CRAY Y-MP® EL Functional Description

HR-04027

Cray Research, Inc.

For FCC information, please contact your Cray Research account manager.

Requests for copies of Cray Research, Inc. publications should be directed to:

CRAY RESEARCH, INC.
Distribution
2360 Pilot Knob Road
Mendota Heights, MN 55120
(800) 284-2729 extension 5907

Comments about this publication should be directed to:

CRAY RESEARCH, INC.
Hardware Publications and Training
770 Industrial Blvd.
Chippewa Falls, WI 54729

Record of Revision

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 which is indicated in the publication number on each page of the manual.

Changes to part of a page are indicated by a change bar in the margin directly opposite the change. A change bar in the footer indicates that most, if not all, of the page is new. If the manual is rewritten, the revision level changes but the manual does not contain change bars.

REVISION	DESCRIPTION
	Original printing. January 1992.
001	August 1992. This change packet updates the DAS-2 disk array subsystem transfer-rate information.

PREFACE

This manual is written primarily for customers of Cray Research, Inc. and describes the basic functions of the CRAY Y-MP EL system manufactured by Cray Research. It also describes the design and architecture of the CRAY Y-MP EL system and its associated peripheral devices.

- Section 1 provides an overview of the major components of a CRAY Y-MP EL system.
- Section 2 describes the major functional areas of the CPU and provides a summary of the CPU instruction set.
- Section 3 describes the input/output subsystem (IOS) and system peripherals.

Each section has a separate table of contents.

Please use one of the reader comment forms located at the front and back of this manual to suggest improvements or to point out technical errors.

The following conventions are used throughout this manual.

Convention	Description
Lowercase italic	Variable information.
X or x or *x*	An unused value.
n	A specified value.
(value)	The **contents** of the register or memory location designated by value.
Register bit	Register bits are numbered from right to left as powers of 2.

Convention	Description
Designators	Bit 2^0 corresponds to the least significant bit of the register. One exception is the vector mask register. The vector mask register bits correspond to a word element in a vector register; bit 2^{63} corresponds to element 0 and bit 2^0 corresponds element 63.
Number base	All numbers used in this manual are decimal unless otherwise indicated. Octal numbers are indicated with an 8 subscript. Exceptions are register numbers, instruction parcels in instruction buffers, and instruction forms, which are given in octal without subscript notation.

The following are examples of the preceding conventions.

Example	Description
Transmit (A*k*) to S*i*	Transmit the contents of the A register specified by the *k* field to the S register specified by the *i* field.
167*ixk*	Machine instruction 167. The *x* represents the *j* field, which is not used.
Read n words from memory	Read a specified number of words from memory.
Bit 2^{63}	The value represents the most significant bit of an S register or element of a V register.
1000_8	The number base is octal

SECTION 1
SYSTEM OVERVIEW

CONTENTS

1 SYSTEM OVERVIEW

2 CPU

2 CPU (continued)

3 INPUT/OUTPUT SUBSYSTEM

GLOSSARY

BIBLIOGRAPHY

INDEX

FIGURES

FIGURES (continued)

TABLES

1 SYSTEM OVERVIEW

The CRAY Y-MP EL minisupercomputer system (refer to Figure 1-1) makes real supercomputing available to more prospective supercomputer users with its affordable entry-level system. It is easy to install and can be installed in almost any office environment. Each cabinet is air cooled and consumes a maximum of only 6 kW of power.

The CRAY Y-MP EL system is a minisupercomputer system that includes one to four 30-ns central processing units (CPUs), central memory, input/output subsystem (IOS), and disk and tape storage units that are all contained in one mainframe cabinet (cabinet 1). One or more optional peripheral cabinets may contain additional IOSs and optional peripheral devices (cabinets 2, 3, and 4). A typical layout of each of these cabinets is shown in Figure 1-1.

The CRAY Y-MP EL system features the following large-scale supercomputer features:

- Vector capability
- Parallel processing
- Memory capacity of 32, 64, or 128 Mwords that use dynamic random-access memory (DRAM) components
- I/O capacity
- Powerful and productive software
- Use of existing CRAY Y-MP supercomputer applications

The following subsections discuss the mainframe and peripheral cabinet and their components. Section 2 contains a more detailed description of the CPU; Section 3 describes the IOS.

Mainframe Cabinet

The mainframe cabinet is a stand-alone system that acts as the base cabinet in a system that can be added to simply by attaching a peripheral cabinet that contains additional IOSs and optional peripheral devices. The amount of memory or number of CPUs in the mainframe cabinet

may be field upgraded by adding additional CPU modules and by adding memory components to the existing memory modules. The mainframe cabinet contains the following components:

- One to four CPUs, each on a single module
- 32, 64, or 128 Mwords of memory across four modules
- Up to four VME-based IOSs
- Optional peripheral devices
- Power supplies
- Cooling fans

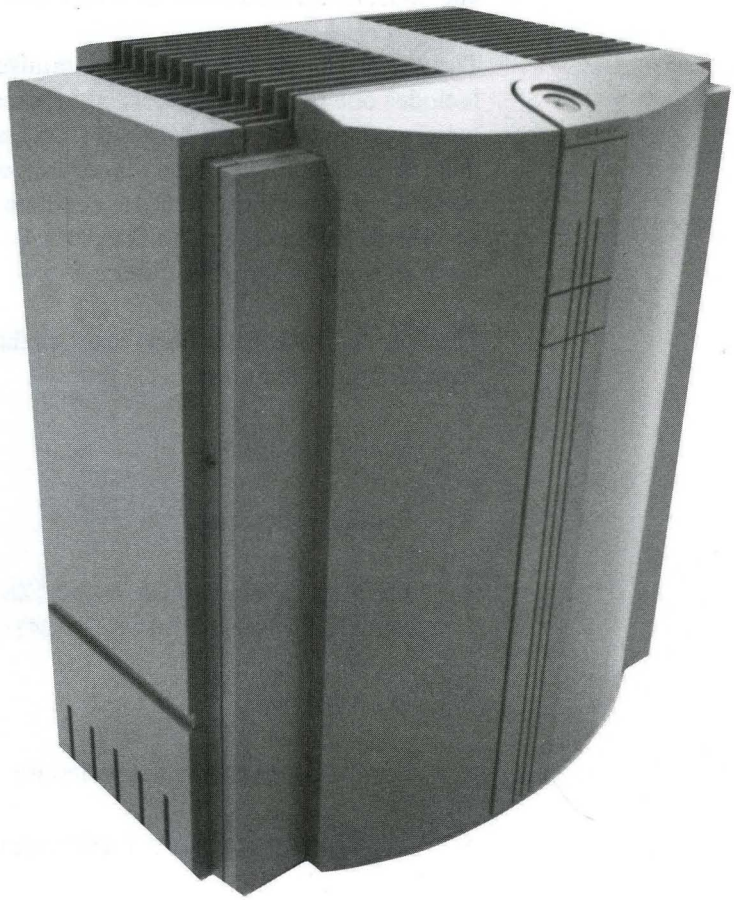

Figure 1-1. CRAY Y-MP EL Computer System

Peripheral Cabinet

One to three peripheral cabinets can be attached to the mainframe when the system is ordered or as part of an upgrade. Peripheral cabinets may be added to provide up to 16 IOSs and over 200 Gbytes of disk storage. A peripheral cabinet can include the following components (refer to Figure 1-2):

- Up to 12 additional IOSs (up to 16 total for entire system)
- Additional peripheral configurations
- Power supplies
- Cooling fans

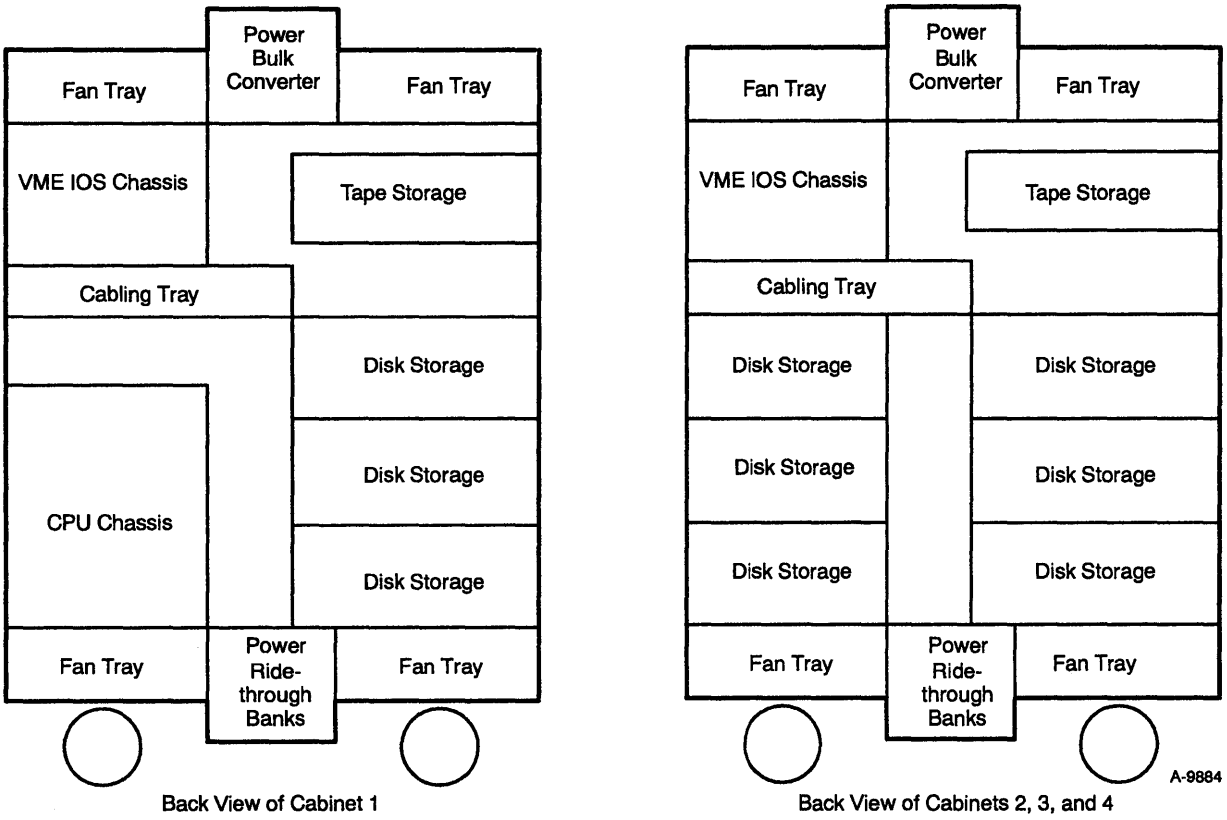

Figure 1-2. Example of Cabinet Layout for the CRAY Y-MP EL System

Disk Subsystems

Cray Research uses other manufacturers' disk subsystem products for the CRAY Y-MP EL system. The CRAY Y-MP EL system uses disk storage systems and disk array subsystems (DASs) for mass storage. The IOS

controls these devices via controller boards. Refer to Section 3 for a description of each disk storage device that is offered as optional equipment with the system.

Tape Subsystems

Cray Research uses other manufacturers' tape drive products for the CRAY Y-MP EL system; the IOS also controls these devices. An autoloading low-profile 9-track tape drive, 1/2-in. cartridge 3480-type tape drive, and an 8-mm helical scan tape system are the tape support options offered with the system. These are described in greater detail in Section 3.

Maintenance Workstation

The CRAY Y-MP EL system uses a maintenance workstation model EL (MWS-EL) that provides a dedicated platform for performing hardware maintenance and monitoring Cray Research, Inc. computer systems. Cray Research owns the MWS-EL and uses it for offline diagnostic testing, diagnostic listings, and hardware error logging. The MWS-EL is supplied as part of the maintenance contract. The MWS-EL is not part of the customer's system and is not connected to the customer's network. System operators use a system console, rather than an operator workstation (OWS).

Power and Cooling

The CRAY Y-MP EL system is completely air cooled and requires no raised floor or refrigeration. Fans are located at the top and bottom of each cabinet. Airflow through the frame is vertical, and the air is filtered at the base of the frame.

A power conditioning and distribution system is located in each mainframe and peripheral cabinet. The CPU and VME components have their own power supplies.

Network Interfaces

The CRAY Y-MP EL system can serve as a stand-alone system or can be networked into an existing computing environment. The system can be connected to a multiple-system network via HYPERchannel or Ethernet local area network using Transmission Control Protocol/Internet Protocol (TCP/IP). The network interfaces have controller boards in the IOS.

System Software

The following Cray Research, Inc. binary system software is shipped with the CRAY Y-MP EL system:

- UNICOS operating system derived from UNIX System Laboratories, Inc. UNIX System V
- Cray CF77 FORTRAN compiling system
- Cray ANSI standard C compiler
- Cray Assembly Language (CAL) - a versatile macro assembler
- Utilities
- Cray subroutine libraries

System Configurations

Customers can upgrade the number of CPUs, size of central memory, number of IOSs, and number and type of peripheral devices. The naming convention for these systems is CRAY Y-MP EL /n-y; n and y represent the following numbers:

n = Maximum number of mainframe CPUs the system can contain

y = Central memory size in megabytes (Mbytes); refer to Table 1-1.

For example, a CRAY Y-MP EL/4-1024 computer system has four CPUs and 1,024 Mbytes (128 Mwords) of central memory.

Table 1-1. CRAY Y-MP EL System Configurations

Product Name	CPUs	Memory Size	VME I/O Subsystems	Central Memory (Mbytes)
CRAY Y-MP EL/1-256	1	32 Mwords	1 to 4	256 (32 Mwords)
CRAY Y-MP EL/1-512	1	64 Mwords	1 to 4	512 (64 Mwords)
CRAY Y-MP EL/1-1024	1	128 Mwords	1 to 4	1,024 (128 Mwords)
CRAY Y-MP EL/2-256	2	32 Mwords	1 to 8	256 (32 Mwords)
CRAY Y-MP EL/2-512	2	64 Mwords	1 to 8	512 (64 Mwords)
CRAY Y-MP EL/2-1024	2	128 Mwords	1 to 8	1,024 (128 Mwords)
CRAY Y-MP EL/4-256	4	32 Mwords	1 to 16	256 (32 Mwords)
CRAY Y-MP EL/4-512	4	64 Mwords	1 to 16	512 (64 Mwords)
CRAY Y-MP EL/4-1024	4	128 Mwords	1 to 16	1,024 (128 Mwords)

SECTION 2
CPU

2 CPU

This section describes the major functional areas of a CRAY Y-MP EL central processing unit (CPU) and special features of the mainframe. Section 2 also provides a summary of the Cray Assembly Language (CAL) instruction set.

CPU Shared Resources

The CPUs share several functional areas (or sections) of the mainframe. These sections include central memory, the I/O section, the interprocessor communication section, and the real-time clock (RTC). The following subsections describe these functional areas.

Central Memory

Central memory is shared by the CPUs and the I/O section. Central memory is divided into sections and banks. This arrangement allows simultaneous and overlapping memory references. Simultaneous references are two or more references that begin at the same time; overlapping references are one or more references that begin while another reference is in progress.

Each CPU in the system has four parallel memory ports. Each port performs specific functions, allowing different types of memory transfers to occur simultaneously. To further enhance memory operations, the bidirectional memory mode allows block read and write operations to occur concurrently.

The mainframe has built-in conflict resolution hardware to minimize the delays caused by memory conflicts and to maintain the integrity of all memory references when conflicts occur. A memory conflict occurs when more than one reference is made to the same area of central memory.

To protect data, single-error correction/double-error detection (SECDED) logic is used in central memory and on data channels to or from central memory. When data is written into central memory, a checkbyte (an 8-bit Hamming code †) is generated for the word and stored with that

† Hamming, R. W. "Error Detection and Correcting Codes." *Bell System Technical Journal.* 29.2 (1950): 147 - 160.

word. When the word is read from central memory, the checkbyte and data word are processed to determine whether any bits were altered. If no errors occurred, the word is passed without modification.

If an error occurred, the 8 bits of the checkbyte are analyzed by the logic to find the number of altered bits. If only a single bit was altered, the correction logic resets that bit to the correct state and passes the corrected word on. The memory error flag in the exchange package sets to indicate that an error occurred, which can generate an interrupt. (Refer to "Flag Register Field" in this section for more information on the memory error flag.) Error information is also sent to an error logger.

If more than a single bit is altered, the logic cannot correct the word and the results are unpredictable. When a double error is detected, the memory error flag in the exchange package sets to indicate an error occurred, which can generate an interrupt.

IOS

Each CPU can be connected to a maximum of four IOSs, each via 40-Mbyte/s channels. These channels are used to transfer data between the CPU and the IOS. Refer to Section 3 for a more detailed description of the IOS.

Interprocessor Communication Section

The interprocessor 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.

The SB and ST registers pass address and scalar information from one CPU to another, while the SM registers control activity between CPUs.

Each CPU cluster number (CLN) register determines which set of shared registers is accessed by a CPU (clustering). The cluster may be accessed by any CPU to which it is allocated in either user or system (monitor) mode. Any CPU in monitor mode can interrupt any other CPU and cause it to switch from user to monitor mode. Additionally, each CPU 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; a deadlock condition occurs when all CPUs in the cluster are holding issue on a test and set instruction.

Real-time Clock

The mainframe contains one real-time clock (RTC) that is shared by all the CPUs. This clock consists of a 64-bit counter that advances one count each clock period (CP). Because the clock advances synchronously with program execution, it can be used to time the program to an exact number of CPs. Contents of the RTC register can be read into or loaded from a scalar (S) register.

CPU Computation Section

Each CPU has an identical, independent computation section consisting of operating registers, functional units, and instruction issue control logic. Figure 2-1 shows the computation section of CPU 0 for a four-processor mainframe. The operating registers and functional units of each computation section are associated with three types of processing: address, scalar, and vector.

Address processing operates on internal control information, such as loop counts, addresses, and indices. This processing uses address (A) registers and dedicated integer arithmetic functional units.

Address information originates from central memory, from constant instruction values, or from control registers. Information in the A registers is distributed to various parts of the instruction issue control for use in controlling the scalar, vector, and I/O operations. The A registers can also supply operands to the add and multiply functional units. The units generate address and index information and return the result to the A registers. Address information can also be transmitted to central memory from the A registers.

Scalar and vector processing can also be performed on data. Scalar processing occurs sequentially and uses one operand or operand pair to produce a single result. Scalar processing is performed using scalar (S) registers, several functional units dedicated solely to scalar processing, and additional floating-point functional units shared with vector operations.

Vector processing allows a single operation to be performed sequentially on a set (or vector) of operands, repeating the same function to produce a series of results. Vector processing is performed by vector (V) registers, several functional units dedicated solely to vector processing, and additional floating-point functional units supporting both scalar and vector operations.

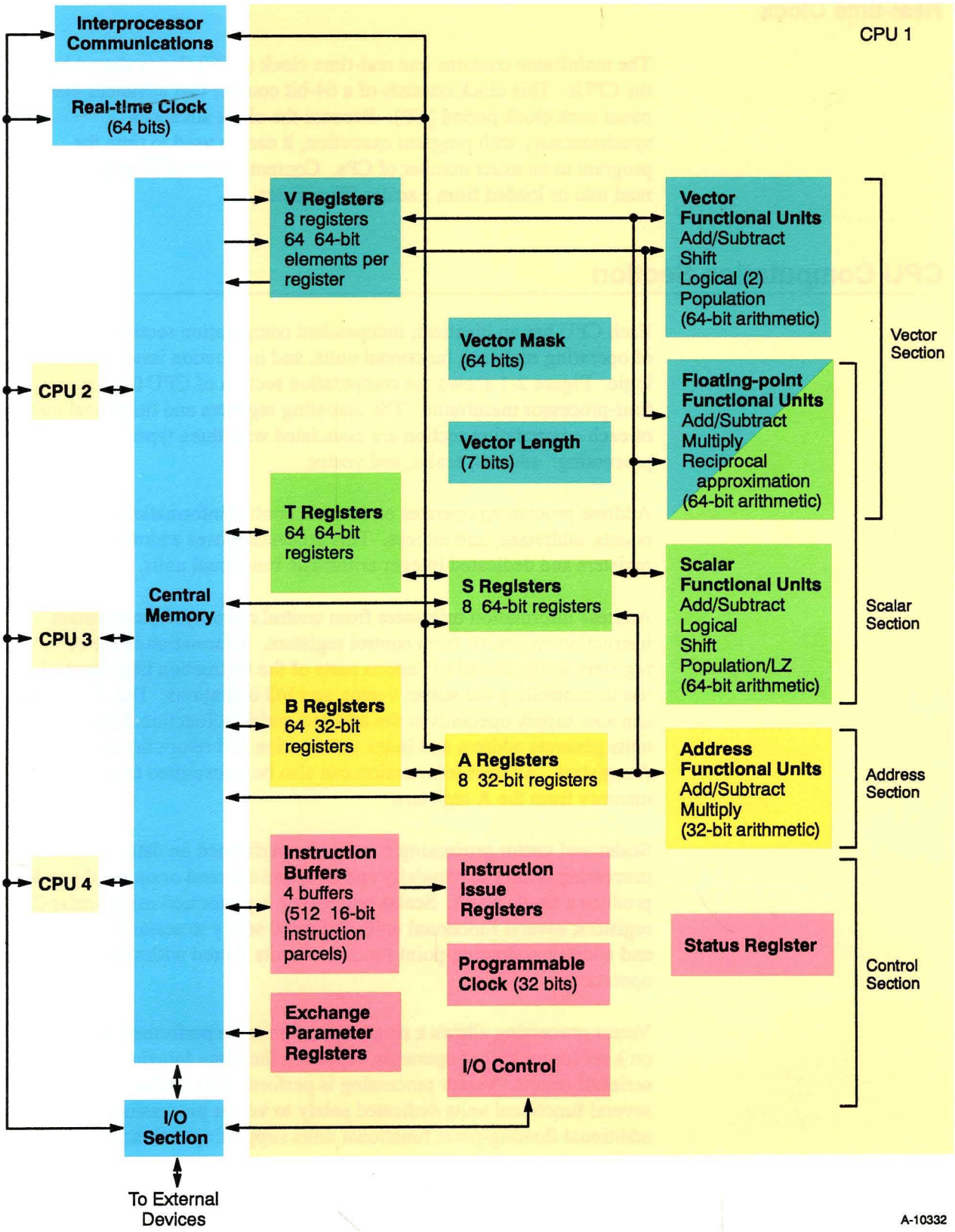

Figure 2-1. CRAY Y-MP EL System CPU Block Diagram

The main advantage of vector processing over scalar processing is that vector processing eliminates instruction start-up time for all but the first operand. Execution time for vector operations is short enough that vector processing is more efficient than scalar processing for vectors containing as few as two elements. Register-to-register vector instructions eliminate the problem of memory conflicts.

