 11), 2 million (model 12), 4 million (model 14), or 8 million (model 18) 64-bit words in Central Memory
Input/Output	<ul style="list-style-type: none"> • 1250 Mbytes per second channel pair to interface an SSD to the mainframe • Up to two 100 Mbyte per second channel pairs for interface to an IOS • Up to four 6 Mbyte per second channel pairs
Physical	<ul style="list-style-type: none"> • 19 sq ft (1.76 m²) floor space for the mainframe • 15 sq ft (1.39 m²) floor space for the IOS • 15 sq ft (1.39 m²) floor space for the SSD • 2.625 tons (2.38 Mg), mainframe weight • 1.5 tons (1.36 Mg), IOS weight • 1.5 tons (1.36 Mg), SSD weight • Liquid refrigeration of each chassis • 400-Hz power from motor-generators

CONVENTIONS

This manual uses the following conventions.

ITALICS

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

REGISTER CONVENTIONS

Parenthesized register names are used frequently 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 register P.

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

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 bit 0 as the most significant and bit 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 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, which are given in octal without the subscript.

CLOCK PERIOD

The basic unit of CPU computation time is the clock period (CP). For mainframes with serial numbers 406 and above, the CP is 8.5-ns. For mainframes with serial numbers 405 and below, the CP is 9.5-ns. Instruction issue, memory references, and other timing considerations are often measured in CPs.

SYSTEM COMPONENTS

The system is composed of a mainframe and an IOS. Mass storage devices, front-end interfaces, and optional tape devices are also integral parts of a system. Optionally, a Cray SSD can be part of the system. Supporting this equipment are condensing units for refrigeration, motor-generators to provide system power, and power distribution units for the mainframe, the IOS, and the SSD. The following pages describe the system components.

CENTRAL PROCESSING UNIT

The CPU for the single-processor CRAY X-MP is an integrated processing unit which has a memory section, a control section, a computation section, an inter-CPU communication section, and an I/O section. (CPU sections are described later.) Figure 1-2 shows the basic organization of the computer. Figure 1-3 shows the components and control and datapaths of the CPU.

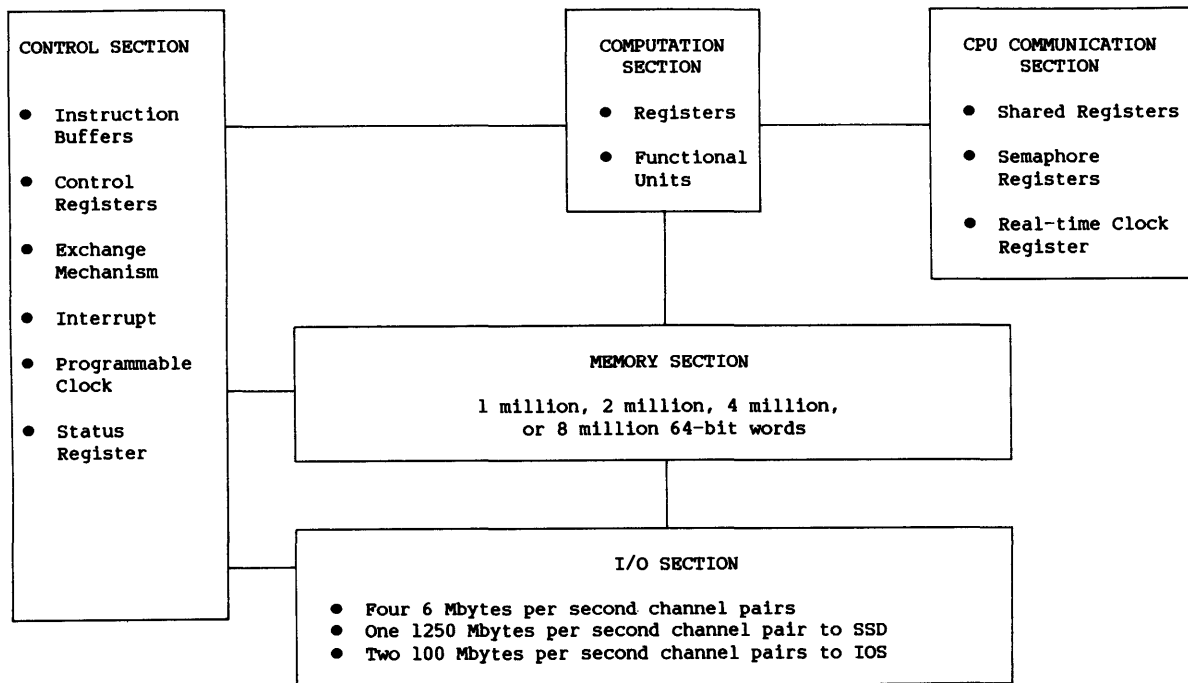


Figure 1-2. Basic Organization of the Single-processor System

INTERFACES

The Cray mainframe is designed for use with front-end computers in a computer network. A front-end computer system is self-contained and executes under the control of its own operating system.

Standard interfaces connect the Cray mainframe's I/O channels to channels of front-end computers, providing input data to the Cray 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 front-end computer system can be connected either to the Master I/O Processor (MIOP) of the IOS or to the mainframe.

The front-end interface is housed in a stand-alone cabinet (figure 1-4) located near the host computer. Its operation is invisible to both the front-end computer user and the Cray user.

A primary goal of the interface is to maximize the use of the front-end channel connected to the Cray system. Since the MIOP channel connected to the interface is faster than any front-end channel connected to the interface, the burst rate of the interface is limited by the maximum rate of the front-end channel.

Interfaces to front-end computers allow the front-end computers to service the Cray mainframe 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 channel
- As a remote batch entry station
- As an interactive communication station

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

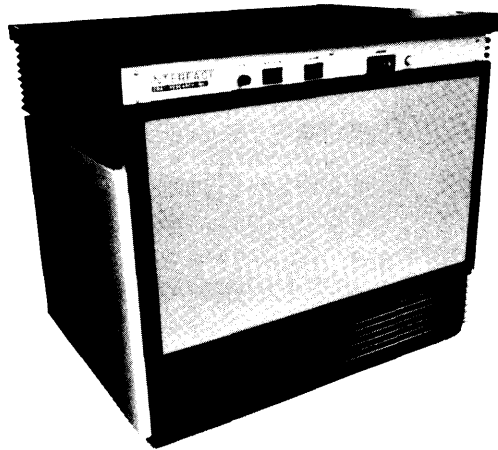


Figure 1-4. Typical Interface Cabinet

I/O SUBSYSTEM

The IOS, shown in figure 1-5, is standard on all CRAY X-MP series computer systems and has two, three, or four IOPs, a Buffer Memory, and required interfaces. It is designed for fast data transfer between front-end computers, peripheral devices, storage devices, and the IOS's Buffer Memory or between its Buffer Memory and the Central Memory of a Cray mainframe.

Four types of IOPs may be configured in an IOS: an MIOP, a Buffer IOP (BIOP), a Disk IOP (DIOP), and an Auxiliary IOP (XIOP). All IOSs must have at least one MIOP and one BIOP. The number of DIOPs and XIOPs is site dependent.

Each IOP of the IOS has a memory section, a control section, a computation section, and an I/O section. I/O sections are independent and handle some portion of the I/O requirements for the subsystem. Each IOP also has six direct memory access (DMA) ports to its Local Memory.

The MIOP controls the front-end interfaces and the standard group of station† peripherals. The Peripheral Expander interfaces the station peripherals to one DMA port of the MIOP. The MIOP also connects to Buffer Memory and to the mainframe over a 6 Mbyte per second channel pair.

† The term station means both hardware and software. Station is the link to the front-end system or can act as a limited front-end system (as the MIOP).

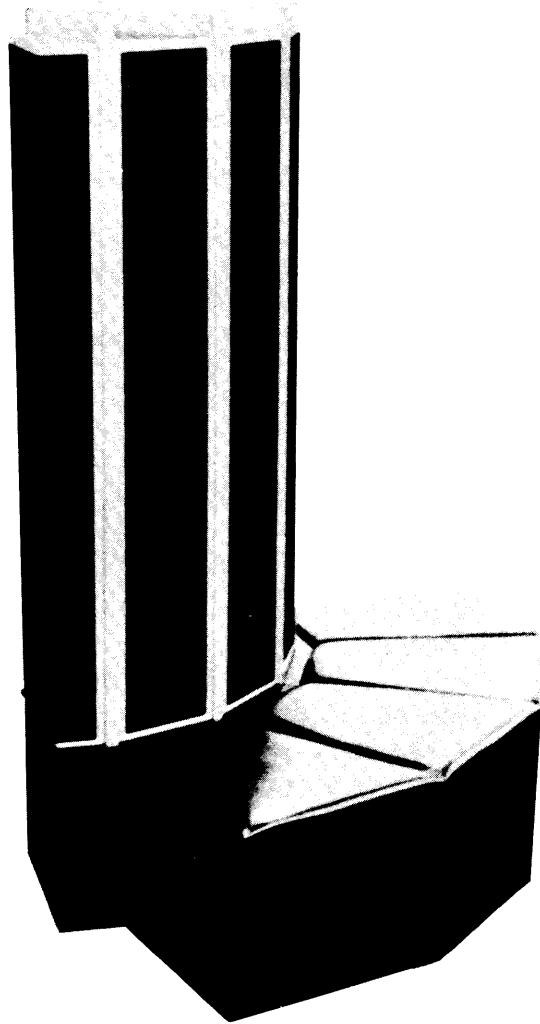


Figure 1-5. I/O Subsystem Chassis

The BIOP is the main link between the mainframe's Central Memory and the mass storage devices. Data from mass storage is transferred through the BIOP's Local Memory to the mainframe's Central Memory through a 100 Mbyte per second channel pair.

The DIOP is used for additional disk storage units (DSUs). This processor can handle up to four disk controller units (DCUs) with up to 16 disk storage units. The DIOP uses one DMA port for each controller, one DMA port to connect to Buffer Memory, and another DMA port to connect a 100 Mbyte per second channel pair to the mainframe Central Memory.

The XIOP is used for block multiplexer channels and interfaces to a maximum of four BMC-4 Block Multiplexer Controllers. Each controller can handle up to four block multiplexer channels. The XIOP uses one DMA port for each controller and another DMA port to connect with Buffer Memory.

IOS hardware allows simultaneous data transfers between the MIOP, BIOP, DIOP, or XIOP of the IOS and the mainframe's Central Memory.†

Section 2 describes the CPU I/O section for the Cray System. Refer to the I/O Subsystem Hardware Reference Manual for a complete description of the IOS.

DISK STORAGE UNITS

For mass storage, the system uses CRI disk storage units. A disk controller unit interfaces the disk storage units with an IOP of an IOS through one DMA port. Up to four disk storage units can be connected to a single disk controller unit.

The IOP and the disk controller unit can transfer data between the DMA port and four disk storage units with all disk storage units operating at full speed without missing data or skipping revolutions. A minimum of 2 and a maximum of 48 disk storage units can be configured on an IOS. The IOS chassis houses the disk controller unit.

Each disk storage unit has two accesses for connecting it to controllers. The second independent datapath to each disk storage unit exists through another CRI controller. Reservation logic provides controlled access to each disk storage unit. The Cray operating system COS software does not support dynamic sharing of devices. The Disk Systems Hardware Reference Manual includes further information about the mass storage subsystem.

† Software to support the 100 Mbyte per second channel pair to the MIOP and XIOP is not currently available.

SOLID-STATE STORAGE DEVICE

The SSD, shown in figure 1-6, is used for temporary data storage. A special Cray interface cable, set at a maximum speed of 1250 Mbytes per second, transfers data between the mainframe's Central Memory and the SSD. Refer to the SSD Solid-state Storage Device Hardware Reference Manual for more information about the SSD to mainframe channel connection.

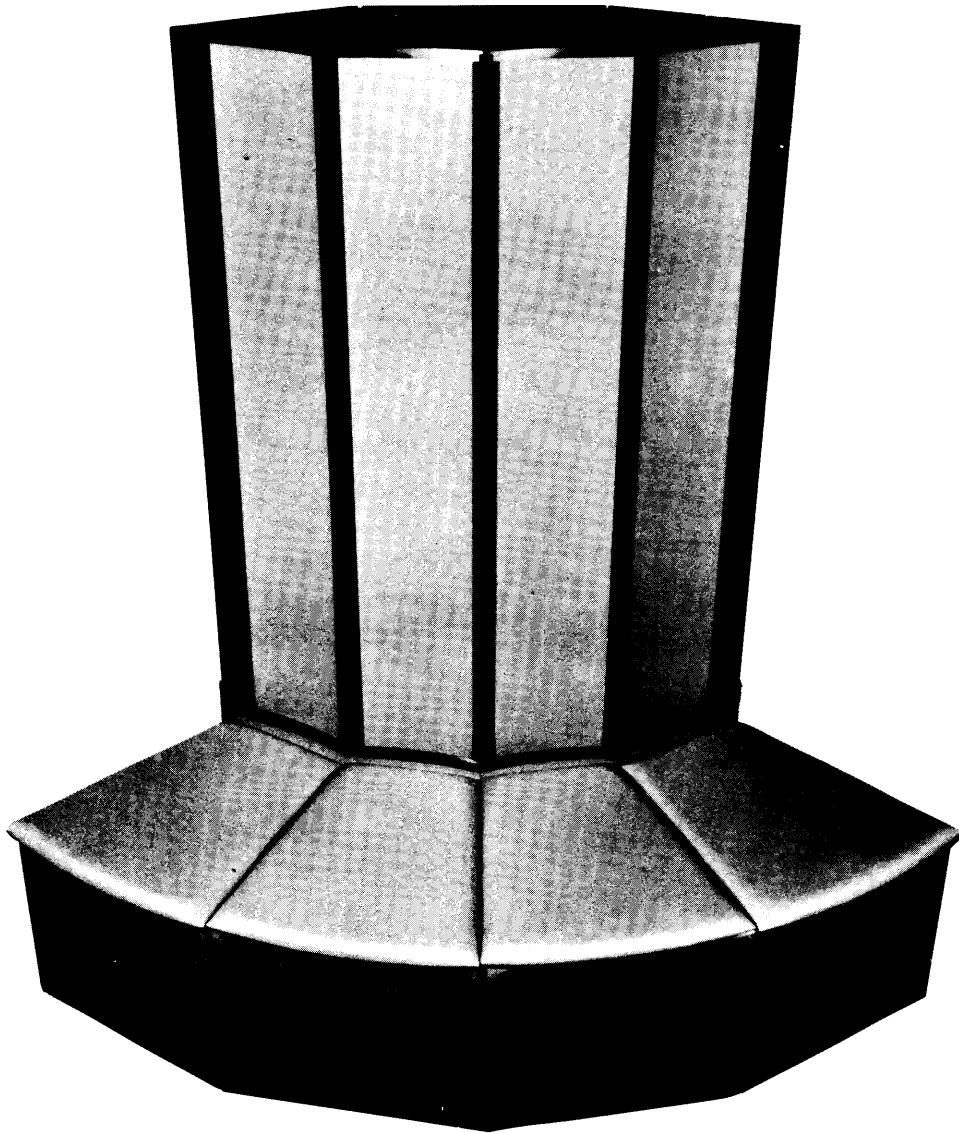


Figure 1-6. Solid-state Storage Device Chassis

CONDENSING UNITS

Condensing units (figure 1-7) 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 computer system. Freon, which cools the computer, picks up heat and transfers it to water in the condensing unit.

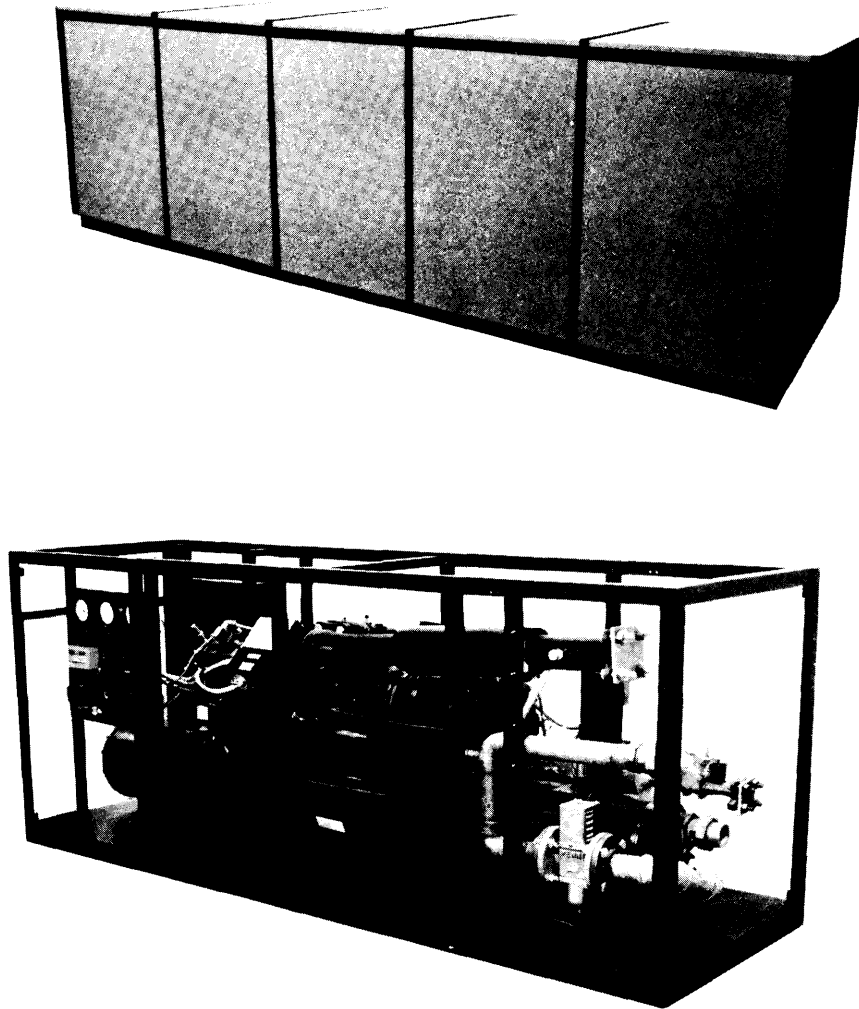


Figure 1-7. Condensing Unit

POWER DISTRIBUTION UNITS

The Cray mainframe, IOS, and SSD all operate from 400-Hz, three-phase power. The mainframe and IOS operate from the same power distribution unit. This unit contains adjustable transformers for regulating the voltage to each chassis column.

The power distribution unit also contains temperature and voltage monitoring equipment that checks temperatures at strategic locations on the chassis columns, and automatic warning and shutdown circuitry to protect the equipment in case of overheating or excessive cooling. Control switches for the motor-generators and the condensing unit are also mounted on the power distribution unit.

A smaller power distribution unit performs similar functions for the SSD chassis.

Figure 1-8 shows the power distribution units for the mainframe and IOS (left) and for the SSD (right).

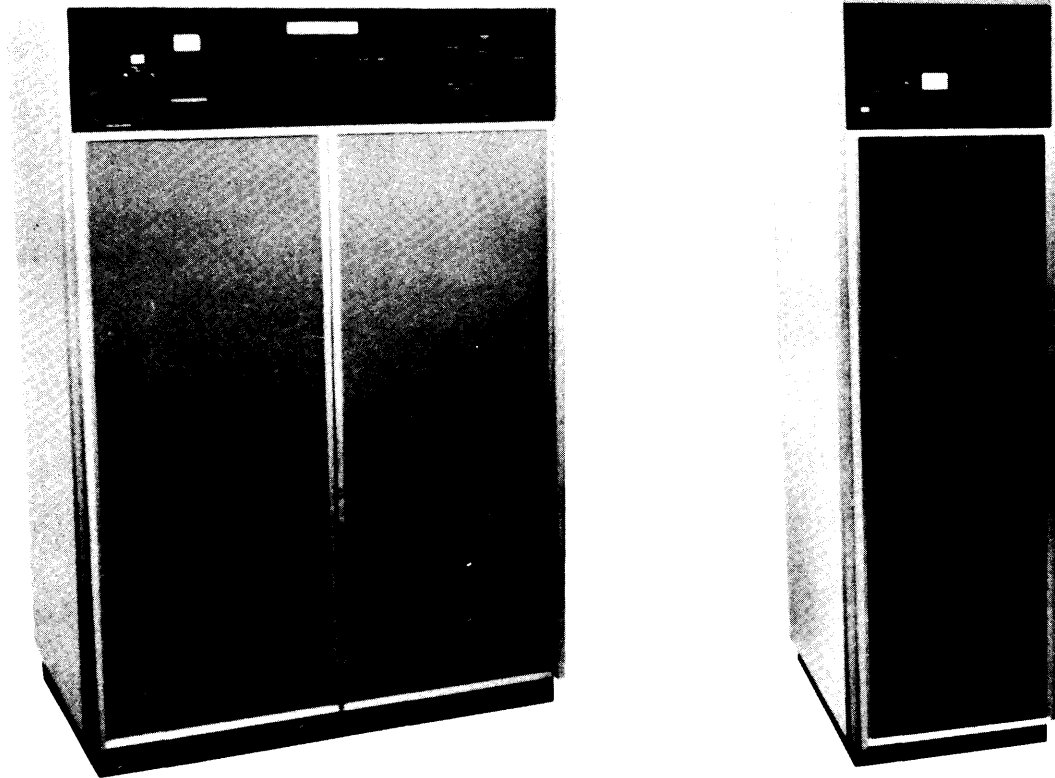


Figure 1-8. Power Distribution Units

MOTOR-GENERATOR UNITS

Motor-generator units convert primary power from the commercial power mains to the 400-Hz power used by the 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 1-9 shows a typical motor-generator and its control cabinet.

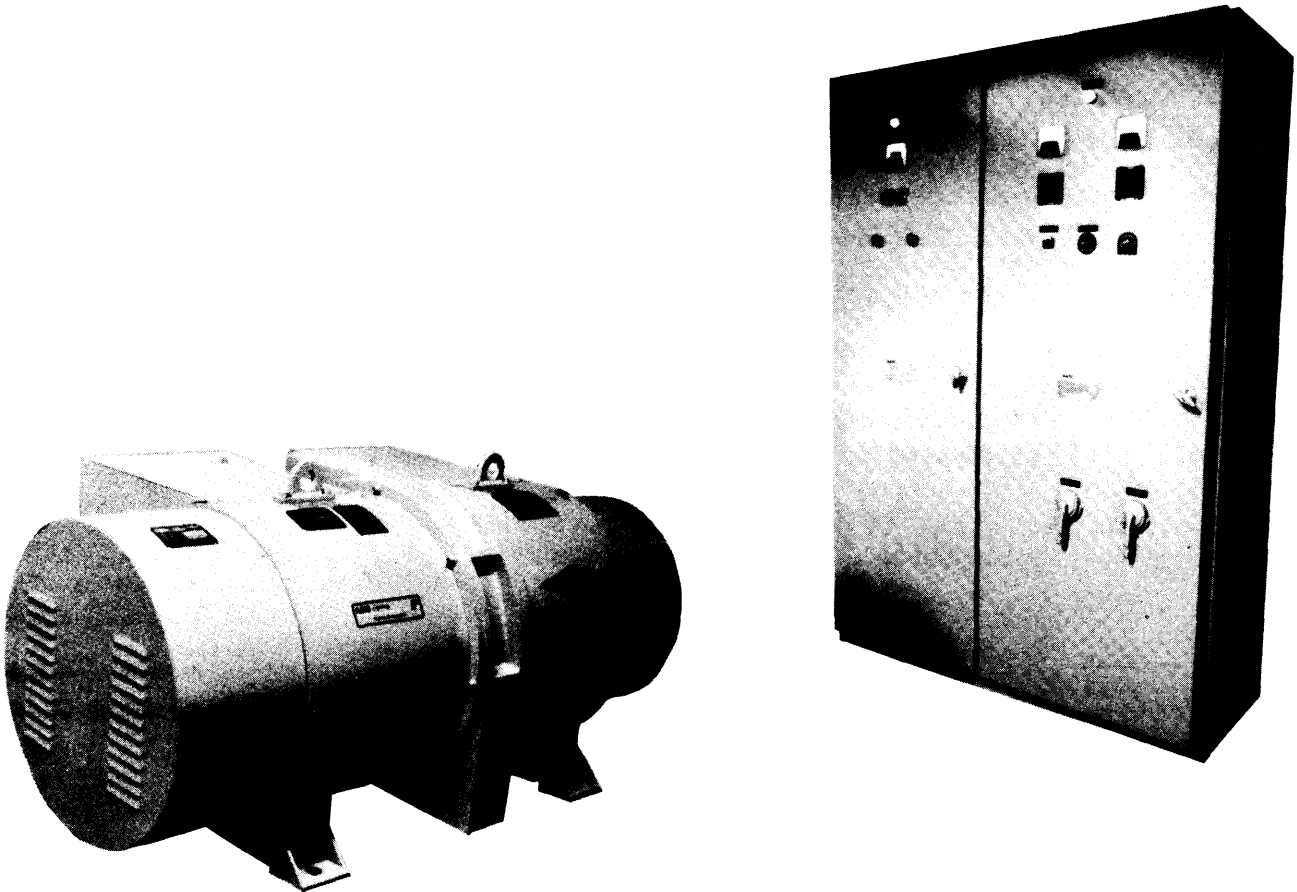


Figure 1-9. Motor-generator Equipment

SYSTEM CONFIGURATION

Figures 1-10 and 1-11 illustrate two configurations for the CRAY X-MP single-processor computer systems.

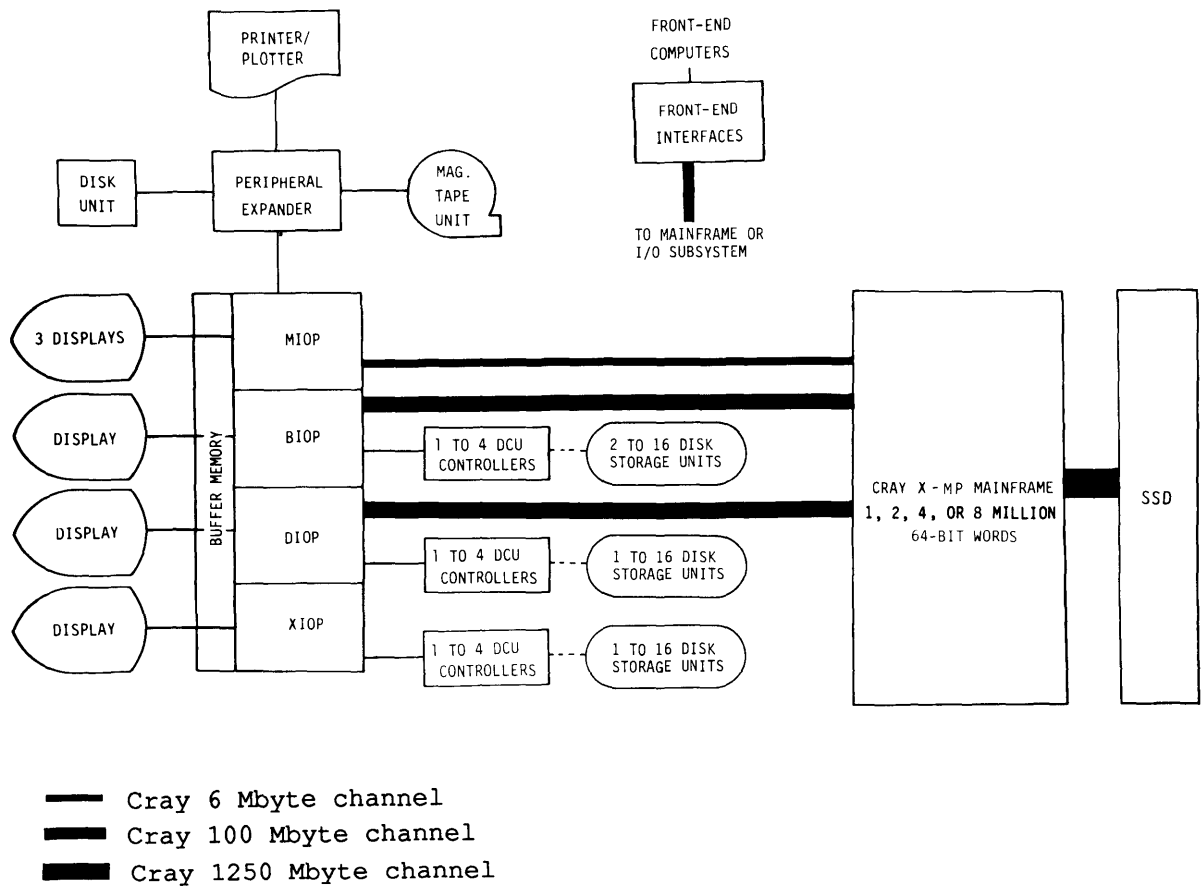


Figure 1-10. Block Diagram of a Typical CRAY X-MP Single-processor System with Full Disk Capacity

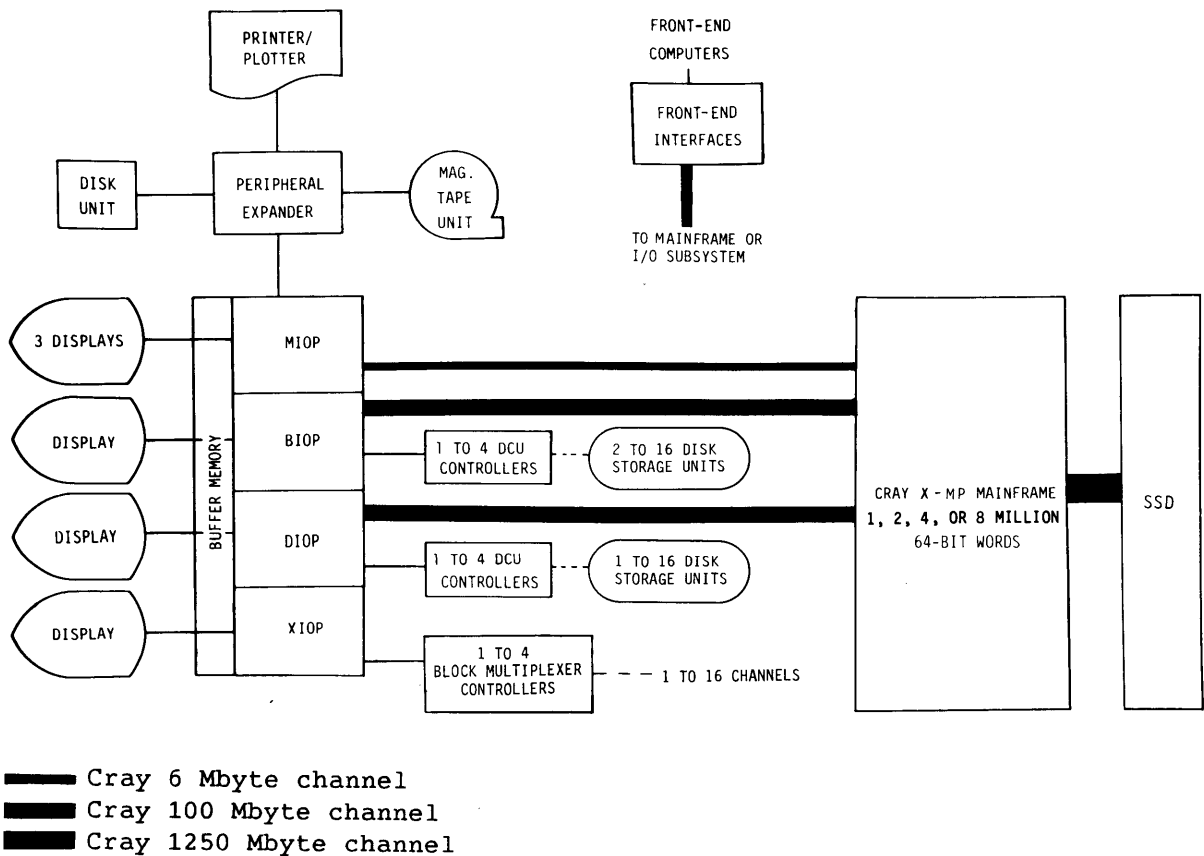


Figure 1-11. Block Diagram of a Typical CRAY X-MP Single-processor System with Block Multiplexer Channels

The Central Processing Unit (CPU) has access to the mainframe's Central Memory, the inter-CPU communication section, and the I/O section. The following pages describe these areas.

CENTRAL MEMORY

Central Memory consists of a number of banks of solid-state, random-access memory (RAM) that is shared by the CPU and the I/O section. Four Central Memory sizes are available with either 16K- or 64K-chip technology: 1 or 2 million words in 16 banks (16K), 4 million words in 32 banks (16K), 4 million words in 16 banks (64K), and 8 million words in 32 banks (64K). Banks are independent of each other; sequentially addressed words reside in sequential banks. Each word is 72 bits with 64 data bits and 8 check bits.

Central Memory cycle time takes 8 clock periods (CPs) to execute. Access time, the time required to fetch an operand from Central Memory to an operating register, is 17 CPs for address (A) and scalar (S) registers. Access time is 20 CPs plus vector length for a vector (V) register and 19 CPs plus block length for a block transfer to a intermediate address (B) or intermediate scalar (T) register.

The maximum transfer rate for B, T, and V registers is 3 words per CP; for A and S registers, it is 1 word every 2 CPs. Transfer of instructions to instruction buffers occurs at a rate of 32 parcels (8 words) per CP. For the I/O section, the transfer rate is 2 words per CP.

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

- 1, 2, 4, or 8 million words of integrated circuit memory
- 64 data bits and 8 error-correction bits per word
- 16 or 32 interleaved banks
- 8-CP bank cycle time
- Single-error-correction/double-error-detection (SEDED)
- 3 words per CP transfer rate to B, T, and V registers
- 1 word per 2-CP transfer rate to A and S registers
- 8 words per CP transfer rate to instruction buffers
- 2 words per CP transfer rate to I/O concurrent with all memory activity except instruction fetch and exchange

MEMORY ORGANIZATION

Memory is organized to provide fast, efficient access for the CPU. Data transfers to and from memory are corrected with SECDED. Central Memory is organized into four sections with 4 or 8 banks in each section. The 16-bank phasing is standard for a 1- or 2-million-word system (16K) and 4-million-word system (64K); 32-bank phasing is standard for a 4-million-word system (16K) and a 8-million-word system (64K).

As shown in figure 2-1, the CPU has an independent access path into each of the four memory sections. For I/O and instruction fetch operations, an additional access path into each section of memory is provided (dashed lines in figure 2-1). These additional access paths allow instruction fetches to proceed at 8 words per CP and I/O to reference 2 words per CP.

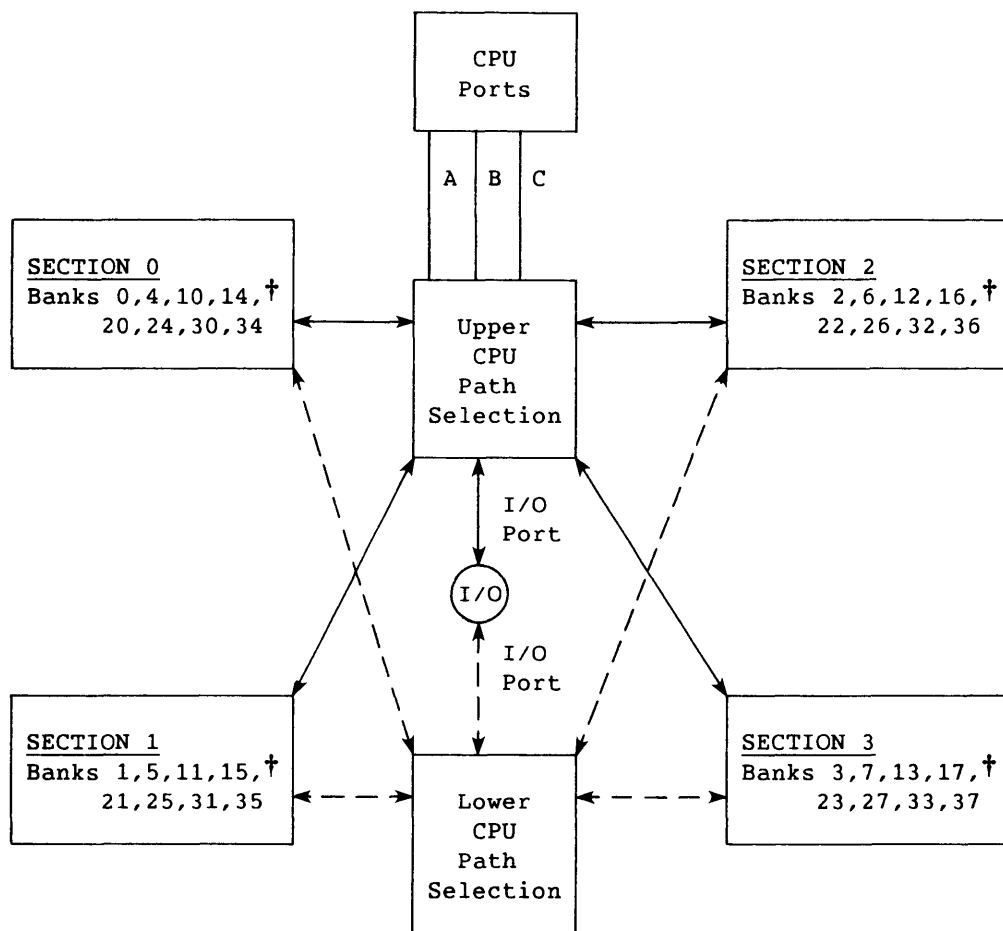


Figure 2-1. Central Memory Organization for a Single-processor System

† Low-numbered 4 banks in each section are in a 16-bank system.

