CRAY
RESEARCH, INC.

CRAY® COMPUTER SYSTEMS

CRAY X-MP COMPUTER SYSTEMS
FUNCTIONAL DESCRIPTION MANUAL

HR-3005

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER HR-3005

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 the Distribution Center and comments about these publications should be directed to:

CRAY RESEARCH, INC.
1345 Northland Drive
Mendota Heights, Minnesota 55120

Revision	Description
	February 1987 - Original printing.

PREFACE

This manual describes the functions of all CRAY X-MP computer systems currently manufactured by Cray Research, Inc. (CRI). For information on earlier models of the CRAY X-MP computer system, contact your local Cray representative.

This manual is written for customers. It describes the overall design of the CRAY X-MP computer systems; provides basic information on the computation section, exchange mechanism, and functional units; and explains the symbolic machine instructions. A fact sheet for each CRAY X-MP model provides specific information pertaining to that model.

This manual contains the following sections:

Section	Description
1	Contains the introduction to this manual
2	Describes the design of the CRAY X-MP CPU. The registers, functional units, and Exchange Package are described, and a block diagram is provided.
3	Provides detailed information on the instructions that operate on the CRAY X-MP computer system. Each machine instruction can be represented symbolically in Cray Assembly Language (CAL). The instructions are listed in octal form in a box format that provides the CAL syntax format, an operand if required, a brief description of each instruction, and the machine instruction. A detailed description of the instruction and an example using the instruction follow the boxed information.
4	Specific information for each of the current CRAY X-MP models is given, along with a photo and a maximum configuration drawing for each model.

For the reader's convenience, a glossary is also included; it defines many of the commonly used Cray acronymns.

CONTENTS

FIGURES

FIGURES (continued)

TABLES

CENTRAL PROCESSING UNITS

Each CPU has a computation section consisting of operating registers, functional units, and an instruction control network. The instruction control network makes all decisions related to instruction issue as well as coordinating the three types of processing (vector, scalar, or address). Each of the processing modes has its associated registers and functional units. In multiple-processor mainframes, the interprocessor section, which coordinates processing between CPUs and Central Memory, is shared.

Refer to section 2 for more information on the computation section.

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 computer system and receiving output from it for distribution to peripheral equipment. An FEI compensates for differences in channel widths, machine word size, electrical logic levels, and control signals. The FEI is housed in a stand-alone cabinet (figure 1-2) located near the host computer. Its operation is transparent to both the front-end computer user and the Cray user. As an option, a 3-Mybte/s fiber-optic link (FOL-3) is available for any front-end interface to provide front-end connections of up to .621 mile (1 km) and complete electrical separation from the Cray computer system.

The High-speed External channel (HSX-1) is a 100-Mybte/s channel that connects a CRAY X-MP computer system to external equipment supplied by the customer. This channel can be used over distances of up to 70 ft (21.3 m) and can drive machines such as high-speed graphic devices. Refer to the specification sheets in section 4 for specific configuration information.

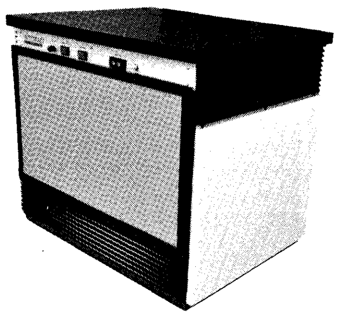


Figure 1-2. Typical Interface Cabinet

I/O SUBSYSTEM

The IOS, shown in figure 1-3, can have multiple I/O Processors (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. The IOS is usually housed in its own stand-alone cabinet. For the CRAY X-MP/14se, however, the IOS is housed in the mainframe chassis; refer to the specification sheets in section 4 for specific information.

Various types of IOPs may be configured in an IOS: a Master IOP (MIOP), a Buffer IOP (BIOP), a Disk IOP (DIOP), or an Auxiliary IOP (XIOP). Each IOP controls different portions of the system; the number of IOPs is site dependent.

Each IOP of the IOS has a memory section, a control section, a computation section, and an input/output (I/O) section. I/O sections are independent and handle some portion of the I/O requirements for the IOS. IOS hardware allows for simultaneous data transfers between the MIOP, BIOP, DIOP, or XIOP of the IOS and the mainframe's Central Memory.†

Each IOP controls a different portion of the system. The MIOP controls the FEIs and the standard group of station†† peripherals. It is connected to the Peripheral Expander; the Peripheral Expander contains controllers for peripheral devices. The MIOP is also connected to Buffer Memory and to the mainframe over a 6-Mbyte/s channel pair. The MIOP can control other functions on some systems, such as the HSX-1 channel interface; refer to the specification sheets in section 4 for more information.

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/s channel pair. The BIOP can control other functions on some systems; refer to the specification sheets in section 4 for more information.

The DIOP is used for additional disk storage units (DSUs). The DIOP connects to Buffer Memory and to the mainframe Central Memory over a 100-Mbyte/s channel pair. The DIOP data transfer sequence is similar to the BIOP's sequence.

---

† Software supporting the 100-Mbyte/s channel pair to the XIOP and MIOP is currently not available.

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

The XIOP is used for block multiplexer channels and interfaces to the block multiplexer controllers (BMCs). In most CRAY X-MP computer systems, the XIOP also interfaces with the HSX channel; refer to the specification sheets in section 4 for more information.

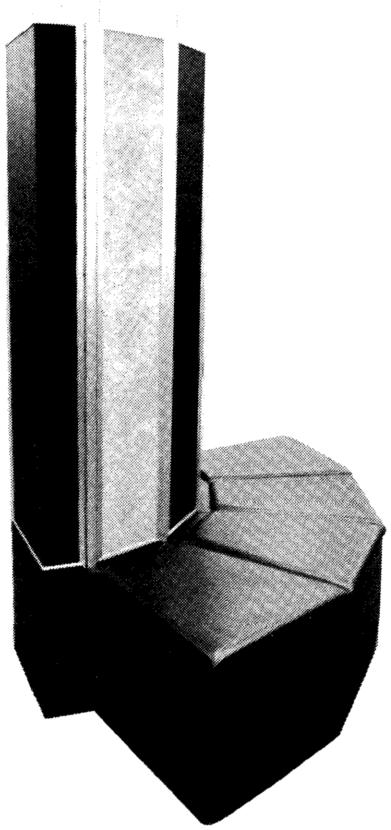


Figure 1-3. I/O Subsystem Chassis

DISK STORAGE UNITS

For mass storage, the system uses CRI disk storage units (DSUs). A disk controller unit (DCU) interfaces the DSUs to an IOP within the IOS.

The IOP and the disk controller unit can transfer data between the direct memory access (DMA) port of the IOP and multiple DSUs, without missing data or skipping revolutions, even when all DSUs are operating at full speed.

SOLID-STATE STORAGE DEVICE

The SSD shown in figure 1-4 is an optional, high-performance device used for temporary data storage. It transfers data between the mainframe's Central Memory and the SSD through one or two special Cray interface cables with a maximum speed of 1000-Mbyte/s each. The actual speed of these transfers depends on the SSD and CRAY X-MP memory size and system configuration. The SSD can also be connected directly to an IOP over a 100-Mbyte/s channel pair.

The SSD-3I is a special version of the SSD; it is housed within the IOS chassis. Refer to section 4 for all the current SSD configurations available.

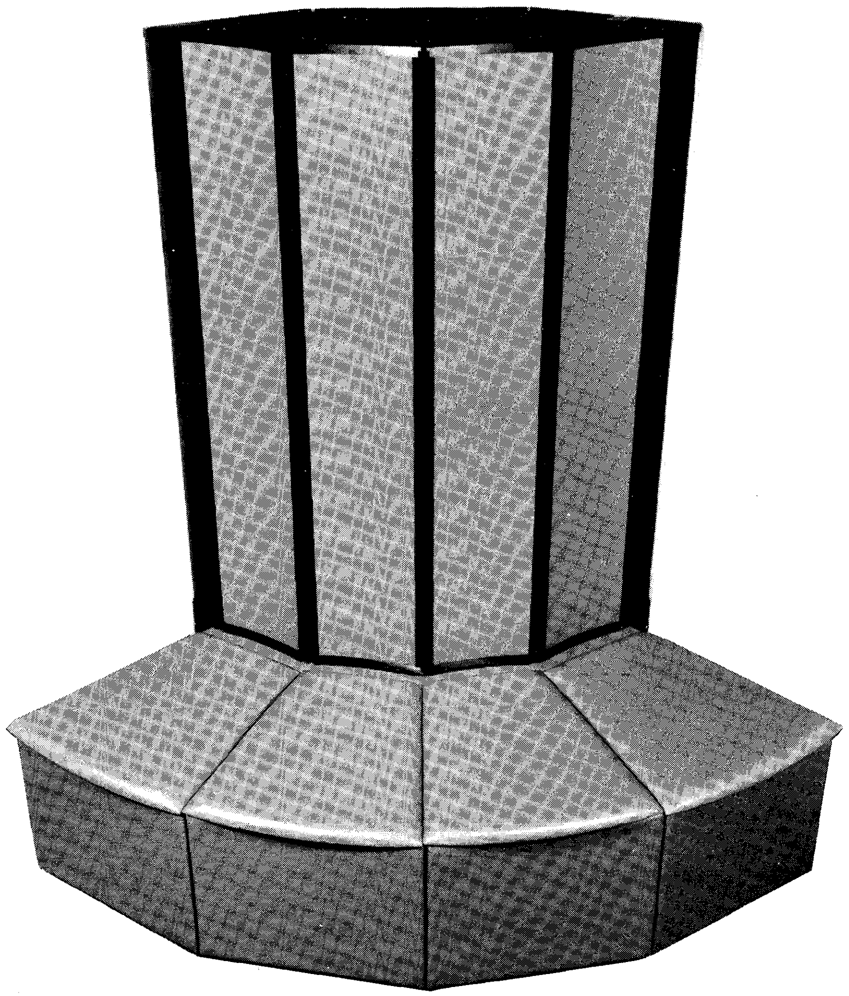
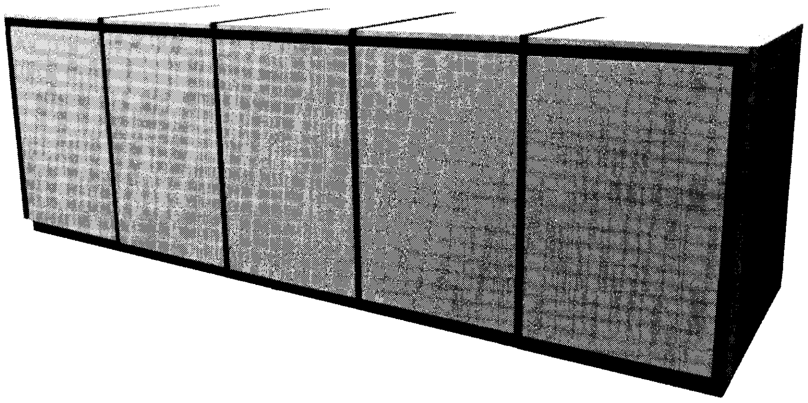


Figure 1-4. Solid-state Storage Device Chassis

CONDENSING UNITS

A condensing unit (figure 1-5) contains the major components of the refrigeration system used to cool the computer chassis. Heat is removed from the condensing unit by a second-level cooling system that is not part of the computer system.



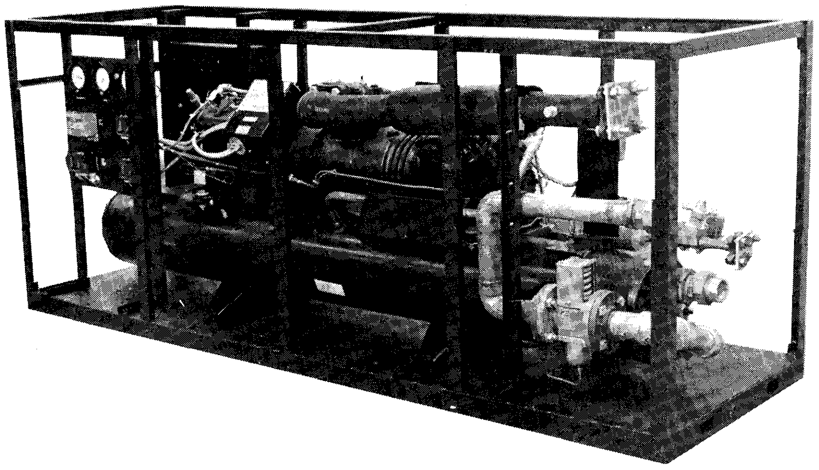


Figure 1-5. Condensing Unit

POWER DISTRIBUTION UNITS

The mainframe, IOS, and SSD may have independent power distribution units (PDUs). Refer to section 4 for required number of PDU's needed for each model.

The PDU for the mainframe contains adjustable transformers for regulating the voltage to each column of the mainframe. The PDU also contains temperature and voltage monitoring equipment that checks temperatures at strategic locations on the mainframe chassis. Automatic warning and shutdown circuitry protects the mainframe in case of overheating or excessive cooling. Control switches for the motor-generators and the condensing unit are also mounted on the mainframe's power distribution unit.

A pair of PDUs perform similar functions for the IOS chassis and the SSD chassis.

Figure 1-6 shows the power distribution units for the CRAY X-MP/4 mainframe (left) and for the CRAY X-MP/1 and CRAY X-MP/2 mainframes, IOS, and SSD (right).

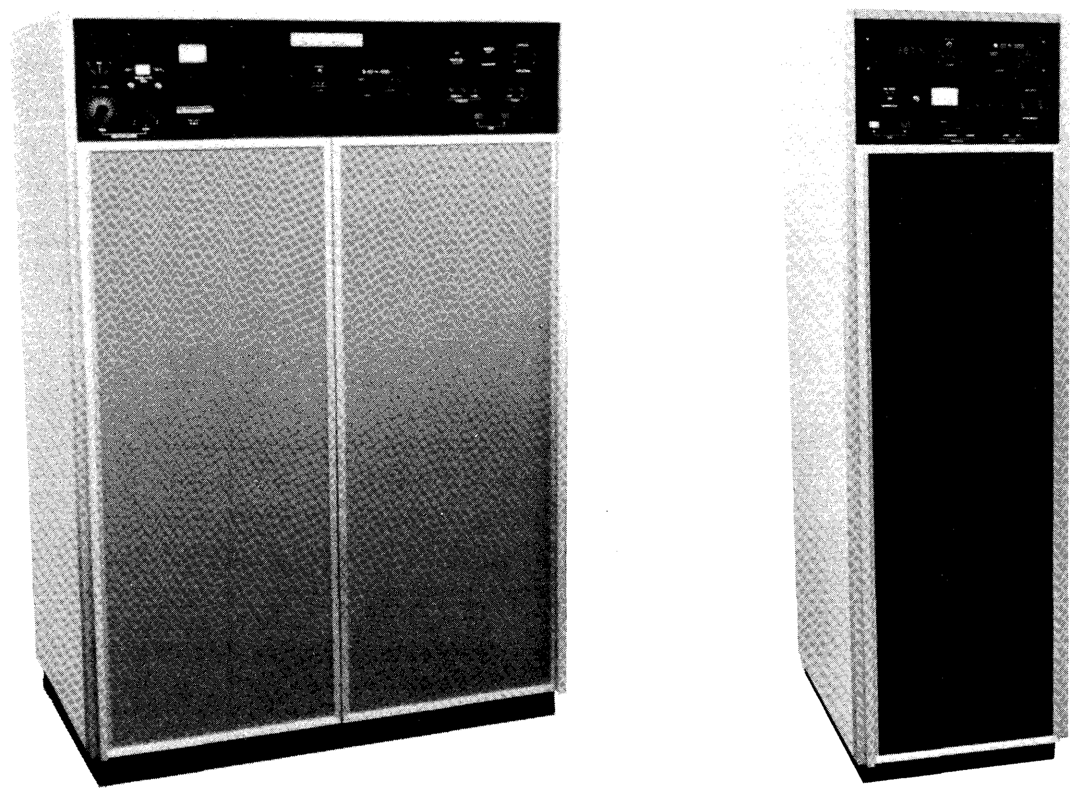


Figure 1-6. Power Distribution Units

MOTOR-GENERATOR UNITS

A motor-generator unit (figure 1-7) converts primary power from commercial power mains to the 400-Hz power used by the system. This unit also isolates the system from transients and fluctuations on the commercial power mains. The equipment consists of three motor-generator units.

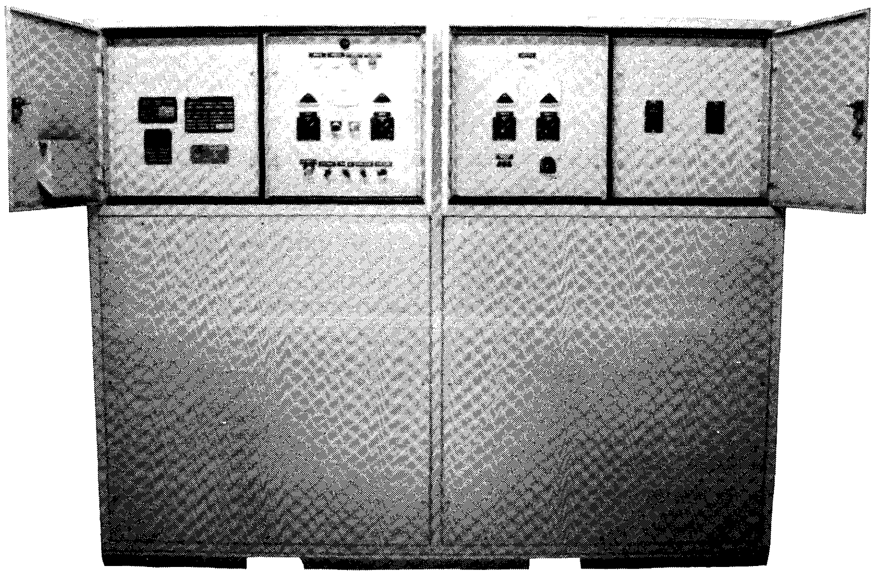


Figure 1-7. Motor-generator Equipment

CONVENTIONS

This manual uses the following conventions:

Convention	Description
lowercase italic	Variable information
X or x or *x*	An ignored value
n	An unknown variable value
(X*x*)	The contents of a register designated by the X*x* value
Register bit designators	Numbered right to left as powers of 2, designators starting with 2^0. Exceptions are the Exchange Package and the Vector Mask register; Exchange Package bits are numbered from left to right and are numbered not as powers of 2 but as bits 0 through 63, with bit 0 as the most significant and bit 63 as the least significant. Vector Mask register bits correspond to a word element in a vector register. Bit 2^{63} corresponds to element 0, bit 2^0 corresponds to element 63.

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.

EXAMPLES

The following are examples of the preceding conventions.

Examples	Description
Transmit (A*k*) to S*i*	Transmit the contents of the A register specified by the *k* designator to the S register specified by the *i* designator.
167*ixk*	Machine instruction 167. The *j* register designator is not used and is an ignored value.

Examples	Description
Read *n* words from memory	Read an unknown variable number of words from memory. You can read, within the stated restrictions, as few or many words from memory as you wish.
Bit 2^{63} of an S or element of a V register	The value represents the most significant bit.

CRAY X-MP DESIGN DETAILS 2

The following subsections describe the major components of a CRAY X-MP central processing unit (CPU).

CPU COMPUTATION SECTION

Each CPU is an identical, independent computation section consisting of operating registers, functional units, and an instruction control network (refer to figure 2-1, a fold-out block diagram of a typical CRAY X-MP computation section). The computation section performs three types of processing: address, scalar, and vector. Address processing operates on internal control information, such as addresses and indexes, while scalar and vector processing are performed on data. The instruction control network makes all decisions related to instruction issue as well as coordinating the three types of processing. Each of the processing modes has its associated registers and functional units.

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

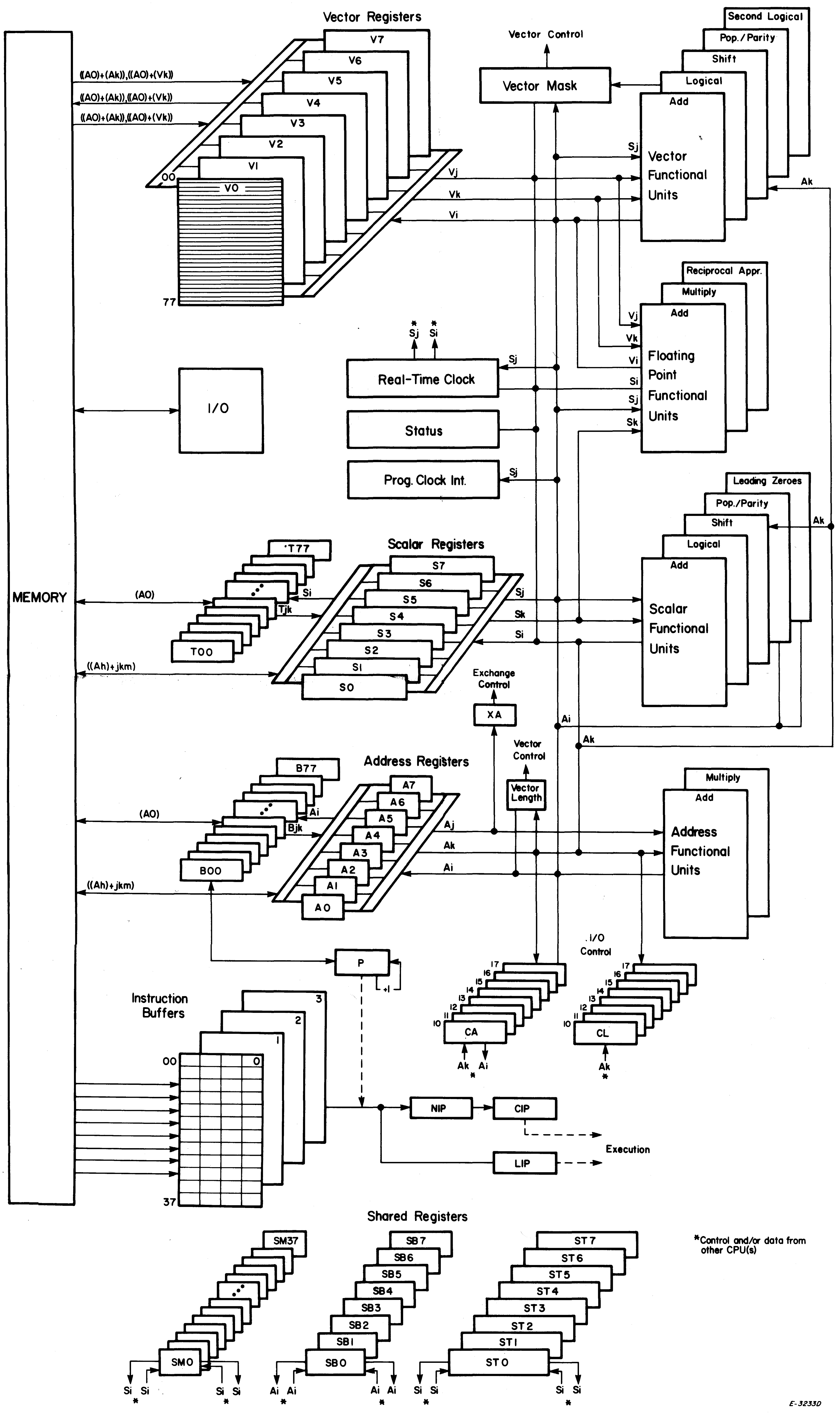
Data flow in a computation section is from Central Memory to registers and from registers to functional units. Results flow from functional units to registers and from registers to Central Memory or back to functional units. Data flows along either the scalar or vector path, depending on the processing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

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 that uses a multiple instruction sequence.

Figure 2-1. CRAY X-MP Block Diagram

CRAY X-MP BLOCK DIAGRAM



Integer or fixed-point operations are integer addition, integer subtraction, and integer multiplication. An 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 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 lets you 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 on the diagonal as well as allowing 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.

PROGRAMMABLE CLOCK

A programmable clock is a standard feature of the CRAY X-MP computer systems. This clock allows the operating system to force interrupts to occur at a particular time or frequency. The clock frequency/intervals vary for different models of the CRAY X-MP computer systems.

REGISTERS

A CPU has three primary and two intermediate sets of registers. The primary sets of registers are the address (A), scalar (S), and vector (V) registers. These registers are considered primary because functional units can access them directly.

For the A and S registers, an intermediate level of registers exists. These registers are not accessible to the functional units but act as a buffer for the primary registers. Block transfers of consecutive locations 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 intermediate address (B) registers. The intermediate registers that support S registers are referred to as scalar-save (T) registers.

Address registers

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

Scalar registers

The S registers are the principal scalar registers for a 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.

The T registers are used as intermediate storage for the S registers. Data is transferred between T and S registers and between T registers and Central Memory.

Vector registers

The V registers are used for vector operations. Successive elements from a V register enter a functional unit in successive clock periods (CPs). The effective length of a vector register for any operation is controlled by the program-selectable Vector Length (VL) register. The Vector Mask (VM) register allows for the logical selection of particular elements of a vector.

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. Most functional units can operate simultaneously. Functional units have independent logic except for the following:

- The Reciprocal Approximation and Vector Population Count units share some logic.

- The Floating-point Multiply and Second Vector Logical units share input and output paths.

- The Scalar Add and Scalar Shift units share output paths.

The preceding cases of shared logic can cause a hold issue condition.

All functional units perform algorithms in a fixed amount of time; delays are impossible once the operands have been delivered to the unit. Functional units are fully segmented. This means that 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.

The functional units identified are arbitrarily described in four groups: address, scalar, vector, and floating-point. Each of the first three groups function 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.

A functional unit engaged in a vector operation remains busy until it is finished. In this state, the functional unit is reserved. Other instructions requiring the same functional unit are not issued until the previous operation is complete. Only one functional unit of each type is available to the vector instruction hardware (with the exception of the Second Vector Logical unit). When the vector operation completes, the reservation is dropped and the functional unit is then available for another operation.

Address functional units

Address functional units perform integer arithmetic on operands obtained from A registers and deliver the results to an A register (refer to section 2 for an explanation of integer arithmetic). The arithmetic is twos complement.

Address Add functional unit - The Address Add functional unit performs integer addition and subtraction; addition and subtraction are performed in a similar manner. The unit detects no overflow.

Address Multiply functional unit - The Address Multiply functional unit forms an integer product from two operands. No rounding is performed.

Scalar functional units

Scalar functional units perform operations on operands obtained from S registers and usually deliver the results to an S register (refer to section 2 for an explanation of integer arithmetic). The exception is the Population/Parity/Leading Zero Count functional unit, which delivers its result to an A register.

The Scalar Add, Scalar Shift, Scalar Logical, and Scalar Population/Parity/Leading Zero functional units are exclusively associated with scalar operations and are described here. Three additional functional units are used for both scalar and vector operations. They are described in the subsection on Floating-point Functional Units.

Scalar Add functional unit - The Scalar Add functional unit performs integer addition and subtraction; addition and subtraction are performed in a similar manner. The unit detects no overflow.

Scalar Shift functional unit - The Scalar Shift functional unit shifts the entire contents of an S register or shifts the contents of two concatenated S registers into a single resultant S register.

Scalar Logical functional unit - The Scalar Logical functional unit performs bit-by-bit manipulation of quantities obtained from S registers.

Scalar Population/Parity/Leading Zero functional unit - This functional unit can count the number of bits in an S register having a value of 1 in the operand and returns a 1-bit population parity count (even parity). It also can count the number of bits of 0 preceding a 1 bit in the operand from left to right; the operand is obtained from an S register and the 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 operands or operand pairs to be processed by a functional unit.

The functional units described in this subsection are exclusively associated with vector operations. Three functional units are associated with both vector operations and scalar operations and are described in the Floating-point Functional Units subsection. 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 Add functional unit - The Vector Add functional unit performs integer addition and subtraction for a vector operation and delivers the results to elements of a V register. Addition and subtraction are performed in a similar manner. The unit detects no overflow.

Vector Shift functional unit - The Vector Shift functional unit shifts the entire contents of a V register element or the 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.

Full Vector Logical functional unit - The Full Vector Logical functional unit performs a bit-by-bit manipulation of specified quantities for specific instructions. The Full Vector Logical functional unit also performs vector register merge, compressed index, and logical operations associated with the vector mask instruction.

Second Vector Logical functional unit - The Second Vector Logical functional unit performs a bit-by-bit manipulation of the specified quantities for specific instructions. 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 are issued there first if possible. If the unit is busy, issue is then attempted to the Full Vector Logical unit. When both units are busy, the first unit to clear is selected for issue. Instructions are issued 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 Floating-point Multiply functional unit reservation).

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. The vector population count instruction delivers the total population count to elements of the destination V register while the vector population count parity instruction delivers the low-order bit of the count to the destination V register for even parity.

Floating-point functional units

Three floating-point functional units perform floating-point arithmetic for scalar and vector operations. When a scalar instruction is executed, operands are obtained from S register(s) and results are delivered to an S register. For 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, which requires only one input operand.

Floating-point Add functional unit - The Floating-point Add functional unit performs addition or subtraction of operands in floating-point format. The final result is normalized even when operands are unnormalized. Normalized numbers are explained later in this section.

Floating-point Multiply functional unit - The Floating-point Multiply functional unit executes instructions that provide for full- and half-precision multiplication of 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. Normalized numbers are explained later in this section.

Out-of-range exponents are detected. 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.

Reciprocal Approximation functional unit - The Reciprocal Approximation functional unit finds the approximate reciprocal of an operand in floating-point format. 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. Normalized numbers are explained later in this section.

CPU CONTROL SECTION

The CPU's control section contains instruction buffers and registers for instruction issue and control. The following subsections describe the registers and buffers.

Instruction buffers

Each CPU has four instruction buffers; each holds 128 consecutive instruction parcels. Instruction parcels are held in the buffers before being delivered to the NIP or LIP registers.

Program Address (P) register

The P register indicates the next parcel of program code to enter the Next Instruction Parcel (NIP) register. New data enters the P register on an instruction branch or on an exchange sequence. The contents of P are then advanced sequentially until the next branch or exchange sequence.

Next Instruction Parcel (NIP) register

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

Current Instruction Parcel (CIP) and Lower Instruction Parcel (LIP) registers

The CIP register holds the instruction waiting to be issued. If the instruction is a 2-parcel instruction, the CIP register holds the first parcel of the instruction and the LIP register holds the second parcel. Instruction formats are explained in section 3.

EXCHANGE SEQUENCE

A 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.

Instruction issue is terminated by the hardware upon detection of an interrupt condition. All memory bank and functional unit activity is allowed to finish. To switch execution in order to handle the interrupt, the CPU executes the exchange sequence. This causes program parameters for the next program to be exchanged with current information in the operating registers. Each program in the system has its associated 16-word Exchange Package, which contains the parameters used in its execution sequence. Only the A and S registers are saved in a program's Exchange Package; the contents of the B, T, V, VM, Shared Address (SB), Shared Scalar (ST), and Semaphore (SM) registers must be saved by software.

Exchange sequences may be initiated by a deadstart sequence or program exit, voluntarily by the software, or automatically upon occurrence of an interrupt condition.

EXCHANGE PACKAGE

The Exchange Package is a block of sixteen 64-bit words in memory associated with a particular program. The Exchange Package contains the basic parameters necessary to provide continuity from one execution interval for the program to the next. 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 following subsections define the contents of the Exchange Package.

Processor number (PN)

The state of the PN in the Exchange Package indicates in which CPU the Exchange Package executed. This value is not read into the CPU; it is a constant inserted only into a package being stored. In single-processor models, this value is always 0.

Memory error data

Error data, consisting of four fields of information, appears in the Exchange Package if the interrupt on correctable memory error bit is set and a correctable memory error is encountered or if the interrupt on uncorrectable memory error bit is set and an uncorrectable memory error is detected.†

Memory error data fields are described below.

Field	Description
Error type (E)	The type of memory error encountered, correctable or uncorrectable, is indicated in this word of the Exchange Package.
Syndrome (S)	The S bits used in defining a memory data error are returned in this word of the Exchange Package.
Read mode (R)	The type of read mode in progress when a memory data error occurred is indicated in these bits of the Exchange Package.
Read address (CSB)	The chip select, bank, and section bits where a memory data error occurred are defined in this word.

Program Address (P) register

The P register contents (address of first program instruction not yet issued) are stored in this word of the Exchange Package. The instruction at this location is the first instruction to be issued when this program begins again.

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

Instruction Base Address (IBA) register

The IBA register holds the base address of the user's instruction field. A user instruction can be executed only 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.

Instruction Limit Address (ILA) register

The ILA register holds the limit address of the user's instruction field. A user instruction can be executed only by the CPU if the absolute address at which the instruction 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.

Mode (M) register

The M register contains part of the Exchange Package for a currently active program. The bits of the M register that are set selectively during an exchange sequence are defined as follows:

- Waiting for Semaphore (WS) flag; when set, the CPU exchanged when a test and set instruction was holding in the CIP register.
- Floating-point Error Status (FPS) flag; when set, a floating-point error has occurred regardless of the state of the Floating-point Error Mode flag.
- Bidirectional Memory Mode (BDM) flag; when set, block reads and writes can operate concurrently.
- Selected for External Interrupts (SEI) flag; when set, this CPU is preferred for I/O interrupts.†
- Interrupt Monitor Mode (IMM) flag; when set, it enables all interrupts in monitor mode except PC, MCU, I/O, NEX, and ICP.
- Operand Range Error Mode (IOR) flag; when set, it enables interrupts on operand address range errors.
- Correctable Memory Error Mode (ICM) flag; when set, it enables interrupts on correctable memory data errors.

† Not available on single-processor systems

- Floating-point Error Mode (IFP) flag; when set, it enables interrupts on floating-point errors.

- Uncorrectable Memory Error Mode (IUM) flag; when set, it enables interrupts on uncorrectable memory data errors.

- Monitor Mode (MM) flag; when set, it inhibits all interrupts except memory errors, normal exit, and error exit.

Vector Not Used (VNU) position

The state of the VNU position in the Exchange Package indicates whether several specific vector instructions were issued during the execution intervals. If none of the instructions were issued, the bit is set. If one or more of the instructions were issued, the bit is not set.

Enable Second Vector Logical (ESVL) position

The contents of the ESVL position in the Exchange Package indicate whether or not the Second Vector Logical unit can be used. If set, the Second Vector Logical unit may be used. If clear, the Second Vector Logical unit cannot be used; only the Full Vector Logical unit may be used.

Flag (F) register

The F register contains part of the Exchange Package for the currently active program. This register contains 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 contains the following flags:

- Interrupt from Internal CPU (ICP) flag; set when the other CPU issues instruction 0014*j*1.[†]

- Deadlock (DL) flag; set when all CPU(s) in a cluster are holding issue on a test and set instruction.

[†] Not available on single-processor systems

- Programmable Clock Interrupt (PCI) flag; set when the interrupt countdown counter in the programmable clock equals 0.
- MCU Interrupt (MCU) flag; set when the MIOP sends this signal.
- 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.
- 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.
- 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.
- 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.
- I/O Interrupt (IOI) flag; set when a 6-Mbyte/s channel or the 1000-Mbyte/s (100 Mbyte/s channel in single-procession models) channel completes a transfer.
- Error Exit (EEX) flag; if not in MM, set by an error exit instruction.
- Normal Exit (NEX) flag; if not in MM and IMM, set by a normal exit instruction.