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 instruction. An exception is that scalar registers can provide one required operand for some vector instructions.

The computation section performs integer or floating-point arithmetic operations such as logical operations, indexing, population count, and leading-zero counts.

Integer, or fixed-point, operations are integer addition, integer subtraction, and integer multiplication. Integer arithmetic is performed in two's complement mode. No integer divide instruction is provided; the operation is accomplished through a software algorithm using floating-point hardware.

Floating-point instructions include addition, subtraction, multiplication, and reciprocal approximation. Floating-point quantities have signed magnitude representation. The reciprocal approximation instructions are used with a multiply instruction sequence to perform a floating-point divide operation.

The instruction set includes logical operations for AND, inclusive OR, exclusive OR, exclusive NOR, and 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 address integer arithmetic, most operations are implemented in vector and scalar instructions.

Index calculation is the integer product of a scalar instruction. Full indexing capability is possible 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 operations.

Population and parity count instructions are provided for both vector and scalar operations. An additional scalar operation is the leading zero count.

Registers

Each 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 central memory and the functional units can access them via intermediate registers.

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 data from 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. Data can then be moved from the intermediate registers to the primary register when needed. 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 intermediate scalar (T) registers.

Address Registers

Each CPU contains eight 32-bit A registers. The A registers serve a variety of applications, but are primarily used as address registers for memory references and as index registers. They also 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 register 0 contains the base address for scalar and vector memory references. In addition, the A registers provide an address increment for vector memory references.

Each CPU contains 64 B registers; each register is 32 bits wide. The B registers are used as intermediate storage for the A registers. Data is transferred between B registers and central memory, and between A and B registers. Typically, B registers contain data to be referenced repeatedly over a long time span, making it unnecessary to retain the data in either A registers or central memory. Examples of data stored in B registers are loop counts, variable array base addresses, and dimensions.

Address processing in the mainframe operates in two modes: the X-mode and the Y-mode. In the X-mode, the A registers, B registers, and the address functional units are limited to 24 bits, as in CRAY X-MP computer systems. Only 1- and 2-parcel instructions run in this mode. In the Y-mode, the A registers, B registers, and address functional units run at a full 32-bit width and the instruction set is expanded to include 3-parcel instructions. Refer to "Instruction Differences between the X-mode and Y-mode" in this section for more information on these modes and instructions.

Scalar Registers

Each CPU contains eight S registers; each register is 64 bits wide. The S registers are the principal scalar registers for a CPU. Scalar registers serve as the source and destination for scalar arithmetic and logical instructions. Scalar registers can also provide an operand for some vector operations.

Each CPU contains 64 T registers; each register is 64 bits wide. The T registers are used as intermediate storage for the S registers. Data is transferred between T registers and central memory, and between T and S registers.

Vector Registers

Each CPU contains eight vector (V) registers. Each V register consists of 64 elements; each element is 64 bits wide. The V registers serve as the source and destination for vector arithmetic and logical instructions. Successive elements from a V register are sent to a functional unit in successive CPs with a single instruction.

The effective length of a V register for any operation is controlled by the program-selectable vector length (VL) register. The VL register is a 7-bit register that specifies the number of vector elements processed by vector instructions. The contents range from 0_8 through 100_8.

The vector mask (VM) register allows for the logical selection of particular elements of a vector during merge and test instructions. The VM register has 64 bits, each corresponding to a word element in a V register. The high-order bit of the VM register corresponds to element 0 of the V register, while the low-order bit corresponds to element 63.

Refer to "Vector Processing" in this section for more information on vector processing.

Functional Units

Instructions other than simple transmit 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 have independent logic and can operate simultaneously.

All functional units perform algorithms in a fixed time; delays are impossible once the operands are 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 one CP. Refer to "Pipelining and

Segmentation" and "Functional Unit Independence" in this section for more information on segmentation, pipelining, and functional unit independence.

The functional units are described in four groups: address, scalar, vector, and floating-point. Each of the first three groups functions with one of the primary register types (A, S, and V) to support the address, scalar, and vector processing modes. The fourth group, floating-point, supports either scalar or vector operations and accepts operands from or delivers results to S or V registers. In addition, central memory can also act as a functional unit for vector operations.

Address Functional Units

Address functional units perform integer arithmetic on operands obtained from A registers and deliver the results to an A register (integer arithmetic is explained later in this section). The following list describes the two address functional units.

- The address add functional unit performs integer addition and subtraction; subtraction is performed by using two's complement arithmetic. Overflow is not detected.

- The address multiply functional unit forms an integer product from two operands. No rounding is performed and overflow is not detected. The unit returns only the least significant bits of the product.

Scalar Functional Units

Scalar functional units perform operations on operands obtained from S registers and usually deliver the results to an S register (integer arithmetic is explained later in this section). 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 count functional units are used exclusively with scalar operations and are described here. Three additional functional units are used for both scalar and vector operations. They are described in the following "Floating-point Functional Units" subsection. The following list describes the four scalar functional units.

- The scalar logical functional unit performs bit-by-bit manipulation of S register operands.

- The scalar add functional unit performs integer addition and subtraction; subtraction is performed by using two's complement arithmetic. Overflow is not detected.

- 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. Single shifts are end-off with zero fill, while double shifts can be circular fill. Shift counts are obtained from an A register or from the *jk* field of the instruction.

- The scalar population/parity/leading zero count functional unit receives operands from an S register and delivers results to an A register. The population/parity functional unit counts the number of bits in an S register having a value of one and then, depending on the instruction issued, returns either a 7-bit population value or a 1-bit population parity count value. For the leading zero function, it counts the number of 0 bits preceding a 1 bit in the operand from left to right.

Vector Functional Units

Vector functional units perform operations on successive elements of one or two V registers, or on successive elements from a V register and a copy of an S register. The vector add and logical functional units require two operands, while the vector shift and population/parity functional units require only one operand. Results from a vector functional unit are delivered to a V register.

For operations using vector elements and a copy of a scalar register, the contents of the scalar register are transferred once to the functional unit and held there to be used as a constant. 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.

There are four general-purpose memory ports for each CPU: port A through port D, of which port D is used for I/O channels and as a fetch operations unit. The VL register determines the number of operands or operand pairs to be processed by a functional unit. Refer to "Special Features of the CPU" in this section for more information on vector processing, chaining, and other special vector processing features.

The CRAY Y-MP EL CPU features four sets of vector functional unit execution units (EUs), EU 0 through EU 3. Each EU contains a copy of every vector functional unit plus floating-point add and multiply functional units. There is only one copy of the reciprocal approximation unit. EU 3 is the only unit that executes vector mask instructions. Each

EU has one data path to registers. As a result, a CPU can simultaneously perform four vector add or four vector multiply operations, or any combination thereof.

The functional units described in this subsection are used exclusively with vector operations. Three functional units are used with both vector and scalar operations, and are described in the following "Floating-point Functional Units" subsection. The following list describes the four vector functional units.

- The vector add functional unit performs integer addition and subtraction for a vector operation and delivers the results to elements of a V register. Subtraction is performed by using two's complement arithmetic. Overflow is not detected.

- 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. All shifts are end-off with zero fill.

- 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 instructions.

- The vector population/parity functional unit counts the number of bits set in each element of the source V register; the result 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. The vector population count parity instruction delivers only the low-order bit of the count to the destination V register. This bit can be used to determine even or odd parity.

Floating-point Functional Units

There are four sets of floating-point functional units for each CRAY Y-MP EL CPU. Each set consists of three floating-point functional units that perform floating-point arithmetic for scalar and vector operations in a CRAY Y-MP EL CPU. When a scalar instruction issues, 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 functional unit, which requires only one input operand. When a

floating-point functional unit is used for a vector operation, the general description of vector functional units given in the individual subsection applies.

The following list describes the three floating-point functional units.

- 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. The floating-point add functional unit detects overflow and underflow conditions; only overflow conditions are flagged. (Refer to "Normalized Floating-point Numbers" in this section for more information on normalized numbers.)

- The floating-point multiply functional unit performs full- and half-precision multiplication of operands in floating-point format. The half-precision product is rounded; the full-precision product can be rounded or not rounded. This functional unit also generates a 32-bit integer product.

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

  Out-of-range exponents are detected. If both operands have zero exponents, the result is considered an integer product, is not normalized, and is not considered to be out of range.

- The reciprocal approximation functional unit calculates the approximate reciprocal of an operand in floating-point format. The input operand is assumed to be normalized. The high-order bit of the coefficient is not tested, but is assumed to be a 1. Out-of-range exponents are detected. Reciprocal approximation operations are used in conjunction with multiply operations to perform an add operation.

Functional Unit Operations

Functional units in a CPU perform logical operations, integer arithmetic, and floating-point arithmetic. Both types of arithmetic are performed using two's complement arithmetic. The following subsections explain and define the logical operations, the integer arithmetic, and the floating-point arithmetic used by the mainframe.

Logical Operations

Scalar and vector logical units perform bit-by-bit manipulation of 64-bit quantities. Instructions are provided for forming logical products, sums, differences, equivalences, and merges.

A logical product is the AND function shown below.

```
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 shown below.

```
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 shown below.

```
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 shown below.

```
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 shown below. The bits of operand 1 are transmitted when the mask bit is a 1. The bits of operand 2 are transmitted when the mask bit is a 0.

```
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
```

Integer Arithmetic

All integers, whether 24, 32, or 64 bits, are represented in the registers as shown in Figure 2-2. The address add and multiply functional units perform 24-bit arithmetic in X-mode and 32-bit arithmetic in Y-mode (refer to "Instruction Differences between X-mode and Y-mode" in this section for more information on these modes). The scalar add and vector add functional units perform 64-bit arithmetic.

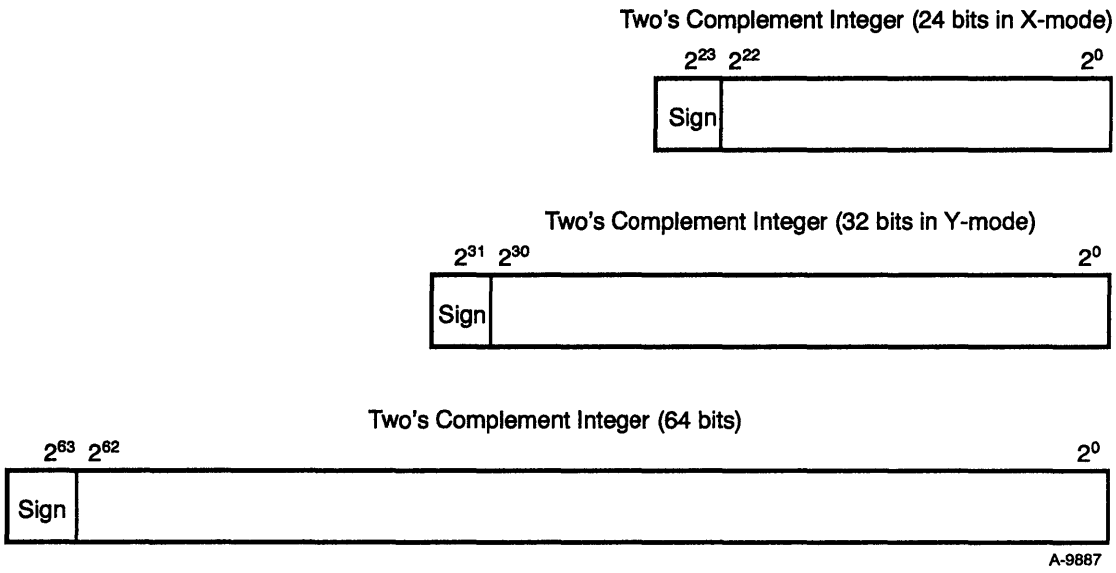

Figure 2-2. Integer Data Formats

Multiplication of two scalar (64-bit) integer operands is done using the floating-point multiply instruction and one of two multiplication methods. The method used depends on the magnitude of the operands and the number of bits available to contain the product. The following paragraphs explain the 24-bit integer multiply operation and the method used for operands greater than 24 bits in length.

The floating-point multiply functional unit recognizes the condition in which both operands have zero exponents as a special case; it is treated as an integer multiply operation, and a complete multiply is performed with no truncation as long as the total number of bits in the two operands does not exceed 48 bit positions. To multiply two integers together, set each operand's exponent equal to zero and place each 24-bit integer value in bit positions 2^{47} through 2^{24} of the operand's coefficient field. To ensure accuracy, the least significant 24 bits must be 0.

When the floating-point multiply functional unit has performed the operation, it returns the high-order 48 bits of the product as the result coefficient and leaves the exponent field as 0. The result is a 48-bit quantity in bit positions 2^{47} through 2^{0}; no normalization shift of the result is performed.

As shown in Figure 2-3, if operand 1 is 4_8 and operand 2 is 6_8, a 48-bit result of 30_8 is produced. Bit 2^{63} follows the usual rules for multiplying signs and the result is a sign-magnitude integer. Both the hardware and software require two's complement format, not sign-magnitude format. Therefore, negative products must be converted to two's complement form.

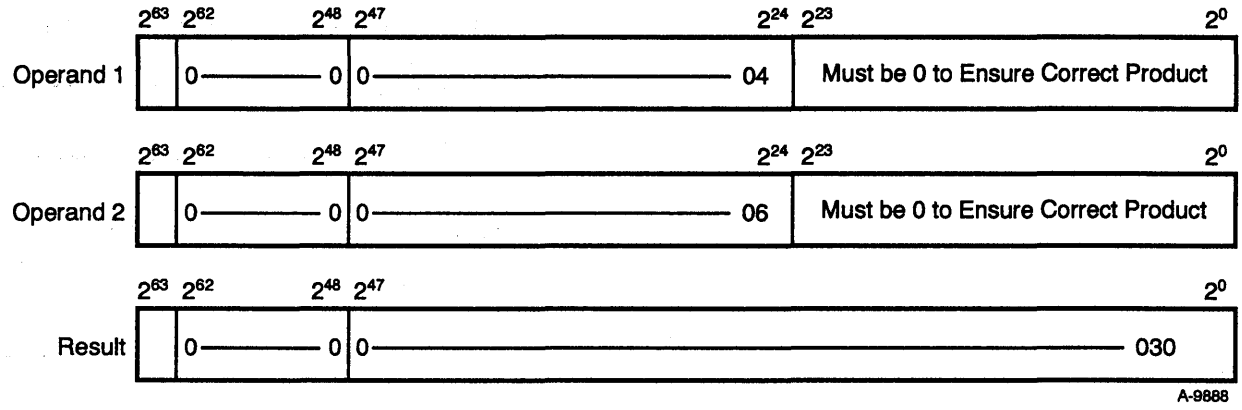

Figure 2-3. 24-bit Integer Multiply Performed in Floating-point Multiply Functional Unit

The second multiplication method is used when the operands are greater than 24 bits in length; multiplication is done by software forming multiple partial products and then shifting and adding the partial products.

A second integer multiplication operation performs a 32-bit multiplication operation on the S*j* and the V*k* operands and puts the result in the V*i* register. The operands must be left-shifted before the operation begins. The S*j* operand must be left-shifted 31_{10} places, leaving the operand in bit positions 2^{62} through 2^{31}; bit positions 2^{30} through 2^{0} must be equal to 0 to ensure accuracy (refer to Figure 2-4). The V*k* operand must be shifted left 16_{10} places, leaving the operand in bit positions 2^{47} through 2^{16}; bit positions 2^{15} through 2^{0} must be equal to 0 to ensure accuracy. The result of the multiply is right justified into positions 2^{31} through 2^{0}, while positions 2^{32} through 2^{63} are filled with zeros.

Although no integer divide operation is provided, integer division can be carried out by converting the numbers to the floating-point format and then using the floating-point functional units. Refer to "Floating-point Division Algorithm" in this section for more information.

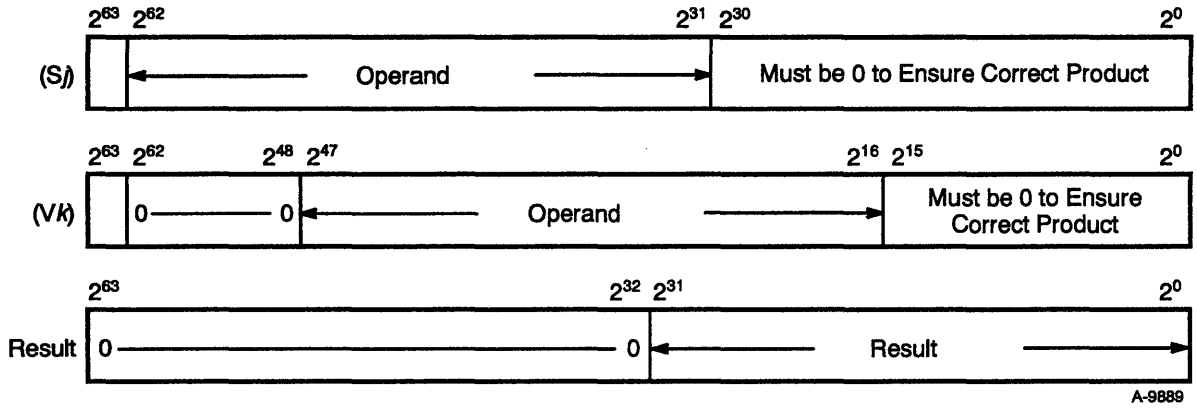

Figure 2-4. 32-bit Integer Multiply Performed in Floating-point Multiply Functional Unit

Floating-point Arithmetic

Floating-point arithmetic is used by the scalar and vector instructions. The following subsections explain the floating-point data format, exponent ranges, normalized floating-point numbers, floating-point range errors, the floating-point addition, multiplication, and division algorithms, and double-precision numbers.

Floating-point Data Format

Floating-point numbers are represented in a standard format throughout the CPU; this format is shown in Figure 2-5. The format has three different fields: coefficient sign, exponent, and coefficient.

This format is a packed representation of a binary coefficient and an exponent (power of two). The coefficient sign is located in bit position 2^{63} and is separated from the rest of the coefficient. If this bit is equal to 0, the coefficient is positive; if this bit is equal to 1, the coefficient is negative. The exponent is represented as a biased integer number in bit positions 2^{62} through 2^{48}; each exponent is biased by 40000_8. Bit 2^{61} is the sign of the exponent; a 0 indicates a positive exponent, while a 1 indicates a negative exponent. Bit 2^{62} is the bias of the exponent.

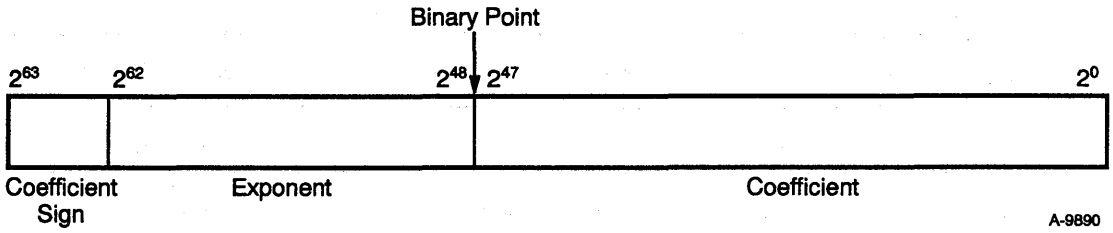

Figure 2-5. Floating-point Data Format

The coefficient is a 48-bit signed fraction; the sign of the coefficient is located in bit position 2^{63}. Because the coefficient is in sign-magnitude format, it is not complemented for negative values. A normalized floating-point number has a 1 in the 2^{47} bit position, while an unnormalized floating-point number has a 0 in this bit position (normalized numbers are discussed in more detail later in this section).

Figure 2-6 and the following steps show the relation between the bias, exponent, and coefficient.

To convert a floating-point number to its decimal equivalent:

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

$$\begin{array}{r} 40011_8 \\ -\ 40000_8 \\ \hline 11_8 \end{array} = 9_{10}$$

2. Multiply the normalized coefficient by the power of 2 indicated in the exponent to get the result:

$$0.5634_8 \times 2^9 = 563.40_8 = 371.5_{10}$$

A zero value or an underflow result is not biased and is represented as a word of all 0's. A negative 0 is not generated by any floating-point functional unit, except when one operand going into the floating-point multiply or floating-point add functional unit has a value of negative 0.

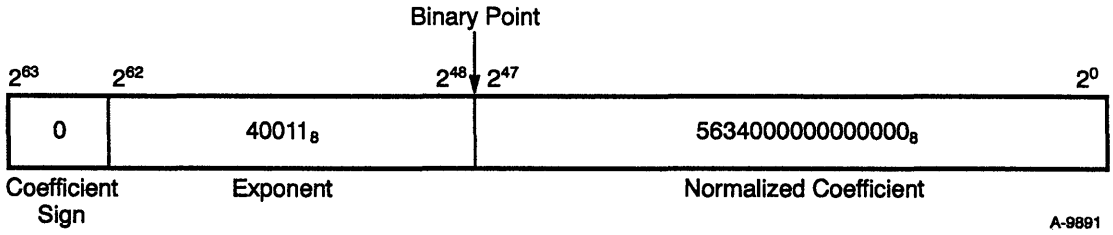

Figure 2-6. Internal Representation of Floating-point Number

Exponent Ranges

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, which represents an exponent of 2^0. Figure 2-7 shows the biased and unbiased exponent ranges.

