CRAY Y-MP C90™ System Programmer Reference Manual

CSM-0500-000

Cray Research Proprietary

Cray Research, Inc.

Any shipment to a country outside of the United States requires a letter of assurance from Cray Research, Inc.

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

CRAY RESEARCH, INC.
Logistics
6251 South Prairie View Road
Chippewa Falls, WI 54729

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
	February 1992. Original printing.

PREFACE

The CRAY Y-MP C90 System Programmer Reference Manual describes the hardware architecture and functions of the CRAY Y-MP C90 computer system manufactured by Cray Research, Inc. (CRI). This manual is written primarily for system analysts and system programmers. The primary goal of this manual is to explain and define the special hardware features of the system in enough detail to help programmers write and optimize program code.

This manual is divided into the following tabbed sections.

Section 1, "Computer System Overview," introduces and describes the CRAY Y-MP C90 system components and support equipment.

Section 2, "CPU Shared Resources," describes the hardware shared by all central processing units (CPUs). Its primary emphasis is to define the functions, organization, and special hardware features of central memory, the I/O section, the interprocessor communication section, and the real-time clock. It also explains the shared paths access priority.

Section 3, "CPU Control," describes the basic CPU operations. The section explains the exchange mechanism in detail and defines and explains the deadstart, instruction fetch, and instruction issue sequences. The operations of the programmable clock, the status registers, and the performance monitor are also described.

Section 4, "CPU Computation Section," describes the CPU registers, functional units, and functional unit operations. Logical operations and integer and floating-point arithmetic are defined and explained in detail.

Section 5, "Parallel Processing Features," describes the parallel processing features most closely related to the hardware. This includes information and examples of pipelining and segmentaion, functional unit independence, and multiprocessing and multitasking.

Section 6, "Maintenance Channel," explains the operation of the maintenance channel used to troubleshoot system problems.

Section 7, "CPU Instructions," contains detailed descriptions of all instructions executed by the CRAY Y-MP C90 CPU. The instructions are listed by octal code starting with instruction 000000 and ending with instruction 177*ijk*. Special cases, hold issue conditions, and execution times are explained for each instruction or group of instructions.

The following conventions are used throughout this manual.

Convention	Description
Lowercase italic	Variable information.
x	An unused value.
n	A specified value.
(value)	The contents of the register or memory location designated by value.
Register bit designators	Register bits are numbered from right to left as powers of 2. 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 to element 63. Another exception is when the state of the 32 1-bit semaphore registers is loaded into an S register. SM0 goes into S register bit position 2^{63}, SM1 goes into S register bit position 2^{62}, and so on.
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, the instruction parcel in instruction buffers, and instruction forms, which are given in octal without the subscript.

The following list provides 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 indicates that the *j* field 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.

CONTENTS

1 COMPUTER SYSTEM OVERVIEW

2 CPU SHARED RESOURCES

2 CPU SHARED RESOURCES (continued)

3 CPU CONTROL

3 CPU CONTROL (continued)

4 CPU COMPUTATION SECTION

4 CPU COMPUTATION SECTION (continued)

5 PARALLEL PROCESSING FEATURES

6 MAINTENANCE CHANNEL

7 CPU INSTRUCTIONS

BIBLIOGRAPHY

INDEX

FIGURES

FIGURES (continued)

TABLES

1 COMPUTER SYSTEM OVERVIEW

The Cray Research, Inc. (CRI) CRAY Y-MP C90 computer system is a powerful, general-purpose supercomputer. The large memory, dual vector functional units, and fast clock speed of the CRAY Y-MP C90 computer system provide fast throughput, allowing for more effective use of computing power. The CRAY Y-MP C90 computer system is able to achieve extremely high multiprocessing rates by efficient use of the scalar and vector processing capabilities of the multiple central processing units (CPUs), and by use of the system's solid-state, random-access memory (RAM), and shared registers.

The CRAY Y-MP C90 computer system is carefully designed to deliver optimum overall performance. The unique architecture of the CRAY Y-MP C90 computer system enhances the scalar and vector processing capabilities inherent in all CRI computer systems.

Scalar processing is a sequential operation in which one instruction produces one result. Vector processing, on the other hand, uses a single instruction to perform the same operation sequentially on a whole set of operands to produce a set of results. When two or more vector operations are chained together, two or more different operations are performed simultaneously. Therefore, the computational rate for vector processing greatly exceeds that for conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adaptable to vector techniques.

The start-up time for vector operations on the CRAY Y-MP C90 computer system is short enough so that vector processing is more efficient than scalar processing for vectors containing as few as two elements. This feature allows for rapid long and short vector processing to be balanced with high-speed scalar processing while both vector and scalar processing are supported by powerful input/output capabilities.