Exchange Address (XA) register

The 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 ($10,000_8$) words of memory. When an execution interval terminates, the exchange sequence exchanges the contents of the registers with the contents of the Exchange Package at the beginning address (XA) in memory.

Vector Length (VL) register

The VL register specifies the length of all vector operations performed by vector instructions and the length of the vectors held by the V registers.

__Enhanced Addressing Mode (EAM) position__†

The contents of the EAM position in the Exchange Package indicates whether or not address extension occurs for address calculations.

__Data Base Address (DBA) register__

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

__Program State (PS) register__

The state of the PS register is manipulated by the operating system to represent different program states in the CPUs concurrently processing a single program.

__Cluster Number (CLN) register__

The 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, the CPU does not have access to any SB, ST, or SM register. The CLN register's contents in all CPUs are also used to determine the condition necessary for a deadlock interrupt.

__Data Limit Address (DLA) register__

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

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.

† Not available on all systems

A registers

The current contents of all A registers are stored in a portion of the Exchange Package.

S registers

The current contents of all S registers are stored in a portion of the Exchange Package.

CPU INTERCOMMUNICATION

The inter-CPU communication section of the mainframe contains clusters of shared registers for interprocessor communication and synchronization. Each cluster consists of Shared Address (SB), Shared Scalar (ST), and Semaphore (SM) registers.

In multiprocessor systems, the SB and ST registers are used for passing address and scalar information from one CPU to another, while the SM registers are used for control between CPUs. In single-processor systems, the CPU can use the SB and ST registers, while the SM registers can be used by the CPU for storage and control.

Each CPU's Cluster Number (CLN) register determines which set of shared registers is accessed by a CPU (clustering). The cluster may be accessed by any processor to which it is allocated in either user or system (monitor) mode. Any processor in monitor mode can interrupt any other and cause it to switch from user to monitor mode. Additionally, each processor in a cluster can asynchronously perform scalar or vector operations dictated by user programs. The hardware also provides built-in detection of system deadlock within the cluster.

REAL-TIME CLOCK

In multiprocessor systems, the mainframe contains one real-time clock (RTC), which is shared by all the CPUs. This counter is 64-bits and advances one count each CP. Because the clock advances synchronously with program execution, it can be used to time the program to an exact number of CPs. In single-processor systems, the RTC is not shared but works the same way.

ARITHMETIC OPERATIONS

Functional units in a 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 shown in figure 2-2. The Address Add and Address Multiply functional units perform 24-bit arithmetic. The Scalar Add and the Vector Add functional units perform 64-bit arithmetic.

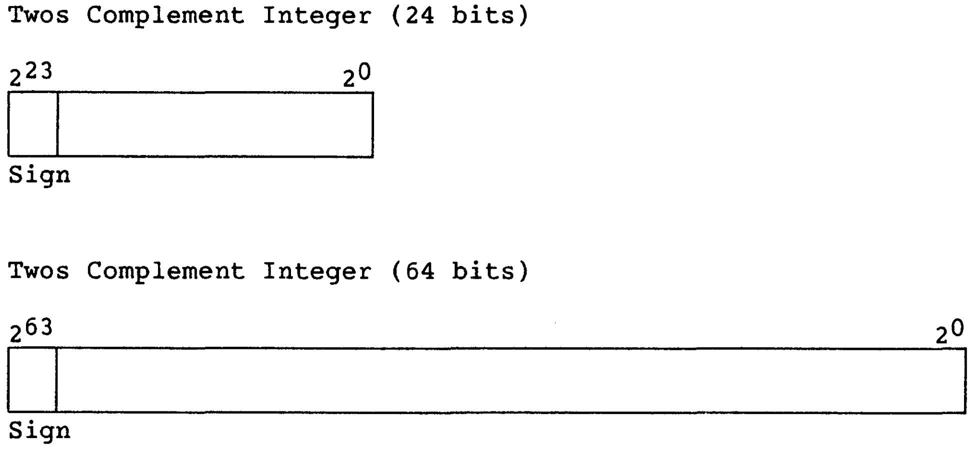

Twos Complement Integer (24 bits)
2^{23}
2^0
Sign
Twos Complement Integer (64 bits)
2^{63}
2^0
Sign

Figure 2-2. Integer Data Formats

Multiplication of two scalar (64-bit) integer operands is accomplished by using the floating-point multiply instruction and one of the two methods that follow. 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 if each is shifted 24 bits to the left before the multiply operation. (The Floating-point Multiply functional unit recognizes the conditions in which 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 as 0. Refer to figure 2-6. If the operand coefficients were generated by a means other than shifting so the low-order 24 bits would be nonzero, the low-order 48 bits of the product could be nonzero, and the high-order 48 bits (the return part) could be one larger than expected because truncation compensation constant is always added during a multiply.

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

Division is done by algorithm; the particular algorithm used depends on the number of bits in the quotient. The quickest and most frequently used method is to convert the numbers to floating-point format and then use the floating-point functional units.

Floating-point arithmetic

Floating-point numbers are represented in a standard format throughout the CPU. This format is a packed representation of a binary coefficient and an exponent (power of 2). 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 2-3. Because the coefficient is of signed magnitude, it is not complemented for negative values.

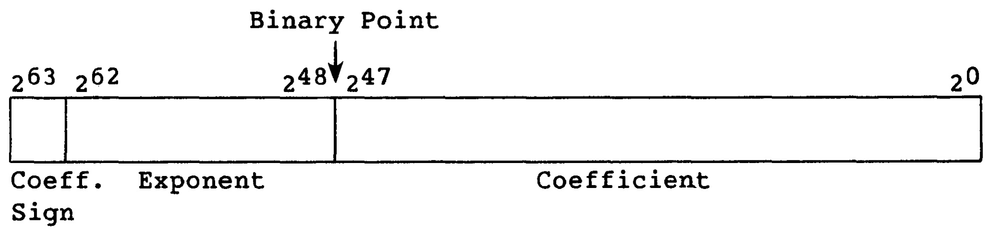


Figure 2-3. Floating-point Data Format

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} 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^{+2466}.

Figure 2-4 and the following steps show 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

```
-40000
------
     1
```

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

   $2^1 \times 0.4_8 = 1.0$

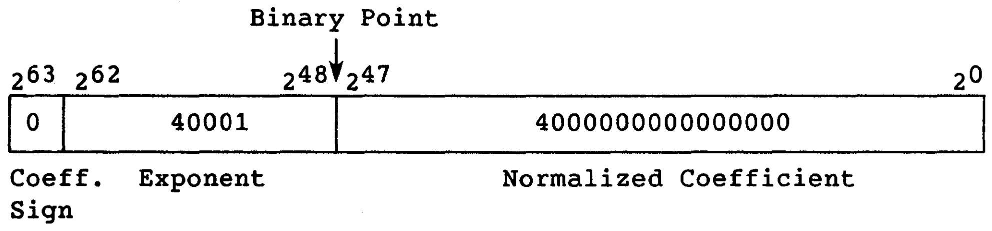


Figure 2-4. Internal Representation of Floating-point Number (Octal)

A 0 value or an underflow result is not biased and is represented as a word of all zeros.

A negative 0 is not generated by any floating-point functional unit, except in the case in which a negative 0 is one operand going into the Floating-point Multiply or Floating-point Add 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 that the coefficient has been shifted as far left as possible and that the exponent has been 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 and a 48-bit integer word into the coefficient the result should be normalized before being used in a floating-point operation. Normalization is accomplished by adding the unnormalized floating-point operand to 0. Since S0 provides a 64-bit zero when used in the S*j* field of an instruction, an operand in S*k* is normalized with the 062*i*0*k* instruction. S*i*, which can be S*k*, contains the normalized result.

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

Floating-point range errors

Overflow of the floating-point range is indicated by an exponent value of 60000_8 for Floating-point Add or Multiply and 60002_8 for Floating-point Reciprocal 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.

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

```
 60000.4xxxxxxxxxxxxxxx   Range Error
+57777.4xxxxxxxxxxxxxxx
 60000.6xxxxxxxxxxxxxxx   Result Register
```

NOTE

If a floating-point add or subtract generates an exponent of less than 20000_8 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 60000_8 or greater) or if the final sum or difference is out of bounds (exponent of 60000_8 or greater), the exponent is set to 60000_8, 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 the way they are handled can be determined by using the exponent matrix shown in figure 2-5. 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.

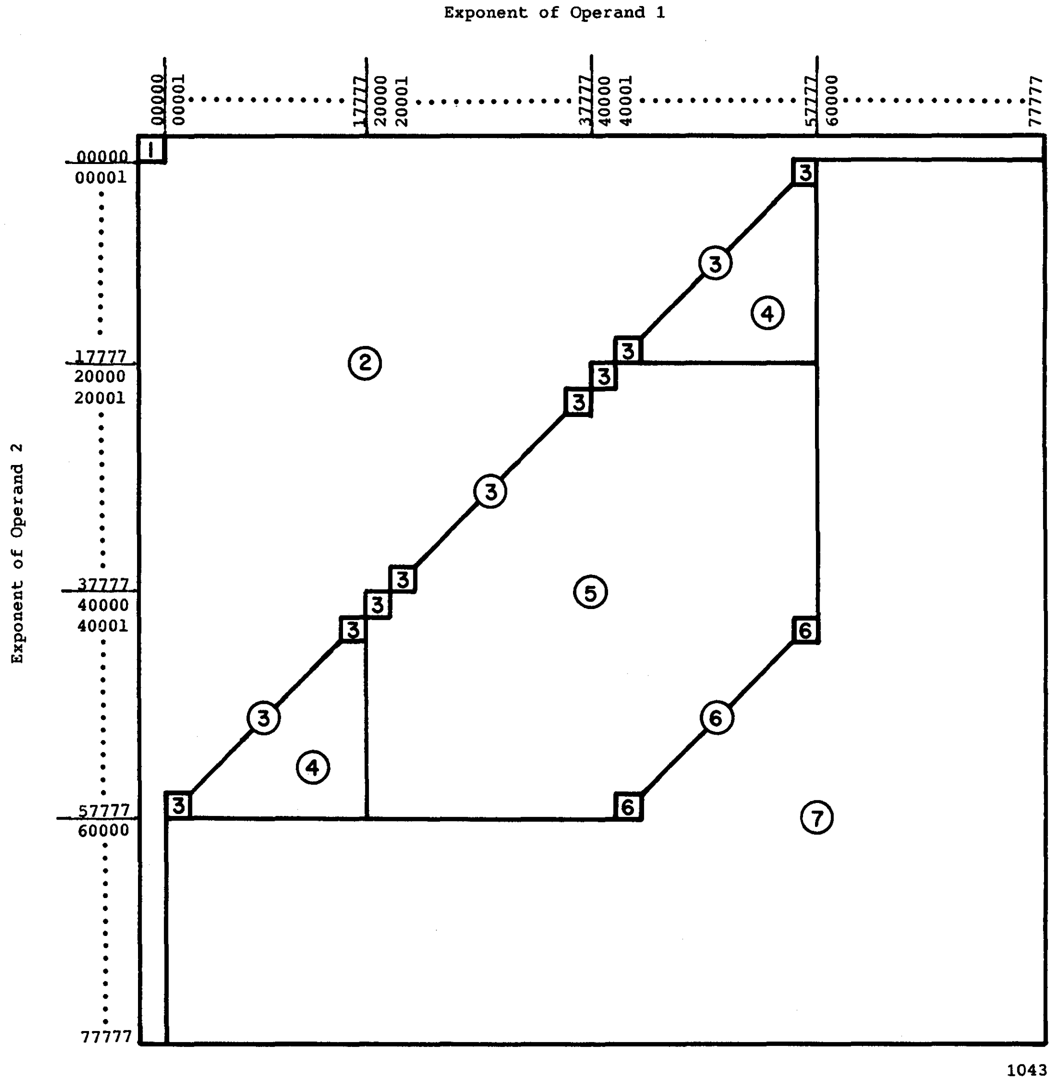


Figure 2-5. Exponent Matrix for Floating-point Multiply Unit

Zone	Description
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 the 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 is within the range 20000_8 to 57777_8.
⑤	This is the normal operand range, and normal results are produced.
⑥	Overflow is flagged on this boundary. If a normalized shift is required, the value should be within bounds with a 57777_8 exponent. Because overflow is detected, however, a 60000_8 is inserted in the product as the final exponent when the exponent for the unnormalized shift condition is used.
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 2-6, 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 2-2) 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 2-6 have any 1 bits, the product might be 1 (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 2-6) does not need to be the same for both operands. To get a correct product, the only requirement is that the sum of the number of bits in the shaded area be 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 (assuming that the operand binary points are at the left boundary of the shaded areas).

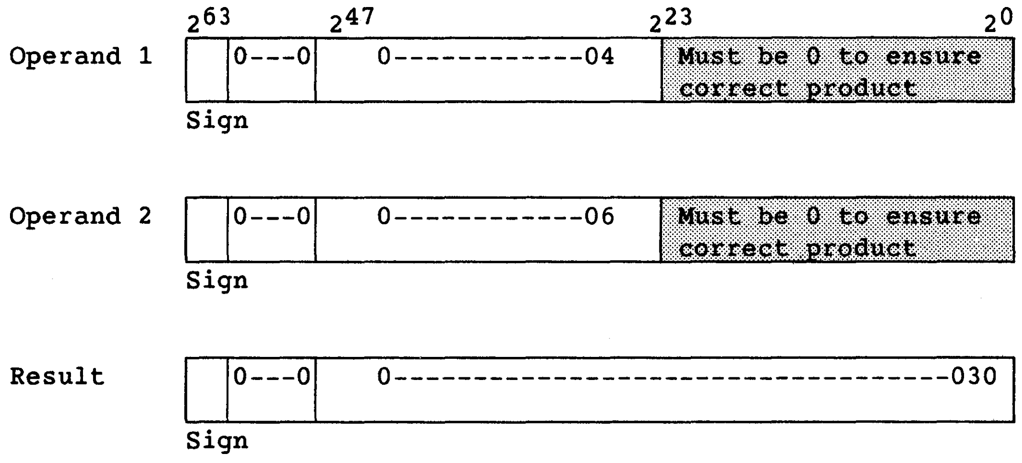


Figure 2-6. Integer Multiply in Floating-point Multiply Functional Unit

<u>Floating-point Reciprocal Approximation functional unit</u> - 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 60002_8 causes a floating-point range error. The error flag is set and an exponent of 60000_8 and the computed coefficient with 2^{47} set to 0 are sent to the result register.

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 2-7). Trial subtraction of the exponents selects the operand to be shifted down for aligning the operands. The larger exponent operand carries the sign. The coefficient of the number with the smaller exponent is shifted right to align with the coefficient of the number with the larger exponent. Bits shifted out of the register are lost; no 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.

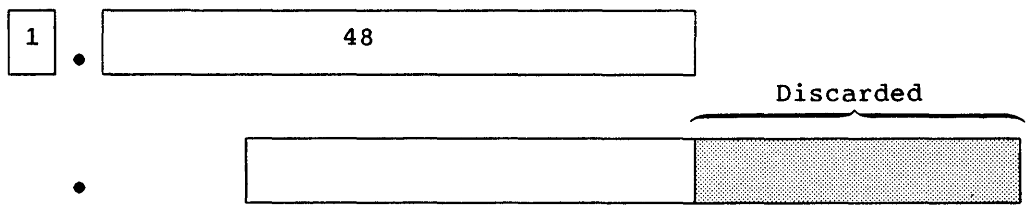


Figure 2-7. 49-bit Floating-point Addition

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 the functional unit. If the coefficients are both normalized, 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 functional unit 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 x 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 1 smaller than expected. With compensation, the results range from 1 too large to 1 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 product been present. The multiplication is commutative; that is, A times B equals B times A.

Rounding is optional where truncation compensation is not used. The rounding method used adds a constant so that it is 50 percent high (0.25 x 2^{-48}; high) 38 percent of the time and 25 percent low (0.125 x 2^{-48}; low) 62 percent of the time, resulting in a near-zero average rounding error. In a full-precision rounded multiply, 2 round bits are entered into the functional unit at bit positions 2^{-50} and 2^{-51} and allowed to propagate.

For a half-precision multiply, round bits are entered into the functional unit 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 variations due to this truncation and rounding are in the following range:

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

or

-8.17×10^{-16} to $+20.25 \times 10^{-16}$

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

-0.5×2^{-48} 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, $F(x) = 0$, for which $F(x)$ must be twice differentiable with a continuous second derivative. The method requires an initial approximation (guess), x_0, sufficiently close to the true root, x_t, being sought (refer to figure 2-8). 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.

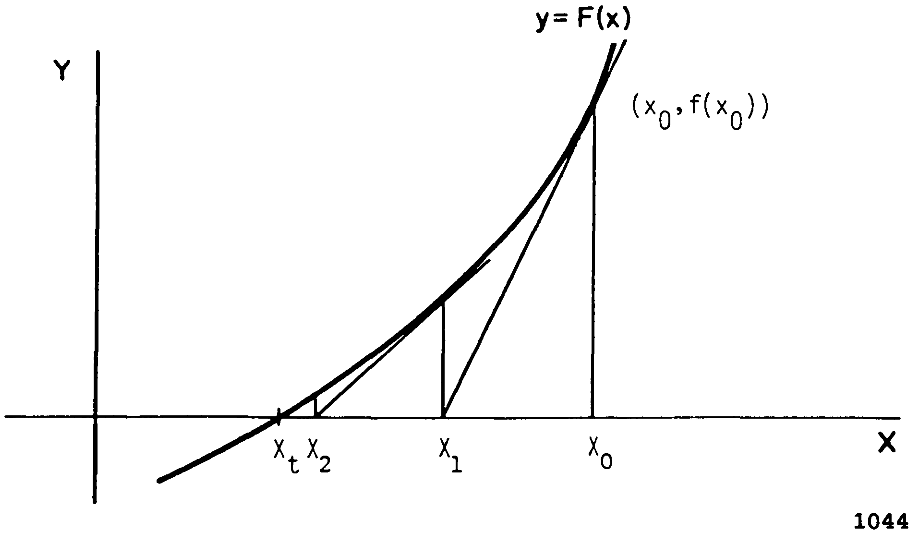


Figure 2-8. Newton's Method

Derivation of the division algorithm

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

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

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

For any point x_i near x_t,

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

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 because $x_i + (x_t - x_i)$ equals x_t. Solving approximation (1) for $(x_t - x_i)$ gives the following:

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

that is, $-\dfrac{F(x_i)}{F'(x_t)}$ 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:

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

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 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:

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

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

and applying equation (1),

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

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

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

$$x_2 = 2x_1 - x_1^2 B = x_1(2 -{}_1x\ B).$$

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 lookup provides an initial guess, x_0, to start the process.

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

The 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:

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

Operation	Performed By
S3 = 1/S2	Reciprocal Approximation functional unit
S6 = S1 * S3	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 division is as follows:

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

A scalar quotient is computed in a certain number of CPs because 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 because operations 1 and 2 are chained together. This overlaps one of the multiply operations. (A vector time is 1 CP for each element in the vector.)

****

CAUTION

The coefficient of the reciprocal produced by the alternative method can be different by as much as 2 x 2^{-48} 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 that the reciprocals of numbers are always the same.

****

LOGICAL OPERATIONS

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

A logical product is the AND function:

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

A logical sum is the inclusive OR function:

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

A logical difference is the exclusive OR function:

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

A logical equivalence is the exclusive NOR function:

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

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

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

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

CENTRAL MEMORY

The CRAY X-MP Central Memory is shared by the CPUs on multiprocessor systems and is arranged in interleaved banks. The interleaved memory banks enable extremely high transfer rates through the I/O section and provide low read/write times for vector processing. All banks can be accessed independently and in parallel during each machine clock period.

Each CRAY X-MP processor has four parallel memory ports: three for vector and scalar operations and one for I/O. The multiport memory has built-in conflict resolution hardware to minimize delays and maintain the integrity of all memory references to the same bank at the same time.

All CRAY X-MP models provide a flexible hardware chaining mechanism for vector processing. This feature enables a result vector to be used at any time as an operand in a succeeding operation. Also, vector chaining to and from memory is possible.

In addition, the CRAY X-MP computer system provides hardware support for vector conditionals. Gather/scatter operations (chainable from other vector memory fetches and stores) and compressed-index generation facilitate and speed execution of various conditional vector operations realized from ordinary user programs. All models allow execution of two vector logical operations of the same type at the same time.

Central Memory comes in various sizes; refer to section 4 for the different memory configurations.

INPUT/OUTPUT SECTION

The I/O section is shared by the CPUs in multiprocessor systems and may be equipped with a variety of high-performance channels for communicating with the mainframe, the IOS, and the SSD. The latter two devices are high-speed data transfer devices designed to support CRAY X-MP processing speeds. Refer to section 4 for information on channel types and transfer rates for your specific CRAY X-MP computer system.

CRAY X-MP SYMBOLIC MACHINE INSTRUCTIONS 3

Each CRAY X-MP mainframe machine instruction can be represented symbolically in Cray Assembly Language (CAL). This section provides information on the symbolic machine instructions used with the CRAY X-MP computer systems.

This section provides information on symbolic machine instruction format for a 1-parcel (16-bit) instruction or a 2-parcel (32-bit) instruction. It also describes special register values that may be referenced by the instructions and the symbolic notation used for coding the machine instructions.

Detailed information on the CAL instructions that operate on the CRAY X-MP computer systems is provided in the Machine Instruction Description subsection. Each machine instruction begins with boxed information consisting of the CAL syntax format, an operand if required, a brief description of each instruction, and the machine instruction.

Following the boxed information is a detailed description of the instruction and an example.

The Symbolic Instruction Summary section provides a summary of functional units and the symbolic machine instructions; the Functional Instruction Summary section lists the instructions by function.

INSTRUCTION SYNTAX

The assembler identifies a symbolic instruction according to its syntax and generates a corresponding binary machine code. An instruction is generated in the assembly section in use when the instruction is interpreted.

This section describes the format of symbolic machine instructions, special register values, and notation used for coding symbolic machine instructions for CAL Assembler Version 2 on a CRAY X-MP computer system.

INSTRUCTION FORMAT

Each instruction is either a 1-parcel (16-bit) instruction or a 2-parcel (32-bit) instruction. Instructions are packed 4 parcels per word.

Parcels 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 parcel 3 of a word ends in parcel 0 of the next word. No padding to word boundaries is required. Figure 3-1 illustrates the general form of instructions.

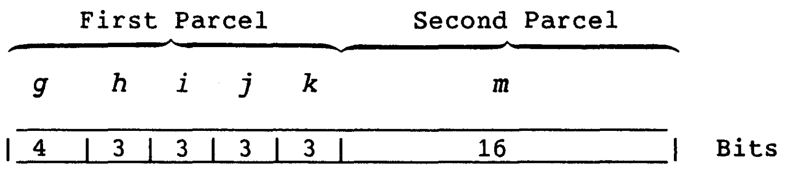


Figure 3-1. General Form for Instructions

Four variations of this general format use the fields differently. The formats of the following variations are described in this section:

- 1-parcel instruction format with discrete *j* and *k* fields
- 1-parcel instruction format with combined *j* and *k* fields
- 2-parcel instruction format with combined *j, k,* and *m* fields
- 2-parcel instruction format with combined *i, j, k,* and *m* fields

1-parcel instruction format with discrete *j* and *k* fields

The most common of the 1-parcel instruction formats uses the *i, j,* and *k* fields as individual designators for operand and result registers (see figure 3-2). The *g* and *h* fields define the operation code. The *i* field designates a result register and the *j* and *k* fields designate operand registers. Some instructions ignore one or more of the *i, j,* and *k* fields. The following types of instructions use this format:

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

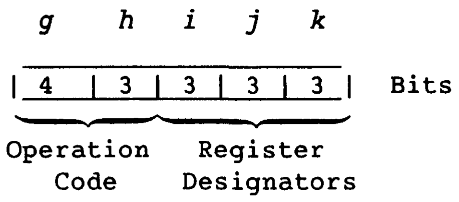


Figure 3-2. 1-parcel Instruction Format with Discrete *j* and *k* Fields

<u>1-parcel instruction format with combined *j* and *k* fields</u>

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

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

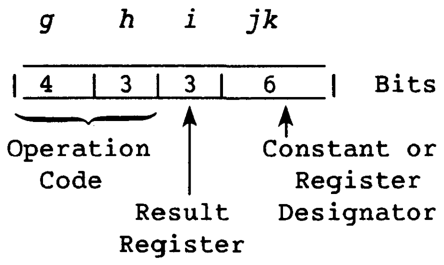


Figure 3-3. 1-parcel Instruction Format with Combined *j* and *k* Fields

<u>2-parcel instruction format with combined *j, k,* and *m* fields</u>

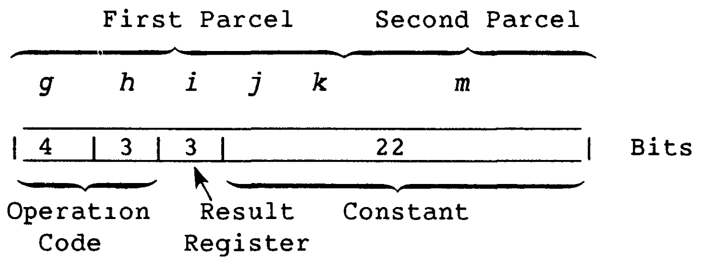
The instruction type for a 22-bit immediate constant uses the combined *j, k,* and *m* fields to hold the constant. The 7-bit *g* and *h* fields contain an operation code, and the 3-bit *i* field designates a result register. The instruction type using this format transfers the 22-bit *jkm* constant 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 Special Register Values subsection.)

Figure 3-4 shows the two general applications for the 2-parcel instruction format with combined *j, k,* and *m* fields.

NOTE

When an immediate constant with both relocatable and parcel attributes is used, the result of the relocation will be incorrect if the loader-determined actual address (within the user's field length) is greater than 1,048,575. This is because the resulting relocated value has more than 22 significant bits. A CAL caution message is issued if this occurs. The exception to this is when A*h exp* executes on a CRAY X-MP computer system model 48 or 416.



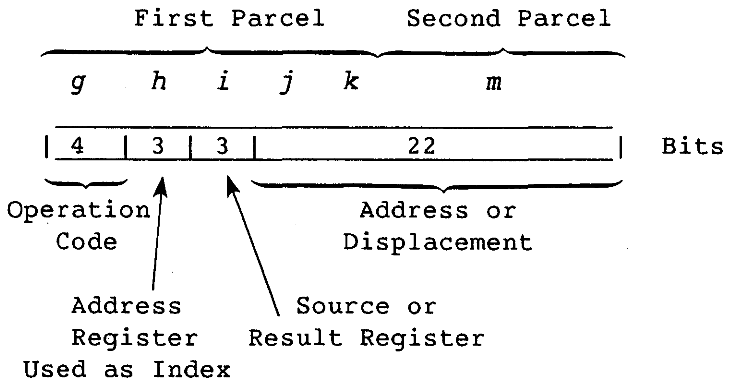


Figure 3-4. 2-parcel Instruction Format with Combined *j, k,* and *m* Fields

2-parcel instruction format with combined *i, j, k,* and *m* fields

The 2-parcel branch instruction type uses the combined *i, j, k,* and *m* fields to contain a 24-bit address that allows branching to an instruction parcel (see figure 3-5). A 7-bit operation code (*gh*) is followed by an *ijkm* field. The high-order bit of the *i* field is equal to 0.

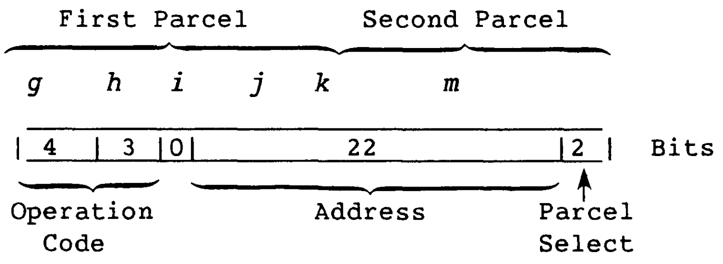


Figure 3-5. 2-parcel Instruction Format with Combined *i, j, k,* and *m* Fields

The 2-parcel instruction type for a 24-bit immediate constant (figure 3-6) 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.

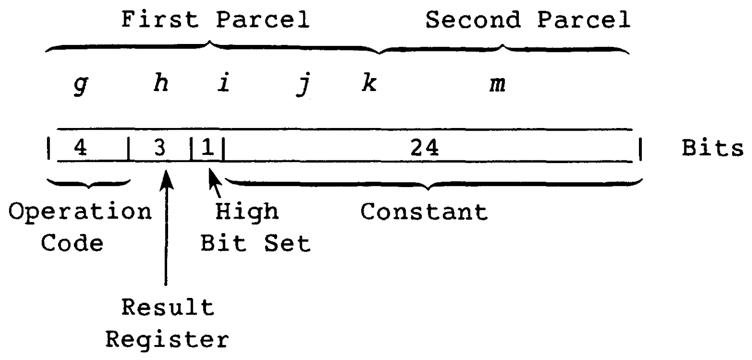


Figure 3-6. 2-parcel Instruction Format for a 24-bit Immediate Constant with Combined *i, j, k,* and *m* Fields

SPECIAL REGISTER VALUES

If the S0 and A0 registers are referenced in the *j* or *k* fields of certain instructions, the contents of the respective register are not used; instead, a special operand is generated. The special operand is available regardless of existing A0 or S0 reservations (and in this case is 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. CAL issues a caution-level error message for A0 or S0 when 0 does not apply to the *i* field. Table 3-1 shows the special register values.

Table 3-1. 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}

SYMBOLIC NOTATION

The following information describes the notation used for coding symbolic machine instructions. CAL contains two syntax forms: general and special.

General syntax

Register designators and the location, result, operand, and comment fields have the following general syntax requirements.

Register designators - A, B, SB, S, T, ST, SM, and V registers can be referenced with numeric or symbolic designators. The symbolic designators can be entered in uppercase, lowercase, or any mixture of upper and lowercase.

In symbolic notation, the *h*, *i*, *j*, and *k* designators indicate the field of the machine instruction into which the register designator constant or symbol value is placed. An expression (*exp*) occupies the *jk*, *ijk*, *jkm*, or *ijkm* field depending on the operation code and magnitude of the expression value. Supporting registers have the following designators:

Designator	Register
CA	Current Address
CL	Channel Limit
CI	Channel Interrupt flag
CE	Channel Error flag
RT	Real-time Clock
MC	Master Clear
SB	Sign Bit (S*k*, with *k*=0)
SM	Semaphore
VL	Vector Length
VM	Vector Mask
XA	Exchange Address

Location field - The location field of a symbolic instruction optionally contains a symbol. When a symbol is present, it is assigned a parcel address as indicated by the current value of the location counter after any required force to parcel boundary occurs.

Result field - The result field of a symbolic machine instruction can consist of one, two, or three subfields separated by commas. A subfield can be null or it can contain a register designator or an expression. The expression specifies a memory address that indicates the register or memory location to receive the results of the operation. The result field may contain a mnemonic indicating the function being performed (for example, J for jump or ex for exit). The mnemonics are case sensitive and must be entered in either all uppercase or all lowercase letters; they cannot be mixed. For example, EX is a valid mnemonic for exit, while Ex is not.

Operand field - The operand field of a symbolic machine instruction consists of no subfield or one, two, or three subfields separated by commas. A subfield can be null, contain an expression (with no register designators), or consist of register designators and operators.

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

Character	Operation
+	Arithmetic sum of specified registers
-	Arithmetic difference of specified registers
*	Arithmetic product of specified registers
/	Reciprocal of approximation
#	Use ones complement
>	Shift value or form mask from left to right
<	Shift value or form mask from right to left
&	Logical product of specified registers
!	Logical sum of specified registers
\	Logical difference of specified registers

In some instructions, register designators are prefixed by the following letters which have special meaning to the assembler. These letters can be entered in either uppercase or lowercase (case insensitive).

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

Comment field - The comment field of the symbolic machine instructions begins in column 35. By convention, the comment should be preceded by a semicolon (;) in column 35, and a space.

Special syntax forms

The CAL instruction repertoire has been expanded for the convenience of programmers and to allow for special forms of symbolic instructions. Because of this expansion, certain Cray machine instructions can be generated from two or more different CAL instructions. For example, both of the following instructions generate instruction 002000, which causes a 1 to be entered into the VL register:

```
VL  A0
VL  1
```

The first instruction is the basic form of the Enter VL instruction, which takes advantage of the special case in which (A*k*)=1 if *k*=0; the second instruction is a special syntax form providing the programmer with a more convenient notation for the special case.

Any of the operations performed by special instructions can be performed by instructions in the basic set. Instructions having a special syntax form are identified as such in the instruction description found later in this section.

In several cases, a single syntax form of an instruction can result in any of several different machine instructions being generated. These cases provide for entering the value of an expression into an A register or into an S register or for shifting S register contents, the assembler determines which instruction to generate from characteristics of the expression.

MONITOR MODE INSTRUCTIONS

The monitor mode instructions (channel control, set real-time clock, and programmable clock interrupts) perform specialized functions that are useful to the operating system. These instructions execute only when the CPU is operating in monitor mode. If an instruction is executed while the CPU is not in the monitor mode, it is treated as a no-op.

MACHINE INSTRUCTION DESCRIPTIONS

This section contains detailed information about individual instructions or groups of related instructions. Each instruction begins with boxed information consisting of the CAL syntax format. It consists of a result field description, an operand field description, a brief description of each instruction, and the machine instruction (octal code sequence defined by the *gh* fields). The appearance of an *m* in a format description designates an instruction consisting of two parcels. An *x* in the format description signifies that the field containing the *x* is ignored during a CRAY X-MP computer system instruction execution. CAL inserts a 0 for each occurrence of *x*.

Following the boxed information is a detailed description of the instruction or instructions and an example using the instruction.

****

CAUTION

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

****

INSTRUCTION 000

Result	Operand	Description	Machine Instruction
ERR		Error exit	000000

The 000 instruction is treated as an error condition and an exchange sequence occurs. The contents of the instruction buffers are voided by the exchange sequence. 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 the results of previously issued instructions have arrived at the operating registers, an exchange occurs to the Exchange Package designated by the contents of the Exchange Address (XA) register. The program address stored in the Exchange Package on the terminating exchange sequence is advanced by 1 parcel from the address of the error exit instruction.

The error exit instruction is not generally used in program code. This instruction is used to halt execution of an incorrectly coded program that branches to an unused area of memory or into a data area.

The expression in the operand field is optional and has no effect on instruction execution; the low-order 9 bits of the expression value are placed in the *ijk* fields of the instruction.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
000000		ERR		

INSTRUCTION 0010

Result	Operand	Description	Machine Instruction
CA,A*j*†	A*k*	Set the Current Address (CA) register, for the channel indicated by (A*j*), to (A*k*) and activate the channel	0010*jk*
PASS††		Pass	001000

† Privileged to monitor mode
†† Special CAL syntax

The 0010*jk* instruction sets the Current Address (CA) register for the channel indicated by the contents of A*j* to the value specified in A*k*. It then activates the channel.

Before this instruction is issued, the Channel Limit (CL) register should be initialized. As the transfer progresses, the address in CA is increased. When the contents of CA equals the contents of CL, the transfer is complete for the words at the initial address in CA through 1 less than the address in CL.