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}.

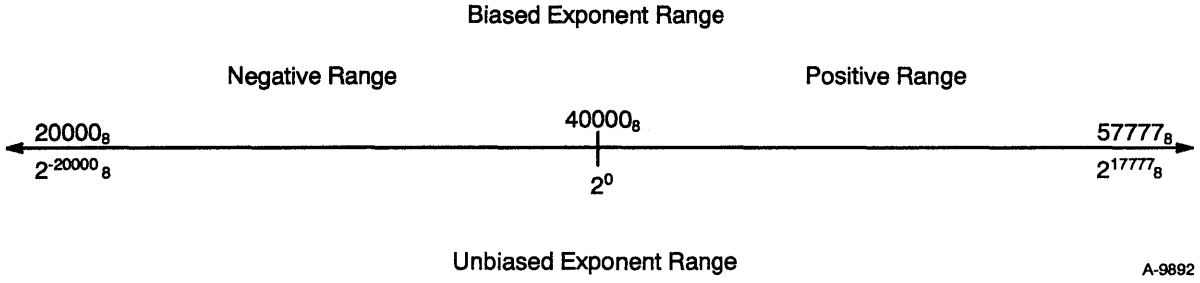

Figure 2-7. Biased and Unbiased Exponent Ranges

Normalized Floating-point Numbers

A nonzero floating-point number is normalized if the most significant bit of the coefficient, bit 2^{47}, 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, a normalized floating-point number has no leading 0's in its coefficient. The exception is a normalized floating-point 0, which is all 0's.

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.

The reciprocal approximation functional unit must use normalized numbers to produce correct results; using unnormalized numbers will produce inaccurate results. The floating-point multiply functional unit does not require the use of normalized numbers to get correct results; however, more accurate results occur when normalized numbers are used.

The floating-point add functional unit does not require normalized numbers to get correct results. The floating-point add functional unit does, however, automatically normalize all its results; unnormalized floating-point numbers may be routed through this functional unit to take advantage of this process.

Floating-point Range Errors

To make sure the limits of the functional units will not be exceeded, a range check is made on the exponent of each floating-point number for overflow and underflow conditions. In the floating-point add and multiply functional units, bits 2^{61} and 2^{62} are checked; if both bits are equal to 1, the exponent is equal to or greater than 60000_8 and an overflow condition is detected. The calculated coefficient is reported, but the exponent is set to 60000_8 and the floating-point error flag is set (refer to Figure 2-8).

When an overflow condition is detected, an interrupt occurs only if the interrupt-on floating-point error (IFP) bit is set in the mode register and the system is not in monitor mode. The IFP flag can be set or cleared by a user mode program.

To check for an underflow condition in the floating-point add and multiply functional units, bits 2^{61} and 2^{62} are checked; if both are equal to 0, then the exponent is less than or equal to 17777_8 and an underflow condition is detected. No flag is set, but the exponent and coefficient are both set to 0's (refer to Figure 2-8).

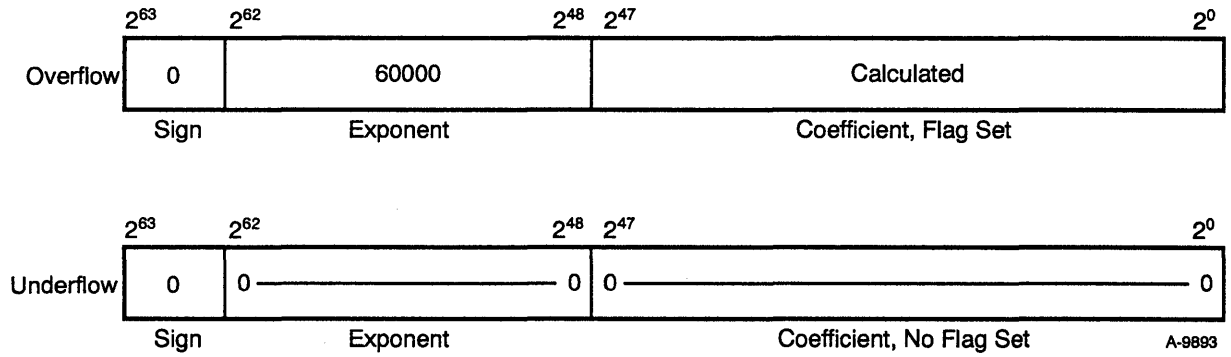

Figure 2-8. Floating-point Add and Multiply Range Errors

In the reciprocal approximation functional unit, the exponent is complemented and the value of 2 is added before the operation proceeds. When the check is made in a reciprocal approximation operation, the exponent must be equal to or greater than 60002_8 to cause an overflow

condition. In this case, the calculated coefficient is reported, but bit 2^{47} is set to 0, the exponent is set to 60000_8, and the floating-point error flag is set (refer to Figure 2-9).

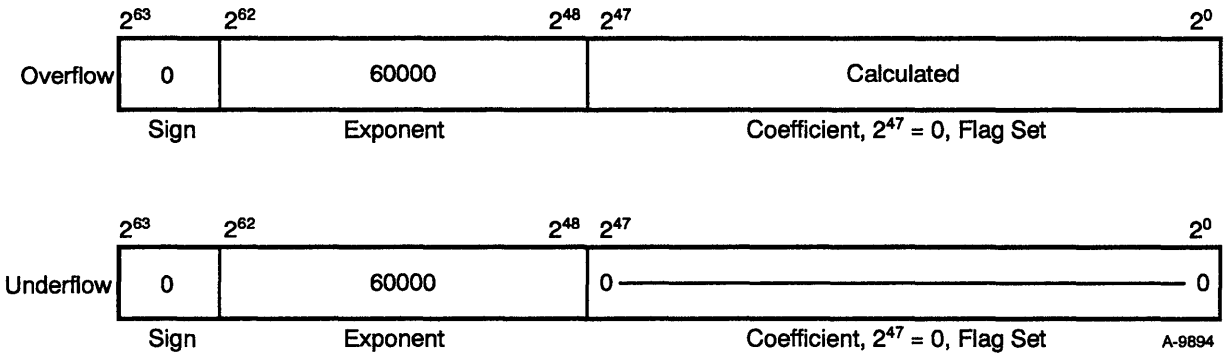

Figure 2-9. Floating-point Reciprocal Approximation Range Errors

Again, because the reciprocal approximation operation complements and adds 2 to a floating-point number, the exponent must be less than or equal to 20001_8 for an underflow condition to occur. This underflow condition then causes an overflow condition in the original exponent. In this case, the calculated coefficient is reported, but bit 2^{47} is set to 0, the exponent is set to 60000_8, and the floating-point error flag is set (refer to Figure 2-9).

Floating-point Addition Algorithm

Floating-point addition or subtraction is performed in a 49-bit register to allow for a sum that might carry into an additional bit position. The algorithm performs three operations: equalizing exponents, adding coefficients, and normalizing the results.

To equalize the exponents, the larger of the two exponents is retained. The coefficient of the smaller exponent is right-shifted by the difference of the two exponents, or until both exponents are equal. Bits shifted out of the register are lost; no round-up operation occurs. Because the coefficient is only 48 bits in length, any shift beyond 48 bits causes the smaller coefficient to become 0's.

After equalizing the two coefficients, they are added together. Two conditions are analyzed to determine whether an addition or subtraction operation occurs. The two conditions are the sign bits of the two coefficients and the type of instruction (an add or subtract) issued.

The following list shows how the operation is determined.

- If the sign bits are equal and an add instruction is issued, an add operation is performed.
- If the sign bits are not equal and an add instruction is issued, a subtraction operation is performed.
- If the sign bits are equal and a subtract instruction is issued, a subtract operation is performed.
- If the sign bits are not equal and a subtract instruction is issued, an add operation is performed.

The last operation performed normalizes the results. To normalize the result, the coefficient is left-shifted by the number of leading 0's (the coefficient is normalized when bit 2^{47} is a 1). The exponent must also decrement accordingly. If a carry across the binary point occurs during an addition operation, the coefficient is right-shifted by 1 and the exponent increases by 1. If a carry across the binary point occurs during a subtraction operation, an end-around carry occurs.

The normalization feature of the floating-point add functional unit can be used to normalize any floating-point number. Adding an unnormalized number to a zero operand will normalize the number.

A range check is performed on the result of all additions; refer to "Floating-point Range Errors" in this section for more information on how the result is checked.

Floating-point Multiplication Algorithm

The floating-point multiply functional unit receives two floating-point operands from either an S or V register. The signs of the two operands are combined using an exclusive OR operation, the exponents are added together, and the two 48-bit coefficients are multiplied together. If the coefficients are both normalized, multiplying them together produces a full product of either 95 or 96 bits. A 96-bit product is normalized as generated, while a 95-bit product requires a left-shift of one to generate the final coefficient. If the shift is performed, the final exponent is reduced by 1 to reflect the shift.

Because the result register (an S or V register) can hold only 48 bits in the coefficient, only the upper 48 bits of the 96-bit result are used. The lower 48 bits are never generated. The following paragraphs describe the truncation process used to compensate for the loss of bits in the product. It assumes no shift was required to generate the final product; power of two designators are used.

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 9.25×2^{-56}, which is 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 inserted at bit position 2^{-56} 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% 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, $A \times B = B \times A$.

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

For a half-precision multiply, round bits are entered into the summation 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 one of the following ranges:

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

or

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

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

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

Floating-point Division Algorithm

The mainframe does not have a single functional unit that is dedicated to the division operation. Rather, the floating-point multiply and reciprocal approximation functional units together carry out the algorithm. The following paragraphs explain how the algorithm is determined and how it is used in the functional units.

Finding the quotient of two floating-point numbers involves two steps. For example, to find the quotient of A/B, first, the B operand is sent through the reciprocal approximation functional unit to obtain its reciprocal, 1/B. Then, this result, along with the A operand is sent to the floating-point multiply functional unit to obtain the product of A x 1/B.

The reciprocal approximation functional unit uses an application of Newton's method for approximating the real root of an arbitrary equation, F(x) = 0, to find reciprocals.

To find the reciprocal, the equation, F(x) = 1/x – B, must be solved. To do this, a number, A, must be found so that F(A) = 1/A – B = 0. That is, the number A is the root of the equation 1/x – B = 0. The method requires an initial approximation (or guess, which is shown as x_0 in Figure 2-10) sufficiently close to the true root (which is shown as x_t in Figure 2-10). x_0 is then used to obtain a better approximation; this is done by drawing a tangent line (line 1 in Figure 2-10) to the graph of y = F(x) at the point $[x_0, F(x_0)]$. The intercept of this tangent line becomes the second approximation, x_1. This process is repeated, using tangent line 2 to obtain x_2, and so on.

The following iteration equation is derived from this process:

$$x_{(i+1)} = 2x_i - x^2{}_i B = x_i (2 - x_i B)$$

In the equation, $x_{(x+1)}$ is the next iteration, x_i is the current iteration, and B is the divisor. Each $x_{(i+1)}$ is a better approximation than x_i to the true value, x_t. The exact answer is generally not obtained at once because the correction term is not exact. The operation is repeated until the answer becomes sufficiently close for practical use.

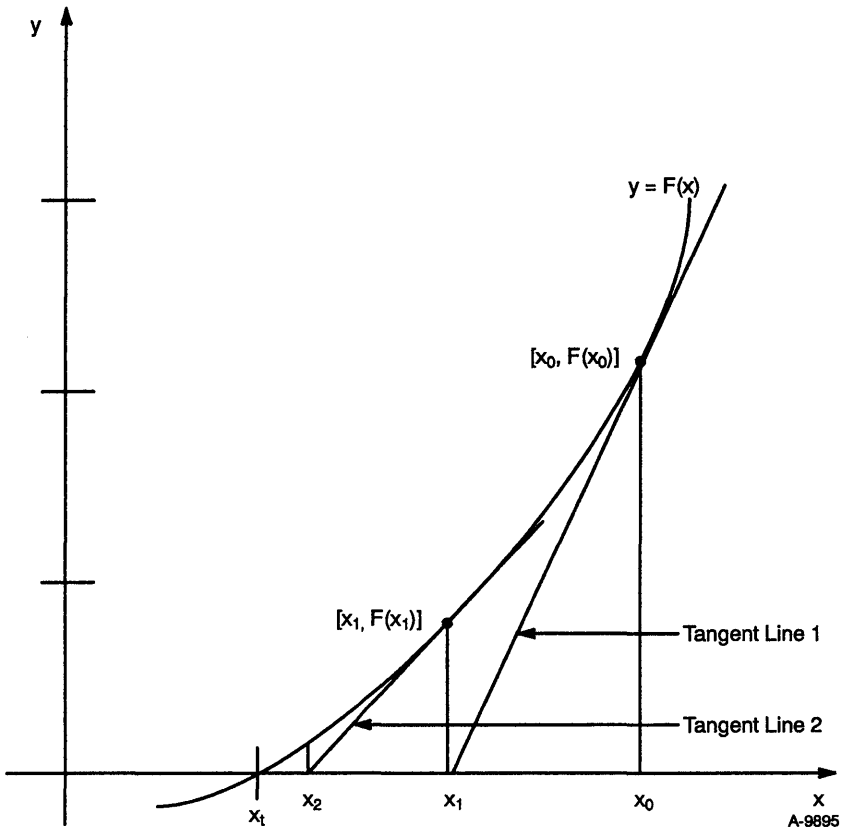

Figure 2-10. Newton's Method for Approximating Roots

The CPU uses this approximation technique based on Newton's method. A hardware lookup table provides an initial guess, x_0, to start the process. The following iterations are then calculated.

Iteration	Operation	Description
1	$x_1 = x_0(2 - x_0B)$	The first approximation is done in the reciprocal approximation functional unit and is accurate to 8 bits.
2	$x_2 = x_1(2 - x_1B)$	The second approximation is done in the reciprocal approximation functional unit and is accurate to 16 bits.
3	$x_3 = x_2(2 - x_2B)$	The third approximation is done in the reciprocal approximation functional unit and is accurate to 30 bits.

4	$x_4 = x_3(2 - x_3B)$	The fourth approximation is done in the floating-point multiply functional unit to calculate the correction term.

The reciprocal approximation functional unit calculates the first three iterations, while the the floating-point multiply functional unit calculates the fourth iteration. The 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 approximation functional unit's answer to full precision (the floating-point multiply functional unit can provide both full- and half-precision results). A fifth iteration should not be done.

The following example shows how the floating-point multiply functional unit is used to provide a full-precision result, solving the equation S1/S2.

Step	Operation	Performed By
1	S3 = 1/S2	Reciprocal approximation functional unit
2	S4 = [2 – (S3 * S2)]	Floating-point multiply functional unit
3	S5 = S4 * S3	Floating-point multiply functional unit using full-precision; S5 now equals 1/S2 to 48-bit accuracy
4	S6 = S5 * S1	Floating-point multiply functional unit using full-precision rounded

The reciprocal approximation in Step 1 is correct to 30 bits. By Step 3, it is accurate to 48 bits. This iteration answer is applied as an operand in a full-precision rounded multiply operation (Step 4) to obtain a quotient accurate to 48 bits. Additional iterations may produce erroneous results.

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, as shown in the following example.

Step	Operation	Performed by
1	S3 = 1/S2	Reciprocal approximation functional unit

Step	Operation	Performed by
2	S6 = S1 * S3	Floating-point multiply functional unit in half-precision

The 19 low-order bits of the half-precision results are returned as 0's with a rounding applied to the low-order bit of the 29-bit result.

The reciprocal iteration is designed for use once with each half-precision reciprocal generated. If the iteration performed by the floating-point multiply functional unit 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.

The following process is another method of computing division:

Step	Operation	Performed by
1	S3 = 1/S2	Reciprocal approximation functional unit
2	S5 = S1 * S3	Floating-point multiply functional unit
3	S4 = [2 – (S3 * S2)]	Floating-point multiply functional unit
4	S6 = S4 * S5	Floating-point multiply functional unit

In this method the correction to reach a full-precision reciprocal is done after the number is multiplied by the half-precision reciprocal, rather than before the multiplication.

The coefficient of the reciprocal produced by this alternative method can be different by as much as 2×2^{-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, use the same method each time the reciprocals are generated.

Double-precision Numbers

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

CPU Control Section

The control section of the CPU issues program instructions. Before program instructions can issue, exchange and instruction fetch sequences must occur. The following subsections describe the exchange mechanism (which includes defining both the exchange package and exchange sequence), and the instruction fetch and instruction issue sequences.

Exchange Mechanism

Each CPU uses an exchange mechanism for switching instruction execution from program to program. This exchange mechanism uses a CPU operation referred to as an exchange sequence and blocks of program parameters known as exchange packages.

Exchange Sequence

An exchange sequence occurs before a program can begin running. An exchange sequence performs two simultaneous functions. First, program parameters for the next program are loaded from central memory into registers in the CPU. Second, parameters from the previous program are read from the registers and stored back into central memory.

The program parameters are held in an exchange package, which is described in the following subsections. The contents of the A and S registers are automatically saved in the 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. All instructions previously issued are allowed to complete before the exchange sequence begins. An instruction fetch always follows an exchange sequence. Refer to "Instruction Fetch" in this section for more information on this sequence.

Exchange Package

The exchange package is composed of a number of parameters, which are held in fields. These fields contain the contents of certain registers that are swapped during an exchange sequence. The following subsections define the fields of the exchange package.

Processor Number Field

The contents of the processor number (PN) field in the exchange package indicates which CPU performed the exchange sequence. This value is not stored initially in the exchange package before program execution; it is a constant inserted into the exchange package after the program ran and exchanged out.

Memory Error Data Fields

Memory error data, consisting of six fields of information, appears in the exchange package only if one of two conditions is met. The first condition is that the interrupt-on-correctable memory error bit is set and a correctable memory error is encountered. The second condition is that the interrupt-on-uncorrectable memory error bit is set and an uncorrectable memory error is detected. Memory error data fields are described below.

- The syndrome field defines a SECDED error on a memory read or I/O channel.
- The read address bank field defines the bank where a memory read error occurred.
- The read error type field defines the type of memory or I/O error encountered; the error can be either correctable or uncorrectable.
- The port field defines the port where a memory read or I/O error occurred; these bits are used with the read mode bits to identify the operation in error.
- The read address chip select field identifies the chip on which a memory read error occurred.
- The read mode field determines the type of read mode in progress when a memory data error occurred; these bits are used with the port bits to identify the operation in error.

Program Address Register Field

The program address (P) register contents are stored in this field of the exchange package. The instruction at this location is the first instruction issued when this program begins.

Address Base and Limit Fields

Four registers in the exchange package define a program's data range and instruction range anywhere in memory and allocate specific amounts of memory to each range. This memory allocation technique has two benefits. First, all programs are relocatable. When a program is written, the programmer does not need to know where in memory the instruction and data fields will be located. Second, each program can have its memory access restricted to certain parts of memory. A program can be halted if it tries to run an instruction outside of its allowed instruction range or if it tries to read or write data outside of its allowed data range. This is especially important where more than one program occupies memory at the same time; programs can be prevented from executing instructions or operating on data that belongs to other programs. The four registers are described in the following list.

- The instruction base address (IBA) register holds the base address of the user's instruction range. It determines where in memory an instruction fetch is made. This is done by adding the contents of the P register to the contents of the IBA register. The sum equals the absolute memory address for the fetch.

- The instruction limit address (ILA) register holds the upper limit address of the user's instruction range. It determines the highest absolute address that can be accessed during an instruction fetch sequence. If this absolute address exceeds the limit, a program range error flag is set, which generates an interrupt.

- The data base address (DBA) register holds the base address of the user's data range. It determines where in memory a program's data field is located. This is done by adding the memory address generated by the instruction to the contents of the DBA register. The sum equals the absolute address for any memory read or write operation.

- The data limit address (DLA) register holds the upper limit address of the user's data range. It determines the highest absolute memory address that a program can use for reading or writing data. If this absolute address exceeds the limit, the memory reference is aborted. The operand range error flag is set, which generates an interrupt if the interrupt-on-operand range error bit is set.

Exchange Address Register Field

The exchange address (XA) register specifies the first word address of a 16-word exchange package loaded by an exchange sequence. 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 because an

exchange package must begin on a 16-word boundary. The 12-bit limit requires that the absolute address be in the lower 4,096 (10000_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 Register Field

The vector length (VL) register specifies the length of all vector operations performed by vector instructions and the number of elements held by the V registers. The value in the VL register can be changed by software while a program is running.

Cluster Number Register Field

The cluster number (CLN) register determines 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 contents of the CLN register in all CPUs are also used to determine the condition necessary for a deadlock interrupt.

Flag Register Field

The flag (F) register contains several flags, which when set, interrupt program execution by initiating an exchange sequence. The contents of the F register are stored along with the rest of the exchange package during the exchange sequence. The monitor program can then analyze the flags for the cause of the interrupt. Before the monitor program exchanges back, it must clear the flags in the F register area of the exchange 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 another CPU issues instruction 0014*j*1.
- Deadlock (DL) flag; set when all CPUs in a cluster are holding issue on a test and set instruction.
- Programmable clock interrupt (PCI) flag; set when the programmable clock reaches a count of 0.
- MCU interrupt (MCU) flag; set when the IOS 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 interrupt-on-floating-point error (IFP) bit in the M register 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 interrupt-on-operand range error bit is set.
- Program range error (PRE) flag; set when an instruction fetch is made outside the boundaries of the IBA and ILA registers.
- Memory error (ME) flag; set when a correctable or uncorrectable memory error occurs and the corresponding interrupt-on-memory error (IME) bit in the M register is set.
- I/O interrupt (IOI) flag; set when a 6-Mbyte/s channel or a 1000-Mbyte/s channel completes a transfer.
- Error exit (EEX) flag; if not in monitor mode or if the interrupt-in-monitor mode bit is set, this flag is set by an error exit instruction.
- Normal exit (NEX) flag; if not in monitor mode, this flag is set by a normal exit instruction.