The multiprocessor environment of the CRAY Y-MP C90 computer system allows the use of multiprocessing or multitasking techniques. Multiprocessing allows several programs to run concurrently on multiple CPUs of a single mainframe. Multitasking allows two or more parts of a single program to run in parallel and share a common memory space.

The CRAY Y-MP C90 computer system is composed of a mainframe; up to two input/output subsystems, model E (IOS-Es); and an optional SSD solid-state storage device, model E (SSD-E). The IOS-E and SSD-E may be housed in a single cabinet. Support equipment for the mainframe

includes a heat exchanger unit (HEU) and a refrigeration condensing unit (RCU). Power distribution occurs inside the mainframe; 400-Hz power is supplied by the mainframe's motor-generator set (MGS). Support equipment for the IOS-E and SSD-E includes RCUs, a power distribution unit (PDU), and an MGS. Figure 1-1 shows a CRAY Y-MP C90 mainframe with an attached IOS-E/SSD-E and two HEUs (one for the mainframe and one for the combined IOS-E/SSD-E). For more information on support equipment for your CRAY Y-MP C90 computer system, refer to the appropriate site planning reference manual.

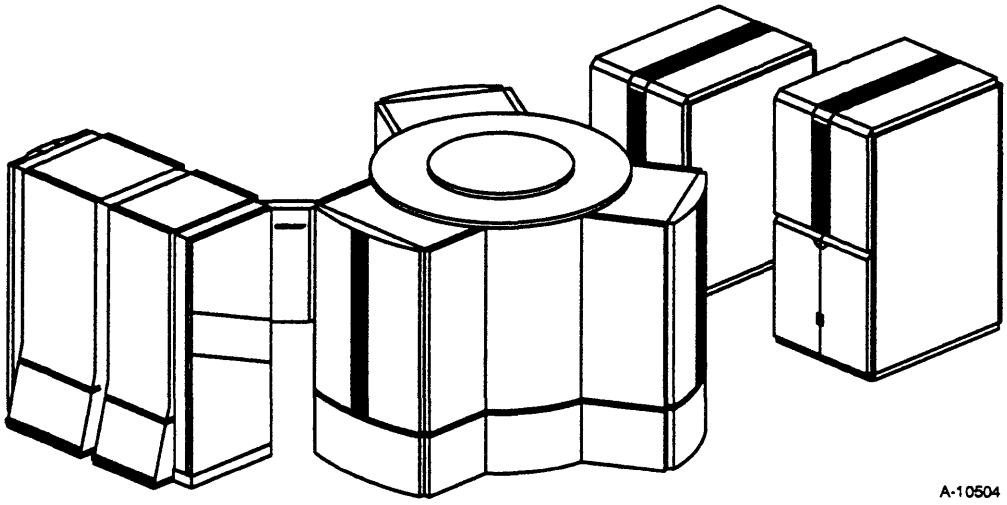


Figure 1-1. CRAY Y-MP C90 Computer System

Mass storage devices, such as disk drives, tape drives, and front-end interfaces (FEIs) are configured with the system through the IOS-E. A typical configuration is described in the "I/O Subsystem" subsection in this section.

Mainframe

The CRAY Y-MP C90 mainframe contains the CPUs, an I/O section, an interprocessor communication section, a real-time clock, and central memory. Figure 1-2 is a block diagram of a CRAY Y-MP C90 mainframe showing one CPU with the maximum I/O configuration. Each CPU has a computation section consisting of operating registers, functional units, and a control section.

The control section determines instruction issue and coordinates the three types of processing (vector, scalar, and address). The I/O section, interprocessor communication section, real-time clock, and central memory are shared by the CPUs and are called shared resources.