If not in monitor mode or when the *j* designator is 0 or when the contents of A*j* is less than 2 or greater than 25, the instruction executes as a pass instruction. When the *k* designator is 0, CA is set to 1.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
001035		CA,A3	A5	
001000		PASS		

INSTRUCTION 0011

Result	Operand	Description	Machine Instruction
CL,A*j*†	A*k*	Set the channel (A*j*) limit address to (A*k*)	0011*jk*

† Privileged to monitor mode

The 0011*jk* instruction sets the Channel Limit (CL) register for the channel indicated by the contents of A*j* to the address specified in A*k*.

The instruction is usually issued before issuing the CA,A*j* A*k* instruction.

If not in monitor mode or when the *j* designator is 0 or when the contents of A*j* is less than 2 or greater than 25, the instruction is executed as a pass instruction. When the *k* designator is 0, CL is set to 1.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
001134		CL,A3	A4	

INSTRUCTION 0012

Result	Operand	Description	Machine Instruction
CI,A*j*†		Clear Channel (A*j*) Interrupt flag	0012*j*0
MC,A*j*		Clear Channel (A*j*) Interrupt flag and Error flag; set device master-clear (output channel); clear device ready-held (input channel).	0012*j*1

† Privileged to monitor mode

Instruction 0012*j*0 clears the Interrupt flag and Error flag for the channel indicated by the contents of A*j*.

If not in monitor mode or when the *j* designator is 0 or when the contents of A*j* is less than 2 or greater than 25, the instruction is executed as a pass instruction.

Instruction 0012*j*1 sets the device Master Clear. If (A*j*) represents an output channel, the master clear is set; if (A*j*) represents an input channel, the ready flag is cleared.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
001210		CI,A1		
001241		MC,A4		
001201		MC,A0		

INSTRUCTION 0013

Result	Operand	Description	Machine Instruction
XA†	A*j*	Enter XA register with (A*j*)	0013*j*0

† Privileged to monitor mode

The 0013*j*0 instruction transmits bits 12 through 19 of register A*j* to the Exchange Address (XA) register.

If the *j* designator is 0, the XA register is cleared.

A monitor program activates a user job by initializing the XA register to point to the user job's Exchange Package and then executing a normal exit (EX).

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
001350		XA	A5	

NOTE

Instruction 0013 is privileged to monitor mode and is treated as a pass instruction if the monitor mode bit is not set.

INSTRUCTION 0014

Result	Operand	Description	Machine Instruction
RT	S*j*	Enter RTC with (S*j*)	0014*j*0
SIPI†	*exp*	Set interprocessor interrupt request of CPU *exp*; $0 \le exp \le 3$	0014*j*1
SIPI† ††		Set interprocessor interrupt request	001401
CIPI†		Clear interprocessor interrupt	001402
CLN† †††	*exp*	Cluster number = *exp* where $0 \le exp \le 5$	0014*j*3
PCI¶	S*j*	Set program interrupt interval	0014*j*4
CCI¶		Clear clock interrupt	001405
ECI¶		Enable clock interrupts	001406
DCI¶		Disable clock interrupts	001407

† CRAY X-MP computer systems with multiple CPUs. This instruction is available when the numeric trait NUMCPUS, which is specified on the CPU parameter of the CAL invocation statement, is greater than 1.

†† Special CAL syntax

††† This instruction is available when the numeric trait NUMCLSTR, which is specified on the CPU parameter of the CAL invocation statement, is greater than zero.

¶ This instruction is available through the logical trait PC specified on the CPU parameter of the CAL invocation statement.

NOTE

Instruction 0014 is privileged to monitor mode and is treated as a pass instruction if the monitor mode bit is not set.

The 0014*j*0 instruction transmits the contents of register S*j* to the Real-time Clock register. When the *j* designator is 0, the Real-time Clock register is set to 0.

The 001401 and 001402 instructions handle interprocessor interrupt requests. When the *k* designator is 1, the instruction sets the internal CPU interrupt request in another CPU. If the other CPU is not in monitor mode, the ICP (Interrupt from Internal CPU) flag sets in the F register, causing an interrupt. The request remains until cleared by the receiving CPU.

When the *k* designator is 2, the instruction clears the internal CPU interrupt request set by another CPU.

The 0014*j*3 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 0 value to A*i* or S*i*).
CLN = 1	Cluster 1
CLN = 2	Cluster 2
CLN = 3	Cluster 3
CLN = 4	Cluster 4
CLN = 5	Cluster 5

Each of clusters 1, 2, 3, 4, and 5 has a separate set of SM, SB, and ST registers.

The 0014*j*4 instruction loads the low-order 32 bits from the S*j* register into the Interrupt Interval register (II) and the Interrupt Countdown counter (ICD). The Interrupt Countdown counter is a 32-bit counter that is decreased by 1 each clock period until the contents of the counter is equal to 0. At this time, the real-time clock (RTC) interrupt request is set. The counter is then set to the interval value held in the Interrupt Interval register and repeats the countdown to 0 cycle. When an RTC interrupt request is set, it remains set until a clear clock interrupt (CCI) instruction is executed.

The 001405 instruction clears an RTC interrupt.

The 001406 instruction enables RTC interrupts at a rate determined by the value in the Interrupt Interval (II) register.

The 001407 instruction disables RTC interrupts until an enable clock interrupt (ECI) instruction is executed.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
001420		RT	S2	; Set clock to
				; low-order 32
				; bits
001400		RT	S0	; Set clock to 0
001401		SIPI	1	; Set
				; interprocessor
				; interrupt
				; request
001402		CIPI		; Clear
				; interprocessor
				; interrupt
				; request
001403		CLN	0	
001413		CLN	1	
001423		CLN	2	
001433		CLN	3	
001434		PCI	S3	; Load the
				; low-order 32
				; bits from (S3)
				; to (II)
001405		CCI		; Clear clock
				; interrupt
001406		ECI		; Enable clock
				; interrupt
001407		DCI		; Disable clock
				; interrupt

INSTRUCTION 0015

Result	Operand	Description	Machine Instruction
		Select performance monitor	0015*j*0
		Set maintenance read mode	001501
		Load diagnostic checkbyte with S1	001511
		Set maintenance write mode 1	001521
		Set maintenance write mode 2	001531

NOTE

The 0015 instructions are not supported by CAL at this time.

Instruction 0015*j*0 selects one of four groups of hardware related events to be monitored by the performance counters.

Instructions 001501 through 001531 check the operation of the modules concerned with SECDED and to verify error detection and correction.

Instructions 001501 and 001521 verify check bit memory storage. Instructions 001511 and 001531 verify error detection and correction.

INSTRUCTION 0020

Result	Operand	Description	Machine Instruction
VL	A*k*	Transmit (A*k*) to VL	00200*k*
VL†	1	Enter 1 into VL	002000

† Special CAL syntax

Instruction 00200*k* and its special form (002000) enter the low-order 7 bits of the contents of register A*k* into the VL register.

The contents of the VL register determines the number of operations performed by a vector instruction. Since a vector register has 64 elements, from 1 to 64 operations can be performed. The number of operations is (VL) modulo 64. When (VL) is 0, the number of operations performed is 64.

In this publication, a reference to register V*i* implies operations involving the first *n* elements where *n* is the vector length unless a single element is explicitly noted as in the instructions S*i* V*j*,A*k* and V*i*,A*k* S*j*.

Vector operations controlled by the contents of VL begin with element 0 of the vector registers and operate on consecutive elements.

Examples:

In the first example, if (A3)=6 then (VL)=6 following instruction execution and subsequent vector instructions operate on elements 0 through 5 of vector registers.

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
002003		VL	A3	

In the second example, since the *k* designator is assembled as 0, (VL)=1 and vector instructions operate on only one element, element 0.

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
002000		VL	1	

Lastly, if (A5)=0, then (VL)=64 and vector instructions operate on all 64 elements of the vectors.

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
002005		VL	A5	

Result	Operand	Description	Machine Instruction
EFI		Enable floating-point interrupt	002100
DFI		Disable floating-point interrupt	002200
ERI†		Enable interrupt on address range error	002300
DRI†		Disable interrupt on address range error	002400
DBM††		Disable bidirectional memory transfers	002500
EBM††		Enable bidirectional memory transfers	002600
CMR††		Complete memory references	002700

† This instruction is available through the logical trait CORI specified on the CPU parameter of the CAL invocation statement.
†† This instruction is available through the logical trait BDM specified on the CPU parameter of the CAL invocation statement.

The EFI and DFI instructions provide for setting and clearing the Floating-point Interrupt flag in the Mode register. These instructions do not check the previous state of the flag.

****

CAUTION

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

****

The ERI and DRI instructions set and clear the Operand Range Mode flag in the Mode 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.

The DBM and EBM instructions disable and enable 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.

The CMR instruction assures completion of all memory references within a particular CPU issuing the instruction. This instruction 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 is assured of receiving the updated data if the load is issued after the CMR instruction. Synchronization of memory references between processors can be done by this instruction in conjunction with semaphore instructions.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
002300		ERI		
002400		DRI		
002500		DBM		
002600		EBM		
002700		CMR		

Result	Operand	Description	Machine Instruction
VM	S*j*	Transmit (S*j*) to VM	0030*j*0
VM†	0	Clear VM	003000
SM*jk*††	1,TS	Test and set semaphore *jk*, $0 \le jk \le 31$ (decimal)	0034*jk*
SM*jk*††	0	Clear semaphore *jk*, $0 \le jk \le 31$ (decimal)	0036*jk*
SM*jk*††	1	Set semaphore *jk*, $0 \le jk \le 31$ (decimal)	0037*jk*

† Special CAL syntax
†† This instruction is available when the numeric trait NUMCLSTR, which is specified on the CPU parameter of the CAL invocation statement, is greater than zero.

Instruction 0030*j*0 and its special form transmit the contents of register S*j* to the VM register. The VM register is zeroed if the *j* designator is 0; the special form accommodates this case.

This instruction may be used in conjunction with the vector merge instructions where an operation is performed depending on the contents of the VM register.

Instruction 0034*jk* tests and sets the semaphore designated by *jk*. If the semaphore is set, issue is held until another CPU clears that semaphore. If the semaphore is clear, the instruction issues and sets the semaphore.

If all CPUs in a cluster are holding issue on a test and set, the DL flag is set in the Exchange Package (if it is 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.

The 0036*jk* instruction clears the semaphore designated by *jk*.

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

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
003040		VM	S4	
003000		VM	0	; Clear VM
003407		SM7	1,TS	
003607		SM7	0	
003707		SM7	1	

INSTRUCTION 0040

Result	Operand	Description	Machine Instruction
EX		Normal exit	004000

Instruction 004000 causes an exchange sequence. The contents of the instruction buffers are voided by the exchange sequence. If monitor mode is not in effect, the Normal Exit flag in the F register is set. All instructions issued before this instruction are run to completion.

When the results of previously issued instructions have arrived at the operating registers, an exchange occurs to the Exchange Package designated by the contents of the Exchange Address (XA) register. The program address stored in the executing Exchange Package is advanced 1 parcel from the address of the normal exit instruction. This instruction is used to issue a monitor request from a user program, or to transfer control from a monitor program to another program.

The expression in the operand field is optional and has no effect on instruction execution; the low-order 9 bits of the expression value are placed in the *ijk* fields of the instruction.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
004000		EX		

INSTRUCTION 0050

Result	Operand	Description	Machine Instruction
J	B*jk*	Jump to (B*jk*)	0050*jk*

The 0050*jk* unconditional branch instruction sets the P register to the parcel address specified by the contents of register B*jk*. Execution continues at that address.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
005017		J	B17	
005003		J	B.RTNADDR	RTNADDR=03 (octal)

INSTRUCTION 0060

Result	Operand	Description	Machine Instruction
J	*exp*	Jump to *exp*	006*ijkm*

The 006*ijkm* unconditional branch instruction sets the P register to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
006 00002124b+		J	TAG1	
006 00001753a+		J	LDY3+1	

INSTRUCTION 0070

Result	Operand	Description	Machine Instruction
R	*exp*	Return jump to *exp;* set B00 to (P) + 2.	007*ijkm*

Instruction 007*ijkm* sets register B00 to the address of the parcel following the instruction. The P register is then set to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

The purpose of the instruction is to provide a return linkage for subroutine calls. The subroutine is entered via a return jump. The subroutine returns to the caller at the instruction following the call by executing a branch to the contents of the B register containing the saved address.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
007 00001142d+		R	HELP	

Result	Operand	Description	Machine Instruction
JAZ	*exp*	Branch to *exp* if (A0)=0	010*ijkm*
JAN	*exp*	Branch to *exp* if (A0)≠0	011*ijkm*
JAP	*exp*	Branch to *exp* if (A0) positive	012*ijkm*
JAM	*exp*	Branch to *exp* if (A0) negative	013*ijkm*

NOTE

When executing the above instructions on the CRAY X-MP computer systems, the high-order bit of *i* must be 0.

The above instructions test the contents of A0 for the specified condition. If the condition is satisfied, the P register is set to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

If the condition is not satisfied, execution continues with the instruction following the branch instruction. For the JAP and JAM instructions, a 0 value in A0 is considered positive.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
010 00002243d+		JAZ	TAG3+2	
011 00004520a+		JAN	P.CON1	
012 00002221c+		JAP	TAG2	
013 00002124b+		JAM	TAG1	

Result	Operand	Description	Machine Instruction
JSZ	*exp*	Branch to *exp* if (S0)=0	014*ijkm*
JSN	*exp*	Branch to *exp* if (S0)≠0	015*ijkm*
JSP	*exp*	Branch to *exp* if (S0) positive	016*ijkm*
JSM	*exp*	Branch to *exp* if (S0) negative	017*ijkm*

NOTE

When executing the above instructions on the CRAY X-MP computer system, the high-order bit of *i* must be 0.

The above instructions test the contents of S0 for the specified condition. If the condition is satisfied, the P register is set to the parcel address specified by the low-order 24 bits of the expression. Execution continues at that address.

If the condition is not satisfied, execution continues with the instruction following the branch instruction. For the JSP and JSM instructions, a zero value in S0 is considered positive.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
014 00002221c+		JSZ	TAG2	
015 00002124d+		JSN	TAG1+2	
017 00002367c+		JSM	TAG4	

INSTRUCTION 01*h*

Result	Operand	Description	Machine Instruction
A*h*†	*exp*	Transmit *ijkm* to A*h*; where the high-order bit of *i* is 1	01*hijkm*

† This instruction is available through the logical trait EMA specified on the CPU parameter of the CAL invocation statement, and CAL will then generate one of these instructions: 01*h*, 020, 021, 022, or 031.

Instruction 01*h* will not be generated if NOEMA is specified.

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

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
		EXT	EXTSYM	
0a 0114 00000001+		A1	TABSYM	
c 0124 00000000X		A2	EXTSYM	
1 45	TABSYM	BSS	O'45	

Result	Operand	Description	Machine Instruction
A*i*[†]	*exp*	Enter *exp* into A*i*	020*ijkm* or 021*ijkm* or 022*ijk*

† These instructions are available through the logical trait NOEMA specified on the CPU parameter of the CAL invocation statement, and CAL will generate one of these instructions: 020, 021, 022, 031.

The above instruction enters a quantity into A*i*. The syntax differs from most CAL symbolic instructions in that the assembler generates any of three Cray machine instructions depending on the form, value, and attributes of the expression.

The assembler generates an instruction 022*ijk* where the *jk* fields contain the 6-bit value of the expression if all of the following conditions are true:

- The value of the expression is positive and less than 64
- All symbols (if any) within the expression are previously defined
- The expression has a relative attribute of absolute

If any of the conditions are not true, the assembler generates either the 2-parcel instruction 020*ijkm* or 021*ijkm*. If the expression has a positive value, or has a relative attribute of either relocatable or external, instruction 020*ijkm* is generated with the value entered in the 22-bit *jkm* field. If the expression value is negative and has a relative attribute of absolute, instruction 021*ijkm* is generated with the ones complement of the expression value entered into the 22-bit *jkm* field except where the *exp* value is explicitly "-1".

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
022310		A3	O'10	
0212 00000010		A2	#O'10	
	AREG	=	2	
0212 00000007		A.AREG	-O'10	
0202 00000130		A2	O'130	
0203 00000021		A3	VAL+1	; VAL=20 (octal)
0204 01777777		A4	O'1777777	
0205 00051531		A5	A'SY'R	
0226 00000000		A6	#MINUS1	; MINUS1=-1
		EXT	X	
0204 17777777		A4	X-1	; 020*ijkm* used if
				; expression is
				; external

INSTRUCTION 023

Result	Operand	Description	Machine Instruction
A*i*	S*j*	Transmit (S*j*) to A*i*	023*ij*0
A*i*†	VL	Transmit (VL) to A*i*	023*i*01

† This instruction is available through the logical trait READVL specified on the CPU parameter of the CAL invocation statement.

Instruction 023*ij*0 transmits the low-order 24 bits of the contents of register S*j* to register A*i*. A*i* is zeroed if the *j* designator is 0.

Instruction 023*i*01 enters the contents of the VL register into A*i*.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
023420		A4	S2	
023201		A2	VL	

Result	Operand	Description	Machine Instruction
A*i*	B*jk*	Transmit (B*jk*) to A*i*	024*ijk*
B*jk*	A*i*	Transmit (A*i*) to B*jk*	025*ijk*

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

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

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
024517		A5	B17	
	SVNTN	=	O'17	
024517		A5	B.SVNTN	
025634		B34	A6	
025634		B.THRTY4	A6	; THRTY4=34 (octal

INSTRUCTION 026

Result	Operand	Description	Machine Instruction
A*i*	PS*j*	Population count of (S*j*) to A*i*	026*ij*0
A*i*[†]	QS*j*	Population count parity of (S*j*) to A*i*	026*ij*1
A*i*[††]	SB*j*	Transfer (SB*j*) to A*i*	026*ij*7

† This instruction is available through the logical trait VPOP specified on the CPU parameter of the CAL invocation statement.

†† This instruction is available when the numeric trait NUMCLSTR, which is specified on the CPU parameter of the CAL invocation statement, is greater than zero.

Instruction 026*ij*0 counts the number of 1 bits in the contents of S*j* and enters the result into A*i*. A*i* is zeroed if the *j* designator is 0.

Instruction 026*ij*1 enters a 0 in A*i* if S*j* has an even number of 1 bits and enters a 1 in A*i* if S*j* has an odd number of 1 bits.

These two instructions execute in the Scalar Leading Zero/Population Count functional unit.

Instruction 026*ij*7 transfers the contents of the SB*j* register shared between the CPUs in the current cluster to A*i*.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
026720		A7	PS2	; Pop count of ; S2 to A7
026271		A2	QS7	; Pop count ; parity of ; S7 to A2
026007		A0	SB0	
026017		A0	SB1	

INSTRUCTION 027

Result	Operand	Description	Machine Instruction
A*i*	ZS*j*	Leading zero count of (S*j*) to A*i*	027*ij*0
SB*j*†	A*i*	Transfer (A*i*) to SB*j*	027*ij*7

† This instruction is available when the numeric trait NUMCLSTR, which is specified on the CPU parameter of the CAL invocation statement, is greater than zero.

Instruction 027*ij*0 counts the number of leading zeros in the contents of S*j* and enters the result into A*i*. A*i* is set to 64 if the *j* designator is 0, or if the S*j* register contains 0.

This instruction executes in the Scalar Leading Zero/Population Count functional unit.

Instruction 027*ij*7 transfers the contents of register A*i* into register SB*j*, which is shared between the CPUs in the current cluster.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
027130		A1	ZS3	
027007		SB0	A0	
027107		SB0	A1	

Result	Operand	Description	Machine Instruction
A*i*	A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*	030*ijk*
A*i*†	A*j*+1	Integer sum of (A*j*) and 1 to A*i*	030*ij*0
A*i*†	A*k*	Transmit (A*k*) to A*i*	030*i*0*k*
A*i*	A*j*-A*k*	Integer difference of (A*j*) less (A*k*) to A*i*	031*ijk*
A*i*†	A*j*-1	Integer difference of (A*j*) less 1 to A*i*	031*ij*0
A*i*†	-A*k*	Transmit negative of (A*k*) to A*i*	031*i*0*k*
A*i*†	-1	Enter -1 into A*i*	031*i*00

† Special CAL syntax

Instruction 030*ijk* and its special form (030*ij*0) add the contents of register A*j* to the contents of register A*k* and enter the result into register A*i*. A*k* is transmitted to A*i* when the *j* designator is 0 and the *k* designator is nonzero. The value 1 is transmitted to A*i* when the *j* and *k* designators are both 0. (A*j*)+1 is transmitted to A*i* when the *j* designator is nonzero and the *k* designator is 0. The assembler allows an alternate form of the instruction when the *k* designator is 0.

The instruction executes in the Address Integer Add functional unit.

Instruction 030*i*0*k* enters the contents of register A*k* into register A*i*. The value 1 is entered if the *k* designator is 0.

Instruction 031*ijk* and its special form (031*ij*0) subtract the contents of register A*k* from the contents of register A*j* and enter the result into register A*i*. The negative of A*k* is transmitted to A*i* when the *j* designator is 0 and the *k* designator is nonzero. A -1 is transmitted to A*i* when the *j* and *k* designators are both 0. (A*j*)-1 is transmitted to A*i* when the *j* designator is nonzero and the *k* designator is 0.

The instruction 031*ijk* executes in the Address Integer Add functional unit.

The special form represents the case where (A*k*)=1 if *k*=0.

Instruction 031*i*0*k* enters the negative (twos complement) of the contents of register A*k* into register A*i*. The value -1 is entered into A*i* if the *k* designator is 0.

Instruction 031*i*00 is generated in place of instruction 020*ijkm* if the operand is explicitly -1.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
030123		A1	A2+A3	
030102		A1	A2	
030230		A2	A3+1	
030602		A6	A2	
031456		A4	A5-A6	
031102		A1	-A2	
031450		A4	A5-A1	
031703		A7	-A3	
031300		A3	-1	

INSTRUCTION 032

Result	Operand	Description	Machine Instruction
A*i*	A*j**A*k*	Integer product of (A*j*) and (A*k*) to A*i*	032*ijk*

Instruction 032*ijk* forms the integer product of the contents of register A*j* and register A*k* and enters the low-order 24 bits of the result into A*i*. A*i* is cleared when the *j* designator is 0. A*j* is transmitted to A*i* when the *k* designator is 0 and the *j* designator is nonzero.

The instruction executes in the Address Integer Multiply functional unit. There is no overflow detection.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
032712		A7	A1*A2	

INSTRUCTION 033

Result	Operand	Description	Machine Instruction
A*i*	CI	Channel number of highest priority interrupt request to A*i*	033*i*00
A*i*	CA,A*j*	Address of channel (A*j*) to A*i* (*j*≠0)	033*ij*0
A*i*	CE,A*j*	Error flag of channel (A*j*) to A*i*	033*ij*1

Instruction 033*i*00 enters the channel number of the highest priority interrupt request into A*i*.

Instruction 033*ij*0 enters the contents of the Current Address (CA) register for the channel specified by the contents of A*j* into register A*i*.

Instruction 033*ij*1 enters the error flag for the channel specified by the contents of A*j* into the low-order 7 bits of A*i*. The high-order bits of A*i* are cleared. The error flag can be cleared only in monitor mode using the CI,A*j* instruction, or the CRAY X-MP computer system instruction MC,A*j*.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
033100		A1	CI	
033230		A2	CA,A3	
033341		A3	CE,A4	

Result	Operand	Description	Machine Instruction
B*jk*,A*i*	,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	034*ijk*
B*jk*,A*i*†	0,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	034*ijk*
,A0	B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	035*ijk*
0,A0†	B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	035*ijk*
T*jk*,A*i*	,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	036*ijk*
T*jk*,A*i*†	0,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	036*ijk*
,A0	T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	037*ijk*
0,A0†	T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	037*ijk*

† Special CAL syntax

Instruction 034*ijk* and its special form are used to transfer words from memory directly into B registers. A0 contains the address of the first word of memory to be transferred. The *jk* designator specifies the first B register to be used in the transfer. The low-order 24 bits of consecutive words of memory are loaded into consecutive B registers.

Processing of B registers is circular. B00 is loaded after B77 if the count specified in A*i* is not exhausted after B77 is loaded. The low-order 7 bits of the contents of A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of (A*i*) are greater than 64.

If (A*i*)=0, no words are transferred. Note also that if *i*=0, (A0) is used for the block length as well as the starting memory address. The CAL assembler issues a warning message in this case.

Instruction 035*ijk* and its special form are used to store words from B registers directly into memory. A0 contains the address of the first word of memory to receive data. The *jk* designator specifies the first B register to be used in the transfer. Subsequent B register contents are stored in consecutive words of memory.

Processing of B registers is circular. B00 is processed after B77 if the count specified in A*i* is not exhausted after B77 is processed. The low-order 7 bits of the contents of A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of A*i* are greater than 64.

If (A*i*)=0, no words are transferred. Note also that if *i*=0, (A0) is used for the block length as well as the starting memory address. The CAL assembler issues a warning message in this case.

Instruction 036*ijk* and its special form are used to transfer words from memory directly into T registers. A0 contains the address of the first word of memory to be transferred. The *jk* designator specifies the first T register to be used in the transfer. The loading of T registers is circular. T00 is loaded after T77 if the count specified in A*i* is not exhausted after T77 is loaded. The low-order 7 bits of the contents of A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of A*i* are greater than 64.

If (A*i*)=0, no words are transferred. If *i*=0, (A0) is used for the block length and the starting memory address. The CAL assembler issues a warning message in this case.

Instruction 037*ijk* and its special form are used to store words from T registers directly into memory. A0 contains the address of the first word of memory to receive data. The *jk* designator specifies the first T register to be used in the transfer. Subsequent T register contents are stored in consecutive words of memory. Processing of T registers is circular. T00 is processed after T77 if the count specified in A*i* is not exhausted after T77 is processed. The low-order 7 bits of the contents of register A*i* specify the number of words transmitted. Wraparound occurs if the low-order 7-bits of A*i* are greater than 64.

If (A*i*)=0, no words are transferred. Note also that if *i*=0, (A0) is used for the block length as well as the starting memory address, and CAL issues a warning message.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
034407		B7,A4	,A0	
	BB	=	O'22	
	FWAR	=	5	
034522		B.BB,A.FWAR	0,A0	
035522		,A0	B22,A5	
	BB	=	O'22	
	FWAR	=	5	
035522		0,A0	B.BB,A.FWAR	
036407		T7,A4	,A0	
	TT	=	O'22	
	FWAR	=	5	
036522		T.TT,A.FWAR	0,A0	
37522		,A0	T22,A5	
	TT	=	O'22	
	FWAR	=	5	
037522		0,A0	T.TT,A.FWAR	

Result	Operand	Description	Machine Instruction
S*i*	*exp*	Enter *exp* into S*i*	040*ijkm* or 041*ijkm*

The above instruction enters a quantity into S*i*. Either the 2-parcel 040*ijkm* instruction or the 2-parcel 041*ijkm* instruction is generated, depending on the value of the expression.

If the expression has a positive value or a relative attribute of either relocatable or external, instruction 040*ijkm* is generated with the 22-bit *jkm* field containing the expression value. If the expression has a negative value and a relative attribute of absolute, instruction 041*ijkm* is generated with the 22-bit *jkm* field containing the ones complement of the expression value.

Refer to the 042-043 instructions for additional information on S*i* *exp* instructions.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
0402 00000130		S2	O'130	
	SREG	=	3	
0403 00000021		S.SREG	VAL+1	; VAL=20 (octal)
0404 01777777		S4	O'1777777	
0405 00051531		S5	A'SY'R	
			3	
0406 00000000		S6	#MINUS1	; MINUS1=-1
0413 00000002		S3	#2	
0414 01777776		S4	-O'1777777	
0414 00000003		S4	#VAL2	; VAL2=3
		EXT	X	
0401 17777777		S1	X-1	; 040*ijkm* used ; if expression ; is external

Result	Operand	Description	Machine Instruction
S*i*	<*exp*	Form ones mask in S*i* from right	042*ijk*
S*i*†	#>*exp*	Form zeros mask in S*i* from left	042*ijk*
S*i*†	1	Enter 1 into S*i*	042*i*77
S*i*†	-1	Enter -1 into Si	042*i*00
S*i*†	0	Clear S*i*	043*i*00
S*i*	>*exp*	Form ones mask in S*i* from left	043*ijk*
S*i*†	#<*exp*	Form zeros mask in S*i* from right	043*ijk*

† Special CAL syntax

Instruction 042*ijk* generates a mask of ones from the right. The assembler evaluates the expression to determine the mask length.

In the first instruction, the mask length is the value of the expression. In the second instruction, the mask length is 64 minus the expression value. The mask length must be a positive integer not exceeding 64; 64 minus the mask length is inserted into the *jk* fields of the instruction. If the value of the expression is 0 for the first instruction or 64 for the second instruction, the assembler generates instruction 043*i*00.

Instruction 042*ijk* executes in the Scalar Logical functional unit.

Instructions 042*i*77, 042*i*00, and 043*i*00 are initially recognized by the assembler as the symbolic instruction S*i* *exp*. The assembler then checks the expression to see if it has one of these three forms. If it finds one of the forms in the exact syntax shown, it generates the corresponding Cray machine instruction. If none of these forms is found, instruction 040*ijkm* or 041*ijkm* is generated. These special forms allow more efficient instructions for entering often used values into S1.

Instructions 043*i*00, 042*i*77, and 042*i*00 execute in the Scalar Logical functional unit.

Instruction 043*ijk* generates a mask of ones from the left. The assembler evaluates the expression to determine the mask length.

In instruction 043*ijk*, the mask length is the value of the expression. In the special syntax form, the mask length is 64 minus the expression value. The mask length must be a positive integer not exceeding 64 and is inserted into the *jk* fields of the instruction. If the expression value is 64 for the first instruction or 0 for the second instruction, the assembler generates instruction 042*i*00.

Instruction 043*ijk* executes in the Scalar Logical functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
042200		S2	-1	
042273		S2	<5	
042273		S2	#>O'73	
042366		S3	<D'10	
042400		S4	<O'100	
043500		S5	<0	
043600		S6	0	; Clear S6
042677		S6	1	; Set S6 to 1
043205		S2	>5	
043205		S2	#<O'73	
043500		S5	<0	

Result	Operand	Description	Machine Instruction
S*i*	S*j*&S*k*	Logical product of (S*j*) and (S*k*) to S*i*	044*ijk*
S*i*†	S*j*&SB	Sign bit of (S*j*) to S*i*	044*ij*0
S*i*†	SB&S*j*	Sign bit of (S*j*) to S*i*; *j*≠0	044*ij*0
S*i*	#S*k*&S*j*	Logical product of (S*j*) and #(S*k*) to S*i*	045*ijk*
S*i*†	#SB&S*j*	(S*j*) with sign bit cleared to S*i*	045*ij*0
S*i*	S*j*\S*k*	Logical difference of (S*j*) and (S*k*) to S*i*	046*ijk*
S*i*†	S*j*\SB	Enter (S*j*) into S*i* with sign bit toggled	046*ij*0
S*i*†	SB\S*j*	Enter (S*j*) into S*i* with sign bit toggled; *j*≠0	046*ij*0
S*i*	#S*j*\S*k*	Logical equivalence of (S*j*) and (S*k*) to S*i*	047*ijk*
S*i*†	#S*j*\SB	Logical equivalence of (S*j*) and sign bit to S*i*	047*ij*0
S*i*†	#SB\S*j*	Logical equivalence of sign bit and (S*j*) to S*i*; *j*≠0	047*ij*0

† Special CAL syntax

NOTE

When the above instructions execute, SB with no register designator is the sign bit, not Shared B register.

INSTRUCTIONS 044 - 051 (continued)

Result	Operand	Description	Machine Instruction
S*i*†	#S*k*	Transmit ones complement of (S*k*) to S*i*	047*i*0*k*
S*i*†	#SB	Enter ones complement of sign bit in S*i*	047*i*00
S*i*	S*j*!S*i*&S*k*	Scalar merge of (S*i*) and (S*j*) to S*i*	050*ijk*
S*i*†	S*j*!S*i*&SB	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*	050*ij*0
S*i*	S*j*!S*k*	Logical sum of (S*j*) and (S*k*) to S*i*	051*ijk*
S*i*†	S*j*!SB	Logical sum of (S*j*) and sign bit to S*i*	051*ij*0
S*i*†	SB!S*j*	Logical sum of sign bit and (S*j*) to S*i*; *j*≠0	051*ij*0
S*i*†	S*k*	Transmit (S*k*) to S*i*	051*i*0*k*
S*i*†	SB	Enter sign bit into S*i*	051*i*00

† Special CAL syntax

NOTE

When the above instructions execute, SB with no register designator is the sign bit, not Shared B register.

Instruction 044*ijk* forms the logical product of the contents of S*j* and S*k* and enters the result into S*i*. If the *j* and *k* designators have the same nonzero value, the contents of S*j* is transmitted to S*i*.

If the *j* designator is 0, register S*i* is zeroed. If the *j* designator is nonzero and the *k* designator is 0, the sign bit of the contents of S*j* is extracted. The two special forms of the instruction accommodate this case. The two forms perform identical functions, but *j* must not be equal to 0 in the second form. If *j* is equal to 0, an assembly error results.

Instruction 045*ijk* forms the logical product of the contents of S*j* and the ones complement of the contents of S*k* and enters the result into S*i*. If the *j* and *k* designators have the same value or if the *j* designator is 0, register S*i* is zeroed.

If the *j* designator is nonzero and the *k* designator is 0, the contents of S*j* with the sign bit cleared is transmitted to S*i*. The special syntax form accommodates this case.

Instruction 046*ijk* forms the logical difference of the contents of S*j* and the contents of S*k* and enters the result into S*i*. If the *j* and *k* designators have the same nonzero value, S*i* is zeroed.

If the *j* designator is 0 and the *k* designator is nonzero, the contents of S*k* is transmitted to S*i*. If the *j* designator is nonzero and the *k* designator is 0, the sign bit of the contents of S*j* is complemented and the result is transmitted to S*i*. The two special syntax forms provide for this case. The two forms perform identical functions; however, in the second form, *j* must not equal 0. If *j* equals 0, an assembly error results.

Instruction 047*ijk* forms the logical equivalence of the contents of S*j* and the contents of S*k* and enters the result into S*i*. Bits of S*i* are set to 1 when the corresponding bits of the contents of S*j* and the contents of S*k* are both 1 or both 0.

If the *j* and *k* designators have the same nonzero value, the contents of S*i* is set to all ones. If the *j* designator is 0 and the *k* designator is nonzero, the ones complement of the contents of S*k* is transmitted to S*i*. If the *j* designator is nonzero and the *k* designator is 0, all bits other than the sign bit of the contents of S*j* are complemented and the result is transmitted to S*i*.

The two special forms of the instruction accommodate this case. The two forms perform identical functions; however, in the second form, *j* must not equal 0. If *j* equals 0, an error results.

Instruction 047*i*0*k* forms the ones complement of the contents of register S*k* and enters the value into S*i*. The complement of the sign bit is entered into S*i* if the *k* designator is 0.

Instruction 047*i*00 clears the sign bit and sets all other bits.

Instructions 050*ijk* and 050*ij*0 merge the contents of S*j* with the contents of S*i* depending on the ones mask in S*k*.