Mode Register Field

The mode (M) register contains user-selectable bits that dictate the execution of the program. It also contains 2 status bits (program state and floating-point error status) that are set by software and hardware, respectively, during an exchange sequence. The M register contains the following bits:

- Program state (PS) bit; this bit is set by the operating system to show whether a CPU, concurrently processing a program with another CPU, is the master or slave in a multitasking situation.
- Floating-point error status (FPS) bit; when set, a floating-point error occurred regardless of the state of the interrupt-on-floating-point error bit.
- Bidirectional memory (BDM) bit; when set, block reads and writes can operate concurrently.
- Interrupt-on-operand range error (IOR) bit; when set, this bit enables interrupts on operand address range errors.

- Interrupt-on-floating-point error (IFP) bit; when set, this bit enables interrupts on floating-point errors.
- Interrupt-on-uncorrectable memory error (IUM) bit; when set, this bit enables interrupts on uncorrectable memory data errors and on register parity bits.
- Interrupt-on-correctable memory error (ICM) bit; when set, this bit enables interrupts on correctable memory data errors.
- Extended addressing mode (EAM) bit; when set, this bit indicates that 32-bit (Y-mode) addressing takes place. When it is not set, this bit indicates that 24-bit (X-mode) addressing takes place.
- Selected for external interrupts (SEI) bit; this bit designates the CPU with priority for I/O interrupts.
- Interrupt monitor mode (IMM) bit; when set, this bit enables all interrupts in monitor mode except the programmable clock interrupt, MCU interrupt, interrupt-on-internal CPU, and I/O interrupt.
- Monitor mode (MM) bit; when set, this bit inhibits all interrupts except memory errors, normal exit, and error exit.

Vector Not Used Field

The state of the vector not used (VNU) bit in the exchange package indicates whether vector register instructions were issued during the execution intervals. If no vector register instructions were issued, the bit is set. If one or more of the vector register instructions were issued, the bit is not set.

Waiting for Semaphore Field

The state of the waiting for semaphore (WS) bit indicates that the CPU exchanged when a test and set instruction was holding in the current instruction parcel (CIP) register.

Concurrent Block Write

Because the memory ports A through C are either read or write ports, more that one write port may be active at the same time when the concurrent block write (CBW) bit is equal to a 1. When the CBW bit is equal to 0, the memory ports allow only one write port acive at a time.

Scalar Block Overlap

The scalar block overlap (SBO) allows scalar and block references to mix when SBO is equal to 1. When SBO is equal to a 0, scalar memory references wait until all block references are complete before isssuing.

A Registers Field

The current contents of all A registers are stored in this portion of the exchange package.

S Registers Field

The current contents of all S registers are stored in this portion of the exchange package.

Instruction Fetch

An instruction fetch operation loads program code from central memory to one of the instruction buffers. Each CPU has eight instruction buffers; each holds 128 consecutive instruction parcels for a total of 1,024 parcels. Refer to "Instruction Formats" later in this section for more information on instruction formats and parcels. Instruction parcels are held in the buffers before being delivered to the instruction issue registers (refer to the following "Instruction Issue" subsection for a definition of these registers).

The contents of the program address (P) register determines when a fetch is made (refer to the following "Instruction Issue" subsection for a definition of the P register). If the P register is pointing to an instruction parcel not currently held in one of the instruction buffers, a fetch operation occurs.

A fetch operation always occurs following an exchange sequence. The instruction buffers are filled circularly as needed. When the P register counts 128 parcels, it reaches the end of the first instruction buffer. A second fetch occurs, filling the second instruction buffer, and so on, until all buffers are filled. If a program exceeds 1,024 parcels, the ninth fetch reloads the first instruction buffer.

Instruction Issue

The P register receives instruction parcels from the instruction buffers, decodes the instructions, checks the availability of the necessary hardware, and issues the instruction.

The P register selects an instruction parcel from one of the instruction buffers. This parcel is sent to the next instruction parcel (NIP*ghijk*) register. Under normal circumstances, the P register increments sequentially as instructions are issued. However, branch instructions and exchange sequences can load the P register with any value.

Programmable Clock

Each CPU has one programmable clock. This 32-bit clock can be loaded with a count value, then decremented one count each CP. An interrupt is generated when the clock reaches a count of 0. These clocks allow the operating system to force interrupts at a particular time or frequency and enhance the use of multitasking in programs.

Status Register

The status register contains bits that reflect the operating modes of the CPU. These bits can be transferred to the high-order bit positions of a selected S register. The status register bits reflect the following CPU states:

- Clustered, CLN not set to 0
- Uncorrectable memory error occurred
- Correctable memory error occurred
- Program state status
- Floating-point error occurred
- Floating-point interrupt enabled
- Operand range interrupt enabled
- Bidirectional memory enabled
- Processor number count (bits 2^0 through 2^2)
- Cluster number count (bits 2^0 through 2^3)

Special Features of the CPU

The mainframe has several special features that enhance the parallel processing capabilities inherent in all Cray Research mainframes. Parallel processing can mean different things in different environments; the following subsections describe parallel processing within a single CPU of a CRAY Y-MP EL series mainframe.

Parallel processing features within a single CPU include pipelining and segmentation, functional unit independence, and vector processing (vectorization). The first two features are inherent hardware features of

the mainframe. Vector processing is a feature that can be manipulated by a programmer to provide optimum throughput. These features are explained in following subsections.

Pipelining and Segmentation

Pipelining is defined as an operation or instruction beginning before a previous operation or instruction has completed. Pipelining is accomplished through the use of fully segmented hardware. Segmentation refers to the process whereby an operation is divided into a discrete number of sequential steps, or segments. Fully segmented hardware is designed to implement this segmentation by performing one segment of the operation during a single CP. At the beginning of the next CP, the partial results obtained are sent to the next segment of the hardware for processing the next step of the operation. During this CP, the previous hardware segment can process the next operation. If segmented hardware is not used, the whole operation or instruction has to finish before another starts.

In the mainframe, segmented hardware includes all the hardware associated with exchange sequences, memory references, instruction fetch sequences, instruction issue sequences, and functional unit operations. The pipelining and segmentation features are critical to the execution of vector instructions.

Figure 2-11 shows how a set of elements is pipelined through a segmented vector functional unit. In the first CP, element 1 of register V1 and element 1 of register V2 enters the first segment of the functional unit. During the next CP, the partial result is moved to the second segment of the functional unit, and element 2 of both vector registers enters the first segment. This process continues each CP until all elements are completely processed.

In this example, the functional unit is divided into five segments; the functional unit can process up to five different pairs of elements simultaneously. After the first result leaves the functional unit and enters register V3, subsequent results are available at the rate of one result per CP.

Functional Unit Independence

The specialized functional units in the mainframe handle the arithmetic, logical, and shift operations. Most units are fully independent of the others and any number of functional units can process instructions concurrently. This functional unit independence allows different operations, such as multiplications, additions, and so on, to proceed in parallel.

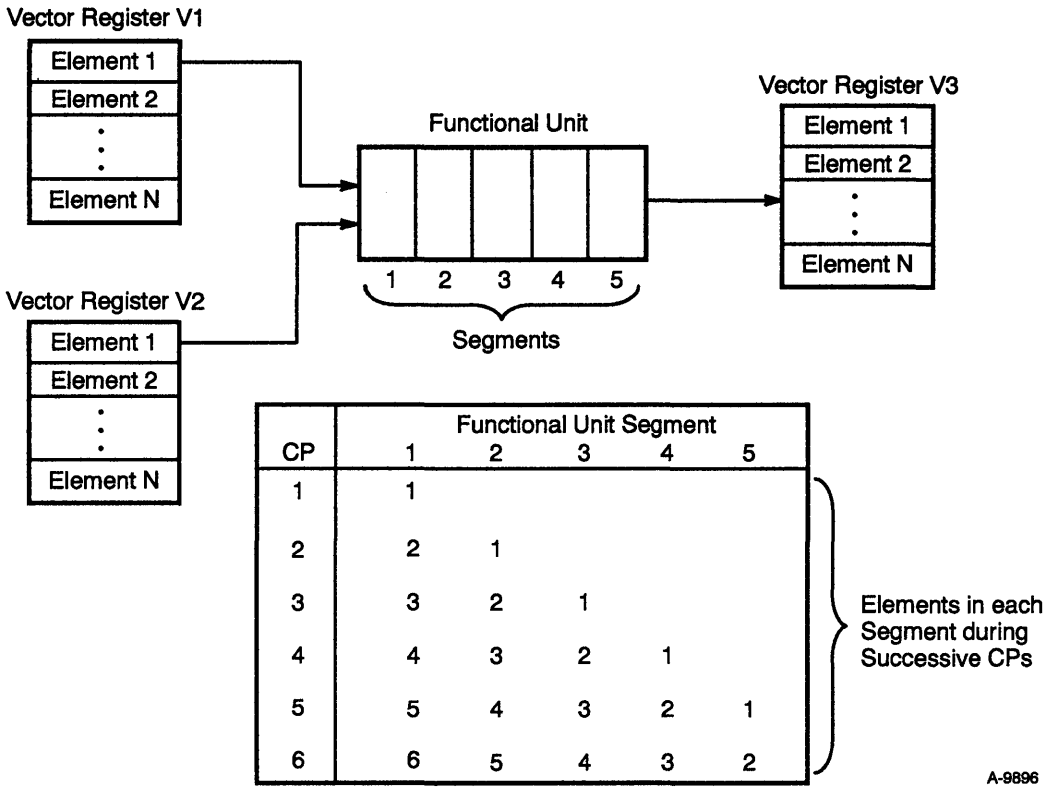

Figure 2-11. Segmentation and Pipelining Example

For example, the equation, A = (B + C) x D x E, could be run as follows. If operands B, C, D, and E are already loaded into the S registers, three instructions are generated for the equation: one that adds B and C; one that multiplies D and E, and one that multiplies the results of these two operations. The multiplication of D and E is issued first, followed by the addition of B and C. The addition and the multiplication proceed concurrently, and because the add takes less time to run than the multiply, the add and multiply complete at the same time. The add operation is essentially hidden in that it occurs during the same time interval as the multiply operation. The results of these two operations are then multiplied to obtain the final result.

Vector Processing

One of the most powerful features of the mainframe is its vector processing capability. This feature increases processing speed and efficiency by allowing an operation to be performed sequentially on a set (or vector) of operands, through the execution of a single instruction. The following subsections describe vector processing, the advantages of using vector processing, and the types of vector instructions.

Definition of Vector Processing

Each CPU of the mainframe contains V registers and a number of vector and floating-point functional units that perform vector operations. Refer to "Vector Registers," "Vector Functional Units," and "Floating-point Functional Units" in this section for more information on these registers and functional units.

A vector is an ordered set of elements; each is represented as a 64-bit word. A vector is distinguished from a scalar, which is a single 64-bit word. Examples of structures in Fortran that can be represented as vectors are one-dimensional arrays and rows, columns, and diagonals of multidimensional arrays. Vector processing occurs when arithmetic or logical operations are applied to vectors; it is distinguished from scalar processing in that it operates on many elements rather than on one.

In vector processing, successive elements are provided each CP, and as each operation is completed, the result is delivered to a successive element of the result register. The vector operation continues until the number of operations performed by the instructions equals the count specified by the vector length (VL) register.

Advantages of Vector Processing

In general, vector processing is faster and more efficient than scalar processing. Vector processing reduces overhead associated with maintenance of the loop control variable (for example, incrementing and checking the count). In many cases, loops processed as vectors reduce to a simple sequence of instructions without branching backwards. Vector instructions are usually the register-to-register type so that memory access conflicts are reduced. Finally, functional unit segmentation is exploited through vector processing because results from the units can then be obtained at the rate of one result per CP.

Vectorization typically speeds up a code segment by approximately a factor of 10. If a segment of code that previously accounted for 50% of a program's run time is vectorized, the overall run time is 55% of the original run time (50% for the unvectorized portion plus 0.1 x 50% for the vectorized portion). Vectorizing 90% of a program causes run time to drop to 19% of the original execution time.

Vector Chaining

The mainframe allows a vector register reserved for results to become the operand register of a succeeding instruction. This process, called chaining, allows a continuous stream of operands to flow through the

vector registers and functional units. Even when a vector load operation pauses due to memory conflicts, chained operations may proceed as soon as data is available.

This chaining mechanism allows chaining to begin at any point in the result vector data stream. The amount of concurrency in a chained operation depends on the relation between the issue time of the chaining instruction and the result data stream. For full chaining to occur, the chaining instruction must have issued and be ready to use element 0 of the result at the same time element 0 arrives at the V register. Partial chaining occurs if the chaining instruction issues after the arrival of element 0.

Figure 2-12 shows how the results of four instructions are chained together. The sequence of instructions uses both the pipelining and segmentation features described in the previous subsection, along with the chaining mechanism to efficiently process the elements. The sequence of instructions performs the following operations:

1. Read a vector of integers from memory to vector register V0.

2. Add the contents of V0 to the contents of V1 and send the results to V2.

3. Shift the results obtained in Step 2 and send the results to V3.

4. Form the logical product of the shifted sum obtained in Step 3 with V4, and send the results to V5.

Elements are loaded into vector register V0. As soon as the first element arrives from memory into V0, it is added to the first element of vector register V1. Subsequent elements are pipelined through the segmented functional unit, so that a continuous stream of results is sent to the destination register, which is vector register V2. As soon as the first element arrives at V2, it becomes the operand for the shift operation. The results are sent to V3, which immediately becomes the source of one of the operands necessary for the logical operation between V3 and V4. The results of the logical operation are then sent to vector register V5.

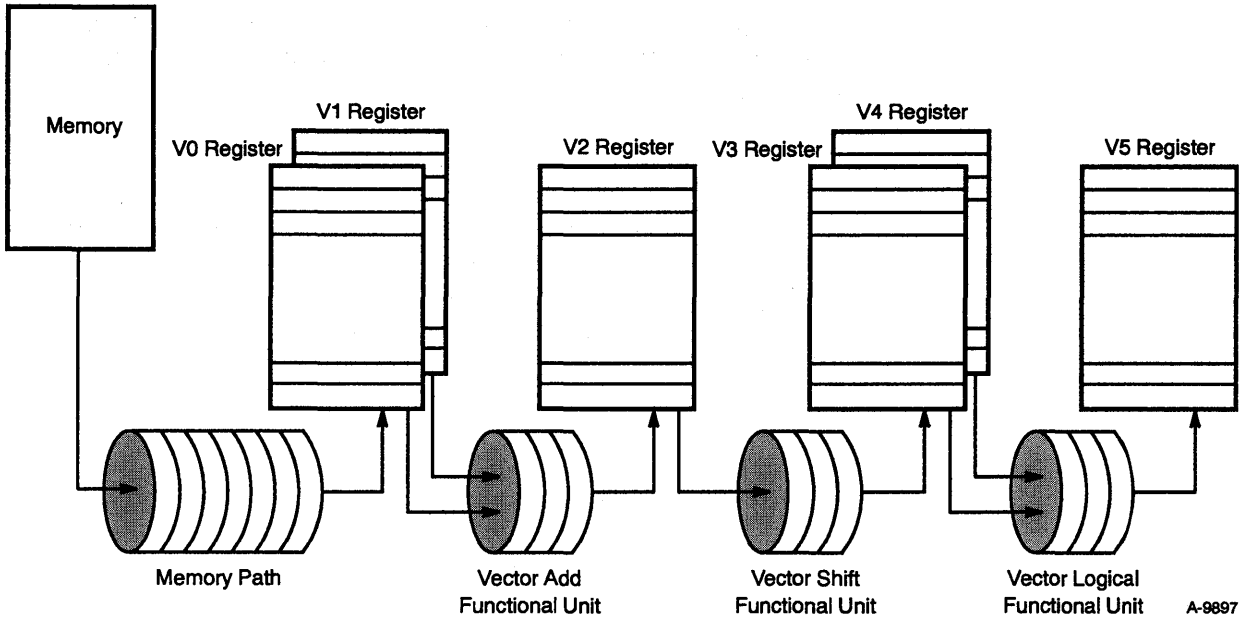

Figure 2-12. Vector Chaining Example

Types of Vector Instructions

The instructions that operate on vectors can be divided into four types.

- Vector-vector operand instructions that obtain operand(s) from one or two V registers and enter results into another V register
- Vector-scalar operand instructions that obtain one operand (a constant) from an S register and one operand from a V register and enter results in another V register
- Vector memory instructions that load (read) or store (write) elements to memory
- Vector instructions that set the vector mask (VM) register or set/read the vector length (VL) register

The vector-vector operand instructions obtain operands from one or two V registers and enter results into another V register. Refer to "Functional Instruction Summary" in this section for more information on the specific instructions.

Figure 2-13 shows how the data flows for these instructions. Successive operands or operand pairs are transmitted from V*j* and/or V*k* to the segmented functional unit each CP. Corresponding results emerge from the functional unit n CPs later; n is a constant for a given functional unit

and is called the functional unit time. Results are then entered into result register V*i*. Contents of the VL register determine the number of operand pairs processed by the functional unit.

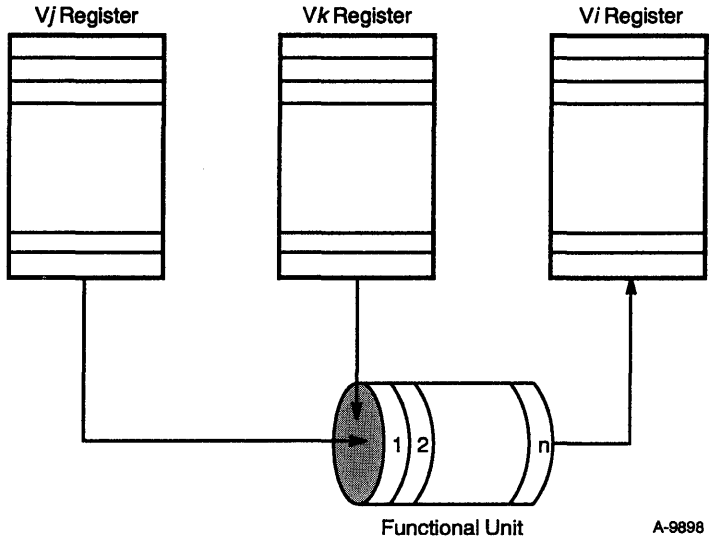

Figure 2-13. Vector-vector Operand Instructions

The vector-scalar operand instructions obtain one operand from an S register and one from a V register (refer to Figure 2-14). A copy of the S register is transmitted to the functional unit with each V register operand. Refer to "Functional Instruction Summary" in this section for more information on the specific instructions.

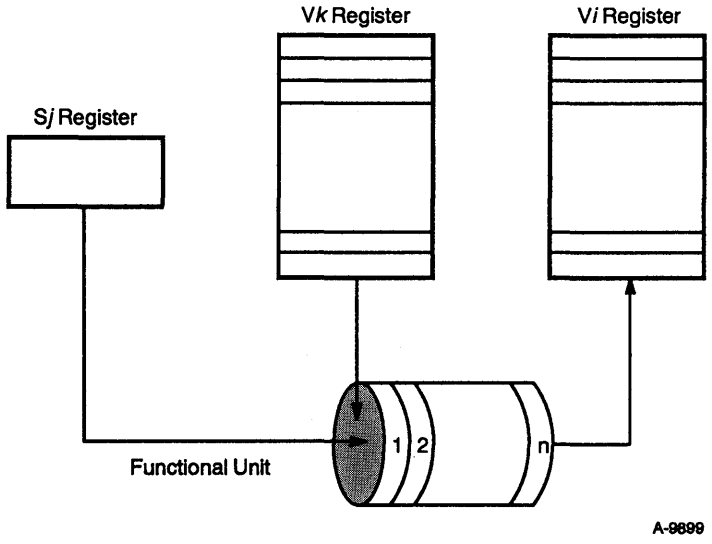

Figure 2-14. Vector-scalar Operand Instructions

Vector memory instructions transmit data between memory and the V registers (refer to Figure 2-15). A path between memory and the V registers is considered a functional unit for timing considerations. Refer to "Functional Instruction Summary" in this section for more information on the instructions.

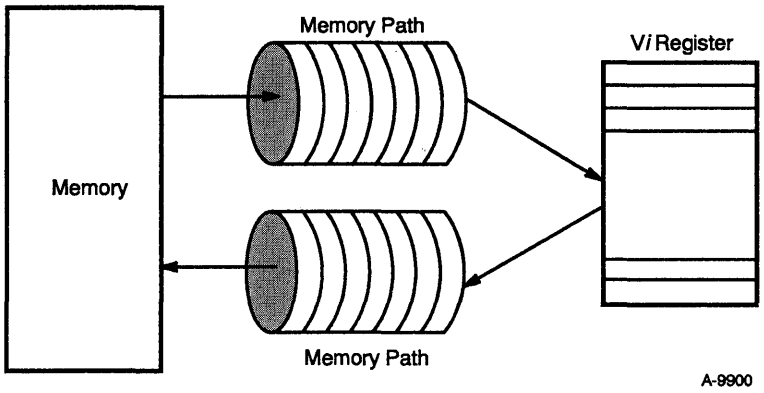

Figure 2-15. Vector Memory Instructions

Memory access and vector processing are closely related. A special gather/scatter mechanism is available on the mainframe to allow access to memory for vector operations in cases where vectorization would otherwise not be possible.

Most vector memory instructions access memory addresses with a fixed increment value. The gather and scatter instructions use two vector registers to gather or scatter elements randomly throughout memory. The first vector register contains the data and the second vector register is used as an index to gather or scatter the data from/to random memory locations.