MEMORY ADDRESSING

Memory addressing is dependent on system memory architecture (chip size and number of banks) and memory size. The following paragraphs describe the memory addressing for the different configurations of the single-processor system.

Memory addressing for 16-bank, 16K-chip, 1- and 2-million-word system

A word in a 16-bank, 16K-chip memory is addressed in a maximum of 21 bits, as shown in table 2-1. The low-order 4 bits specify one of the 16 banks. The next 14-bit field specifies an address within the chip. The high-order 3 bits specify one chip on a module.†

Memory addressing for 32-bank, 16K-chip, 4-million-word system

A word in a 32-bank, 16K-chip memory is addressed in a maximum of 22 bits, as shown in table 2-1. The low-order 5 bits specify one of the 32 banks. The next 14-bit field specifies an address within the chip. The high-order 3 bits specify one chip on the module.

Memory addressing for 16-bank, 64K-chip, 4-million-word system

A word in a 16-bank, 64K-chip memory is addressed in a maximum of 22 bits, as shown in table 2-1. The low-order 4 bits specify one of the 16 banks. The next 16-bit field specifies an address within the chip. The high-order 2 bits specify one chip on the module.†

Memory addressing for 32-bank, 64K-chip, 8-million-word system

A word in a 32-bank, 64K-chip memory is addressed in a maximum of 23 bits, as shown in table 2-1. The low-order 5 bits specify one of the 32 banks. The next 16-bit field specifies an address within the chip. The high-order 2 bits specify one chip on the module.

† Hardware assembles the address using a 4-bit bank field. The software, when assembling the address for memory error correction, receives 5 significant bits from the Exchange Package. The high-order bit (bit 4 counting right to left from 0) must be discarded by the software when assembling the address for memory error correction.

Table 2-1. Memory Addressing Formats

Chip Type	Memory Size	No. of Banks	No. of Columns	Address Format
16K	1 or 2	16	6	2^{20} 2^{17} 2^3 2^0 <div style="border: 1px solid black; padding: 5px; width: fit-content;"> Chip address select Internal bit address in chip 4-bit bank </div>
16K	4	32	6	2^{21} 2^{18} 2^4 2^0 <div style="border: 1px solid black; padding: 5px; width: fit-content;"> Chip address select Internal bit address in chip 5-bit bank </div>
64K	4	16	6	2^{21} 2^{19} 2^3 2^0 <div style="border: 1px solid black; padding: 5px; width: fit-content;"> Chip address select Internal bit address in chip 4-bit bank </div>
64K	8	32	6	2^{22} 2^{20} 2^4 2^0 <div style="border: 1px solid black; padding: 5px; width: fit-content;"> Chip address select Internal bit address in chip 5-bit bank </div>

MEMORY ACCESS

The CPU has five memory access ports: Port A, Port B, Port C, and two I/O ports. Each port is capable of making one reference per CP. Both I/O ports can be active simultaneously. Ports A, B, and C are used for CPU register transfers. B, T, and vector memory instructions issue to a particular memory port:

- Vector read (block reads only), and B read instructions (176, 034) use Port A
- Vector read (block reads only), and T read instructions (176, 036) use Port B
- Vector store, B, or T store instructions (177, 035, and 037) and scalar instructions (100 through 137) use Port C

Once an instruction issues to a port, that port is reserved until all references are made for that instruction.

The references for each element of a block transfer (V, B, or T) are made and completed in sequence through a port. Since each reference is examined individually for possible conflicts, the data flow for a transfer may not be continuous. If an instruction requires a port that is busy, issue is blocked. Total execution time of the transfer depends on the number and type of conflicts encountered during the transfer.

NOTE

Because concurrent block reads and writes are not examined for memory overlap hazard conditions (that is, read before write or write before read), the software must detect where this condition occurs and ensure sequential operation.

The bidirectional memory mode enable (002600), bidirectional memory mode disable (002500), and the complete memory reference (002700) instructions are provided to resolve these cases and assure sequential operation. If the bidirectional memory mode is clear, block reads and writes are not allowed to operate concurrently. Instruction 0027 allows the program to wait until the last references of all preceding block transfers are past the conflict resolution stage and the transferred data is being transmitted to the designated memory or register locations. Instruction 0027 provides software a mechanism, wherever necessary in the program, to guarantee sequential memory operation.

Issue of scalar memory references requires Ports A, B, and C to be available, ensuring sequential operation between block transfers and scalar references.

A scalar reference conflict is detected in CP 3 of execution. If a conflict occurs, one more scalar reference is allowed to issue. A third scalar reference holds issue if the conflict condition still exists for the preceding scalar reference.

Scalar references always execute in the order they are issued. Instruction 0027 detects when all scalar references are past the conflict resolution stage within the CPU.

One-half of the I/O channels reference memory through each of the I/O ports. The I/O ports can be active regardless of the activities on Ports A, B, or C.

When an instruction fetch request occurs, all referencing from the eight memory ports is inhibited (in this regard, the CRAY X-MP single-processor system is like the CRAY X-MP dual-processor system). When memory is quiet, the fetch proceeds and references 32 banks in the next 4 CPs (10 CPs if 16 banks). Then nonfetch referencing from the ports is enabled.

NOTE

A fetch sequence that follows a scalar store can, under certain conditions, complete before the store. For this to happen, however, an out-of-buffer condition must arise before the scalar store is in CP 2 of execution. The out-of-buffer condition can occur before the scalar store is in CP 2 of execution if a buffer boundary is crossed without doing a branch. This presents a problem only if the fetch and store are to the same area in memory. Therefore, software that uses dynamic coding should ensure that the code generated is actually in memory before that area of memory is fetched into the instruction buffers.

An exchange requires all activities within the CPU to complete before the exchange request is made.

When the exchange request is made, all referencing from the memory ports is inhibited. When memory is quiet, the exchange proceeds and references 16 banks in the next 25 CPs. Each bank is referenced twice during this time, once for a read and once for a write. A fetch request follows immediately after the exchange reference is complete and then referencing from the memory ports is enabled.

Conflict resolution

During each CP, references to the memory ports in the system are examined for memory access conflicts. If a conflict occurs for a reference, the reference is held and no further referencing from that port is allowed until the conflict is resolved.

Two types of memory access conflicts can occur: Bank Busy and Section Access.

Bank Busy conflict - The Bank Busy conflict is caused by any port requesting a bank currently in a reference cycle. Resolution of this conflict occurs when the bank cycle is complete. Hold reference because of a Bank Busy conflict, 1 to 7 CPs.

Section Access conflict - The Section Access conflict is caused by two or more ports in the CPU requesting any bank in the same section. Resolution of this conflict is based on priority and the Bank Busy conflict. The highest priority port with no Bank Busy conflict is allowed to proceed, all other ports involved in this conflict hold (refer to Memory access priorities subsection). Hold reference 1 CP because of a section access conflict.

Memory access priorities

The following statements are used to resolve memory access conflicts and determine the priority between Ports A, B, and C:

- Any port with an odd increment always has a higher priority than a port with an even increment regardless of their issued sequence.
- Among all ports with the same type of increment (odd or even), the relative time of issue determines the priority, with the first issued having the highest priority.
- I/O ports are always lowest priority.

16-BANK PHASING

The effect of 16-bank phasing on instruction fetches is a predictable increase of 6 CPs for filling instruction buffers. Otherwise, the amount of performance degradation for 16 banks instead of 32 banks is not readily predictable since it largely results from an increased number of memory conflicts.

For maintenance purposes, a 32-bank system can be modified to operate with only 16 banks and use either the lower or upper half of memory. Maintenance is accomplished by setting the bank select switch on the mainframe's control panel to the lower or upper banks.

MEMORY ERROR CORRECTION

A SECDED network is used between the CPU and memory. SECDED assures that data written into memory can be returned to the CPU with consistent precision (figure 2-2).

If a single bit of a data word is altered, the single error alteration is automatically corrected before passing the data word to the computer. If 2 bits of the same data word are altered, the 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.

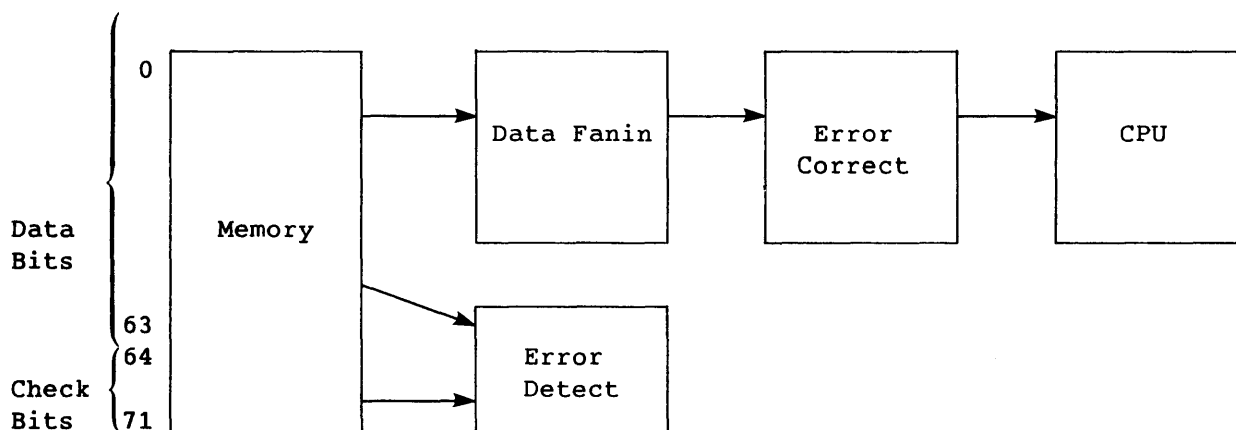


Figure 2-2. Memory Datapath with SECDED

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 before the data is written in memory. The 8 check bits are generated as even parity bits for a specific group of data bits. Figure 2-3 shows the bits of the data word used to determine the state of each check bit. An X in the horizontal row indicates that data bit contributes to the generation of that check bit. Thus, check bit 0 is the bit that makes 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} .

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

	CHECK BYTE																										
	271	270	269	268	267	266	265	264	263	262	261	260	259	258	257	256	255	254	253	252	251	250	249	248			
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					
check bit 3					x										x	x	x	x	x	x	x	x					
check bit 4				x											x				x								
check bit 5			x												x	x					x	x					
check bit 6		x													x	x	x	x									
check bit 7	x														x						x	x					
	247	246	245	244	243	242	241	240	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224			
	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	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				
	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	207	206	205	204	203	202	201	200			
	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 2-3. Error Correction Matrix

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 2-3 is used to generate a new set of check bits, which are 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.

† Syndrome: Any set of characteristics regarded as identifying a certain type, condition, and so on. (Webster's New World Dictionary).

The matrix is designed so that:

- If all S bits are 0, no error is assumed.
- If only 1 S bit is 1, the associated check bit is in error.
- If more than 1 S bit is 1 and the parity of S bits S0 through S7 is even, a double error (or an even number of bit errors) occurred within the data bits or check bits.
- If more than 1 S bit is 1 and the parity of all S bits is odd, 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 all S bits is odd and results are ambiguous.

Modules involved with generating and interpreting the 8-bit check byte used for SECDED include logic that can be used for verifying check bit storage, check bit generation, and error detection and correction. Refer to appendix D for information on SECDED maintenance functions.

INTER-CPU COMMUNICATION SECTION

The inter-CPU communication section of the mainframe contains special hardware for data storage, control, and for a Real-time Clock (RTC). The RTC, Shared Address (SB), Shared Scalar (ST), and Semaphore (SM) registers are available for use by the CPU. These registers, with their sources and destinations, are shown in figure 2-4 and described in the following paragraphs.

REAL-TIME CLOCK

The mainframe contains one RTC register. Programs can be timed precisely by using the CP counter. This counter is 64 bits and 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. In such an application, however, the counting can contain counts from other tasks if an interrupt occurs before the end time is read.

Instructions used with the RTC register are:

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
0014j0	RT Sj	Enter the RTC register with (Sj)
072i00	Si RT	Transmit (RTC) to Si

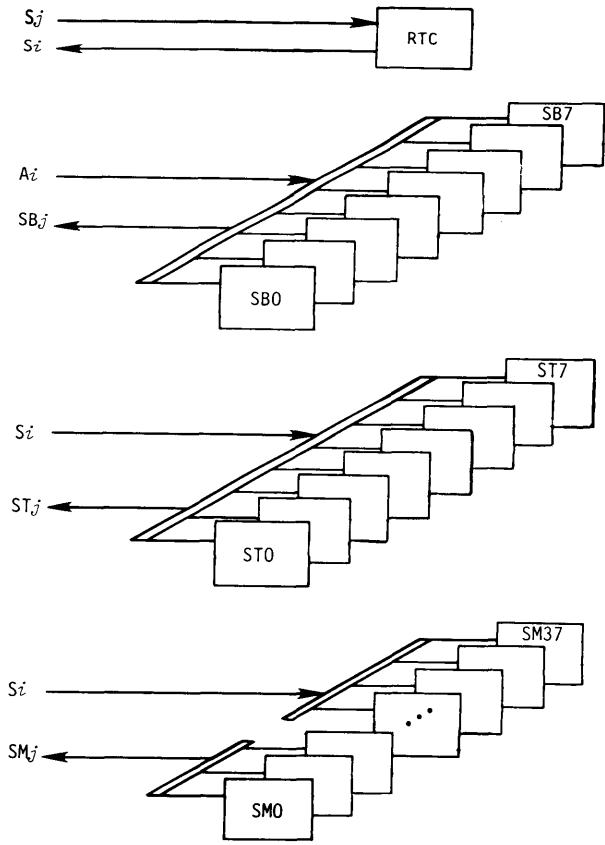


Figure 2-4. Shared Registers and Real-time Clock

In monitor mode, a program reads the CP counter by using instruction 072 and resets it with instruction 0014j0.

INTER-CPU COMMUNICATION AND CONTROL

Three sets of shared registers can be used by the CPU for storage and control. Each set contains eight 24-bit SB registers, eight 64-bit ST registers, and 32 1-bit SM registers.

The CPU's Cluster Number (CLN) register determines which set of shared registers is accessed by the CPU (clustering). The CLN register is loaded from the Exchange Package or if the CPU is in monitor mode, through instruction 0014j3.

The CLN register can contain one of four different values. Values 1, 2, or 3 allow the CPU to access one of the three sets of shared registers. Value 0 prevents any access to shared registers by the CPU. If the value is 0, instructions regarding the shared registers become no-ops, except for the instructions returning values to A_i or S_i , which return a zero value.

Shared Address and Shared Scalar registers

The Shared Address (SB) and Shared Scalar (ST) registers require no hardware reservations. Instructions used with the SB and ST registers are:

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
026ij7	A_i SBj	Transmit (SBj) to A_i
027ij7	SBj A_i	Transmit (A_i) to SBj
072ij3	S_i STj	Transmit (STj) to S_i
073ij3	STj S_i	Transmit (S_i) to STj

Semaphore registers

The SM registers can be used by the CPU for storage and control. The test and set instruction first tests the value of the selected SM register. If the value is 0, the instruction issues and sets that SM register to a 1. If the value is 1, the instruction holds issue until the value is 0.

If the CPU holds issue on a test and set instruction, it receives a deadlock interrupt. No deadlock interrupt can occur in cluster 0 (CLN=0).

When an interrupt occurs, normally the instructions already in the Next Instruction Parcel (NIP) and Current Instruction Parcel (CIP) registers are allowed to issue before the exchange sequence starts. If a test and set instruction is holding in the CIP register and an interrupt occurs, a special exchange startup sequence is initiated. Here, the instruction in the NIP register and the test and set instruction in the CIP register are discarded and the Program Counter (P) register is adjusted to point to the discarded test and set instruction. The Waiting on Semaphore (WS) flag in the Exchange Package sets, indicating a test and set instruction was holding in the CIP register when the interrupt occurred. The exchange sequence is then started.

Instructions used with the SM registers are:

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
0034jk	SMjk 1,TS	Test and set, SMjk
0036jk	SMjk 0	Clear SMjk

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
0037jk	SMjk 1	Set SMjk
072i02	Si SM	Transmit (SM) to Si
073i02	SM Si	Transmit (Si) to SM

CPU INPUT/OUTPUT SECTION (Maximum Configuration)

The mainframe supports channels connecting it to the IOS, the optional SSD, and front-end interfaces. The IOS channel operates at 100 Mbytes per second, the SSD channel operates at 1250 Mbyte per second, and the front-end interface channels operate at 6 Mbytes per second.

One 1250 Mbyte per second channel pair is used to transfer data between Central Memory and the SSD. These channels are 128 bits wide and use 16 check bits in each direction. A maximum transfer rate of over 10 gigabits/s is possible on the channel. The channel is two parallel 64-bit channels, each with SECDED; therefore, under certain circumstances the full-width channel can correct double errors.

Two 100 Mbyte per second channel pairs transfer data between Central Memory and an IOS. A 100 Mbyte per second channel is 64 bits wide and uses 8 check bits in each direction. Data words are transferred in blocks of 16 under control of Data Ready and Data Transmit control signals. Each 100 Mbyte per second channel has a maximum transfer rate of approximately 850 Mbits per second.

IOS communication with the CPUs is over four pairs of control channels, each with a maximum transfer rate of 6 Mbytes per second. Each 6 Mbyte per second channel is 16 bits wide.

There are two I/O ports. The channels are hardwired into a port with two 6 Mbyte per second channel pairs, one 100 Mbyte per second channel pair, and one-half of the SSD's 1250 Mbyte per second channel per port. Each port can transfer data at a rate of 1 word per CP. For the 100 Mbyte per second channels and one-half of the 1250 Mbyte per second channels, each time a buffer makes a reference, it holds the port until complete, usually 16 words.

All I/O uses the I/O ports to memory, and a scanner controls access to these ports. All CPU memory ports (Ports A, B, and C) have higher priority than the I/O ports.

Channel features of the I/O section are summarized below and described in the remainder of this section.

- One channel pair with a 1250 Mbytes per second maximum transfer rate per channel; 128 data bits and 16 check bits in each direction.

- Two channel pairs with a 100 Mbytes per second maximum transfer rate per channel; 64 data bits, 3 control bits, and 8 check bits in each direction.
- Four I/O channel pairs with a 6 Mbytes per second maximum transfer rate per channel
 - 16 data bits, 3 control bits, and 4 parity bits in each direction
 - Lost data detection
- Channels are divided into groups, each group contains either input or output channels
- Channel groups are served equally by memory (each group is scanned every 4 CPs)
- Channel priority resolved within channel groups

DATA TRANSFER FOR SOLID-STATE STORAGE DEVICE

Data is transferred directly between the SSD and the mainframe using 1250 Mbyte per second channels. This 1250 Mbyte per second channel is 128 bits wide and is programmed through software. The Solid-state Storage Device (SSD®) Reference Manual describes programming details for the SSD.

DATA TRANSFER FOR I/O SUBSYSTEM

A 100 Mbyte per second channel pair transfers data between Central Memory of the mainframe and the IOP. Each channel is 64 bits wide and handles data at approximately 100 Mbytes per second. Each channel uses an additional 8 check bits for SECDED, as is used in Central Memory.

The CPU side of a 100 Mbyte per second channel pair 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 IOP, 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 a 100 Mbyte per second channel pair, data passing into Local Memory (an IOP's memory) is double-buffered and disassembled into 16-bit parcels. The channel side passing data from Local Memory simply assembles 16-bit parcels into 64-bit words for transmission to a CPU.

An IOP controls a 100 Mbyte per second channel pair 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 Mbyte per second channel pair. The I/O Subsystem Hardware Reference Manual for your IOS contains programming details for the 100 Mbyte per second channel pair.

6 MBYTE PER SECOND CHANNELS

Standard control channels for the system are 6 Mbyte per second channels. Each 6 Mbyte per second channel has 16-bit asynchronous control logic used for front-end interfaces. The instructions used with 6 Mbyte per second channels follow.

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
0010jk	CA,Aj Ak	Set the Current Address (CA) register for the channel indicated by (Aj) to (Ak) and activate the channel
0011jk	CL,Aj Ak	Set the Limit Address (CL) register for the channel indicated by (Aj) to (Ak)
0012jk	CI,Aj	Clear the Interrupt flag and Error flag for the channel indicated by (Aj): Output channel k=0; clear MC, k=1; set MC. Input channel k=0; no operation, k=1; clear held ready.
033i00	Ai CI	Transmit channel number to Ai
033ij0	Ai CA,Aj	Transmit address of channel (Aj) to Ai
033ij1	Ai CE,Aj	Transmit Error flag of channel (Aj) to Ai

6 MBYTE PER SECOND CHANNEL OPERATION

Each input or each 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 2-2. In both input and output operations, parcel 0 is always transferred first.

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 a pair has a Master Clear line. Appendix B describes the signal sequence of a 6 Mbyte per second channel.

The following conditions must be met for an I/O interrupt to occur.

- CPU is not waiting for an exchange.
- CPU is not in monitor mode.
- An interrupt is present.

Table 2-2. Channel Word Assembly/Disassembly

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

I/O interrupts can be caused by the following:

- On all output channels, if (CA) becomes equal to (CL), then the resume for the last parcel transmitted sets interrupt.
- 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 Ai the highest priority channel number requesting an interrupt. The lowest numbered channel has the highest priority. The interrupt request continues until cleared by the monitor program when an interrupt from the next highest priority channel, if present, is sensed. All interrupts are available through instruction 033. Channel numbers for 6 Mbyte per second channels are 10_8 through 17_8 (10/11, 12/13, 14/15, and 16/17; even for input, odd for output).

INPUT CHANNEL PROGRAMMING

To start an input operation, the CPU program (refer to figure 2-5):

1. Sets the channel CL to the last word address (LWA) + 1 (LWA+1)
2. Sets the channel CA to the first word address (FWA)

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.

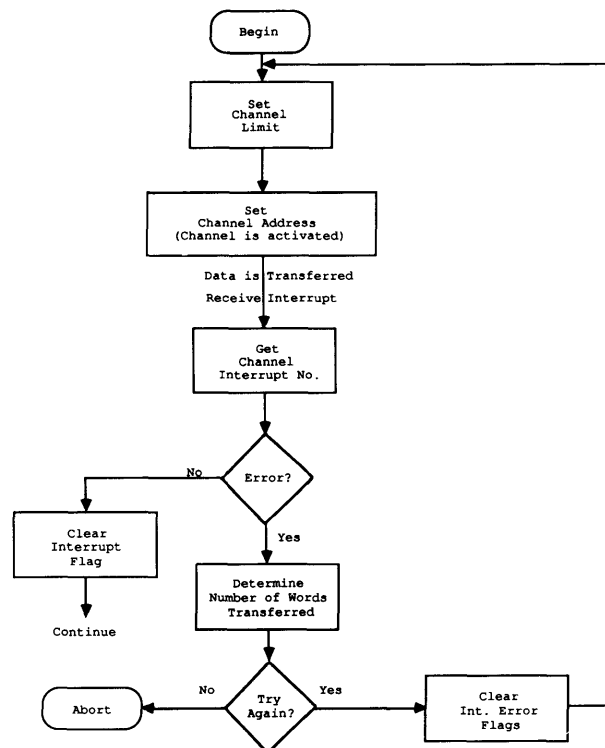


Figure 2-5. Basic I/O Program Flowchart

An external transmitting device sends a Disconnect signal to indicate the end of a 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 the channel Error flag is set.

INPUT CHANNEL ERROR CONDITIONS

Input channel error conditions can occur at a parcel level (parity error) or channel level (unexpected Ready signal). When a parcel in error occurs, the Parity Fault flag sets immediately. The Parity Fault flag does not generate an interrupt; it is saved and sets the Error flag when a disconnect occurs or if CA = CL. Therefore, the program should check the state of the Error flag when an interrupt is honored. All parcels stored after the error are zeroed.

If a Ready signal is received when the channel is not active (unexpected Ready signal), the Ready condition is held until the channel is activated. At this time a Resume signal is sent. No Error flag is set and no interrupt request is generated. Since the Ready condition is held when the channel is inactive, it is sometimes advantageous to be able to clear this Ready signal before setting up the channel, especially on a deadstart or a resynchronization of the channel after an error. The Ready signal can be cleared by using instruction 0012j1 to input channel (Ai), clearing any Ready signal being held before issue of instruction 0012j1.

OUTPUT CHANNEL PROGRAMMING

To start an output operation, the CPU program:

1. Sets the channel CL to the last word address + 1 (LWA+1)
2. Sets the channel CA 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 CA register contents. 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.

The Interrupt flag also sets if an error is detected. The output channel detects two errors; a Resume signal received when the channel is inactive and a Resume signal received while a Read Reference Request is present. No external response is generated.

PROGRAMMED MASTER CLEAR TO EXTERNAL DEVICE

The system can send a Master Clear signal to an external device through the output channel. The external Master Clear sequence is as follows:

<u>Step</u>	<u>Octal Code</u>	<u>Description</u>
1.	0012jk	Clears input channel to ensure external activity on the channel pair has stopped
2.	0012jl	Clears output channel to ensure CPU activity on the channel pair has stopped; sets Master Clear.
3.	Delay 1	Device dependent; determines the duration of the Master Clear signal.
4.	0012j0	Clears the output channel; this turns off the Master Clear signal.
5.	Delay 2	Device dependent; allows time for initialization activities in the attached device to complete.

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

MEMORY ACCESS

Each of the channel groups shown below is assigned a time slot (figure 2-6) that is scanned once every 4 CPs for a memory request. The lowest numbered channel in the group has the highest priority. During the next 3 CPs, the scanner allows requests from the other channel groups. Therefore, it is possible to have an I/O memory request every CP on an I/O port. The scanner stops for all memory conflicts caused by an I/O reference and also stops for a block reference while a buffer is referencing, maximum 16 words (figure 2-7).

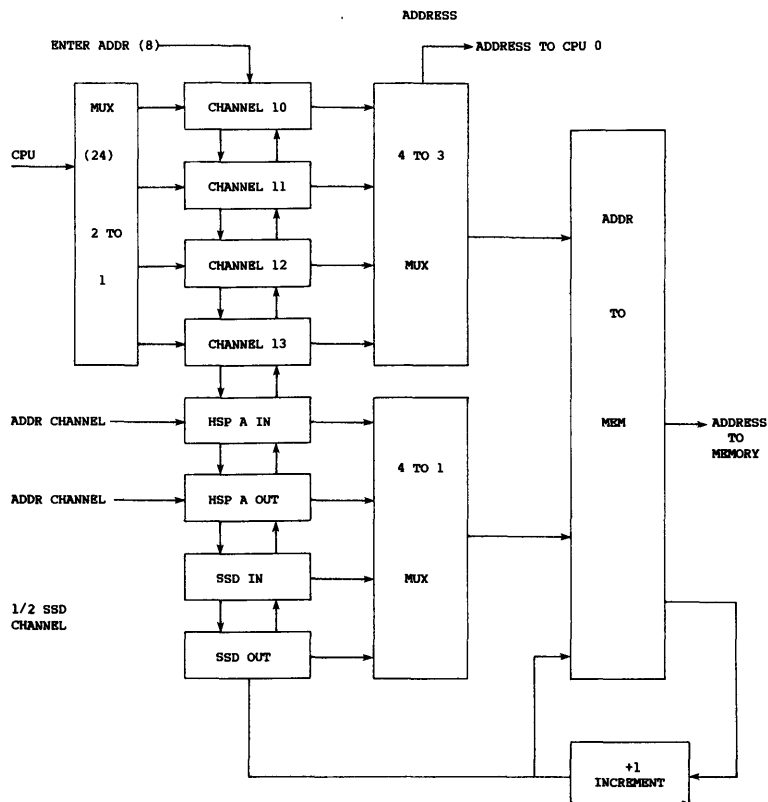
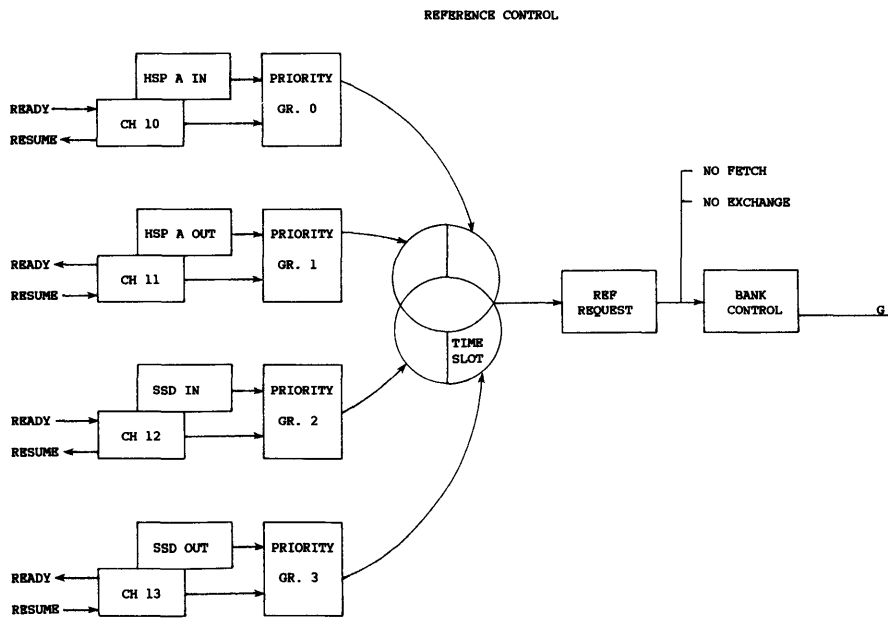
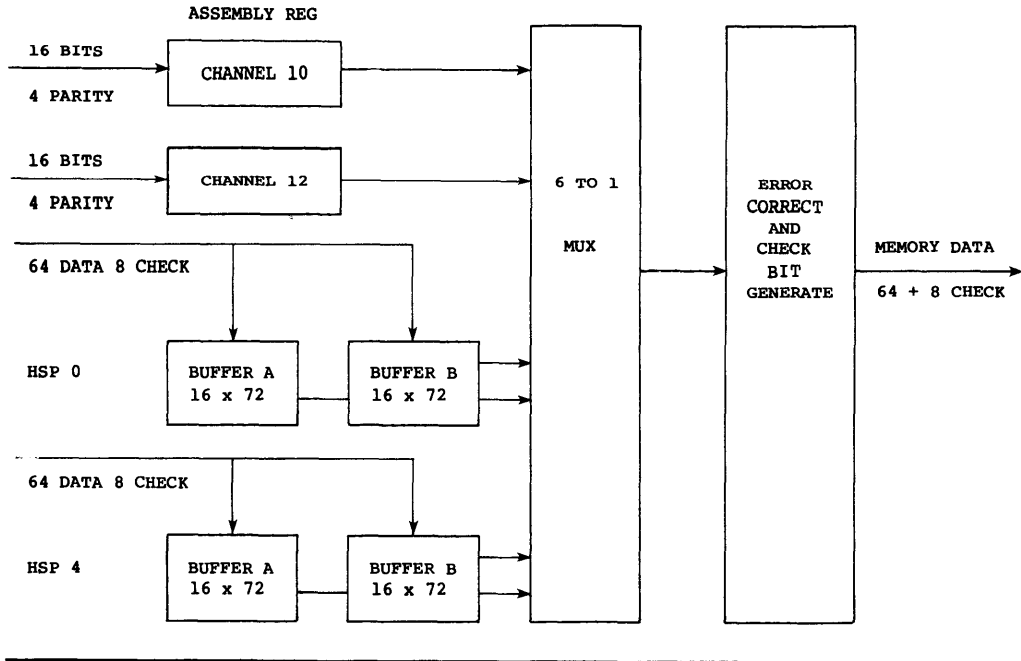


Figure 2-6. Channel I/O Control

INPUT DATA PATH



OUTPUT DATA PATH

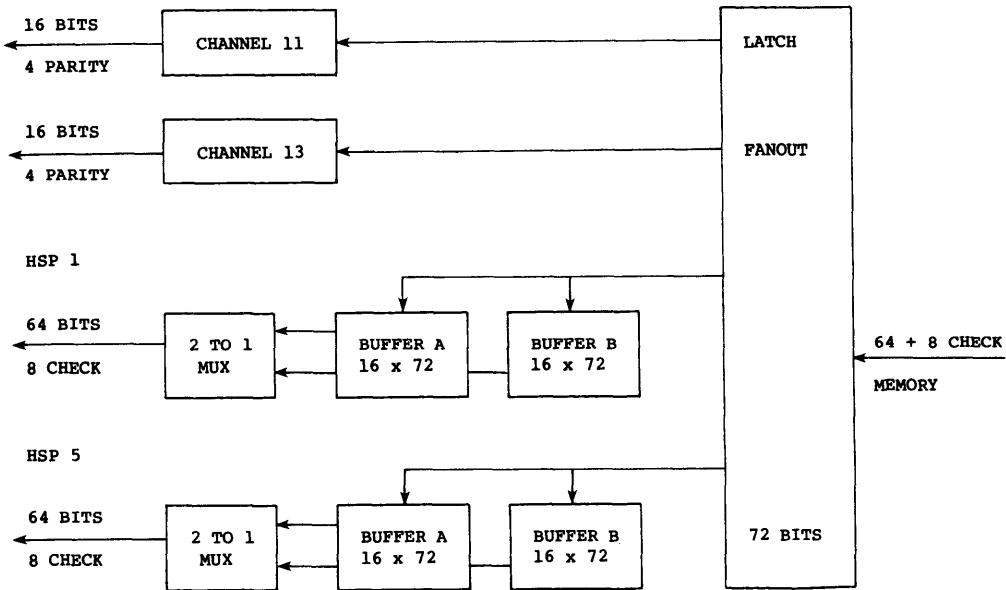


Figure 2-7. Input/Output Datapaths

The 6 Mbyte per second channels are numbered 10_8 through 17_8 , and the 100 Mbyte per second channels are A and B. The SSD channel number is 7. The channels are grouped as follows:

<u>Group</u>	<u>Upper I/O Port</u>	<u>Lower I/O Port</u>
0 input channels	A,10	B,14
1 output channels	A,11	B,15
2 input channels	7,12	7,16
3 output channels	7,13	7,17

I/O LOCKOUT

An I/O memory request can be locked out by an exchange sequence or instruction fetch sequence.

MEMORY BANK CONFLICTS

Memory bank conflicts are tested for CPU scalar, vector, and I/O memory references. When an exchange sequence or instruction fetch sequence is in progress, all other memory references are locked out.

Each memory bank can accept a new request every 8 CPs. To test for a memory bank conflict, the 5 low-order bits[†] of the memory address are checked against Bank Busy conflicts and other memory references. The bank is busy for 8 CPs on a reference.

I/O MEMORY CONFLICTS

Before testing for a memory bank conflict, a check is made to ensure no exchange sequence or instruction fetch sequence is in progress. If either of these conditions exists, the I/O request is held. The 5 low-order address bits[†] of an I/O reference are tested against Bank Busy conflicts and other memory references. If a bank being referenced is busy, the reference is held and the scanner is stopped.

[†] The 4 bits for 16-bank phasing; refer to subsection on Central Memory.