The result is defined by (S*j*&S*k*)!(S*i*&#S*k*) as in the following example:

```
(Sk) = 11110000
(Si) = 11001100
(Sj) = 10101010
(Si) = 10101100
```

This instruction is intended for merging portions of 64-bit words into a composite word. S*i* bits are cleared when the corresponding S*k* bits 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 as provided for by the special syntax form of the instruction. The sign bit of S*i* is cleared if the *j* and *k* designators are both 0.

Instruction 051*ijk* forms the logical sum of the contents of S*j* and the contents of S*k* and enters the result into S*i*. If the *j* and *k* designators have the same nonzero value, the contents of S*j* are transmitted to S*i*. If the *j* designator is 0 and the *k* designator is nonzero, the contents of S*k* are transmitted to S*i*.

If the *j* designator is nonzero and the *k* designator is 0, the contents of S*j* with the sign bit set to 1 are transmitted to S*i*. The two special syntax forms provide for this case. If the *j* and *k* designators are both 0, a ones mask consisting of only the sign bit is entered into S*i*.

The two special forms perform an identical function but in the second form $j \neq 0$; if $j=0$, an assembly error results.

Instruction 051*i*0*k* enters the contents of register S*k* into register S*i*. The sign bit is set to 1 in S*i* if the *k* designator is 0.

Instruction 051*i*00 can be used to set the sign bit of S*i* and zero all other bits.

Instructions 044*ijk* through 051 execute in the Scalar Logical functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
044235		S2	S3&S5	
044655		S6	S5&S5	; S5 to S6
044160		S1	S6&SB	; Get sign of S6
044160		S1	SB&S6	; Get sign of S6
045271		S2	#S1&S7	
045430		S4	#SB&S3	; Clear sign bit ; of S3 and ; enter into S4
045506		S5	#S6&S0	; Clear S5
045670		S6	#SB&S7	; Clear sign bit
046123		S1	S2\S3	
046455		S4	S5\S5	; Clear S4
046506		S5	S0\S6	; S6 to S5
046770		S7	S7\SB	; Toggle sign ; bit
047345		S3	#S4\S5	
047260		S2	#S6\SB	
047260		S2	#SB\S6	
047203		S2	#S3	
047200		S2	#SB	
050123		S1	S2!S1&S3	
050760		S7	S6!S7&S0	

Example (continued):

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
051472		S4	S7!S2	
051366		S3	S6!S6	
051710		S7	SB!S1	
051701		S7	S1	
	I	=	1	
051100		S.I	SB	

Result	Operand	Description	Machine Instruction
S0	S*i*<*exp*	Shift (S*i*) left *exp* places to S0	052*ijk*
S0	S*i*>*exp*	Shift (S*i*) right *exp* places to S0	053*ijk*
S*i*	S*i*<*exp*	Shift (S*i*) left *exp* places to S*i*	054*ijk*
S*i*	S*i*>*exp*	Shift (S*i*) right *exp* places to S*i*	055*ijk*

Instruction 052*ijk* shifts the contents of S*i* to the left by the amount specified by the expression and enters the result into S0. The shift count must be a positive integer value not exceeding 64. The shift is end off with zero fill. If the shift count is 64, instruction 053000 is generated and S0 is zeroed.

Instruction 053*ijk* shifts the contents of S*i* to the right by the amount specified by the expression and enters the result into S0. The shift count must be a positive integer value not exceeding 64. The assembler stores 64 minus the shift count in the *jk* field of the instruction. The shift is end off with zero fill. If the shift count is 0, instruction 052*i*00 is generated and the content of S0 is not altered.

Instruction 054*ijk* shifts the contents of S*i* to the left by the amount specified by the expression and enters the result into S*i*. The shift count must be a positive integer value not exceeding 64. The shift is end off with zero fill. If the shift count is 64, instruction 055*i*00 is generated and S*i* is zeroed.

Instruction 055*ijk* shifts the contents of S*i* to the right by the amount specified by the expression and enters the result into S*i*. The shift count must be a positive integer value not exceeding 64. The assembler stores 64 minus the shift count in the *jk* field of the instruction. If the shift count is 0, instruction 054*i*00 is generated and the content of S*i* is not altered. The shift is end off with zero fill.

Instructions 052*ijk*, 053*ijk*, 054*ijk*, and 055*ijk* execute in the Scalar Shift functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
052305		S0	S3<5	
052724		S0	S7<VAL+4	
053373		S0	S3>5	
053066		S0	S0>D'10	
053754		S0	S7>VAL+4	
052100		S0	S1>0	
054703		S7	S7<3	
054622		S6	S6<VAL+2	
055775		S7	S7>3	
055656		S6	S6>VAL+2	

Result	Operand	Description	Machine Instruction
S*i*	S*i*,S*j*<A*k*	Left shift by (A*k*) of (S*i*) and (S*j*) to S*i*	056*ijk*
S*i*†	S*i*,S*j*<1	Left shift by 1 of (S*i*) and (S*j*) to S*i*	056*ij*0
S*i*†	S*i*<A*k*	Left shift by (A*k*) of (S*i*) to S*i*	056*i*0*k*
S*i*	S*j*,S*i*>A*k*	Right shift by (A*k*) of (S*j*) and (S*i*) to S*i*	057*ijk*
S*i*†	S*j*,S*i*>1	Right shift by 1 of (S*j*) and (S*i*) to S*i*	057*ij*0
S*i*†	S*i*>A*k*	Right shift by (A*k*) of (S*i*) to S*i*	057*i*0*k*

† Special CAL syntax

Instruction 056*ijk* and its special forms produce a 128-bit quantity by concatenating the contents of S*i* and the contents of S*j*, shifting the resulting value to the left by an amount specified by the low-order bits of A*k* and entering the high-order bits of the result into S*i*. The shift is end off with zero fill.

Replacing the A*k* reference with 1 is the same as setting the *k* designator to 0; a reference to A0 provides a shift count of 1. Omitting the S*j* reference is the same as setting the *j* designator to 0; the contents of S*i* are concatenated with a word of zeros.

S*i* is cleared if the shift count exceeds 127. The shift is a left circular shift of the contents of S*i* if the shift count does not exceed 64 and the *i* and *j* designators are equal and nonzero. The instruction produces the same result as the S*i* S*i*<*exp* instruction if the shift count does not exceed 63 and the *k* designator is 0. The contents of S*j* are not affected if the *i* and *j* designators are unequal.

Instruction 057*ijk* and its special forms produce a 128-bit quantity by concatenating the contents of S*j* and the contents of S*i*, shifting the resulting value to the right by an amount specified by the low-order 7 bits of the contents of A*k* and entering the low-order bits of the result into S*i*. The shift is end off with zero fill.

Replacing the A*k* reference with 1 is the same as setting the *k* designator to 0; a reference to A0 provides a shift count of 1. Omitting the S*j* reference is the same as setting the *j* designator to 0; the contents of S*i* are concatenated with a word of zeros.

S*i* is cleared if the shift count exceeds 127. The shift is a right circular shift of the contents of S*i* if the shift count does not exceed 64 and the *i* and *j* designators are equal and nonzero. The instruction produces the same result as the S*i* S*i*>*exp* instruction if the shift count does not exceed 63 and the *j* designator is 0. The contents of S*j* are not affected if the *i* and *j* designators are unequal.

Instruction 056*ijk* and 057*ijk* executes in the Scalar Shift functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
056235		S2	S2,S3<A5	
056340		S3	S3,S4<1	; Left 1 place
056604		S6	S6<A4	
057235		S2	S3,S2>A5	
057604		S6	S6>A4	
057340		S3	S4,S3>1	; Right 1 place

Result	Operand	Description	Machine Instruction
S*i*	S*j*+S*k*	Integer sum of (S*j*) and (S*k*) to S*i*	060*ijk*
S*i*	S*j*-S*k*	Integer difference of (S*j*) less (S*k*) to S*i*	061*ijk*
S*i*†	-S*k*	Transmit negative of (S*k*) to S*i*	061*i*0*k*

† Special CAL syntax

Instruction 060*ijk* adds the contents of register S*k* to the contents of register S*j* and enters the result into S*i*. S*k* is transmitted to S*i* if the *j* designator is 0 and the *k* designator is nonzero. The sign bit is entered in S*i* and all other bits of S*i* are cleared if the *j* and *k* designators are both 0.

Instruction 061*ijk* subtracts the contents of register S*k* from the contents of register S*j* and enters the result into S*i*. The high-order bit of S*i* is set and all other bits of S*i* are cleared when the *j* and *k* designators are both 0. The negative (twos complement) of S*k* is transmitted to S*i* if the *j* designator is 0 and the *k* designator is nonzero.

Instruction 061*i*0*k* enters the negative (twos complement) of the contents of S*k* into S*i*. The sign bit is set if the *k* designator is 0.

Instructions 060*ijk*, 061*ijk*, 061*i*0*k* execute in the Scalar Integer Add functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
060237		S2	S3+S7	
060405		S4	S0+S5	
061123		S1	S2-S3	
061506		S5	-S6	

Result	Operand	Description	Machine Instruction
S*i*	S*j*+FS*k*	Floating-point sum of (S*j*) and (S*k*) to S*i*	062*ijk*
S*i*†	+FS*k*	Normalize (S*k*) to S*i*	062*i*0*k*
S*i*	S*j*-FS*k*	Floating-point difference of (S*j*) less (S*k*) to S*i*	063*ijk*
S*i*†	-FS*k*	Transmit the negative of (S*k*) as a normalized floating-point value	063*i*0*k*

† Special CAL syntax

Instruction 062*ijk* and its special form produce the floating-point sum of the contents of the S*j* and S*k* registers and enters the result into S*i*. The result is normalized even if the operands are unnormalized. The *k* designator is not normally 0. In the special form, the *j* designator is assumed to be 0 so that the normalized contents of S*k* are entered into S*i*.

Instruction 063*ijk* forms the floating-point difference of the contents of register S*j* less the contents of register S*k*, and enters the normalized result into S*i*. The result is normalized even if the operands are unnormalized.

The negative (twos complement) of the floating-point quantity in S*k* is transmitted to S*i* as a normalized floating-point number if the *j* designator is 0 and the *k* designator is nonzero. The special form accommodates this special case. The *k* designator is normally nonzero.

Instructions 062*ijk*, 063*ijk*, and 063*i*0*k* execute in the Floating-point Add functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
062345		S3	S4+FS5	
062404		S4	+FS4	
063302		S3	-FS2	
063761		S7	S6-FS1	

Result	Operand	Description	Machine Instruction
S*i*	S*j**FS*k*	Floating-point product of (S*j*) and (S*k*) to S*i*	064*ijk*
S*i*	S*j**HS*k*	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*	065*ijk*
S*i*	S*j**RS*k*	Rounded floating-point product of (S*j*) and (S*k*) to S*i*	066*ijk*
S*i*	S*j**IS*k*	2-floating-point product of (S*j*) and (S*k*) to S*i*	067*ijk*

Instruction 064*ijk* forms the floating-point product of the contents of S*j* and S*k* and enters the result into S*i*. The result is not normalized if either operand is unnormalized.

Instruction 065*ijk* forms the half-precision rounded floating-point product of the contents of the S*j* and S*k* registers and enters the result into S*i*. The result is not normalized if either operand is unnormalized. The low-order 18 bits of the result are zeroed. This instruction can be used in a divide algorithm when only 30 bits of accuracy are required.

Instruction 066*ijk* forms the rounded floating-point product of the contents of the S*j* and S*k* registers and enters the result into S*i*. The result is not normalized if either operand is unnormalized. This operation is used in the reciprocal approximation sequence.

Instruction 067*ijk* forms 2 minus the floating-point product of the contents of S*j* and S*k* and enters the result into S*i*. The result is not normalized if either operand is unnormalized.

Instructions 064*ijk*, 065*ijk*, 066*ijk*, and 067*ijk* execute in the Floating-point Multiply functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
064234		S2	S3*FS4	
065167		S1	S6*HS7	
066147		S1	S4*RS7	
067324		S3	S2*IS4	

INSTRUCTION 070

Result	Operand	Description	Machine Instruction
S*i*	/HS*j*	Floating-point reciprocal approximation of (S*j*) to S*i*	070*ij*0

Instruction 070*ij*0 forms an approximation to the reciprocal of the floating-point value in S*j* and enters the result into S*i*. The result is meaningless if the contents of S*j* is unnormalized or 0. This instruction is used in the divide sequence as illustrated in the following example.

Instruction 070*ij*0 executes in the Floating-point Reciprocal functional unit.

Example:

```
|Code Generated          |Location|Result     |Operand        |Comment
|________________________|1_______|10_________|20_____________|35____
|                        |        |           |               |
|                        |*       |Divide S1 by S2; result to S1
|070320                  |        |S3         |/HS2           |; Approximate
|                        |        |           |               |; reciprocal
|                        |        |           |               |
|064113                  |        |S1         |S1*FS3         |; Approximate
|                        |        |           |               |; result
|                        |        |           |               |
|067223                  |        |S2         |S2*IS3         |; Correction
|                        |        |           |               |; factor
|                        |        |           |               |
|064112                  |        |S1         |S1*FS2         |
|                        |        |           |               |
|                        |        |           |               |
|                        |*       |Divide S1 by S2 with result accurate to
|                        |*       |30 bits    |               |
|070320                  |        |S3         |/HS2           |
|                        |        |           |               |
|065313                  |        |S3         |S1*HS3         |
```

Example (continued):

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
	*	Integer divide A1 by A2; Result to A3		
071222		S2	+FA2	; Denominator
071121		S1	+FA1	; Numerator
062202		S2	S0+FS2	; Normalize
062101		S1	S0+FS1	
070220		S2	/HS2	; Reciprocal ; approximation ; to 1/D
065110		S1	S1*HS2	; Rounded ; half-precision ; multiply
071230		S2	0.6	
062112		S1	S1+FS2	; Fix quotient
023310		A3	S1	; 24-bit signed ; result to A3

INSTRUCTION 071

Result	Operand	Description	Machine Instruction
S*i*	A*k*	Transmit (A*k*) to S*i* without sign extension	071*i*0*k*
S*i*	+A*k*	Transmit (A*k*) to S*i* with sign extension	071*i*1*k*
S*i*	+FA*k*	Transmit (A*k*) to S*i* as an unnormalized floating-point value	071*i*2*k*
S*i*	0.6	Enter 0.75*(2**48) into S*i* as normalized floating-point constant	071*i*30
S*i*	0.4	Enter 0.5 into S*i* as normalized floating-point constant	071*i*40
S*i*	1.	Enter 1 into S*i* as normalized floating-point constant	071*i*50
S*i*	2.	Enter 2 into S*i* as normalized floating-point constant	071*i*60
S*i*	4.	Enter 4 into S*i* as normalized floating-point constant	071*i*70

Instruction 071*i*0*k* transfers the 24-bit value in register A*k* into the low-order 24 bits of register S*i*. The value is treated as an unsigned integer. The high-order bits of S*i* are zeroed. A value of 1 is entered into S*i* when the *k* designator is 0.

Instruction 071*i*1*k* transfers the 24-bit value in register A*k* into the low-order 24 bits of register S*i*. The value is treated as a signed integer and the sign bit of the contents of register A*k* is extended to the high-order bits of S*i*. A value of 1 is entered into S*i* when the *k* designator is 0.

Instruction 071*i*2*k* transmits the contents of register A*k* to S*i* as an unnormalized floating-point value. The result can then be added to 0 to normalize. When the *k* designator is 0, an unnormalized floating-point 1 is entered into S*i*.

Instructions 071*i*30 through 071*i*70 are initially recognized by the assembler as the symbolic instruction S*i* *exp*. The assembler then checks the expression to see if it has any of the indicated forms. If it finds one of the instructions in the exact syntax shown, it generates the corresponding Cray machine instruction. If none of the indicated forms are found, instruction 040*ijkm* or 041*ijkm* is generated as previously described under the 040 instruction. These special forms allow more efficient instructions for entering commonly used values into S*i*.

The syntax form S*i* 0.6 (071*i*30) is useful for extracting the integer part of a floating-point quantity (that is, fix) as illustrated in the examples.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
071707		S7	A7	
071717		S7	+A7	
071324		S3	+FA4	
	FIX	=	6	
071630		S.FIX	0.6	
071240		S2	0.4	
071350		S3	1.	
071460		S4	2.	
071570		S5	4.	
	*	Fix a floating-point number in S1		
	*	Separate integer and fractional parts		
071230		S2	0.6	
062312		S3	S1+FS2	
023130		A1	S3	; Integer part
063332		S3	S3-FS2	; Floating-point ; integer part
063113		S1	S1-FS3	; Fractional ; part

Result	Operand	Description	Machine Instruction
S*i*	RT	Transmit (RTC) to S*i*	072*i*00
S*i*†	SM	Read semaphore to S*i*	072*i*02
S*i*†	ST*j*	Read (ST*j*) register to S*i*	072*ij*3
S*i*	VM	Transmit (VM) to S*i*	073*i*00
††		Read performance counter into S*i*	073*i*11
††		Increment performance counter	073*i*21
††		Clear all maintenance modes	073*i*31
S*i*†††	SR0	Transmit (SR0) to S*i*	073*i*01
SM†	S*i*	Load semaphores from S*i*	073*i*02
ST*j*†	S*i*	Transfer (S*i*) to ST*j*	073*ij*3
S*i*	T*jk*	Transmit (T*jk*) to S*i*	074*ijk*
T*jk*	S*i*	Transmit (S*i*) to T*jk*	075*ijk*

† This instruction is available when the numeric trait NUMCLSTR, which is specified on the CPU parameter of the CAL invocation statement, is greater than zero.
†† Not currently supported
††† This instruction is available through the logical trait STATRG specified on the CPU parameter of the CAL invocation statement.

Instruction 072*i*00 enters the 64-bit contents of the real-time clock into register S*i*. The clock is increased by 1 each clock period. The real-time clock can be reset only when in monitor mode using instruction 072*i*00.

Instruction 072*i*02 enters the values of all of the semaphores into S*i*. The 32-bit SM register is left justified in S*i* with SM00 occupying the sign bit.

Instruction 072*ij*3 enters the contents of register ST*j* into register S*i*.

Instruction 073*i*00 enters the 64-bit contents of the VM register into register S*i*. The VM register is normally read after having been set by instruction 1750*jk*.

Instruction 073*i*11 is used for performance monitoring and is priviledged to monitor mode.

Instructions 073*i*21 and 073*i*31 are part of the SECDED maintenance mode functions and are executed only if the maintenance mode switch on the mainframe's control panel is on.

Instruction 073*i*01 enters the contents of the Status register into S*i*.

Instruction 073*i*02 sets the semaphores from 32 high-order bits of S*i*. SM00 receives the sign bit of S*i*.

Instruction 073*ij*3 transfers the contents of register S*i* into register ST*j*, which is shared between the CPUs in the current cluster.

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

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

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
072700		S7	RT	
072002		S0	SM	
072602		S6	SM	
072003		S0	ST0	
072013		S0	ST1	
073200		S2	VM	
073001		S0	SR0	
073301		S3	SR0	
073002		SM	S0	
073102		SM	S1	

Example (continued):

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
073502		SM	S5	
073003		ST0	S0	
073103		ST0	S1	
074306		S3	T6	
074566		S5	T66	
075306		T6	S3	
075567		T67	S5	

Result	Operand	Description	Machine Instruction
S*i*	V*j*,A*k*	Transmit (V*j*, element (A*k*)) to S*i*	076*ijk*
V*i*,A*k*	S*j*	Transmit (S*j*) to V*i* element (A*k*)	077*ijk*
V*i*,A*k*†	0	Clear element (A*k*) of register V*i*	077*i*0*k*

† Special CAL syntax

Instruction 076*ijk* enters the contents of the element of V*j* indicated by the contents of the low-order 6 bits of A*k* into S*i*. The second element (that is, element 1) is selected if the *k* designator is 0.

Instruction 077*ijk* transmits the contents of register S*j* to an element of V*i* as determined by the low-order 6 bits of the contents of A*k*. Element 1, the second element of V*i*, is selected if the *k* designator is 0.

Instruction 077*i*0*k* zeros element (A*k*) of register V*i*. The low-order 6 bits of A*k* determine which element is zeroed. The second element of register V*i* is zeroed (that is, element 1) if the *k* designator is 0.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
076456		S4	V5,A6	
	I	=	4	
	J	=	5	
	K	=	6	
076456		S.I	V.J,A.K	
077167		V1,A7	S6	
077602		V6,A2	0	

Result	Operand	Description	Machine Instruction
A*i*	*exp*,A*h*	Read from ((A*h*) + *exp*) to A*i*	10*hijkm*
A*i*†	*exp*,0	Read from (*exp*) to A*i*	100*ijkm*
A*i*†	*exp*,	Read from (*exp*) to A*i*	100*ijkm*
A*i*†	,A*h*	Read from (A*h*) to A*i*	10*hi*000
exp,A*h*	A*i*	Store (A*i*) to (A*h*) + *exp*	11*hijkm*
exp,0†	A*i*	Store (A*i*) to *exp*	110*ijkm*
exp,†	A*i*	Store (A*i*) to *exp*	110*ijkm*
,A*h*†	A*i*	Store (A*i*) to (A*h*)	11*hi*000
S*i*	*exp*,A*h*	Read from ((A*i*) + *exp*) to S*i*	12*hijkm*
S*i*†	*exp*,0	Read from (*exp*) to S*i*	120*ijkm*
S*i*†	*exp*,	Read from (*exp*) to S*i*	120*ijkm*
S*i*†	,A*h*	Read from (A*h*) to S*i*	12*hi*000
exp,A*h*	S*i*	Store (S*i*) to (A*h*) + *exp*	13*hijkm*
exp,0†	S*i*	Store (S*i*) to *exp*	130*ijkm*
exp,†	S*i*	Store (S*i*) to *exp*	130*ijkm*
,A*h*†	S*i*	Store (S*i*) to (A*h*)	13*hi*000

† Special CAL syntax

For these instructions, only the value of the expression is used if the *h* designator is 0 or if a zero or blank field is used in place of A*h*. Only the content of A*h* is used if the expression is omitted. An expression, if present, must not have a parcel-address attribute or an assembly error occurs.

Instructions 10*hijkm* through 10*hi*000 load the low-order 24 bits of a memory word directly into an A register. The memory address is determined by adding the address in the register A*h* to the expression value. Only the value of the expression is used if the *h* designator is 0, or a 0 or blank field is used in place of A*h*. Only the contents of A*h* is used if the expression is omitted. An assembler error will occur if an expression has a parcel-address attribute.

Instructions 11*hijkm* through 11*hi*000 store 24 bits from register A*i* directly into memory. The high-order bits of the memory word are zeroed. The memory address is determined by adding the address in register A*h* to the expression value.

Instructions 12*hijkm* through 12*hi*000 load the contents of a memory word directly into an S register. The memory address is determined by adding the address in register A*h* to the expression value. Only the value of the expression is used if the *h* designator is 0 or a zero or blank field is used in place of A*h*. Only the contents of A*h* is used if the expression is omitted. An assembler error will occur if an expression has a parcel-address attribute.

Instructions 13*hijkm* through 13*hi*000 store the contents of register S*i* directly into memory. The memory address is determined by adding the address in register A*h* to the expression value.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
1001 00004520+		A1	CON1,A0	
1002 00004520+		A2	CON1,0	
1013 00004521+		A3	CON1+1,A1	
1024 17777777+		A4	-1,A2	
1005 00003000+		A5	ADDR,	
1046 00004647+		A6	CON,A4	
1046 00000000+		A6	,A4	
1061 00000001+		A1	1,A6	
1072 00000177+		A2	O'177,A7	
1101 00004520+		CON1,A0	A1	

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
1102 00004520+		CON1,0	A2	
1113 00004521+		CON1+1,A1	A3	
1124 17777777+		-1,A2	A4	
1105 00003000+		ADDR,	A5	
1146 00004647+		CON,A4	A6	
1146 00000000+		,A4	A6	
1161 00000001+		1,A6	A1	
1172 00000177+		O'177,A7	A2	
1201 00004520+		S1	CON1,A0	
1202 00004520+		S2	CON1,0	
1213 00004521+		S3	CON1+1,A1	
1224 17777777+		S4	-1,A2	
1205 00003000+		S5	ADDR,	
1246 00004647+		S6	CON,A4	
1246 00000000+		S6	,A4	
1261 00000001+		S1	1,A6	
1272 00000177+		S2	O'177,A7	
1301 00004520+		CON1,A0	S1	
1302 00004520+		CON1,0	S2	
1346 00000000+		,A4	S6	
1324 17777777+		-1,A2	S4	
1305 00003000+		ADDR,	S5	

Result	Operand	Description	Machine Instruction
V*i*	S*j*&V*k*	Logical products of (S*j*) and (V*k*) to V*i*	140*ijk*
V*i*	V*j*&V*k*	Logical products of (V*j*) and (V*k*) to V*i*	141*ijk*
V*i*	S*j*!V*k*	Logical sums of (S*j*) and (V*k*) to V*i*	142*ijk*
V*i*†	V*k*	Transmit (V*k*) to V*i*	142*i*0*k*
V*i*	V*j*!V*k*	Logical sums of (V*j*) and (V*k*) to V*i*	143*ijk*
V*i*	S*j*\V*k*	Logical differences of (S*j*) and (V*k*) to V*i*	144*ijk*
V*i*	V*j*\V*k*	Logical differences of (V*j*) and (V*k*) to V*i*	145*ijk*
V*i*†	0	Clear V*i*	145*iii*
V*i*	S*j*!V*k*&VM	Vector merge of (S*j*) and (V*k*) to V*i*	146*ijk*
V*i*†	#VM&V*k*	Vector merge of (V*k*) and zero to V*i*	146*i*0*k*
V*i*	V*j*!V*k*&VM	Vector merge of (V*j*) and (V*k*) to V*i*	147*ijk*

† Special CAL syntax

Instruction 140*ijk* forms the logical products of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. If the *j* designator is 0, elements of register V*i* are zeroed. The number of operations performed by this instruction is determined by the contents of the VL register.

Instruction 141*ijk* forms the logical products of the contents of elements of register V*j* and elements of register V*k* and enters the results into elements of V*i*. If the *j* designator is the same as the *k* designator, the contents of the V*j* elements are transmitted to the V*i* elements.

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

Instruction 142*ijk* forms the logical sums of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. The contents of the V*k* elements are transmitted to the V*i* elements if the *j* designator is 0. The VL register determines the number of operations performed by this instruction.

Instruction 142*i*0*k* transmits the contents of the elements of register V*k* to the elements of register V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 143*ijk* forms the logical sums of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*.

If the *j* and *k* designators are equal, the contents of the V*j* elements are transmitted to V*i*. The VL register determines the number of operations performed by this instruction.

Instruction 144*ijk* forms the logical differences of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. If the *j* designator is 0, the contents of the V*k* elements are entered into the V*i* elements. The VL register determines the number of operations performed by this instruction.

Instruction 145*ijk* forms the logical differences of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. If the *j* and *k* designators are equal, the V*i* elements are zeroed. The VL register determines the number of operations performed by this instruction.

Instruction 145*iii* zeros elements of V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 146*ijk* transmits the contents of S*j* or the contents of element *n* of V*k* to element *n* of V*i* depending on the ones mask in the VM register. The content of S*j* is transmitted if bit *n* of VM is 1; the content of element *n* of V*k* is transmitted if bit *n* of VM is 0.

Element *n* of V*i* is 0 if the *j* designator is 0 and bit *n* of VM is 1. The VL register determines the number of operations performed by this instruction.

Instruction 146*i*0*k* zeroes element *n* of register V*i* or transmits the contents of element *n* of V*k* to element *n* of V*i* depending on the ones mask in the VM register. If bit *n* of VM is 1, element *n* of V*i* is zeroed; if bit *n* is 0, element *n* of V*k* is transmitted. The VL register determines the number of operations performed by this instrction.

Instruction 147*ijk* transmits the contents of element *n* of V*j* or element *n* of V*k* to element *n* of V*i* depending on the ones mask in the VM register. The content of the V*j* element is transmitted if bit *n* of VM is 1; the content of the V*k* element is transmitted if bit *n* of VM is 0. The VL register determines the number of operations performed by this instruction.

Instructions 140*ijk* through 147*ijk* execute in the Vector Logical functional unit.

For these instructions (except 145*iii*), a warning level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and either *i*=*j* or *i*=*k* (for V registers only). A comment level message is issued of NOVRECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
140123		V1	S2&V3	
141257		V2	V5&V7	
141033		V0	V3&V3	
142615		V6	S1!V5	
142102		V1	V2	
143714		V7	V1!V4	
144267		V2	S6\V7	
145513		V5	V1\V3	
145500		V5	0	

Example (continued):

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
146726		V7	S2!V6&VM	

For the above instruction, assume the following initial register conditions exist:

```
                (VL) =  4
                (VM) =  0 60000 0000 0000 0000 0000
                (S2) = -1
    Element 0 of V6 =   1
    Element 1 of V6 =   2
    Element 2 of V6 =   3
    Element 3 of V6 =   4
```

After instruction execution, the first four elements of V7 are modified as follows:

```
    Element 0 of V7 =  1
    Element 1 of V7 = -1
    Element 2 of V7 = -1
    Element 3 of V7 =  4
```

The remaining elements of V7 are unaltered.

Example (continued):

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
146607		V6	#VM&V7	

Assume the following initial register conditions for the above instruction:

```
               (VL) = 4
               (VM) = 0 50000 0000 0000 0000 0000
    Element 0 of V7 = 1
    Element 1 of V7 = 2
    Element 2 of V7 = 3
    Element 3 of V7 = 4
```

After instruction execution, the first four elements of V6 have been modified as follows:

```
Element 0 of V6 = 1
Element 1 of V6 = 0
Element 2 of V6 = 3
Element 3 of V6 = 0
```

Example (continued):

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
147123		V1	V2!V3&VM	

Assume the following initial register conditions exist for the above instruction:

```
              (VL) =  4
              (VM) =  0 60000 0000 0000 0000 0000
Element 0 of V2 =  1
Element 1 of V2 =  2
Element 2 of V2 =  3
Element 3 of V2 =  4
Element 0 of V3 = -1
Element 1 of V3 = -2
Element 2 of V3 = -3
Element 3 of V3 = -4
```

After instruction execution, the first four elements of V*i* have been modified as follows:

```
Element 0 of V1 = -1
Element 1 of V1 =  2
Element 2 of V1 =  3
Element 3 of V1 = -4
```

The remaining elements of V1 are unaltered.

Result	Operand	Description	Machine Instruction
V*i*	V*j*<A*k*	Shift (V*j*) left (A*k*) places to V*i*	150*ijk*
V*i*†	V*j*<1	Shift (V*j*) left one place to V*i*	150*ij*0
V*i*	V*j*>A*k*	Shift (V*j*) right (A*k*) places to V*i*	151*ijk*
V*i*†	V*j*>1	Shift (V*j*) right one place to V*i*	151*ij*0

† Special CAL syntax

Instruction 150*ijk* and its special form shift the contents of the elements of register V*j* to the left by the amount specified by the contents of A*k* and enter the results into the elements of V*i*. The VL register determines the number of elements performed by this instruction. For each element, the shift is end off with zero fill. Elements of V*i* are zeroed if the shift count exceeds 63. Element contents are shifted left one place if the *k* designator is 0; this can be specified through the special form of the instruction.

Instruction 151*ijk* and its special form shift the contents of the elements of register V*j* to the right by the amount specified by the contents of A*k* and enter the results into the elements of V*i*. The VL register determines the number of elements performed by this instruction. For each element, the shift is end off with zero fill. Elements of V*i* are zeroed if the shift count exceeds 63. Element contents are shifted right one place if the *k* designator is 0; a special form of the instruction accommodates this feature.

Instructions 150*ijk* and 151*ijk* execute in the Vector Shift functional unit.

For these instructions, a warning-level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and *i*=*j*. A comment-level message is issued if NORECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
150123		V1	V2<A3	
150450		V4	V5<1	; Left 1 place
151341		V3	V4>A1	
151450		V4	V5>1	; Right 1 place

Result	Operand	Description	Machine Instruction
V*i*	V*j*,V*j*<A*k*	Double shift (V*j*) left (A*k*) places to V*i*	152*ijk*
V*i*†	V*j*,V*j*<1	Double shift (V*j*) left one place to V*i*	152*ij*0
V*i*	V*j*,V*j*>A*k*	Double shift (V*j*) right (A*k*) places to V*i*	153*ijk*
V*i*†	V*j*,V*j*>1	Double shift (V*j*) right one place to V*i*	153*ij*0

† Special CAL syntax

Instruction 152*ijk* and its special form shift 128-bit quantities from elements of V*j* by the amount specified in A*k* and enter the result into elements of V*i*. Element *n* of V*j* is concatenated with element *n*+1 and the 128-bit quantity is shifted left by the amount specified in A*k*. The shift is end off with zero fill. The high-order 64 bits of the results are transmitted to element *n* of V*i*.

The VL register determines the number of elements performed by this instruction. The last element of V*j*, as determined by VL, is concatenated with 64 bits of zeros. The 128-bit quantities are shifted left one place if the *k* designator is 0; the special form of the instruction accommodates this feature.

Instruction 153*ijk* and its special form shift 128-bit quantities from elements of V*j* by the amount specified in A*k* and enter the result into elements of V*i*. Element *n*-1 of V*j* is concatenated with element *n* and the 128-bit quantity is shifted right by the amount specified in A*k*. The shift is end off with zero fill. The low-order 64 bits are transmitted to element *n* of V*i*.

The VL register determines the number of elements performed by this instruction. The first element of V*j* is concatenated with 64 bits of zeros. The 128-bit quantities are shifted right one place if the *k* designator is 0; the special form of the instruction accommodates this feature.

Instructions 152*ijk* and 153*ijk* execute in the Vector Shift functional unit.

For these instructions, a warning-level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and *i=j*. A comment level message is issued if NORECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
152541		V5	V4,V4<A1	

Assume the following initial register conditions for the above instruction:

```
              (VL) = 4
              (A1) = 3
    Element 0 of V4 = 0 00000 0000 0000 0000 0007
    Element 1 of V4 = 0 60000 0000 0000 0000 0005
    Element 2 of V4 = 1 00000 0000 0000 0000 0006
    Element 3 of V4 = 1 60000 0000 0000 0000 0007
```

After instruction execution, the first four elements of V5 have been modified as follows:

```
    Element 0 of V5 = 0 00000 0000 0000 0000 0073
    Element 1 of V5 = 0 00000 0000 0000 0000 0054
    Element 2 of V5 = 0 00000 0000 0000 0000 0067
    Element 3 of V5 = 0 00000 0000 0000 0000 0070
```

The remaining elements of V5 are unaltered.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
153026		V0	V2,V2>A6	

Assume the following initial register conditions for the above instruction.

```
                  (VL) = 4
                  (A6) = 3
    Element 0 of V2 = 0 00000 0000 0000 0000 0017
    Element 1 of V2 = 0 60000 0000 0000 0000 0005
    Element 2 of V2 = 1 00000 0000 0000 0000 0006
    Element 3 of V2 = 1 60000 0000 0000 0000 0007
```

After instruction execution, the first four elements of V0 have been modified as follows:

```
    Element 0 of V0 = 0 00000 0000 0000 0000 0001
    Element 1 of V0 = 1 66000 0000 0000 0000 0000
    Element 2 of V0 = 1 30000 0000 0000 0000 0000
    Element 3 of V0 = 1 56000 0000 0000 0000 0000
```

The remaining elements of V0 are unaltered.