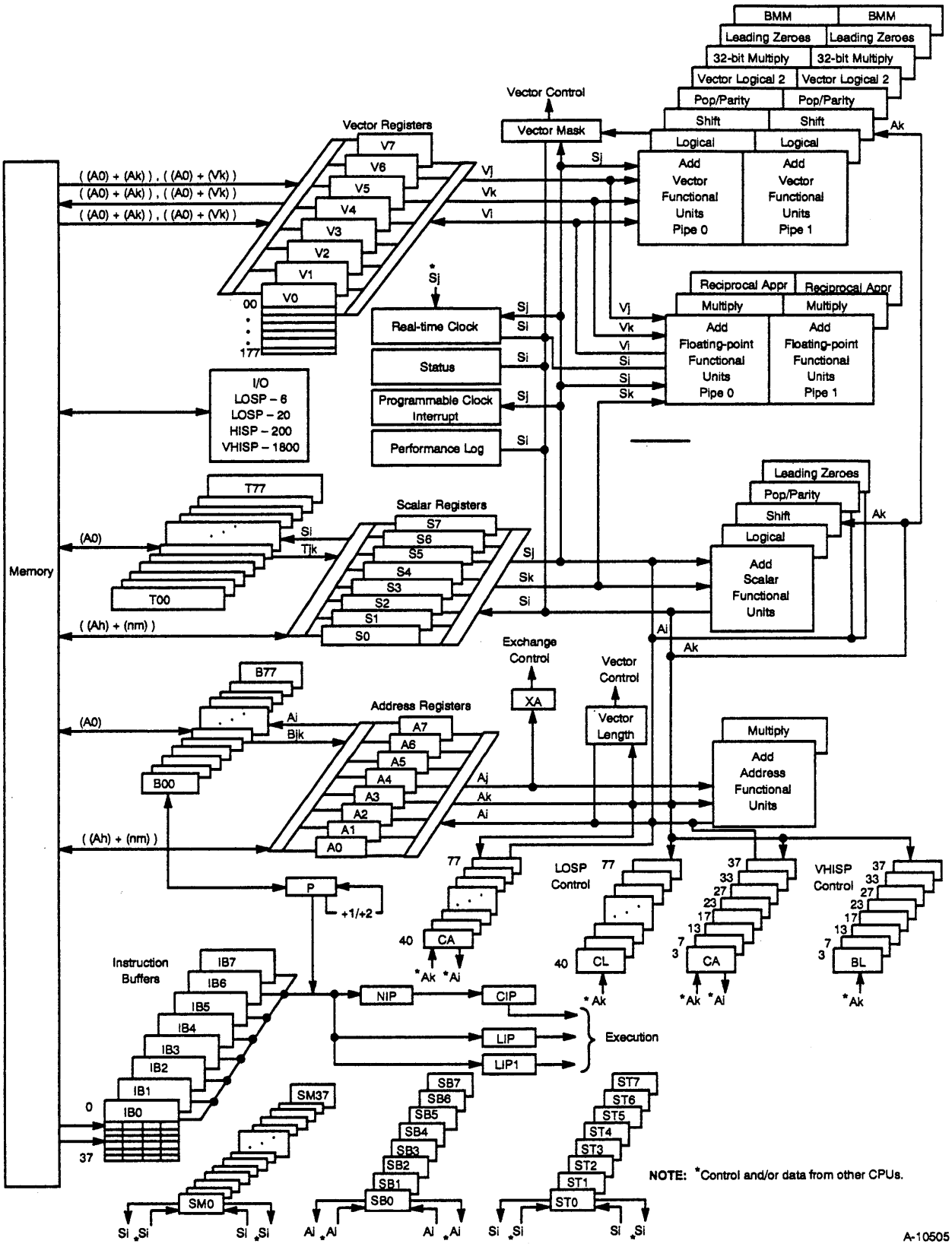


Figure 1-2. CRAY Y-MP C90 CPU Block Diagram

I/O Subsystem

All CRAY Y-MP C90 computer systems include an IOS-E; a second IOS-E is optional. The IOS-E is designed for rapid data transfer between the IOS-E's buffer memory and front-end computers, peripheral devices, and storage devices. The IOS-E also transfers data between its buffer memory and the mainframe's central memory.

Each IOS-E contains up to eight I/O clusters depending on the site specifications. An I/O cluster comprises four I/O processors (EIOPs), each with four independent I/O buffers and four channel adapters. Each channel adapter is dedicated to a specific peripheral device. Each I/O cluster also includes a dedicated low-speed (LOSP) channel and two dedicated high-speed (HISP) channels.

Each EIOP controls different portions of the system. Each EIOP has a memory section, a control section, a computation section, and an I/O section. I/O sections are independent and control some portion of the total I/O data stream for the IOS-E. IOS-E hardware allows simultaneous data transfers between the EIOPs and the mainframe's central memory over HISP channels.

The IOS-E also provides connections to the High Performance Parallel Interface (HIPPI) channel. The HIPPI channel connects external peripheral equipment, such as high-speed graphic devices, to the mainframe. CRI does not provide external peripheral equipment but does provide the hardware connections and software drivers for the HIPPI channel.

Figure 1-3 shows the minimum configuration for a CRAY Y-MP C90 computer system with two I/O clusters. For more information on the IOS-E, refer to the *IOS Model E System Programmer Reference Manual*, publication number CSM-1010-000.

SSD-E Solid-state Storage Device

The SSD-E is an optional high-performance device used for temporary data storage. The SSD-E transfers data between the mainframe's central memory and the SSD-E through special very high-speed (VHISP) channels. The actual speed of these transfers depends on the SSD-E and CRAY Y-MP C90 system configuration. The SSD-E can also be connected directly to an IOP through a HISP channel pair.