I/O MEMORY REQUEST CONDITIONS

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

- I/O request
- Bank not busy
- No simultaneous conflicts
- No fetch request
- No exchange sequence

NOTE

As mentioned previously, the CPU has four access paths to memory (one to each section) available for use by Ports A, B, and C, and the I/O ports. There are also four additional access paths available to a fetch and the I/O ports. The I/O ports are partitioned according to which group of four access paths they use. The I/O port using the paths available to Ports A, B, and C is allowed to make a reference if that reference is not to the same section as a Port A, B, or C reference. The other I/O port is allowed to make a reference provided that reference is not to the same bank as a Port A, B, or C reference.

I/O MEMORY ADDRESSING

All I/O memory references are absolute. The CA and CL registers are 24 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.

The CPU's control section contains registers and instruction buffers for instruction issue and control, and uses an exchange mechanism for switching instruction execution from program to program. This section describes these registers and buffers and the exchange mechanism. Memory field protection, programmable clock, and deadstart sequence are also described.

INSTRUCTION ISSUE AND CONTROL

The following paragraphs describe the registers and instruction buffers involved with instruction issue and control. Figure 3-1 illustrates the general flow of instruction parcels through the registers and buffers.

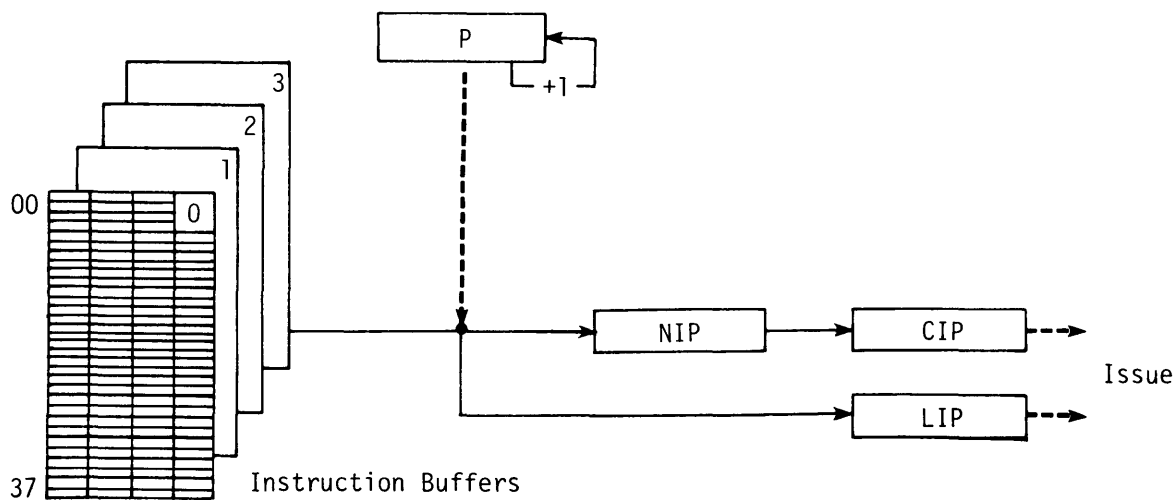


Figure 3-1. Instruction Issue and Control Elements

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 relative to the base address. The low-order 2 bits indicate the parcel within the word. Except on a branch instruction when the branch is taken or on an exchange, the P register contents are advanced 1 when an instruction parcel 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. The value stored in P might not be accurate during the deadstart sequence.

NEXT INSTRUCTION PARCEL REGISTER

The 16-bit NIP register holds a parcel of program code before it enters the Current Instruction Parcel (CIP) register.

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.

CURRENT INSTRUCTION PARCEL REGISTER

The 16-bit CIP register holds the instruction waiting to issue. The term issue indicates the transition of an instruction in CIP to its execution phase. If an instruction is a 2-parcel instruction, the CIP register holds the first parcel of the instruction and the Lower Instruction Parcel (LIP) register holds the second parcel. Issue of an instruction in CIP can be delayed until conflicting operations have been completed. 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.

The 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 LIP register holds the second parcel of a 2-parcel instruction at the time the first parcel of the 2-parcel instruction is in the CIP register.

INSTRUCTION BUFFERS

The CPU has four instruction buffers; each of which can hold 128 consecutive 16-bit instruction parcels (figure 3-2). Instruction parcels are held in the buffers before being delivered to the NIP or LIP registers.

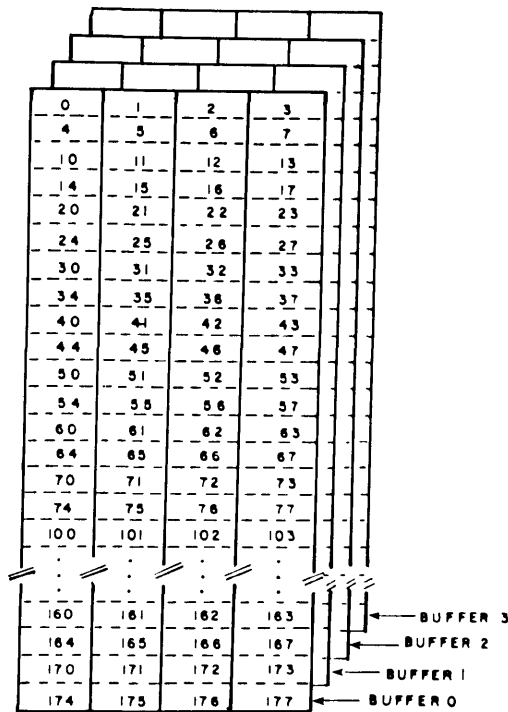


Figure 3-2. Instruction Buffers

The beginning instruction parcel in a buffer always has a word address that is a multiple of 40_g (a parcel address that is a multiple of 200_g) allowing the entire range of addresses for instructions in a buffer to be defined by the high-order 17 bits of the parcel address. Each buffer has a 17-bit Beginning Address (IBAR) register containing this value.

The Beginning Address registers are scanned each CP. If the high-order 17 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. 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 becomes available when the first parcel issues from the CIP register. Simultaneously, 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 the instruction reaching the NIP register.

An out-of-buffer condition exists when the high-order 17 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 8 words per CP, fully occupying memory. The first group of 32 parcels delivered to the buffer always contains the next instruction required for execution. For this reason, the branch out-of-buffer time is 19 CPs for 32-bank memories and 25 CPs for 16-bank memories, providing memory is not busy (if busy, the branch fetch is delayed until the busy is resolved). Once the fetch proceeds, the remaining groups arrive at a rate of 32 parcels per CP and circularly fill the buffer.

An instruction buffer is loaded with 1 word of instructions from each of the 32 memory banks or 2 words from each of the 16 banks. The first four instruction parcels residing in an instruction buffer are always from bank 0. An exchange sequence voids the instruction buffers, preventing a match with the P register and causing the buffers to be loaded as needed.

Forward and backward branching is 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 parcels cannot occur in the instruction buffers. Because instructions are held in instruction buffers before issue and after (until the buffer is reloaded), self-modifying code should not be used. Also, because of independent data and instruction memory protection, self-modifying code may be impossible. 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 512 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 or exchange causes reloading of a buffer.

EXCHANGE MECHANISM

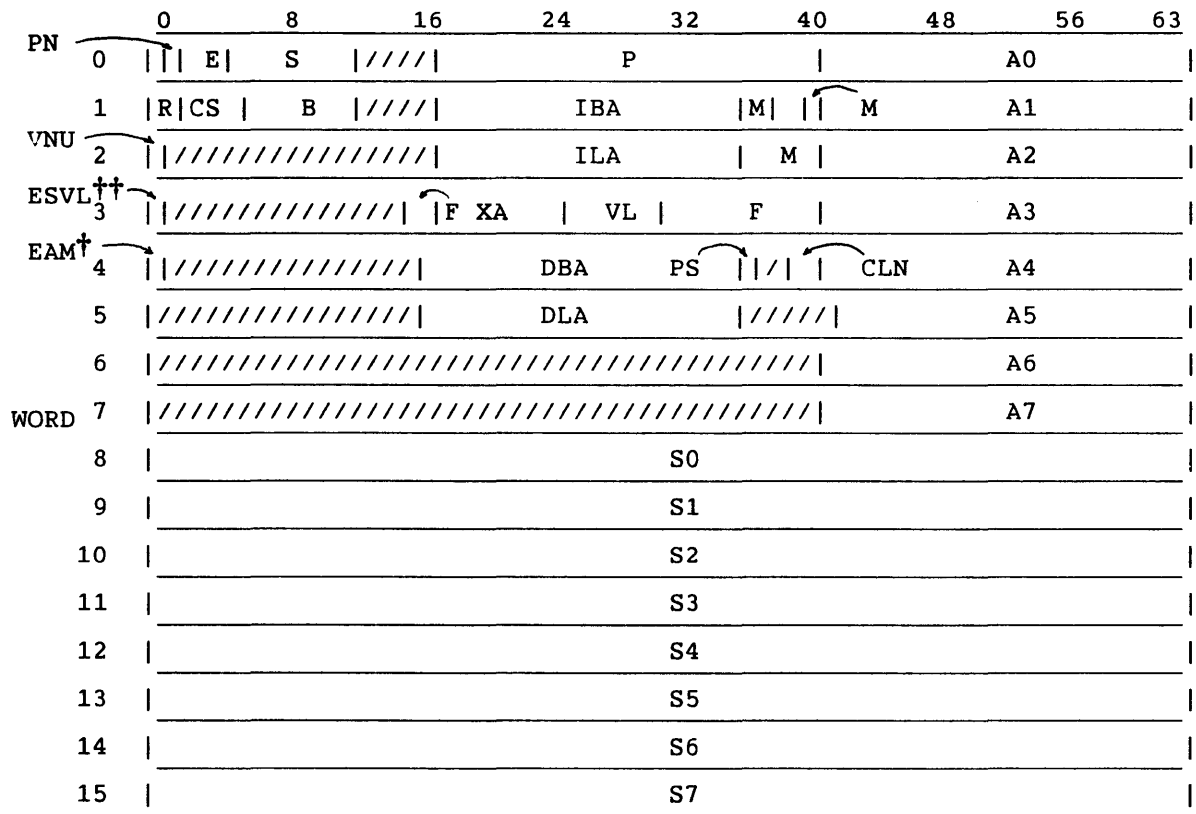
The CPU uses an exchange mechanism for switching instruction execution from program to program. This exchange mechanism involves the use of blocks of program parameters known as Exchange Packages and a CPU operation referred to as an exchange sequence. For the convenience of Cray Assembly Language (CAL) 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 3-3) 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 execution interval for the program to the next.

The Exchange Package contents are arranged in a 16-word block; the contents are explained in the paragraphs following table 3-1. The exchange sequence swaps data from memory to the operating registers and back to memory. This sequence exchanges data in an active Exchange Package residing in the operating registers with an inactive Exchange Package in memory. The Exchange Address (XA) register address of the active Exchange Package specifies the memory address to be used for the swap. Data is exchanged and a new program execution interval is initiated by the exchange sequence.

The contents of the B, T, V, VM, SB, ST, and SM 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. (Refer to section 4 for descriptions of the operating registers and the VL register.)



† Present only on 8-million-word systems
 †† Not available on all systems

Figure 3-3. Exchange Package for a Four-processor System

Field	Word	Bits	Description
PN	0	1	Processor number
E	0	2-3	Error type
S	0	4-11	Syndrome bits
P	0	16-39	Program Address register
R	1	0-1	Read mode
CSB	1	2-4 (CS);	Read address
	1	7-11 (B)	
IBA	1	16-34	Instruction Base Address
ILA	2	16-34	Instruction Limit Address
M	1	35-37,39	Mode register
	2	35-39	

<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
VNU	2	0	Vector not used
ESVL	3	0	Enable Second Vector Logical
F	3	15; 31-39	Flag register
XA	3	16-23	Exchange Address register
VL	3	24-30	Vector Length register
EAM	4	0	Enhanced Addressing Mode
DBA	4	16-34	Data Base Address
PS	4	35	Program State
CLN	4	38-39	Cluster Number
DLA	5	16-34	Data Limit Address
	0-7	40-63	Eight A register contents
	8-15	0-63	Eight S register contents

Processor number

The state of the PN position in the Exchange Package is always 0. This value is not read into the CPU; it is a constant inserted only into the package being stored.

Memory error data

Bit 36 (interrupt on correctable memory error bit) and bit 38 (interrupt on uncorrectable memory error bit) in the 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 correctable memory error is encountered or if bit 38 is set and an uncorrectable memory error is detected.†

Memory error data fields are described as follows.

† For multiple bit memory errors, the hardware always sets the Correctable Memory Error flag in the interrupted Exchange Package.

<u>Field</u>	<u>Description</u>
Error type (E)	The type of memory error encountered, uncorrectable or correctable, is indicated in word 0, bits 2 and 3 of the Exchange Package. Bit 2 is set for an uncorrectable memory error; bit 3 is set for a correctable memory error.
Syndrome (S)	The 8 S bits used in detecting a memory data error are returned in word 0, bits 4 through 11, of the Exchange Package. Refer to section 2 for additional information.
Read mode (R)	Indicates the read mode in progress when a memory data error occurred and is in word 1, bits 0 and 1 of the Exchange Package. These bits assume the following values: 00 I/O 01 Scalar (memory references with A or S) 10 Vector, B, or T 11 Instruction fetch or exchange
Read address (CSB)	The 3-bit CS field and 5-bit B field contain the address where a memory data error occurred. Word 1, bits 7 through 11 (B), of the Exchange Package contain the 5 low-order bits of the address and can be considered as the bank address. Word 1, bits 2 through 4, of the Exchange Package contain the chip select bits of the address. For the 16K-chip mainframe, the high-order 3 bits of this field can be considered as the chip select; for the 64K-chip mainframe, only the low-order 2 bits can be considered as the chip select.

Program Address register

The contents of the Program Address (P) register (address of first program instruction not yet issued) are stored in bits 16 through 39 of word 0. The instruction at this location is the first instruction to be issued when this program begins again.

Memory field registers

Each object program has a designated field of memory for instructions and data that is specified by the monitor program when the object program is loaded and initiated. All memory addresses contained in the object program code are relative to one of two base addresses specifying the beginning of the appropriate field. Each object program reference to

memory is checked against the limit and base addresses to determine if the address is within the bounds assigned. These field limits are contained in four registers that are saved in the Exchange Package. The four registers are: the Instruction Base Address (IBA) register, the Instruction Limit Address (ILA) register, the Data Base Address (DBA) register, and the Data Limit Address (DLA) register. Refer to the subsection on Memory Field Protection later in this section for an explanation of the registers.

Mode register

The 9-bit M register contains part of the Exchange Package for a currently active program. The M register bits are assigned in words 1 and 2 of the Exchange Package as follows:

Word 1

<u>Bit</u>	<u>Description</u>
------------	--------------------

- | | |
|----|---|
| 35 | Waiting for Semaphore (WS) flag; when set, the CPU exchanged when a test and set instruction was holding in the CIP register. |
| 36 | Floating-point Error Status (FPS) flag; when set, a floating-point error has occurred regardless of the Floating-point Error Mode flag/state. |
| 37 | Bidirectional Memory Mode (BDM) flag; when set, block reads and writes can operate concurrently. |
| 39 | Interrupt Monitor Mode (IMM) flag; when set, enables all interrupts in monitor mode except PC, MCU, I/O, and ICP. |

Word 2

<u>Bit</u>	<u>Description</u>
------------	--------------------

- | | |
|----|---|
| 35 | Operand Range Error Mode (IOR) flag; when set, enables interrupts on operand address range errors. |
| 36 | Correctable Memory Error Mode (ICM) flag; when set, enables interrupts on correctable memory data errors. |
| 37 | Floating-point Error Mode (IFP) flag; when set, enables interrupts on floating-point errors. |
| 38 | Uncorrectable Memory Error Mode (IUM) flag; when set, enables interrupts on uncorrectable memory data errors. |
| 39 | Monitor Mode (MM) flag; when set, inhibits all interrupts except memory errors, error exit, and normal exit. |

The 9 bits are set selectively during an exchange sequence.

Word 1, bit 37, (Bidirectional Memory Mode flag) can be set or cleared by using instructions 0026 (enable bidirectional memory transfers) and 0025 (disable bidirectional memory transfers).

Word 2, bit 35, (Operand Range Error Mode flag) can be set or cleared during the execution interval of a program by using instructions 002300 (enable interrupt on operand address range error) and 002400 (disable interrupt on operand address range error).

Word 2, bit 37, (Floating-point Error Mode flag) can be set or cleared during the execution interval for a program by using instructions 002100 (enable interrupt on floating-point error) and 002200 (disable interrupt on floating-point error).

Word 1, bits 36 and 37, and word 2, bits 35 and 37, can be read with instruction 073i01. Word 1, bits 35 and 36, indicate the state of the CPU at the time of the exchange. The remaining bits are not altered during the execution interval for the Exchange Package and can be altered only when the Exchange Package is inactive in storage.

Vector not used (VNU)

The state of the VNU position in the Exchange Package indicates whether or not instructions 076, 077, or 140 through 177 were issued during the execution interval. If none of the instructions were issued, the bit remains set. If one or more of the instructions issued, the bit is cleared. Once cleared, the bit remains clear until reset through a memory store to the dormant Exchange Package.

Enable Second Vector Logical (ESVL)[†]

The state of the ESVL position in the Exchange Package indicates if the Second Vector Logical unit can be used. If set, instructions 140 through 145 may select the Second Vector Logical unit. If clear, the Second Vector Logical unit cannot be used; only the Full Vector Logical unit may be used.

[†] Not available on all systems

Flag register

The 10-bit F register contains part of the Exchange Package for the currently active program. This register is located in word 3 and contains 10 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 F register contents are stored along with the rest of the Exchange Package. The monitor program can analyze the 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 in word 3 of the Exchange Package as follows:

Word 3

<u>Bit</u>	<u>Description</u>
------------	--------------------

- | | |
|----|--|
| 15 | Deadlock (DL) flag; set when the CPU (CLN≠0) is holding issue on a test and set instruction. |
| 31 | Programmable Clock Interrupt (PCI) flag; set when the interrupt countdown counter in the programmable clock equals 0. The programmable clock is explained later in this section. |
| 32 | MCU Interrupt (MCU) flag; set when the MIOP sends this signal. |
| 33 | Floating-point Error (FPE) flag; set when a floating-point range error occurs in any of the floating-point functional units and the Enable Floating-point Interrupt flag is set. Section 4, Computation, explains floating-point functional units. |
| 34 | Operand Range Error (ORE) flag; set when a data reference is made outside the boundaries of the DBA and DLA registers and the Enable Operand Range Interrupt flag is set. Operand range error is explained later in this section. |
| 35 | Program Range Error (PRE) flag; set when an instruction fetch is made outside the boundaries of the Instruction Base Address (IBA) and Instruction Limit Address (ILA) registers. Program range error is explained later in this section. |
| 36 | Memory Error (ME) flag; set when a correctable or uncorrectable memory error occurs and the corresponding enable memory error mode bit is set in the M register. |

Word 3 (continued)

Bit Description

- 37 I/O Interrupt (IOI) flag; set when a 6 Mbyte channel or the 100 Mbyte to SSD channel completes a transfer.
- 38 Error Exit (EEX) flag; if not in MM, set by an error exit instruction (000).
- 39 Normal Exit (NEX) flag; if not in MM and IMM, set by a normal exit instruction (004).

Any flag (except the ME flag) can be set in the F register only if the active Exchange Package is not in monitor mode. Such flags are set only if word 2, bit 39 of the M register is 0. Except for the ME flag, if the program is in monitor mode and the conditions for setting an F register are present, the flag remains cleared and no exchange sequence is initiated.

Exchange Address register

The 8-bit XA register specifies the first word address (FWA) 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 that the absolute address be in the lower 4096 (10,000₈) 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.

Enhanced Addressing Mode (EAM)[†]

The state of the EAM position in the Exchange Package indicates whether or not address extension occurs for address calculations. If set, instructions 100 through 137 will sign-extend the 22-bit value (*jkm*) to 24 bits for address calculations (compatible with an 8-million-word system). If clear, all instructions 100 through 137 (not I/O) have address bits 2²² and 2²³ replaced by database address bits 2²² and 2²³.

[†] Available only on 8-million-word systems

Data Base Address register

Refer to the Memory field register subsection for register explanation.

Program State register

The state of the 1-bit Program State (PS) register is manipulated by the operating system to represent different program states in the CPU.

Cluster Number register

The Cluster Number (CLN) register determines the CPU's cluster. The CLN register contents are used to determine which set of SB, ST, and SM registers the CPU can access. If the CLN register is 0, then the CPU does not have access to any SB, ST, or SM register. The CLN register contents in the CPU is also used to determine the condition necessary for a deadlock interrupt.

Data Limit Address register

Refer to the Memory field registers subsection for explanation.

A registers

The current contents of all A registers are stored in bits 40 through 63 of words 0 through 7 during exchange.

S registers

The current contents of all S registers are stored in bits 0 through 63 of words 8 through 15 during exchange.

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 the 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 moves back to memory in a subsequent exchange sequence.

EXCHANGE SEQUENCE

The exchange sequence is the vehicle for moving an inactive Exchange Package from memory into the operating registers. Simultaneously, the exchange sequence moves the currently active Exchange Package from the operating registers back into memory. This swapping operation is done in a fixed sequence when all computational activity associated with the currently active Exchange Package has stopped. The same 16-word block of memory is used as the source of the inactive Exchange Package and the destination of the currently active Exchange Package. Location of this block is specified by the XA register contents and is a 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 contents to 0 and also forces an interrupt in the CPU. These two actions cause an exchange 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 swapped to address 0 is largely indeterminate because of the deadstart operation. New data entered at these storage addresses then discards the old 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 of one or more flags causes a Request Interrupt signal to initiate 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:

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
000	ERR	Error exit
004	EX	Normal exit

The two exits enable a program to request its own termination. A nonmonitor (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 times, and special cases for an exchange sequence.

Hold conditions:

- NIP register contains a valid instruction
- S, V, or A registers busy

Execution times:

- For 32 banks, 51 CPs; consists of an exchange sequence (32 CPs) and a fetch operation (19 CPs).
- For 16 banks, 57 CPs; consists of an exchange sequence (32 CPs) and a fetch operation (25 CPs).

Special cases:

If a test and set instruction is holding in the CIP register, both CIP and NIP registers are cleared and the exchange occurs with the Waiting for Semaphore (WS) flag set and the P register pointing to the test and set instruction.

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 (10,000₈) words of memory. The package at address 0 is the deadstart monitor program's Exchange Package. Other packages provide for object programs and monitor tasks. Nonmonitor packages lie outside of the field lengths for the programs they represent as determined by the base and limit addresses for the programs. Only the monitor program has a field defined so that it can access all of memory, including 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. Since no interlock exists between an exchange sequence in a CPU and memory transfers in another CPU, modification of Exchange Packages which can be used by another CPU should be avoided, except under software controlled situations.

Proper management of Exchange Packages dictates that a nonmonitor program always exchanges back to the monitor program that exchanged to it. The exchange ensures that the program information is always exchanged into its proper Exchange Package.

For example, the monitor program (A) begins an execution interval following deadstart. No interrupts (except memory) can terminate its execution interval since it is in monitor mode. Program A voluntarily exits by issuing a normal exit instruction (004). Before doing so, however, program A sets the XA register contents 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. Simultaneously, 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.

To illustrate the exchange sequence, assume that 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 area during the execution interval exchange back into the operating registers.

Program A, upon resuming execution, determines an interrupt has 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 operating task, program C can execute in monitor mode or in user mode.

MEMORY FIELD PROTECTION

At execution time each object program has a designated field of memory for instructions and data. The field limits are specified by the monitor program when the object program is loaded and initiated. The fields can begin at any word address that is a multiple of 32 (that is, 40₈) and can continue to another address that is one less than a multiple of 32. The fields can overlap.

All memory addresses contained in the object program code are relative to one of the two base addresses specifying the beginning of the appropriate field. An object program cannot read or alter any memory location with an absolute address lower than that 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.

Field limits are contained in four registers: the Instruction Base Address (IBA) register, the Instruction Limit Address (ILA) register, the Data Base Address (DBA) register, and the Data Limit Address (DLA) register. The following paragraphs describe the four registers and flags associated with the field limits.

INSTRUCTION BASE ADDRESS REGISTER

The IBA register holds the base address of the user's instruction field. An instruction can only be executed by the CPU if the absolute address at which the instruction is located is greater than or equal to the contents of the current Exchange Package IBA register of the program executing. This determination is made at instruction buffer fetch time by the CPU.

The contents of the IBA register are interpreted as the high-order 19 bits of a 24-bit memory address. The low-order 5 bits of the address are assumed to be 0 because of the number of banks, 32 (decimal) banks. Absolute memory addresses for an instruction fetch are formed by adding the IBA register to the P register (high-order 22 bits) modulo two to the twenty-second power.

A reference to an absolute address less than the address defined by IBA can only occur through a jump or branch instruction to an address beyond the memory capacity of the machine.

INSTRUCTION LIMIT ADDRESS REGISTER

The ILA register holds the limit address of the user's field. An instruction can only be executed by the CPU if the absolute address where it is located is less than the contents of the current Exchange Package ILA register of the program executing. This determination is made at instruction buffer fetch time by the CPU.

The ILA register contents are interpreted as the high-order 19 bits of a 24-bit memory address. The low-order 5 bits of the address are assumed to be 0 because of the number of banks, 32 (decimal) banks. The largest absolute address that can be executed by a program is defined by $[(ILA) \times 2^5] - 1$.

If the final absolute address of the instruction buffer fetch as computed by the CPU does not fall between the range of addresses contained within the currently executing Exchange Package IBA and ILA registers, the CPU generates a program range error interrupt.

DATA BASE ADDRESS REGISTER

The DBA register holds the base address of the user's data field. An operand can only be fetched or stored by the CPU if the absolute address where the operand is located is greater than or equal to the current Exchange Package DBA register contents of the program executing. This determination is made each time an operand is fetched or stored by the CPU.

The DBA register contents are interpreted as the high-order 19 bits of a 24-bit memory address. The low-order 5 bits of the DBA register are assumed to be 0. Absolute memory addresses for operands are formed by adding the DBA register to the modified operand address modulo two to the twenty-second power.

DATA LIMIT ADDRESS REGISTER

The DLA register holds the (upper) limit address of the user's data field. An operand can only be fetched or stored by the CPU if the absolute address where the operand is located is less than the current Exchange Package DLA register contents of the program executing. This determination is made each time an operand is fetched or stored by the CPU.

The DLA register contents are interpreted as the high-order 19 bits of a 24-bit memory address. The low-order 5 bits of the DLA register are assumed to be 0. The largest absolute address that can be referenced for data by a program is defined by $[(DLA) \times 2^5] - 1$.

If the final absolute address of the operand as computed by the CPU does not fall between the range of addresses contained within the currently executing Exchange Package DBA and DLA registers, the CPU generates an operand (address) range error interrupt.

PROGRAM RANGE ERROR

The Program Range Error flag sets if a memory reference outside the boundaries of the IBA and ILA registers is for an instruction fetch. An out-of-range memory reference can occur in a nonmonitor 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 the Operand Range Error Mode flag is set and a memory reference outside the boundaries of the DBA and DLA registers is called to read or write an operand for an A, B, S, T, or V register and the Operand Range Interrupt Error flag is set. 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.

PROGRAMMABLE CLOCK

The programmable clock can be used to accurately measure the duration of intervals. Intervals selected under monitor program control/generate a periodic interrupt. Clock frequency/intervals are as follows:

<u>CPU Speed</u>	<u>Frequency</u>	<u>Interval</u>
8.5-ns CP	117 Mhz	8.5-ns through 36.5 s
9.5-ns CP	105 Mhz	9.5-ns through 40.8 s

Intervals shorter than 100-ms are not practical due to the monitor overhead involved in processing the interrupt. Supporting the programmable clock are the Interrupt Interval (II) register, the Interrupt Countdown (ICD) counter, and four monitor mode instructions.

INSTRUCTIONS

Four monitor mode instructions support the programmable clock:

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
0014j4	PCI Sj	Enter Interrupt Interval (II) register with (Sj)

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
001405	CCI	Clear the programmable clock interrupt request
001406	ECI	Enable the programmable clock interrupt request
001407	DCI	Disable the programmable clock interrupt request

INTERRUPT INTERVAL REGISTER

The 32-bit Interrupt Interval (II) register can be 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 *Sj* register into the II register and the ICD counter when instruction 0014j4 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. The II register contents is changed only by another instruction 0014j4.

INTERRUPT COUNTDOWN COUNTER

The 32-bit ICD counter is preset to the II register contents when instruction 0014j4 is executed. This counter runs continuously but counts down, decrementing by 1 each CP until the counter content 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 zero cycle, setting the programmable clock interrupt request at regular intervals determined by the interval value.

When the programmable clock interrupt request is set, it remains set until a 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 001405.

Following any deadstart, the monitor program should ensure the state of the programmable clock interrupt by issuing instructions 001405 and 001407.

PERFORMANCE MONITOR

The system contains a set of eight performance counters to track certain hardware related events that can be used to indicate relative performance. The events that can be tracked are the number of specific instructions issued, hold issue conditions, the number of fetches, references, and so on. and are selected through instruction 0015j0. Refer to appendix C for complete information on performance monitoring.

DEADSTART SEQUENCE

The deadstart sequence of operations starts a program running in the mainframe after power has been turned off and then turned on again or whenever the operating system is to be reinitialized in the mainframe. All registers in the machine, all control latches, and all words in memory should be considered invalid after power has been turned on. The IOS initiates the following sequence of operations to begin the program:

1. Turns on Master Clear signal
2. Turns on I/O Clear signal
3. Turns off I/O Clear signal
4. Loads memory via IOS
5. Turns 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 CA register of the MCU channel and activates the MCU input channel. All other input channels remain inactive. The IOS 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 in CPU 0 (PN=0).

The CPU's 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. Vector and scalar 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, and produces a single result.

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 the computation section is 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.

[†] Five vector functional units are available on systems equipped with a Second Vector Logical unit.

The computation section performs integer or floating-point arithmetic operations. 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 count.

Characteristics of the computation section are summarized as follows.

- 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

† Fourteen functional units if the system is equipped with a Second Vector Logical unit.

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 the functional units. A single functional unit can require one to three operands per clock period (CP) to perform the necessary functions and can deliver 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 as A, S, and V, respectively. These registers are considered primary because functional units can access them directly.

For the A and S registers, an intermediate level of registers exists which is not accessible to the functional units but acts as a buffer for the 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. The intermediate registers that support the A registers are referred to as B registers. The intermediate registers that support S registers are referred to as T registers.

ADDRESS REGISTERS

Figure 4-1 shows 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 used as address registers for memory references and as index registers. They 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 address increment for vector memory references.

The address functional units support address and index generation by performing 24-bit integer arithmetic on operands obtained from A registers and by delivering the results to A registers.

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 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 5.

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

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
010 <i>ijkm</i>	JAZ <i>exp</i>	Branch to <i>ijkm</i> if (A0)=0
011 <i>ijkm</i>	JAN <i>exp</i>	Branch to <i>ijkm</i> if (A0)≠0
012 <i>ijkm</i>	JAP <i>exp</i>	Branch to <i>ijkm</i> if (A0) is positive, includes (A0)=0
013 <i>ijkm</i>	JAM <i>exp</i>	Branch to <i>ijkm</i> if (A0) is negative
034 <i>ijk</i>	<i>Bjk,Ai</i> ,A0	Read (<i>Ai</i>) words to B register <i>jk</i> from (A0)
035 <i>ijk</i>	,A0 <i>Bjk,Ai</i>	Store (<i>Ai</i>) words at B register <i>jk</i> to (A0)
036 <i>ijk</i>	<i>Tjk,Ai</i> ,A0	Read (<i>Ai</i>) words to T register <i>jk</i> from (A0)
037 <i>ijk</i>	,A0 <i>Tjk,Ai</i>	Store (<i>Ai</i>) words at T register <i>jk</i> to (A0)
176 <i>i0k</i>	<i>Vi</i> ,A0, <i>Ak</i>	Read (VL) words to <i>Vi</i> from (A0) incremented by (<i>Ak</i>)
176 <i>i1k</i>	<i>Vi</i> ,A0, <i>Vk</i>	Read (VL) words to <i>Vi</i> using (A0) + (<i>Vk</i>)
1770 <i>jk</i>	,A0, <i>Ak</i> <i>Vj</i>	Store (VL) words from <i>Vj</i> to (A0) incremented by (<i>Ak</i>)
1771 <i>jk</i>	,A0, <i>VkVj</i>	Store (VL) words from <i>Vj</i> using (A0) + (<i>Vk</i>)

Section 5 contains additional information on the use of A registers by instructions.

B REGISTERS

The computation section contains sixty-four 24-bit B registers used as intermediate storage for the 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 of uses 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 can be transferred to or from Central Memory at the maximum rate of one 24-bit value per CP. A reservation is made on all B registers during block transfers to and from B registers.

NOTE

Other instructions can issue on the CRAY X-MP while a block of B registers is being transferred to or from Central Memory.

B registers are individually referred to by the letter B followed by a 2-digit number ranging from 00₈ through 77₈. Instructions reference B registers by specifying the B register number in the *jk* designator as described in section 5.

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

SCALAR REGISTERS

Figure 4-2 shows 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.

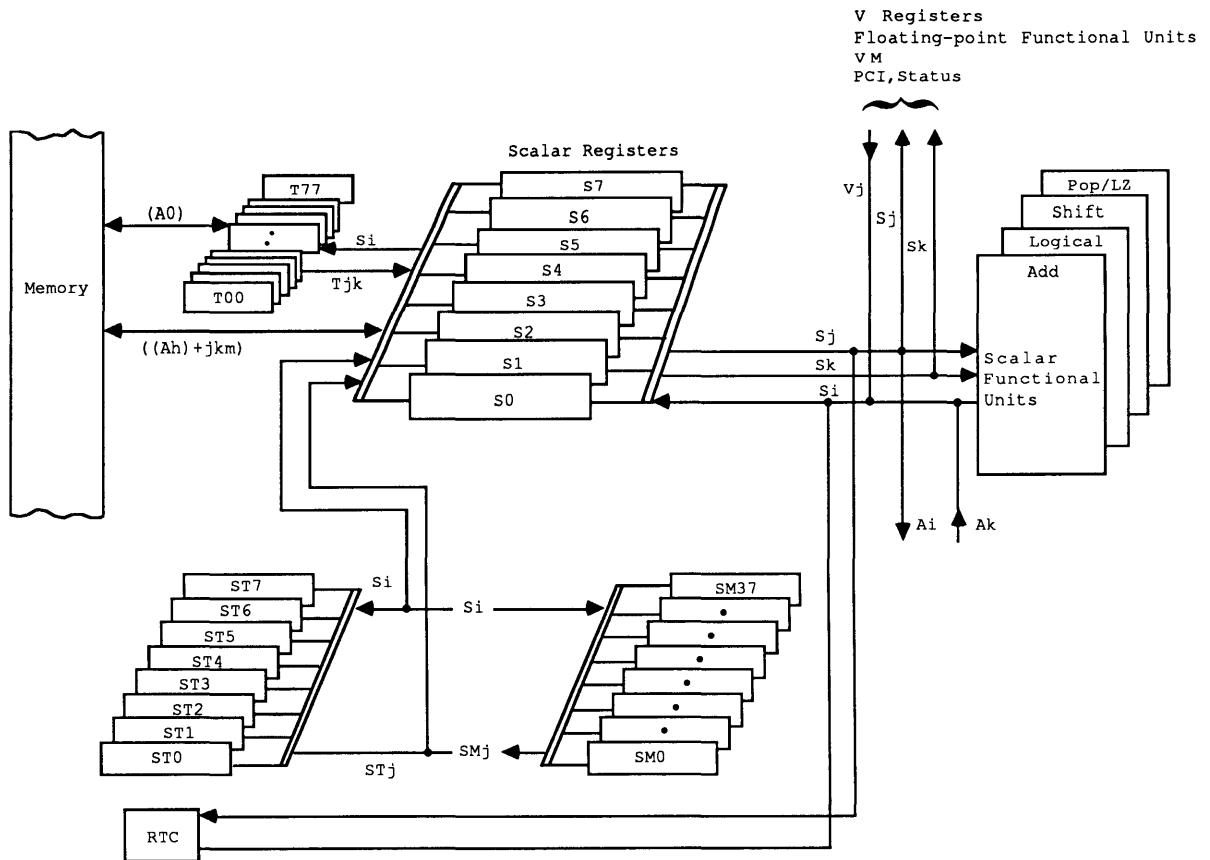


Figure 4-2. Scalar Registers and Functional Units

S REGISTERS

Eight 64-bit S registers are the principal scalar registers for the CPU serving as the 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.