Result	Operand	Description	Machine Instruction
V*i*	S*j*+V*k*	Integer sums of (S*j*) and (V*k*) to V*i*	154*ijk*
V*i*	V*j*+V*k*	Integer sums of (V*j*) and (V*k*) to V*i*	155*ijk*
V*i*	S*j*-V*k*	Integer differences of (S*j*) and (V*k*) to V*i*	156*ijk*
V*i*†	-V*k*	Transmit twos complement of (V*k*) to V*i*	156*i*0*k*
V*i*	V*j*-V*k*	Integer differences of (V*j*) less (V*k*) to V*i*	157*ijk*

† Special CAL syntax

Instruction 154*ijk* adds the contents of S*j* to each element of V*k* and enters the results into elements of V*i*. Elements of V*k* are transmitted to V*i* if the *j* designator is 0.

The VL register determines the number of operations performed by this instruction.

Instruction 155*ijk* adds the contents of elements of register V*j* to the contents of corresponding elements of register V*k* and enters the results into elements of register V*i*.

The VL register determines the number of operations performed by this instruction.

Instruction 156*ijk* subtracts the contents of each element of V*k* from the contents of register S*j* and enters the results into elements of register V*i*. The negative (twos complement) of each element of V*k* is transmitted to V*i* if the *j* designator is 0.

The VL register determines the number of operations performed by this instruction.

Instruction 156*i*0*k* transmits the twos complement of the contents of elements of register V*k* to the elements of register V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 157*ijk* subtracts the contents of elements of register V*k* from the contents of corresponding elements of register V*j* and enters the results into elements of register V*i*.

The VL register determines the number of operations performed by this instruction.

Instructions 154*ijk* through 157*ijk* execute in the Vector Integer Add functional unit.

For these instructions, a warning-level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and either *i*=*j* or *i*=*k* (for V registers only). A comment level message is issued if NOVRECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
154213		V2	S1+V3	
155456		V4	V5+V6	
156712		V7	S1-V2	
156102		V1	-V2	
157345		V3	V4-V5	

Result	Operand	Description	Machine Instruction
V*i*	S*j**FV*k*	Floating-point products of (S*j*) and (V*k*) to V*i*	160*ijk*
V*i*	V*j**FV*k*	Floating-point products of (V*j*) and (V*k*) to V*i*	161*ijk*
V*i*	S*j**HV*k*	Half-precision rounded floating-point products of (S*j*) and (V*k*) to V*i*	162*ijk*
V*i*	V*j**HV*k*	Half-precision rounded floating-point products of (V*j*) and (V*k*) to V*i*	163*ijk*
V*i*	S*j**RV*k*	Rounded floating-point products of (S*j*) and (V*k*) to V*i*	164*ijk*
V*i*	V*j**RV*k*	Rounded floating-point products of (V*j*) and (V*k*) to V*i*	165*ijk*
V*i*	S*j**IV*k*	2-floating-point products of (S*j*) and (V*k*) to V*i*	166*ijk*
V*i*	V*j**IV*k*	2-floating-point products of (V*j*) and (V*k*) to V*i*	167*ijk*

Instruction 160*ijk* forms the floating-point products of the contents of S*j* and elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The number of operations performed is determined by the contents of the VL register.

Instruction 161*ijk* forms the floating-point products of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The number of operations performed is determined by the contents of the VL register.

Instruction 162*ijk* forms the half-precision rounded floating-point products of the contents of the S*j* register and the contents of elements of the V*k* register and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The low-order 18 bits of the results are zeroed.

The number of operations performed by this instruction is determined by the contents of the VL register. This instruction can be used in a divide algorithm when only 30 bits of accuracy are required.

Instruction 163*ijk* forms the half-precision rounded floating-point products of the contents of elements of the V*j* register and elements of the V*k* register and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The low-order 18 bits of the results are zeroed.

The VL register determines the number of operations performed by this instruction. This instruction can be used in a divide algorithm when only 30 bits of accuracy are required.

Instruction 164*ijk* forms the rounded floating-point products of the contents of the S*j* register and the contents of elements of V*k* and enters the results into elements of V*i*. The results will not be normalized if either operand is unnormalized. The VL register determines the number of operations performed by this instruction.

Instruction 165*ijk* forms the rounded floating-point products of the contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. The results will not be normalized if either operand is unnormalized. The VL register determines the number of operations performed by this instruction.

Instruction 166*ijk* forms 2 minus the floating-point products of the contents of S*j* and the contents of elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. The VL register determines the number of operations performed by this instruction.

Instruction 167*ijk* forms 2 minus the floating-point products of contents of elements of V*j* and elements of V*k* and enters the results into elements of V*i*. The results are not normalized if either operand is unnormalized. This instruction is used in the divide sequence. The VL register determines the number of operations performed by this instruction.

Instructions 160*ijk* through 167*ijk* execute in the Floating-point Multiply functional unit. For these instructions, a warning-level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and either *i*=*j* or *i*=*k* (for V registers only). A comment-level message is issued if NORECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
160627		V6	S2*FV7	
161123		V1	V2*FV3	
162456		V4	S5*HV6	
163712		V7	V1*HV2	
164314		V3	S1*RV4	
165567		V5	V6*RV7	
166123		V1	S2*IV3	
167456		V4	V5*IV6	

Result	Operand	Description	Machine Instruction
V*i*	S*j*+FV*k*	Floating-point sums of (S*j*) and (V*k*) to V*i*	170*ijk*
V*i*†	+FV*k*	Normalize (V*k*) to V*i*	170*i*0*k*
V*i*	V*j*+FV*k*	Floating-point sums of (V*j*) and (V*k*) to V*i*	171*ijk*
V*i*	S*j*-FV*k*	Floating-point differences of (S*j*) less (V*k*) to V*i*	172*ijk*
V*i*†	-FV*k*	Transmit normalized negative of (V*k*) to V*i*	172*i*0*k*
V*i*	V*j*-FV*k*	Floating-point differences of (V*j*) less (V*k*) to V*i*	173*ijk*

† Special CAL syntax

Instruction 170*ijk* forms the floating-point sums of the contents of S*j* and elements of register V*k* to elements of register V*i*. The results are normalized even if the operands are unnormalized. The VL register determines the number of operations performed by this instruction.

The special form of the instruction (170*i*0*k*) normalizes the contents of the elements of V*k* and enters the results into elements of V*i*.

Instruction 171*ijk* forms the floating-point sums of the contents of elements of V*j* and elements of V*k* and enters the results into the elements of register V*i*. The results are normalized even if the operands are unnormalized. The number of operations performed is determined by the contents of the VL register.

Instruction 172*ijk* forms the floating-point differences of the contents of S*j* and elements of register V*k* and enters the results into register V*i*. The results are normalized even if the operands are unnormalized. The negatives (twos complements) of floating-point quantities in elements of V*k* are transmitted to V*i* if the *j* designator is 0. The special form (172*i*0*k*) accommodates this special case. The number of operations performed is determined by the contents of the VL register.

Instruction 173*ijk* forms the floating-point differences of the contents of elements of register V*j* less the contents of elements of registers V*k* and enters the results into elements of register V*i*. The results are normalized even if the operands are unnormalized. The VL register determines the number of operations performed by this instruction.

Instructions 170*ijk* through 173*ijk* execute in the Floating-point Add functional unit. For these instructions, a warning level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and either *i*=*j* or *i*=*k* (for V registers only). A comment level message is issued if NORECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
170712		V7	S1+FV2	
170501		V5	+FV1	; Normalize (V1) ; to V5
171234		V2	V3+FV4	
172516		V5	S1-FV6	
173712		V7	V1-FV2	

INSTRUCTION 174

Result	Operand	Description	Machine Instruction
V*i*	/HV*j*	Floating-point reciprocal approximation of (V*j*) to V*i*	174*ij*0

Instruction 174*ij*0 forms the approximations to the reciprocals of the floating-point values in elements of V*j* and enters the results into elements of V*i*. The results are meaningless if the contents of elements are unnormalized or 0. This instruction is used in the divide sequence. The VL register determines the number of operations performed by this instruction.

Instruction 174*ij*0 executes in the Floating-point Reciprocal functional unit. For these instructions, a warning level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and *i*=*j*. A comment level message is issued if NORECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

```
|Code Generated          |Location|Result     |Operand         |Comment
|                        |1       |10         |20              |35
|                        |        |           |                |
|                        |*       |Divide elements of V1 by elements of V2;
|                        |*       |Result to V6                |
|174320                  |        |V3         |/HV2            |
|                        |        |           |                |
|161413                  |        |V4         |V1*FV3          |
|                        |        |           |                |
|167532                  |        |V5         |V3*IV2          |
|                        |        |           |                |
|161645                  |        |V6         |V4*FV5          |
|                        |        |           |                |
|                        |*       |Divide elements of V1 by elements of V2;
|                        |*       |Results accurate to 30 bits, result to V6
|174320                  |        |V3         |/HV2            |
|                        |        |           |                |
|165613                  |        |V6         |V1*HV3          |
|                        |        |           |                |
|                        |*       |Divide S1 by elements of V2; Result to V6
|174320                  |        |V3         |/HV2            |
|                        |        |           |                |
```

Example (continued):

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
160413		V4	S1*FV3	
167532		V5	V3*IV2	
161645		V6	V4*FV5	

Result	Operand	Description	Machine Instruction
V*i*†	PV*j*	Population count of (V*j*) to (V*i*)	174*ij*1
V*i*†	QV*j*	Population count parity of (V*j*) to (V*i*)	174*ij*2

† Vector Population Count; this instruction is available through the logical trait VPOP specified on the CPU parameter of the CAL invocation statement.

Instruction 174*ij*1 counts the number of 1 bits in the elements of register V*j* and enters the result into the elements of register V*i*. The VL register determines the number of elements performed by this instruction.

Instruction 174*ij*2 enters a 0 or 1 into the elements of V*i* depending on whether the elements of V*j* have an even or odd number of 1 bits. A 0 is entered into element *n* of V*i* if there is an even number of 1 bits in element *n* of V*j*; a 1 is entered into element *n* of V*i* if there is an odd number of 1 bits in element *n* of V*j*. The number of elements involved is determined by the VL register.

Instructions 174*ij*1 and 174*ij*2 execute in the Reciprocal Approximation functional unit. For these instructions, a warning level message is issued if the logical trait VRECUR is specified on the CPU parameter of the CAL invocation statement and *i*=*j*. A comment level message is issued if NORECUR is specified on the CPU parameter of the CAL invocation statement.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
174311		V3	PV1	; Pop count of ; V1 to V3
174522		V5	QV2	; Pop count ; parity of V2 ; to V5

Result	Operand	Description	Machine Instruction
VM	V*j*,Z	Set VM bits for zero elements of V*j*	1750*j*0
VM	V*j*,N	Set VM bits for nonzero elements of V*j*	1750*j*1
VM	V*j*,P	Set VM bits for positive elements of V*j*	1750*j*2
VM	V*j*,M	Set VM bits for negative elements of V*j*	1750*j*3
V*i*,VM†	V*j*,Z	Set VM bits and register V*i* to V*j*, for zero elements of V*j*	175*ij*4
V*i*,VM†	V*j*,N	Set VM bits and register V*i* to V*j*, for nonzero elements of V*j*	175*ij*5
V*i*,VM†	V*j*,P	Set VM bits and register V*i* to V*j*, for positive elements of V*j*	175*ij*6
V*i*,VM†	V*j*,M	Set VM bits and register V*i* to V*j*, for negative elements of V*j*	175*ij*7

† This instruction is available through the logical trait CIGS specified on the CPU parameter of the CAL invocation statement.

Instructions 1750*j*0 through 1750*j*3 create a mask in the VM register. The 64 bits of the VM register correspond to the 64 elements of V*j*. Elements of V*j* are tested for the specified condition. If the condition is true for an element, the corresponding bit is set to 1 in the VM register. If the condition is not true, the bit is zeroed.

The number of elements tested is determined by the contents of the VL register; however, the entire VM register is zeroed before elements of V*j* are tested. If the contents of an element is 0, it is considered positive. Element 0 corresponds to bit 0, element 1 to bit 1, and so on, from left-to-right in the register.

Instructions 175*ij*4 through 175*ij*7 create an identical vector mask as in the above instructions, and in addition create a compressed index list in register V*i* based on the results of testing the contents of the elements of register V*j*.

These instructions execute in the Vector Logical functional unit.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
175050		VM	V5,Z	
175061		VM	V6,N	
175072		VM	V7,P	
175013		VM	V1,M	

Result	Operand	Description	Machine Instruction
V*i*	,A0,A*k*	Read from memory starting at (A0) increased by (A*k*) and load into V*i*	176*i*0*k*
V*i*†	,A0,1	Read from consecutive memory addresses starting with (A0) and load into V*i*	176*i*00
V*i*††	,A0,V*k*	Read from memory using memory address (A0) + (V*k*) and load into V*i*	176*i*1*k*
,A0,A*k*	V*j*	Store (V*j*) to memory starting at (A0) increased by (A*k*)	1770*jk*
,A0,1	V*j*	Store (V*j*) to memory in consecutive addresses starting with (A0)	1770*j*0
,A0,V*k*††	V*j*	Store (V*j*) to memory using memory address (A0) + (V*k*)	1771*jk*

† Special CAL syntax
†† This instruction is available through the logical trait CIGS specified on the CPU parameter of the CAL invocation statement.

Instruction 176*i*0*k* and 176*i*00 load words into elements of register V*i* directly from memory. A0 contains the starting memory address. This address is increased by the contents of register A*k* for each word transmitted. The contents of A*k* can be positive or negative allowing both forward and backward streams of references. If the *k* designator is 0 or if 1 replaces A*k* in the operand field of the instruction, the address is increased by 1.

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

For instruction 176*i*1*k*, register elements begin with 0 and are increased by 1 for each transfer. The low-order 24 bits of each element of V*k* contain a signed 24-bit integer which is added to (A0) to obtain the current memory address.

The VL register determines the number of words transferred.

Instructions 1770*jk* and 1770*j*0 store words from elements of register V*j* directly into memory. A0 contains the starting memory address. This address is increased by the contents of register A*k* for each word transmitted. The contents of A*k* can be positive or negative allowing both forward and backward streams of references. If the *k* designator is 0 or if 1 replaces A*k* in the result field of the instruction, the address is increased by 1.

The VL register determines the number of elements transferred.

For instruction 1771*jk*, register elements begin with 0 and are increased by 1 for each transfer. The low-order 24 bits of each element of V*k* contains a signed 24-bit integer which is added to (A0) to obtain the current memory address.

The VL register determines the number of elements transferred.

Example:

Code Generated	Location	Result	Operand	Comment
	1	10	20	35
176201		V2	,A0,A1	
176500		V5	,A0,1	
177032		,A0,A2	V3	
177030		,A0,1	V3	

SYMBOLIC INSTRUCTION SUMMARY

This section contains symbolic instructions summary charts for the CRAY X-MP computer system. It also lists the functional units for the CRAY X-MP computer system.

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 Unit	Instructions
Address Integer Add	030, 031
Address Integer Multiply	032
Scalar Integer Add	060, 061
Scalar Logical	042-051
Scalar Shift	052-055
	056, 057
Scalar Pop/Parity/	026
Leading Zero	027
Vector Integer Add	154-157
Vector Logical	140-147, 175
Second Vector Logical	140-145
Vector Shift	150, 151, 153
	152
Vector Pop/Parity	174*ij*1, 174*ij*2
Floating-point Add	062, 063, 170-173
Floating-point Multiply	064-067, 160-167
Floating-point Reciprocal	070, 174*ij*0
Memory (Scalar)	100-130
	100-130
Memory (Vector)	176, 177

† Multiprocessor CRAY X-MP computer systems

†† Single-processor CRAY X-MP computer systems

Logical Operations

Si	Sj&Sk	Vi	Sj&Vk	Vi	Vj&Vk
Si	Sj&SB				
Si	SB&Sj				
Si	#Sk&Sj				
Si	#SB&Sj				
Si	Sj!Sk	Vi	Sj!Vk	Vi	Vj!Vk
Si	Sj!SB				
Si	SB!Sj				
Si	Sj\Sk	Vi	Sj\Vk	Vi	Vj\Vk
Si	Sj\SB				
Si	SB\Sj				
Si	#Sj\Sk				
Si	#Sj\SB				
Si	#SB\Sj				
		VM	Vj,Z	Vi,VM	Vj,Z
		VM	Vj,N	Vi,VM	Vj,N
		VM	Vj,P	Vi,VM	Vj,P
		VM	Vj,M	Vi,VM	Vj,M
Si	Sj!Si&Sk				
Si	Sj!Si&SB	Vi	Sj!Vk&VM	Vi	Vj!Vk&VM
		Vi	#VM&Vk		

Floating-point Operations

EFI					
DFI					
Si	Sj+FSk	Vi	Sj+FVk	Vi	Vj+FVk
Si	+FSk	Vi	+FVk		
Si	Sj-FSk	Vi	Sj-FVk	Vi	Vj-FVk
Si	-FSk	Vi	-FVk		
Si	Sj*FSk	Vi	Sj*FVk	Vi	Vj*FVk
Si	Sj*HSk	Vi	Sj*HVk	Vi	Vj*HVk
Si	Sj*RSk	Vi	Sj*RVk	Vi	Vj*RVk
Si	Sj*ISk	Vi	Sj*IVk	Vi	Vj*IVk
Si	/HSj			Vi	/HVj

Shift Instructions

```
S0   Si<exp        S0   Si>exp
Si   Si<exp        Si   Si>exp

Si   Si,Sj<Ak      Si   Sj,Si>Ak
Si   Si,Sj<1       Si   Sj,Si>1
Si   Si<Ak         Si   Si>Ak

Vi   Vj<Ak         Vi   Vj>Ak
Vi   Vj<1          Vi   Vj>1

Vi   Vj,Vj<Ak      Vi   Vj,Vj>Ak
Vi   Vj,Vj<1       Vi   Vj,Vj>1
```

Register Entry Instrucitons

```
Ah   exp           Si   <exp
Ai   -1            Si   #>exp
                   Si   >exp
Si   exp           Si   #<exp

Si   0             Si   SB
Si   1             Si   #SB
Si   -1
Si   1             Vi,Ak  0
Si   2             Vi   0
Si   4
Si   0.4           SMjk  1,TS
Si   0.6           SMjk  0
                   SMjk  1
```

Program Branches and Exits

```
J    exp
J    Bjk

JAZ  exp           JSZ  exp
JAN  exp           JSN  exp
JAP  exp           JSP  exp
JAM  exp           JSM  exp

R    exp

EX                 ERR
PASS
```

Bit Count Instructions

```
Ai   PSj           Vi   PVj
Ai   QSj           Vi   QVj
Ai   ZSj
```

Monitor Operations

```
CA,Aj   Ak         CCI
CL,Aj   Ak         ECI
CI,Aj              DCI
MC,Aj              ERI
XA      Aj         DRI
RT      Sj         CLN   exp
PCI     Sj
SIPI    exp
SIPI
CIPI
```

Integer Arithmetic Operations

```
Ai   Aj+Ak
Ai   Aj+1
Ai   Aj-Ak
Ai   Aj-1
Ai   Aj*Ak

Si   Sj+Sk      Vi   Sj+Vk      Vi   Vj+Vk
Si   Sj-Sk      Vi   Sj-Vk      Vi   Vj-Vk
```

Inter-register Transfers

A*i*	A*k*	S*i*	S*k*
A*i*	-A*k*	S*i*	-S*k*
		S*i*	#S*k*
A*i*	S*j*	S*i*	A*k*
		S*i*	+A*k*
A*i*	VL	S*i*	+FA*k*
A*i*	B*jk*	S*i*	T*jk*
A*i*	SB*j*	S*i*	ST*j*
A*i*	CI	S*i*	V*j*,A*k*
A*i*	CA,A*j*	S*i*	VM
A*i*	CE,A*j*	S*i*	RT
		S*i*	SM
		S*i*	SR*j*
B*jk*	A*i*	T*jk*	S*i*
SB*j*	A*i*	ST*j*	S*i*
		V*i*	V*k*
		V*i*	-V*k*
		V*i*,A*k*	S*j*
VL	A*k*	VM	S*j*
VL	1	VM	0
		SM	S*i*

Memory Transfers

DBM
EBM
CMR

(store)		(load)	
,A0	B*jk*,A*i*	B*jk*,A*i*	,A0
0,A0	B*jk*,A*i*	B*jk*,A*i*	0,A0
,A0	T*jk*,A*i*	T*jk*,A*i*	,A0
0,A0	T*jk*,A*i*	T*jk*,A*i*	0,A0
exp,A*h*	A*i*	A*i*	*exp*,A*h*
exp,0	A*i*	A*i*	*exp*,0
exp,	A*i*	A*i*	*exp*,
,A*h*	A*i*	A*i*	,A*h*
exp,A*h*	S*i*	S*i*	*exp*,A*h*
exp,0	S*i*	S*i*	*exp*,0
exp,	S*i*	S*i*	*exp*,
,A*h*	S*i*	S*i*	,A*h*
,A0,A*k*	V*j*	V*i*	,A0,A*k*
,A0,1	V*j*	V*i*	,A0,1
,A0,V*k*	V*j*	V*i*	,A0,V*k*

Register	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}

Logical Operators	
& AND	0101 / $\underline{1100}$ / 0100
! OR	0101 / $\underline{1100}$ / 1101
\ XOR	0101 / $\underline{1100}$ / 1001

FUNCTIONAL INSTRUCTION SUMMARY

This subsection contains an instruction summary, listed by function, for the CRAY X-MP computer system. A detailed description can be found on the referenced pages.

REGISTER ENTRY INSTRUCTIONS

Instructions in this category provide for entering values such as constants, expression values, or masks directly into registers.

Entries into A registers

Machine Instruction	CAL	Description	Page
01*hijkm*	A*h exp*	Transmit *ijkm* to A*h*; where the high-order bit of *i* is 1	3-31
020*ijkm* or 021*ijkm* or 022*ijk*	A*i exp*	Enter *exp* into A*i*	3-32
031*i*00†	A*i* -1	Enter -1 into A*i*	3-38

Entries into S registers

Machine Instruction	CAL	Description	Page
040*ijkm* or 041*ijkm*	S*i exp*	Enter *exp* into S*i*	3-45
042*i*00†	S*i* -1	Enter -1 into S*i*	3-46
042*ijk*	S*i* <*exp*	Form ones mask in S*i* from right	3-46
042*ijk*†	S*i* #>*exp*	Form zeros mask in S*i* from left	3-46

† Special CAL syntax

Machine Instruction	CAL	Description	Page
042*i*77†	S*i* 1	Enter 1 into S*i*	3-46
043*i*00†	S*i* 0	Clear S*i*	3-46
043*ijk*	S*i* >*exp*	Form ones mask in S*i* from left	3-46
043*ijk*†	S*i* #<*exp*	Form zeros mask in S*i* from right	3-46
047*i*00†	S*i* #SB	Enter ones complement of sign bit in S*i*	3-49
051*i*00†	S*i* SB	Enter sign bit into S*i*	3-49
071*i*30	S*i* 0.6	Enter 0.75*(2**48) into S*i* as normalized floating-point constant	3-65
071*i*40	S*i* 0.4	Enter 0.5 into S*i* as normalized floating-point constant	3-65
071*i*50	S*i* 1.	Enter 1 into S*i* as normalized floating-point constant	3-65
071*i*60	S*i* 2.	Enter 2 into S*i* as normalized floating-point constant	3-65
071*i*70	S*i* 4.	Enter 4 into S*i* as normalized floating-point constant	3-65

Entries into V registers

Machine Instruction	CAL	Description	Page
077*i*0*k*†	V*i*,A*k* 0	Clear element (A*k*) of register V*i*	3-70
145*iii*†	V*i* 0	Clear V*i*	3-74

† Special CAL syntax

Entries into Semaphore registers

Machine Instruction	CAL	Description	Page
0034*jk*	SM*jk* 1,TS	Test and set semaphore *jk*, $0 \leq jk \leq 31$ (decimal)	3-23
0036*jk*	SM*jk* 0	Clear semaphore *jk*, $0 \leq jk \leq 31$ (decimal)	3-23
0037*jk*	SM*jk* 1	Set semaphore *jk*, $0 \leq jk \leq 31$ (decimal)	3-23

INTER-REGISTER TRANSFER INSTRUCTIONS

Instructions in this category provide for transferring the contents of one register to another register. In some cases, the register contents can be complemented, converted to floating-point format, or sign extended as a function of the transfer.

Transfers to A registers

Machine Instruction	CAL	Description	Page
023*ij*0	A*i* S*j*	Transmit (S*j*) to A*i*	3-34
023*i*01	A*i* VL	Transmit (VL) to A*i*	3-34
024*ijk*	A*i* B*jk*	Transmit (B*jk*) to A*i*	3-35
026*ij*7	A*i* SB*j*	Transfer (SB*j*) to A*i*	3-36
030*i*0*k*†	A*i* A*k*	Transmit (A*k*) to A*i*	3-38
031*i*0*k*†	A*i* -A*k*	Transmit negative of (A*k*) to A*i*	3-38
033*i*00	A*i* CI	Channel number of highest priority interrupt request to A*i*	3-41
033*ij*0	A*i* CA,A*j*	Address of channel (A*j*) to A*i* ($j \neq 0$)	3-41
033*ij*1	A*i* CE,A*j*	Error flag of channel (A*j*) to A*i*	3-41

† Special CAL syntax

Transfers to S registers

Machine Instruction	CAL	Description	Page
025*ijk*	B*jk* A*i*	Transmit (A*i*) to B*jk*	3-35
027*ij*7	SB*j* A*i*	Transfer (A*i*) to SB*j*	3-37
047*i*0*k*†	S*i* #S*k*	Transmit ones complement of (S*k*) to S*i*	3-49
051*i*0*k*†	S*i* S*k*	Transmit (S*k*) to S*i*	3-49
061*i*0*k*†	S*i* -S*k*	Transmit negative of (S*k*) to S*i*	3-58
071*i*0*k*	S*i* A*k*	Transmit (A*k*) to S*i* without sign extension	3-65
071*i*1*k*	S*i* +A*k*	Transmit (A*k*) to S*i* with sign extension	3-65
071*i*2*k*	S*i* +FA*k*	Transmit (A*k*) to S*i* as an unnormalized floating-point value	3-65
072*i*00	S*i* RT	Transmit (RTC) to S*i*	3-67
072*i*02	S*i* SM	Read semaphore to S*i*	3-67
072*ij*3	S*i* ST*j*	Read (ST*j*) register to S*i*	3-67
073*i*00	S*i* VM	Transmit (VM) to S*i*	3-67
073*i*01	S*i* SR0	Transmit (SR0) to S*i*	3-67
073*ij*3	ST*j* S*i*	Transfer (S*i*) to ST*j*	3-67
074*ijk*	S*i* T*jk*	Transmit (T*jk*) to S*i*	3-67
075*ijk*	T*jk* S*i*	Transmit (S*i*) to T*jk*	3-67
076*ijk*	S*i* V*j*,A*k*	Transmit (V*j*, element (A*k*)) to S*i*	3-70

† Special CAL syntax

Transfers to V registers

Machine Instruction	CAL	Description	Page
077*ijk*	V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*)	3-70
142*i*0*k*†	V*i* V*k*	Transmit (V*k*) to V*i*	3-74
156*i*0*k*†	V*i* -V*k*	Transmit twos complement of (V*k*) to V*i*	3-84

Transfer to Vector Mask register

Machine Instruction	CAL	Description	Page
0030*j*0	VM S*j*	Transmit (S*j*) to VM	3-23
003000†	VM 0	Clear VM	3-23

Transfer to Vector Length register

Machine Instruction	CAL	Description	Page
00200*k*	VL A*k*	Transmit (A*k*) to VL	3-19
002000†	VL 1	Enter 1 into VL	3-19

Transfer to Semaphore register

Machine Instruction	CAL	Description	Page
073*i*02	SM S*i*	Load semaphores from S*i*	3-67

† Special CAL syntax

MEMORY TRANSFERS

This category contains instructions that transfer data between registers and memory, enable and disable concurrent block memory transfers, and assure completion of memory references.

Bidirectional memory transfers

Machine Instruction	CAL	Description	Page
002500	DBM	Disable bidirectional memory transfers	3-21
002600	EBM	Enable bidirectional memory transfers	3-21

Memory references

Machine Instruction	CAL	Description	Page
002700	CMR	Complete memory references	3-21

Stores

Machine Instruction	CAL	Description	Page
035*ijk*	,A0 B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	3-42
035*ijk*†	0,A0 B*jk*,A*i*	Store (A*i*) words starting at B*jk* to memory starting at (A0)	3-42
037*ijk*	,A0 T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	3-42
037*ijk*†	0,A0 T*jk*,A*i*	Store (A*i*) words starting at T*jk* to memory starting at (A0)	3-42

† Special CAL syntax

Machine Instruction	CAL	Description	Page
11*hijkm*	*exp*,A*h* A*i*	Store (A*i*) to (A*h*) + *exp*	3-71
11*hi*000†	,A*h* A*i*	Store (A*i*) to (A*h*)	3-71
110*ijkm*†	*exp*,0 A*i*	Store (A*i*) to *exp*	3-71
110*ijkm*†	*exp*, A*i*	Store (A*i*) to *exp*	3-71
13*hijkm*	*exp*,A*h* S*i*	Store (S*i*) to (A*h*) + *exp*	3-71
130*ijkm*†	*exp*,0 S*i*	Store (S*i*) to *exp*	3-71
130*ijkm*†	*exp*, S*i*	Store (S*i*) to *exp*	3-71
13*hi*000†	,A*h* S*i*	Store (S*i*) to (A*h*)	3-71
1770*jk*	,A0,A*k* V*j*	Store (V*j*) to memory starting at (A0) increased by (A*k*)	3-96
1770*j*0	,A0,1 V*j*	Store (V*j*) to memory in consecutive addresses starting with (A0)	3-96
1771*jk*	,A0,V*k* V*j*	Store (V*j*) to memory using memory address (A0) + (V*k*)	3-96

Loads

Machine Instruction	CAL	Description	Page
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	3-42
034*ijk*†	B*jk*,A*i* 0,A0	Read (A*i*) words starting at B*jk* from memory starting at (A0)	3-42
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	3-42
036*ijk*†	T*jk*,A*i* 0,A0	Read (A*i*) words starting at T*jk* from memory starting at (A0)	3-42

† Special CAL syntax

Machine Instruction	CAL	Description	Page
10*hijkm*	A*i* *exp*,A*h*	Read from ((A*h*) + *exp*) to A*i*	3-71
10*hi*000†	A*i* ,A*h*	Read from (A*h*) to A*i*	3-71
100*ijkm*†	A*i* *exp*,0	Read from (*exp*) to A*i*	3-71
100*ijkm*†	A*i* *exp*,	Read from (*exp*) to A*i*	3-71
12*hijkm*	S*i* *exp*,A*h*	Read from ((A*i*) + *exp*) to S*i*	3-71
120*ijkm*†	S*i* *exp*,0	Read from (*exp*) to S*i*	3-71
120*ijkm*†	S*i* *exp*	Read from (*exp*) to S*i*	3-71
12*hi*000†	S*i* ,A*h*	Read from (A*h*) to S*i*	3-71
176*i*0*k*	V*i* ,A0,A*k*	Read from memory starting at (A0) increased by (A*k*) and load into V*i*	3-96
176*i*00†	V*i* ,A0,1	Read from consecutive memory addresses starting with (A0) and load into V*i*	3-96
176*i*1*k*	V*i* ,A0,V*k*	Read from memory using memory address (A0) + (V*k*) and load into V*i*	3-96

INTEGER ARITHMETIC OPERATIONS

Integer arithmetic operations obtain operands from registers and return results to registers. No direct memory references are allowed.

The assembler recognizes several special syntax forms for increasing or decreasing register contents, such as the operands A*i*+1 and A*i*-1; however, these references actually result in register references such that the 1 becomes a reference to A*k* with *k*=0.

All integer arithmetic, whether 24-bit or 64-bit, is twos complement and is so represented in the registers. The Address Add functional unit and

† Special CAL syntax

Address Multiply functional unit perform 24-bit arithmetic. The Scalar Add functional unit and the Vector Add functional unit perform 64-bit arithmetic.

No overflow is detected by Integer functional units.

Multiplication of two fractional operands can be accomplished using the floating-point multiply instruction. The Floating-point Multiply functional unit recognizes conditions in which both operands have zero exponents as a special case and returns the high-order 48 bits of the result as an unnormalized fraction. Division of integers requires that they first be converted to floating-point format and then divided using the floating-point units.

24-bit integer arithmetic

Machine Instruction	CAL	Description	Page
030*ijk*	A*i* A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*	3-38
030*ij*0†	A*i* A*j*+1	Integer sum of (A*j*) and 1 to A*i*	3-38
031*ijk*	A*i* A*j*-A*k*	Integer difference of (A*j*) less (A*k*) to A*i*	3-38
031*ij*0†	A*i* A*j*-1	Integer difference of (A*j*) less 1 to A*i*	3-38
032*ijk*	A*i* A*j**A*k*	Integer product of (A*j*) and (A*k*) to A*i*	3-40

64-bit integer arithmetic

Machine Instruction	CAL	Description	Page
060*ijk*	S*i* S*j*+S*k*	Integer sum of (S*j*) and (S*k*) to S*i*	3-58
061*ijk*	S*i* S*j*-S*k*	Integer difference of (S*j*) less (S*k*) to S*i*	3-58

† Special CAL syntax

Machine Instruction	CAL	Description	Page
154*ijk*	V*i* S*j*+V*k*	Integer sums of (S*j*) and (V*k*) to V*i*	3-84
155*ijk*	V*i* V*j*+V*k*	Integer sums of (V*j*) and (V*k*) to V*i*	3-84
156*ijk*	V*i* S*j*-V*k*	Integer differences of (S*j*) and (V*k*) to V*i*	3-84
157*ijk*	V*i* V*j*-V*k*	Integer differences of (V*j*) less (V*k*) to V*i*	3-84

FLOATING-POINT ARITHMETIC

All floating-point arithmetic operations use registers as the source of operands and return results to registers.

Floating-point numbers are represented in a standard format throughout the CPU. This format is a packed representation of a binary coefficient and an exponent or power of 2. The coefficient is a 48-bit signed fraction. The sign of the coefficient is separated from the rest of the coefficient. Since the coefficient is signed magnitude, it is not complemented for negative values.