Figure 2-16 shows an example of the gather instruction. The gather instruction transfers the contents of nonsequential memory locations to elements of a V register. In the example, the VL register is set to 4, resulting in a transfer of 4 elements. The gather instruction adds the contents of A0 to the contents of each element of the index V register (V0) to form a memory address. The contents of that address are then stored in the result V register (V1). Since A0 = 100 and V0 element 0 = 4, the contents of address 104 are stored in V1 element 0. Similarly, A0 + V0 element 1 = 102, and the contents of memory location 102 are stored in V1 element 1. This process continues until the number of elements transferred equals the VL count.

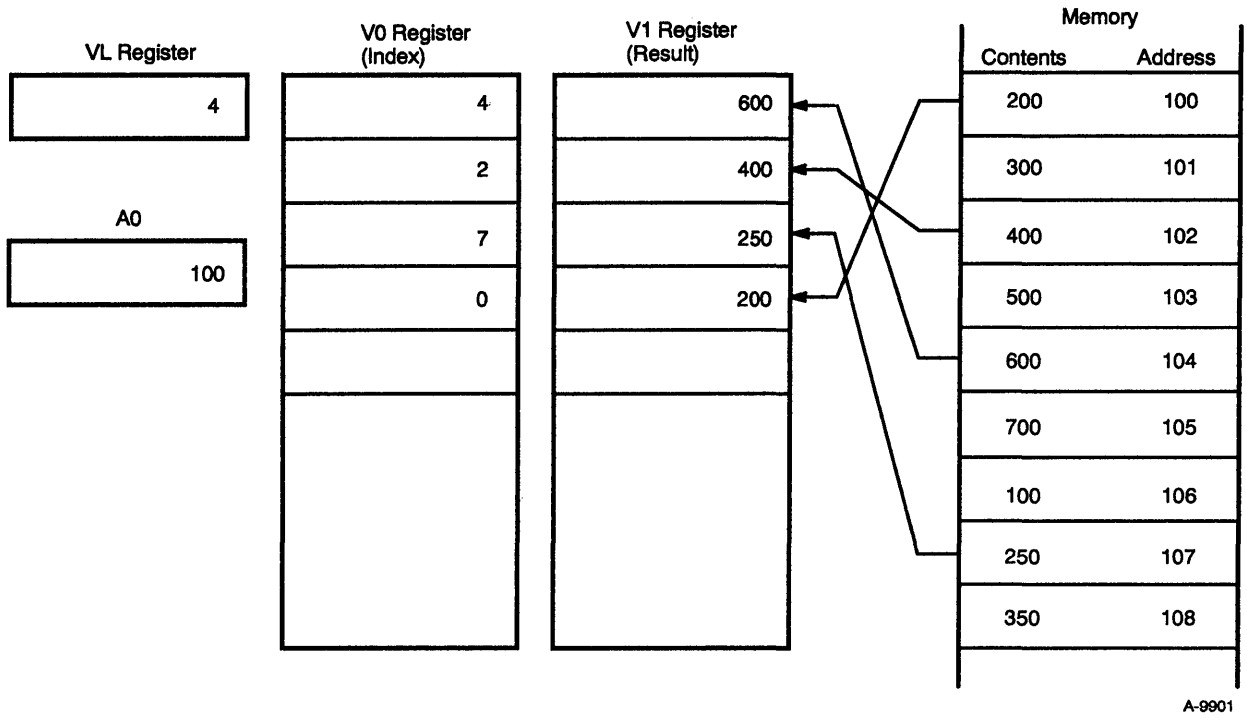

Figure 2-16. Gather Instruction Example

Figure 2-17 shows an example of the scatter instruction. The scatter instruction transfers elements of a V register to nonsequential memory locations. In the example, the VL register is set to 4, resulting in a transfer of 4 elements. The scatter instruction adds the contents of A0 to the contents of each element of the index V register (V0) to form a memory address. An element of V1 is stored at the resulting memory address. Because A0 = 100 and V0 element 0 = 4, the contents of V1 element 0 are stored in address 104. Similarly, A0 + V0 element 1 = 102, and the contents of V1 element 1 are stored in memory location 102. This process continues until the number of elements transferred equals the VL count.

The fourth group of instructions sets the VM register or reads and sets the VL register. (Refer to "Functional Instruction Summary" in this section for more information on the specific instructions.) The VM register has 64 bits, each corresponding to a word element in a V register. The high-order bit of the VM register corresponds to element 0 of the V register, while the low-order bit corresponds to element 63. The mask is used with vector merge and test instructions to perform operations on individual elements.

The VM instructions can be issued only to a single integrated circuit. The VM instructions include four compressed index instructions. These instructions test for zero, nonzero, positive, and negative elements, and generate a vector mask at the same time. Figure 2-18 shows an example of a compressed index instruction.

In the example, the elements in V0 are individually tested for a nonzero status; if the element is 0, a 0 is entered in the VM register. If the element is nonzero, a 1 is entered in the VM register and the index of the nonzero elements is loaded into register Vl. This process continues until the number of elements specified in the VL register has been tested.

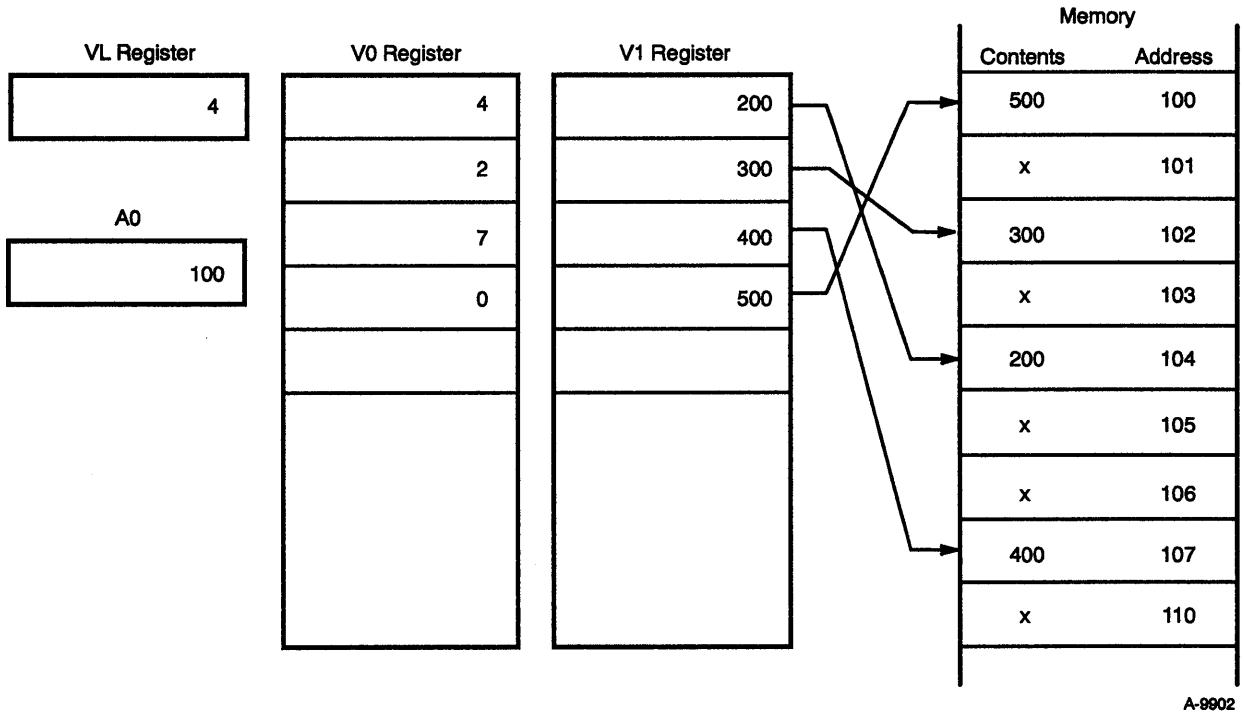

Figure 2-17. Scatter Instruction Example

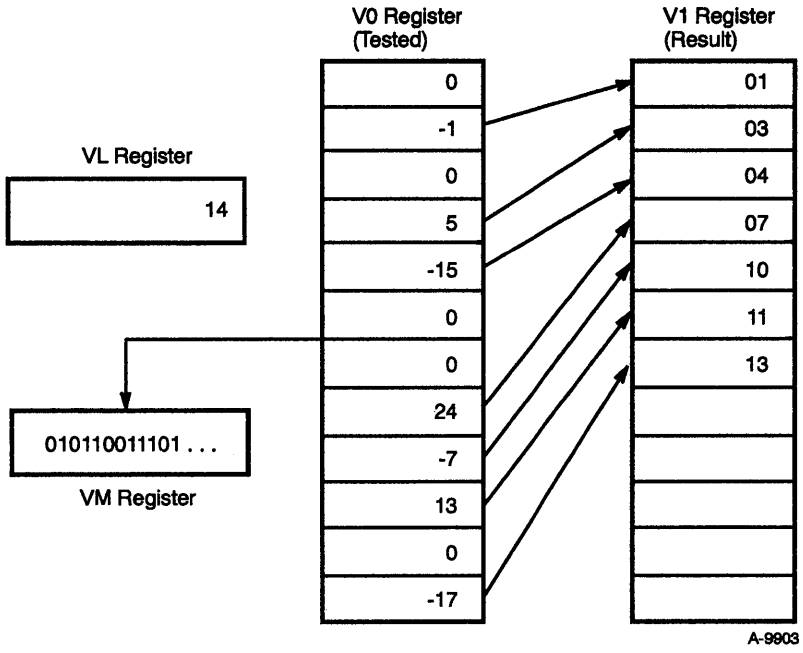

Figure 2-18. Compressed Index Example

CPU INSTRUCTIONS

The following subsections explain the instruction formats, instruction differences between X-mode and Y-mode, and special register values used by the CPUs. A CPU instruction summary is also included.

Instruction Formats

Instructions can be 1 parcel (16-bit), 2 parcels (32-bit), or 3 parcels (48-bit, Y-mode only) long. Instructions are packed 4 parcels per word and parcels are numbered 0 through 3 from left to right. Any parcel position can be addressed in branch instructions. A 2-parcel or 3-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 2-19 shows the general format for instructions.

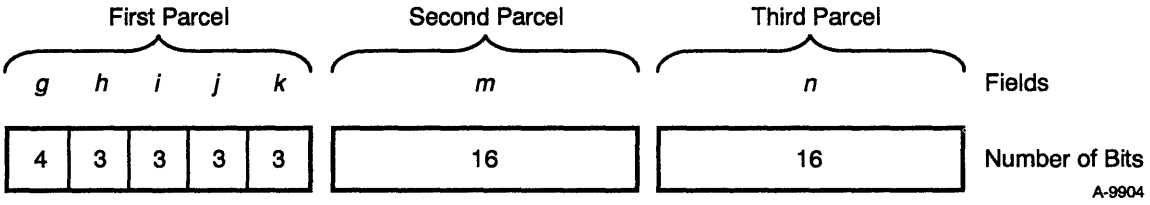

Figure 2-19. General Format for Instructions

Five variations of this general format use the fields differently. The formats of the following variations are described in the following subsections.

- 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
- 3-parcel instruction format with combined *m* and *n* 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 (refer to Figure 2-20). 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. The arithmetic, logical, double shift, and floating-point constant instructions ignore one or more of the *i*, *j*, and *k* fields.

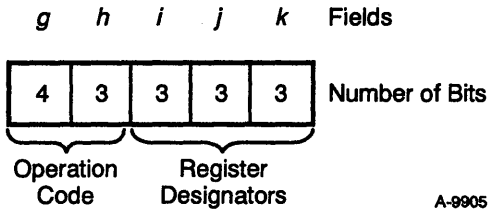

Figure 2-20. 1-parcel Instruction Format with Discrete *j* and *k* fields

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

Some 1-parcel instructions use the *j* and *k* fields as a combined 6-bit field (refer to Figure 2-21). 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*ijk* 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

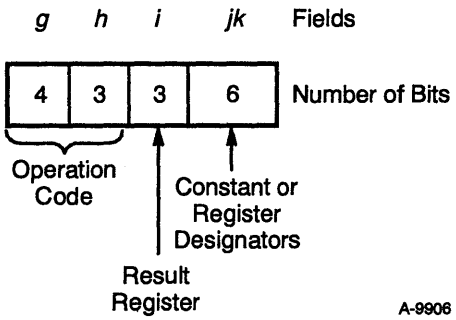

Figure 2-21. 1-parcel Instruction Format with Combined *j* and *k* Fields

2-parcel Instruction Format with Combined *j*, *k*, and *m* Fields

The format 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 using this format transfers the 22-bit *jkm* constant to an A or S register.

The instruction format used for scalar memory transfers also requires a 22-bit *jkm* field for address displacement. This format 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.

Figure 2-22 shows the two general applications for the 2-parcel instruction format with combined *j*, *k*, and *m* fields.

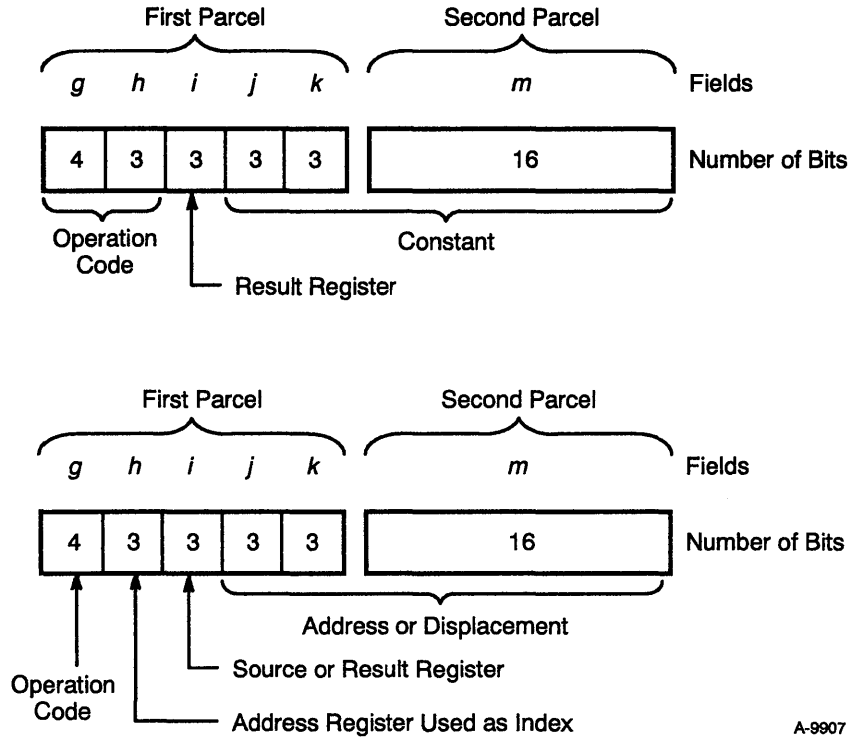

Figure 2-22. 2-parcel Instruction Format with Combined *j, k,* and *m* Fields

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

This 2-parcel format uses the combined *i, j, k,* and *m* fields to contain a 24-bit address that allows branching to an instruction parcel (refer to Figure 2-23). A 7-bit operation code (*gh*) is followed by an *ijkm* field. The high-order bit of the *i* field is equal to 0.

The 2-parcel format for a 24-bit immediate constant (refer to Figure 2-24) uses the combined *i, j, k,* and *m* fields to hold the constant. This format 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 this *i* field is set.

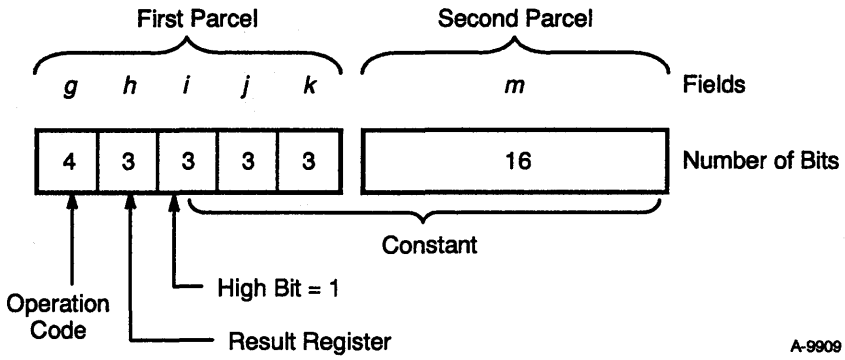

Figure 2-23. 2-parcel Instruction Format with Combined *i, j, k,* and *m* Fields

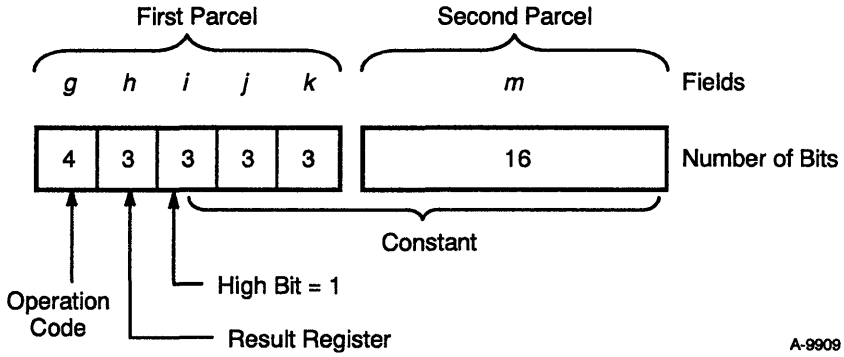

Figure 2-24. 2-parcel Instruction Format for a 24-bit Immediate Constant with Combined *i, j, k,* and *m* Fields

3-parcel Instruction Format with Combined *m* and *n* Fields

The format for a 32-bit immediate constant uses the combined *m* and *n* 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 *j* and *k* fields are a constant 0. The instruction using this format transfers the 32-bit *mn* constant to an A or S register.

NOTE: The *m* field of the 3-parcel instruction contains bits 2^0 through 2^{15} of the expression, while the *n* field contains bits 2^{16} through 2^{31} of the expression. When the instruction is assembled, the *mn* field is "reversed" and actually appears as the *nm* field when used as an expression.

The format used for scalar memory transfers also requires a 32-bit *mn* field for address or displacement. This format 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.

Figure 2-25 shows the two general applications for the 3-parcel instruction format with combined *m* and *n* fields.

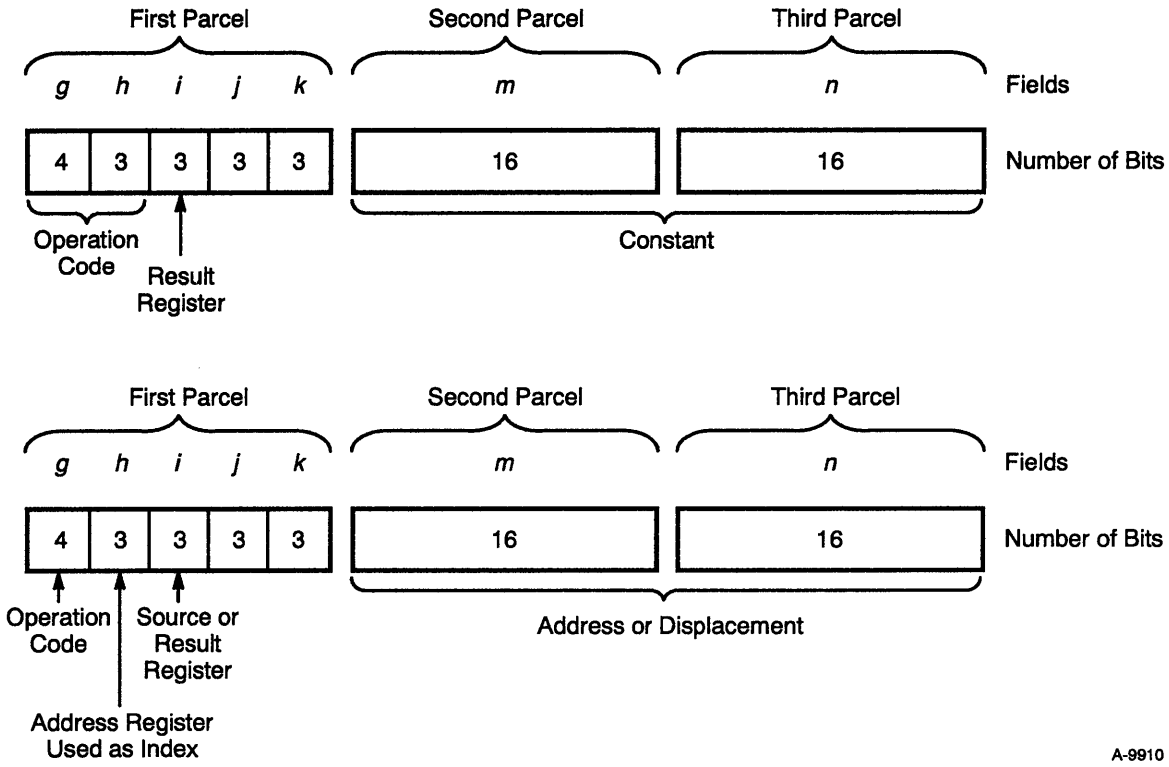

Figure 2-25. 3-parcel Instruction Format with Combined *m* and *n* Fields

Instruction Differences between X-mode and Y-mode

CRAY Y-MP EL CPUs run either of two instruction modes: the X-mode or the Y-mode. In the Y-mode, the instruction set is expanded to include 3-parcel instructions (refer to Table 2-1), and the A registers, B registers, and the address functional units operate at a full 32-bit width. These 3-parcel instructions run only if the system is operating in Y-mode; use of these instructions while in X-mode produces errors. The program range remains 4 Mwords in both the X-mode and Y-mode.