Data is moved 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, between S and Shared Scalar (ST) registers, and between S and Semaphore (SM) registers.

Other uses of the S registers are the setting or reading of the Vector Mask (VM) register or the Real-time Clock (RTC) register or setting the Interrupt Interval (II) register.

When an instruction delivering new data to an S register issues, a reservation is set for that register preventing issue of instructions that read the register until the new data is delivered.

In this manual, the 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 5.

The only register implicitly referenced is the S0 register, as illustrated in the following instructions.

<u>Octal Code</u>	<u>CAL Syntax</u>	<u>Description</u>
014 <i>ijklm</i>	JSZ <i>exp</i>	Branch to <i>ijklm</i> if (S0)=0
015 <i>ijklm</i>	JSN <i>exp</i>	Branch to <i>ijklm</i> if (S0)≠0
016 <i>ijklm</i>	JSP <i>exp</i>	Branch to <i>ijklm</i> if (S0) is positive, includes (S0)=0.
017 <i>ijklm</i>	JSM <i>exp</i>	Branch to <i>ijklm</i> if (S0) is negative.
052 <i>ijk</i>	S0 <i>Si</i> < <i>exp</i>	Shift (<i>Si</i>) left <i>jk</i> places to S0
053 <i>ijk</i>	S0 <i>Si</i> > <i>exp</i>	Shift (<i>Si</i>) right <i>jk</i> places to S0

The Status register provides the status of the following flags:

- Processor Number (PN)
- Program State (PS)
- Clustered, CLN ≠ 0 (CL)
- Floating-point Interrupts Enabled (IFP)
- Floating-point Error (FPE)
- Bidirectional Memory Enabled (BDM)
- Operand Range Interrupts Enabled (IOR)
- Cluster number bits 2⁰ through 2³ (CLN)

Instruction 073 sends the contents of the Status register to an S register.

Section 5 of this manual has additional information on the use of S registers by instructions.

T REGISTERS

The computation section has sixty-four 64-bit T registers used as intermediate storage for the 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. A reservation is made on all T registers during block transfers to and from T registers.

NOTE

Other instructions can issue on the CRAY X-MP while a block of T registers is being transferred to or from Central Memory.

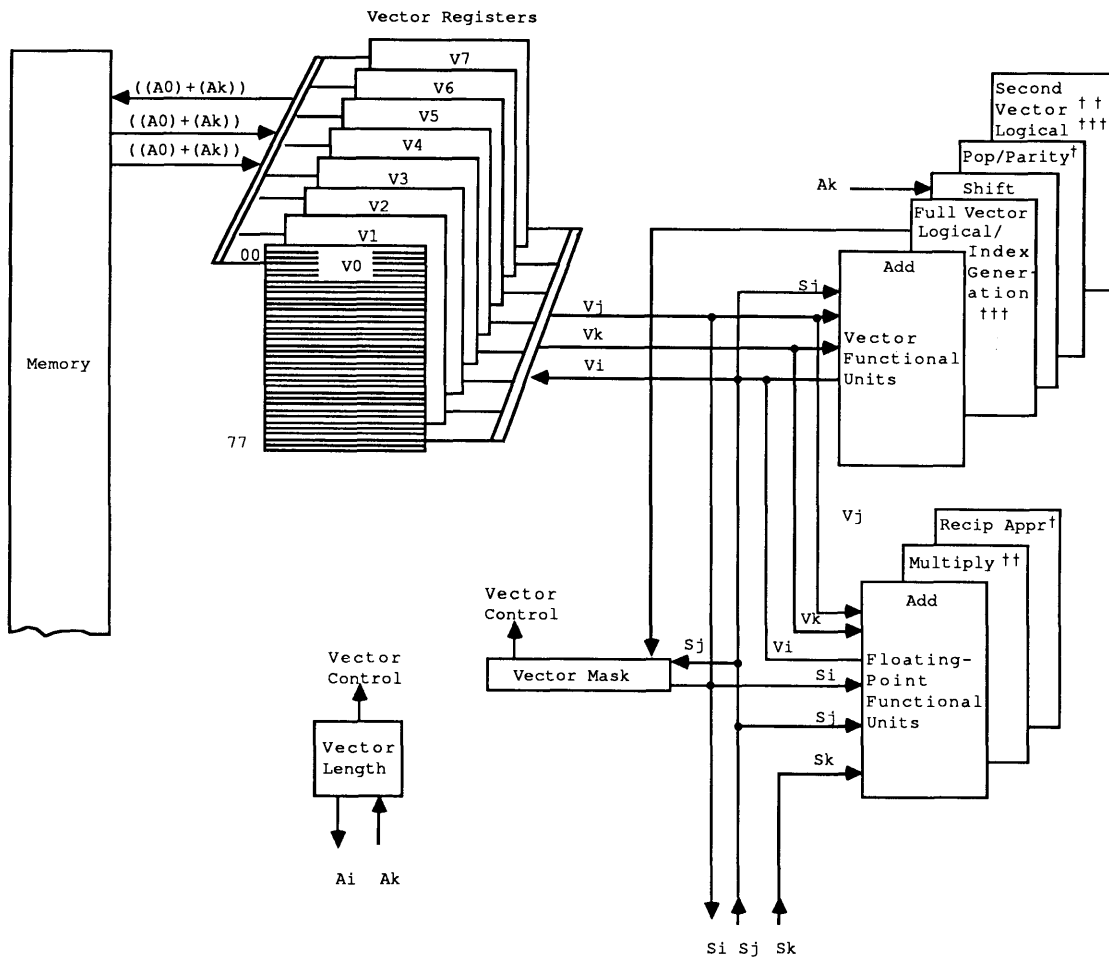
T registers are referred to by the letter T and a 2-digit number ranging from 00₈ through 77₈. Instructions reference T registers by specifying the octal number as the *jk* designator as described in section 5.

VECTOR REGISTERS

Figure 4-3 illustrates the registers and functional units used for vector operations. The following paragraphs describe the Vector registers and Vector Control registers.

V REGISTERS

The major computational registers of the CPU are eight V registers, each with 64 elements. Each V register element has 64 bits. When associated data is grouped into successive elements of a V register, the register quantity can be treated as a vector. Examples of vector quantities are rows or columns of a matrix or elements of a table. Computational efficiency is achieved by identically processing each element of a vector. Vector instructions provide for the iterative processing of successive V register elements. A vector operation always begins when operands are obtained from the first element of the operand V registers and the result is delivered to the first element of a V register.



- † The Vector Pop/Parity shares its input path with the Reciprocal Approximation unit.
- †† The Second Vector Logical shares its input and output path with the Floating-point Multiply unit.
- ††† Second Vector Logical and Index Generation are not available on all systems.

Figure 4-3. Vector Registers and Functional Units

Successive elements are provided each CP and as each operation is performed, the result is delivered to successive elements of the result V register. The vector operation continues until the number of operations performed by the instruction equals a count specified by the VL register contents.

V register contents 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. The transfer then proceeds beginning with the first element of the V register at a maximum rate of 1 word per CP, depending upon bank conflicts. Discontinuities in the vector data stream can occur as a result of memory conflicts. These discontinuities, although not inhibiting chained

operations, can appear in the chained operation data stream. Any discontinuity in the data stream adds proportionally to the total execution time of the vector operation.

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

Since many vectors exceed 64 elements, a long vector is processed as one or more 64-element segments and a possible remainder of less than 64 elements. Generally, it is convenient to compute the remainder and process this short segment before processing the remaining number of 64-element segments. A programmer, however, 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 an operand register in two different operations, allows for the chaining together of two or more vector operations, and two or more results can be produced per CP. The CPU automatically detects chained operations, as they are not explicitly specified by the programmer. A programmer can reorder certain code segments to gain as much concurrency as possible in chained operations.

A conflict can occur between vector and scalar operations involving either floating-point operations or 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 the vector is processed.

Parallel vector operations can be processed in two ways:

- 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 the generation of two or more results per CP. Most vector operations use two V registers as operands, or one S and one V register as operands. Exceptions are vector shifts, vector logicals, vector reciprocals, and the load or store instructions.

The 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 5.

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, V6₂₉ refers to element 29 of V register 6.

NOTE

Parallel loading and storing of V registers is possible; two load operations and one store operation can occur simultaneously.

V register reservations and chaining

Reservation describes the condition of a register in use; that is, the register is not available for another operation as a result or as an operand register. Each register has two reservation conditions, one reserving it as an operand register and one reserving it as a result register. During execution of a vector instruction, reservations are placed on the operand V registers and on the result V register. These reservations are placed on the registers themselves, not on individual elements of the V register.

If a V register is reserved as a result and not as an operand, it can be used at any time as an operand and chaining occurs. This flexible chaining mechanism allows chaining to begin at any point in the result vector data stream. Full chaining occurs if the instruction causing chaining is issued before or at the time element 0 of the result arrives at the V register. Partial chaining occurs if the instruction issues after the arrival of element 0. Thus, the amount of concurrency in a chained operation depends upon the relationship between the issue time of the chaining instruction and the result vector data stream.

If a V register is reserved as an operand, it cannot be used as a result or operand register until the operand reservation clears. A V register can be used, however, as both an operand and result in the same vector operation. A V register can serve only one vector operation as the source of one or both operands. A V register can serve only one vector operation as a result.

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 length and scalar operand (if appropriate) of each vector operation is maintained apart from the VL register and S register. Vector operations employing different lengths can proceed concurrently.

Even when a vector load operation pauses, allowing instructions to get synchronized, a few cycles later chained operations may proceed as soon as data becomes available. (Thus, if a late chain slot is made, the loop might run at full speed.)

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

CAUTION

CRI cautions against using a vector register as both a result and an operand if compatibility between a CRAY-1 and a CRAY X-MP computer system is necessary because vector recursion is not available on all Cray computer systems.

VECTOR CONTROL REGISTERS

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

Vector Length register

The 7-bit VL register is set to 1 through 100₈ (VL = 0 gives VL = 100₈) 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 or read using instruction 023i01.

Vector Mask register

The VM register has 64 bits, each corresponding to a word element in a V register. Bit 2⁶³ corresponds to element 0, bit 2⁰ 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 VM register contents to an S register.

FUNCTIONAL UNITS

Instructions other than simple transmits or control operations are performed by specialized hardware known as functional units. Each unit implements an algorithm or a portion of the instruction set. Functional

units have independent logic except for the Reciprocal Approximation, and Vector Population Count units (described later in this section), which share some logic. (On systems equipped with a Second Vector Logical unit, the Floating-point Multiply and Second Vector Logical units share input and output paths.) All functional units can be in operation simultaneously.

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 three-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 the operands have been delivered to the unit. Time required from delivery of the operands to the functional unit until completion of the calculation is called the functional unit time and is measured in CPs.

Functional units are fully segmented. That is, a new set of operands for unrelated computation can enter a functional unit each CP even though the functional unit time can be 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.

The functional units identified are arbitrarily described in four groups: address, scalar, vector, and floating-point. Each of the first three groups functions with one of the primary register types (A, S, and V) to support the address, scalar, and vector modes of processing available in the mainframe. 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 can also act as a 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. The unit 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. The Address Add functional unit detects no overflow.

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 unit does not detect overflow of the product and significant portions of the product could be lost.

The Address Multiply functional unit time is 4 CPs.

SCALAR FUNCTIONAL UNITS

Scalar functional units perform operations on 64-bit operands obtained from S registers and usually deliver the 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 they are described in the section 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. 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. The Scalar Add functional unit detects no overflow.

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

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 performs bit-by-bit manipulation of 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 one or two V registers or from a V register and an S register. The Reciprocal, 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 to a functional unit. The corresponding result emerges from the functional unit *n* CPs later, where *n* is the functional unit time and is constant for a given functional unit. The VL register determines the number of operand pairs to be processed by a functional unit.

The 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 entitled 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 do not issue until the previous operation is completed. Only one functional unit of each type is available to the vector instruction hardware (with the exception of systems equipped with a Second Vector Logical unit where instructions 140 through 145 may use either of the vector logical units). When the vector operation completes, the reservation is dropped and the functional unit is then available for another operation. A vector functional unit is reserved for $(VL) + 4$ CPs.

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 V_k operand is complemented before addition and a 1 is added into the low-order bit position of the result. The unit detects no overflow.

The Vector Add functional unit time is 3 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.

All shift counts are considered positive unsigned integers. If any bit higher than 2^6 is set, the shifted result is all zeros.

The Vector Shift functional unit executes instructions 150 through 153. The functional unit time is 4 CPs for instruction 152, and the functional unit time is 3 CPs for instructions 150, 151, and 153.

Full Vector Logical functional unit

The Full Vector Logical functional unit performs a bit-by-bit manipulation of the 64-bit quantities for instructions 140 through 147. The Full 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 instructions.

NOTE

If the system is equipped with a Second Vector Logical unit and the unit is enabled, instruction 175 can be chained with instructions 140 through 145. For this to happen, the 140 through 145 instructions must use the Second Vector Logical functional unit and not the Full Vector Logical unit.

The Full Vector Logical functional unit time is 2 CPs.

Second Vector Logical functional unit†

The Second Vector Logical functional unit performs a bit-by-bit manipulation of the 64-bit quantities for instructions 140 through 145. At the time of CIP for a 140 through 145 instruction, a selection is made as to which of the two vector logical functional units to use: the Full Vector Logical functional unit or the Second Vector Logical functional unit. If the Second Vector Logical unit is enabled (through the Exchange Package), instructions 140 through 145 attempt to issue there first. If the unit is busy, issue is attempted to the Full Vector Logical unit. When both units are busy, the first unit to clear is selected for issue. Instructions issue to the Full Vector Logical unit first, even though the Second Vector Logical unit is not busy, if another conflict is present for the Second Vector Logical unit (for example, a register reservation).

The Second Vector Logical functional unit can be disabled through software by clearing bit 0 of word 3 in the Exchange Package of a user program. When the Second Vector Logical unit is disabled (by clearing the Enable Second Vector Logical bit in the Exchange Package), the functional unit Busy signal for the unit always appears to be set and causes all 140 through 145 instructions to use the Full Vector Logical unit.

† Not available on all single-processor systems

NOTE

Since the Second Vector Logical and Floating-point Multiply functional units share input and output datapaths, they cannot be used simultaneously. When the Second Vector Logical unit is enabled, the two units share the same functional unit Busy signal. Also, because using the Second Vector Logical functional unit ties up the Floating-point Multiply functional unit, some codes that rely on floating-point products may run slower if the Second Vector Logical functional unit is enabled. If the Floating-point Multiply is busy and the Full Vector Logical is not busy, the Vector Logical instruction uses the Full Vector Logical functional unit.

The Second Vector Logical functional unit 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, as shown by its low-order bit.

Instructions 174ij1 (vector population count) and 174ij2 (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 (refer to the 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 5 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.

The subsection on Floating-point Arithmetic contains information on floating-point out-of-range conditions.

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. (The subsection on Floating-point Arithmetic describes normalized floating-point numbers.) Out-of-range exponents are detected as described in the subsection on Floating-point Arithmetic.

Floating-point Add functional unit time is 6 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- 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.

NOTE

On systems equipped with the Second Vector Logical functional unit, the Floating-point Multiply and Second Vector Logical functional units cannot be used simultaneously since they share input and output data paths. A reservation on one is a reservation on the other.

Out-of-range exponents are detected as described in the subsection on floating-point arithmetic. If both operands have zero exponents, however, 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 must be shifted before the multiply operation.

The Floating-point Multiply functional unit time is 7 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 174ij0. 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. Out-of-range exponents are detected as described under Floating-point Arithmetic.

The Reciprocal Approximation functional unit time is 14 CPs.

ARITHMETIC OPERATIONS

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

INTEGER ARITHMETIC

All integer arithmetic, whether 24 bits or 64 bits, is twos complement and is represented in the registers as illustrated in figure 4-4. 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.

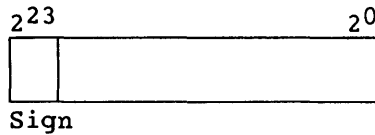
Multiplication of two scalar (64-bit) integer operands is accomplished by using the floating-point multiply instruction and one of the two methods that follows. The method used depends on the magnitude of the operands and the number of bits to contain the product.

If the operands 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 (refer to figure 4-8). If the operand coefficients are generated by other than 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 as a truncation compensation constant is always added during a 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.

Twos Complement Integer (24 bits)



Twos Complement Integer (64 bits)

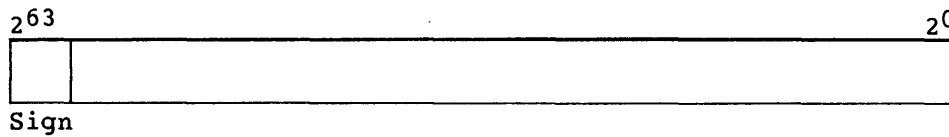


Figure 4-4. Integer Data Formats

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 4-5. Since the coefficient is signed magnitude, it is not complemented for negative values.

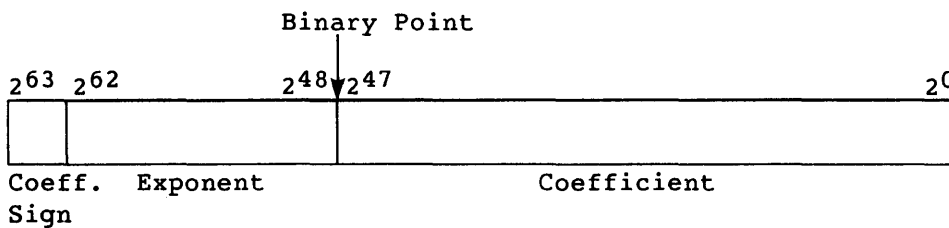


Figure 4-5. Floating-point Data Format

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 (the negative range is one larger):

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

In terms of decimal values, the floating-point format of the system allows the accurate expression of numbers to about 15 decimal digits in the approximate decimal range of 10^{-2466} through 10^{+2465} .

Figure 4-6 and the following steps shows the relationship between the bias, exponent, and coefficient. To convert the number to its decimal equivalent:

1. Subtract the bias from the exponent to get the integer value of the exponent:

$$\begin{array}{r} -40000 \\ \hline 1 \end{array}$$

2. Multiply 2 raised to the integer value of the exponent by the normalized coefficient, expressed as a fraction, to get the result:

$$\begin{array}{r} 2^1 \\ \times 0.4_8 \\ \hline 1.0 \end{array}$$

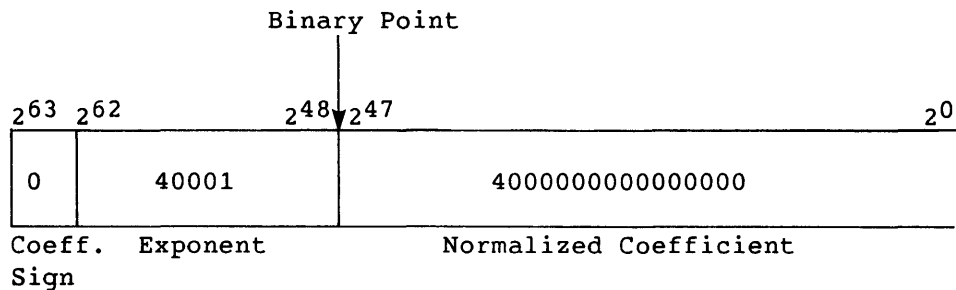


Figure 4-6. Internal Representation of Floating-point Number (Octal)

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

A negative zero is not generated by any floating-point functional unit, except in the case where a negative zero is one operand going into the Floating-point Multiply functional unit.

The remainder of this subsection describes normalized floating-point numbers, floating-point range errors, double-precision numbers, and the addition, multiplication, and division algorithms.

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 Sj field of an instruction, an operand in Sk is normalized using the $062i0k$ instruction. Si , which can be Sk , contains the normalized result.

The $170i0k$ instruction normalizes Vk into Vi .

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 put the appropriate bit 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 60000₈. This condition sets the Floating-point Error flag with an exponent of 60000₈ being sent to the result register along with the computed coefficient, as in the following example:

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

NOTE

If a floating-point add or subtract generates an exponent less than 20000₈ or a coefficient of 0, the condition is considered an underflow, no fault is generated, and the word returned from the functional unit is all 0 bits. If either operand is out-of-bounds (exponent of 60000₈ or greater) or if the final sum or difference is out-of-bounds (exponent of 60000₈ or greater), the exponent is set to 60000₈ and a floating-point error is flagged. If floating-point faults are enabled, an interrupt occurs. Refer to the floating-point range errors subsection for more information.

Floating-point Multiply functional unit - Whether or not out-of-range conditions occur, and how they are handled, can be determined using the exponent matrix shown in figure 4-7. The exponent of the result, for any set of exponents, falls into one of seven unique zones. A description of each zone follows.

NOTE

Only zones 6 and 7 can generate floating-point faults.

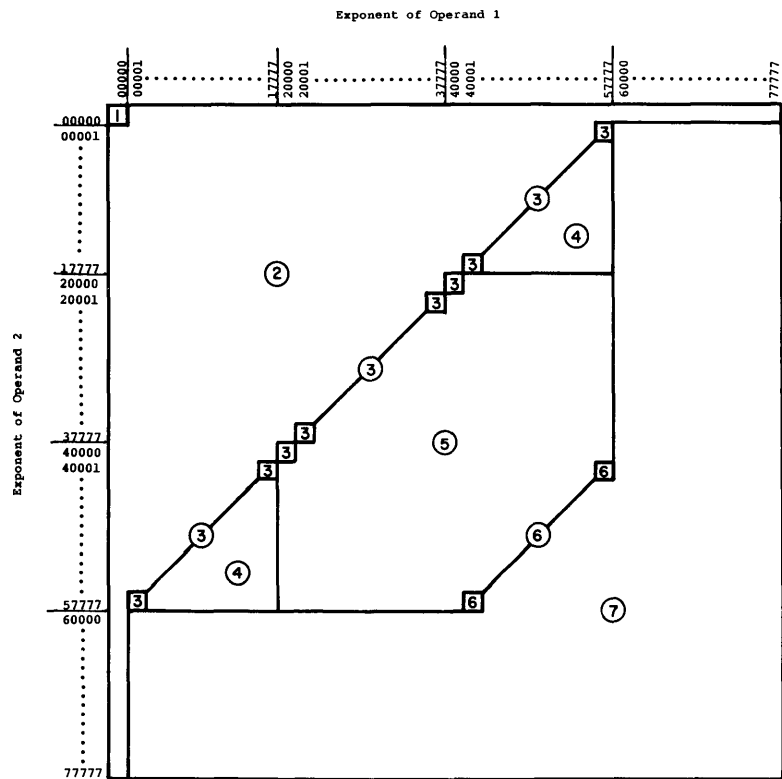


Figure 4-7. Exponent Matrix for Floating-point Multiply Unit

<u>Zone</u>	<u>Description</u>
1	This indicates a simple integer multiply; no fault is possible.
2	These exponents would result in an underflow condition. It is flagged as such, and the result is set to +0. (Multiply by 0 is in this group.)
3	Underflow may occur on this boundary. When a normalize shift is required, the underflow is not detected, and the coefficient and exponent are not zeroed out. The exponent used before the shift is 20000_8 ; the exponent used after the shift is 17777_8 . Underflow detection is done on the exponent used for an unshifted product coefficient.
4	The use of an operand with an underflow exponent is allowed if the final result operand is within the range 20000_8 to 57777_8 .
5	This is the normal operand range and normal results are produced.

<u>Zone</u>	<u>Description</u>
6	Overflow is flagged on this boundary. If a normalized shift is required, the value should be within bounds with a 57777_8 exponent. Since overflow is detected, however, using the exponent for the unnormalized shift condition (which is 60000_8), a 60000_8 is inserted in the product as the final exponent.
7	Within this zone, an overflow fault is flagged and the product exponent is set to 60000_8 .

Out-of-range conditions are tested before normalizing in the Floating-point Multiply functional unit. As shown, 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 4-7, if operand 1 is 4 and operand 2 is 6, a 48-bit result of 30_8 is produced. Bit 2^{63} obeys the usual rules for multiplying signs and the result is a sign and magnitude integer. The form of integers (refer to figure 4-4) 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 4-8 have any 1 bits, the product might be one (2^0) too large 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 (figure 4-8) 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 areas).

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.

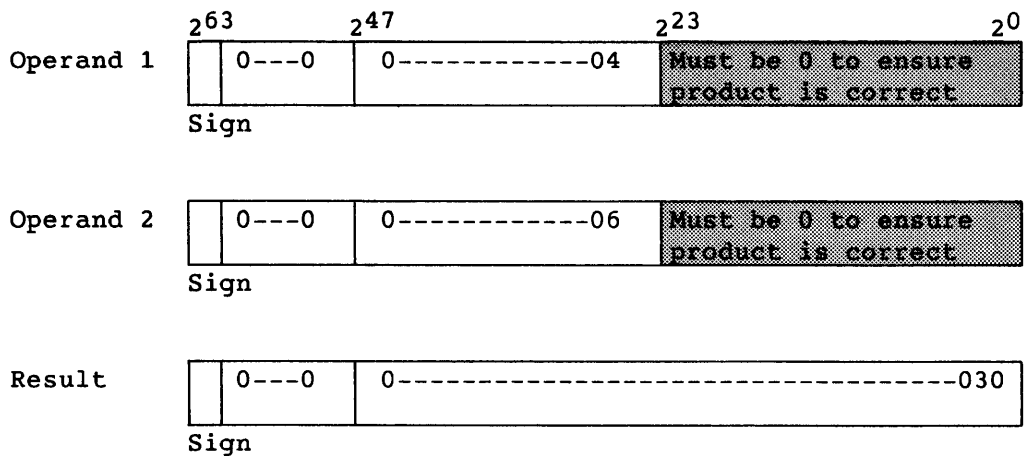


Figure 4-8. Integer Multiply in Floating-point Multiply Functional Unit

Double-precision numbers

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

Addition algorithm

Floating-point addition or subtraction is performed in a 49-bit register (figure 4-9). 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 roundup occurs. 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.

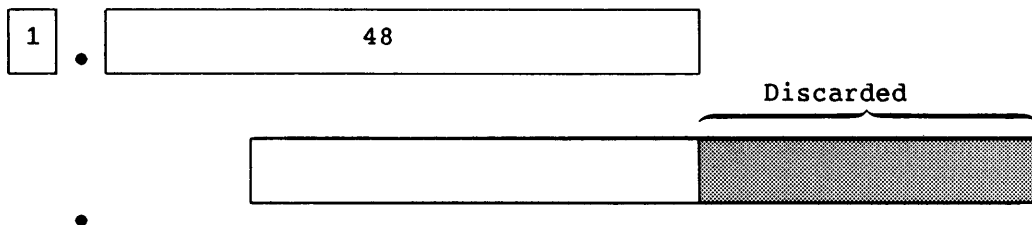


Figure 4-9. 49-bit Floating-point Addition

The Floating-point Add functional unit normalizes any floating-point number within the format of the mainframe's 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

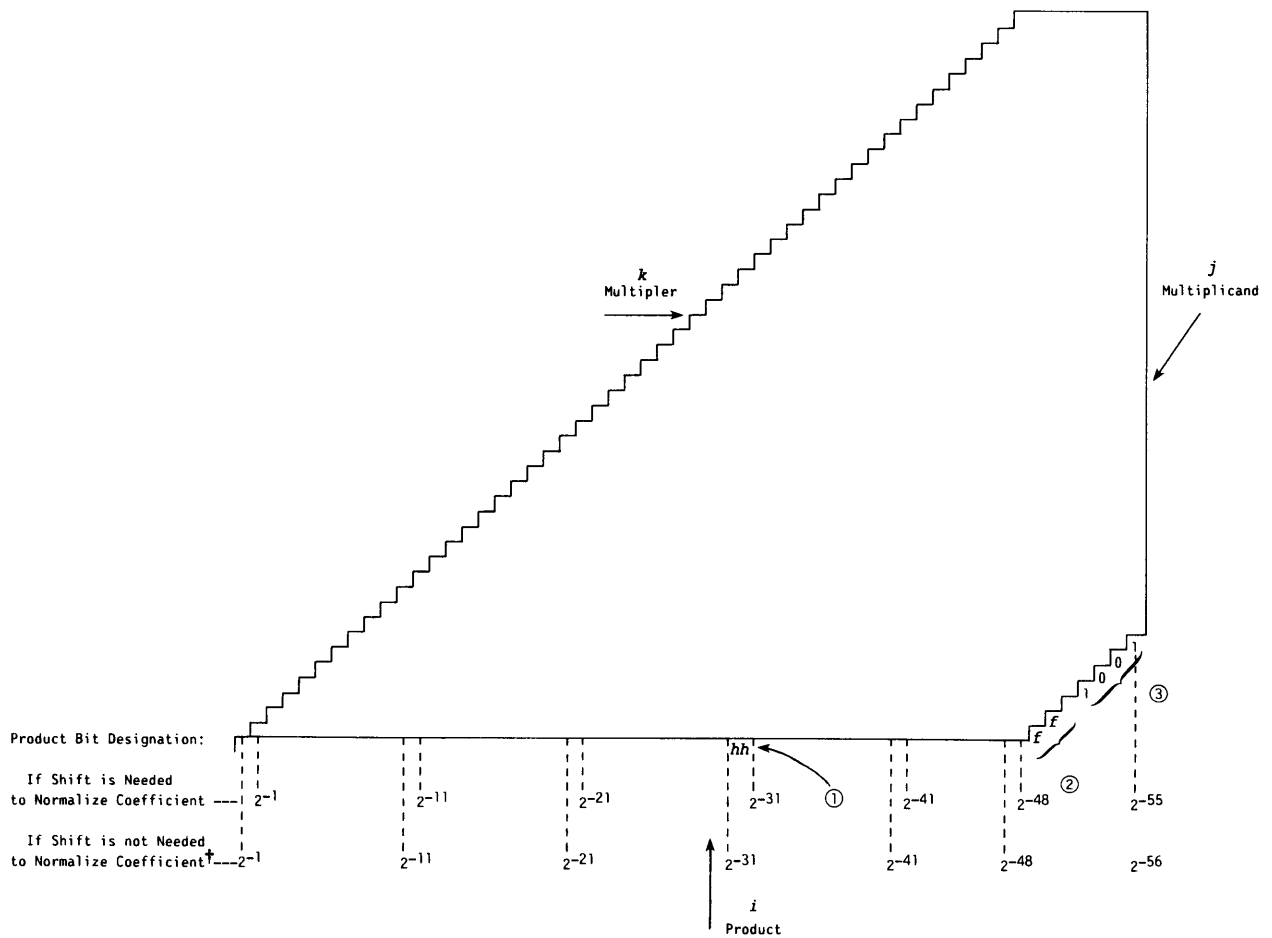
The Floating-point Multiply functional unit has the two 48-bit coefficients as input into a multiply pyramid (refer to figure 4-10). If the coefficients are both normalized, then a full product is either 95 bits or 96 bits, depending on the value of the coefficients. A 96-bit product is normalized as 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 1 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 system, 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 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 (0.25×2^{-48} ; high) 38 percent of the time and 25 percent low (0.125×2^{-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.



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

Figure 4-10. Floating-point Multiply Partial-product Sums Pyramid

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

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:

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

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 system 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 system 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 $F(x) \neq 0$, 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 (refer to figure 4-11). 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 , and so on.

Derivation of the division algorithm

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

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

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

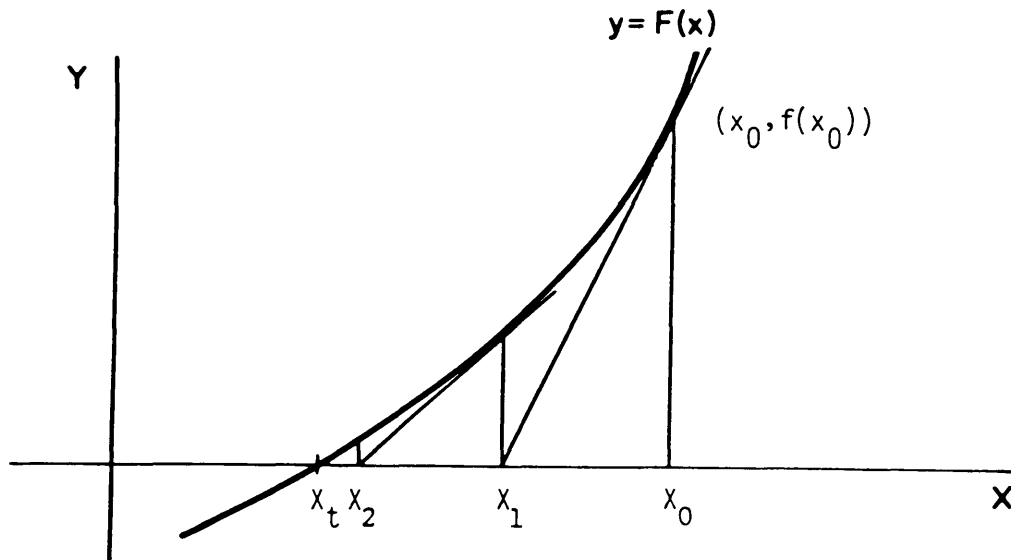


Figure 4-11. Newton's Method

For any point x_i near to x_t ,

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

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 equation above can be replaced by 0, giving the following approximation:

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

$x_t - x_i$ 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:

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

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. The operation is, however, repeated until the answer becomes sufficiently close for practical use.

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

First calculating $F'(x)$, where:

$$F'(x) = \left(\frac{1}{x} - B\right)' = \left(-\frac{1}{x^2}\right)$$

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

and applying equation (1),

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

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

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

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

On the system, x_1 times the quantity in parentheses is performed by a floating-point multiply. $2-x_1B$ is performed by the reciprocal approximation instruction. x_1 is the x near $1/B$ and is formed by the half-precision reciprocal approximation instruction.

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

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

The system's Reciprocal Approximation functional unit performs three iterations: I1, I2, and I3. I1 is accurate to 8 bits and is found after a table lookup 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:

<u>Operation</u>	<u>Performed By</u>
$S3 = 1/S2$	The Reciprocal Approximation functional unit
$S4 = (2 - (S3 * S2))$	The Floating-point Multiply functional unit in iteration mode
$S5 = S4 * S3$	The Floating-point Multiply functional unit using full-precision; S5 now equals $1/S2$ to 48-bit accuracy.
$S6 = S5 * S1$	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.

<u>Operation</u>	<u>Performed By</u>
S3 = 1/S2	The Reciprocal Approximation functional unit
S6 = S1 * S3	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:

<u>Operation</u>	<u>Performed By</u>
S3 = 1/S2	The Reciprocal Approximation functional unit
S5 = S1 * S3	The Floating-point Multiply functional unit
S4 = (2 - (S3 * S2))	The Floating-point Multiply functional unit
S6 = S4 * S5	The Floating-point Multiply functional unit

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.

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.)

For example, two 64-element vectors may be 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	<u>1 1 0 0</u>
Result	1 0 0 0

A logical sum is the inclusive OR function:

Operand 1	1 0 1 0
Operand 2	<u>1 1 0 0</u>
Result	1 1 1 0

A logical difference is the exclusive OR function:

Operand 1	1 0 1 0
Operand 2	<u>1 1 0 0</u>
Result	0 1 1 0

A logical equivalence is the exclusive NOR function:

Operand 1	1 0 1 0
Operand 2	<u>1 1 0 0</u>
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	<u>1 1 1 1 0 0 0 0</u>
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.



This section explains the instruction formats and the specific instructions for the CRAY X-MP single-processor computer systems.

INSTRUCTION FORMAT

Each instruction used in the computer is either a 1-parcel (16-bit) instruction or a 2-parcel (32-bit) instruction. Instructions are packed 4 parcels per word. Parcels in a word are numbered 0 through 3 from left to right and any parcel position 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 5-1 illustrates the general form of instructions.

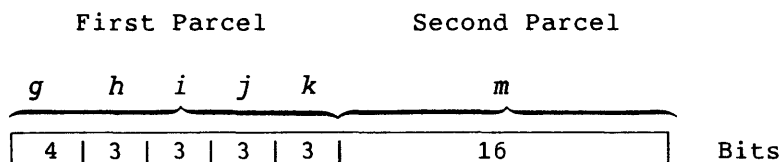


Figure 5-1. General Form for Instructions

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 (refer to figure 5-2). 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

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.