Floating-point range errors

Machine Instruction	CAL	Description	Page
002100	EFI	Enable floating-point interrupt	3-21
002200	DFI	Disable floating-point interrupt	3-21

Floating-point addition and subtraction

Machine Instruction	CAL	Description	Page
062*ijk*	S*i* S*j*+FS*k*	Floating-point sum of (S*j*) and (S*k*) to S*i*	3-59

Machine Instruction	CAL	Description	Page
062*i*0*k*†	S*i* +FS*k*	Normalize (S*k*) to S*i*	3-59
063*ijk*	S*i* S*j*-FS*k*	Floating-point difference of (S*j*) less (S*k*) to S*i*	3-59
063*i*0*k*†	S*i* -FS*k*	Transmit the negative of (S*k*) as a normalized floating-point value	3-59
170*ijk*	V*i* S*j*+FV*k*	Floating-point sums of (S*j*) and (V*k*) to V*i*	3-89
170*i*0*k*†	V*i* +FV*k*	Normalize (V*k*) to V*i*	3-89
171*ijk*	V*i* V*j*+FV*k*	Floating-point sums of (V*j*) and (V*k*) to V*i*	3-89
172*ijk*	V*i* S*j*-FV*k*	Floating-point differences of (S*j*) less (V*k*) to V*i*	3-89
172*i*0*k*†	V*i* -FV*k*	Transmit normalized negative of (V*k*) to V*i*	3-89
173*ijk*	V*i* V*j*-FV*k*	Floating-point differences of (V*j*) less (V*k*) to V*i*	3-89

Floating-point multiplication

Machine Instruction	CAL	Description	Page
064*ijk*	S*i* S*j**FS*k*	Floating-point product of (S*j*) and (S*k*) to S*i*	3-61
065*ijk*	S*i* S*j**HS*k*	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*	3-61
066*ijk*	S*i* S*j**RS*k*	Rounded floating-point product of (S*j*) and (S*k*) to S*i*	3-61

† Special CAL syntax

Machine Instruction	CAL	Description	Page
160*ijk*	V*i* S*j**FV*k*	Floating-point products of (S*j*) and (V*k*) to V*i*	3-86
161*ijk*	V*i* V*j**FV*k*	Floating-point products of (V*j*) and (V*k*) to V*i*	3-86
162*ijk*	V*i* S*j**HV*k*	Half-precision rounded floating-point products of (S*j*) and (V*k*) to V*i*	3-86
163*ijk*	V*i* V*j**HV*k*	Half-precision rounded floating-point products of (V*j*) and (V*k*) to V*i*	3-86
164*ijk*	V*i* S*j**RV*k*	Rounded floating-point products of (S*j*) and (V*k*) to V*i*	3-86
165*ijk*	V*i* V*j**RV*k*	Rounded floating-point products of (V*j*) and (V*k*) to V*i*	3-86

Reciprocal iteration

Machine Instruction	CAL	Description	Page
067*ijk*	S*i* S*j**IS*k*	2-floating-point product of (S*j*) and (S*k*) to S*i*	3-61
166*ijk*	V*i* S*j**IV*k*	2-floating-point products of (S*j*) and (V*k*) to V*i*	3-86
167*ijk*	V*i* V*j**IV*k*	2-floating-point products of (V*j*) and (V*k*) to V*i*	3-86

Reciprocal approximation

Machine Instruction	CAL	Description	Page
070*ij*0	S*i* /HS*j*	Floating-point reciprocal approximation of (S*j*) to S*i*	3-63
174*ij*0	V*i* /HV*j*	Floating-point reciprocal approximation of (V*j*) to V*i*	3-91

LOGICAL OPERATIONS

The Scalar and Vector Logical functional units perform bit-by-bit manipulation of 64-bit quantities. Operations provide for logical products, logical differences, logical sums, logical equivalence, and merges.

A logical product (& operator) is the AND function.

A logical difference (\ operator) is the EXCLUSIVE OR function.

A logical sum (! operator) is the INCLUSIVE OR function.

A logical merge combines two operands depending on a ones mask in a third operand. The result is defined by (operand 2 & mask)!(operand 1 & #mask).

Logical products

Machine Instruction	CAL	Description	Page
044*ijk*	S*i* S*j*&S*k*	Logical product of (S*j*) and (S*k*) to S*i*	3-48
044*ij*0†	S*i* S*j*&SB	Sign bit of (S*j*) to S*i*	3-48
044*ij*0†	S*i* SB&S*j*	Sign bit of (S*j*) to S*i*; *j*≠0	3-48
045*ijk*	S*i* #S*k*&S*j*	Logical product of (S*j*) and #(S*k*) to S*i*	3-48
045*ij*0†	S*i* #SB&S*j*	(S*j*) with sign bit cleared to S*i*	3-48
140*ijk*	V*i* S*j*&V*k*	Logical products of (S*j*) and (V*k*) to V*i*	3-74
141*ijk*	V*i* V*j*&V*k*	Logical products of (V*j*) and (V*k*) to V*i*	3-74

† Special CAL syntax

Logical sums

Machine Instruction	CAL	Description	Page
051*ijk*	S*i* S*j*!S*k*	Logical sum of (S*j*) and (S*k*) to S*i*	3-49
051*ij*0†	S*i* S*j*!SB	Logical sum of (S*j*) and sign bit to S*i*	3-49
051*ij*0†	S*i* SB!S*j*	Logical sum of sign bit and (S*j*) to S*i*; *j*≠0	3-49
142*ijk*	V*i* S*j*!V*k*	Logical sums of (S*j*) and (V*k*) to V*i*	3-74
143*ijk*	V*i* V*j*!V*k*	Logical sums of (V*j*) and (V*k*) to V*i*	3-74

Logical differences

Machine Instruction	CAL	Description	Page
046*ijk*	S*i* S*j*\S*k*	Logical differences of (S*j*) and (S*k*) to S*i*	3-48
046*ij*0†	S*i* S*j*\SB	Enter (S*j*) into S*i* with sign bit toggled	3-48
046*ij*0†	S*i* SB\S*j*	Enter (S*j*) into S*i* with sign bit toggled; *j*≠0	3-48
144*ijk*	V*i* S*j*\V*k*	Logical differences of (S*j*) and (V*k*) to V*i*	3-74
145*ijk*	V*i* V*j*\V*k*	Logical differences of (V*j*) and (V*k*) to V*i*	3-74

† Special CAL syntax

Logical equivalence

Machine Instruction	CAL	Description	Page
047*ijk*	S*i* #S*j*\S*k*	Logical equivalence of (S*j*) and (S*k*) to S*i*	3-48
047*ij*0†	S*i* #S*j*\SB	Logical equivalence of (S*j*) and sign bit to S*i*	3-48
047*ij*0†	S*i* #SB\S*j*	Logical equivalence of sign bit and (S*j*) to S*i*; *j*≠0	3-48

Vector mask

Machine Instruction	CAL	Description	Page
1750*j*0	VM V*j*,Z	Set VM bits for zero elements of V*j*	3-94
1750*j*1	VM V*j*,N	Set VM bits for nonzero elements of V*j*	3-94
1750*j*2	VM V*j*,P	Set VM bits for positive elements of V*j*	3-94
1750*j*3	VM V*j*,M	Set VM bits for negative elements of V*j*	3-94
175*ij*4	V*i*,VM V*j*,Z	Set VM bits and register V*i* to V*j*, for zero elements of V*j*	3-94
175*ij*5	V*i*,VM V*j*,N	Set VM bits and register V*i* to V*j*, for nonzero elements of V*j*	3-94
175*ij*6	V*i*,VM V*j*,P	Set VM bits and register V*i* to V*j*, for positive elements of V*j*	3-94
175*ij*7	V*i*,VM V*j*,M	Set VM bits and register V*i* to V*j*, for negative elements of V*j*	3-94

† Special CAL syntax

Merge

Machine Instruction	CAL	Description	Page
050*ijk*	S*i* S*j*!S*i*&S*k*	Scalar merge of (S*i*) and (S*j*) to S*i*	3-49
050*ij*0†	S*i* S*j*!S*i*&SB	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*	3-49
146*ijk*	V*i* S*j*!V*k*&VM	Vector merge of (S*j*) and (V*k*) to V*i*	3-74
146*i*0*k*†	V*i* #VM&V*k*	Vector merge of (V*k*) and zero to V*i*	3-74
147*ijk*	V*i* V*j*!V*k*&VM	Vector merge of (V*j*) and (V*k*) to V*i*	3-74

SHIFT INSTRUCTIONS

The Scalar Shift functional unit and Vector Shift functional unit shift 64-bit quantities or 128-bit quantities. A 128-bit quantity is formed by concatenating two 64-bit quantities. The number of bits a value is shifted left or right is determined by the value of an expression for some instructions and by the contents of an A register for other instructions. If the count is specified by an expression, the value of the expression must not exceed 64.

Machine Instruction	CAL	Description	Page
052*ijk*	S0 S*i*<*exp*	Shift (S*i*) left *exp* places to S0	3-54
053*ijk*	S0 S*i*>*exp*	Shift (S*i*) right *exp* places to S0	3-54
054*ijk*	S*i* S*i*<*exp*	Shift (S*i*) left *exp* places to S*i*	3-54
055*ijk*	S*i* S*i*>*exp*	Shift (S*i*) right *exp* places to S*i*	3-54
056*ijk*	S*i* S*i*,S*j*<A*k*	Left shift by (A*k*) of (S*i*) and (S*j*) to S*i*	3-56

† Special CAL syntax

Machine Instruction	CAL	Description	Page
056*ij*0†	S*i* S*i*,S*j*<1	Left shift by 1 of (S*i*) and (S*j*) to S*i*	3-56
056*i*0*k*†	S*i* S*i*<A*k*	Left shift by (A*k*) of (S*i*) to S*i*	3-56
057*ijk*	S*i* S*j*,S*i*>A*k*	Right shift by (A*k*) of (S*j*) and (S*i*) to S*i*	3-56
057*ij*0†	S*i* S*j*,S*i*>1	Right shift by 1 of (S*j*) and (S*i*) to S*i*	3-56
057*i*0*k*†	S*i* S*i*>A*k*	Right shift by (A*k*) of (S*i*) to S*i*	3-56
150*ijk*	V*i* V*j*<A*k*	Shift (V*j*) left (A*k*) places to V*i*	3-79
150*ij*0†	V*i* V*j*<1	Shift (V*j*) left one place to V*i*	3-79
151*ijk*	V*i* V*j*>A*k*	Shift (V*j*) right (A*k*) places to V*i*	3-79
151*ij*0†	V*i* V*j*>1	Shift (V*j*) right one place to V*i*	3-79
152*ijk*	V*i* V*j*,V*j*<A*k*	Double shift (V*j*) left (A*k*) places to V*i*	3-81
152*ij*0†	V*i* V*j*,V*j*<1	Double shift (V*j*) left one place to V*i*	3-81
153*ijk*	V*i* V*j*,V*j*>A*k*	Double shift (V*j*) right (A*k*) places to V*i*	3-81
153*ij*0†	V*i* V*j*,V*j*>1	Double shift (V*j*) right one place to V*i*	3-81

BIT COUNT INSTRUCTIONS

This category of instructions provides for counting the number of bits in an S or V register or counting the number of leading 0 bits in an S or V register.

† Special CAL syntax

Scalar population count

Machine Instruction	CAL	Description	Page
026*ij*0	A*i* PS*j*	Population count of (S*j*) to A*i*	3-36

Vector population count

Machine Instruction	CAL	Description	Page
174*ij*1	V*i* PV*j*	Population count of (V*j*) to (V*i*)	3-93

Scalar population count parity

Machine Instruction	CAL	Description	Page
026*ij*1	A*i* QS*j*	Population count parity of (S*j*) to A*i*	3-36
174*ij*2	V*i* QV*j*	Population count parity of (V*j*) to (V*i*)	3-93

Scalar leading zero count

Machine Instruction	CAL	Description	Page
027*ij*0	A*i* ZS*j*	Leading zero count of (S*j*) to A*i*	3-37

BRANCH INSTRUCTIONS

Instructions in this category include conditional and unconditional branch instructions. An expression or the contents of a B register specify the branch address. An address is always taken to be a parcel address when the instruction is executed. If an expression has a word-address attribute, the assembler issues an error message.

Unconditional branch instructions

Machine Instruction	CAL	Description	Page
0050*jk*	J B*jk*	Jump to (B*jk*)	3-26
006*ijkm*	J *exp*	Jump to *exp*	3-27

Conditional branch instructions

Machine Instruction	CAL	Description	Page
010*ijkm*	JAZ *exp*	Branch to *exp* if (A0)=0	3-29
011*ijkm*	JAN *exp*	Branch to *exp* if (A0)≠0	3-29
012*ijkm*	JAP *exp*	Branch to *exp* if (A0) positive	3-29
013*ijkm*	JAM *exp*	Branch to *exp* if (A0) negative	3-29
014*ijkm*	JSZ *exp*	Branch to *exp* if (S0)=0	3-30
015*ijkm*	JSN *exp*	Branch to *exp* if (S0)≠0	3-30
016*ijkm*	JSP *exp*	Branch to *exp* if (S0) positive	3-30
017*ijkm*	JSM *exp*	Branch to *exp* if (S0) negative	3-30

Return jump

Machine Instruction	CAL	Description	Page
001000†	PASS	Pass	3-11
007*ijkm*	R *exp*	Return jump to *exp*; set B00 to (P) + 2	3-28

† Special CAL syntax

Normal exit

Machine Instruction	CAL	Description	Page
004000	EX	Normal exit	3-25

Error exit

Machine Instruction	CAL	Description	Page
000000	ERR	Error exit	3-10

MONITOR MODE INSTRUCTIONS

Instructions described in this category are executed only when the CPU is in monitor mode. An attempt to execute one of these instructions when not in monitor mode is treated as a pass instruction.

The instructions perform specialized functions useful to the operating system.

Channel control

Machine Instruction	CAL	Description	Page
0010*jk*†	CA,A*j* A*k*	Set the Current Address (CA) register, for the channel indicated by (A*j*), to (A*k*) and activate the channel	3-11
0011*jk*†	CL,A*j* A*k*	Set the channel (A*j*) limit address to (A*k*)	3-12

† Privileged to monitor mode

Machine Instruction	CAL	Description	Page
0012*j*0†	CI,A*j*	Clear Channel (A*j*) Interrupt flag	3-13
0012*j*1	MC,A*j*	Clear Channel (A*j*) Interrupt flag and Error flag; set device master-clear (output channel); clear device ready-held (input channel)	3-13
0013*j*0†	XA A*j*	Enter XA register with (A*j*)	3-14

Set real-time clock

Machine Instruction	CAL	Description	Page
0014*j*0	RT S*j*	Enter RTC with (S*j*)	3-15

Programmable clock interrupt instructions

Machine Instruction	CAL	Description	Page
0014*j*4	PCI S*j*	Set program interrupt interval	3-15
001405	CCI	Clear clock interrupt	3-15
001406	ECI	Enable clock interrupts	3-15
001407	DCI	Disable clock interrupts	3-15

Interprocessor interrupt instructions†

Machine Instruction	CAL	Description	Page
0014*j*1††	SIPI *exp*	Set interprocessor interrupt request of CPU *exp*; $0 \le exp \le 3$	3-15

† Privileged to monitor mode
†† CRAY X-MP computer systems with multiprocessors

Machine Instruction	CAL	Description	Page
001401† †††	SIPI	Set interprocessor interrupt request	3-15
001402†††	CIPI	Clear interprocessor interrupt	3-15

Cluster number instructions

Machine Instruction	CAL	Description	Page
0014*j*3†††	CLN *exp*	Cluster number = *exp* where $0 \leq exp \leq 5$	3-15

Operand range error interrupt instructions

Machine Instruction	CAL	Description	Page
002300	ERI	Enable interrupt on address range error	3-21
002400	DRI	Disable interrupt on address range error	3-21

Performance counters††

Machine Instruction	CAL	Description	Page
0015*j*0		Select performance monitor	3-18
001501		Set maintenance read mode	3-18
001511		Load diagnostic checkbyte with S1	3-18

† Special CAL syntax
†† Instructions not supported by CAL at this time
††† CRAY X-MP computer systems with multiprocessors

Machine Instruction	CAL	Description	Page
001521†		Set maintenance write mode 1	3-18
001531†		Set maintenance write mode 2	3-18
073*i*11††		Read performance counter into S*i*	3-67
073*i*21††		Increment performance counter	3-67
073*i*31††		Clear all maintenance modes	3-67

† Instructions not supported by CAL at this time
†† Not currently supported

CRAY X-MP SYSTEM CONFIGURATIONS 4

The CRAY X-MP mainframe, I/O Subsystem (IOS), and associated equipment units are available with a number of options in a variety of system configurations. The options, such as the number of central processing units (CPUs), I/O Processors (IOPs), and channel connections, and a variety of memory sizes, banking arrangements, and peripheral devices, are used to produce several unique models. Table 4-1 shows an overview of all CRAY X-MP models currently available. A specification sheet is provided for each model later in this section; each specification sheet contains specific information for each of the CRAY X-MP models.

Table 4-1. CRAY X-MP Computer Systems Overview

Model	Number of CPUs	Number of Columns	Memory Size (MWs)	Number of Banks	Clock Speed	Channel Pairs (Maximum)				IOS Information (all totals are maximum configuration)					SSD (32, 64, 128 256, or 512 Mword)
						6 Mbyte/s	100 Mbyte/s	1000 Mbyte/s	HSX - 100 Mbyte/s	Number of IOPs (2 is minimum number)	Buffer Memory Size (Mword)	Disk Storage Units	Magnetic Tape Channels	Front-end Interfaces (or NSC A130 Adapters)	
X-MP/14se	1	6	4	16	xx.x ns	2	2	NA	Optional	2	2 (4, 8 optional)	8	4	3	Not Available
X-MP/14	1	8	4	16	8.5 ns	4	2	1	Optional	4	4 (8 optional)	32	8	7	Optional
X-MP/18	1	8	8	32	8.5 ns	4	2	1	Optional	4	4 (8 optional)	23	8	7	Optional
X-MP/116	1	8	16	32	8.5 ns	4	2	1	Optional	4	4 (8 optional)	32	8	7	Optional
X-MP/22	2	8	2	16	8.5 ns	4	2	1	Optional	4	4 (8 optional)	32	8	7	Optional
X-MP/24	2	8	4	16	8.5 ns	4	2	1	Optional	4	4 (8 optional)	32	8	7	Optional
X-MP/28	2	8	8	32	8.5 ns	4	2	1	Optional	4	4 (8 optional)	32	8	7	Optional
X-MP/216	2	8	16	32	8.5 ns	4	2	1	Optional	4	4 (8 optional)	32	8	7	Optional
X-MP/44	4	12	4	32	8.5 ns	4	4	2	Optional	4	4 (8 optional)	32	8	7	Optional
X-MP/48	4	12	8	32	8.5 ns	4	4	2	Optional	4	8	32	8	7	Optional
X-MP/416	4	12	16	64	8.5 ns	4	4	2	Optional	4	8	32	8	7	Optional

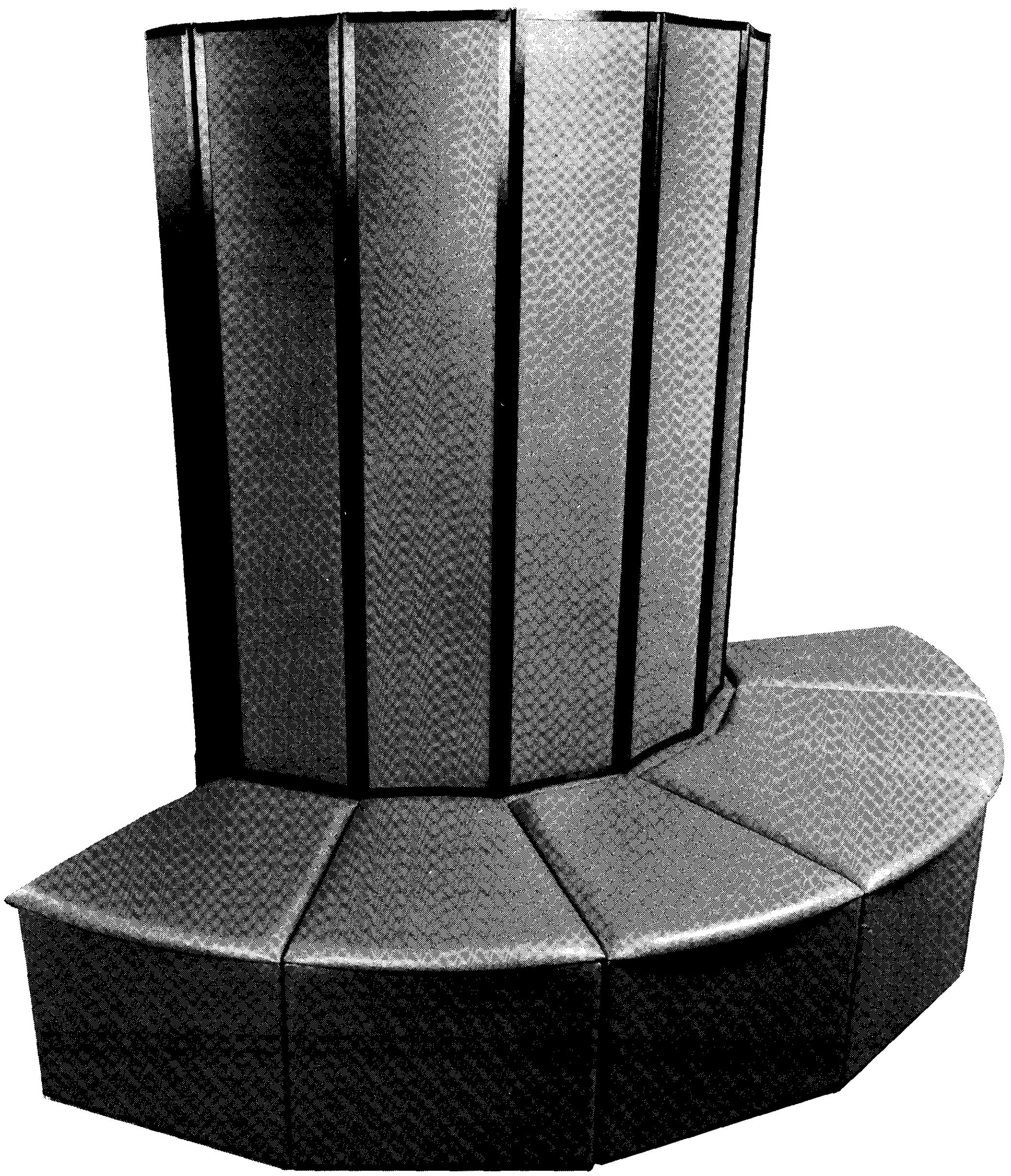


Figure 4-1. CRAY X-MP/14se Computer System

Table 4-2. CRAY X-MP/14se Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	1	Address functional units:
Clock speed	xx.x ns	Addition (A)
Memory size	4 Mword	Multiplication (A)
Number of banks	16	Scalar functional units:
Number of memory ports	4 (A, B C, I/O)	Addition (S)
Channels	Two 100 Mbyte/s to IOS	Shift-single (S)
	Two 6 Mbyte/s to IOS	Shift-double (S)
		Logical (S)
		Population, parity, and leading zero (S)
Number of columns	6	Vector functional units:
Arc	135°	Addition (S and V)
Floor Space	19.4 sq ft (1.8 m^2)	Shift (V)
		Full vector logical (S and V)
Weight	2.95 tons (2676.2 kg)	2nd vector logical (S and V)
		Population, parity (V)
		Floating-point functional units:
		Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

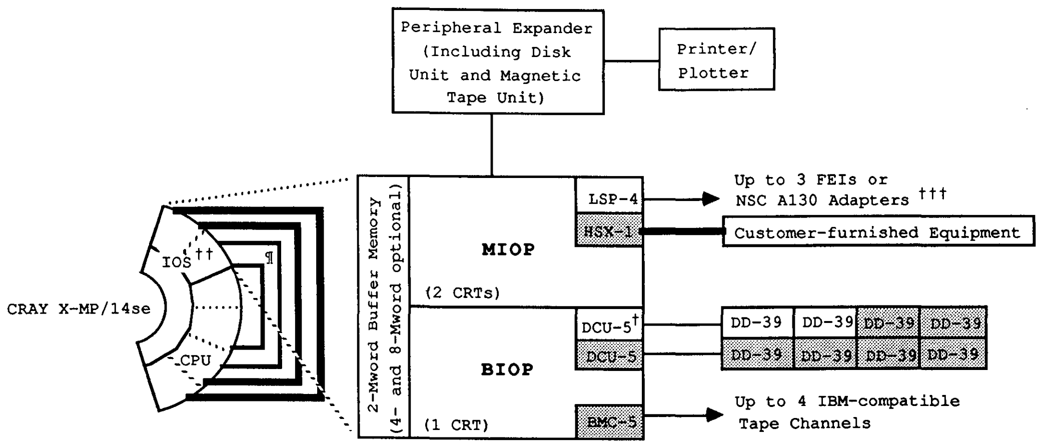
Unique Features

- IOS resides in mainframe cabinet.
- Unattended operation is possible; environmental monitor shuts system down in emergency power-off situations.
- Installation process is simpler than with standard CRAY X-MP model.
- Motor alternator and condenser unit in computer room.
- Pre-installation requirements are floor cut-outs and water supply.

Table 4-2. CRAY X-MP/14se Features (continued)

Special Instruction Information
The following instructions are not available: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Power distribution unit	1
Quietized motor alternator (75 KVA)	1
Quietized condensing unit	1
Environmental monitor (for unattended emergency power-off situations)	1


Peripheral Expander (Including Disk Unit and Magnetic Tape Unit)
Printer/ Plotter
CRAY X-MP/14se
IOS ††
¶
CPU
2-Mword Buffer Memory (4- and 8-Mword optional)
MIOP
(2 CRTs)
LSP-4
HSX-1
BIOP
(1 CRT)
DCU-5†
DCU-5
BMC-5
Up to 3 FEIs or NSC A130 Adapters †††
Customer-furnished Equipment
DD-39
Up to 4 IBM-compatible Tape Channels

▨ Optional equipment

─ 6-Mbyte/s channel

━ 100-Mbyte/s channel

† One DCU-5 is required, but priced separately.

†† A second 100-Mbyte/s channel is needed if the optional BMC-5 or HSX-1 is used.

††† An FEI-1 box is not included.

¶ Software support for the second 6-Mbyte/s channel is not currently available.

1774

Figure 4-2. CRAY X-MP/14se Configuration (Maximum)

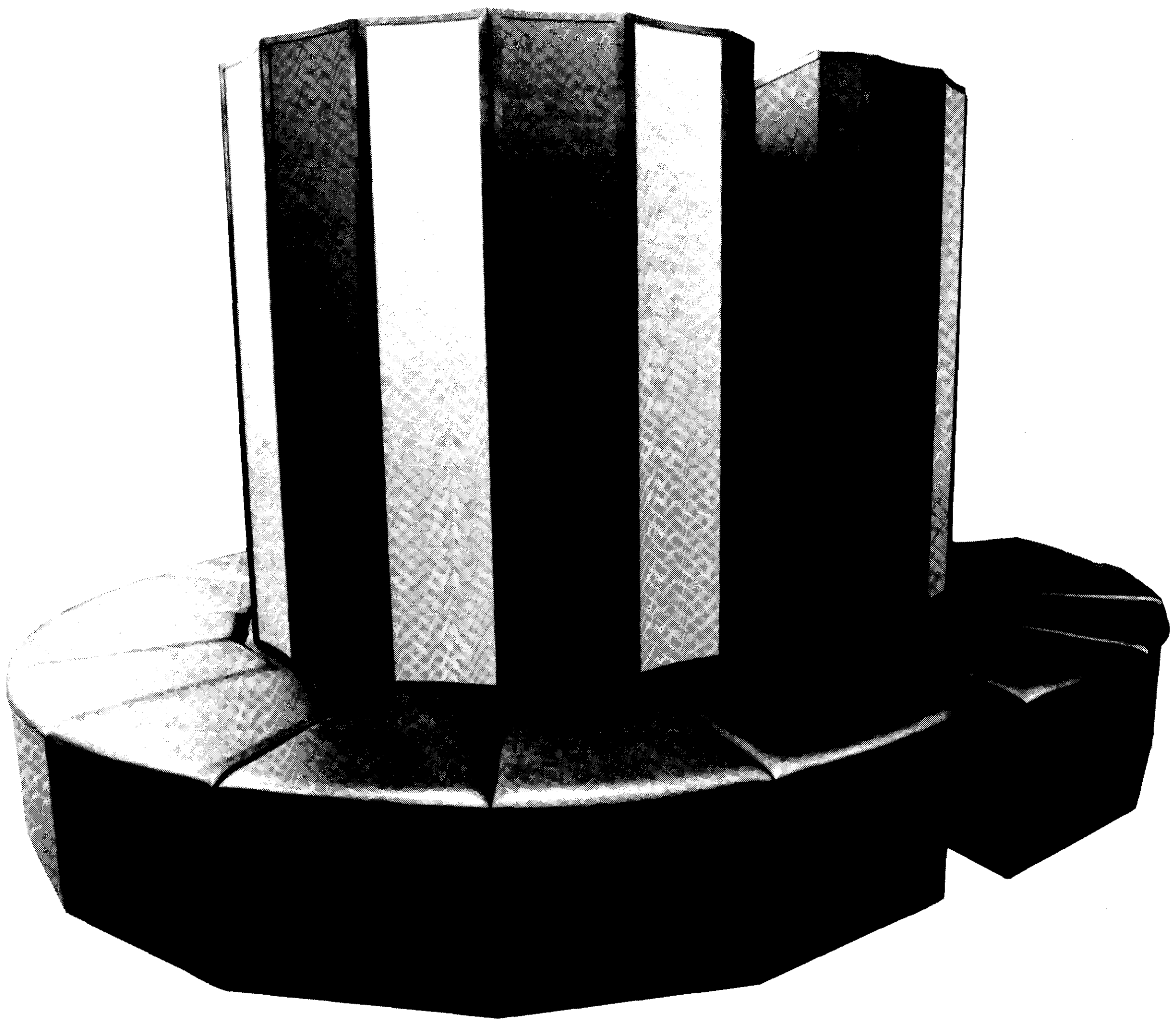


Figure 4-3. CRAY X-MP/14 Computer System

Table 4-3. CRAY X-MP/14 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	1	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	4 Mword	Multiplication (A)
Number of banks	16	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
Channels	One 1000 Mbyte/s to SSD	Shift-single (S)
	Up to two 100 Mbyte/s to IOS	Shift-double (S)
	Up to four 6 Mbyte/s	Logical (S)
		Population, parity, and leading zero (S)
		Vector functional units:
		Addition (S and V)
		Shift (V)
Number of columns	8	Full vector logical (S and V)
Arc	180°	2nd vector logical (S and V)
Floor space	25.75 sq ft ($2.39\ m^2$)	Population, parity (V)
		Floating-point functional units:
Weight	3.76 tons (3411.1 kg)	Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Unique Features
• Can be field-upgraded to a dual-processor system

Table 4-3. CRAY X-MP/14 Features (continued)

Special Instruction Information
The following instructions are not available: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Motor-generator sets	2
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

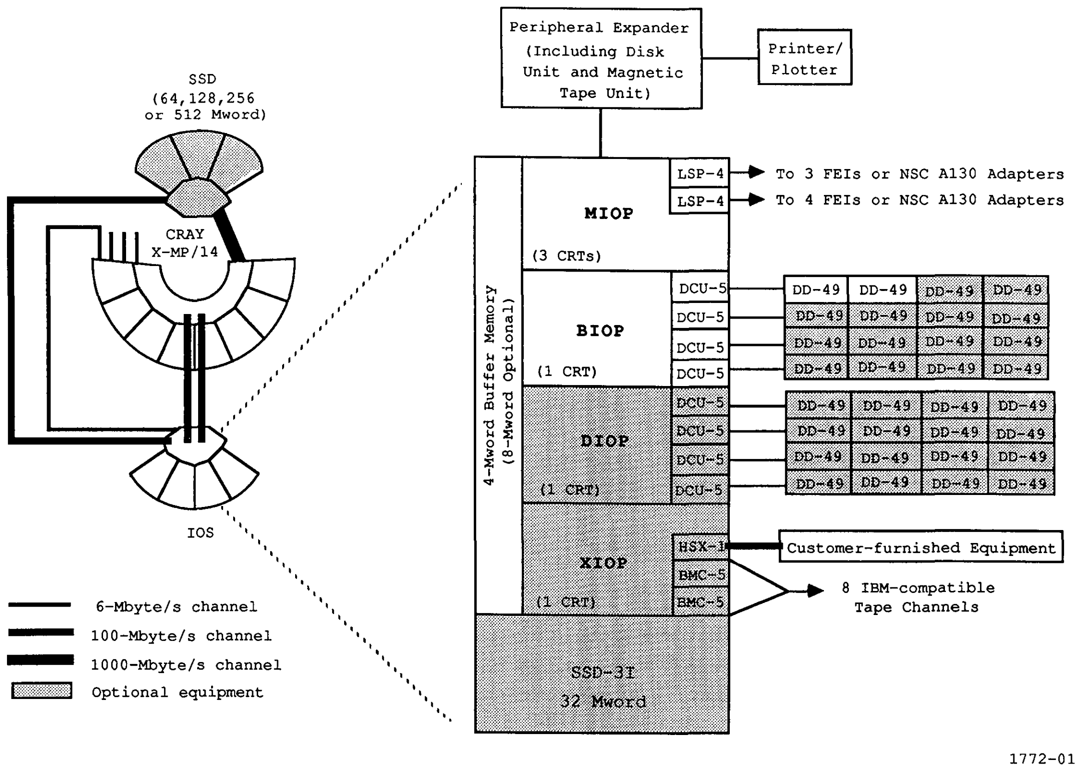


Figure 4-4. CRAY X-MP/14 Configuration (Maximum)

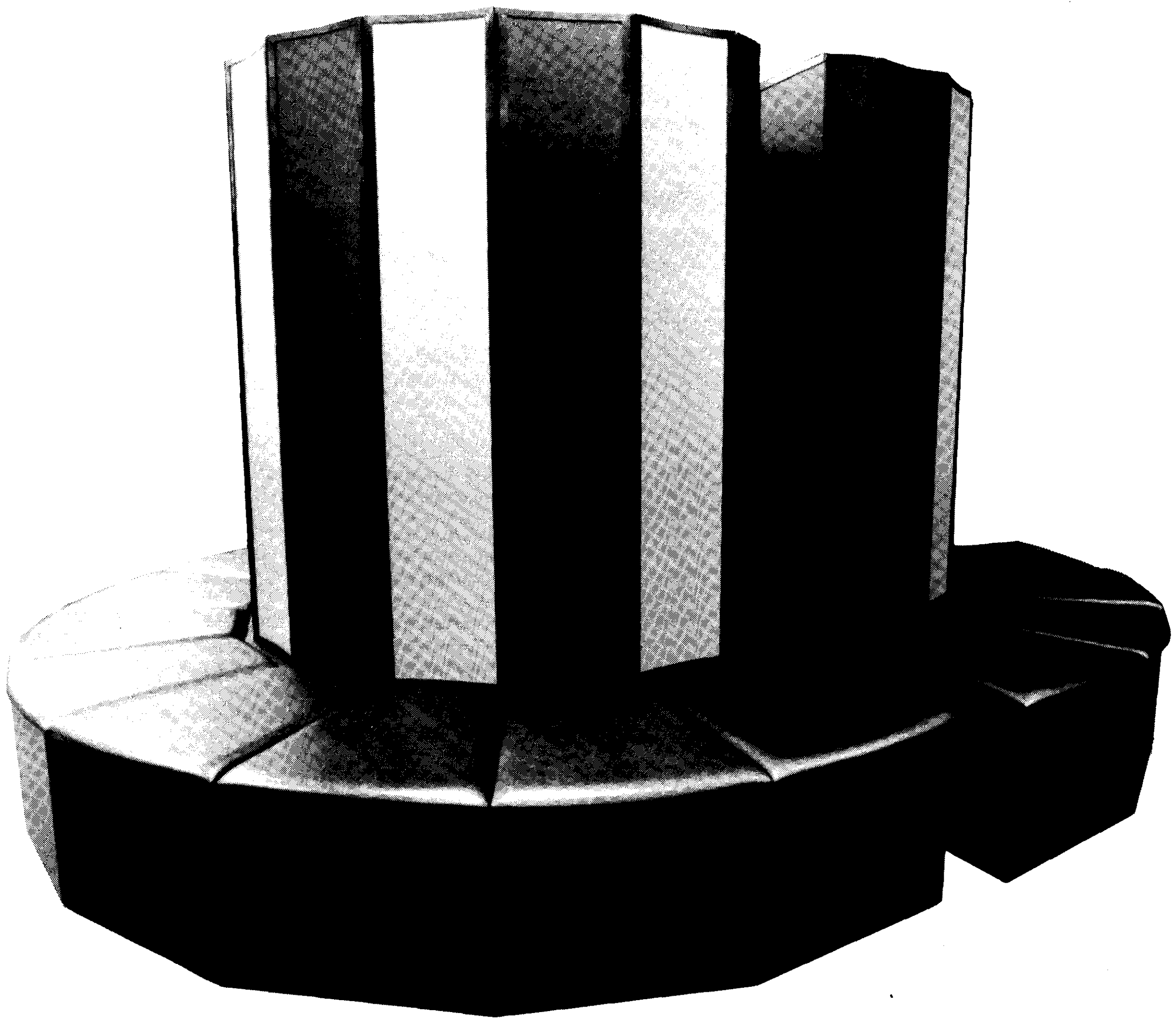


Figure 4-5. CRAY X-MP/18 Computer System

Table 4-4. CRAY X-MP/18 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	1	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	8 Mword	Multiplication (A)
Number of banks	32	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
Channels	One 1000 Mbyte/s to SSD	Shift-single (S)
	Up to two 100 Mbyte/s to IOS	Shift-double (S)
	Up to four 6 Mbyte/s	Logical (S)
		Population, parity, and leading zero (S)
		Vector functional units:
		Addition (S and V)
		Shift (V)
Number of columns	8	Full vector logical (S and V)
Arc	180°	2nd vector logical (S and V)
Floor space	25.75 sq ft ($2.39\ m^2$)	Population, parity (V)
Weight	3.76 tons (3411.1 kg)	Floating-point functional units:
		Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Unique Features
• Can be field-upgraded to a dual-processor system