Table 2-1. 3-parcel Instructions

CAL Syntax	Octal Code
A*i exp*	020*i*00*mn*
A*i exp*	020*i*00*mn*
S*i exp*	040*i*00*mn*
S*i exp*	041*i*00*mn*
A*i exp*, A*h*	10*hi*00*mn*
exp, A*h* A*i*	11*hi*00*mn*
S*i exp*, A*h*	12*hi*00*mn*
exp, A*h* S*i*	13*hi*00*mn*

The X-mode can be selected by resetting the extended addressing mode bit in the exchange package. In this mode, the system runs only the X-mode (1- and 2-parcel) instruction set. The upper 8 bits of the 32-bit registers and 32-bit results are discarded, leaving the operation exactly the same as the 24-bit CRAY X-MP computer system results.

All instructions operate the same as in the CRAY X-MP computer systems, except those listed in Table 2-2.

Table 2-2. CRAY Y-MP and CRAY X-MP Instruction Differences

Instruction	X-mode	Y-mode	Comments
01*hijkm*	A*h exp*	N/A	Not allowed in Y-mode
0014*j*1	SIPI *exp*	SIPI A*j*	Change is necessary because more CPUs are available
0014*j*3	CLN *exp*	CLN A*j*	Change is necessary because there are more clusters in the mainframe
166*ijk*	V*i* S*j**|V*k*	V*i* S*j**V*k*	Runs differently in the X-mode than in the Y-mode

Special Register Values

If the S0 and A0 registers are referenced in the *h*, *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 2-3 shows the special register values.

Table 2-3. 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
Si, *i* = 0	(S0)
Sj, *j* = 0	0
Sk, *k* = 0	2^{63}

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 run only when the CPU is operating in monitor mode. If a monitor mode instruction issues while the CPU is not in monitor mode, it is treated as a no-operation instruction.

Special CAL Syntax Forms

The CAL instruction set has special forms of symbolic instructions. Because of this expansion, certain machine instructions can be generated from two or more different CAL instructions. Any of the operations performed by special instructions can be performed by instructions in the basic set.

For example, both of the following CAL instructions generate instruction 002000, which enters a 1 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 where (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.

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

Instructions having a special syntax form are identified in the instruction summary in this section.

CPU Instruction Summary

This subsection introduces and summarizes all CPU instructions. The instructions are summarized two ways: by the functional unit that executes the instruction and by the function the instruction performs.

The following instruction summaries use the acronyms and abbreviations that were defined in previous sections. A glossary is provided at the end of this manual; acronyms and abbreviations are defined there.

In some instructions, register designators are prefixed by the following letters that have special meaning to the assembler. The letters and their meanings are listed as follows.

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

The following list describes some of the notations used in the instruction set.

Character	Operation
+	Arithmetic sum of specified registers
-	Arithmetic difference of specified registers
*	Arithmetic product of specified registers

Character	Operation
/	Reciprocal of approximation
#	Use one's 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

An expression (*exp*) occupies the *jk*, *ijk*, *jkm*, *ijkm*, or *ijkmn* field. 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.

Functional Units Instruction Summary

Instructions other than simple transmit or control operations are performed by functional units. The following list summarizes the instructions performed by each of the functional units.

Functional Unit	Instructions
Address Add (Integer)	030, 031
Address Multiply (Integer)	032
Scalar Add (Integer)	060, 061
Scalar Logical	042 through 051
Scalar Shift	052 through 055, 056, 057

Functional Unit	Instructions
Scalar Pop/Parity/ Leading Zero	026 027
Vector Add (Integer)	154 through 157
Vector Logical	140 through 147, 175
Second Vector Logical	140 through 145
Vector Shift	150, 151, 153, 152
Vector Pop/Parity	174*ij*1, 174*ij*2
Floating-point Add	062, 063, 170 through 173
Floating-point Multiply	064 through 067, 160 through 167
Floating-point Reciprocal	070, 174*ij*0
Memory (Scalar)	100 through 130
Memory (Vector)	176, 177

Functional Instruction Summary

This subsection summarizes the instructions by the functions they perform. Included is a brief, general description of the function of each group of instructions; then the machine instruction, the CAL syntax, and a description are listed.

The following footnotes are used throughout the instruction summary:

Footnote	Description
1	Privileged to monitor mode
2	Special syntax mode
3	Not supported by CAL Version 2
4	Generated depending on the value of *exp*
5	X-mode only
6	Y-mode only

Register Entry Instructions

The register entry instructions transmit values, such as constants, expression values, or masks, directly into registers.

Transfers into A Registers

The following instructions transmit values into the A registers.

Machine Instruction	CAL Syntax	Description
01*hijkm*[5]	A*h* *exp*	Transmit *exp* to A*h* ($i^2 = 1$)
020*ijkm*[4, 5] or	A*i* *exp*	Transmit *exp* into A*i* (020) or
021*ijkm*[4, 5]		Transmit one's complement of *exp* into A*i* (021)
020*i*00*mn*[4, 6]	A*i* *exp*	Transmit *exp* into A*i* (020) or
021*i*00*mn*[4, 6]		Transmit one's complement of *exp* into A*i* (021)
022*ijk*[4]	A*i* *exp*	Transmit *exp* = *jk* to A*i*
031*i*00[2]	A*i* -1	Transmit -1 into A*i*

Transfers into S Registers

The following instructions transmit values into the S registers.

Machine Instruction	CAL Syntax	Description
040*ijkm*[4,5] or 041*ijkm*[4,5]	S*i* *exp*	Transmit *exp* into S*i* (040) or Transmit one's complement of *exp* into S*i* (041)
040*i*00*mn*[4,5] 041*i*00*mn*[4,5]	S*i* *exp*	Transmit *exp* into S*i* (040) or Transmit one's complement of *exp* into S*i* (041)
042*i*00[2]	S*i* -1	Enter -1 into S*i*
042*ijk*	S*i* <*exp*	Form ones mask in S*i* *exp* bits from right; *jk* field gets 64-*exp*
042*ijk*[2]	S*i* #>*exp*	Form zeros mask in S*i* *exp* bits from left; *jk* field gets exp
042*i*77[2]	S*i* 1	Enter 1 into S*i*
043*i*00[2]	S*i* 0	Clear S*i*
043*ijk*	S*i* >*exp*	Form ones mask in S*i* *exp* bits from left; *jk* field gets *exp*
043*ijk*[2]	S*i* #<*exp*	Form zeros mask in S*i* *exp* bits from right; *jk* field gets 64-*exp*
047*i*00[2]	S*i* #SB	Enter one's complement of sign bit into S*i*
051*i*00[2]	S*i* SB	Enter sign bit into S*i*
071*i*30	S*i* 0.6	Transmit (0.75×2^{48}) as normalized floating-point constant into S*i*
071*i*40	S*i* 0.4	Transmit 0.4 as normalized floating-point constant into S*i*

Machine Instruction	CAL Syntax	Description
071*i*50	S*i* 1.	Transmit 1.0 as normalized floating-point constant into S*i*
071*i*60	S*i* 2.	Transmit 2.0 as normalized floating-point constant into S*i*
071*i*70	S*i* 4.	Transmit 4.0 as normalized floating-point constant into S*i*

Transfers Into V Registers

The following instructions transmit values into the V registers.

Machine Instruction	CAL Syntax	Description
077*i*0*k*^{2}	V*i* ,A*k* 0	Clear element (A*k*) of register V*i*
145*iii*^{2}	V*i* 0	Clear V*i* elements

Transfers Into Semaphore Register

The following instructions transmit values into the semaphore registers.

Machine Instruction	CAL Syntax	Description
0034*jk*	SM*jk* 1,TS	Test and set semaphore *jk*, $0 < jk < 31_{10}$
0036*jk*	SM*jk* 0	Clear semaphore *jk*, $0 < jk < 31_{10}$
0037*jk*	SM*jk* 1	Set semaphore *jk*, $0 < jk < 31_{10}$

Interregister Transfer Instructions

The interregister transfer instructions transmit 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

The following instructions transfer the contents of other registers into the A registers.

Machine Instruction	CAL Syntax	Description
023*ij*0	A*i* S*j*	Transmit (S*j*) to A*i*
023*i*01	A*i* VL	Transmit (VL) to A*i*
024*ijk*	A*i* B*jk*	Transmit (B*jk*) to A*i*
026*ij*7	A*i* SB*j*	Transmit (SB*j*) to A*i*
030*i*0*k*[2]	A*i* A*k*	Transmit (A*k*) to A*i*
031*i*0*k*[2]	A*i* -A*k*	Transmit negative of (A*k*) to A*i*
033*i*00	A*i* CI	Channel number of highest priority interrupt request to A*i* ($j = 0$)
033*ij*0	A*i* CA, A*j*	Current address of channel (A*j*) to A*i* ($j \neq 0, k = 0$)
033*ij*1	A*i* CE, A*j*	Error flag of channel (A*j*) to A*i* ($j \neq 0, k = 1$)

Transfers to S Registers

The following instructions transmit the contents of other registers into the S registers.

Machine Instruction	CAL Syntax	Description
025*ijk*	B*jk* A*i*	Transmit (A*i*) to B*jk*
027*ij*7	SB*j* A*i*	Transmit (A*i*) to SB*j*
047*i*0*k*[2]	S*i* #S*k*	Transmit one's complement of (S*k*) to S*i*
051*i*0*k*[2]	S*i* S*k*	Transmit (S*k*) to S*i*
061*i*0*k*[2]	S*i* -S*k*	Transmit negative of (S*k*) to S*i*

Machine Instruction	CAL Syntax	Description
071*i*0*k*	S*i* A*k*	Transmit (A*k*) to S*i* with no sign extension
071*i*1*k*	S*i* +A*k*	Transmit (A*k*) to S*i* with sign extension
071*i*2*k*	S*i* +FA*k*	Transmit (A*k*) to S*i* as unnormalized floating-point number
072*i*00	S*i* RT	Transmit (RTC) to S*i*
072*i*02	S*i* SM	Transmit semaphore to S*i*
072*ij*3	S*i* ST*j*	Transmit (ST*j*) register to S*i*
073*i*00	S*i* VM	Transmit (VM) to S*i*
073*i*01	S*i* SR*j*	Transmit (SR*j*) to S*i* (*j* = 0)
073*ij*3	ST*j* S*i*	Transmit (S*i*) to ST*j*
074*ijk*	S*i* T*jk*	Transmit (T*jk*) to S*i*
075*ijk*	T*jk* S*i*	Transmit (S*i*) to T*jk*
076*ijk*	S*i* V*j*, A*k*	Transmit (V*j* element (A*k*)) to S*i*

Transfers to V Registers

The following instructions transmit the contents of other registers into the V registers.

Machine Instruction	CAL Syntax	Description
077*ijk*	V*i* ,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*)
142*i*0*k*[2]	V*i* V*k*	Transmit (V*k* elements) to V*i* elements
156*i*0*k*[2]	V*i* -V*k*	Transmit two's complement of (V*k* elements) to V*i* elements

Transfer to Vector Mask Register

The following instructions transmit the contents of other registers into the VM register.

Machine Instruction	CAL Syntax	Description
0030*j*0	VM S*j*	Transmit (S*j*) to VM register
003000^2	VM 0	Clear VM register

Transfer to Vector Length Register

The following instructions transmit the contents of other registers into the vector length register.

Machine Instruction	CAL Syntax	Description
00200*k*	VL A*k*	Transmit (A*k*) to VL register
002000^2	VL 1	Transmit 1 to VL register

Transfer to Semaphore Register

The following instruction transmits the contents of other registers into the semaphore registers.

Machine Instruction	CAL Syntax	Description
073*i*02	SM S*i*	Read semaphores from S*i*

Memory Transfer Instructions

The memory transfer instructions enable or disable bidirectional memory transfers, transfer data between registers and memory, and ensure completion of memory references.

Bidirectional Memory Transfers

The following instructions enable or disable bidirectional memory transfers.

Machine Instruction	CAL Syntax	Description
002500	DBM	Disable bidirectional memory transfers
002404	DCBW	Concurrent block write
002506	DSBO	Scalar and block overlap
002600	EBM	Enable bidirectional memory transfers
002604	ECBW	Enable concurrent block write
002606	ESBO	Enable scalar and block overlap

Memory References

The following instruction ensures completion of instructions for bidirectional memory transfers.

Machine Instruction	CAL Syntax	Description
002700	CMR	Complete memory references

Writes

The following instructions write values into memory.

Machine Instruction	CAL Syntax	Description
035*ijk*	,A0 B*jk*,A*i*	Write (A*i*) words from B registers starting at register *jk* to memory starting at address (A0) + DBA
035*ijk*[2]	0,A0 B*jk*,A*i*	Write (A*i*) words from B registers starting at register *jk* to memory starting at address (A0) + DBA

Machine Instruction	CAL Syntax	Description
037*ijk*	,A0 T*jk*,A*i*	Write (A*i*) words from T registers starting at register *jk* to memory starting at address (A0) + DBA
037*ijk*[2]	0,A0 T*jk*,A*i*	Write (A*i*) words from T registers starting at register *jk* to memory starting at address (A0)
11*hijkm*[5]	*exp*,A*h* A*i*	Write (A*i*) to ((A*h*) + *exp*) + DBA
11*hi*00*mn*[6]	*exp*,A*h* A*i*	Write (A*i*) to ((A*h*) + *exp*) + DBA
11*hi*000[2, 5]	,A*h* A*i*	Write (A*i*) to (A*h*) + DBA
11*hi*0000[2, 6]	,A*h* A*i*	Write (A*i*) to (A*h*) + DBA
110*ijkm*[2, 5]	*exp*,0 A*i*	Write (A*i*) to *exp* + DBA
110*i*00*mn*[2, 6]	*exp*,0 A*i*	Write (A*i*) to *exp* + DBA
110*ijkm*[2, 5]	*exp*, A*i*	Write (A*i*) to *exp* + DBA
110*i*00*mn*[2, 6]	*exp*, A*i*	Write (A*i*) to *exp* + DBA
13*hijkm*[5]	*exp*,A*h* S*i*	Write (S*i*) to ((A*h*) + *exp*) + DBA
13*hi*00*mn*[6]	*exp*,A*h* S*i*	Write (S*i*) to ((A*h*) + *exp*) + DBA
130*ijkm*[2, 5]	*exp*,0 S*i*	Write (S*i*) to *exp* + DBA
130*i*00*mn*[2, 6]	*exp*,0 S*i*	Write (S*i*) to *exp* + DBA
130*ijkm*[2, 5]	*exp*, S*i*	Write (S*i*) to *exp* + DBA
130*i*00*mn*[2, 6]	*exp*, S*i*	Write (S*i*) to *exp* + DBA
13*hi*000[2, 5]	,A*h* S*i*	Write (S*i*) to (A*h*) + DBA
13*hi*0000[2, 6]	,A*h* S*i*	Write (S*i*) to (A*h*) + DBA
1770*jk*	,A0,A*k* V*j*	Write (V*j*) to memory starting at (A0) + DBA increased by (A*k*)

Machine Instruction	CAL Syntax	Description
1770*j*0	,A0,1 V*j*	Write (V*j*) to memory in consecutive addresses starting with (A0) + DBA
1771*jk*	,A0,V*k* V*j*	Write (V*j*) to memory using memory address ((A0) + (V*k*) + DBA)

Reads

The following instructions load values from memory.

Machine Instruction	CAL Syntax	Description
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words from memory starting at address (A0) + DBA to B registers starting at address *jk*
034*ijk*[2]	B*jk*,A*i* 0,A0	Read (A*i*) words from memory starting at address (A0) + DBA to B registers starting at address *jk*
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words from memory starting at address (A0) + DBA to T registers starting at address *jk*
036*ijk*[2]	T*jk*,A*i* 0,A0	Read (A*i*) words from memory starting at address (A0) + DBA to T registers starting at address *jk*
10*hijkm*[5]	A*i* *exp*,A*h*	Read from ((A*h*) + *exp* + DBA) to A*i*
10*hi*00*mn*[6]	A*i* *exp*,A*h*	Read from ((A*h*) + *exp* + DBA) to A*i*
10*hi*000[2, 5]	A*i* ,A*h*	Read from (A*h*) DBA to A*i*
10*hi*0000[2, 6]	A*i* ,A*h*	Read from (A*h*) DBA to A*i*
100*ijkm*[2, 5]	A*i* *exp*,0	Read from (*exp*) DBA to A*i*
100*i*00*mn*[2, 6]	A*i* *exp*,0	Read from (*exp*) DBA to A*i*
100*ijkm*[2, 5]	A*i* *exp*,	Read from (*exp*) DBA to A*i*

Machine Instruction	CAL Syntax	Description
100*i*00*mn*[2, 6]	A*i* *exp*,	Read from (*exp*) DBA to A*i*
12*hijk*m[5]	S*i* *exp*,A*h*	Read from ((A*h*) + *exp* + DBA) to S*i*
12*hi*00*mn*[6]	S*i* *exp*,A*h*	Read from ((A*h*) + *exp* + DBA) to S*i*
120*ijkm*[2, 5]	S*i* *exp*,0	Read from (*exp*) DBA to S*i*
120*i*00*mn*[2, 6]	S*i* *exp*,0	Read from (*exp*) DBA to S*i*
120*ijkm*[2, 5]	S*i* *exp*	Read from (*exp*) DBA to S*i*
120*i*00*mn*[2, 6]	S*i* *exp*	Read from (*exp*) DBA to S*i*
12*hi*000[2, 5]	S*i* ,A*h*	Read from (A*h*) + DBA to S*i*
12*hi*0000[2, 6]	S*i* ,A*h*	Read from (A*h*) + DBA to S*i*
176*i*0*k*	V*i* ,A0,A*k*	Read from memory starting at (A0) + DBA increased by (A*k*) and load into V*i*
176*i*00[2]	V*i* ,A0,1	Read from consecutive memory addresses starting with (A0) + DBA into V*i*
176*i*1*k*	V*i* ,A0,V*k*	Read from memory using memory address (A0) + (V*k*) + DBA into V*i*

Integer Arithmetic Instructions

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, 32-bit, or 64-bit, is two's complement and is represented as such in the registers. The address add and address multiply functional units perform 24-bit (X-mode) and

32-bit (Y-mode) arithmetic. The scalar add functional unit and the vector add functional unit perform 64-bit arithmetic. No overflow is detected by functional units when performing integer arithmetic.

Multiplication of two fractional operands is accomplished using a 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 functional units. Refer to "Floating-point Arithmetic" in this section for more information on these algorithms.

24-bit or 32-bit Integer Arithmetic

The following instructions perform 24-bit (X-mode) or 32-bit (Y-mode) integer arithmetic.

Machine Instruction	CAL Syntax	Description
030*ijk*	A*i* Aj + A*k*	Integer sum of (A*j*) and (A*k*) to A*i*
030*ij*0[2]	A*i* Aj + 1	Integer sum of (A*j*) and 1 to A*i*
031*ijk*	A*i* Aj–A*k*	Integer difference of (A*j*) and (A*k*) to A*i*
031*ij*0[2]	A*i* Aj–1	Integer difference of (A*j*) and 1 to A*i*
032*ijk*	A*i* Aj*A*k*	Integer product of (A*j*) and (A*k*) to A*i*

64-bit Integer Arithmetic

The following instructions perform 64-bit integer arithmetic.

Machine Instruction	CAL Syntax	Description
060*ijk*	S*i* S*j* + S*k*	Integer sum of (S*j*) and (S*k*) to S*i*
061*ijk*	S*i* S*j*–S*k*	Integer difference of (S*j*) and (S*k*) to S*i*
154*ijk*	V*i* S*j* + V*k*	Integer sums of (S*j*) and (V*k* elements) to V*i* elements

Machine Instruction	CAL Syntax	Description
155*ijk*	V*i* V*j* + V*k*	Integer sums of (V*j* elements) and (V*k* elements) to V*i* elements
156*ijk*	V*i* S*j*–V*k*	Integer differences of (S*j*) and (V*k* elements) to V*i* elements
157*ijk*	V*i* V*j*–V*k*	Integer differences of (V*j* elements) and (V*k* elements) to V*i* elements

Floating-point Arithmetic Instructions

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. Because the coefficient is signed magnitude, it is not complemented for negative values. Refer to "Floating-point Arithmetic" in this section for more information on floating-point numbers and arithmetic.

Floating-point Range Errors

The following instructions enable or disable floating-point range errors to be flagged.

Machine Instruction	CAL Syntax	Description
002100	EFI	Enable interrupt on Floating-point error
002200	DFI	Disable interrupt on Floating-point error

Floating-point Addition and Subtraction

The following instructions perform floating-point addition or subtraction.

Machine Instruction	CAL Syntax	Description
062*ijk*	S*i* S*j* + FS*k*	Floating-point sum of (S*j*) and (S*k*) to S*i*
062*i*0*k*[2]	S*i* +FS*k*	Normalize (S*k*) to S*i*
063*ijk*	S*i* S*j*–FS*k*	Floating-point difference of (S*j*) and (S*k*) to S*i*
063*i*0*k*[2]	S*i* –FS*k*	Transmit the normalized negative of (S*k*) to S*i*
170*ijk*	V*i* S*j* + FV*k*	Floating-point sums of (S*j*) and (V*k* elements) to V*i* elements
170*i*0*k*[2]	V*i* + FV*k*	Transmit normalized (V*k* elements) to V*i* elements
171*ijk*	V*i* V*j* + FV*k*	Floating-point sums of (V*j* elements) and (V*k* elements) to V*i* elements
172*ijk*	V*i* S*j*–FV*k*	Floating-point differences of (S*j*) and (V*k* elements) to V*i* elements
172*i*0*k*[2]	V*i* –FV*k*	Transmit normalized negative of (V*k* elements) to V*i* elements
173*ijk*	V*i* V*j*–FV*k*	Floating-point differences of (V*j* elements) and (V*k* elements) to V*i* elements

Floating-point Multiplication

The following instructions perform floating-point multiplication.