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. (Refer to the subsection on Special Register Values.)

Figure 5-4 shows the two general applications for the 2-parcel instruction format with combined *j*, *k*, and *m* fields.

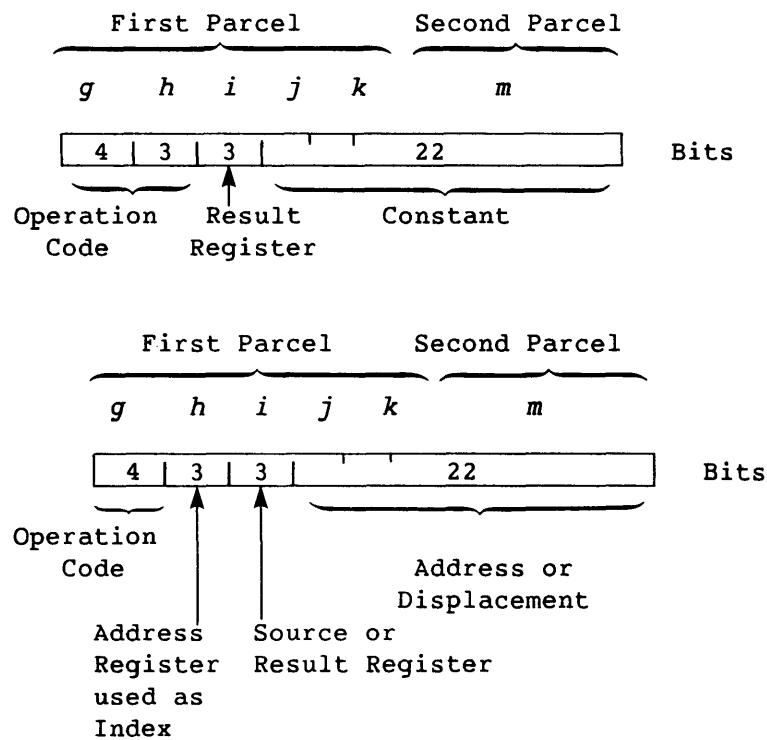


Figure 5-4. 2-parcel Instruction Format with Combined *j*, *k*, and *m* Fields

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

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

The 2-parcel instruction type for a 24-bit immediate constant (figure 5-6) uses the combined *i*, *j*, *k*, and *m* fields to hold the constant. This instruction type uses the 4-bit *g* field for an operation code and the 3-bit *h* field to designate the result address register. The high-order bit of the *i* field is set.

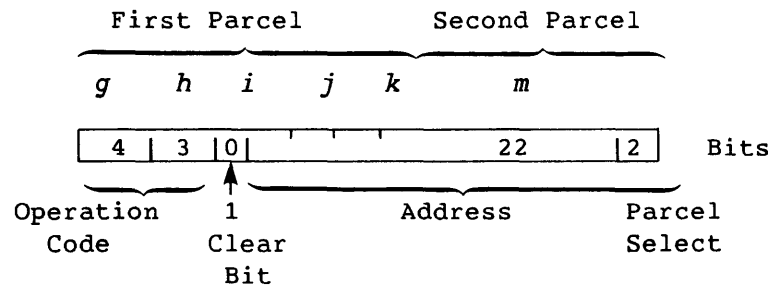


Figure 5-5. 2-parcel Instruction Format with Combined *i*, *j*, *k*, and *m* Fields

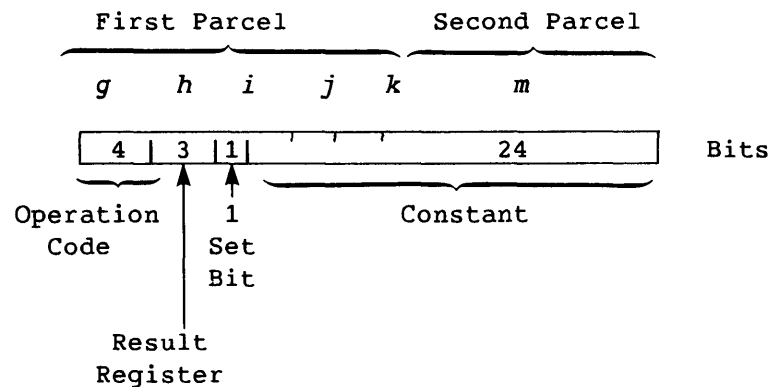


Figure 5-6. 2-parcel Instruction Format for a 24-bit Immediate Constant with Combined *i*, *j*, *k*, and *m* Fields

SPECIAL REGISTER VALUES

If the S0 and A0 registers are referenced in the *j* or *k* fields of an instruction, the respective register contents are not used; instead, a special operand is generated. The special value is available regardless of existing A0 or S0 reservations (which, 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. Table 5-1 shows the special register values.

Table 5-1. 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 1 parcel at a time from the instruction buffers and delivered to the Next Instruction Parcel (NIP) register. The instruction is then 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 unit and registers are such that functions required for execution can be performed without conflicting with a previously issued instruction. Instruction parcels can issue out of the CIP register at a maximum rate of one per CP.

Execution times (the time from issue to delivery of data to the destination operating registers) are fixed for instructions 000 through 077, except those that reference memory (instructions 000, 004, branch instructions 005 through 017, and block transfer instructions 034 through 037). Scalar memory instructions 100 through 137 complete in variable lengths of time. Vector operation instructions 140 through 177 complete in a fixed time if the instructions are not chained to memory fetches.

Execution times can be affected by instruction 0034*jk*, which tests and sets the semaphore designated by *jk*. If the semaphore is set, instruction issue is held. If the semaphore is clear, the instruction issues and sets the semaphore. If the CPU is holding issue on a test and set, a flag is set in the Exchange Package (if not in monitor mode) and an exchange occurs. If an interrupt occurs while a test and set instruction is holding in the CIP register, a flag is set in the Exchange Package, CIP and NIP registers clear, and an exchange occurs with the P register pointing to the test and set instruction.

Entry to the NIP register is blocked for the second parcel of a 2-parcel instruction, leaving NIP blanked. 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 *Ah*, *Aj*, *Ak*, *Sj*, or *Sk*, the normal hold issue until operand ready conditions do not apply. These values are always immediately available.

INSTRUCTION DESCRIPTIONS

This section contains detailed information about individual instructions or groups of related instructions. Each instruction begins with boxed information consisting of the Cray Assembly 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 2 parcels.

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 CP *n*, the next instruction issues at CP *n* + issue time[†] if its own issue conditions have been met.

NOTE

The following instruction descriptions assume a 32-bank machine.

[†] Previous instruction issued

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

<u>Character</u>	<u>Description</u>
+	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.

<u>Letter</u>	<u>Description</u>
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

CAUTION

Instructions with *g*, *h*, *i*, *j*, *k*, and *m* fields not explicitly described in the following instructions may produce indeterminate results.

INSTRUCTION 000

CAL Syntax	Description	Octal Code
ERR	Error exit	000000

Instruction 000 is treated as an error condition and an exchange sequence occurs. The exchange sequence voids the contents of the instruction buffers. 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 Flag (F) register is set. All instructions issued before this instruction are run to completion. When results of previously issued instructions arrive at the operating registers, an exchange occurs to the Exchange Package designated by the Exchange Address (XA) register contents. The program address stored during the exchange on the terminating exchange sequence is the P register contents advanced by one count (that is, the address of the instruction following the error exit instruction).

HOLD ISSUE CONDITIONS: Any A, S, or V register reserved

EXECUTION TIME: Instruction issue, 51 CPs; this time includes an exchange sequence (32 CPs) and a fetch operation (19 CPs).

SPECIAL CASES: None

INSTRUCTIONS 0010 - 0013

CAL Syntax	Description	Octal Code
CA, <i>Aj</i> <i>Ak</i>	Set the Current Address (CA) register for the channel indicated by (<i>Aj</i>) to (<i>Ak</i>) and activate the channel	0010 <i>jk</i>
CL, <i>Aj</i> <i>Ak</i>	Set the Limit Address (CL) register for the channel indicated by (<i>Aj</i>) to (<i>Ak</i>)	0011 <i>jk</i>
CI, <i>Aj</i>	Clear the interrupt flag and error flag for the channel indicated by (<i>Aj</i>); clear device master-clear (output channel).	0012 <i>j0</i>
MC, <i>Aj</i>	Clear the interrupt flag and error flag for the channel indicated by (<i>Aj</i>); set device master-clear (output channel); clear device ready-held (input channel).	0012 <i>j1</i>
XA <i>Aj</i>	Enter the XA register with (<i>Aj</i>)	0013 <i>j0</i>

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 I/O 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 4 low-order bits of *Aj* are less than 10_8 , the functions are executed as pass instructions. Valid channel numbers are 10_8 through 17_8 . When the *k* designator is 0, CA or CL is set to 1.

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

Instruction 0012*j0* is used to clear the device Master Clear. For instruction 0012, if the *k* designator is 1 for an output channel, the master clear is set; if the *k* designator is 1 for an input channel, the ready flag is cleared.

INSTRUCTIONS 0010 - 0013 (continued)

HOLD ISSUE CONDITIONS: For instructions 0010 and 0011, *Aj* or *Ak* reserved (except *A0*)

For instructions 0012 or 0013, *Aj* reserved (except *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 effective.

For instructions 0010, 0011, and 0012:

If *j=0*, the instruction is a no-op.

If *k=0*, *CA* or *CL* is set to 1.

If 4 low-order bits of (*Aj*) are less than 10_8 , the instruction is a no-op, (that is, 20 through 27 are invalid, 30 through 37 are valid, 40 through 47 are invalid, 50 through 57 are valid, and so on).

If *k=0*, *CA* or *CL* is set to 1.

For instruction 0012:

The correct priority interrupting channel number cannot be read (through instruction 033) until 2 CPs after issue of instruction 0012.

For instruction 0013:

If *j=0*, *XA* register is cleared.

INSTRUCTION 0014

CAL Syntax	Description	Octal Code
RT <i>Sj</i>	Enter the Real-time Clock (RTC) register with (<i>Sj</i>)	0014 <i>j</i> 0
CLN 0	Cluster number = 0	001403
CLN 1	Cluster number = 1	001413
CLN 2	Cluster number = 2	001423
CLN 3	Cluster number = 3	001433
PCI <i>Sj</i>	Enter Interrupt Interval (II) register with (<i>Sj</i>)	0014 <i>j</i> 4
CCI	Clear the programmable clock interrupt request	001405
ECI	Enable programmable clock interrupt request	001406
DCI	Disable programmable clock interrupt request	001407

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

When the *k* designator is 0, the instruction loads the *Sj* register contents into the RTC register. When the *j* designator is 0 or (*Sj*)=0, the RTC register is cleared.

When the *k* designator is 3, the instruction sets the cluster number to *j* to make the following cluster selections:

CLN = 0 No cluster; all shared register and semaphore operations are no-ops, (except SB, ST, or SM register reads, which return a zero value to *Ai* or *Si*).

CLN = 1 Cluster 1

CLN = 2 Cluster 2

CLN = 3 Cluster 3

Clusters 1, 2, and 3 each have a separate set of SM, SB, and ST registers.

INSTRUCTION 0014 (continued)

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 the ICD counter. When the j designator is 0 or $(S_j)=0$, II and ICD are cleared.

When the k designator is 5, the instruction clears the programmable clock interrupt request if the request is previously set by ICD counting down 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 001406 is executed to enable the requests.

HOLD ISSUE CONDITIONS: S_j reserved (except S_0)

For instruction 0014j3, hold issue 2 CPs

EXECUTION TIME: Instruction issue, 1 CP

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

For instructions 0014j0 and 0014j4, if $j=0$, $(S_j)=0$.

For instruction 0014j0, the value is entered into the RTC register 4 CPs after instruction 0014j0 issues.

INSTRUCTION 0015

CAL Syntax	Description	Octal Code
†	Select performance monitor	0015j0
†	Set maintenance read mode	001501
†	Load diagnostic check byte with S1	001511
†	Set maintenance write mode 1	001521
†	Set maintenance write mode 2	001531

† Not currently supported

These instructions are all privileged to monitor mode.

Instruction 0015j0 selects one of four groups of hardware related events to be monitored by the performance counters. Refer to appendix C for a description of how performance monitoring is accomplished.

Instructions 001501 through 001531 are used to check the operation of the modules concerned with SECDED and to verify error detection and correction. The maintenance mode switch on the mainframe's control panel must be switched on during execution of these instructions or they become no-ops. Refer to appendix D for a description of SECDED maintenance mode functions.

Instructions 001501 and 001521 are used to verify check bit memory storage. Instruction 001501 allows the 8 check bits for SECDED to replace certain data bit positions in any subsequent memory read for the CPU path (including fetch and I/O). Instruction 001521 allows certain write data bits to replace the 8 check bits for SECDED for any subsequent CPU write to memory.

Instructions 001511 and 001531 are used to verify error detection and correction. Instruction 001511 loads a diagnostic check byte with the high-order 8 bits of S1. Instruction 001531 enables a diagnostic check byte to replace the 8 check bits for SECDED being written into memory for any subsequent write to memory.

INSTRUCTION 0020

CAL Syntax	Description	Octal Code
VL Ak	Transmit (Ak) to Vector Length register (VL)	00200k
VL 1†	Transmit 1 to VL register	002000

† Special CAL syntax

Instruction 00200k enters the VL register with a value determined by the contents of Ak. The low-order 6 bits of (Ak) are entered into the VL register. The 7th bit of VL is set if the 6 low-order bits of (Ak)=0.

For example, if (Ak)=0 or a multiple of 100₈, then VL=100₈. The contents of VL is always between 1 and 100₈.

Instruction 002000 transmits the value of 1 to the VL register.

HOLD ISSUE CONDITIONS: Ak reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP
VL register ready, 1 CP

SPECIAL CASES: Maximum vector length is 64.
(Ak)=1 if k=0.
(VL)=100₈ if k≠0 and (Ak)=0 or a multiple of 100₈

INSTRUCTIONS 0021 - 0027

CAL Syntax	Description	Octal Code
EFI	Enable interrupt on floating-point error	002100
DFI	Disable interrupt on floating-point error	002200
ERI	Enable interrupt on operand (address) range error	002300
DRI	Disable interrupt on operand (address) range error	002400
DBM	Disable bidirectional memory transfers	002500
EBM	Enable bidirectional memory transfers	002600
CMR	Complete memory references (CMR)	002700

Instruction 002100 sets the Floating-point Mode flag in the M register. Instruction 002200 clears the Floating-point Mode flag in the M register. The two instructions do not check the previous state of the flag. When set, the Floating-point Mode flag enables interrupts on floating-point range errors as described in section 4. Issuing either of these instructions also clears the Floating-Point Error Status flag.

Instruction 002300 sets the Operand Range Mode flag in the M register. Instruction 002400 clears the Operand Range Mode flag in the M register. The two instructions do not check the previous state of the flag. When set, the Operand Range Mode flag enables interrupts on operand (address) range errors as described in section 3.

Instruction 002500 disables the bidirectional memory mode. Instruction 002600 enables the bidirectional memory mode. Block reads and writes can operate concurrently in bidirectional memory mode. If the bidirectional memory mode is disabled, only block reads can operate concurrently.

Instruction 002700 assures completion of all memory references within the CPU. Instruction 002700 does not issue until all memory references before this instruction are at the stage of execution where completion occurs in a fixed amount of time. For example, a load of any data that has been stored by the CPU issuing instruction CMR, 002700, is assured of receiving the updated data if the load is issued after the CMR instruction.

INSTRUCTIONS 0021 - 0027 (continued)

HOLD ISSUE CONDITIONS: Instructions 002500 and 002600, hold issue 2 CPs

Instruction 002700, Ports A, B, and C busy

Instruction 002700, scalar memory reference active in CP 1, 2, or 3

Ak reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP

SPECIAL CASES: Instructions 002100 and 002200 are issued even if there are other floating-point operations in process resulting from previous issues. The interrupts are enabled or disabled at CP + 1; floating-point overflows occurring after that time cause interrupts if they are enabled even if the overflow is generated by a previously issued floating-point instruction.

Instructions 002300 and 002400 are issued even if there are other memory references in process resulting from previous issues. The interrupts are enabled or disabled at CP + 1; operand range errors occurring after that time cause interrupts if they are enabled even if the operand range error is generated by a previous memory reference.

INSTRUCTIONS 0030, 0034, 0036, and 0037

CAL Syntax	Description	Octal Code
VM <i>Sj</i>	Transmit (<i>Sj</i>) to Vector Mask register (VM)	0030 <i>j0</i>
VM 0†	Clear VM register	003000
SM <i>jk</i> 1,TS	Test and set semaphore <i>jk</i> , $0 \leq jk \leq 31_{10}$	0034 <i>jk</i>
SM <i>jk</i> 0	Clear semaphore <i>jk</i> , $0 \leq jk \leq 31_{10}$	0036 <i>jk</i>
SM <i>jk</i> 1	Set semaphore <i>jk</i> , $0 \leq jk \leq 31_{10}$	0037 <i>jk</i>

† Special CAL syntax

Instruction 0030*j0* enters the VM register with the *Sj* contents. The VM register is cleared if the *j* designator is 0 in instruction 003000. These instructions are used in conjunction with the vector merge instructions (146 and 147) in which an operation is performed depending on the contents of VM.

Instruction 0034*jk* tests and sets the semaphore designated by *jk*. If the semaphore is set, issue is held. If the semaphore is clear, the instruction issues and sets the semaphore. If the CPU is holding issue on a test and set, the DL flag is set in the Exchange Package (if not in monitor mode) and an exchange occurs. If an interrupt occurs while a test and set instruction is holding in the CIP register, the WS flag in the Exchange Package sets, CIP and NIP registers clear, and an exchange occurs with the P register pointing to the test and set instruction. The SM register is 32 bits with SM0 being the most significant bit.

Instruction 0036*jk* clears the semaphore designated by *jk*.

Instruction 0037*jk* sets the semaphore designated by *jk*.

HOLD ISSUE CONDITIONS: For instruction 0030*j0*:
Sj reserved (except S0)
 Instruction 003 in process, unit busy 1 CP
 Instruction 14X in process, unit busy
 (VL) + 5 CPs
 Instruction 175 in process, unit busy
 (VL) + 5 CPs

For instruction 0034*jk*:
 If current Cluster Number ≠ 0 and SM*jk* is set, holds issue.

INSTRUCTIONS 0030, 0034, 0036, and 0037 (continued)

EXECUTION TIME: Instruction issue, 1 CP

SPECIAL CASES: (Sj)=0 if j=0.

Instructions 0034jk, 0036jk, and 0037jk
are no-ops if CLN=0.

INSTRUCTION 004

CAL Syntax	Description	Octal Code
EX	Normal exit	004000

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 XA register contents. The program address stored into 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: Any A, S, or V register reserved

EXECUTION TIME: Instruction issue, 51 CPs; this time includes an exchange sequence (32 CPs) and a fetch operation (19 CPs).

SPECIAL CASES: None

INSTRUCTION 005

CAL Syntax	Description	Octal Code
J Bjk	Branch to (Bjk)	0050jk

Instruction 005 sets the P register to the 24-bit parcel address specified by the contents of Bjk causing execution to continue at that address. The instruction is used to return from a subroutine.

HOLD ISSUE CONDITIONS: Instruction 034 or 035 in process

Instruction 025 issued in the previous CP

Second parcel in a different buffer, 2-CP delay

Second parcel not in a buffer

EXECUTION TIME:

Instruction issue:

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

Instruction parcel and following parcel both in a buffer and branch address not in a buffer, 21 CPs. Additional time is needed if a memory conflict exists; the time to resolve a memory conflict depends on factors present.

SPECIAL CASES:

Instruction 0050jk executes as if it were a 2-parcel instruction. Even though the parcel following the first parcel of instruction 0050jk is not used, it can cause a delay of instruction 0050jk if it is out of buffer. Refer to execution times.

INSTRUCTION 006

CAL Syntax	Description	Octal Code
J exp	Branch to <i>ijkm</i>	006 <i>ijkm</i>

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: Second parcel in different buffer, 2-CP delay

Second parcel not in a buffer

EXECUTION TIME:

Instruction issue:

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

Both parcels of instruction in the same buffer and branch address not in a buffer, 19 CPs.

Additional time is needed if a memory conflict exists. The time to resolve a memory conflict depends on factors present.

SPECIAL CASES:

None

INSTRUCTION 007

CAL Syntax	Description	Octal Code
R <i>exp</i>	Return jump to <i>ijkm</i> ; set B00 to (P) + 2.	007 <i>ijkm</i>

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 through a return jump. The subroutine can return to the caller at the instruction following the call by executing a branch to the B00 register contents.

HOLD ISSUE CONDITIONS: Instruction 034 or 035 in process

Second parcel in a different buffer, 2-CP delay

Second parcel not in a buffer

EXECUTION TIME:

Instruction issue:

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

Both parcels of instruction in the same buffer and branch address not in a buffer, 19 CPs.

Additional time is needed if a memory conflict exists. The time to resolve a memory conflict depends on factors present.

SPECIAL CASES:

None

INSTRUCTIONS 010 - 013

CAL Syntax	Description	Octal Code
JAZ <i>exp</i>	Branch to <i>ijklm</i> if (A0)=0 ($i_2=0$)	010 <i>ijklm</i>
JAN <i>exp</i>	Branch to <i>ijklm</i> if (A0)≠0	011 <i>ijklm</i>
JAP <i>exp</i>	Branch to <i>ijklm</i> if (A0) positive, includes (A0)=0	012 <i>ijklm</i>
JAM <i>exp</i>	Branch to <i>ijklm</i> if (A0) negative	013 <i>ijklm</i>

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 *ijklm* field and execution continues at that address. The high-order bit of the *ijklm* field is ignored. If the condition is not satisfied, execution continues with the instruction following the branch instruction.

HOLD ISSUE CONDITIONS: A0 busy in any one of the previous 3 CPs

Second parcel in a different buffer, 2-CP delay

Second parcel not in a buffer

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.

Both parcels of instruction in the same buffer, branch taken, and branch address not in a buffer; 19 CPs. Additional time is needed if a memory conflict exists. The time to resolve a memory conflict is indeterminate.

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

Both parcels of instruction in different buffers, branch taken, and branch address not in a buffer; 21 CPs.

INSTRUCTIONS 010 - 013 (continued)

EXECUTION TIME:
(continued)

Second parcel of instruction not in a buffer,
branch taken, and branch address in a buffer;
21 CPs.

Second parcel of instruction not in a buffer,
branch taken, and branch address not in buffer;
35 CPs.

Instruction issue for branch not taken:

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

Both parcels of instruction in the same buffer,
branch not taken, and next instruction in
different instruction buffer; 4 CPs.

Both parcels of instruction in the same buffer
and branch not taken with next instruction in
memory; 19 CPs.

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

Second parcel of instruction not in a buffer
and branch not taken; 18 CPs.

SPECIAL CASES:

(A0)=0 is considered a positive condition.

INSTRUCTIONS 014 - 017

CAL Syntax	Description	Octal Code
JSZ <i>exp</i>	Branch to <i>ijklm</i> if (S0)=0 ($i_2=0$)	014 <i>ijklm</i>
JSN <i>exp</i>	Branch to <i>ijklm</i> if (S0)≠0	015 <i>ijklm</i>
JSP <i>exp</i>	Branch to <i>ijklm</i> if (S0) positive, includes (S0)=0	016 <i>ijklm</i>
JSM <i>exp</i>	Branch to <i>ijklm</i> if (S0) negative	017 <i>ijklm</i>

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 *ijklm* field and execution continues at that address. The high-order bit of the *ijklm* field is ignored. If the condition is not satisfied, execution continues with the instruction following the branch instruction.

HOLD ISSUE CONDITIONS: S0 busy in any one of the previous 3 CPs

Second parcel in a different buffer, 2-CP delay

Second parcel not in a buffer

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.

Both parcels of instruction in the same buffer, branch taken, and branch address not in a buffer; 19 CPs. Additional time is needed if a memory conflict exists. The time to resolve a memory conflict is indeterminate.

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

Both parcels of instruction in different buffers, branch taken, and branch address not in a buffer; 21 CPs.

INSTRUCTIONS 014 - 017 (continued)

EXECUTION TIME:
(continued)

Second parcel of instruction not in a buffer,
branch taken, and branch address in a buffer;
21 CPs.

Second parcel of instruction not in a buffer,
branch taken, and branch address not in buffer;
35 CPs.

Instruction issue for branch not taken:

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

Both parcels of instruction in the same buffer,
branch not taken, and next instruction in
different instruction buffer; 4 CPs.

Both parcels of instruction in the same buffer
and branch not taken with next instruction in
memory; 19 CPs.

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

Second parcel of instruction not in a buffer
and branch not taken; 18 CPs.

SPECIAL CASES:

(S0)=0 is considered a positive condition.

INSTRUCTION 01h

CAL Syntax	Description	Octal Code
Ah exp	Transmit <i>ijklm</i> to Ah ($i_2=1$)	01hijklm

The 2-parcel instruction 01h enters a 24-bit value into Ah that is composed of the low-order 24 bits of the *ijklm* field. The high-order bit of the *ijklm* field must be set to distinguish the 01h instruction from the 010 through 017 branches.

HOLD ISSUE CONDITIONS: Ah reserved

Second parcel not in a buffer

Second parcel in a different buffer

EXECUTION TIME:

Instruction issue:

Both parcels in same buffer, 2 CPs

Both parcels in different buffers, 4 CPs

Ah ready, 1 CP

SPECIAL CASES:

High-order bit of *i* designator (i_2) must be 1

INSTRUCTIONS 020 - 021

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

The 2-parcel instruction 020 enters a 24-bit value into *Ai* 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 *Ai*. 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 *Ai* are set to 1. The instruction provides a means of entering a negative value into *Ai*. If the instruction is used, however, 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: *Ai* reserved

Second parcel not in a buffer

EXECUTION TIME:

Instruction issue:

Both parcels in same buffer, 2 CPs

Both parcels in different buffers, 4 CPs

Ai ready, 1 CP

SPECIAL CASES:

None

INSTRUCTION 022

CAL Syntax	Description	Octal Code
<i>Ai exp</i>	Transmit <i>jk</i> to <i>Ai</i>	022ijk

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

HOLD ISSUE CONDITIONS: *Ai* reserved

EXECUTION TIME: Instruction issue, 1 CP

Ai ready, 1 CP

SPECIAL CASES: None

INSTRUCTION 023

CAL Syntax	Description	Octal Code
$A_i S_j$	Transmit (S_j) to A_i	023ij0
$A_i VL$	Read VL	023i01

Instruction 023ij0 enters the low-order 24 bits of (S_j) into A_i . The high-order bits of (S_j) are ignored.

Instruction 023i01 enters the VL register contents into A_i .

HOLD ISSUE CONDITIONS: A_i reserved

For instruction 023ij0, S_j reserved (except S_0)

EXECUTION TIME: Instruction issue, 1 CP

A_i ready, 1 CP

SPECIAL CASES: (S_j)=0 if $j=0$.

If (A_1)=0, the sequence:

VL A_1
 A2 VL
 leaves (A_2)=100_g

If (A_1)=23_g, the sequence:

VL A_1
 A2 VL
 leaves (A_2)=23_g

If (A_1)=123_g, the sequence:

VL A_1
 A2 VL
 leaves (A_2)=23_g

The 2^6 bit in the VL register is a 1 if the low-order 6 bits are 0; otherwise, the 2^6 bit is a 0.

INSTRUCTIONS 024 - 025

CAL Syntax	Description	Octal Code
<i>Ai Bjk</i>	Transmit (<i>Bjk</i>) to <i>Ai</i>	024 <i>ijk</i>
<i>Bjk Ai</i>	Transmit (<i>Ai</i>) to <i>Bjk</i>	025 <i>ijk</i>

Instruction 024 enters the contents of *Bjk* into *Ai*.

Instruction 025 enters the contents of *Ai* into *Bjk*.

HOLD ISSUE CONDITIONS: Instruction 034 or 035 in process

For instruction 024*ijk*, instruction 025*ijk*
issued in previous CP

Ai reserved

EXECUTION TIME: For instruction 024, *Ai* ready, 1 CP

Instruction issue, 1 CP

SPECIAL CASES: None

INSTRUCTION 026

CAL Syntax	Description	Octal Code
<i>Ai PSj</i>	Population count of (<i>Sj</i>) to <i>Ai</i>	026 <i>ij</i> 0
<i>Ai QSj</i>	Population count parity of (<i>Sj</i>) to <i>Ai</i>	026 <i>ij</i> 1
<i>Ai SBj</i>	Transfer (<i>SBj</i>) to <i>Ai</i>	026 <i>ij</i> 7

Instruction 026*ij*0 counts the number of bits set to 1 in (*Sj*) and enters the result into the low-order 7 bits of *Ai*. The high-order 17 bits of *Ai* are zeroed. If (*Sj*)=0, then (*Ai*)=0.

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

Instructions 026*ij*0 and 026*ij*1 are executed in the Population/Leading Zero Count functional unit.

Instruction 026*ij*7 transfers the *SBj* register contents to *Ai*.

HOLD ISSUE CONDITIONS: *Ai* reserved

Sj reserved (except *S0*)

Instruction 027*ij*7 or 073*ij*3 issued 3 CPs earlier.

EXECUTION TIME:

Instruction issue, 1 CP

For instructions 026*ij*0 and 026*ij*1, *Ai* ready 4 CPs

For instruction 026*ij*7, *Ai* ready 1 CP

SPECIAL CASES:

For instructions 026ij0 and 026ij1, $(A_i)=0$ if $j=0$.

For instruction 026ij7, $(A_i)=0$ if $CLN=0$.

If instruction 027ij7, write SB_j , has been issued within the previous 2 CPs, then the original value (instead of the new value) of (SB_j) is delivered to A_i as a result of this instruction.

INSTRUCTION 027

CAL Syntax	Description	Octal Code
A_i ZS_j	Leading zero count of (S_j) to A_i	027ij0
SB_j A_i	Transfer (A_i) to SB_j	027ij7

Instruction 027ij0 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. Instruction 027ij0 is executed in the Population/Leading Zero Count functional unit.

Instruction 027ij7 stores (A_i) to the SB_j register.

HOLD ISSUE CONDITIONS: For instruction 027ij0, instruction 033 issued in CP 2

A_i reserved

S_j reserved (except S_0)

EXECUTION TIME: Instruction issue, 1 CP

For instruction 027ij0, A_i ready, 3 CPs

For instruction 027ij7, SB_j ready, 3 CPs

SPECIAL CASES: For instruction 027ij0, $(A_i)=64$ if $j=0$.

For instruction 027ij0, $(A_i)=0$ if (S_j) is negative.

Instruction 027ij7 is a no-op if $CLN=0$.

INSTRUCTIONS 030 - 031

CAL Syntax	Description	Octal Code
$A_i \quad A_j + A_k$	Integer sum of (A_j) and (A_k) to A_i	030ijk
$A_i \quad A_k^\dagger$	Transmit (A_k) to A_i	030i0k
$A_i \quad A_j + 1^\dagger$	Integer sum of (A_j) and 1 to A_i	030ij0
$A_i \quad A_j - A_k$	Integer difference (A_j) less (A_k) to A_i	031ijk
$A_i \quad -1^\dagger$	Transmit -1 to A_i	031i00
$A_i \quad -A_k^\dagger$	Transmit the negative of (A_k) to A_i	031i0k
$A_i \quad A_j - 1^\dagger$	Integer difference (A_j) less 1 to A_i	031ij0

† Special CAL syntax

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.

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

HOLD ISSUE CONDITIONS: A_i reserved

A_j or A_k reserved (except A_0)

EXECUTION TIME: Instruction issue, 1 CP

A_i ready, 2 CPs

SPECIAL CASES:

For instruction 030:

$(A_i) = (A_k)$ if $j=0$ and $k \neq 0$.

$(A_i) = 1$ if $j=0$ and $k=0$.

$(A_i) = (A_j) + 1$ if $j \neq 0$ and $k=0$.

For instruction 031:

$(A_i) = -(A_k)$ if $j=0$ and $k \neq 0$.

$(A_i) = -1$ if $j=0$ and $k=0$.

$(A_i) = (A_j) - 1$ if $j \neq 0$ and $k=0$.

INSTRUCTION 032

CAL Syntax	Description	Octal Code
$A_i \quad A_j * A_k$	Integer product of (A_j) and (A_k) to A_i	032ijk

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.

Instruction 032 is executed in the Address Multiply functional unit.

HOLD ISSUE CONDITIONS: A_i reserved

A_j or A_k reserved (except A0)

EXECUTION TIME: Instruction issue, 1 CP

A_i ready, 4 CPs

SPECIAL CASES:

$(A_i) = 0$ if $j = 0$.

$(A_k) = 1$ if $k = 0$.

Thus, $(A_i) = (A_j)$ if $j \neq 0$ and $k = 0$.

INSTRUCTION 033

CAL Syntax	Description	Octal Code
A_i CI	Channel number of highest priority interrupt request to A_i	033i00
A_i CA, A_j	Current address of channel (A_j) to A_i	033ij0
A_i CE, A_j	Error flag of channel (A_j) to A_i	033ij1

Instruction 033 enters channel status information into A_i . The j and k designators and the 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. The high-order bits of A_i are cleared. The error flag can be cleared only in monitor mode using instruction 0012.

Instruction 033 does not interfere with channel operation and is not protected from user execution.

HOLD ISSUE CONDITIONS: 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) \neq 0 and k =0.

(A_i)=I/O error flag of channel (A_j) if (A_j) \neq 0 and k =1.

(A_i)=0 if (A_j)=1.

INSTRUCTION 033 (continued)

SPECIAL CASES:
(continued)

2 CPs must elapse after instruction 0012j0 issues before issuing instruction 033i00.

If instruction 033 issues every 10 CPs (in a loop), the same results may be returned to Ai.

When $k=1$:

Bits 2^{12} through 2^{20} contain the remaining block length.

Bit 2^{18} indicates a request in progress.

Bit 2^{19} indicates either an SSD single-bit memory error (during a read SSD operation) or an SSD single-bit channel error (during a write SSD operation).

Bit 2^{20} indicates a block length error.

Bit 2^{21} indicates either an SSD double-bit memory error (during a read SSD operation) or an SSD double-bit channel error (during a write SSD operation).

Bit 2^{22} indicates a CPU double-bit memory error.

Bit 2^{23} indicates a fatal error (if bit 2^{20} , 2^{21} , or 2^{22} is set).

INSTRUCTIONS 034 - 037

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

† Special CAL syntax

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 (Ai). Refer to special cases for details.

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 (Ai) is not exhausted.

INSTRUCTIONS 034 - 037 (continued)

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. 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 reserved

Ai reserved

Scalar reference in CP1, CP2, CP3, or CP4

For instruction 034, Port A busy or instruction 035 in process or unidirectional memory mode and Port C busy

For instruction 035, Port C busy or instruction 034 in process or unidirectional memory mode and Port A or Port B busy

For instruction 036, Port B busy or instruction 037 in process or unidirectional memory mode and Port C busy

For instruction 037, Port C busy or instruction 036 in process or unidirectional memory mode and Port A or Port B busy

EXECUTION TIME: Instruction issue, 1 CP

For instruction 034 or 036:

B or T register reserved 19 CPs + (Ai) if (Ai)≠0; 6 CPs if (Ai)=0.

Port A or B busy for (Ai) + 5 CPs if (Ai)≠0; 4 CPs if (Ai)=0.

For instruction 035 or 037:

B or T register reserved 5 CPs + (Ai) if (Ai)≠0; 4 CPs if (Ai)=0.

Port C busy for (Ai) + 5 CPs if (Ai)≠0; 4 CPs if (Ai)=0.

INSTRUCTIONS 034 - 037 (continued)

SPECIAL CASES:

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

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

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

NOTE

Instruction 034 uses Port A, instruction 035 uses Port C, instruction 036 uses Port B, and instruction 037 uses Port C.

INSTRUCTIONS 040 - 041

CAL Syntax	Description	Octal Code
<i>Si exp</i>	Transmit <i>jkm</i> to <i>Si</i>	040 <i>ijkm</i>
<i>Si exp</i>	Transmit complement of <i>jkm</i> to <i>Si</i>	041 <i>ijkm</i>

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 *Si*.

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 *Si*. 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 *Si* are set to 1's. The instruction provides for entering a negative value into *Si*. 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, and so on.