Table 4-4. CRAY X-MP/18 Features (continued)

Special Instruction Information
The following instructions are not available: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Motor-generator sets	2
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

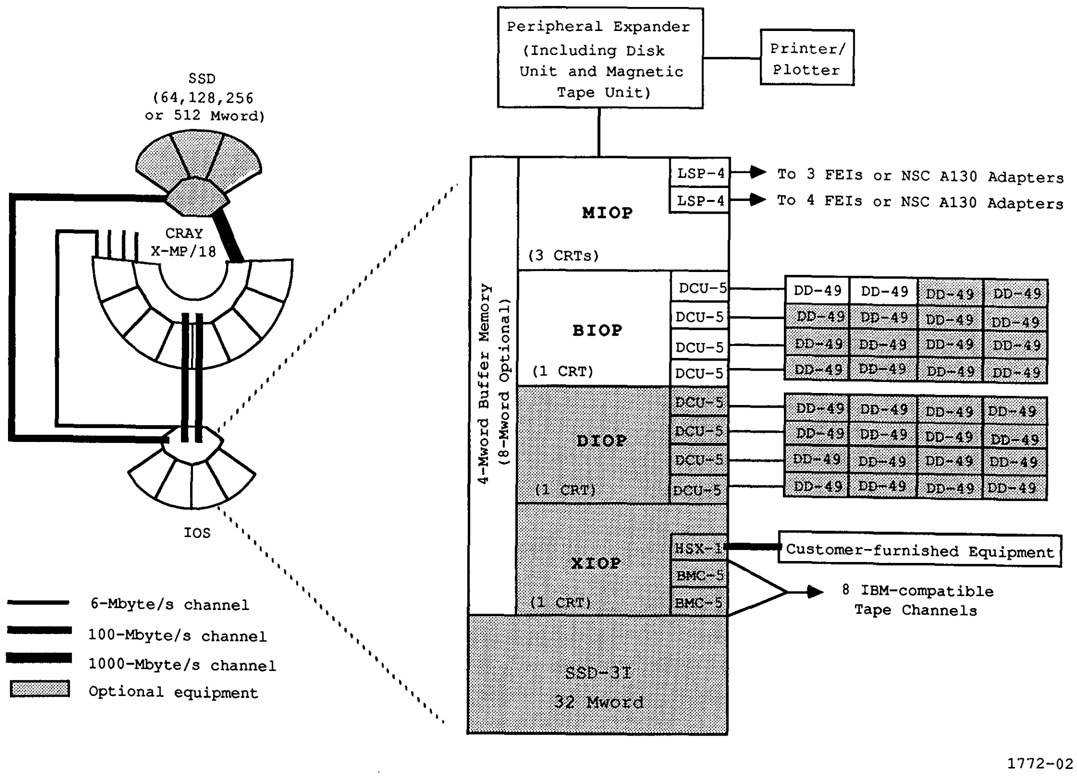

SSD
(64,128,256 or 512 Mword)
CRAY X-MP/18
IOS
6-Mbyte/s channel
100-Mbyte/s channel
1000-Mbyte/s channel
Optional equipment
Peripheral Expander (Including Disk Unit and Magnetic Tape Unit)
Printer/ Plotter
4-Mword Buffer Memory (8-Mword Optional)
MIOP
(3 CRTs)
LSP-4
LSP-4
To 3 FEIs or NSC A130 Adapters
To 4 FEIs or NSC A130 Adapters
BIOP
(1 CRT)
DCU-5
DD-49
DIOP
(1 CRT)
XIOP
(1 CRT)
HSX-1
BMC-5
Customer-furnished Equipment
8 IBM-compatible Tape Channels
SSD-3I
32 Mword
1772-02

Figure 4-6. CRAY X-MP/18 Configuration (Maximum)

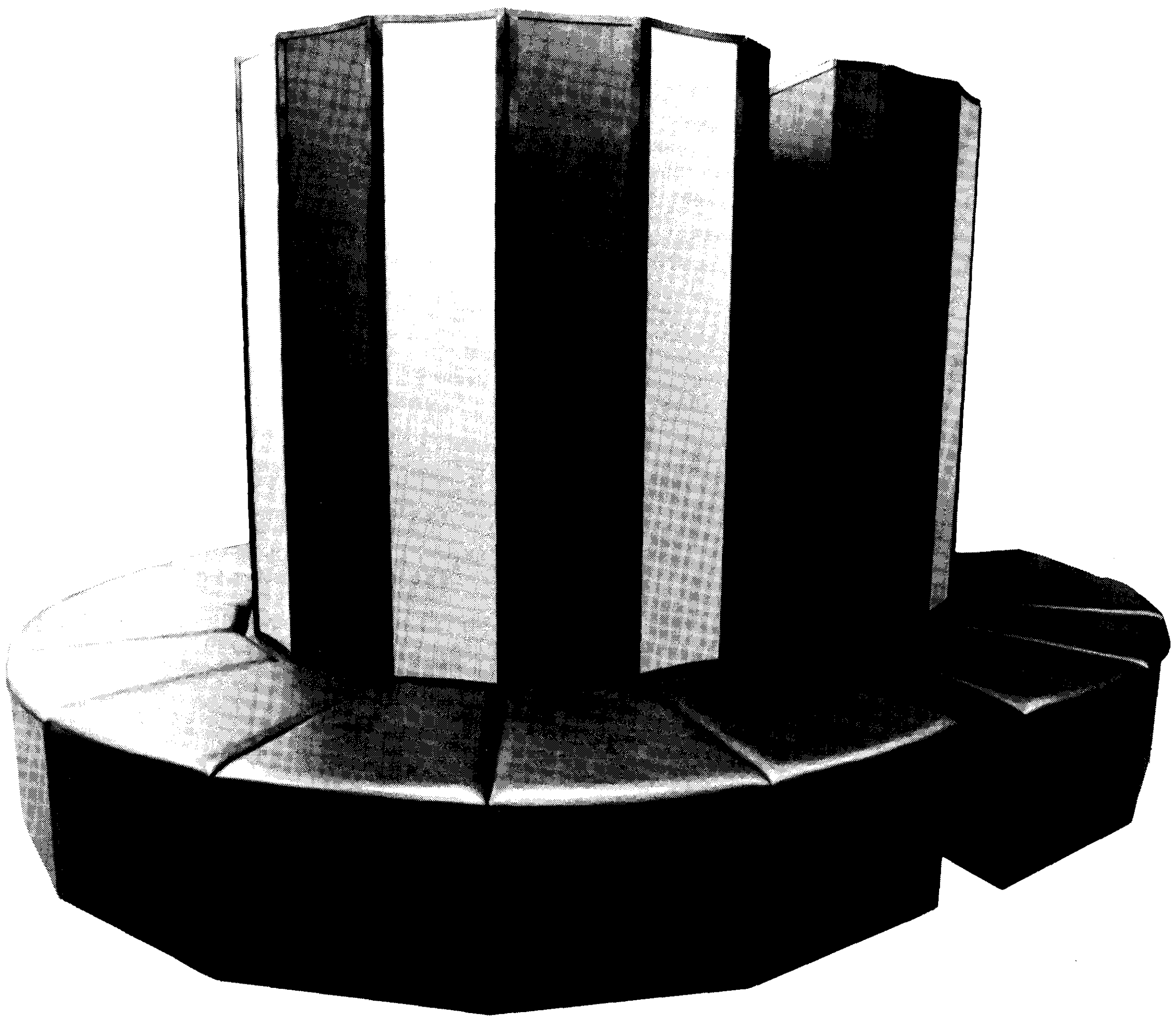


Figure 4-7. CRAY X-MP/116 Computer System

Table 4-5. CRAY X-MP/116 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	1	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	16 Mword	Multiplication (A)
Number of banks	32	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
		Shift-single (S)
Channels	One 1000 Mbyte/s to SSD	Shift-double (S)
		Logical (S)
	Up to two 100 Mbyte/s to IOS	Population, parity, and leading zero (S)
		Vector functional units:
	Up to four 6 Mbyte/s	Addition (S and V)
		Shift (V)
Number of columns	8	Full vector logical (S and V)
Arc	180°	2nd vector logical (S and V)
Floor space	25.75 sq ft (2.39 m^2)	Population, parity (V)
		Floating-point functional units:
Weight	3.76 tons (3411.1 kg)	Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

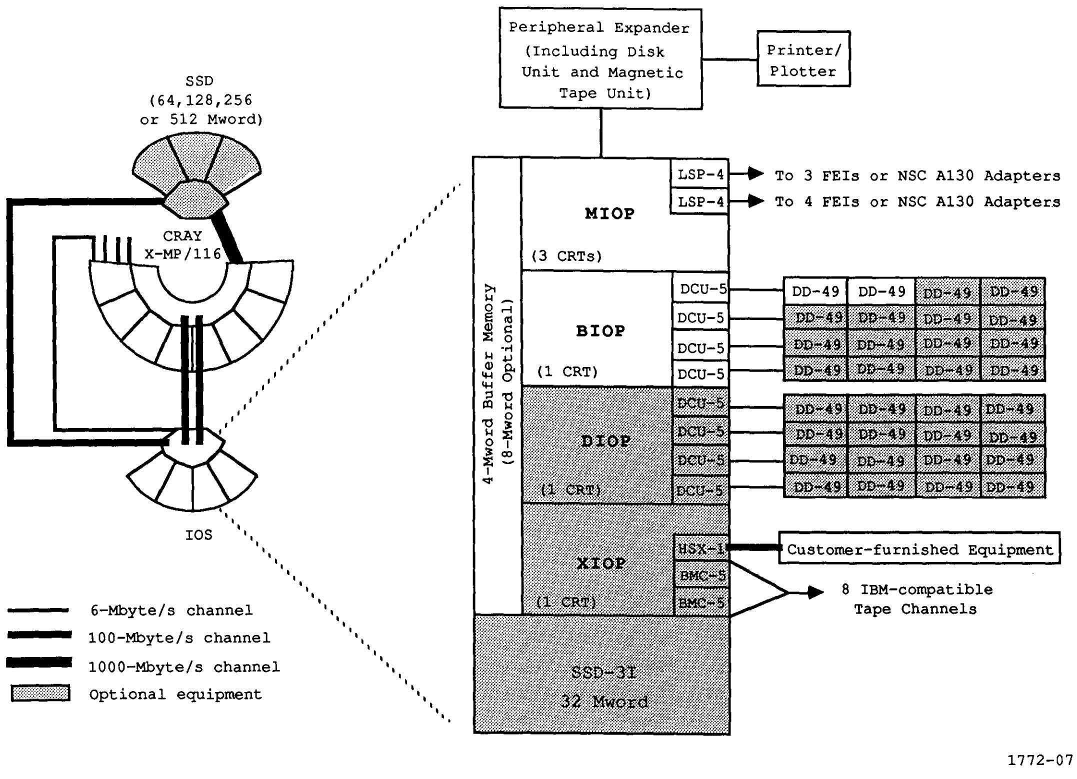
Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Unique Features
• Can be field-upgraded to a dual-processor system

Table 4-5. CRAY X-MP/116 Features (continued)

Special Instruction Information
The following instructions are not available: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Motor-generator sets	2
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1



SSD (64,128,256 or 512 Mword)
CRAY X-MP/116
IOS
6-Mbyte/s channel
100-Mbyte/s channel
1000-Mbyte/s channel
Optional equipment
Peripheral Expander (Including Disk Unit and Magnetic Tape Unit)
Printer/ Plotter
4-Mword Buffer Memory (8-Mword Optional)
MIOP (3 CRTs)
LSP-4
LSP-4
To 3 FEIs or NSC A130 Adapters
To 4 FEIs or NSC A130 Adapters
BIOP (1 CRT)
DCU-5
DD-49
DIOP (1 CRT)
XIOP (1 CRT)
HSX-1
BMC-5
BMC-5
Customer-furnished Equipment
8 IBM-compatible Tape Channels
SSD-3I 32 Mword
1772-07

Figure 4-8. CRAY X-MP/116 Configuration (Maximum)

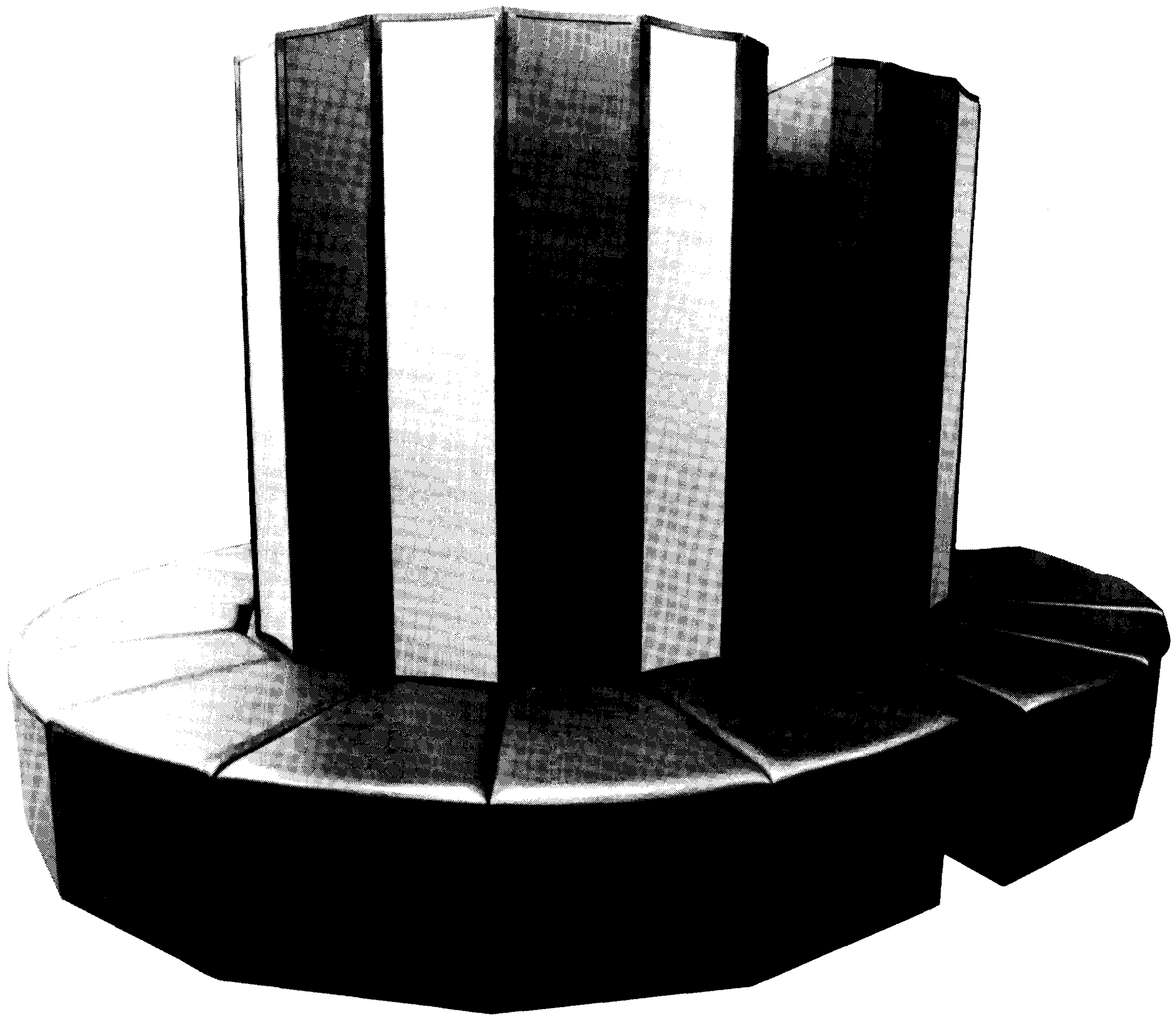


Figure 4-9. CRAY X-MP/22 Computer System

Table 4-6. CRAY X-MP/22 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	2	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	2 Mword	Multiplication (A)
Number of banks	16	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
		Shift-single (S)
Channels	One 1000 Mbyte/s to SSD	Shift-double (S)
		Logical (S)
	Up to two 100 Mbyte/s to IOS	Population, parity, and leading zero (S)
		Vector functional units:
	Up to four 6 Mbyte/s	Addition (S and V)
		Shift (V)
Number of columns	8	Full vector logical (S and V)
Arc	180°	2nd vector logical (S and V)
Floor space	25.75 sq ft ($2.39\ m^2$)	Population, parity (V)
		Floating-point functional units:
Weight	3.76 tons (3411.1 kg)	Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Table 4-6. CRAY X-MP/22 Features (continued)

Special Instruction Information
The following instructions are specific to multi-processor systems: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Motor-generator sets	2
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

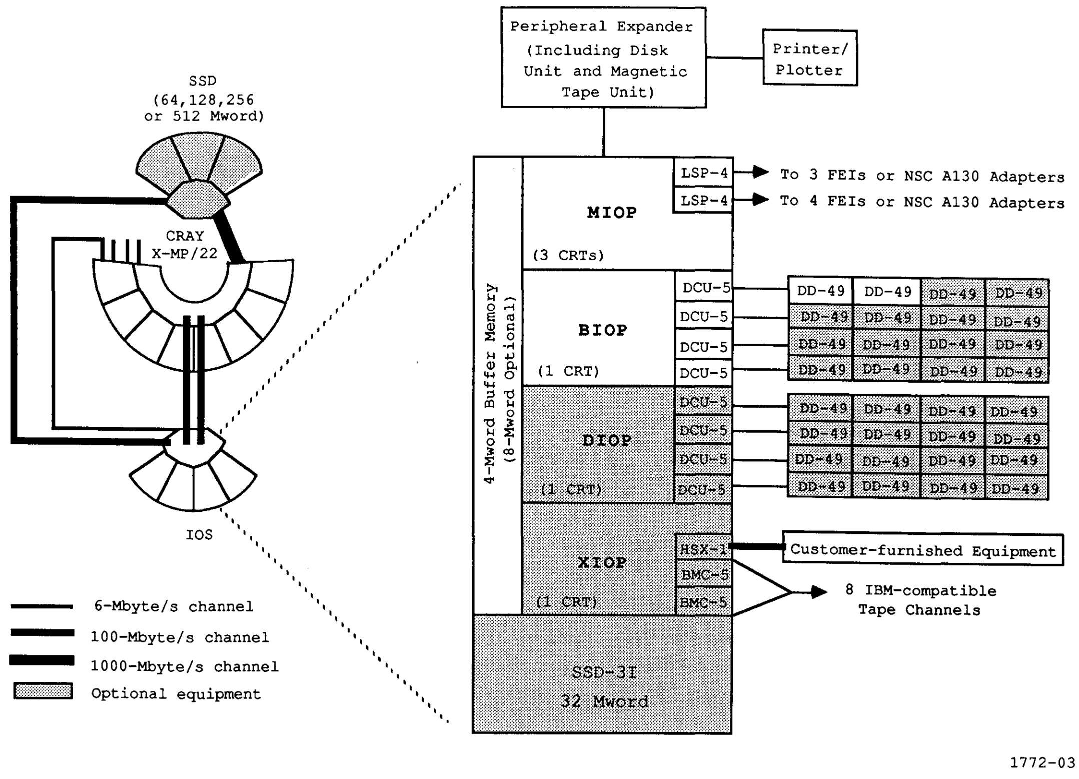


Figure 4-10. CRAY X-MP/22 Configuration (Maximum)

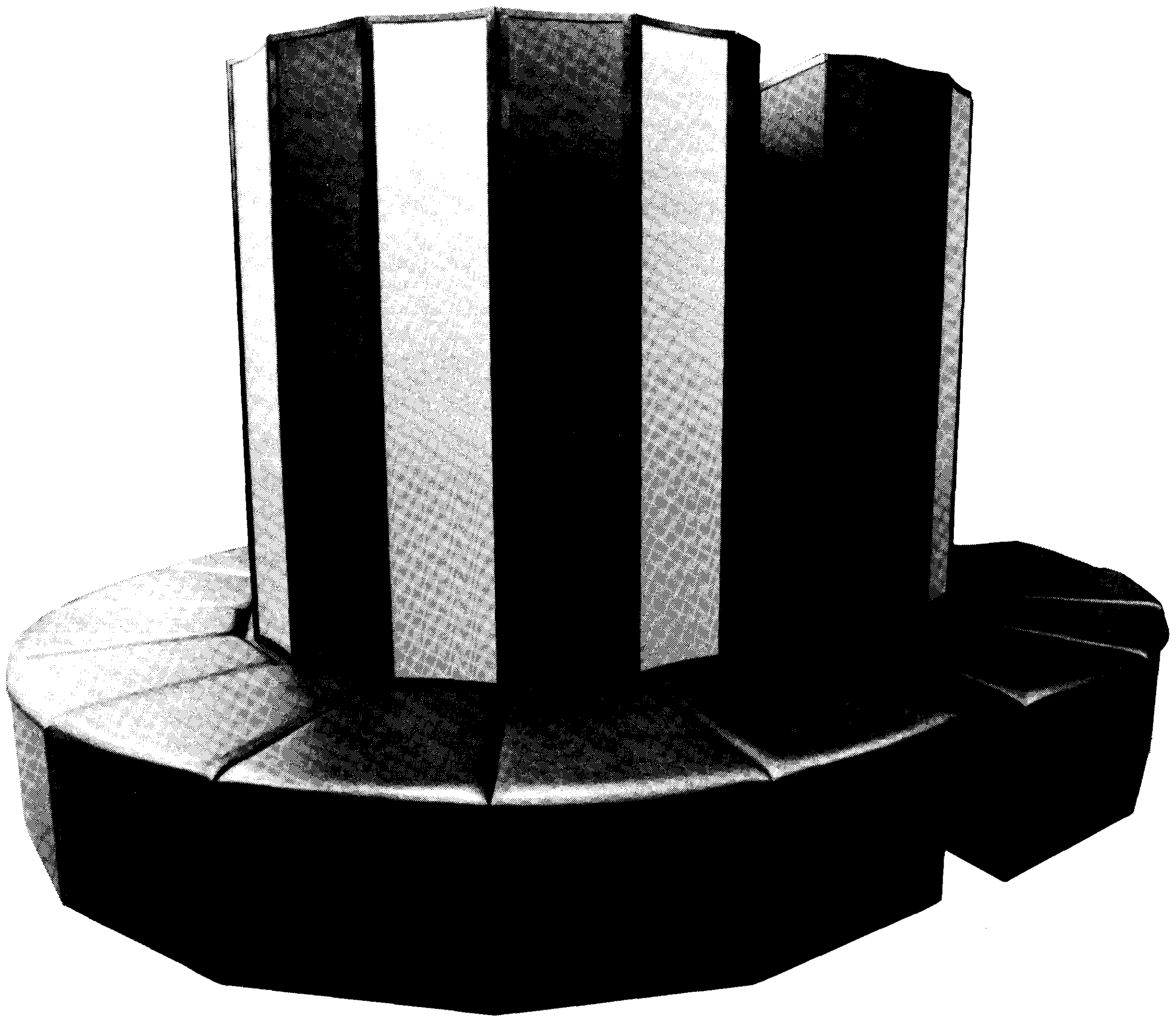


Figure 4-11. CRAY X-MP/24 Computer System

Table 4-7. CRAY X-MP/24 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	2	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	4 Mword	Multiplication (A)
Number of banks	16	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
Channels	One 1000 Mbyte/s to SSD	Shift-single (S)
	Up to two 100 Mbyte/s to IOS	Shift-double (S)
	Up to four 6 Mbyte/s	Logical (S)
Number of columns	8	Population, parity, and leading zero (S)
Arc	180°	Vector functional units:
Floor space	25.75 sq ft (2.39 m^2)	Addition (S and V)
Weight	3.76 tons (3411.1 kg)	Shift (V)
		Full vector logical (S and V)
		2nd vector logical (S and V)
		Population, parity (V)
		Floating-point functional units:
		Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Table 4-7. CRAY X-MP/24 Features (continued)

Special Instruction Information
The following instructions are specific to multi-processor systems: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit (RCU-1)	1
Motor-generator sets	2
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

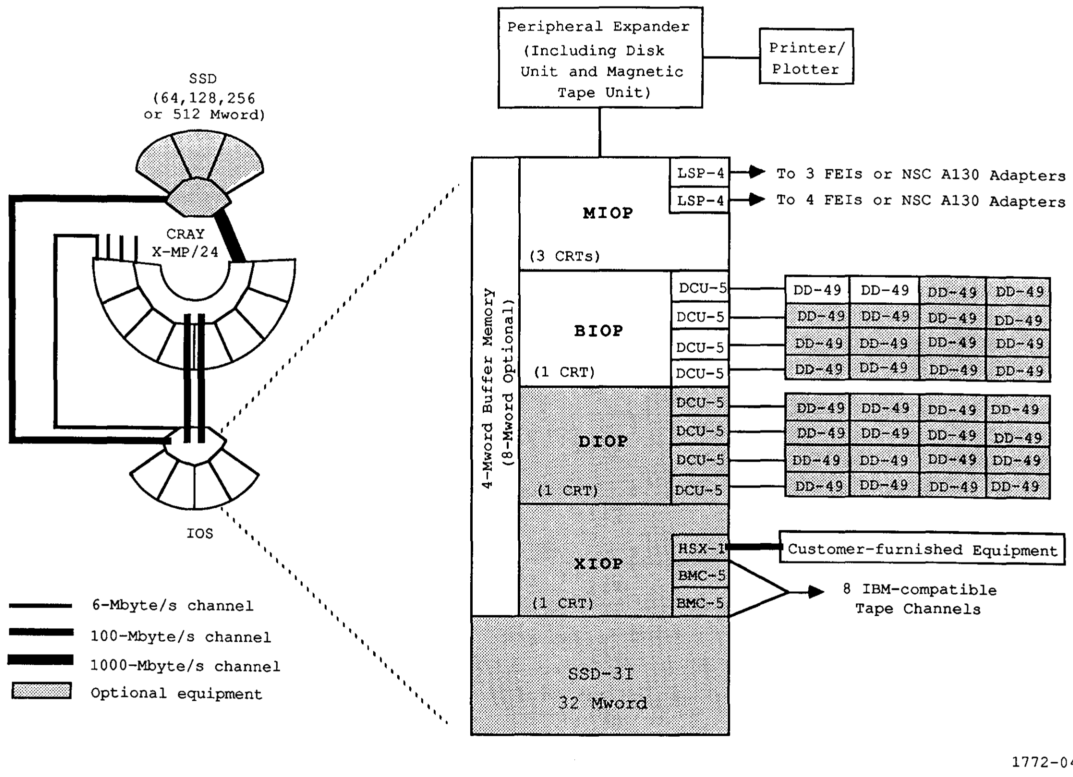


Figure 4-12. CRAY X-MP/24 Configuration (Maximum)

CRAY RESEARCH, INC.

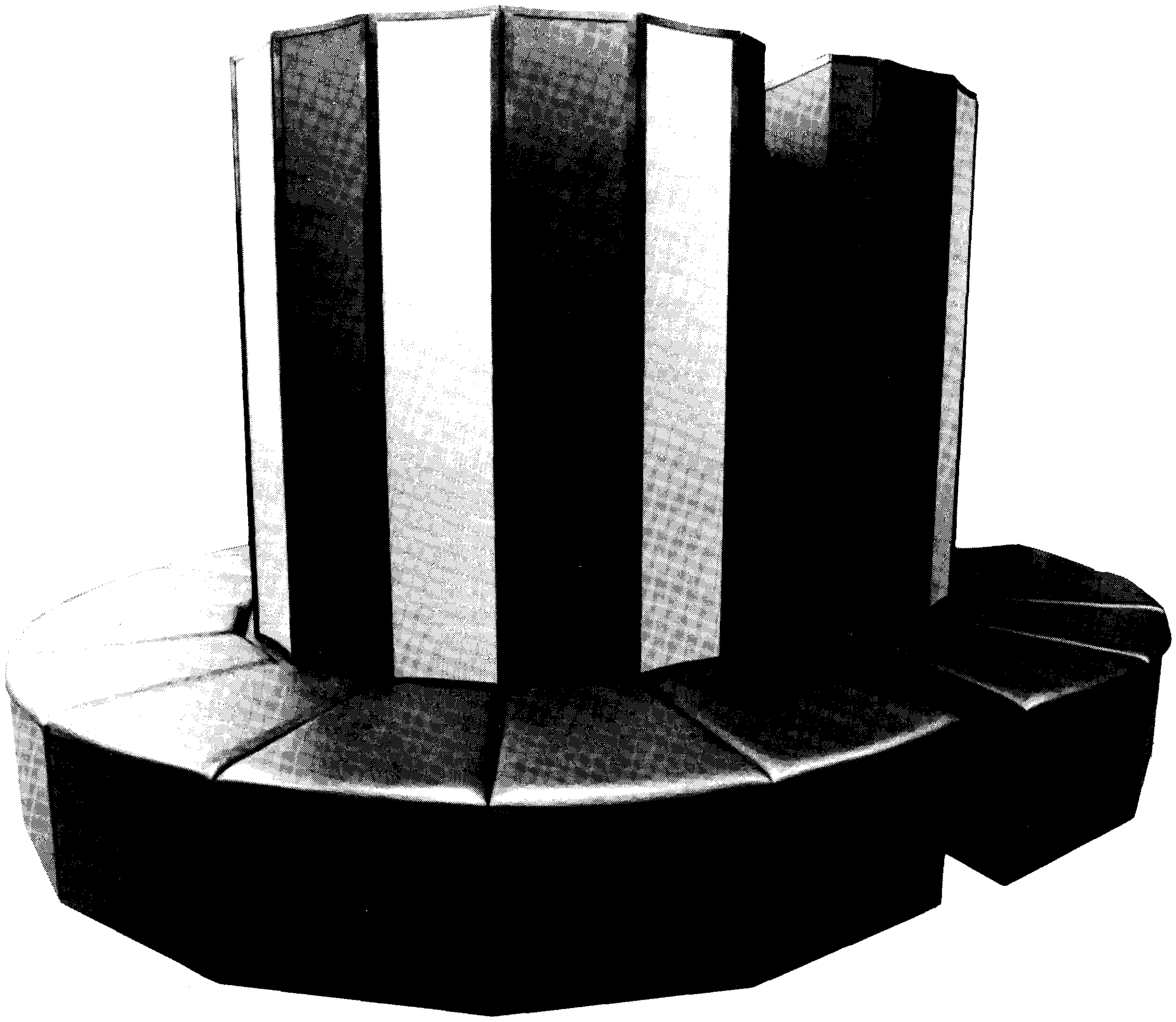


Figure 4-13. CRAY X-MP/28 Computer System

Table 4-8. CRAY X-MP/28 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	2	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	8 Mword	Multiplication (A)
Number of banks	32	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
Channels	One 1000 Mbyte/s to SSD	Shift-single (S)
	Up to two 100 Mbyte/s to IOS	Shift-double (S)
	Up to four 6 Mbyte/s	Logical (S)
		Population, parity, and leading zero (S)
		Vector functional units:
		Addition (S and V)
		Shift (V)
Number of columns	8	Full vector logical (S and V)
Arc	180°	2nd vector logical (S and V)
Floor space	25.75 sq ft (2.39 m^2)	Population, parity (V)
Weight	3.76 tons (3411.1 kg)	Floating-point functional units:
		Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Table 4-8. CRAY X-MP/28 Features (continued)

Special Instruction Information
The following instructions are specific to multi-processor systems: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Motor-generator sets	2
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

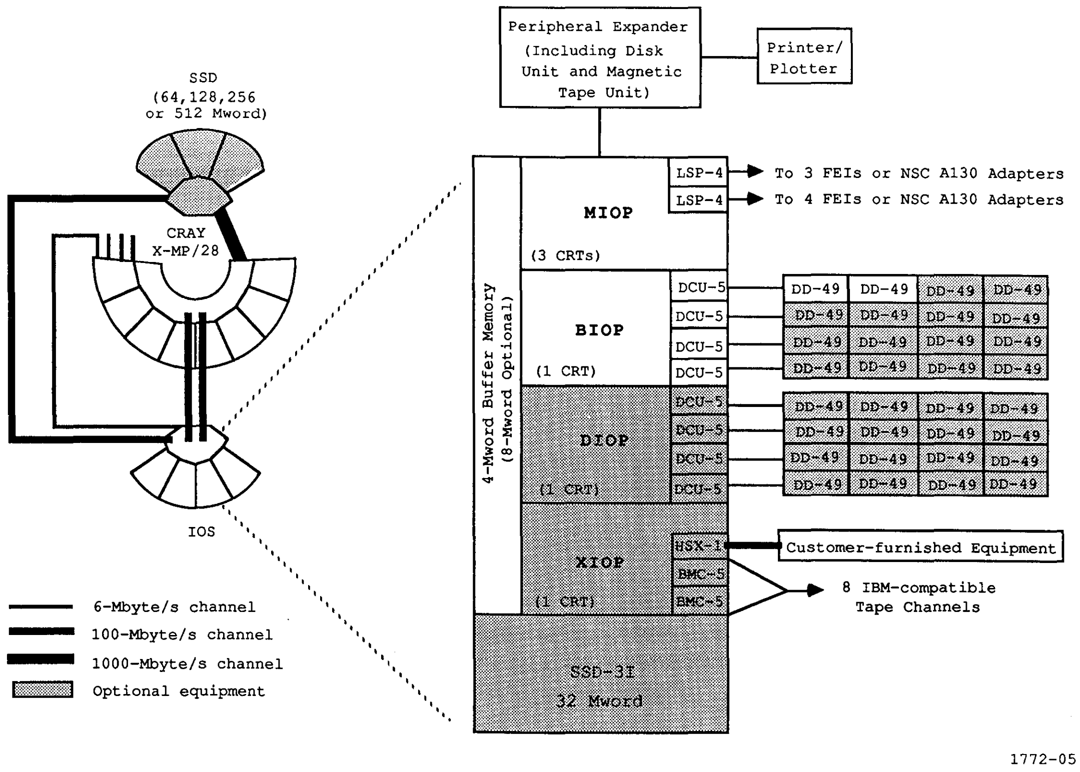


Figure 4-14. CRAY X-MP/28 Configuration (Maximum)