Machine Instruction	CAL Syntax	Description
064*ijk*	S*i* S*j**FS*k*	Floating-point product of (S*j*) and (S*k*) to S*i*
065*ijk*	S*i* S*j**HS*k*	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*
066*ijk*	S*i* S*j**RS*k*	Rounded floating-point product of (S*j*) and (S*k*) to S*i*
160*ijk*	V*i* S*j**FV*k*	Floating-point products of (S*j*) and (V*k* elements) to V*i* elements
161*ijk*	V*i* V*j**FV*k*	Floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements
162*ijk*	V*i* S*j**HV*k*	Half-precision rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements
163*ijk*	V*i* V*j**HV*k*	Half-precision rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements
164*ijk*	V*i* S*j**RV*k*	Rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements
165*ijk*	V*i* V*j**RV*k*	Rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements

Reciprocal Iteration

The following instructions perform reciprocal iteration operations.

Machine Instruction	CAL Syntax	Description
067*ijk*	S*i* S*j**IS*k*	Reciprocal iteration: 2 – (S*j*) x (S*k*) to S*i*
166*ijk*[5]	V*i* S*j**IV*k*	Reciprocal iteration: 2 – (S*j*) x (V*k* elements) to V*i* elements
166*ijk*[6]	V*i* S*j**V*k*	32-bit integer product of (S*j*) and (V*k* elements) to V*i* elements
167*ijk*	V*i* V*j**IV*k*	Reciprocal iteration: 2 – (V*j* elements) x (V*k* elements) to V*i* elements

Reciprocal Approximation

The following instructions perform floating-point reciprocal approximation operations.

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

Logical Operation Instructions

The scalar and vector logical functional units perform bit-by-bit manipulation of 64-bit quantities. Logical operations include logical products, logical sums, logical differences, logical equivalence, vector mask, and merges. Logical operations are defined below.

- 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

The following instructions produce logical products.

Machine Instruction	CAL Syntax	Description
044*ijk*	S*i* S*j*&S*k*	Logical product of (S*j*) and (S*k*) to S*i*
044*ij*0[2]	S*i* S*j*&SB	Sign bit of (S*j*) to S*i*
044*ij*0[2]	S*i* SB&S*j*	Sign bit of (S*j*) to S*i* (*j* ≠ 0)
045*ijk*	S*i* #S*k*&S*j*	Logical product of (S*j*) and complement of (S*k*) to S*i*
045*ij*0[2]	S*i* #SB&S*j*	(S*j*) with sign bit cleared to S*i*
140*ijk*	V*i* S*j*&V*k*	Logical products of (S*j*) and (V*k* elements) to V*i* elements
141*ijk*	V*i* V*j*&V*k*	Logical products of (V*j* elements) and (V*k* elements) to V*i* elements

Logical Sums

The following instructions produce logical sums.

Machine Instruction	CAL Syntax	Description
051*ijk*	S*i* S*j*!S*k*	Logical sum of (S*j*) and (S*k*) to S*i*
051*ij*0[2]	S*i* S*j*!SB	Logical sum of (S*j*) and sign bit to S*i*
051*ij*0[2]	S*i* SB!S*j*	Logical sum of (S*j*) and sign bit to S*i* (*j* ≠ 0)
142*ijk*	V*i* S*j*!V*k*	Logical sums of (S*j*) and (V*k* elements) to V*i* elements
143*ijk*	V*i* V*j*!V*k*	Logical sums of (V*j* elements) and (V*k* elements) to V*i* elements

Logical Differences

The following instructions produce logical differences.

Machine Instruction	CAL Syntax	Description
046*ijk*	S*i* S*j*\S*k*	Logical difference of (S*j*) and (S*k*) to S*i*
046*ij*0[2]	S*i* S*j*\SB	Toggle sign bit of (S*j*), then enter into S*i*
046*ij*0[2]	S*i* SB\S*j*	Toggle sign bit of (S*j*), then enter into (S*i*) (*j* ≠ 0)
144*ijk*	V*i* S*j*\V*k*	Logical differences of (S*j*) and (V*k* elements) to V*i* elements
145*ijk*	V*i* V*j*\V*k*	Logical differences of (V*j* elements) and (V*k* elements) to V*i* elements

Logical Equivalence

The following instructions produce logical equivalence.

Machine Instruction	CAL Syntax	Description
047*ijk*	S*i* #S*j*\S*k*	Logical equivalence of (S*j*) and (S*k*) to S*i*
047*ij*0[2]	S*i* #S*j*\SB	Logical equivalence of (S*j*) and sign bit to S*i*
047*ij*0[2]	S*i* #SB\S*j*	Logical equivalence of (S*j*) and sign bit to S*i* (*j* ≠ 0)

Vector Mask

The following instructions perform a mask operation that sets a vector operand for certain elements depending on the mask.

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

Merge

The following instructions merge two operands together depending on a ones mask in a third operand.

Machine Instruction	CAL Syntax	Description
050*ijk*	S*i* Sj!S*i*&S*k*	Logical product of (S*i*) and (S*k*) complement ORed with logical product of (S*j*) and (S*k*) to S*i*
050*ij*0[2]	S*i* Sj!S*i*&SB	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*
146*ijk*	V*i* Sj!V*k*&VM	Transmit (S*j*) if VM bit = 1; (V*k*) if VM bit = 0 to V*i*

Machine Instruction	CAL Syntax	Description
146*i*0*k*^{2}	V*i* #VM&V*k*	Vector merge of (V*k*) and 0 to V*i*
147*ijk*	V*i* V*j*!V*k*&VM	Transmit (V*j*) if VM bit = 1; (V*k*) if VM bit = 0 to V*i*

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.

The following instructions shift values by a specified amount.

Machine Instruction	CAL Syntax	Description
052*ijk*	S0 S*i*<*exp*	Shift (S*i*) left *exp* places to S0; *exp* = *jk*
053*ijk*	S0 S*i*>*exp*	Shift (S*i*) right *exp* places to S0; *exp* = 64–*jk*
054*ijk*	S*i* S*i*<*exp*	Shift (S*i*) left *exp* places to S*i*; *ex* = *jk*
055*ijk*	S*i* S*i*>*exp*	Shift (S*i*) right *exp* places to S*i*; *exp* = 64–*exp*
056*ijk*	S*i* S*i*,S*j*<A*k*	Shift (S*i*) and (S*j*) left by (A*k*) places to S*i*
056*ij*0^2	S*i* S*i*,S*j*<1	Shift (S*i*) and (S*j*) left one place to S*i*
056*i*0*k*^{2}	S*i* S*i*<A*k*	Shift (S*i*) left (A*k*) places to S*i*
057*ijk*	S*i* S*j*,S*i*>A*k*	Shift (S*j*) and (S*i*) right by (A*k*) places to (S*i*)
057*ij*0^2	S*i* S*j*,S*i*>1	Shift (S*j*) and (S*i*) right one place to (S*i*)

Machine Instruction	CAL Syntax	Description
057*i*0k^2	S*i* S*i*>A*k*	Shift (S*i*) right (A*k*) places to S*i*
150*ijk*	V*i* V*j*<A*k*	Shift (V*j* elements) left by (A*k*) places to V*i* elements
150*ij*0^2	V*i* V*j*<1	Shift (V*j* elements) left one place to V*i* elements
151*ijk*	V*i* V*j*>A*k*	Shift (V*j* elements) right by (A*k*) places to V*i* elements
151*ij*0^2	V*i* V*j*>1	Shift (V*j* elements) right one place to V*i* elements
152*ijk*	V*i* V*j*,V*j*<A*k*	Double shift of (V*j* elements) left (A*k*) places to V*i* elements
152*ij*0^2	V*i* V*j*,V*j*<1	Double shift of (V*j* elements) left one place to V*i* elements
153*ijk*	V*i* V*j*,V*j*>A*k*	Double shift of (V*j* elements) right (A*k*) places to V*i* elements
153*ij*0^2	V*i* V*j*,V*j*>1	Double shift of (V*j* elements) right one place to V*i* elements

Bit Count Instructions

Bit count instructions count the number of set bits or the number of leading 0 bits in an S or V register.

Scalar Population Count

The following instruction performs the scalar population count.

Machine Instruction	CAL Syntax	Description
026*ij*0	A*i* PS*j*	Population count of (S*j*) to A*i*

Vector Population Count

The following instruction performs the vector population count.

Machine Instruction	CAL Syntax	Description
174*ij*1	V*i* PV*j*	Population count of (V*j* elements) to (V*i* elements)

Population Parity Count

The following instructions perform population parity count.

Machine Instruction	CAL Syntax	Description
026*ij*1	A*i* QS*j*	Population count parity of (S*j*) to A*i*
174*ij*2	V*i* QV*j*	Population count parity of (V*j* elements) to (V*i* elements)

Scalar Leading Zero Count

The following instruction performs leading zero count.

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

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 runs. If an expression has a word-address attribute, the assembler issues an error message.

Unconditional Branch Instructions

The following instructions perform unconditional branch operations.

Machine Instruction	CAL Syntax	Description
0050*jk*	J B*jk*	Jump to (B*jk*)
006*ijk*m	J *exp*	Jump to *exp*

Conditional Branch Instructions

The following instructions perform conditional branch operations.

Machine Instruction	CAL Syntax	Description
010*ijkm*	JAZ *exp*	Jump to *exp* if (A0) = 0 (*i*2 = 0)
011*ijkm*	JAN *exp*	Jump to *exp* if (A0) ≠ 0 (*i*2 = 0)
012*ijkm*	JAP *exp*	Jump to *exp* if (A0) positive; includes (A0) = 0 (*i*2 = 0)
013*ijkm*	JAM *exp*	Jump to *exp* if (A0) negative (*i*2 = 0)
014*ijkm*	JSZ *exp*	Jump to *exp* if (S0) = 0 (*i*2 = 0)
015*ijkm*	JSN *exp*	Jump to *exp* if (S0) ≠ 0 (*i*2 = 0)
016*ijkm*	JSP *exp*	Jump to *exp* if (S0) positive; includes (S0) = 0 (*i*2 = 0)
017*ijkm*	JSM *exp*	Jump to *exp* if (S0) negative (*i*2 = 0)

Return Jump

The following instruction performs a return jump operation.

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

Normal Exit

The following instruction performs a normal exit operation.

Machine Instruction	CAL Syntax	Description
004000	EX	Normal exit

Error Exit

The following instruction performs an error exit operation.

Machine Instruction	CAL Syntax	Description
000000	ERR	Error exit

Monitor Mode Instructions

Monitor mode instructions are executed only when the CPU is in monitor mode. An attempt to execute one of these instructions when the CPU is not in monitor mode is treated as a pass instruction. Monitor mode instructions perform specialized functions useful to the operating system.

Channel Control

The following instructions perform channel control operations.

Machine Instruction	CAL Syntax	Description
0010*jk*[1]	CA,A*j* A*k*	Set the CA register for the channel indicated by (A*j*) to (A*k*) and activate the channel
001000	PASS	Pass
0011*jk*[1]	CL,A*j* A*k*	Set the CL register for the channel indicated by (A*j*) to (A*k*) address
0012*j*0[1]	CI,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*); clear device master-clear (output channel)

Machine Instruction	CAL Syntax	Description
0012*j*1[1]	MC,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*); set device master-clear (output channel); clear device ready-held (input channel)
0013*j*0[1]	XA A*j*	Enter XA register with (A*j*)

Set Real-time Clock

The following instruction performs a real-time clock operation.

Machine Instruction	CAL Syntax	Description
0014*j*0[1]	RT S*j*	Load RTC register with (S*j*)

Programmable Clock Interrupt Instructions

The following instructions perform programmable clock operations.

Machine Instruction	CAL Syntax	Description
0014*j*4[1]	PCI S*j*	Load II register with (S*j*)
001405[1]	CCI	Clear programmable clock interrupt request
001406[1]	ECI	Enable programmable clock interrupt request
001407[1]	DCI	Disable programmable clock interrupt request

Interprocessor Interrupt Instructions

The following instructions perform interprocessor interrupt operations.

Machine Instruction	CAL Syntax	Description
0014*j*1[1]	SIPI A*j*	Set interprocessor interrupt request to CPU (A*j*)
001401[1, 2]	SIPI	Set interprocessor interrupt request to CPU 0
001402[1]	CIPI	Clear interprocessor interrupt

Cluster Number Instructions

The following instruction sets the cluster number.

Machine Instruction	CAL Syntax	Description
0014*j*3[1]	CLN A*j*	Load CLN register with (A*j*) where $0 \leq exp \leq 7$

Operand Range Error Interrupt Instructions

The following instructions enable or disable operand range error interrupts.

Machine Instruction	CAL Syntax	Description
002300	ERI	Enable interrupt on address range error
002400	DRI	Disable interrupt on address range error

SECTION 3
INPUT/OUTPUT SUBSYSTEM

3 INPUT/OUTPUT SUBSYSTEM

The CRAY Y-MP EL system uses a VMEbus architecture input/output subsystem (IOS) for data transfers from central memory to the peripherals. The VMEbus is a high-performance industry standard backplane that can connect any vendor input/output (I/O) controller to the IOS. The backplane distributes power and delivers a common set of data, address, and control signals to the boards plugged into it.

The IOS offloads the mainframe from peripheral control tasks, allowing parallel I/O operations and overlapping I/O and computation, thus freeing the CPU for high-speed numerical computation.

The CRAY Y-MP EL IOS supports the following devices and operations via the VME controller boards. These devices are discussed in the "Peripheral Devices" subsection of this section.

- System console operation
- MWS operation
- Disk subsystems
- Tape subsystem
- Network subsystem
- Printer or plotter

IOS Configurations

A CRAY Y-MP EL system can contain as many as 16 IOSs. Each CPU can handle up to four IOSs. The first IOS resides in the mainframe cabinet; the mainframe cabinet may contain up to four IOSs. As many as 12 more IOSs can be located in the peripheral cabinets.

Each VME IOS supports 8, 4, or 2 I/O controllers plus 2 standard boards. The input/output buffer board (IOBB) and the CPU board are included with the system's first IOS. The number of controllers supported depends on the VME backplane selected when the system configuration is ordered.

Each IOS can consist of a 10-slot, 6-slot, or 4-slot backplane.

Channel Communications and Networking

The IOS controls all data transfers between components of the CRAY Y-MP EL system. It provides fast data transmission among central memory and peripheral devices and networks. Data travels from a peripheral device across a data channel to the IOBB board in the IOS, then to the CRAY Y-MP EL memory via a 40-Mbyte/s channel. There are four of these data channels for each CPU. Refer to Figure 3-1.

The CRAY Y-MP EL system can be connected to a multiple-system network using the Transmission Control Protocol/Internet Protocol (TCP/IP), a widely used protocol for interconnecting UNIX systems. This network connection is supported using Ethernet, HYPERchannel, and fiber-optic distributed data interface (FDDI) connections.

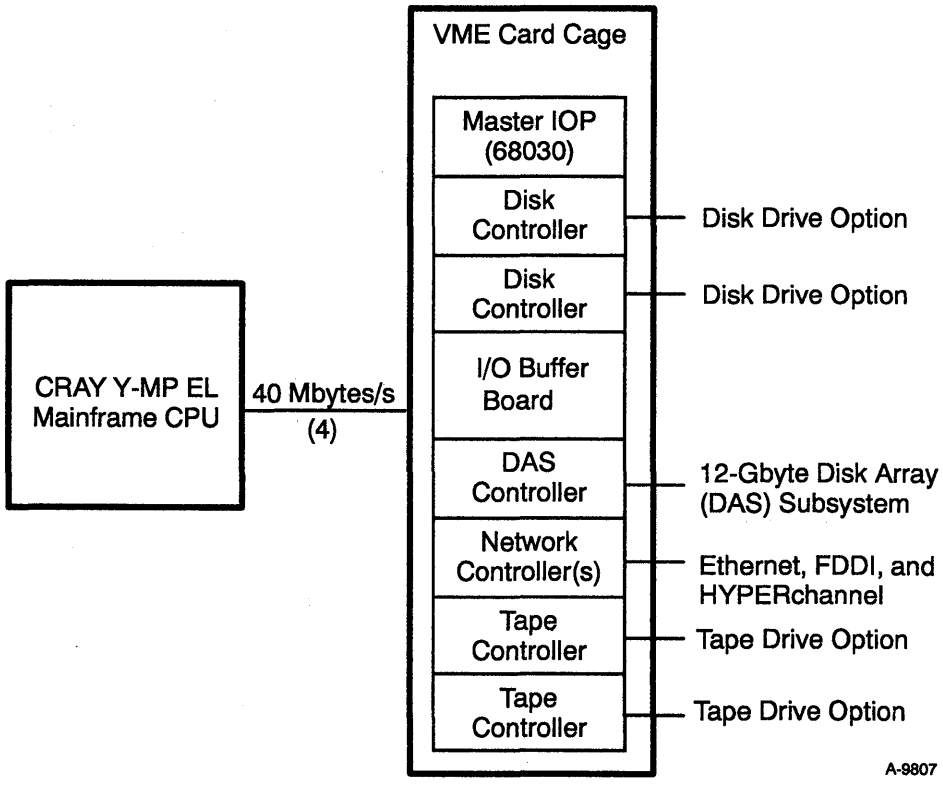

Figure 3-1. IOS Block Diagram

Master I/O Processor

The IOS master input/output processor (MIOP) is the CPU board in the IOS. The MIOP carries out I/O functions for the I/O controllers. It also processes external interrupts and CPU I/O requests and executes peripheral driver routines.

The MIOP is a self-contained system that consists of a 68030-based processor board, local memory, and control store. It controls a wide variety of peripheral devices and networks, including disk drives and drive arrays, 9-track and 3480-compatible tape drives, and networks. System diagnostics can be accessed either locally or remotely through the MIOP.

The system console and MWS are connected to the CRAY Y-MP EL system via separate cables connected to a MIOP port.

The first MIOP, which is located in the mainframe IOS, contains the following devices:

- Winchester disk drive
- 1/4-in. cartridge tape drive
- Communications ports
- Optional cartridge tape drive (1 or 2)

The MIOP performs the following operations:

- Controls the advanced diagnostic system
- Controls the central operating system start-up
- Maintains a log of detected and corrected system errors

Peripheral Devices

Many peripheral devices can be included in the CRAY Y-MP EL system. The following subsections describe these optional peripheral devices, controllers, and interfaces. Customers may upgrade their systems by adding peripheral cabinets that can include additional IOSs and peripheral devices.

DD-3 Disk Drive

The DD-3 disk drive is a compact 5 1/4-in. format high-performance enhanced small disk interface (ESDI) disk drive with a capacity of 1.3 Gbytes formatted. This disk drive operates at sustained data-transfer rates of approximately 1.9 Mbytes/s with a peak of 2 Mbytes/s.

DC-3 Disk Controller

The DC-3 disk controller interfaces the DD-3 disk drive to the IOS. The DD-3 disk drive connects to the DC-3 disk controller board in the IOS. The DC-3 disk controller is an intelligent controller that supports from one to four DD-3 disk drives. Intelligent disk management techniques include overlapping seek operations on multiple drives connected to the disk controller.

The DC-3 controller transfers data from one drive at a time.

DD-4 Disk Drive

The DD-4 disk drive is a high-performance, two-head parallel intelligent peripheral interface (IPI) -2 drive. Its drive capacity is 2.7 Gbytes formatted. The DD-4 disk drive operates at sustained data-transfer rates of approximately 6 to 7 Mbytes/s with a peak transfer rate of 7.5 Mbytes/s.

DC-4 Disk Controller

The DC-4 disk controller interfaces the DD-4 disk drive to the IOS. The DC-4 is an intelligent controller that supports from one to four DD-4 disk drives. The DC-4 controller transfers data from two drives simultaneously.

DAS-2 Disk Array Subsystem

The DAS-2 disk array subsystem uses the 5 1/4-in. DD-3 disk drives for data storage. This subsystem consists of one to four banks of ten 1.3-Gbyte drives, eight for data storage plus one for parity and error recovery and one spare drive. The spare drive serves as a standby should one of the data storage drives become inoperable. If a disk becomes inoperable, the parity drive allows the DAS controller to dynamically regenerate the data on that disk to the spare drive. The spare drive automatically replaces the failed drive, and operation of the DAS continues uninterrupted.

The DAS-2 disk array subsystem can transfer data at the following rates:

- 0.23 Mbytes/sec for data blocks that contain 4,096 bytes
- 2 Mbytes/sec for data blocks that contain 32,768 bytes
- 13 Mbytes/sec for data blocks that contain 1 Mbyte or more

DAS-2 Disk Array Subsystem Controller

Stored data from the DAS-2 subsystem is transferred by the DAS-2 controller to and from all drives. The DAS-2 controller can control eight DAS-2 disk subsystems.

DEB-2 Disk Array Bank

The DEB-2 disk array bank subsystem offers all the features of the DAS-2 subsystem. A DEB-2 tray consists of ten 1.3-Gbyte ESDI drives; eight for data storage plus one for parity and error recovery and one spare drive. This tray uses one peripheral tray slot in the cabinet.

The DEB-2 subsystem requires a disk array subsystem multiplexer (DAS-M) that allows up to four DEB-2 subsystems to be added to a DAS-2 controller.

RD-1 Removable Disk Subsystem

The RD-1 removable disk subsystem includes two 1.3-Gbyte removable drives with a 2.75-Mbyte/s peak transfer rate. These RD-1 ESDI disk drives require a DC-3 disk controller; this controller can control two RD-1 subsystems.

The RD-1C removable disk drive is identical in operation to the RD-1 disk drives, but the additional drives are contained in a removable canister.

RD-2 Removable Maintenance Drive

The RD-2 removable maintenance drive is the same small computer system interface (SCSI) drive that is used in the MIOP. These Winchester drives allow the CRAY Y-MP EL system to be configured entirely with removable media.

TD-2 Tape Drive Subsystem

The TD-2 9-track tape drive is a high-performance digital vacuum-buffered drive for 1/2-in. open reel tape. The TD-2 tape drive transfers data at a rate of 2 Mbytes/s. A CRAY Y-MP EL cabinet can contain one TD-2 tape drive.

TC-2 Tape Controller

The TC-2 tape controller interface card interfaces the TD-2 tape drive to the IOS.