For more information on the SSD-E, refer to the *SSD Solid-state Storage Device System Programmer Reference Manual*, publication number CSM-1116-000.

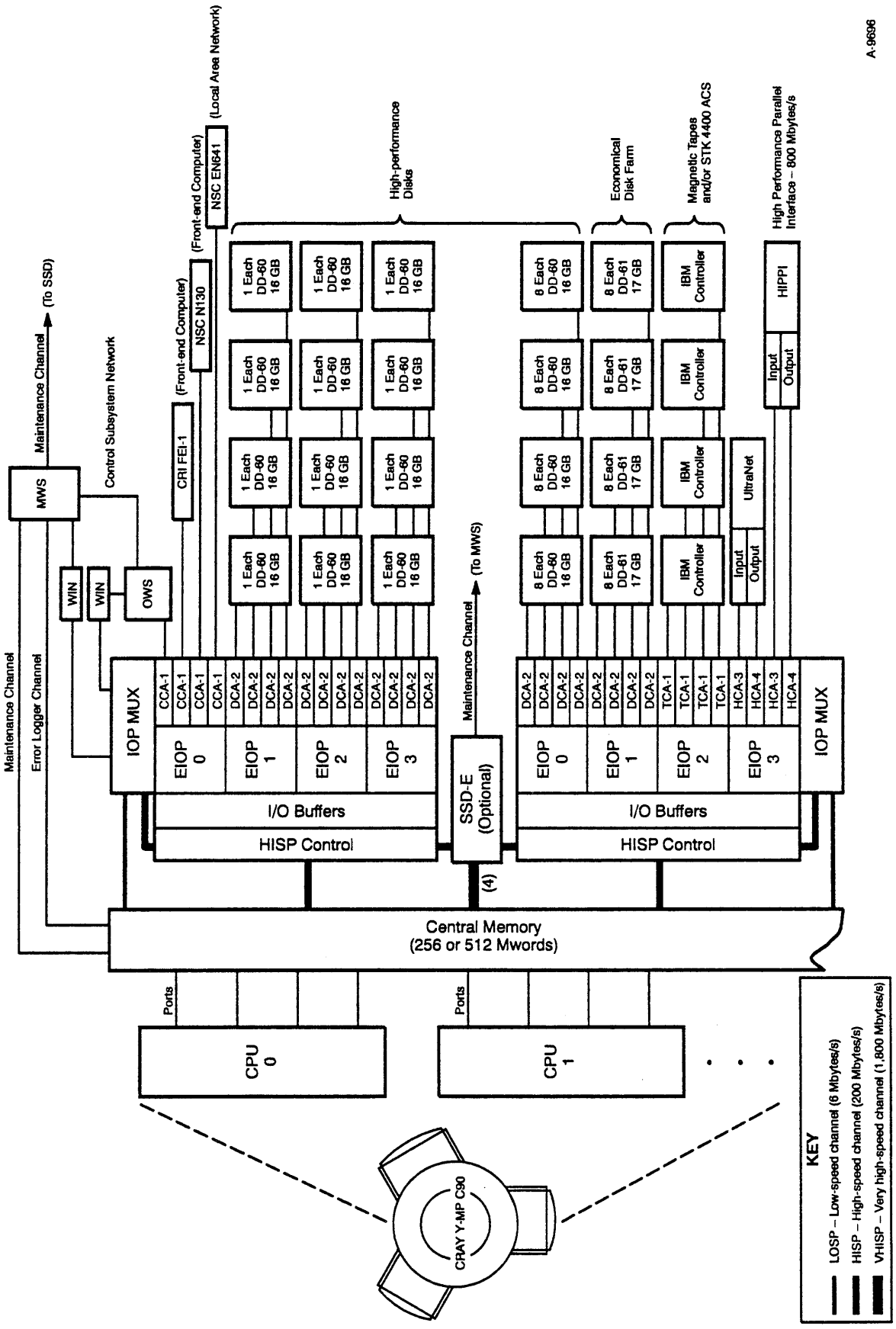


Figure 1-3. Minimum CRAY Y-MP C90 Configuration with Two I/O Clusters

Disk Storage Units

The CRAY Y-MP C90 computer system uses CRI disk storage units (DSUs) for mass data storage. A disk controller unit (DCU) serves as the interface between the DSUs and an EIOP. The EIOP and the DCU can transfer data between the EIOP and multiple DSUs without missing data or skipping revolutions. For more information on the DSUs, refer to the *60 Series Disk Systems Guide*, CRI publication number COM-1124-000.

Network Interfaces

The CRAY Y-MP C90 mainframe is designed to communicate easily with front-end computer systems and computer networks.