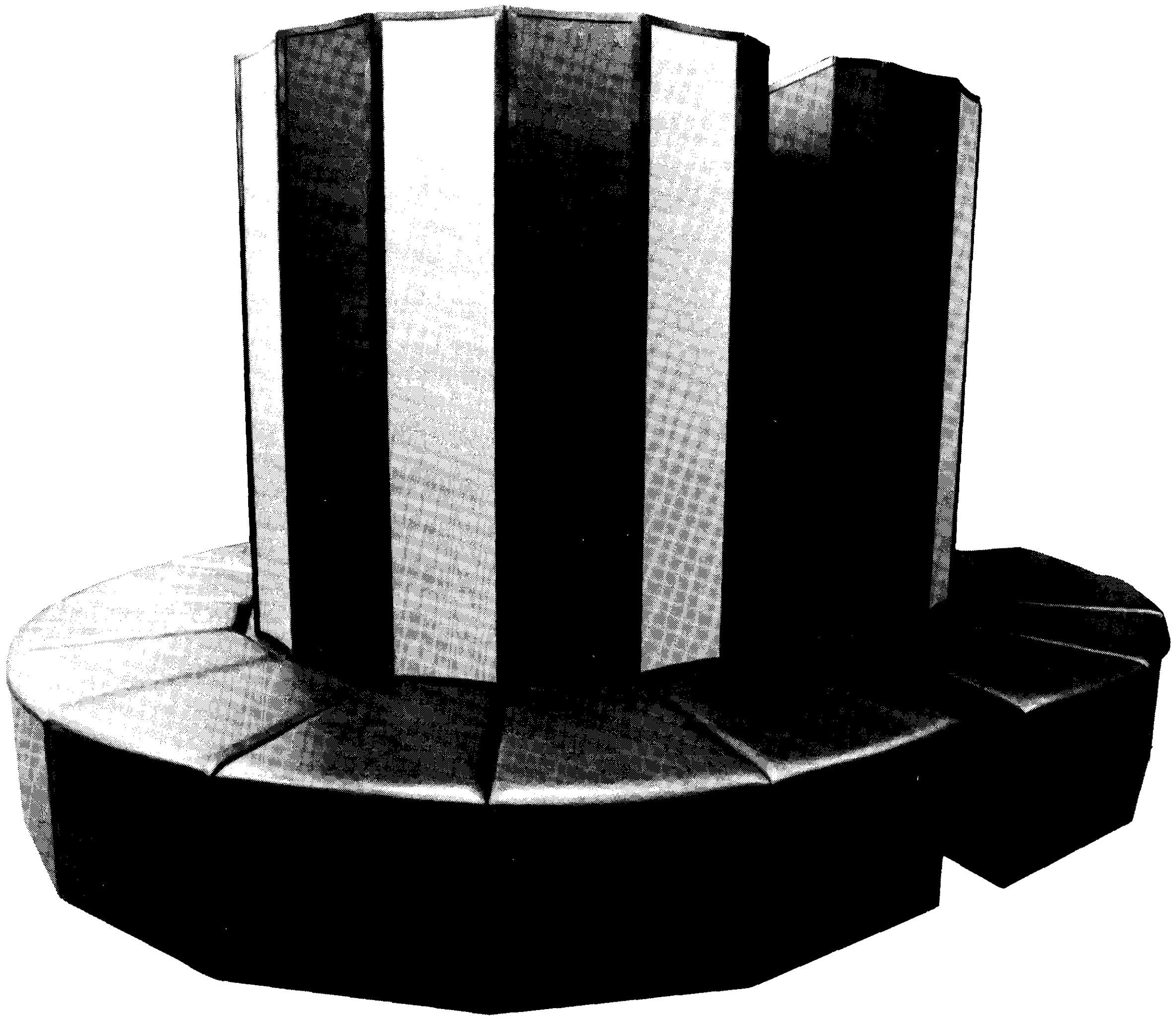


Figure 4-15. CRAY X-MP/216 Computer System

Table 4-9. CRAY X-MP/216 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	2	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	16 Mword	Multiplication (A)
Number of banks	32	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
Channels	One 1000 Mbyte/s to SSD	Shift-single (S)
	Up to two 100 Mbyte/s to IOS	Shift-double (S)
	Up to four 6 Mbyte/s	Logical (S)
Number of columns	8	Population, parity, and leading zero (S)
Arc	180°	Vector functional units:
Floor space	25.75 sq ft (2.39 m^2)	Addition (S and V)
Weight	3.76 tons (3411.1 kg)	Shift (V)
		Full vector logical (S and V)
		2nd vector logical (S and V)
		Population, parity (V)
		Floating-point functional units:
		Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Table 4-9. CRAY X-MP/216 Features (continued)

Special Instruction Information
The following instructions are specific to multi-processor systems: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Motor-generator sets	2
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

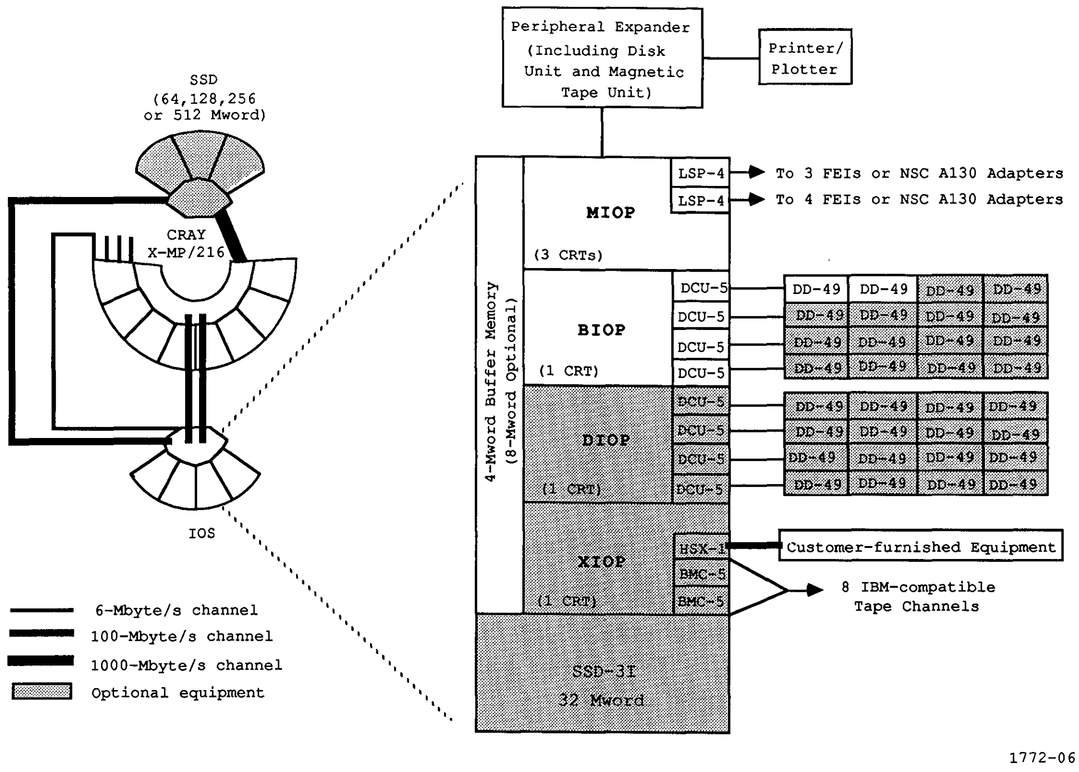

SSD
(64,128,256 or 512 Mword)
CRAY X-MP/216
IOS
6-Mbyte/s channel
100-Mbyte/s channel
1000-Mbyte/s channel
Optional equipment
Peripheral Expander (Including Disk Unit and Magnetic Tape Unit)
Printer/ Plotter
4-Mword Buffer Memory (8-Mword Optional)
MIOP
(3 CRTs)
LSP-4
LSP-4
To 3 FEIs or NSC A130 Adapters
To 4 FEIs or NSC A130 Adapters
BIOP
(1 CRT)
DCU-5
DD-49
DIOP
(1 CRT)
XIOP
(1 CRT)
HSX-1
BMC-5
BMC-5
Customer-furnished Equipment
8 IBM-compatible Tape Channels
SSD-3I
32 Mword
1772-06

Figure 4-16. CRAY X-MP/216 Configuration (Maximum)

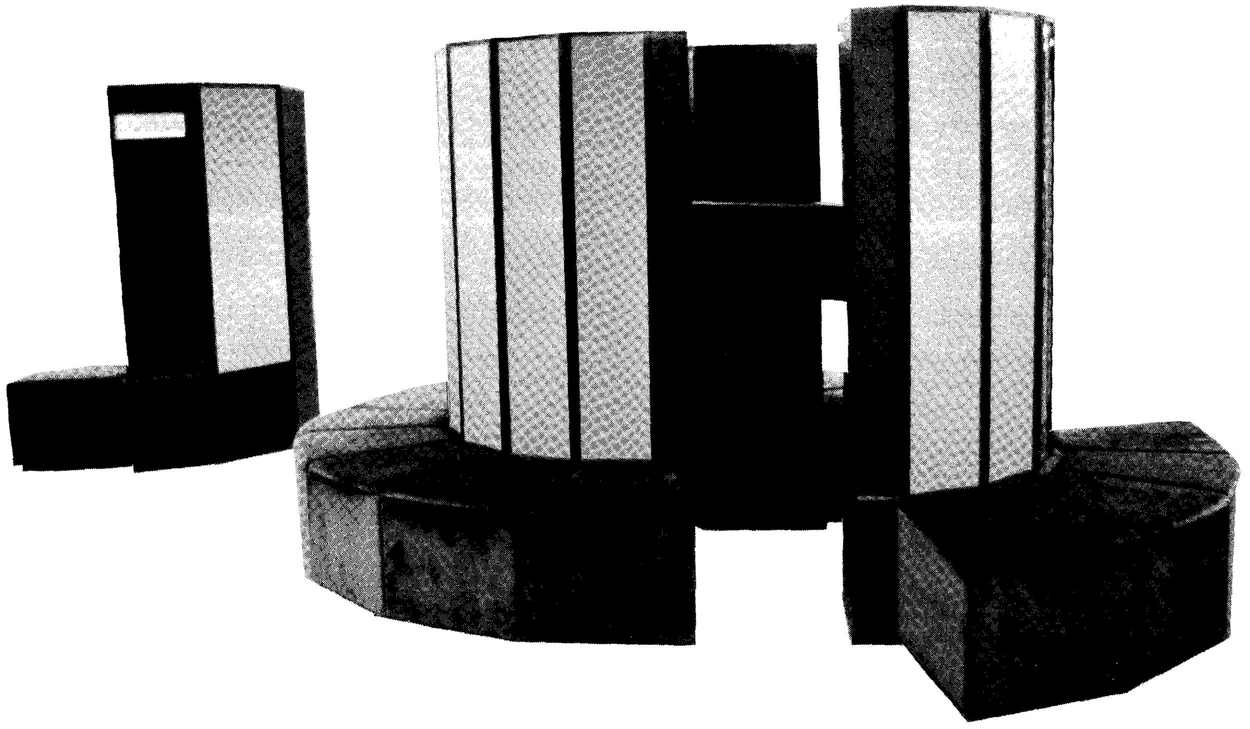


Figure 4-17. CRAY X-MP/44 Computer System

Table 4-10. CRAY X-MP/44 Features

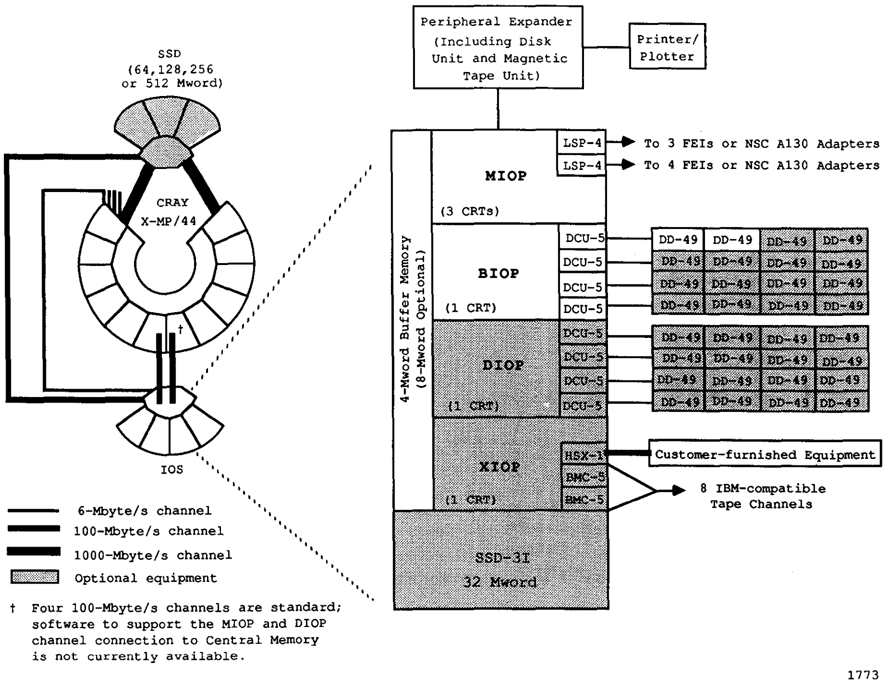
CPU Features		Functional Units Available (Register Usage)
Number of CPUs	4	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	4 Mword	Multiplication (A)
Number of banks	32	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
Channels	Two 1000 Mbyte/s to SSD	Shift-single (S)
	Up to four 100 Mbyte/s to IOS	Shift-double (S)
	Up to four 6 Mbyte/s	Logical (S)
Number of columns	12	Population, parity, and leading zero (S)
Arc	270°	Vector functional units:
Floor space	38.5 sq ft (3.57 m^2)	Addition (S and V)
Weight	5.65 tons (5125.5 kg)	Shift (V)
		Full vector logical (S and V)
		2nd vector logical (S and V)
		Population, parity (V)
		Floating-point functional units:
		Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Table 4-10. CRAY X-MP/44 Features (continued)

Special Instruction Information
The following instructions are specific to multi-processor systems: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Auxilliary refrigeration condensing units (with SSD - 2 units needed)	1
Motor-generator sets	3
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1


SSD (64,128,256 or 512 Mword)
CRAY X-MP/44
†
IOS
6-Mbyte/s channel
100-Mbyte/s channel
1000-Mbyte/s channel
Optional equipment
Peripheral Expander (Including Disk Unit and Magnetic Tape Unit)
Printer/ Plotter
4-Mword Buffer Memory (8-Mword Optional)
MIOP
(3 CRTs)
LSP-4
LSP-4
To 3 FEIs or NSC A130 Adapters
To 4 FEIs or NSC A130 Adapters
BIOP
(1 CRT)
DCU-5
DD-49
DIOP
(1 CRT)
XIOP
(1 CRT)
HSX-1
BMC-5
BMC-5
Customer-furnished Equipment
8 IBM-compatible Tape Channels
SSD-3I
32 Mword
1773

† Four 100-Mbyte/s channels are standard; software to support the MIOP and DIOP channel connection to Central Memory is not currently available.

Figure 4-18. CRAY X-MP/44 Configuration (Maximum)

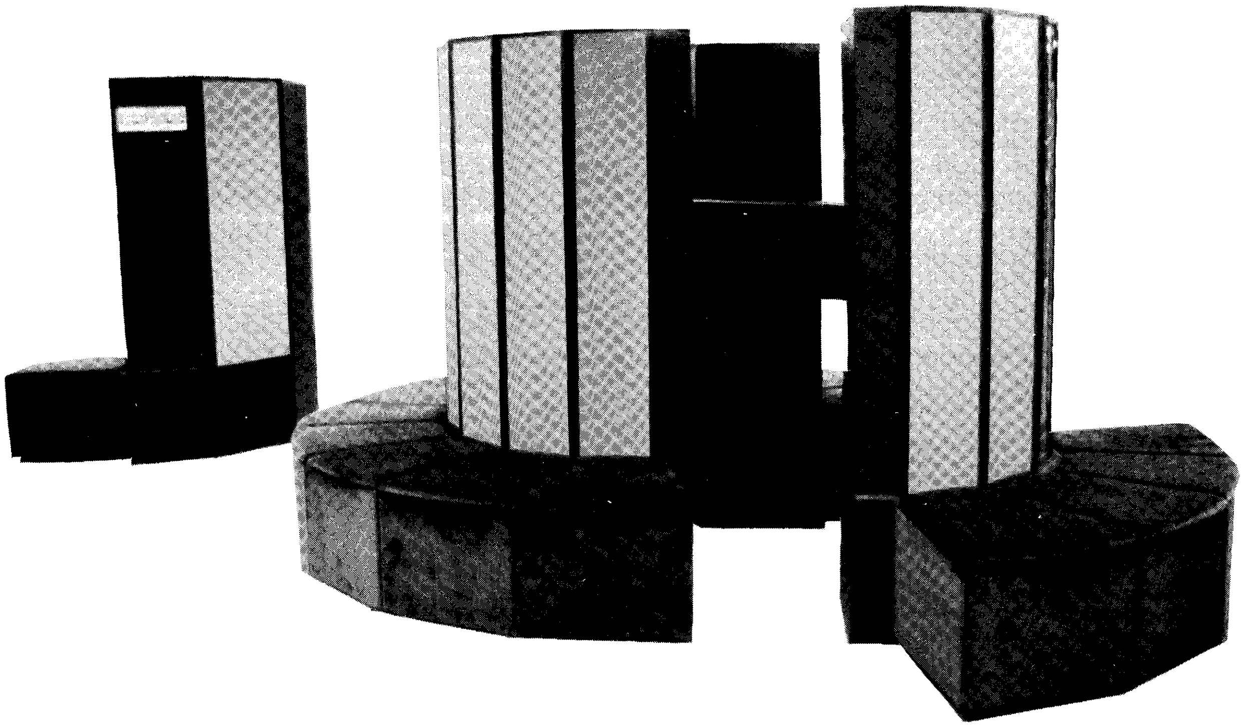


Figure 4-19. CRAY X-MP/48 Computer System

Table 4-11. CRAY X-MP/48 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	4	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	8 Mword	Multiplication (A)
Number of banks	32	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
Channels	Two 1000 Mbyte/s to SSD	Shift-single (S)
	Up to four 100 Mbyte/s to IOS	Shift-double (S)
	Up to four 6 Mbyte/s	Logical (S)
Number of columns	12	Population, parity, and leading zero (S)
Arc	270°	Vector functional units:
Floor space	38.5 sq ft (3.57 m^2)	Addition (S and V)
Weight	5.65 tons (5125.5 kg)	Shift (V)
		Full vector logical (S and V)
		2nd vector logical (S and V)
		Population, parity (V)
		Floating-point functional units:
		Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Unique Features
• 8-Mword Buffer Memory is standard.

Table 4-11. CRAY X-MP/48 Features (continued)

Special Instruction Information
The following instructions are specific to multi-processor systems: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Auxilliary refrigeration condensing units (with SSD - 2 units needed)	1
Motor-generator sets	3
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

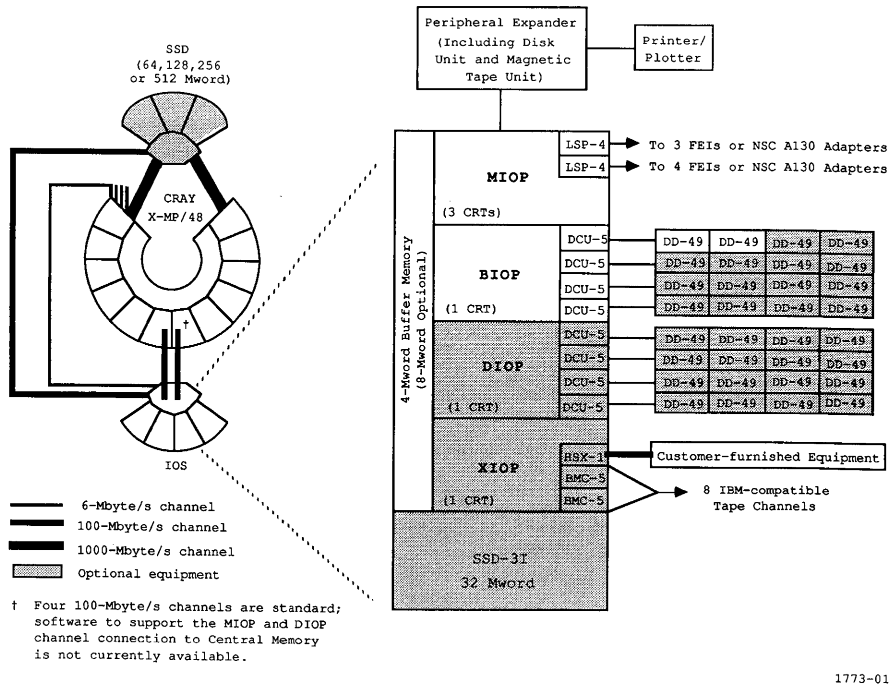

SSD (64,128,256 or 512 Mword)
CRAY X-MP/48
IOS
6-Mbyte/s channel
100-Mbyte/s channel
1000-Mbyte/s channel
Optional equipment
† Four 100-Mbyte/s channels are standard; software to support the MIOP and DIOP channel connection to Central Memory is not currently available.
Peripheral Expander (Including Disk Unit and Magnetic Tape Unit)
Printer/ Plotter
4-Mword Buffer Memory (8-Mword Optional)
MIOP (3 CRTs)
LSP-4
LSP-4
To 3 FEIs or NSC A130 Adapters
To 4 FEIs or NSC A130 Adapters
BIOP (1 CRT)
DIOP (1 CRT)
DCU-5
DD-49
XIOP (1 CRT)
BSX-1
BMC-5
BMC-5
Customer-furnished Equipment
8 IBM-compatible Tape Channels
SSD-3I 32 Mword
1773-01

Figure 4-20. CRAY X-MP/48 Configuration (Maximum)

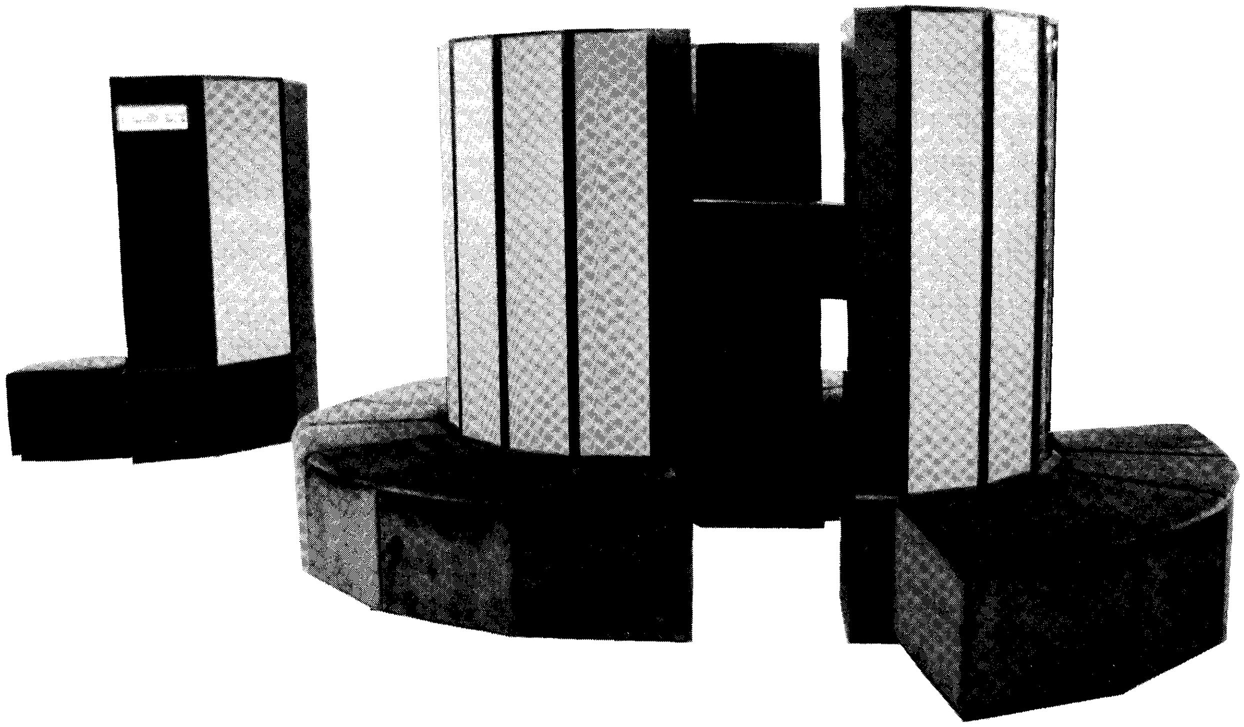


Figure 4-21. CRAY X-MP/416 Computer System

Table 4-11. CRAY X-MP/416 Features

CPU Features		Functional Units Available (Register Usage)
Number of CPUs	4	Address functional units:
Clock speed	8.5 ns	Addition (A)
Memory size	16 Mword	Multiplication (A)
Number of banks	64	Scalar functional units:
Number of memory ports	4 (A, B, C, I/O)	Addition (S)
		Shift-single (S)
Channels	Two 1000 Mbyte/s to SSD	Shift-double (S)
		Logical (S)
	Up to four 100 Mbyte/s to IOS	Population, parity, and leading zero (S)
		Vector functional units:
	Up to four 6 Mbyte/s	Addition (S and V)
		Shift (V)
Number of columns	12	Full vector logical (S and V)
Arc	270°	2nd vector logical (S and V)
Floor space	38.5 sq ft (3.57 m^2)	Population, parity (V)
		Floating-point functional units:
Weight	5.65 tons (5125.5 kg)	Addition (S and V)
		Multiplication (S and V)
		Reciprocal approximation (S and V)

Register Type Available:	Quantity	Size
Address (A)	8	24 bits
Intermediate (B)	64	24 bits
Scalar (S)	8	64 bits
Scalar-save (T)	64	64 bits
Vector (V)	8	64 elements (64 bits per element)

Unique Features
• 8-Mword Buffer Memory is standard.

Table 4-11. CRAY X-MP/416 Features (continued)

Special Instruction Information
The following instructions are specific to multi-processor systems: Interprocessor instructions - 0014*j*1, 001401, 001402, 0014*j*3, 0034*jk*, 0036*jk*, 0037*jk*, 026*ij*7, 027*ij*7, 072*i*02, 072*ij*3, 073*ij*1, 073*i*02, 073*ij*3

Support Equipment	Number of Units Needed
Mainframe power distribution unit	1
IOS power distribution unit	1
Refrigeration condensing unit	1
Auxilliary refrigeration condensing units (with SSD - 2 units needed)	1
Motor-generator sets	3
SSD power distribution unit (optional)	1
Maintenance micro computer unit	1

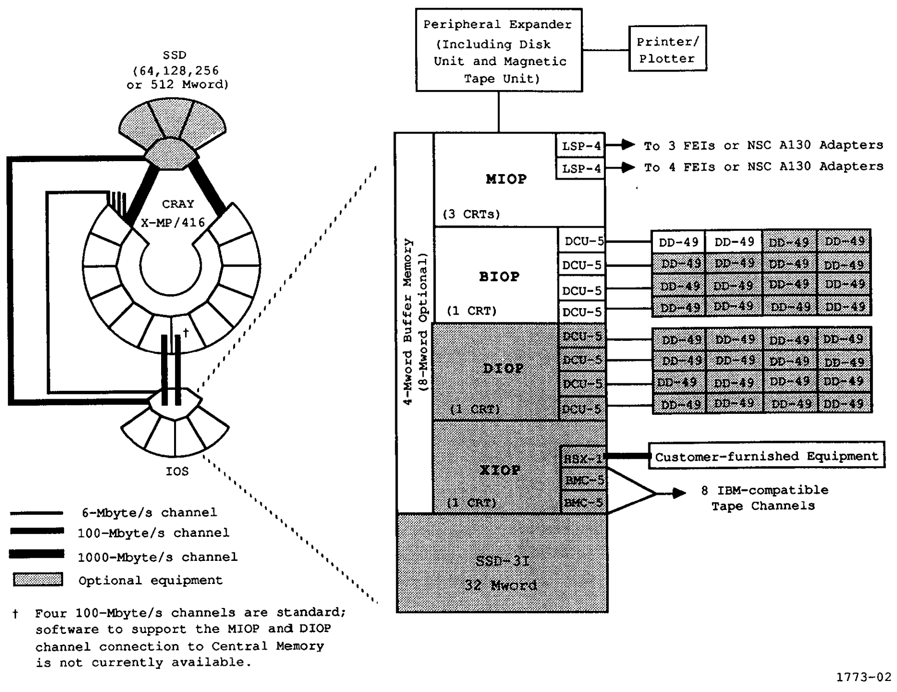

SSD
(64,128,256 or 512 Mword)
CRAY X-MP/416
IOS
†
6-Mbyte/s channel
100-Mbyte/s channel
1000-Mbyte/s channel
Optional equipment
† Four 100-Mbyte/s channels are standard; software to support the MIOP and DIOP channel connection to Central Memory is not currently available.
Peripheral Expander (Including Disk Unit and Magnetic Tape Unit)
Printer/ Plotter
4-Mword Buffer Memory (8-Mword Optional)
MIOP
(3 CRTs)
LSP-4
LSP-4
To 3 FEIs or NSC A130 Adapters
To 4 FEIs or NSC A130 Adapters
BIOP
(1 CRT)
DCU-5
DD-49
DIOP
(1 CRT)
XIOP
(1 CRT)
HSX-1
BMC-5
Customer-furnished Equipment
8 IBM-compatible Tape Channels
SSD-3I
32 Mword
1773-02

Figure 4-22. CRAY X-MP/416 Configuration (Maximum)

GLOSSARY

GLOSSARY

A

A register - Address register

B

B register - Intermediate Address register

BDM - Bidirectional Memory Mode flag

BIOP - Buffer I/O Processor

BMC - Block multiplexer controller

C

CA - Current Address register

CAL - Cray Assembly Language

CCI - Clear clock interrupt

CE - Channel Error flag

CI - Channel Interrupt flag

CIP - Current Instruction Parcel

CIPI - Clear interprocessor interrupt

CL - Channel Limit register

CLN - Cluster Number register (Exchange Package)

CMR - Complete memory references

CP - Clock Period

CPU - Central processing units

CRI - Cray Research, Inc.

CSB - Read Address register (Exchange Package)

D

DBA - Data Base Address register (Exchange Package)

DBM - Disable bidirectional memory transfers

DCI - Disable clock interrupts

DCU - Disk controller unit

DFI - Disable floating-point interrupt

DIOP - Disk I/O Processor

DL - Deadlock flag

DLA - Data Limit Address register (Exchange Package)

DMA - Direct Memory Address

DRI - Disable interrupt on range error interrupt

DSU - Disk Storage Unit

E

E - Error type (Exchange Package)

EAM - Enhanced Addressing Mode (Exchange Package)

EBM - Enable bidirectional memory transfers

ECI - Enable clock interrupts

EEX - Error Exit flag

EFI - Enable floating-point interrupt

ESVL - Enable Second Vector Logical (Exchange Package)

EX - Normal exit

F

F - Flag register (Exchange Package)

F - Floating-point operation (symbolic instructions)

FEI - Front-end interface

FOL-3 - Fiber-optic link (3 Mbyte/s)

FPE - Floating-point Error flag

FPS - Floating-point Error Status flag

FWA - First Word Address

H

H - Half-precision floating-point operation (symbolic instructions)

HSX-1 - High-speed External Channel

I

I - Reciprocal iteration (symbolic instructions)

IBA - Instruction Base Address register (Exchange Package)

ICM - Correctable Memory Error Mode flag

ICP - Interrupt From Internal CPU flag

IFP - Floating-point Error Mode flag

ILA - Instruction Limit Address register (Exchange Package)

IMM - Interrupt Monitor Mode flag

IOI - I/O Interrupt flag

IOP - I/O Processor

IOR - Operand Range Error Mode flag

IOS - I/O Subsystem

IUM - Uncorrectable Memory Error Mode flag

J

JAM - acronym for a series of branch instruction

JAN - acronym for a series of branch instruction

JAP - acronym for a series of branch instruction

JAZ - acronym for a series of branch instruction

JR - acronym for a series of branch instruction

JSM - acronym for a series of branch instruction

JSN - acronym for a series of branch instruction

JSP - acronym for a series of branch instruction

JSZ - acronym for a series of branch instruction

L

LIP - Last Instruction Parcel

M

M - Mode register (Exchange Package)

MC - Master Clear

MCU - MCU Interrupt flag

MCU - Maintenance control unit

ME - Memory Error flag

MIOP - Master I/O Processor

MM - Monitor Mode flag

N

NEX - Normal Exit flag

NIP - Next Instruction Parcel

O

ORE - Operand Range Error flag

P

P - Population count (symbolic instructions)

P register - Program Address register (Exchange Package)

PCI - Programmable Clock Interrupt flag

PDU - Power distribution unit

PN - Processor number (Exchange Package)

PRE - Program Range Error flag

PS - Program State register (Exchange Package)

Q

Q - Parity count (symbolic instructions)

R

R - Read mode (Exchange Package)

R - Rounded floating-point operation (symbolic instructions)

RT - Real-time clock (symbolic instructions)

RTC - Real-time clock

S

S - Syndrome bits (Exchange Package)

S register - Scalar register

SB - Shared Address register

SB - Sign Bit (symbolic instructions)

SEI - Selected for External Interrupts flag

SIPI - Set Interprocessor Interrupt

SM - Semaphore register

SSD - Solid-State storage device

ST - Shared Scalar register

T

<u>T register</u> - Scalar-save register

V

<u>VL</u> - Vector Length

<u>VM</u> - Vector Mask

<u>VNU</u> - Vector Not Used (Exchange Package)

W

<u>WS</u> - Waiting for Semaphore flag

X

<u>XA</u> - Exchange Address register (Exchange Package)

<u>XIOP</u> - Auxiliary I/O Processor

Z

<u>Z</u> - Leading-zero count (symbolic instructions)

INDEX

INDEX

READER'S COMMENT FORM

CRAY X-MP Computer Systems Functional Description Manual HR-3005

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
   ____ other (please specify): ______________________
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
   ____ for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy ____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name ____________	Address ____________
Title ____________	City ____________
Company ____________	State/ Country ____________
Telephone ____________	Zip Code ____________
Today's Date ____________	

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

STAPLE