HOLD ISSUE CONDITIONS: *Si* reserved

Second parcel not in a buffer

EXECUTION TIME:

Instruction issue:

Both parcels in same buffer, 2 CPs

Both parcels in different buffers, 4 CPs

Si ready, 1 CP

SPECIAL CASES:

None

INSTRUCTIONS 042 - 043

CAL Syntax	Description	Octal Code
$Si \text{ } \langle exp$	Form exp bits of ones mask in Si from right; jk field gets $64 - exp$.	$042ijk$
$Si \text{ } \# \rangle exp \dagger$	Form exp bits of zeros mask in Si from left; jk field gets exp .	$042ijk$
$Si \text{ } 1 \dagger$	Enter 1 into Si	$042i77$
$Si \text{ } -1 \dagger$	Enter -1 into Si	$042i00$
$Si \text{ } \rangle exp$	Form exp bits of ones mask in Si from left; jk field gets exp .	$043ijk$
$Si \text{ } \# \langle exp \dagger$	Form exp bits of zeros mask in Si from right; jk field gets $64 - exp$.	$043ijk$
$Si \text{ } 0 \dagger$	Clear Si	$043i00$

† Special CAL syntax

Instruction 042 generates a mask of $64 - jk$ ones from right to left in Si . For example, if $jk=0$, Si contains all 1 bits (integer value=-1) and if $jk=77_8$, Si 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 Si . For example, if $jk=0$, Si contains all 0 bits (integer value=0) and if $jk=77_8$, Si contains ones in all but the low-order bit (integer value=-2).

The Scalar Logical functional unit executes instructions 042 and 043.

HOLD ISSUE CONDITIONS: Si reserved

EXECUTION TIME: Instruction issue, 1 CP

Si ready, 1 CP

SPECIAL CASES: None

INSTRUCTIONS 044 - 051

CAL Syntax	Description	Octal Code
$S_i S_j \& S_k$	Logical product of (S_j) and (S_k) to S_i	044ijk
$S_i S_j \& SB^\dagger$	Sign bit of (S_j) to S_i	044ij0
$S_i SB \& S_j^\dagger$	Sign bit of (S_j) to S_i ($j \neq 0$)	044ij0
$S_i \# S_k \& S_j$	Logical product of (S_j) and complement of (S_k) to S_i	045ijk
$S_i \# SB \& S_j^\dagger$	(S_j) with sign bit cleared to S_i	045ij0
$S_i S_j \setminus S_k$	Logical difference of (S_j) and (S_k) to S_i	046ijk
$S_i S_j \setminus SB^\dagger$	Toggle sign bit of (S_j) , then enter into S_i	046ij0
$S_i SB \setminus S_j^\dagger$	Toggle sign bit of (S_j) , then enter into S_i ($j \neq 0$)	046ij0
$S_i \# S_j \setminus S_k$	Logical equivalence of (S_k) and (S_j) to S_i	047ijk
$S_i \# S_k^\dagger$	Transmit ones complement of (S_k) to S_i	047i0k
$S_i \# S_j \setminus SB^\dagger$	Logical equivalence of (S_j) and sign bit to S_i	047ij0
$S_i \# SB \setminus S_j^\dagger$	Logical equivalence of (S_j) and sign bit to S_i ($j \neq 0$)	047ij0
$S_i \# SB^\dagger$	Enter ones complement of sign bit into S_i	047i00
$S_i S_j ! S_i \& S_k$	Logical product of (S_i) and (S_k) complement ORed with logical product of (S_j) and (S_k) to S_i	050ijk
$S_i S_j ! S_i \& SB^\dagger$	Scalar merge of (S_i) and sign bit of (S_j) to S_i	050ij0
$S_i S_j ! S_k$	Logical sum of (S_j) and (S_k) to S_i	051ijk
$S_i S_k^\dagger$	Transmit (S_k) to S_i	051i0k
$S_i S_j ! SB^\dagger$	Logical sum of (S_j) and sign bit to S_i	051ij0
$S_i SB ! S_j^\dagger$	Logical sum of (S_j) and sign bit to S_i ($j \neq 0$)	051ij0
$S_i SB^\dagger$	Enter sign bit into S_i	051i00

† Special CAL syntax

INSTRUCTIONS 044 - 051 (continued)

NOTE

For instructions 044 through 051, SB with no register designator is the sign bit, not Shared Address register.

The Scalar Logical functional unit executes instructions 044 through 051.

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:

$$\begin{array}{r} (S_j) = 1\ 1\ 0\ 0 \\ (S_k) = \underline{1\ 0\ 1\ 0} \\ (S_i) = 1\ 0\ 0\ 0 \end{array}$$

(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) :

$$\begin{array}{r} \text{if } (S_k) = 1\ 0\ 1\ 0 \\ \\ (S_j) = 1\ 1\ 0\ 0 \\ (S_k') = \underline{0\ 1\ 0\ 1} \\ (S_i) = 0\ 1\ 0\ 0 \end{array}$$

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:

$$\begin{array}{r} (S_j) = 1\ 1\ 0\ 0 \\ (S_k) = \underline{1\ 0\ 1\ 0} \\ (S_i) = 0\ 1\ 1\ 0 \end{array}$$

INSTRUCTIONS 044 - 051 (continued)

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:

$$\begin{array}{r} (S_j) = 1\ 1\ 0\ 0 \\ (S_k) = \underline{1\ 0\ 1\ 0} \\ (S_i) = 1\ 0\ 0\ 1 \end{array}$$

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:

$$(S_i) = (S_j)(S_k) + (S_i)(S_k')$$

if $(S_k) = 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0$

$$\begin{array}{r} (S_k') = 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \\ (S_i) = 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0 \\ (S_j) = \underline{1\ 0\ 1\ 0\ 1\ 0\ 1\ 0} \\ (S_i) = 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0 \end{array}$$

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:

$$\begin{array}{r} (S_j) = 1\ 1\ 0\ 0 \\ (S_k) = \underline{1\ 0\ 1\ 0} \\ (S_i) = 1\ 1\ 1\ 0 \end{array}$$

INSTRUCTIONS 044 - 051 (continued)

(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: S_i reserved

S_j or S_k reserved (except S_0)

EXECUTION TIME: Instruction issue, 1 CP

S_i ready, 1 CP

SPECIAL CASES: (S_j)=0 if $j=0$.

(S_k)= 2^{63} if $k=0$.

INSTRUCTIONS 052 - 055

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

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

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

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

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

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

HOLD ISSUE CONDITIONS: Instruction 056, 057, 060, or 061 issued in previous CP

Si reserved

For instructions 052 and 053, S0 reserved

EXECUTION TIME: Instruction issue, 1 CP

For instructions 052 and 053, S0 ready, 2 CPs

For instructions 054 and 055, Si ready, 2 CPs

SPECIAL CASES: None

INSTRUCTIONS 056 - 057

CAL Syntax	Description	Octal Code
$S_i S_j \langle A_k$	Shift (S_i) and (S_j) left by (A_k) places to S_i	056ijk
$S_i S_j \langle 1^\dagger$	Shift (S_i) and (S_j) left one place to S_i	056ij0
$S_i S_i \langle A_k^\dagger$	Shift (S_i) left (A_k) places to S_i	056i0k
$S_i S_j, S_i \rangle A_k$	Shift (S_j) and (S_i) right by (A_k) places to S_i	057ijk
$S_i S_j, S_i \rangle 1^\dagger$	Shift (S_j) and (S_i) right one place to S_i	057ij0
$S_i S_i \rangle A_k^\dagger$	Shift (S_i) right (A_k) places to S_i	057i0k

† Special CAL syntax

The Scalar Shift functional unit executes instructions 056 and 057. 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 and 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 were 64 bits rather than 128 bits because the S_j value used is 0.

The shifts are 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 $i \neq j$. 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 of 64 or 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.

INSTRUCTIONS 056 - 057 (continued)

HOLD ISSUE CONDITIONS: S_i reserved
 S_j or A_k reserved (except S_0 and/or A_0)

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 greater than or equal to 0 and less than or equal to 64.

INSTRUCTIONS 060 - 061

CAL Syntax	Description	Octal Code
$S_i \quad S_j+S_k$	Integer sum of (S_j) and (S_k) to S_i	060ijk
$S_i \quad S_j-S_k$	Integer difference of (S_j) and (S_k) to S_i	061ijk
$S_i \quad -S_k^\dagger$	Transmit negative of (S_k) to S_i	061i0k

† Special CAL syntax

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.

The Scalar Add functional unit executes instructions 060 and 061.

HOLD ISSUE CONDITIONS: S_i reserved

S_j or S_k reserved (except S_0)

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 $k \neq 0$.

$(S_i)=(S_j)$ with 2^{63} complemented if $j \neq 0$ and $k=0$.

For instruction 061:

$(S_i) = -(S_k)$ if $j=0$ and $k \neq 0$.

$(S_i)=(S_j)$ with 2^{63} complemented if $j \neq 0$ and $k=0$.

INSTRUCTIONS 062 - 063

CAL Syntax	Description	Octal Code
$S_i \quad S_j + F S_k$	Floating-point sum of (S_j) and (S_k) to S_i	062ijk
$S_i \quad + F S_k^\dagger$	Normalize (S_k) to S_i	062i0k
$S_i \quad S_j - F S_k$	Floating-point difference of (S_j) and (S_k) to S_i	063ijk
$S_i \quad - F S_k^\dagger$	Transmit normalized negative of (S_k) to S_i	063i0k

The Floating-point Add functional unit executes instructions 062 and 063. 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 .

Section 4 describes overflow conditions. For floating-point operands with the sign bit set (bit=1), zero exponent and zero coefficient are treated as 0 (that is, all 64 bits=0).^{††}

HOLD ISSUE CONDITIONS: S_i reserved

S_j or S_k reserved (except S_0)

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

^{††} 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.

INSTRUCTIONS 062 - 063 (continued)

SPECIAL CASES:

For instruction 062:

$(S_i) = (S_k)$ normalized if (S_k) exponent is valid, $j=0$ and $k \neq 0$.

$(S_i) = (S_j)$ normalized if (S_j) exponent is valid, $j \neq 0$ and $k=0$.

For instruction 063:

$(S_i) = -(S_k)$ normalized if (S_k) exponent is valid, $j=0$ and $k \neq 0$. Sign of (S_i) is opposite that of (S_k) if $(S_k) \neq 0$.

$(S_i) = (S_j)$ normalized if (S_j) exponent is valid, $j \neq 0$ and $k=0$.

INSTRUCTIONS 064 - 067

CAL Syntax	Description	Octal Code
S_i $S_j * F S_k$	Floating-point product of (S_j) and (S_k) to S_i	064ijk
S_i $S_j * H S_k$	Half-precision rounded floating-point product of (S_j) and (S_k) to S_i	065ijk
S_i $S_j * R S_k$	Rounded floating-point product of (S_j) and (S_k) to S_i	066ijk
S_i $S_j * I S_k$	Reciprocal iteration; $2 - (S_j) * (S_k)$ to S_i	067ijk

The Floating-point Multiply functional unit executes instructions 064 through 067. 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 4 under Floating-point Arithmetic.

In the evaluation $C = 2 - B * A$, B must be a reciprocal of A of less than 47 significant bits 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: S_i reserved

S_j or S_k reserved (except S_0)

Instructions 160 through 167 in process, unit busy (VL) + 4 CPs

INSTRUCTIONS 064 - 067 (continued)

HODL ISSUE CONDITIONS: For mainframes with a Second Vector Logical
(continued) unit: instructions 140 through 145 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	070ij0

The Reciprocal Approximation functional unit executes instruction 070.

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 4 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: 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 2⁴⁷ 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	071i0k
$S_i \ +A_k$	Transmit (A_k) to S_i with sign extension	071i1k
$S_i \ +FA_k$	Transmit (A_k) to S_i as unnormalized floating-point number	071i2k
$S_i \ 0.6$	Transmit constant 0.75×2^{48} to S_i	071i30
$S_i \ 0.4$	Transmit constant 0.5 to S_i	071i40
$S_i \ 1.$	Transmit constant 1.0 to S_i	071i50
$S_i \ 2.$	Transmit constant 2.0 to S_i	071i60
$S_i \ 4.$	Transmit constant 4.0 to S_i	071i70

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 an integer whose absolute value is less than 24 bits to floating-point format:

```

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

```

INSTRUCTION 071 (continued)

When the j designator is 3, the floating-point constant of 0.75×2^{48} is entered into S_i (0 40060 6000 0000 0000 0000₈). 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 S_1 :

```
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 0000₈) is entered into S_i .

When the j designator is 5, the floating-point constant 1.0 (= 0 40001 4000 0000 0000 0000₈) is entered into S_i .

When the j designator is 6, the floating-point constant 2.0 (= 0 40002 4000 0000 0000 0000₈) is entered into S_i .

When the j designator is 7, the floating-point constant 4.0 (= 0 40003 4000 0000 0000 0000₈) is entered into S_i .

HOLD ISSUE CONDITIONS: S_i reserved

A_k reserved (except A_0); 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 $j=2$.

$(S_i)=0.6 \times 2^{60}$ (octal) if $j=3$.

$(S_i)=0.4 \times 2^0$ (octal) if $j=4$.

$(S_i)=0.4 \times 2^1$ (octal) if $j=5$.

$(S_i)=0.4 \times 2^2$ (octal) if $j=6$.

$(S_i)=0.4 \times 2^3$ (octal) if $j=7$.

INSTRUCTIONS 072 - 075

CAL Syntax	Description	Octal Code
<i>Si</i> RT	Transmit (RTC) to <i>Si</i>	072i00
<i>Si</i> SM	Read semaphores to <i>Si</i>	072i02
<i>Si</i> STj	Read (STj) register to <i>Si</i>	072ij3
<i>Si</i> VM	Transmit (VM) to <i>Si</i>	073i00
<i>Si</i> SRO	Transmit (SRO) to <i>Si</i>	073i01
†	Read performance counter into <i>Si</i>	073i11
†	Increment performance counter	073i21
†	Clear all maintenance modes	073i31
SM <i>Si</i>	Load semaphores from <i>Si</i>	073i02
STj <i>Si</i>	Load (STj) register from <i>Si</i>	073ij3
<i>Si</i> Tjk	Transmit (Tjk) to <i>Si</i>	074ijk
Tjk <i>Si</i>	Transmit (<i>Si</i>) to Tjk	075ijk

† Not currently supported

Instruction 072i00 enters the 64-bit value of the real-time clock (RTC) into *Si*. The clock is incremented by 1 each CP. The RTC can be set only by the monitor through use of instruction 0014j0.

Instruction 072i02 enters the values of all of the semaphores into *Si*. The 32-bit SM register is left-justified in *Si* with SM00 occupying the sign bit.

Instruction 072ij3 enters the contents of STj into *Si*.

Instruction 073i00 enters the 64-bit value of the VM register into *Si*. The VM register is usually read after being set by instruction 175.

Instruction 073i11 is used for performance monitoring and is privileged to monitor mode. Each execution of the 073i11 instruction advances a pointer and enters either the high-order or low-order bits of a performance counter into the high-order bits of *Si*. Refer to appendix C for information on performance monitoring.

INSTRUCTIONS 072 - 075 (continued)

Instruction 073i31 is part of the SECDED maintenance mode functions and is executed only if the maintenance mode switch on the mainframe's control panel is on. Instruction 073i31 clears all three SECDED maintenance mode instructions: 001501, 001521, and 001531. Refer to appendix D for complete information on the SECDED maintenance modes.

Instruction 073i01 sets the low-order 32 bits to 1's and returns the following status to the high-order bits of *Si*:

<u>Si Bit</u>	<u>Description</u>
263	Clustered, CLN≠0 (CL)
257	Program state (PS)
251	Floating-point error occurred (FPS)
250	Floating-point interrupt enabled (IFP)
249	Operand range interrupt enabled (IOR)
248	Bidirectional memory enabled (BDM)
240†	Processor number (PN) (This bit is always 0.)
233†	Cluster number bit 1 (CLN1)
232†	Cluster number bit 0 (CLN0)

Instruction 073i02 sets the semaphores from 32 high-order bits of *Si*. SM00 receives the sign bit of *Si*.

Instruction 073ij3 enters the contents of *Si* into *STj*.

Instruction 074 enters the contents of *Tjk* into *Si*.

Instruction 075 enters the contents of *Si* into *Tjk*.

HOLD ISSUE CONDITIONS: *Si* reserved

For instructions 074 and 075, instructions 036 through 037 in process

For instruction 074, instruction 075 issued in the previous CP

For instruction 073i00:
 Instruction 14x or 175 in process, VM busy for (VL) + 5 CPs
 Instruction 003 in process, VM busy for 1 CP

† These bit positions return a value of 0 if not executed in monitor mode.

INSTRUCTIONS 072 - 075 (continued)

HOLD ISSUE CONDITIONS: Instruction 073*ij*3 or 027*ij*7 issued by the
(continued) CPU 3 CPs earlier.

EXECUTION TIME: Instruction issue, 1 CP

All cases except 073*ij*3, result register ready,
1 CP

For 073*i*02, SM ready, 1 CP

SPECIAL CASES: For instructions 072*i*02 and 072*ij*3, (Si)=0
if CLN=0.

Instructions 073*i*02 and 073*ij*3 are no-ops if
CLN=0.

There must be a 2 CP delay between sequential
073*i*11 instructions.

For instruction 072*ij*3:
If an 073*ij*3 instruction has been issued
within the previous 2 CPs, then the original
value (instead of the new value) of (ST*j*) is
delivered to Si as a result of this
instruction.

INSTRUCTIONS 076 - 077

CAL Syntax	Description	Octal Code
$S_i V_j, A_k$	Transmit (V_j element (A_k)) to S_i	076ijk
$V_i, A_k S_j$	Transmit (S_j) to V_i element (A_k)	077ijk
$V_i, A_k 0^\dagger$	Clear V_i element (A_k)	077i0k

† Special CAL syntax

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: A_k reserved (except A0)

For instruction 076, S_i reserved or V_j reserved as operand or as result

For instruction 077, V_i reserved as operand or as result or S_j reserved

EXECUTION TIME: Instruction issue, 1 CP

For instruction 076, S_i ready, 4 CPs

For instruction 077, V_i ready, 1 CP

SPECIAL CASES: (S_j)=0 if $j=0$.

(A_k)=1 if $k=0$.

INSTRUCTIONS 10h - 13h

CAL Syntax	Description	Octal Code
<i>Ai exp,Ah</i>	Read from ((<i>Ah</i>) + <i>jkm</i>) to <i>Ai</i>	10 <i>hijkm</i>
<i>Ai exp,0†</i>	Read from (<i>jkm</i>) to <i>Ai</i>	100 <i>ijkm</i>
<i>Ai exp,†</i>	Read from (<i>jkm</i>) to <i>Ai</i>	100 <i>ijkm</i>
<i>Ai ,Ah†</i>	Read from (<i>Ah</i>) to <i>Ai</i>	10 <i>hi000</i>
<i>exp,Ah Ai</i>	Store (<i>Ai</i>) to (<i>Ah</i>) + <i>jkm</i>	11 <i>hijkm</i>
<i>exp,0 Ai†</i>	Store (<i>Ai</i>) to <i>jkm</i>	110 <i>ijkm</i>
<i>exp, Ai†</i>	Store (<i>Ai</i>) to <i>exp</i>	110 <i>ijkm</i>
<i>,Ah Ai†</i>	Store (<i>Ai</i>) to (<i>Ah</i>)	11 <i>hi000</i>
<i>Si exp,Ah</i>	Read from ((<i>Ah</i>) + <i>jkm</i>) to <i>Si</i>	12 <i>hijkm</i>
<i>Si exp,0†</i>	Read from (<i>exp</i>) to <i>Si</i>	120 <i>ijkm</i>
<i>Si exp,†</i>	Read from (<i>exp</i>) to <i>Si</i>	120 <i>ijkm</i>
<i>Si ,Ah†</i>	Read from (<i>Ah</i>) to <i>Si</i>	12 <i>hi000</i>
<i>exp,Ah Si</i>	Store (<i>Si</i>) to (<i>Ah</i>) + <i>jkm</i>	13 <i>hijkm</i>
<i>exp,0 Si†</i>	Store (<i>Si</i>) to <i>exp</i>	130 <i>ijkm</i>
<i>exp, Si†</i>	Store (<i>Si</i>) to <i>exp</i>	130 <i>ijkm</i>
<i>,Ah Si†</i>	Store (<i>Si</i>) to (<i>Ah</i>)	13 <i>hi000</i>

† Special CAL syntax

The 2-parcel instructions 10h through 13h transmit data between memory and an A register or an S register. The content of *Ah* (treated as a 22-bit signed integer) is added to the signed 22-bit integer in the *jkm* field to determine the memory address. If *h* is 0, (*Ah*) is 0 and only the *jkm* field is used for the address. The address arithmetic is performed by an address adder similar to but separate from the Address Add functional unit.

INSTRUCTIONS 10h - 13h (continued)

Instructions 10h and 11h transmit 24-bit quantities to or from A registers. When transmitting data from memory to an A register, the high-order 40 bits of the memory word are ignored. On a store from Ai into memory, the high-order 40 bits of the memory word are zeroed.

Instructions 12h and 13h transmit 64-bit quantities to or from register Si.

HOLD ISSUE CONDITIONS: Port A, B, or C busy

Ah reserved or busy previous CP

For instructions 10h and 11h, Ai reserved

For instructions 12h and 13h, Si reserved

Instructions 10x through 13x in CP 2 and CP 3 and conflict

Second parcel not in a buffer

Second parcel in different buffer, 2 CP

EXECUTION TIME:

Instruction issue:

Both parcels in same buffer, 2 CPs

For instruction 10h, Ai ready, 17 CPs

For instruction 12h, Si ready, 17 CPs

Bank ready for next scalar read or store, 8 CPs

NOTE

After issuing instructions 10h through 13h, attempting to issue instructions 034 through 037, 176, or 177 causes Ports A, B, or C to be considered busy until referenced bank is available.

SPECIAL CASES:

None

INSTRUCTIONS 140 - 147

CAL Syntax	Description	Octal Code
<i>Vi Sj&Vk</i>	Logical products of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	140ijk
<i>Vi Vj&Vk</i>	Logical products of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	141ijk
<i>Vi Sj!Vk</i>	Logical sums of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	142ijk
<i>vi vk†</i>	Transmit (<i>Vk</i> elements) to <i>Vi</i> elements	142i0k
<i>Vi Vj!Vk</i>	Logical sums of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	143ijk
<i>Vi Sj Vk</i>	Logical differences of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	144ijk
<i>Vi Vj Vk</i>	Logical differences of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	145ijk
<i>vi 0†</i>	Clear <i>Vi</i> elements	145iii
<i>Vi Sj!Vk&VM</i>	If VM bit=1, transmit (<i>Sj</i>) to the corresponding element in <i>Vi</i> . If VM bit=0, transmit the (corresponding <i>Vk</i> element) to the (corresponding <i>Vi</i> element).	146ijk
<i>Vi #VM&Vk†</i>	If VM bit=1, transmit (0) to the corresponding element in <i>Vi</i> . If VM bit=0, transmit the (corresponding <i>Vk</i> element) to the (corresponding <i>Vi</i> element).	146i0k
<i>Vi Vj!Vk&VM</i>	If VM bit=1, transmit the (corresponding <i>Vj</i> element) to the (corresponding <i>Vi</i> element). If VM bit=0, transmit the (corresponding <i>Vk</i> element) to the (corresponding <i>Vi</i> element).	147ijk

† Special CAL syntax

On mainframes equipped with Second Vector Logical functional units, instructions 140 through 145 can be executed in either the Full Vector or the Second Vector Logical units, provided the Second Vector Logical unit is enabled. If the Second Vector Logical unit is disabled, instructions 140 through 145 can be executed only in the Full Vector Logical unit.

INSTRUCTIONS 140 - 147 (continued)

Instructions 146 and 147 execute in the Full Vector Logical unit only. The number of operations performed is determined by the VL register contents. 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 .

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:

$$\begin{aligned} (S_j) \text{ or } (V_j \text{ element}) &= 1\ 1\ 0\ 0 \\ (V_k \text{ element}) &= \underline{1\ 0\ 1\ 0} \\ (V_i \text{ element}) &= 1\ 0\ 0\ 0 \end{aligned}$$

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:

$$\begin{aligned} (S_j) \text{ or } (V_j \text{ element}) &= 1\ 1\ 0\ 0 \\ (V_k \text{ element}) &= \underline{1\ 0\ 1\ 0} \\ (V_i \text{ element}) &= 1\ 1\ 1\ 0 \end{aligned}$$

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:

$$\begin{aligned} (S_j) \text{ or } (V_j \text{ element}) &= 1\ 1\ 0\ 0 \\ (V_k \text{ element}) &= \underline{1\ 0\ 1\ 0} \\ (V_i \text{ element}) &= 0\ 1\ 1\ 0 \end{aligned}$$

Instructions 146 and 147 transmit operands to V_i depending on the VM register contents. 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.

INSTRUCTIONS 140 - 147 (continued)

Example 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.

Example 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.

INSTRUCTIONS 140 - 147 (continued)

HOLD ISSUE CONDITIONS: For instructions 141, 143, 145, and 147, V_j reserved as operand

For instructions 146 and 147, or instructions 140 through 145 with Second Vector Logical disabled:[†]

Instruction 14x or 175 in process, Full Vector Logical unit busy (VL) + 4 CPs

For instructions 140 through 145 with Second Vector Logical unit enabled:[†]

Refer to discussion on Second Vector Logical issue in section 4

Instructions 140 through 145 or 16x in process in Second Vector Logical[†]/Floating-point Multiply unit, Second Vector Logical unit busy (VL) + 4 CPs

Instruction 140 through 147 or 175 in process in Full Vector Logical unit, Full Vector Logical unit busy (VL) + 4 CPs

EXECUTION TIME: Instruction issue, 1 CP

V_j or V_k ready in (VL) + 3 CPs if data available^{††}

V_i ready in (VL) + 7 CPs if data available^{††}

Unit ready, (VL) + 4 CPs if data available^{††}

SPECIAL CASES: (S_j)=0 if $j=0$.

[†] Second Vector Logical unit is not available on all machines.

^{††} Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 150 - 151

CAL Syntax	Description	Octal Code
$V_i \ V_j < A_k$	Shift (V_j) elements left by (A_k) places to V_i elements	150ijk
$V_i \ V_j < 1^\dagger$	Shift (V_j) elements left one place to V_i elements	150ij0
$V_i \ V_j > A_k$	Shift (V_j) elements right by (A_k) places to V_i elements	151ijk
$V_i \ V_j > 1^\dagger$	Shift (V_j) elements right one place to V_i elements	151ij0

Instructions 150 and 151 are executed in the Vector Shift functional unit. The number of operations performed is determined by the VL register contents. 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 all 24 bits of A_k are used for the shift count. Elements of V_i are cleared if the shift count exceeds 63. All shift counts (A_k) are considered positive.

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

HOLD ISSUE CONDITIONS: V_j reserved as operand

V_i reserved as operand or result

A_k reserved (except A0)

Instructions 150 through 153 in process, unit busy $(VL) + 4$ CPs^{††}

[†] Special CAL syntax

^{††} Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 150 - 151 (continued)

EXECUTION TIME: V_j ready in (VL) + 3 CPs if data available[†]
 V_i ready in (VL) + 8 CPs if data available[†]
 Unit ready, (VL) + 4 CPs if data available[†]

SPECIAL CASES: $(Ak)=1$ if $k=0$.

[†] Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 152 - 153

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	152ijk
$V_i V_j, V_j < 1^\dagger$	Double shifts of (V_j elements) left one place to V_i elements	152ij0
$V_i V_j, V_j > A_k$	Double shifts of (V_j elements) right (A_k) places to V_i elements	153ijk
$V_i V_j, V_j > 1^\dagger$	Double shifts of (V_j elements) right one place to V_i elements	153ij0

\dagger Special CAL syntax

The Vector Shift functional unit executes instructions 152 and 153. 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 VL register contents.

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 5-7 shows this operation.

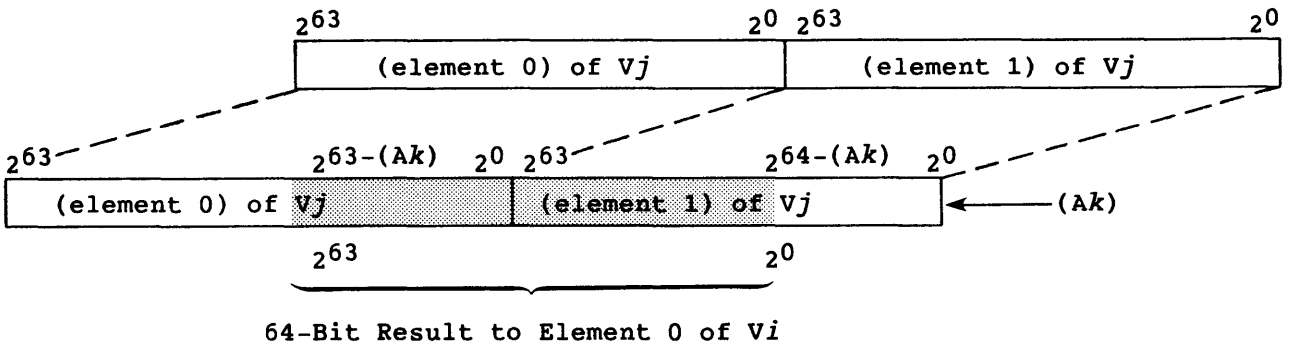


Figure 5-7. Vector Left Double Shift, First Element, VL Greater than 1

If (VL) is greater than 2, the operation continues by joining element 1 with element 2 and transmitting the 64-bit result to element 1 of Vi. Figure 5-8 shows this operation.

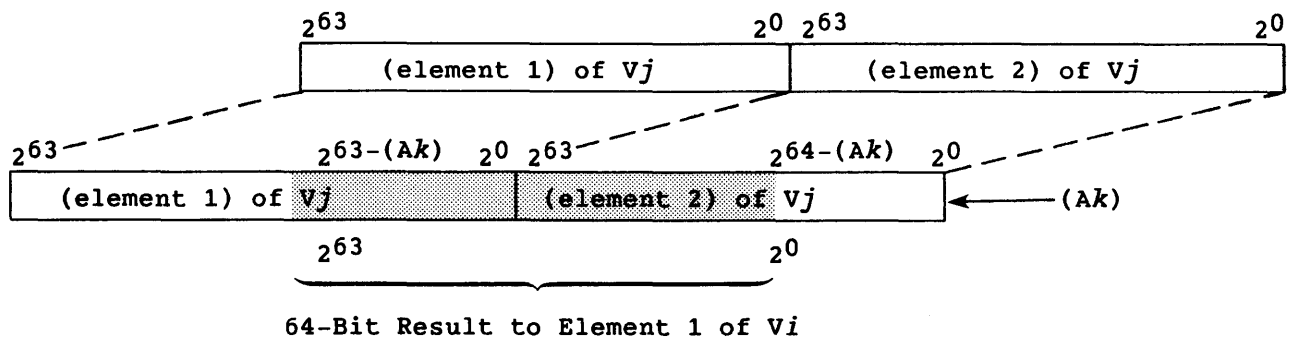


Figure 5-8. Vector Left Double Shift, Second Element, VL Greater than 2

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 Vj as determined by (VL) is joined with 64 bits of zeros. Figure 5-9 shows this operation.

INSTRUCTIONS 152 - 153 (continued)

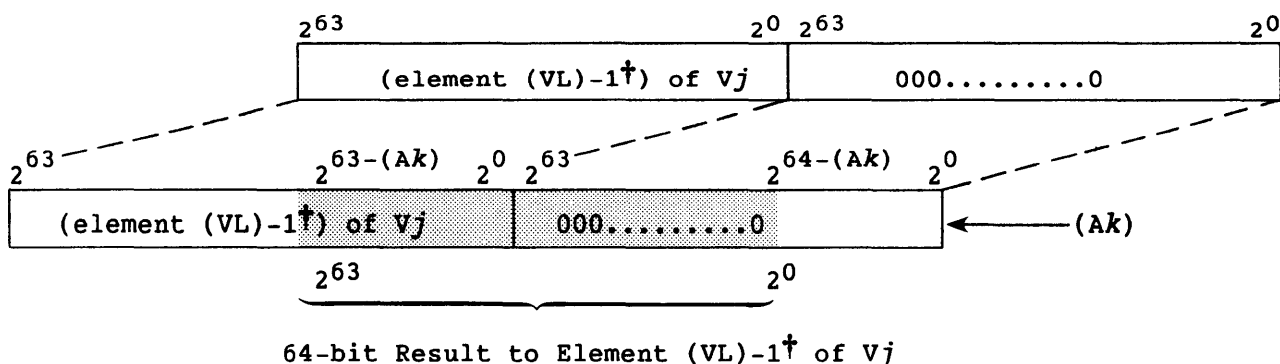


Figure 5-9. Vector Left Double Shift, Last Element

If (Ak) is greater than or equal to 128, the result is all zeros. If (Ak) is greater than 64, the result register contains at least $(Ak) - 64$ zeros.

Example 1:

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. 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
```

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

INSTRUCTIONS 152 - 153 (continued)

Instruction 153 performs right shifts. The original 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 (Ak) . The 64 low-order bits of the result are transmitted to element 0 of V_i . Figure 5-10 shows this operation.

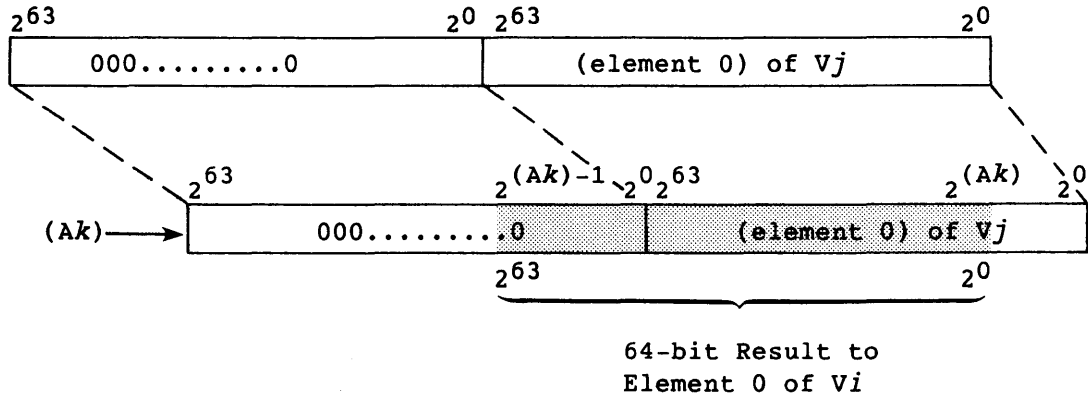


Figure 5-10. Vector Right Double Shift, First Element

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 (Ak) , and transmitting the result to element 1 of V_i . Figure 5-11 shows this operation.

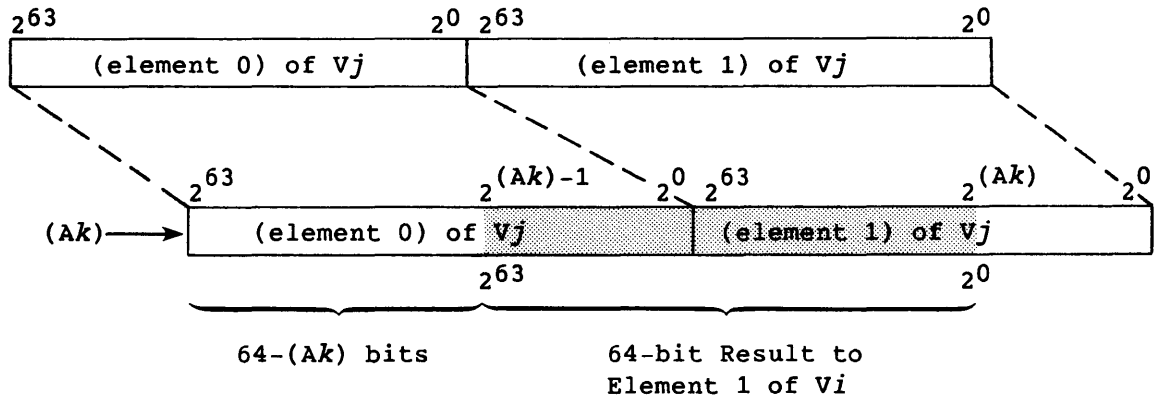


Figure 5-11. Vector Right Double Shift, Second Element, VL Greater than 1

INSTRUCTIONS 152 - 153 (continued)

The last operation performed by the instruction joins the last element of V_j as determined by (VL) with the preceding element. Figure 5-12 show this operation.