Standard front-end interfaces (FEIs) connect either the I/O channels of the CRAY Y-MP C90 mainframe or the IOS-E to front-end computer channels. These connections provide input data to the system and receive output from the system for distribution to peripheral equipment. An FEI compensates for differences in channel widths, machine word size, electrical logic levels, and control signals.

Some FEIs are housed in a stand-alone cabinet located near the host computer, and others are installed directly into the front-end computer system. Operation of the FEI is transparent to both the front-end computer and CRI system users.

As an option, a fiber-optic link (FOL-3) is available for some FEIs to provide front-end connections of up to 6,560 ft (2,000 m) and complete electrical separation from the CRAY Y-MP C90 computer system.

The CRAY Y-MP C90 mainframe can be connected to computer networks directly or through a front-end computer system.

Operator and Maintenance Workstations

The operator workstation (OWS-E) and the maintenance workstation (MWS-E) are based on a Sun 4/370 SPARCstation, 12-slot chassis. The SPARCstation is a Sun version of the reduced instruction set computer (RISC) architecture. A VMEbus is provided in slots 4 through 12 of the workstations.

Both workstations run the SunOS 4.1.1 operating system and OpenWindows 2.0 software; the MWS-E also runs the ME maintenance diagnostic software release, and the OWS-E runs the OWS-E software release. The Sun operating system is an enhanced version of UNIX; it combines features of UNIX System Laboratories, Inc.'s System V UNIX and Berkeley Software Distribution's version 4.3 UNIX.

The OWS-E is part of the CRAY Y-MP C90 computer system. The MWS-E is owned by CRI and is supplied as part of the maintenance contract; it enables CRI engineers to perform system maintenance independent of any customer activity on the system.

The OWS-E and MWS-E communicate through the Control Subsystem Network, which is a dedicated, modified, Ethernet cable link used only for maintenance and control-related functions.

The OWS-E provides a dedicated workstation that Cray Research analysts and customer operators use to operate, administrate, and monitor a Cray Research computer system. The OWS-E is also used for system boot, dump, and clear operations and for software and upgrade support.

The OWS-E communicates with the CRAY Y-MP C90 computer system through a LOSP channel from EIOP 0 in the IOS-E. The LOSP channel allows the mainframe to use the tape drives, disk drives, printer, and time-of-day clock. The OWS-E also provides an Ethernet interface to network workstations in a multiple-system site or for multiple-system operators.

The MWS-E provides multiple connections for hardware maintenance and monitoring of the CRAY Y-MP C90 computer system. The MWS-E supports CRI diagnostics, enhanced diagnostic displays, code simulation, and maintenance and error channels. It monitors environmental conditions and can shut down the system if severe variances occur. The MWS-E also serves as a platform for remote support, with customer approval. The MWS-E communicates with the CRAY Y-MP C90 computer system through a LOSP maintenance channel from the IOS-E.

Refer to the following publications for additional information on the OWS-E and MWS-E:

- *MWS-E User Guide*, CRI publication number CDM-1123-0A0.
- *Operator Workstation (OWS) Guide*, CRI publication number SN-3030.
- *MWS-E and OWS-E Hardware Maintenance Manual*, CRI publication number CMM-1122-0A0.

2 CPU SHARED RESOURCES

All central processing units (CPUs) in the CRAY Y-MP C90 mainframe share the following resources:

- Central memory
- I/O section
- Interprocessor communication section
- Real-time clock

Central Memory

Central memory consists of solid-state, random-access memory (RAM) that is shared by all the CPUs and the I/O section. Each memory word consists of 80 bits: 64 data bits and 16 error-correction bits (check bits). Storage for data and check bits is provided by 256 Kbyte x 4 bit bipolar complementary metal oxide semiconductor (BiCMOS) chips with a 15-ns access time. In order to improve memory access speed, central memory is divided into multiple banks that can be active simultaneously. The banks have a 6-clock period (CP) cycle time; each bank can be accessed once every 6 CPs.

In each CPU, the operating registers, instruction buffers, and exchange package have access to central memory through memory ports. Each CPU has four ports. Each of these ports is 2 words wide, allowing up to eight simultaneous memory references from each CPU. The I/O section shares one port in each CPU.

Memory Instructions

Table 2-1 shows all the CPU machine instructions that transfer data between CPU registers and central memory, or that affect memory operation. The contents of the data base address (DBA) register are added to instruction-generated memory addresses to form absolute memory addresses. Refer to "Absolute Memory Address Calculating" later in this section.