TD-3 Tape Drive Subsystem

The TD-3 3480-compatible 18-track cartridge drive subsystem is a 1/2-in. cartridge tape drive. The TD-3 cartridge drive transfers data at a rate of 3 Mbytes/s. A CRAY Y-MP EL cabinet can contain up to two TD-3 cartridge drive subsystems. Each TD-3 cartridge drive is controlled by the SI-1 interface card.

SI-1 Small Computer System Interface

The SI-1 small computer system interface board interfaces the TD-3 cartridge drive to the IOS. This SI-1 interface board supports a maximum of seven SCSI devices in a daisy-chained arrangement.

EX-2 8mm Cartridge Tape Subsystem

The EX-2 cartridge tape subsystem consists of a helical-scan cartridge tape drive recording technology that uses read and write head pairs. The EX-2 subsystem provides up to 5 Gbytes of data storage on a single cartridge and transfers data at a rate of 500 Kbytes/s.

The EX-2 subsystem is controlled directly from the MIOP and does not use a VME slot.

Network Connections

A wide variety of network connections are available with the CRAY Y-MP EL system to provide access to other network computer systems and workstations. The CRAY Y-MP EL system supports Ethernet, fiber-optic distributed data interface (FDDI), and HYPERchannel connections. These interface controllers are discussed in the following subsections.

Ethernet Interface Controller

An Ethernet cable connects a CRAY Y-MP EL system to multiple computer systems on the Ethernet network. An Ethernet connection uses TCP/IP.

The Ethernet controller uses one VME slot.

Fiber-optic Distributed Data Interface Controller

The FDDI interface enables the CRAY Y-MP EL system to communicate with other front-end systems on the FDDI network. The FDDI connection uses TCP/IP.

The FDDI uses one VME slot.

HYPERchannel Interface Controller

The HYPERchannel interface enables the CRAY Y-MP EL system to communicate with other front-end systems on the network via an NSC box. The HYPERchannel connection also uses TCP/IP.

The HYPERchannel controller uses one VME slot.

Printer and Plotter Controller

Printers and plotters can be connected to the CRAY Y-MP EL system externally by using the bulkhead located on the back of the system.

GLOSSARY

GLOSSARY

A

Application Software designed to perform a particular job or set of related jobs.

A register Address register. A registers are primarily used as address registers for memory references and as index registers.

B

Backplane The printed circuit board connector panel used in the VMEbus.

Bank The smallest addressable division of central memory.

B register Intermediate address register. B registers are used as intermediate storage for the A registers.

BDM Bidirectional memory mode (bit). The modes field in the exchange package contains the BDM mode bit. When the BDM mode bit is set, block read and write operations can operate concurrently.

C

CA Current address (register). The CA register contains the initial address for a channel transfer. The contents of the CA register are incremented until the transfer is complete.

CAL Cray Assembly Language. A symbolic language that generates machine instructions on a one-for-one basis and allows programs to call subroutines from the library through the use of pseudoinstructions.

CCI Clear clock interrupt (instruction). The CCI instruction clears a programmable clock interrupt request.

Central memory Memory residing in the mainframe.

Central processing unit (CPU) A module used in the mainframe that controls the flow of system data, performs mathematical and logical functions on system data, and executes program instructions.

C (continued)

Chaining The process of sequencing logical operations so the results of one operation may be used by another operation without needing a memory reference in between.

Check bits Used to determine whether one or more bits in a word have an incorrect value.

CIP Current instruction parcel.

CIPI Clear interprocessor interrupt (instruction). The CIPI instruction clears an interprocessor interrupt.

CL Channel limit (register.) The contents of the CL register are one greater than the last address for a channel transfer. When the contents of the CA register equal the contents of the CL register, the transfer is complete.

CLN Cluster number (register). The CLN register in the exchange package determines which set of the 17_{10} available clusters of SB, ST, and SM registers the CPU can access.

Clusters A set of shared registers accessible by all CPUs. There are 17_{10} valid clusters of shared registers in a CPU.

CMOS Complementary metal oxide semiconductor. An integrated circuit option.

CMR Complete memory reference (instruction). The CMR instruction assures completion of all memory references within a particular CPU issuing the instruction.

CP Clock period. The CP is the interval in which the system clock completes one oscillation.

D

DAS Disk Array Subsystem. A disk drive subsystem that includes a bank of ten disk drives and an I/O controller.

DBA Data base address (register). The DBA register, part of the exchange package, holds the base address of the user's data range.

DBM Disable bidirectional memory transfers (instruction). The DBM instruction disables the bidirectional memory mode.

DC-3 Disk Controller-3. An I/O controller board that supports from one to four DD-3 disk drives.

D (continued)

DC-4 Disk Controller-4. An I/O controller board that supports from one to four DD-4 disk drives.

DCI Disable clock interrupts (instruction). The DCI instruction disables programmable clock interrupts until an enable clock interrupt instruction is executed.

DD-3 Disk Drive-3. A disk subsystem that utilizes the enhanced small disk interface (ESDI) drives.

DD-4 Disk Drive-4. A disk subsystem that utilizes the intelligent peripheral interface (IPI) drives.

DEB-2 Disk Array Bank-2. A disk subsystem that is identical to the DAS with the exception that all of the ESDI drives are contained in one peripheral tray.

DBA Data base address (register). The DBA register, part of the exchange package, holds the base address of the user's data range.

Deadlock A state resulting in the inability to continue processing due to an unresolvable conflict. Deadlock occurs when all CPUs in a cluster are holding issue on a test and set instruction.

Deadstart The sequence of operations required to start an operating system running in a Cray Research computer system.

DFI Disable floating-point interrupts (instruction). The DFI instruction clears the floating-point interrupt flag in the mode register.

DL Deadlock interrupt (flag). The deadlock interrupt flag sets if the IDL interrupt mode bit is set, the program is not in monitor mode, and a deadlock condition occurs because all CPUs in a cluster are holding issue on a test and set instruction.

DLA Data limit address (register). The DLA register holds the upper limit address of the user's data range.

DRAM Dynamic random-access memory. A memory device that must be refreshed periodically in order to store data.

DRI Disable interrupt on address range error (instruction). The DRI instruction clears the operand range error mode flag in the exchange package M register.

E

ESDI Enhanced small disk interface. A particular type of disk drive that is used in various disk subsystems.

Ethernet A particular type of network hardware that forms a physical link between computers; a trademark of Xerox Corporation.

EX-2 EXABYTE-2. An 8-mm helical scan tape cartridge subsystem.

Exchange mechanism The technique used in the CRAY Y-MP EL computer system for switching instruction execution from program to program. Refer to exchange package.

Exchange package A 16-word block of data in memory reserved for exchange packages. The exchange package contains the necessary registers and flags associated with a particular program. Each program has its own exchange package.

Execution unit (EU) An arithmetic unit utilized by the CRAY Y-MP EL system.

F

F Floating-point (operation). This operation is identified by an F appearing in front of a register designator in a symbolic machine.

F register Flag register. The F register contains part of the exchange package for the currently active program. The F register contains flags identified individually within the exchange package. Setting any of these flags interrupts program execution.

FDDI Fiber-optic distributed device interface. A network protocol that uses fiberoptics to transmit data.

Fetch sequence A fetch sequence transfers a block of instructions from memory to an instruction buffer.

Floating-point operation A mathematical or logical operation on two or more real numbers.

FPE Floating-point error (flag). The interrupt flags field in the exchange package contains the FPE flag. The FPE flag sets when a floating-point range error occurs in any of the floating-point functional units and the interrupt-on-floating-point error (IFP) flag is set.

F (continued)

FPS Floating-point error status (bit). The status field in the exchange package contains the FPS status bit. The floating-point status bit sets if a floating-point error occurred during the execution interval.

Functional unit Circuitry designed to perform a particular mathematical or logical operation.

G

Gather/scatter An operation that places data at various intervals in the available memory storage and then gathers the data back into its original organization.

H

H Half-precision floating-point (operation). When an H appears in front of a register designator in a symbolic machine instruction, the calculation is a half-precision floating-point operation.

Helical scan Storage tape solution that uses video recording technology.

HYPERchannel A trademark and product of Network Systems Corporation (NSC) that interfaces a channel to other brands of computers.

I

I Reciprocal iteration (operation). When an I appears in front of a register designator in a symbolic machine instruction, the calculation is a reciprocal iteration operation.

IBA Instruction base address (register). The IBA register is in the exchange package. The IBA register holds the base address of the user's instruction range.

ICM Interrupt-on-correctable memory error mode (bit). The interrupt modes field in the exchange package contains the ICM bit. When the ICM bit is set, it enables interrupts on correctable memory data errors while reading data from memory.

ICP Interprocessor interrupt (flag). The interprocessor interrupt flag sets if the IIP interrupt mode bit is set and enabled and another CPU requests an interrupt of this CPU by issuing instruction 0014*j*1.

I (continued)

IFP Interrupt-on-floating-point error mode (bit). The interrupt modes field in the exchange package contains the IFP bit. When the IFP bit is set, it enables interrupts on floating-point errors.

ILA Instruction limit address (register). The ILA register is in the exchange package. The ILA register holds the limit address of the user's instruction field.

Input/output subsystem (IOS) The CRAY Y-MP EL system uses a VMEbus architecture IOS for its data transfers between central memory and peripheral devices.

Instruction buffer A set of registers in a CRAY Y-MP EL CPU used for temporary storage of instructions before issue. Each instruction buffer can hold 128 consecutive instruction parcels.

Instruction fetch The process of loading program code from central memory to an instruction buffer.

Instruction set A set of instructions that a particular computer can perform.

Input/output buffer board (IOBB) The I/O buffer board provides temporary storage for data transfers between the mainframe and peripheral devices.

IOI I/O interrupt (flag). The F register in the exchange package contains the IOI flag. The IOI flag sets when a 6-Mbyte/s channel or when the 1000-Mbyte/s channel completes a transfer.

IOR Operand range error mode (bit). The interrupt modes field in the exchange package contains the IOR bit. When the IOR bit is set, it enables interrupts on operand address range errors.

Intelligent peripheral interface - 2 (IPI-2) An interface used to transfer control and system data between the CRAY Y-MP EL CPU and the IOS.

Issue sequence The issue sequence selects the instruction indicated by the program address (P) register, decodes it, determines whether the required registers or functional units are available, and if so, enables the CPU to execute the instruction.

IUM Interrupt-on-uncorrectable memory error mode (bit). The M register in the exchange package contains the IUM bit. When the IUM bit is set, it enables interrupts on uncorrectable memory errors.

J

J The unconditional branch instruction.

JAM A conditional branch instruction. A branch occurs if the content of register A0 is negative.

JAN A conditional branch instruction. A branch occurs if the content of register A0 is nonzero.

JAP A conditional branch instruction. A branch occurs if the content of register A0 is positive.

JAZ A conditional branch instruction. A branch occurs if the content of register A0 is 0.

JSM A conditional branch instruction. A branch occurs if the content of register S0 is negative.

JSN A conditional branch instruction. A branch occurs if the content of register S0 is nonzero.

JSP A conditional branch instruction. A branch occurs if the content of register S0 is positive or 0.

JSZ A conditional branch instruction. A branch occurs if the content of register S0 is 0.

M

M register Mode register. The M register in the exchange package contains user-selectable bits that dictate the execution of the program.

Mainframe A component of the CRAY Y-MP EL computer system that contains central memory and central processing units (CPUs).

Maintenance workstation model EL (MWS-EL) A component of a CRAY Y-MP EL computer system that provides an intelligent and dedicated platform for performing offline and online tests, monitoring environmental conditions, and recording hardware errors.

MIOP Master I/O processor. The MIOP is the CPU board in the IOS.

MM Monitor mode (bit). The modes field in the exchange package contains the MM bit. When the MM mode bit is set, it inhibits all interrupts except memory errors, normal exit, and error exit. The program can execute those instructions that are privileged to monitor mode.

M (continued)

MODE Read mode (bits). The MODE bits are part of the memory error data fields in the exchange package. The MODE bits determine the type of read mode in progress when a memory data error occurred; these bits are used with the port bits to identify the operation in error.

Monitor mode A condition in which a CPU inhibits all interrupts except those caused by memory errors, normal exit, or error exit instructions.

Multiprocessing An operation in which several computer processes or jobs are computed at the same time.

N

NEX Normal exit interrupt (flag). The interrupt flags field in the exchange package contains the NEX flag. The normal exit flag sets if the FNX interrupt mode bit is set and enabled and a normal exit instruction (00400) issues. Issuing a normal exit instruction always causes an exchange, regardless of the state of FNX.

NIP Next instruction parcel (register). The NIP register holds a parcel of program code before it enters the current instruction parcel register. Instruction decoding begins in this register.

NSC A trademark of Network Systems Corporation.

O

ORE Operand range error (flag). The interrupt flags field in the exchange package contains the ORE flag. The ORE flag sets when a data reference is made outside the boundaries of the DBA and DLA registers and the interrupt-on-operand range error bit is set.

P

P Population count (operation). When a P appears in front of a register designator in a symbolic machine instruction, the calculation is a population count operation.

Parcel A 16-bit portion of a word that is addressable for instruction execution but not for operand references.

Parity Equivalence in the check bit of transmitted and received data.

P (continued)

P register Program address register. The P register selects an instruction parcel from one of the instruction buffers. The contents of the P register are stored in the program address register field in the exchange package. The P register is 24 bits wide in Y-MP mode and 32 bits wide in EL mode.

PCI Programmable clock interrupt (flag). The programmable clock interrupt flag sets if the IPC interrupt mode bit is set and enabled and the counter in the programmable clock equals 0.

Pipelining An operation or instruction that begins before a previous operation or instruction finishes. Pipelining uses fully segmented hardware.

PN Processor number. The PN field in an exchange package indicates which CPU executed the exchange sequence.

Port A hardware or software access path to memory.

Power bulk converter Converts AC power to DC power.

PRE Program range error (flag). The interrupt flags field in the exchange package contains the PRE flag. The PRE flag sets when an instruction fetch is made outside the boundaries of the IBA and ILA registers.

Programmable clock A 32-bit counter in each CPU that is used to generate interrupts at selectable intervals.

Protocol Software that defines the precise way in which data is transferred from one place to another.

PS Program state status (bit). The interrupt modes field in the exchange package contains the PS status bit.

Q

Q Parity count (operation). When a Q appears in front of a register designator in a symbolic machine instruction, the calculation is a parity count operation.

R

R Rounded floating-point (operation). When an R appears in front of a register designator in a symbolic machine instruction, the calculation is a rounded floating-point operation.

R (continued)

RAM Random-access memory. A memory device that retains the stored data as long as power is applied. When power is removed from the device, the stored information is lost.

RD-1 Removable disk drive-1. A removable disk drive media option available in the CRAY Y-MP EL system.

RD-2 Removable disk drive-2. The removable maintenance disk drive in the CRAY Y-MP EL IOS.

Reciprocal approximation The mathematical process of approximating the value of a real number when one is divided by the number (1/n).

Register A hardware storage location for one word, byte, or element of data.

RPE Register parity error (flag). When a word is written into a register, a set of parity bits is generated and stored with the data bits. This set of parity bits is compared to another set that is generated when the word is read out of the register. An error is indicated when the two sets do not match. Parity errors set the register parity error (RPE) flag in the exchange package.

RT Load real-time clock (instruction). The RT instruction loads the real-time clock register with the contents of an S register.

RTC Real-time clock. The RTC is a 64-bit counter that advances one count each clock period.

S

Scalar A single numerical value that represents a single aspect of a physical quantity.

Section A major addressable division of central memory that may be further divided into subsections and banks.

Semaphore A 1-bit value stored in a register and used by programs to communicate the occurrence of an event.

Shared registers Registers that are available for more than one CPU to write to and read from.

Single-error correction/double-error detection (SECDED) A method of detecting whether one or more bits in a word has an incorrect value. If only one bit has an incorrect value, that bit can be changed back to the correct value.

S (continued)

Small computer system interface (SCSI) devices — Components of the MWS-EL that store and retrieve data used in the workstations.

S register — Scalar register. The S registers are the source and destination registers for operands executing scalar arithmetic and logical instructions.

SB — Shared address (register). The SB register is a shared register used for transferring address information from one CPU to another.

Segmentation — An operation that is divided into a discrete number of sequential steps, or segments. Fully segmented hardware is designed to perform one segment of an operation during a single CP.

SEI — Selected for external interrupts (flag). The interrupt modes field in the exchange package contains the SEI flag. When the SEI flag is set, this CPU is preferred for I/O interrupts.

Shared registers — S registers that are available for more than one CPU to write to and read from.

SIPI — Set interprocessor interrupt (instruction). The SIPI instruction sets an interprocessor interrupt request to a specific CPU.

SI-1 — SCSI-1. The small computer system interface-1 interfaces the TD-3 cartridge tape drive to the IOS.

SM — Semaphore (register). The SM register is a shared register used for control between CPUs.

SPARC — Scaleable Processor ARChitecture. A trademark of SPARC International, Inc.

ST — Shared T (register). The ST register is a shared register used for transferring data from one CPU to another.

Status field — The exchange package contains a status field that is used to determine the operating modes of a CPU.

Status register — A read-only register that is used to determine the operating modes of a CPU.

T

T register Intermediate scalar register. The T registers are used as intermediate storage for the S registers.

TC-2 Tape controller-2. An interface board that interfaces the TD-2 tape drive to the IOS.

TD-2 Tape drive-2. A 9-track tape drive subsystem option used in the CRAY Y-MP EL system.

TD-3 Tape drive-3. An 18-track tape drive subsystem option used in the CRAY Y-MP EL system.

U

UNICOS An operating system for Cray Research computer systems based primarily on the UNIX System Laboratories, Inc. UNIX System V and partially on the Fourth Berkeley Software Distribution. UNICOS is essentially the same in philosophy, structure, and function as UNIX, but has been enhanced to exploit the power of Cray Research computer systems.

V

Vector A single numerical value that contains information on more than one aspect of a physical quantity.

V register Vector register. Each V register contains 64 bits x 64 elements in Y-mode and 64 bits x 128 elements in X-mode.

VL Vector length (register). The program-selectable VL register controls the effective length of a vector register for any operation. The VL register is 7 bits wide in Y-MP mode and 8 bits wide in EL mode.

VM Vector mask (register). The VM field allows for the logical selection of particular elements of a vector.

VMEbus The VMEbus is a high-performance industry standard backplane that can connect any vendor's I/O controller to the IOS.

VNU Vector not used status (bit). The state of the VNU bit in the exchange package status field indicates whether vector instructions (077xxx or 140xxx through 177xxx) were issued during the execution interval.

W

Word A set amount of data that contains 64 bits of system data and 8 check bits.

WS Waiting for semaphore status (bit). The interrupt modes field in the exchange package contains the WS status bit. The waiting on semaphore bit sets if a test and set instruction (0034*jk*) is holding issue.

X

XA Exchange address (register). The XA register in the exchange package specifies the address of the first word of a 16-word exchange package loaded by an exchange sequence.

Z

Z Leading-zero count (operation). When a Z appears in front of a register designator in a symbolic machine instruction, the calculation is a leading-zero count operation.

BIBLIOGRAPHY

BIBLIOGRAPHY

Cray Research, Inc. Customer Publications

Submit orders for Cray Research, Inc. publications to the following address or telephone (800) 284-2729 extension 5907.

Cray Research, Inc.
Distribution
2360 Pilot Knob Road
Mendota Heights, MN 55210

Other publications related to the CRAY Y-MP EL system are listed below and are available from Distribution.

Preparing Your Site for a CRAY Y-MP EL Installation, (available second quarter of 1992).

This manual contains the technical information necessary to plan and prepare a site for the installation of a CRAY Y-MP EL system. This manual contains the information contained in a traditional site planning manual, such as wiring, power consumption, grounding, and access requirements. The name and format have been updated to support the concept that the CRAY Y-MP EL computer system is easy to install and can be located in any office environment.

A Brief Introduction to Your Cray Research Entry-level (EL) Computer System, Cray Research publication number SG-2410.

This manual provides an introductory description of CRAY Y-MP EL hardware and software. It includes the following information: UNICOS overview, Shells, utilities, features, network and connectivity, program generation, and programming tools.

UNICOS System Installation Bulletin for Cray Research Entry-level (EL) Computer Systems, (available second quarter of 1992).

This manual explains how to install 6.1 of the Cray Research, Inc. operating system UNICOS and release 8.3 of the Cray Research ELS Division, I/O subsystem (IOS). This bulletin also explains the procedures for installing diagnostics, customizing configuration, and how to recover from a `root (/)` file system crash.

INDEX

INDEX

Boldface numbers refer to illustrations and charts.

Reader Comment Form

Title: CRAY Y-MP EL Functional Description Number: HR-04027

Your feedback on this publication will help us provide better documentation in the future. Please take a moment to answer the few questions below.

For what purpose did you primarily use this manual?

_____Troubleshooting
_____Tutorial or introduction
_____Reference information
_____Classroom use
_____Other - please explain _____

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria and explain your ratings:

_____Accuracy _____
_____Organization _____
_____Readability _____
_____Physical qualities (binding, printing, page layout) _____
_____Amount of diagrams and photos _____
_____Quality of diagrams and photos _____

Completeness (Check one)

_____Too much information _____
_____Too little information _____
_____Just the right amount of information

Your comments help Hardware Publications and Training improve the quality and usefulness of your publications. Please use the space provided below to share your comments with us. When possible, please give specific page and paragraph references. We will respond to your comments in writing within 48 hours.

NAME _____
JOB TITLE_____
FIRM_____
ADDRESS_____
CITY_____STATE_____ZIP_____
DATE_____

[or attach your business card]

CRAY RESEARCH, INC.

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.

Attn: Hardware Publications and Training
770 Industrial Boulevard
Chippewa Falls, WI 54729

STAPLE

Cray Research, Inc.
Hardware Publications and Training
770 Industrial Boulevard
Chippewa Falls, WI 54729