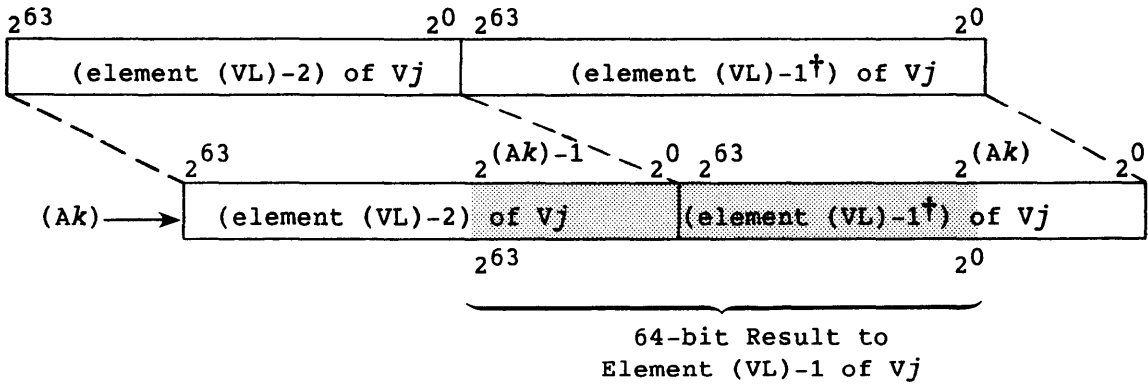


Figure 5-12. Vector Right Double Shift, Last Operation

Example 2:

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. Following execution, register V_0 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 V_0 are unaltered.

† Elements are numbered 0 through 63 in the V registers; therefore, element $(VL)-1$ refers to the VL^{th} element.

INSTRUCTIONS 152 - 153 (continued)

HOLD ISSUE CONDITIONS: V_j reserved as operand
 V_i reserved as operand or result
 A_k reserved (except A0)
Instructions 150 through 153 in process, unit busy (VL) + 4 CPs[†]

EXECUTION TIME: Instruction issue, 1 CP
 V_j ready in (VL) + 3 CPs if data available[†]
For instruction 152, V_i ready in (VL) + 9 CPs if data available[†]
Instruction 153, V_i ready in (VL) + 8 CPs if data available[†]
Unit ready, (VL) + 4 CPs if data available[†]

SPECIAL CASES: $(A_k)=1$ if $k=0$.

[†] Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 154 - 157

CAL Syntax	Description	Octal Code
$V_i \quad S_j + V_k$	Integer sums of (S_j) and (V_k elements) to V_i elements	154ijk
$V_i \quad V_j + V_k$	Integer sums of (V_j elements) and $V(k$ elements) to V_i elements	155ijk
$V_i \quad S_j - V_k$	Integer differences of (S_j) and (V_k elements) to V_i elements	156ijk
$V_i \quad -V_k^\dagger$	Transmit negative of (V_k elements) to V_i elements	156i0k
$V_i \quad V_j - V_k$	Integer differences of (V_j elements) and (V_k elements) to V_i elements	157ijk

The Vector Add functional unit executes instructions 154 through 157.

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 VL register contents. 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: V_k reserved as operand

V_i reserved as operand or result

Instructions 154 through 157 in process, unit busy (VL) + 4 CPs[†]

[†] Special CAL syntax

INSTRUCTIONS 154 - 157 (continued)

HOLD ISSUE CONDITIONS: For instructions 154 and 156, S_j reserved
(continued) (except S_0)

For instructions 155 and 157, V_j reserved as
operand

EXECUTION TIME: Instruction issue, 1 CP

V_j or V_k ready in (VL) + 3 CPs if data
available[†]

V_i ready in (VL) + 8 CPs if data available[†]

Unit ready, (VL) + 4 CPs if data available[†]

SPECIAL CASES: For instruction 154, if $j=0$, then $(S_j)=0$ and
 $(V_i \text{ element}) = (V_k \text{ element})$.

For instruction 156, if $j=0$, then $(S_j)=0$ and
 $(V_i \text{ element}) = -(V_k \text{ element})$.

[†] Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 160 - 167

CAL Syntax	Description	Octal Code
<i>Vi Sj*FVk</i>	Floating-point products of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	160ijk
<i>Vi Vj*FVk</i>	Floating-point products of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	161ijk
<i>Vi Sj*HVk</i>	Half-precision rounded floating-point products of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	162ijk
<i>Vi Vj*HVk</i>	Half-precision rounded floating-point products of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	163ijk
<i>Vi Sj*RVk</i>	Rounded floating-point products of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	164ijk
<i>Vi Vj*RVk</i>	Rounded floating-point products of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	165ijk
<i>Vi Sj*IVk</i>	Reciprocal iterations; $2-(Sj)*(Vk$ elements) to <i>Vi</i> elements.	166ijk
<i>Vi Vj*IVk</i>	Reciprocal iterations; $2-(Vj$ elements)* (<i>Vk</i> elements) to <i>Vi</i> elements.	167ijk

The Floating-point Multiply functional unit executes instructions 160 through 167. The number of operations performed by an instruction is determined by the VL register contents. 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 (*Sj*) to the functional unit where the copy is retained as one of the operands until the completion of the operation. Therefore, *Sj* can be changed immediately without affecting the vector operation. The other operand is an element of *Vk*. For instructions 161, 163, 165, and 167, both operands are obtained from V registers.

All results are delivered to elements of *Vi*. If either operand is not normalized, there is no guarantee that the products are normalized. If neither operand is normalized, the product is not normalized.

Section 4 describes out-of-range conditions.

INSTRUCTIONS 160 - 167 (continued)

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 . Refer to 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 . Refer to the description of instruction 067 for more details.

HOLD ISSUE CONDITIONS: V_k reserved as operand

V_i reserved as operand or result

Instruction 16X in process, unit busy
(VL) + 4 CPs[†]

[†] Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 160 - 167 (continued)

HOLD CONDITIONS: On mainframes equipped with Second Vector
(continued) Logical unit: instructions 140 through 145 in
process in Second Vector Logical unit. Unit busy
(VL) + 4 CPs

For instructions 160, 162, 164, and 166, S_j
reserved (except S_0)

For instructions 161, 163, 165, and 167, V_j
reserved as operand

EXECUTION TIME: Instruction issue, 1 CP

V_j and V_k ready in (VL) + 3 CPs if data
available†

V_i ready in (VL) + 12 CPs if data available†

Unit ready, (VL) + 4 CPs if data available†

SPECIAL CASES: $(S_j)=0$ if $j=0$.

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 170 - 173

CAL Syntax	Description	Octal Code
<i>Vi Sj+FVk</i>	Floating-point sums of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	170ijk
<i>Vi +FVk†</i>	Transmit normalized (<i>Vk</i> elements) to <i>Vi</i> elements	170i0k
<i>Vi Vj+FVk</i>	Floating-point sums of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	171ijk
<i>Vi Sj-FVk</i>	Floating-point differences of (<i>Sj</i>) and (<i>Vk</i> elements) to <i>Vi</i> elements	172ijk
<i>Vi -FVk†</i>	Transmit normalized negatives of (<i>Vk</i> elements) to <i>Vi</i> elements	172i0k
<i>Vi Vj-FVk</i>	Floating-point differences of (<i>Vj</i> elements) and (<i>Vk</i> elements) to <i>Vi</i> elements	173ijk

† Special CAL syntax

The Floating-point Add functional unit executes instructions 170 through 173. Instructions 170 and 171 perform floating-point addition; instructions 172 and 173 perform floating-point subtraction. The number of additions or subtractions performed by an instruction is determined by the VL register contents. All operations start with element 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to *Vi* normalized and results are normalized even if the operands are not normalized.

Instructions 170 and 172 deliver a copy of (*Sj*) 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 *Vk*. For instructions 171 and 173, both operands are obtained from V registers. Section 4 describes out-of-range conditions.

HOLD ISSUE CONDITIONS: *Vk* reserved as operand

Vi reserved as operand or result

INSTRUCTIONS 170 - 173 (continued)

HOLD ISSUE CONDITIONS: Instructions 170 through 173 in process, unit
(continued) busy (VL) + 4 CPs[†]

For instructions 170 and 172, S_j reserved
(except S_0)

For instructions 171 and 173, V_j reserved as
operand

EXECUTION TIME: Instruction issue, 1 CP

V_j and V_k ready in (VL) + 3 CPs if data
available[†]

V_i ready in (VL) + 11 CPs if data available[†]

Unit ready, (VL) + 4 CPs if data available[†]

SPECIAL CASES: (S_j)=0 if $j=0$.

[†] Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTION 174

CAL Syntax	Description	Octal Code
V_i /HV j	Floating-point reciprocal approximation of (V j elements) to V_i elements	174ij0

The Reciprocal Approximation functional unit executes instruction 174. 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 VL register contents.

Instruction 174 occurs in the divide sequence to compute the quotients of floating-point quantities as described in section 4 under Floating-point Arithmetic.

The reciprocal approximation instruction produces results 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: V_i reserved as operand or result

V_j reserved as operand

Instruction 174 in process, unit busy for (VL) + 4 CPs[†]

EXECUTION TIME: Instruction issue, 1 CP

V_j ready in (VL) + 3 CPs if data available[†]

V_i ready in (VL) + 19 CPs if data available[†]

Unit ready, (VL) + 4 CPs if data available[†]

SPECIAL CASES: (V_i element) is meaningless if (V_j element) is not normalized; the unit assumes that bit 2⁴⁷ of (V_j element) is 1; no test of this bit is made.

[†] Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 174ij1 - 174ij2

CAL Syntax	Description	Octal Code
V_i PV_j	Population count of (V_j elements) to V_i elements	174ij1
V_i QV_j	Population count parity of (V_j elements) to V_i elements	174ij2

The Vector Population/Parity functional unit executes instructions 174ij1 and 174ij2, sharing some logic with the Reciprocal Approximation functional unit.

Instruction 174ij1 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 174ij2 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: V_i reserved as operand or result

V_j reserved as operand

Instructions 174xx1 and 174xx2 in process,
unit busy for (VL) + 4 CPs[†]

Instruction 174xx0 in process, unit busy for
(VL) + 9 CPs[†]

Instruction 070 in process, unit busy (070 issue
time) + 7 CPs[†]

[†] Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTIONS 174ij1 - 174ij2 (continued)

EXECUTION TIME: Instruction issue, 1 CP
 Vj ready in (VL) + 3 CPs if data available†
 Vi ready in (VL) + 10 CPs if data available†
 Unit ready, (VL) + 4 CPs if data available†

† Vector instructions may or may not start execution immediately; they execute as data becomes available. In particular, a memory conflict that slows execution of some elements of a vector load can cause delays in all instructions in the operation chain, starting with that load.

INSTRUCTION 175

CAL Syntax	Description	Octal Code
VM Vj,Z	VM=1 when (Vj element)=0	1750j0
VM Vj,N	VM=1 when (Vj element)≠0	1750j1
VM Vj,P	VM=1 when (Vj element) positive, (bit $2^{63}=0$), includes (Vj element)=0	1750j2
VM Vj,M	VM=1 when (Vj element) negative, (bit $2^{63}=1$)	1750j3
Vi,VM Vj,Z	VM=1 and (Vi compress element)=element index when (Vj element)=0	175ij4
Vi,VM Vj,N	VM=1 and (Vi compress element)=element index when (Vj element)≠0	175ij5
Vi,VM Vj,P	VM=1 and (Vi compress element)=element index when (Vj element) positive, (bit $2^{63}=0$), includes (Vj element)=0	175ij6
Vi,VM Vj,M	VM=1 and (Vi compress element)=element index when (Vj element) negative, (bit $2^{63}=1$)	175ij7

Vector mask and compress index instruction 175 is executed in the Full Vector Logical functional unit.

Instruction 1750j*k*, where *k*=0 through 3, creates a vector mask in VM based on the results of testing the contents of the elements of register Vj. Each bit of VM corresponds to an element of Vj. Bit 2^{63} corresponds to element 0; bit 2^0 corresponds to element 63.

Instruction 175ij*k*, where *k*=4 through 7, creates an identical vector mask as in 1750j*k* and in addition creates a compressed index list in register Vi based on the results of testing the contents of the elements of register Vj (refer to example).

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 used to select the compress index option.

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

INSTRUCTION 175 (continued)

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

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

If the k designator is 3, the VM bit is set to 1 when (Vj element) is negative and is set to 0 when (Vj element) is positive. A zero value is considered positive.

If the k designator is 4, the VM bit is set to 1 and register (Vi compress element) is set to Vj element index when (Vj element) is 0. Register Vi elements are written to and Vi element pointer advanced only when (Vj element) is 0.

If the k designator is 5, the VM bit is set to 1 and register (Vi compress element) is set to Vj element index when (Vj element) is nonzero. Register Vi elements are written to and Vi element pointer advanced only when (Vj element) is nonzero.

If the k designator is 6, the VM bit is set to 1 and register (Vi compress element) is set to Vj element index when (Vj element) is positive. Register Vi elements are written to and Vi element pointer advanced only when (Vj element) is positive. A zero value is considered positive.

If the k designator is 7, the VM bit is set to 1 and register (Vi compress element) is set to Vj element index when (Vj element) is negative. Register Vi elements are written to and Vi element pointer advanced only when (Vj element) is negative.

The number of elements tested is determined by the VL register contents. VM bits corresponding to untested elements of Vj are zeroed.

Vector mask instruction 175jkk, $k=0$ through 3, and compress index instruction 175ijk, $k=4$ through 7, provide a vector counterpart to the scalar conditional branch instructions.

HOLD ISSUE CONDITIONS: Vj reserved as operand

Instruction 14X in process, unit busy
(VL) + 4 CPs

Instruction 175 in process, unit busy
(VL) + 4 CPs

For instruction 175 ($k=4$ through 7), if
register Vi reserved as operand or result.

INSTRUCTION 175 (continued)

EXECUTION TIME:

Instruction issue, 1 CP

V_j ready, (VL) + 3 CPs if data available

For instruction 175 ($k=4$ through 7), V_i ready in (VL) + 10 CPs if data is available.

Except for instruction 073, VM ready (VL) + 4 CPs if data is available.

For instruction 073, VM ready (VL) + 5 CPs if data is available.

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) $\neq 0$.

$k=2$ or 6, VM bit $xx=1$ if (V_j element xx) is positive; 0 is a positive condition.

$k=3$ or 7, VM bit $xx=1$ if (V_j element xx) is negative.

$k=4$, (V_i compress element)= xx if (V_j element xx)=0.

$k=5$, (V_i compress element)= xx if (V_j element xx) $\neq 0$.

$k=6$, (V_i compress element)= xx if (V_j element xx) is positive; 0 is a positive condition.

$k=7$, (V_i compress element)= xx if (V_j element xx) is negative.

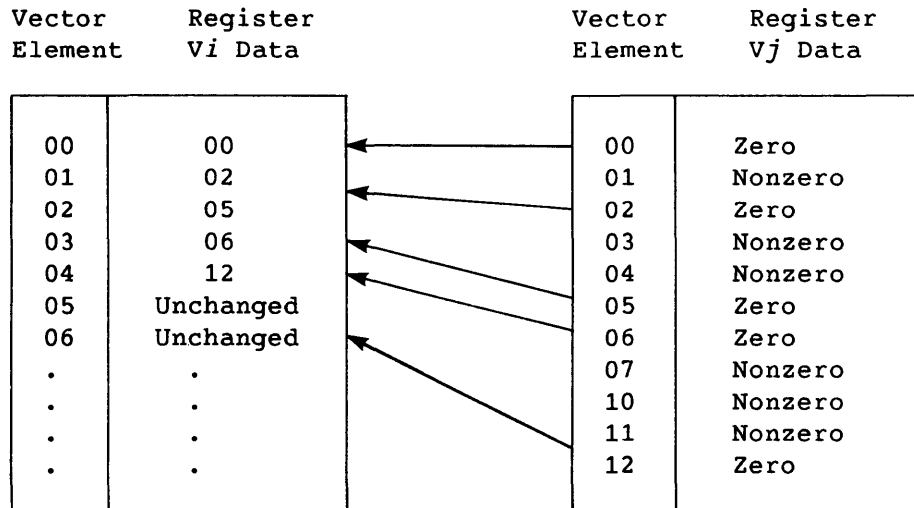
For instruction 175 ($k=4$ through 7), if no test conditions are true, then (VM)=0 and no writes to register V_i occur and the elements of V_i are unchanged by this instruction.

INSTRUCTION 175 (continued)

Example:

This example of the compress index instruction 175ij4 generates the same vector mask as instruction 1750j0 and also generates data into vector register Vi as follows:

Vector length=13g



INSTRUCTIONS 176 - 177

CAL Syntax	Description	Octal Code
V_i ,A0,Ak	Transmit (VL) words from memory to V_i elements starting at memory address (A0) and incrementing by (Ak) for successive addresses	176i0k
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	176i00
V_i ,A0,Vk	Transmit (VL) words from memory to V_i elements using memory address (A0) + (Vk elements)	176i1k
,A0,Ak V_j	Transmit (VL) words from V_j elements to memory starting at memory address (A0) and incrementing by (Ak) for successive addresses	1770jk
,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	1770j0
,A0,Vk V_j	Transmit (VL) words from V_j elements to memory using memory address (A0) + (Vk elements)	1771jk

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.

For instructions 176i0k and 1770jk, 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 Ak. Ak contains a signed 24-bit integer which is added to the address of the current word to obtain the address of the next word. Ak can specify either a positive or negative increment allowing both forward and backward streams of reference.

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

INSTRUCTIONS 176 - 177 (continued)

For instructions 176*ilk* and 1771*jk*, register elements begin with 0 and are incremented by 1 for each transfer. The low-order 24 bits of each element of *Vk* contains a signed 24-bit integer which is added to (A0) to obtain the current memory address.

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

HOLD ISSUE CONDITIONS: For instruction 176 if Ports A and B busy

For instruction 177 if Port C busy

For instructions 176*ilk* and 1771*jk*, if 176*ilk* or 1771*jk* in progress

A0 reserved

For instructions 176*i0k* and 1770*jk*, if *Ak* reserved where *k*=1 through 7

Scalar reference in CP1, CP2, CP3, or CP4

For instruction 176, V register *i* reserved as operand or result

For instruction 177, V register *j* reserved as operand

For instruction 176*ilk* and 1771*jk*, V register *k* reserved as operand

If not bidirectional memory mode, then instruction 176 holds on Port C busy and instruction 177 holds on Port A or B busy.

EXECUTION TIME:

For instruction 176*i0k*:

Instruction issue, 1 CP

Vi ready, (VL) + 17 CPs if memory is available

Port A or B busy, (VL) + 6 CPs

For instruction 1770*jk*:

Instruction issue, 1 CP

Vj ready, (VL) + 3 CPs if data is available

Port C busy, (VL) + 7 CPs

For instruction 176*ilk*:

Instruction issue, 1 CP

INSTRUCTIONS 176 - 177 (continued)

EXECUTION TIME: V_i ready, (VL) + 21 CPs if memory is available
(continued) V_k ready, (VL) + 3 CPs if data is available
Port A or B busy, (VL) + 10 CPs
176i1k busy, (VL) + 10 CPs

For instruction 1771jk:
Instruction issue, 1 CP
 V_i and V_k ready, (VL) + 3 CPs if data is available
Port C busy, (VL) + 10 CPs
1771jk busy, (VL) + 10 CPs

SPECIAL CASES: For instructions 176i0k and 1770jk,
increment (A0)=1 if $k=0$.

Instruction 176 uses Port B. If Port B is busy at issue time, instruction 176 uses Port A.
Instruction 177 uses Port C.

For instructions 176i0k and 1770jk:
(Ak) determines the memory increment.
Successive addresses are located in successive banks. References to the same bank can be made every 4 CPs or more. Incrementing (Ak) by 64 places successive memory references in the same bank, so a word is transferred every 4 CPs or more. If the address is incremented by 32, every other reference is to the same bank, and words can transfer no faster than one every 2 CPs. With any address incrementing that allows 4 CPs before addressing the same bank, the words can transfer each CP.

Memory conflict can slow loading or storing of individual vector elements. The elements are loaded or stored in order, so any delay for any element delays all succeeding elements.

For instruction 176:
If there is an instruction using its destination register as a source, the execution of that instruction is delayed whenever there is a delay in instruction 176 results.

APPENDIX SECTION



INSTRUCTION SUMMARY FOR CRAY X-MP SINGLE-PROCESSOR COMPUTER SYSTEMS

A

Instructions for the CRAY X-MP, models 11, 12, 14, and 18 are listed in numerical order on the following pages. The following abbreviations are used:

<u>Abbreviation</u>	<u>Definition</u>
Pop/LZ	Scalar Population/Parity/Leading Zero functional unit
A Int Add	Address Add functional unit
A Int Mult	Address Multiply functional unit
S Logical	Scalar Logical functional unit
S Shift	Scalar Shift functional unit
S Int Add	Scalar Add functional unit
Fp Add	Floating-point Add functional unit
Fp Mult	Floating-point Multiply functional unit
Fp Rcpl	Reciprocal Approximation functional unit
V Logical	Vector Logical functional unit

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
000000	ERR	-	Error exit
0010jk†	CA, Aj Ak	-	Set the channel (Aj) current address to (Ak) and begin the I/O sequence
0011jk†	CL, Aj Ak	-	Set the channel (Aj) limit address to (Ak)
0012j0†	CI, Aj	-	Clear Channel (Aj) Interrupt flag; clear device master-clear (output channel).
0012j1†	MC, Aj	-	Clear Channel (Aj) Interrupt flag; set device master-clear (output channel); clear device ready-held (input channel).
0013j0†	XA Aj	-	Enter XA register with (Aj)
0014j0†	RT Sj	-	Enter RTC register with (Sj)
001403†	CLN 0	-	Enter CLN register with 0
001413†	CLN 1	-	Enter CLN register with 1
001423†	CLN 2	-	Enter CLN register with 2
001433†	CLN 3	-	Enter CLN register with 3
0014j4†	PCI Sj	-	Enter II register with (Sj)
001405†	CCI	-	Clear PCI request
001406†	ECI	-	Enable PCI request
001407†	DCI	-	Disable PCI request
0015j0†	†††	-	Select performance monitor
001501†	†††	-	Set maintenance read mode
001511†	†††	-	Load diagnostic check byte with S1
001521†	†††	-	Set maintenance write mode 1
001531†	†††	-	Set maintenance write mode 2
00200k	VL Ak	-	Transmit (Ak) to VL register
002000††	VL 1	-	Transmit 1 to VL register
002100	EFI	-	Enable interrupt on floating-point error
002200	DFI	-	Disable interrupt on floating-point error
002300	ERI	-	Enable operand range interrupts
002400	DRI	-	Disable operand range interrupts
002500	DBM	-	Disable bidirectional memory transfers
002600	EBM	-	Enable bidirectional memory transfers
002700	CMR	-	Complete memory references
0030j0	VM Sj	-	Transmit (Sj) to VM register

† Privileged to monitor mode
†† Special CAL syntax
††† Not currently supported

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
003000†	VM 0	-	Clear VM register
0034jk	SMjk 1,TS	-	Test and set semaphore <i>jk</i> in SM
0036jk	SMjk 0	-	Clear semaphore <i>jk</i> in SM
0037jk	SMjk 1	-	Set semaphore <i>jk</i> in SM
004000	EX	-	Normal exit
0050jk	J Bjk	-	Jump to (Bjk)
006ijkm	J exp	-	Jump to <i>exp</i>
007ijkm	R exp	-	Return jump to <i>exp</i> ; set B00 to P.
010ijkm	JAZ exp	-	Branch to <i>exp</i> if (A0)=0
011ijkm	JAN exp	-	Branch to <i>exp</i> if (A0)≠0
012ijkm	JAP exp	-	Branch to <i>exp</i> if (A0) positive; 0 is positive.
013ijkm	JAM exp	-	Branch to <i>exp</i> if (A0) negative
014ijkm	JSZ exp	-	Branch to <i>exp</i> if (S0)=0
015ijkm	JSN exp	-	Branch to <i>exp</i> if (S0)≠0
016ijkm	JSP exp	-	Branch to <i>exp</i> if (S0) positive; 0 is positive.
017ijkm	JSM exp	-	Branch to <i>exp</i> if (S0) negative
01hijkm	Ah exp	-	Transmit <i>exp=ijkm</i> to Ah
020ijkm	Ai exp	-	Transmit <i>exp=jkm</i> to Ai
021ijkm	Ai exp	-	Transmit <i>exp=ones complement of jkm</i> to Ai
022ijk	Ai exp	-	Transmit <i>exp=jk</i> to Ai
023ijo	Ai Sj	-	Transmit (Sj) to Ai
023i01	Ai VL	-	Transmit (VL) to Ai
024ijk	Ai Bjk	-	Transmit (Bjk) to Ai
025ijk	Bjk Ai	-	Transmit (Ai) to Bjk
026ijo	Ai PSj	Pop/LZ	Population count of (Sj) to Ai
026ij1	Ai QSj	Pop/LZ	Population count parity of (Sj) to Ai
026ij7	Ai SBj	-	Transmit (SBj) to Ai
027ijo	Ai ZSj	Pop/LZ	Leading zero count of (Sj) to Ai
027ij7	SBj Ai	-	Transmit (Ai) to SBj
030ijk	Ai Aj+Ak	A Int Add	Integer sum of (Aj) and (Ak) to Ai
030i0k†	Ai Ak	A Int Add	Transmit (Ak) to Ai
030ij0†	Ai Aj+1	A Int Add	Integer sum of (Aj) and 1 to Ai
031ijk	Ai Aj-Ak	A Int Add	Integer difference of (Aj) less (Ak) to Ai
031i00†	Ai -1	A Int Add	Transmit -1 to Ai
031i0k†	Ai -Ak	A Int Add	Transmit the negative of (Ak) to Ai
031ij0†	Ai Aj-1	A Int Add	Integer difference of (Aj) less 1 to Ai
032ijk	Ai Aj*Ak	A Int Mult	Integer product of (Aj) and (Ak) to Ai

† Special CAL syntax

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
033i00	Ai CI	-	Channel number to Ai (j=0)
033ij0	Ai CA,Aj	-	Address of channel (Aj) to Ai (j≠0; k=0)
033ij1	Ai CE,Aj	-	Error flag of channel (Aj) to Ai (j≠0; k=1)
034ijk	Bjk,Ai ,A0	Memory	Read (Ai) words to B register jk from (A0)
034ijk†	Bjk,Ai 0,A0	Memory	Read (Ai) words to B register jk from (A0)
035ijk	,A0 Bjk,Ai	Memory	Store (Ai) words at B register jk to (A0)
035ijk†	0,A0 Bjk,Ai	Memory	Store (Ai) words at B register jk to (A0)
036ijk	Tjk,Ai ,A0	Memory	Read (Ai) words to T register jk from (A0)
036ijk†	Tjk,Ai 0,A0	Memory	Read (Ai) words to T register jk from (A0)
037ijk	,A0 Tjk,Ai	Memory	Store (Ai) words at T register jk to (A0)
037ijk†	0,A0 Tjk,Ai	Memory	Store (Ai) words at T register jk to (A0)
040ijkm	Si exp	-	Transmit jkm to Si
041ijkm	Si exp	-	Transmit exp=ones complement of jkm to Si
042ijk	Si <exp	S Logical	Form ones mask exp bits in Si from the right; jk field gets 64 - exp.
042ijk†	Si #>exp	S Logical	Form zeros mask exp bits in Si from the left; jk field gets 64 - exp.
042i77†	Si 1	S Logical	Enter 1 into Si
042i00†	Si -1	S Logical	Enter -1 into Si
043ijk	Si >exp	S Logical	Form ones mask exp bits in Si from the left; jk field gets exp.
043ijk†	Si #<exp	S Logical	Form zeros mask exp bits in Si from the right; jk field gets 64 - exp.
043i00†	Si 0	S Logical	Clear Si
044ijk	Si Sj&Sk	S Logical	Logical product of (Sj) and (Sk) to Si
044ij0†	Si Sj&SB	S Logical	Sign bit of (Sj) to Si
044ij0†	Si SB&Sj	S Logical	Sign bit of (Sj) to Si (j≠0)
045ijk	Si #Sk&Sj	S Logical	Logical product of (Sj) and ones complement of (Sk) to Si
045ij0†	Si #SB&Sj	S Logical	(Sj) with sign bit cleared to Si

† Special CAL syntax

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
046ijk	Si Sj\Sk	S Logical	Logical difference of (Sj) and (Sk) to Si
046ij0†	Si Sj\SB	S Logical	Toggle sign bit of Sj, then enter into Si
046ij0†	Si SB\Sj	S Logical	Toggle sign bit of Sj, then enter into Si (j≠0)
047ijk	Si #Sj\Sk	S Logical	Logical equivalence of (Sk) and (Sj) to Si
047i0k†	Si #Sk	S Logical	Transmit ones complement of (Sk) to Si
047ij0†	Si #Sj\SB	S Logical	Logical equivalence of (Sj) and sign bit to Si
047ij0†	Si #SB\Sj	S Logical	Logical equivalence of (Sj) and sign bit to Si (j≠0)
047i00†	Si #SB	S Logical	Enter ones complement of sign bit into Si
050ijk	Si Sj!Si&Sk	S Logical	Logical product of (Si) and (Sk) complement ORed with logical product of (Sj) and (Sk) to Si
050ij0†	Si Sj!Si&SB	S Logical	Scalar merge of (Si) and sign bit of (Sj) to Si
051ijk	Si Sj!Sk	S Logical	Logical sum of (Sj) and (Sk) to Si
051i0k†	Si Sk	S Logical	Transmit (Sk) to Si
051ij0†	Si Sj!SB	S Logical	Logical sum of (Sj) and sign bit to Si
051ij0†	Si SB!Sj	S Logical	Logical sum of (Sj) and sign bit to Si (j≠0)
051i00†	Si SB	S Logical	Enter sign bit into Si
052ijk	S0 Si<exp	S Shift	Shift (Si) left exp=jk places to S0
053ijk	S0 Si>exp	S Shift	Shift (Si) right exp=64 - jk places to S0
054ijk	Si Si<exp	S Shift	Shift (Si) left exp=jk places
055ijk	Si Si>exp	S Shift	Shift (Si) right exp=64 - jk places
056ijk	Si Si,Sj<Ak	S Shift	Shift (Si and Sj) left (Ak) places to Si
056ij0†	Si Si,Sj<1	S Shift	Shift (Si and Sj) left one place to Si
056i0k†	Si Si<Ak	S Shift	Shift (Si) left (Ak) places to Si
057ijk	Si Sj,Si>Ak	S Shift	Shift (Sj and Si) right (Ak) places to Si
057ij0†	Si Sj,Si>1	S Shift	Shift (Sj and Si) right one place to Si

† Special CAL syntax

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
057i0k†	Si Si>Ak	S Shift	Shift (Si) right (Ak) places to Si
060ijk	Si Sj+S _k	S Int Add	Integer sum of (Sj) and (Sk) to Si
061ijk	Si Sj-S _k	S Int Add	Integer difference of (Sj) and (Sk) to Si
061i0k†	Si -S _k	S Int Add	Transmit negative of (Sk) to Si
062ijk	Si Sj+F _S k	Fp Add	Floating-point sum of (Sj) and (Sk) to Si
062i0k†	Si +F _S k	Fp Add	Normalize (Sk) to Si
063ijk	Si Sj-F _S k	Fp Add	Floating-point difference of (Sj) and (Sk) to Si
063i0k†	Si -F _S k	Fp Add	Transmit normalized negative of (Sk) to Si
064ijk	Si Sj*F _S k	Fp Mult	Floating-point product of (Sj) and (Sk) to Si
065ijk	Si Sj*H _S k	Fp Mult	Half-precision rounded floating-point product of (Sj) and (Sk) to Si
066ijk	Si Sj*R _S k	Fp Mult	Full-precision rounded floating-point product of (Sj) and (Sk) to Si
067ijk	Si Sj*I _S k	Fp Mult	Two-floating-point product of (Sj) and (Sk) to Si
070ij0	Si /H _S j	Fp Rcpl	Floating-point reciprocal approximation of (Sj) to Si
071i0k	Si Ak	-	Transmit (Ak) to Si with no sign extension
071i1k	Si +Ak	-	Transmit (Ak) to Si with sign extension
071i2k	Si +F _A k	-	Transmit (Ak) to Si as unnormalized floating-point number
071i30	Si 0.6	-	Transmit constant 0.75*2**48 to Si
071i40	Si 0.4	-	Transmit constant 0.5 to Si
071i50	Si 1.	-	Transmit constant 1.0 to Si
071i60	Si 2.	-	Transmit constant 2.0 to Si
071i70	Si 4.	-	Transmit constant 4.0 to Si
072i00	Si RT	-	Transmit (RTC) to Si
072i02	Si SM	-	Transmit (SM) to Si
072ij3	Si STj	-	Transmit (STj) to Si
073i00	Si VM	-	Transmit (VM) to Si
073i11	††	-	Read performance counter into Si
073i21	††	-	Increment performance counter (maintenance)
073i31	††	-	Clear all maintenance modes

† Special CAL syntax
†† Not currently supported

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
073i01	Si SR0	-	Transmit (SR0) to Si
073i02	SM Si	-	Transmit (Si) to SM
073ij3	STj Si	-	Transmit (Si) to STj
074ijk	Si Tjk	-	Transmit (Tjk) to Si
075ijk	Tjk Si	-	Transmit (Si) to Tjk
076ijk	Si Vj,Ak	-	Transmit (Vj, element (Ak)) to Si
077ijk	Vi,Ak Sj	-	Transmit (Sj) to Vi element (Ak)
077i0k†	Vi,Ak 0	-	Clear Vi element (Ak)
10hijkm	Ai exp,Ah	Memory	Read from ((Ah) + exp) to Ai (A0=0)
100ijkm†	Ai exp,0	Memory	Read from (exp) to Ai
100ijkm†	Ai exp,	Memory	Read from (exp) to Ai
10hi000†	Ai ,Ah	Memory	Read from (Ah) to Ai
11hijkm	exp,Ah Ai	Memory	Store (Ai) to (Ah) + exp (A0=0)
110ijkm†	exp,0 Ai	Memory	Store (Ai) to exp
110ijkm†	exp, Ai	Memory	Store (Ai) to exp
11hi000†	,Ah Ai	Memory	Store (Ai) to (Ah)
12hijkm	Si exp,Ah	Memory	Read from ((Ah) + exp) to Si (A0=0)
120ijkm†	Si exp,0	Memory	Read from exp to Si
120ijkm†	Si exp,	Memory	Read from exp to Si
12hi000†	Si ,Ah	Memory	Read from (Ah) to Si
13hijkm	exp,Ah Si	Memory	Store (Si) to (Ah) + exp (A0=0)
130ijkm†	exp,0 Si	Memory	Store (Si) to exp
130ijkm†	exp, Si	Memory	Store (Si) to exp
13hi000†	,Ah Si	Memory	Store (Si) to (Ah)
140ijk	Vi Sj&Vk	V Logical	Logical products of (Sj) and (Vk) to Vi
141ijk	Vi Vj&Vk	V Logical	Logical products of (Vj) and (Vk) to Vi
142ijk	Vi Sj!Vk	V Logical	Logical sums of (Sj) and (Vk) to Vi
142i0k†	Vi Vk	V Logical	Transmit (Vk) to Vi
143ijk	Vi Vj!Vk	V Logical	Logical sums of (Vj) and (Vk) to Vi
144ijk	Vi Sj Vk	V Logical	Logical differences of (Sj) and (Vk) to Vi
145ijk	Vi Vj Vk	V Logical	Logical differences of (Vj) and (Vk) to Vi
145iii†	Vi 0	V Logical	Clear Vi
146ijk	Vi Sj!Vk&VM	V Logical	Transmit (Sj) if VM bit=1; (Vk) if VM bit=0 to Vi.