Table 2-1. Memory Instructions

Machine Instruction	CAL Syntax	Description	Type of Memory Reference
10*hi*00 *nm*	A*i* *exp*,A*h*	Read from ((A*h*) + *exp* + (DBA)) to A*i*.	Scalar
11*hi*00 *nm*	*exp*,A*h* A*i*	Write (A*i*) to ((A*h*) + *exp* + (DBA).	
12*hi*00 *nm*	S*i* *exp*,A*h*	Read from ((A*h*) + *exp* + (DBA)) to S*i*.	
13*hi*00 *nm*	*exp*,A*h* S*i*	Write (S*i*) to ((A*h*) + *exp* + (DBA).	
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words starting at address (A0) + (DBA) to B registers starting at register *jk*.	Block Transfer
035*ijk*	,A0 B*jk*,A*i*	Write (A*i*) words from B registers starting at register *jk* to memory starting at (A0) + (DBA).	
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words starting at address (A0) + (DBA) to T registers starting at register *jk*.	
037*ijk*	,A0 T*jk*,A*i*	Write (A*i*) words from T registers starting at register *jk* to memory starting at (A0) + (DBA).	
176*i*0*k*	V*i* ,A0,A*k*	Read (VL) words to V*i* starting at address (A0) + (DBA), incrementing by (A*k*).	Stride
1770*jk*	,A0,A*k* V*j*	Write (VL) words from (V*j*) to memory starting at address (A0) + (DBA), incrementing by (A*k*).	
176*i*1*k*	V*i* ,A0,V*k*	Read (VL) words to V*i* using memory addresses ((A0) + (V*k*) + (DBA)).	Gather
1771*jk*	,A0,V*k* V*j*	Write (VL) words from (V*j*) to memory using memory addresses ((A0) + (V*k*) + (DBA)).	Scatter
002300	ERI	Enable interrupt on operand range error.	None
002301	EBP	Enable interrupt on breakpoint.	
002400	DRI	Disable interrupt on operand range error.	
002401	DBP	Disable interrupt on breakpoint.	
002500	DBM	Disable bidirectional memory transfers.	
002600	EBM	Enable bidirectional memory transfers.	
002700	CMR	Complete memory references.	
002704	CPA	Complete port reads and writes.	

Table 2-1. Memory Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Memory Reference
002705	CPR	Complete port reads.	None
002706	CPW	Complete port writes.	

Instructions 10*h* through 13*h* perform scalar references; each instruction causes only 1 word to be transferred to or from memory. Instructions 034*ijk* through 037*ijk* perform block transfers. Each instruction transfers a block of from 1 to 127 words to or from consecutive locations in memory. Instructions 176*i*0*k* and 1770*jk* perform stride references. A block of from 1 to 128 words are transferred to or from memory locations separated by a constant increment (stride). Instructions 176*i*1*k* and 1771*jk* perform gather and scatter references. These instructions transfer from 1 to 128 words to or from randomly programmable locations in memory.

Instructions 002300 through 002706 affect memory operation. Instructions 002300 and 002400 set and clear the interrupt-on-operand range error (IOR) interrupt mode. When this interrupt mode is set and enabled, it allows interrupts on operand range errors. Refer to "Address Range Checking" in this section for a more complete explanation.

Instructions 002301 and 002401 set and clear the interrupt-on-breakpoint (IBP) interrupt mode. When this interrupt mode is set and enabled, it allows interrupts on write references within the breakpoint range, which should be set previously by instruction 0017*jk*.

Instructions 002500 and 002600 disable and enable the bidirectional memory mode. When this mode is enabled, block read and write operations can operate concurrently. When this mode is disabled, only block read operations can operate concurrently.

Instruction 002700 ensures completion of all memory references within the particular CPU issuing the instruction. Instruction 002700 does not issue until all previous memory references can complete in a fixed number of CPs. For example, a CPU is assured of receiving updated data when it issues a memory read instruction after instruction 002700. Used in conjunction with semaphore instructions, this instruction synchronizes memory references between processors.

Instructions 002704 through 002706 can be used to ensure sequential memory referencing within a CPU. These instructions do not issue until all previous memory references are at a stage of execution such that they

can run to completion before any subsequent memory references. Instruction 002704 ensures that all read and write references are at this stage. Instruction 002705 ensures that all read references are at this stage, and instruction 002706 ensures that all write references are at this stage.

In addition to direct memory references generated by CPU machine instructions, there are three ways that memory references are generated indirectly. First, a no-coincidence condition in a CPU causes an instruction fetch sequence to begin, during which 32 consecutive words are read from central memory to an instruction buffer. Second, an exchange sequence in a CPU causes 16 words to be read from and 16 words to be written into central memory. (For details on the fetch and exchange sequences, refer to "Exchange Mechanism" and "Instruction Fetch Sequence" in Section 3 of this manual.) The third type of indirect memory reference occurs when an I/O transfer to or from an external device causes a block of words to be read from or written to central memory. For details on I/O transfers, refer to "I/O Section" in this section.

Logical Organization

Figure 2-1 shows a CPU's memory ports and paths to central memory. Refer to this figure while reading the following paragraphs. Central memory is divided into 8 sections. Each section is divided into 8 subsections, and each subsection contains two 8-bank groups. This makes a total of 1,024 banks. This arrangement permits simultaneous memory references (two or more memory references that begin in the same CP) and overlapping memory references (one or more memory references that begin while another reference is in progress).

Memory Paths

Each CPU has an independent path into each memory section. (The I/O section does not have its own paths, but shares the paths of each CPU.) Independent paths allow each CPU to make up to eight simultaneous memory references, one reference to each section. Each CPU can have overlapping references in different sections without restrictions or within a section as long as each reference uses a different subsection. Simultaneous references to the same section are not permitted because each CPU has only one memory path into each memory section.

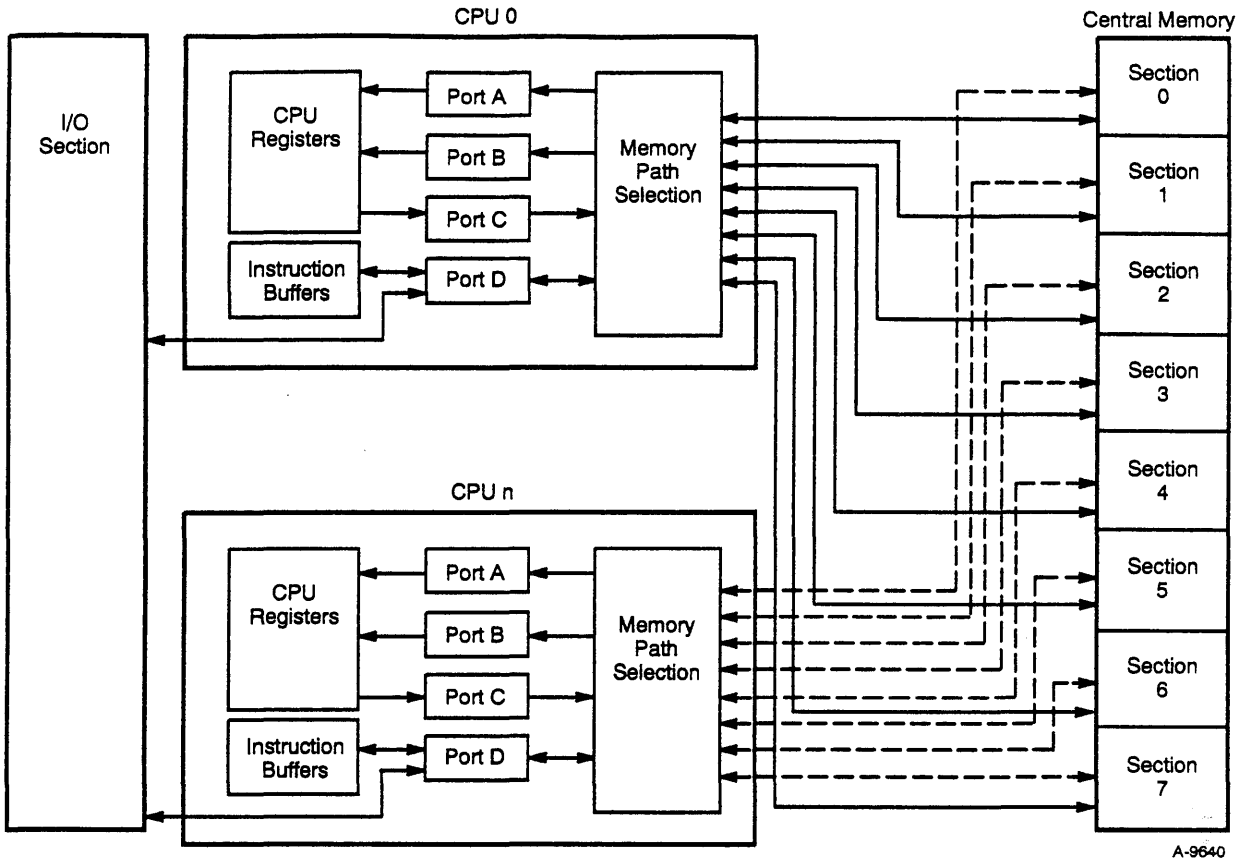


Figure 2-1. Central Memory Architecture

Simultaneous and overlapping memory references involving two or more CPUs have fewer restrictions than those involving a single CPU. Simultaneous and overlapping memory references from different CPUs can occur within a section and a subsection; however, each reference must use a different bank.