† Special CAL syntax

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
146i0k†	Vi #VM&Vk	V Logical	Vector merge of (Vk) and 0 to Vi
147ijk	Vi Vj!Vk&VM	V Logical	Transmit (Vj) if VM bit=1; (Vk) if VM bit=0 to Vi.
150ijk	Vi Vj<Ak	V Shift	Shift (Vj) left (Ak) places to Vi
150ij0†	Vi Vj<1	V Shift	Shift (Vj) left one place to Vi
151ijk	Vi Vj>Ak	V Shift	Shift (Vj) right (Ak) places to Vi
151ij0†	Vi Vj>1	V Shift	Shift (Vj) right one place to Vi
152ijk	Vi Vj,Vj<Ak	V Shift	Double shift (Vj) left (Ak) places to Vi
152ij0†	Vi Vj,Vj<1	V Shift	Double shift (Vj) left one place to Vi
153ijk	Vi Vj,Vj>Ak	V Shift	Double shift (Vj) right (Ak) places to Vi
153ij0†	Vi Vj,Vj>1	V Shift	Double Shift (Vj) right one place to Vi
154ijk	Vi Sj+Vk	V Int Add	Integer sums of (Sj) and (Vk) to Vi
155ijk	Vi Vj+Vk	V Int Add	Integer sums of (Vj) and (Vk) to Vi
156ijk	Vi Sj-Vk	V Int Add	Integer differences of (Sj) and (Vk) to Vi
156i0k†	Vi -Vk	V Int Add	Transmit negative of (Vk) to Vi
157ijk	Vi Vj-Vk	V Int Add	Integer differences of (Vj) and (Vk) to Vi
160ijk	Vi Sj*FVk	Fp Mult	Floating-point products of (Sj) and (Vk) to Vi
161ijk	Vi Vj*FVk	Fp Mult	Floating-point products of (Vj) and (Vk) to Vi
162ijk	Vi Sj*HVk	Fp Mult	Half-precision rounded floating-point products of (Sj) and (Vk) to Vi
163ijk	Vi Vj*HVk	Fp Mult	Half-precision rounded floating-point products of (Vj) and (Vk) to Vi
164ijk	Vi Sj*RVk	Fp Mult	Rounded floating-point products of (Sj) and (Vk) to Vi
165ijk	Vi Vj*RVk	Fp Mult	Rounded floating-point products of (Vj) and (Vk) to Vi
166ijk	Vi Sj*IVk	Fp Mult	Two-floating-point products of (Sj) and (Vk) to Vi
167ijk	Vi Vj*IVk	Fp Mult	Two-floating-point products of (Vj) and (Vk) to Vi

† Special CAL syntax

<u>CRAY X-MP</u>	<u>CAL</u>	<u>Unit</u>	<u>Description</u>
170ijk	Vi Sj+FVk	Fp Add	Floating-point sums of (Sj) and (Vk) to Vi
170i0k†	Vi +FVk	Fp Add	Normalize (Vk) to Vi
171ijk	Vi Vj+FVk	Fp Add	Floating-point sums of (Vj) and (Vk) to Vi
172ijk	Vi Sj-FVk	Fp Add	Floating-point differences of (Sj) and (Vk) to Vi
172i0k†	Vi -FVk	Fp Add	Transmit normalized negatives of (Vk) to Vi
173ijk	Vi Vj-FVk	Fp Add	Floating-point differences of (Vj) and (Vk) to Vi
174ij0	Vi /HVj	Fp Rcpl	Floating-point reciprocal approximations of (Vj) to Vi
174ij1	Vi PVj	V Pop	Population counts of (Vj) to Vi
174ij2	Vi QVj	V Pop	Population count parities of (Vj) to Vi
1750j0	VM Vj,Z	V Logical	VM=1 where (Vj)=0
1750j1	VM Vj,N	V Logical	VM=1 where (Vj)≠0
1750j2	VM Vj,P	V Logical	VM=1 if (Vj) positive; 0 is positive.
1750j3	VM Vj,M	V Logical	VM=1 if (Vj) negative
175ij4	Vi,VM Vj,Z	V Logical	VM=1 and (Vi)=element index if (Vj)=0
175ij5	Vi,VM Vj,N	V Logical	VM=1 and (Vi)=element index if (Vj)≠0
175ij6	Vi,VM Vj,P	V Logical	VM=1 and (Vi)=element index if (Vj) positive
175ij7	Vi,VM Vj,M	V Logical	VM=1 and (Vi)=element index if (Vj) negative
176i0k	Vi ,A0,Ak	Memory	Read (VL) words to Vi from (A0) incremented by (Ak)
176i00†	Vi ,A0,1	Memory	Read (VL) words to Vi from (A0) incremented by 1
176i1k	Vi ,A0,Vk	Memory	Read (VL) words to Vi using (A0) + (Vk)
1770jk	,A0,Ak Vj	Memory	Store (VL) words from Vj to (A0) incremented by (Ak)
1770j0†	,A0,1 Vj	Memory	Store (VL) words from Vj to (A0) incremented by 1
1771jk	,A0,Vk Vj	Memory	Store (VL) words from Vj using (A0) + (Vk)

† Special CAL syntax

6 MBYTE PER SECOND CHANNEL DESCRIPTIONS

B

Each input or output 6 Mbyte 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 (refer to section 2).

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 Mbyte per second input channel and an output channel.

6 MBYTE PER SECOND INPUT CHANNEL SIGNAL SEQUENCE

Table B-1 shows a general view of a 6 Mbyte per second input channel signal sequence. The following paragraphs describe data bits, parity bits, and each signal in the sequence.

DATA BITS 2^0 THROUGH 2^{15}

Data bits 2^0 through 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 25 ns 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.

Table B-1. Input Channel Signal Exchange

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

† Step 2 can initially precede step 1; that is, the first parcel and ready signal can arrive before requested.

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.

<u>Parity Bit</u>	<u>Data Bits</u>
0	2 ⁰ through 2 ³
1	2 ⁴ through 2 ⁷
2	2 ⁸ through 2 ¹¹
3	2 ¹² through 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 SIGNAL

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 \pm 10 ns wide (at 50 percent voltage points). The leading edge of the Ready signal at Central Memory begins the timing for sampling the data bits.

RESUME SIGNAL

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 \pm 8 ns wide (at 50-percent voltage points).

DISCONNECT SIGNAL

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 \pm 10 ns wide (at 50-percent voltage points).

6 MBYTE PER SECOND OUTPUT CHANNEL SIGNAL SEQUENCE

Table B-2 shows a general view of a 6 Mbyte per second output channel signal sequence. The data bits, parity bits, and each signal in the sequence are described following the table.

Table B-2. 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	→	
4	←	Resume
5 Data $2^{47} - 2^{32}$ with Ready	→	
6	←	Resume
7 Data $2^{31} - 2^{16}$ with Ready	→	
8	←	Resume
9 Data $2^{15} - 2^0$ with Ready	→	
10	←	Resume
11 If $(CA) \neq (CL)$, go to step 2		
12 Disconnect	→	
13 Set interrupt and deactivate channel		

DATA BITS 2^0 THROUGH 2^{15}

Data bits 2^0 through 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 ns 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:

<u>Parity Bit</u>	<u>Data Bits</u>
0	2 ⁰ through 2 ³
1	2 ⁴ through 2 ⁷
2	2 ⁸ through 2 ¹¹
3	2 ¹² through 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 SIGNAL

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 \pm 8 ns wide (at 50-percent voltage points). The leading edge of the Ready signal can be used to time data sampling in the external device.

RESUME SIGNAL

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 ns wide (at 50-percent voltage points).

DISCONNECT SIGNAL

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 \pm 8 ns wide (at 50-percent voltage points).

PERFORMANCE MONITOR

C

The system contains a set of eight performance counters to track certain hardware related events that can be used to indicate relative performance. The events that can be tracked are the number of specific instructions issued, hold issue conditions, the number of fetches, references, and so on, and are selected through instruction 0015j0. Table C-1 lists all operations that can be monitored.

Performance monitoring instructions allow the user to select specific hardware related events for monitoring, read the results of the performance monitors into a scalar register, and test the operation of the performance counters.

The instructions used for performance monitoring are:

<u>Octal Code</u>	<u>Description</u>
0015j0	Select performance monitor
073i11	Read performance counter into Si
073i21	Increment performance counter (maintenance)

All instructions are executed in monitor mode.

SELECTING PERFORMANCE EVENTS

Instruction 0015j0 selects for monitoring one of the four groups of hardware related events shown in table C-1 and clears all performance monitors. The low-order 2 bits of the j field select the group.

During each CP in nonmonitor (user) mode, the performance counters advance their totals according to the number of monitored events that occur. Each of the performance counters can increment at a maximum rate of +3 per CP. This allows a counter to continuously monitor for approximately 62 hours before it is reset.

Performance events are monitored only while operating in user (nonmonitor) mode. Entering monitor mode disables advancing of the performance counters.

Table C-1. Performance Counter Group Descriptions

Monitor Function	Performance Counter	Description	Increment Per CP
<i>j=0</i>	0	Number of: Instructions issued	+1
	1	CPs holding issue	+1
	2	Fetches	+1
	3	I/O references	+1
	4	CPU references	+3 max
	5	Floating-point add operations	+1
	6	Floating-point multiply operations	+1
	7	Floating-point reciprocal operations	+1
<i>j=1</i>	0	Hold issue conditions: Semaphores	+1
	1	Shared registers	+1
	2	A registers and functionals	+1
	3	S registers and functionals	+1
	4	V registers	+1
	5	V functional units	+1
	6	Scalar memory	+1
	7	Block memory	+1
<i>j=2</i>	0	Number of: Fetches	+1
	1	Scalar references	+1
	2	Scalar conflicts	+1
	3	I/O references	+1
	4	I/O conflicts	+1
	5	Block references	+3 max
	6	Block conflicts	+3 max
	7	Vector memory references	+3 max
<i>j=3</i>	0	Number of: 000 - 017 instructions	+1
	1	020 - 137 instructions	+1
	2	140 - 157, 175 instructions	+1
	3	160 - 174 instructions	+1
	4	176, 177 instructions	+1
	5	Vector integer operations	+3 max
	6	Vector floating-point operations	+3 max
	7	Vector memory references	+3 max

READING PERFORMANCE RESULTS

Performance counter totals can be read using instruction 073i1, which transmits either the high-order or low-order bits of a performance counter to the high-order bits of scalar register *Si* according to the contents of the performance counter pointer.

Entering monitor mode disables advancing of all performance counters and clears the performance counter pointer. The first execution of a 073i11 instruction reads the low-order bits of counter 0 into *Si* and increments the performance counter pointer. The second 073i11 instruction reads the high-order bits of counter 0 into *Si* and again increments the pointer. After each 073i11 instruction, the performance counter pointer is advanced by 1. Even values of the pointer select the low-order bits of a performance counter to be read into *Si*; odd values of the pointer select the high-order bits of the performance counter to be read.

Low-order bits 0 through 25 of the performance counter are read into bits 32 through 57 of *Si*. High-order bits 26 through 45 of the performance counter are read into bits 38 through 57 of *Si*.

A sequence for reading a set of performance counters appears as follows (there must be a 2-CP delay between sequential 073i11 instructions):

<u>Step</u>	<u>Octal Code</u>	<u>Description</u>
1	073i11	Low-order bits of counter 0 to <i>Si</i>
2		2 CP delay
3	073i11	High-order bits of counter 1 to <i>Si</i>
4		2 CP delay
5	073i11	Low-order bits of counter 1 to <i>Si</i>
6		2 CP delay
7	073i11	High-order bits of counter 2 to <i>Si</i>
8		2 CP delay
	.	.
	.	.
	.	.

TESTING PERFORMANCE COUNTERS

Instruction 073i21 is used to test the operation of the performance counters by incrementing the value stored in the counter while in monitor mode.

Entering monitor mode disables advancing of all performance counters by user programs and clears the performance counter pointer. This pointer determines which performance counter, and which bits in that counter, are incremented. Even values of the pointer increment bits 0 and 6 of the performance counter when instruction 073i21 is executed, odd values of the pointer increment bit 26. The pointer is advanced from even to odd and to the next counter through instruction 073i11.

There must be a 1-CP delay between sequential 073i21 instructions.

Execution of instruction 073i21 loads register *Si* with all ones as a side effect of the basic 073 instruction.

SECDED MAINTENANCE FUNCTIONS D

Modules involved with generating and interpreting the 8-bit check byte used for SECDED include logic that can be used for verifying check bit storage, check bit generation, and error detection and correction.

The instructions used for these maintenance mode functions are.

<u>Octal Code</u>	<u>Description</u>
001501	Set maintenance read mode
001511	Load diagnostic check byte with S1
001521	Set maintenance write mode 1
001531	Set maintenance write mode 2
073i31	Clear all maintenance modes

These instructions are all executed in monitor mode, and for instructions 0015xx, the maintenance mode switch (located on the mainframe's control panel) must be on or the instructions become no-ops.

VERIFICATION OF CHECK BIT STORAGE

To verify the storage ability of the SECDED check bits without moving memory modules, instructions 001501 and 001521 are used.

The maintenance write mode 1 instruction, 001521, replaces the 8 check bits generated by the SECDED circuitry with specific bits of a data word as it is written into memory. The maintenance read mode instruction, 001501, complements the write instruction by replacing the same bits of a data word with the 8 check bits as it is read from memory.

By using the instructions together (and with error correction disabled through the switch on the mainframe's control panel), specified bits of a data word are stored and read back through the check bit storage paths and verification of SECDED check bit storage operation is accomplished.

Instruction 001521, maintenance write mode 1, and 001501, maintenance read mode, replace data bits with check bits and vice versa as follows.

<u>Data Bit</u>		<u>Check Bit</u>
46		0
47		1
62		2
63	Read----->	3
14	<-----Write	4
15		5
30		6
31		7

VERIFICATION OF CHECK BIT GENERATION

The maintenance read mode instruction, 001501, is used to verify the correct generation of SECDED check bits for a word of data.

When the instruction is executed, the 8 check bits for SECDED replace specific data bits as the word is read into memory. A test program can easily extract these check bits and verify their correctness, thus checking the accuracy of the SECDED check bit circuitry.

Since the CPU replaces the data bits with check bits on all reads to memory until instruction 073i31 is executed (including fetch, scalar and vector reads, and I/O for the CPU), the test program should initially rewrite all of memory using the 001501 instruction to set up the SECDED check bits for a subsequent read by fetch or I/O.

Error correction must be disabled during this test.

VERIFICATION OF ERROR DETECTION AND CORRECTION

The maintenance write mode 2 instruction, 001531, and the load diagnostic check byte with S1 instruction, 001511, are used to verify operation of the SECDED circuitry.

To verify operation, a diagnostic check byte is initially loaded with the high-order bits of register S1 through instruction 001511 as follows:

<u>S1 Bit</u>	<u>Diagnostic Check Bit</u>
56	0
57	1
58	2
59	3
60	4
61	5
62	6
63	7

This diagnostic check byte is then written into memory in place of the normal SECEDED check bits on any subsequent CPU write to memory (writes from I/O through this CPU are not affected). With error correction enabled (through the switch on the mainframe's control panel), a subsequent read of the memory location allows different paths within the error detection and correction circuitry to be checked out.

The diagnostic check byte retains its value until a new one is entered.

CLEARING MAINTENANCE MODE FUNCTIONS

Instruction 073i31, clears all maintenance modes, clears the following maintenance mode instructions:

<u>Octal Code</u>	<u>Description</u>
001501	Set maintenance read mode
001521	Set maintenance write mode 1
001531	Set maintenance write mode 2

A Master Clear also clears the instructions.

As a side effect of the 073i31 instruction, *Si* is loaded with all ones.

INDEX



INDEX

- 1-parcel instruction format
 - with combined *j* and *k* fields, 5-2
 - with discrete *j* and *k* fields, 5-1
- 100 Mbyte per second channel to IOP, 2-14
- 1250 Mbyte per second channel to SSD, 2-14
- 16-bank phasing, 2-7
- 2-parcel instruction format
 - with combined *i*, *j*, *k* and *m* fields, 5-4
 - with combined *j*, *k*, and *m* fields, 5-3
- 6 Mbyte per second channel
 - input signal sequence, B-1
 - operation, 2-15
 - output signal sequence, B-3
- 6 Mbyte per second input channel signal sequence
 - data bits 2^0 through 2^{15} , B-1
 - disconnect signal, B-3
 - parity bits 0 through 3, B-2
 - ready signal, B-3
 - resume signal, B-3
- 6 Mbyte per second output channel signal sequence
 - data bits 2^0 through 2^{15} , B-4
 - disconnect signal, B-5
 - parity bits 0 through 3, B-5
 - ready signal, B-5
 - resume signal, B-5
- 8-bit check byte, 2-8
- 8-bit Status register, 4-8

- A registers, 4-3
- Access priorities, Central Memory, 2-7
- Access time, memory, 2-1
- Active Exchange Package, 3-13
- Addition algorithm, 4-28
- Address Add functional unit, 4-15
- Address functional unit, 4-15
- Address Multiply functional unit, 4-15
- Address processing, 4-1
- Address registers, 4-3
- Addressing, memory, 2-3
- Algorithm
 - addition, 4-28
 - derivation of division, 4-31
 - division, 4-31
 - multiplication, 4-29
- AND function, 4-36
- Arithmetic operations
 - floating-point, 4-22
 - Arithmetic operations (continued)
 - integer, 4-21
- Auxiliary I/O Processor (XIOP), 1-10

- B registers, 4-6
- Bank busy conflict, 2-7
- Banks, 2-1
- Beginning Address registers, 3-3
- Bidirectional Memory Mode (BDM) flag, 3-9
- Bidirectional memory references, 2-5
- BIOP, see Buffer I/O Processor
- Block reads and writes, concurrent, 2-5
- Block transfer references, 2-5
- Branching, forward and backward, 3-4
- Buffer I/O Processor (BIOP), 1-10
- Buffers, instruction, 3-3

- CA register, see Current Address register
- Central memory
 - access, 2-5
 - access conflicts, 2-7
 - access ports, 2-5
 - access priorities, 2-7
 - access time, 2-1
 - addressing, 2-3
 - banks, 2-1
 - conflict resolution, 2-7
 - cycle time, 2-1
 - error correction, 2-8
 - features, 2-1
 - organization, 2-2
 - ports, 2-5
 - references per clock period, 2-2
 - sections, 2-2
 - sizes, 2-1
 - transfer rate, 2-1
 - types of conflicts, 2-7
 - word size, 2-1
- Central Processing Unit
 - computation section, 4-1
 - computational section characteristics, 4-2
 - control and data paths, 1-6
 - control sections, 3-1
 - input/output section, 2-13
 - instructions, 5-1
 - shared resources, 2-1
 - speed, 1-3
- Channel
 - 100 Mbyte per second, 2-13

Channel (continued)

- 6 Mbyte per second, 2-13
- features, 2-13
- groups, 2-19
- input channel error conditions, 2-18
- input programming, 2-17
- input/output data paths, 2-21
- I/O, 2-6
- I/O control, 2-20
- numbers, 2-22
- output channel programming, 2-18
- types, 1-4

Channel Limit Address register (CL), 2-15

Characteristics of system, 1-3

Check bits, 2-8

CIP register, see Current Instruction Parcel register

CL register, see Channel Limit Address register

Clear programmable clock interrupt request, 3-21

Clearing maintenance mode function, D-3

CLN register, 3-13

Clock

- programmable, 3-19
- real-time, 2-10

Clock period, 1-4

Cluster Number (CLN) register, 2-11, 3-13

Communication, Inter-CPU, 2-10

Computation section, characteristics, 4-2

Concurrent reads and writes, block, 2-5

Condensing units, 1-12

Configurations of system, 1-15

Conflicts

- bank busy, 2-7
- resolution, 2-7
- section access, 2-7
- scalar reference, 2-6

Control and data paths for the CPU, 1-6

Control signals, 2-16

Control, inter CPU, 2-10

Conventions

- number, 1-4
- register, 1-4

Correctable Memory Error Mode (ICM) flag, 3-9

Counter, Interrupt Countdown (ICD), 3-20

CP, see clock period

CPU, see Central Processing Unit

CSB (read address), 3-8

Current Address (CA) register, 2-15

Current Instruction Parcel register, 3-2

Cycle time, 2-1

Data Base Address (DBA) register, 3-18

Data formats, 4-22

Data Limit Address (DLA) register, 3-18

Data transfer

- I/O subsystem, 2-14
- solid-state storage device, 2-14

DCU, see disk controller unit

Deadlock (DL) flag, 3-11

Deadlock interrupt, 2-12

Deadstart sequence, 3-21

Decimal equivalents, 4-23

Derivation of division algorithm, 4-31

DIOP, see Disk I/O Processor

Disk controller unit (DCU), 1-10

Disk I/O Processor (DIOP), 1-10

Disk storage units (DSU), 1-10

Division algorithm, 4-31

Division algorithm, derivation of, 4-31

Double-precision numbers, 4-28

DSU, see disk storage unit

E (error type), 3-8

Enable second vector logical (ESVL), 3-10

Enhanced Addressing mode (EAM), 3-12

Error correction, see also SECDED

- Central Memory 2-8
- matrix, 2-9

ESVL, see enable second vector logical

Exchange

- initiated by deadstart sequence, 3-14
- initiated by interrupt flag set, 3-14
- initiated by program exit, 3-14
- mechanism, 3-5
- sequence, 3-14
- sequence issue conditions, 3-15

Exchange Address (XA) register, 3-5, 3-12

Exchange Package

- active, 3-13
- assignments, 3-6
- contents, 3-5
- enable Second Vector Logical, 3-10
- management, 3-15
- memory error data, 3-7
- processor number, 3-7
- vector not used (VNU), 3-10

Exclusive NOR function, 4-36

Exclusive OR function, 4-36

Exponent matrix for Floating-point Multiply, 4-26

External master clear sequence, 2-19

F register, see Flag register

Fetch

- following scalar store, 2-6
- request, 2-6

Field limits, 3-16

Flag (F) register, 3-11

Flags

- Bidirectional Memory Mode (BDM), 3-9
- Correctable Memory Error Mode (ICM), 3-9
- Deadlock (DL), 3-11
- Error Exit (EEX), 3-12
- Exchange register flags, 3-11
- Floating-point Error (FPE), 3-11
- Floating-point Error Mode (IFP), 3-9
- Floating-point Error Status (FPS), 3-9
- I/O Interrupt (IOI), 3-12
- Interrupt Monitor Mode (IMM), 3-9
- MCU Interrupt (MCU), 3-11
- Memory Error (ME), 3-11

Flags (continued)

- Monitor Mode (MM), 3-10
- Normal Exit (NEX), 3-12
- Operand Range Error (ORE), 3-11, 3-19
- Operand Range Error Mode (IOR), 3-9
- Program Range Error (PRE), 3-11, 3-19
- Programmable Clock Interrupt (PCI), 3-11
- Uncorrectable Memory Error Mode (IUM), 3-10
- Waiting for Semaphore (WS), 3-9

Floating-point

- Add functional unit, 4-25
- addition, 4-28
- data format, 4-22, 4-23
- Error (FPE) flag, 3-11
- Error Mode (IFP) flag, 3-9
- Error status (FPS) flag, 3-9
- functional units, 4-19
- Multiply functional unit, 4-25
- multiply partial-product sums pyramid, 4-30
- normalized numbers, 4-24
- range errors, 4-24
- range overflow, 4-24
- Reciprocal Approximation functional unit, 4-27
- subtraction, 4-28

Forward and backward branching, 3-4

Full Vector Logical Functional unit, 4-18

Functional units, 4-14

- Address, 4-14
- Address Add, 4-15
- Address Multiply, 4-15
- floating-point, 4-19
- Floating-point Add, 4-20
- Floating-point Multiply, 4-20
- Floating-point Reciprocal Approximation, 4-27
- Full Vector Logical, 4-18
- Reciprocal Approximation, 4-21
- scalar, 4-15
- Scalar Add, 4-15
- Scalar Logical, 4-16
- Scalar Population/Parity/Leading Zero, 4-16
- Scalar Shift, 4-16
- Second Vector Logical, 4-18
- vector, 4-16
- Vector Add, 4-17
- Vector Logical, 4-18
- Vector Population/Parity, 4-19
- vector reservation, 4-17
- Vector Shift, 4-17

g field, 5-1

General form for instructions, 5-1

Group descriptions, performance counter, C-2

h field, 5-1

Half-precision multiply, 4-30

i field, 5-1

IBA register, see Instruction Base address register

IBAR register, see Beginning Address register

ICD, see Interrupt Countdown counter

II register, see Interrupt Interval register

ILA register, see Instruction Limit Address register

In-buffer condition, 3-4

Inclusive OR function, 4-36

Input channel error conditions, 2-18

Input channel programming, 2-17

Input signal sequence, B-1

Instruction

- Base Address (IBA) register, 3-17
- buffers, 3-3
- descriptions, 5-6
- fetch following a scalar store, 2-6
- issue to memory ports, 2-5
- parcel, 3-3
- summary, A-1

Instruction format

- 1-parcel with discrete *j* and *k* fields, 5-1
- 1-parcel with combined *j* and *k* fields, 5-2
- 2-parcel with combined *j*, *k*, and *m* fields, 5-3
- 2-parcel with combined *i*, *j*, *k*, and *m* fields, 5-4

Instruction issue control elements, 3-1

Instruction issue, 5-5

Instruction Limit Address (ILA) register, 3-17

Instruction summary, A-1

Inter-CPU communication section, 2-10

Interfaces, 1-7

Interrupt

- Countdown (ICD) Counter, 3-20
- Interval (II) register, 3-20
- Monitor Mode flag, 3-10

I/O channel, 2-6

I/O interrupt, 2-16

I/O Interrupt (IOI) flag, 3-12

I/O memory

- addressing, 2-23
- conflicts, 2-22
- I/O lockout, 2-22
- request conditions, 2-23

I/O ports, 2-13

I/O Processors, types of

- Auxiliary I/O Processor (XIOP), 1-10
- Buffer I/O Processor (BIOP), 1-10
- Disk I/O Processor (DIOP), 1-10
- Master I/O Processor (MIOP), 1-8

I/O program flowchart, 2-17

I/O Subsystem, 1-8, 2-13

I/O Subsystem data transfer, 2-14

Issue, instruction, 5-5

Issue, 3-2

Italics, 1-4

j field, 5-1

 k field, 5-1

 Limit address (CL) register, 2-15
 Local Memory, 2-14
 Logical operations, 4-36
 Lower Instruction Parcel (LIP) register, 3-3

 m field, 5-1
 Mask operation, 4-37
 Master Clear sequence, to external devices, 2-19
 Master I/O Processor (MIOP), 1-8
 MCU Interrupt (MCU) flag, 3-10
 Memory access conflicts
 bank busy, 2-7
 resolution, 2-7
 section access, 2-7
 Memory, see also Central Memory
 access, 2-19
 access ports, 2-5
 addressing, I/O, 2-23
 bank conflicts, 2-22
 data path with SECEDED, 2-8
 error data, 3-7
 error data fields, 3-7
 Error (ME) flag, 3-11
 field protection, 3-16
 field registers, 3-8
 request condition, 2-23
 MIOP, see Master I/O Processor
 Mode (M) register, 3-9
 Monitor Mode (MM) flag, 3-10
 Motor-generator units, 1-14
 Multiplication algorithm, 4-29
 Multiply, half-precision, 4-30
 Multiply pyramid, 4-30

 Newton's method, 4-31
 Next Instruction Parcel (NIP) register, 3-2
 Normal Exit (NEX) flag, 3-12
 Normalized floating-point numbers, 4-24
 Numbers
 double-precision, 4-28
 normalized floating-point, 4-24
 Number conventions, 1-4

 Operand Range Error, 3-19
 flag (ORE), 3-11, 3-19
 Mode (IOR) 3-9
 Operating registers, 4-3
 Organization
 system, 1-5
 memory, 2-2
 Out-of-buffer condition, 3-4
 Output channel programming, 2-18

 P register, see Program Address register
 Parallel vector operations, 4-11
 Parity error, 2-18
 Performance
 counter group descriptions, C-2
 events, C-1
 Performance (continued)
 reading results, C-3
 monitor, 3-21, C-1
 Physical dimensions of system, 1-3
 PN, see Processor Number
 Ports
 Central Memory, 2-5
 memory access, 2-5
 Power distribution units, 1-13
 Processor number (PN), 3-7
 Program
 Address (P) register, 3-2, 3-8
 Range Error (PRE) flag, 3-19
 State (PS) register, 3-13
 Programmable clock, 3-19
 Programmable Clock Interrupt (PCI)
 flag, 3-11
 Programmed Master Clear to external device, 2-19
 PS register, 3-13

 R (read mode), 3-8
 Reading performance results, C-3
 Real-time Clock (RTC) register, 2-10
 Real-time clock, 2-10
 Reciprocal Approximation functional unit, 4-21
 References, memory, 2-5
 Register
 8-bit Status, 3-8
 A register, 3-13
 Address, 4-3
 B, 4-6
 Beginning Addresses, 3-3
 Cluster Number (CLN), 3-13
 control, 4-13
 Current Address (CA), 2-15
 Current Instruction Parcel (CIP), 3-2
 Data Base Address (DBA), 3-18
 Data Limit Address (DLA), 3-18
 Exchange Address (XA), 3-5
 Flag (F), 3-11
 Instruction Base Address (IBA), 3-17
 Instruction Limit Address (ILA), 3-17
 Interrupt Interval (II), 3-20
 Limit Address (CL), 2-15
 Lower Instruction Parcel (LIP), 3-3
 M (mode), 3-9
 Memory field, 3-8
 Next Instruction Parcel (NIP), 3-2
 operating, 4-3
 Program Address (P), 3-2, 3-8
 Program State (PS), 3-1
 Real-time Clock (RTC), 2-10
 S register, 3-13
 Scalar, 4-6
 Semaphore, 2-12

Register (continued)
 Shared Address (SB), 2-12
 Shared scalar (ST), 2-12
 status, 4-8
 Vector, 4-13
 Vector Length (VL), 4-13
 Vector mask (VM), 4-13
 Register conventions, 1-4
 Reservations and chaining, V register, 4-12
 RTC, see real-time clock

S registers, 4-6
 S (syndrome), 3-8
 SB registers, see Shared Address register
 Scalar
 Add functional unit, 4-15
 functional units, 4-15
 Logical functional units, 4-16
 memory references, 2-5
 Population/Parity/Leading Zero
 functional unit, 4-16
 processing, 4-1
 registers, 4-6
 Shift functional unit, 4-16
 SECEDED, 2-8
 SECEDED maintenance functions
 clearing maintenance mode functions, D-3
 verification of check bit generation,
 D-2
 verification of check bit storage, D-1
 verification of error detection and
 correction, D-2
 Second Vector Logical functional unit, 4-18
 Section access conflict, 2-7
 Semaphore register, 2-12
 Shared Address (SB) register, 2-12
 Shared Scalar (ST) register, 2-12
 SM register, see Semaphore register
 Solid-state Storage Device data transfer,
 2-14
 Solid-state Storage Device, 1-11
 Special characters, 5-7
 Special register values, 5-5
 SSD, see Solid-state Storage Device
 ST registers, see Shared Scalar register
 Status register, 4-8
 Syndrome bits, 2-9
 System
 basic organization, 1-5
 characteristics, 1-3
 components, 1-5
 configurations, 1-15
 description, 1-1
 physical dimensions, 1-3

T registers, 4-9
 Testing performance counters, C-4
 Transfer rate
 instruction buffers, 2-1
 I/O section, 2-1
 Two complement integer arithmetic, 4-21

Uncorrectable Memory Error Mode (IUM) flag,
 3-10

V register reservations and chaining, 4-12
 V registers, 4-9
 Vector
 Add functional unit, 4-17
 control registers, 4-13
 Full Vector Logical functional unit,
 4-18
 functional units, 4-16
 functional unit reservation, 4-17
 Length register, 4-13
 Mask register, 4-13
 Population/Parity functional unit, 4-19
 processing, 4-1
 registers, 4-9
 Shift functional unit, 4-17
 Verification of
 check bit generation, D-2
 check bit storage, D-1
 error detection and correction, D-2
 Vector not used (VNU), 3-10
 VNU, see vector not used

Waiting for Semaphore (WS) flag, 3-9
 Word size, memory, 2-1

XA register, see Exchange Address register
 XIOP, see Auxiliary I/O Processor

READER COMMENT FORM

CRAY X-MP/1 System Programmer Reference Manual

CSM0111000

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 _____

DATE _____



CUT ALONG THIS LINE

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

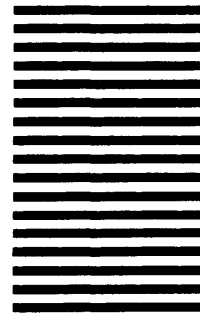
BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE



Attention: PUBLICATIONS
890 Industrial Boulevard
Technical Operations Building
Chippewa Falls, WI 54729



FOLD

STAPLE

READER COMMENT FORM

CRAY X-MP/1 System Programmer Reference Manual

CSM0111000

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 _____

DATE _____



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

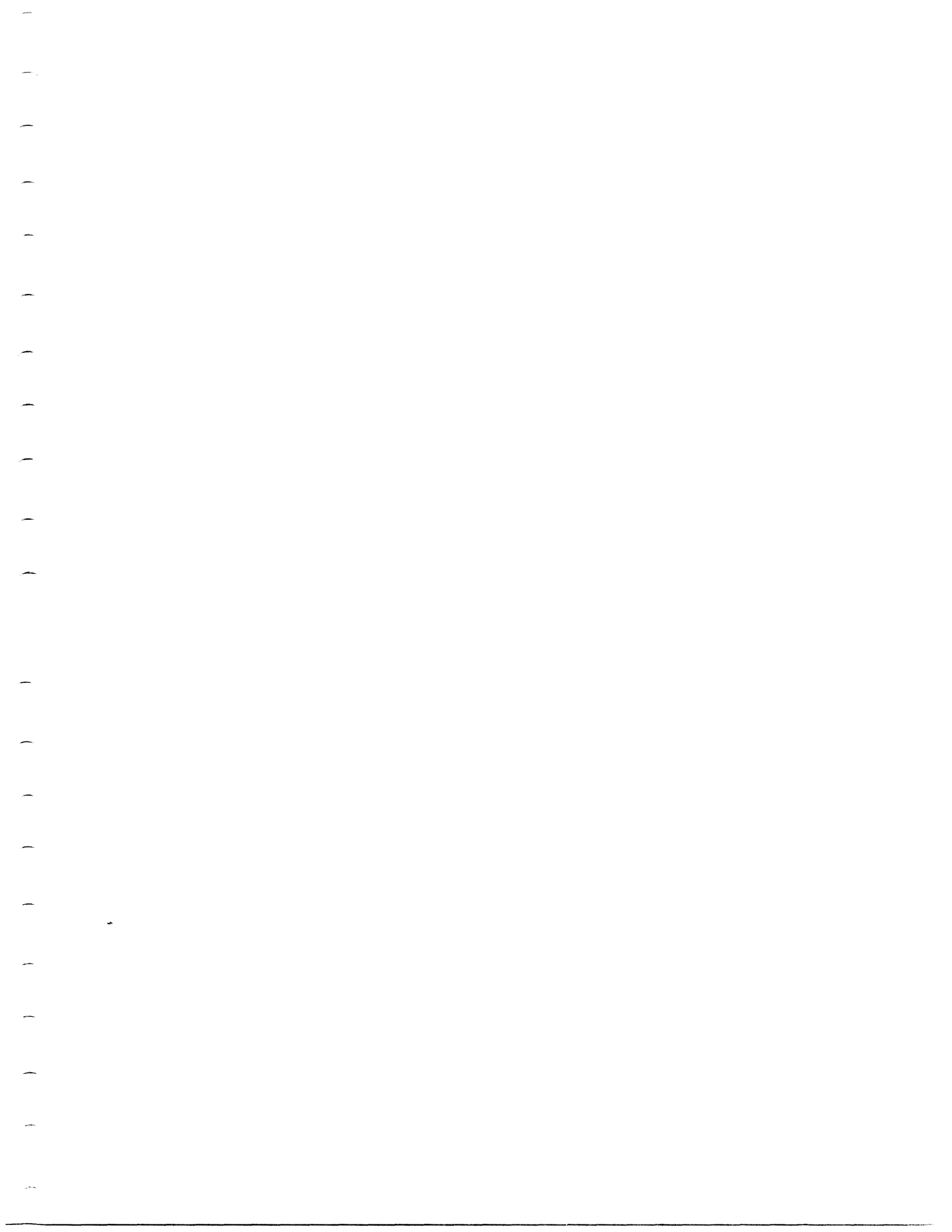
POSTAGE WILL BE PAID BY ADDRESSEE



Attention: PUBLICATIONS
890 Industrial Boulevard
Technical Operations Building
Chippewa Falls, WI 54729

FOLD

STAPLE







Cray Research, Inc.
Hardware Publications
Technical Operations Building
890 Industrial Boulevard
Chippewa Falls, WI 54729
715-723-2206