Memory Ports

Each CPU has four memory ports through which the CPU accesses its paths to central memory. Each port contains two pipes, allowing up to eight simultaneous memory references per CPU. Table 2-2 lists the specific read and write references allocated to each of the ports and pipes.

Table 2-2. Allocation of Memory References to Ports and Pipes

Port	Pipe	Reference Type	User
A	0	Read	A registers (10*h* instruction) B registers (034 instruction) S registers (12*h* instruction) V registers (176 instructions) Exchange data
A	1	Read	B registers (034 instruction) V registers (176 instructions)
B	0	Read	T registers (036 instruction) V registers (176 instructions)
B	1	Read	T registers (036 instruction) V registers (176 instructions)
C	0	Write	A registers (11*h* instruction) B registers (035 instruction) S registers (13*h* instruction) T registers (037 instruction) V registers (177 instructions) Exchange data
C	1	Write	B registers (035 instruction) T registers (037 instruction) V registers (177 instructions)
D	0	Read and Write	Instruction buffers I/O section
D	1	Read and Write	Instruction buffers I/O section

Ports A, B, and C are used by memory reference instructions and by the exchange sequence. Port D is used by the instruction buffers and the I/O section. With the exception of memory reads to vector (V) registers (instructions 176*i*0*k* and 176*i*1*k*), each type of memory reference uses one specific port. On a read to a V register, port B is used if available. If port B is reserved, port A is used if available. If both ports are reserved, the instruction holds issue until one of the ports is available. If both ports become available at the same time, port B is used.

The usage of either pipe 0 or pipe 1 depends on the type of reference to the port. For vector references, the even elements use pipe 0 and the odd elements use pipe 1. For fetch references, the even-address memory words use pipe 0 and the odd-address memory words use pipe 1. For B

and T register block transfers, the first word transferred always uses pipe 0, and the next word uses pipe 1; subsequent words alternate between the two pipes until the transfer is complete.

Ports A, B, and C

Ports A, B, and C operate differently for block and vector transfers than for scalar transfers. A memory reference instruction that transfers data to or from B, T, or V registers holds issue if the associated port is in use by another memory operation. When the port becomes available, the instruction issues and reserves the port. The port remains reserved until the instruction completes all its memory references. The port reservation is then cleared, making the port available for other memory operations. A block or vector transfer normally reads or writes 2 words of data each clock period (CP). However, if the instruction encounters a memory conflict during its execution, it temporarily suspends operation until the conflict is resolved. Therefore, the number of CPs the instruction runs and the number of CPs the port is reserved are unpredictable. Refer to "Conflict Resolution" in this section for additional information on port conflicts.

Block and vector transfer instructions that use different ports normally operate simultaneously. Under some circumstances, this mode of operation can cause memory references to occur in an unwanted sequence. For example, if instruction 035*ijk* (write to memory from a block of B registers) precedes instruction 176*i*0*k* (read from memory to a V register) and both instructions reference one or more of the same memory addresses, data from some memory addresses may be read before the new data is written to them. Both of these instructions can operate simultaneously, and the read instruction may reference an address before the write instruction.

There are several ways to prevent out-of-sequence references. Instruction 002700 (complete memory references), instruction 002704 (complete port reads and writes), or instruction 002706 (complete port writes) can be inserted between the write and read instructions. Although these instructions do not perform any operation, they prevent the read instruction from issuing until the write instruction completes all its memory references and clears the port C reservation. Usually, instructions 002704 and 002706 are used to insure sequential memory referencing within a CPU, and instruction 002700 is used to synchronize memory references between CPUs.

Clearing the bidirectional memory (BDM) mode in the exchange package also prevents out-of-sequence memory references. In this case, instructions that use port A or B also require port C to be available, and instructions that use port C require ports A and B to be available. The memory read instruction holds issue until the write instruction completes all its memory references.

Before it can issue, a scalar transfer instruction requires that ports A, B, and C be available to ensure sequential operation between block transfers and scalar references within a CPU. A scalar reference conflict is detected in CP 4 of execution. If a conflict occurs, up to two additional scalar references are still allowed to issue. A fourth scalar reference holds issue if the first reference still has a conflict. Scalar references always execute in the order they are issued within a CPU.

Port D

An instruction fetch sequence has priority over an I/O transfer in port D. That is, if a fetch request occurs while an I/O transfer is in progress, the I/O transfer is suspended and the fetch begins. When the fetch completes, the I/O transfer continues.

Conflict Resolution

A memory conflict occurs whenever a memory port tries to access a shared part of memory in use, or whenever two or more ports try to access a shared part of memory at the same time. Intra-CPU conflicts involve ports in the same CPU. Inter-CPU conflicts involve ports in different CPUs. In both cases, conflict resolution logic uses predefined priority schemes to sequence the conflicting memory references and to maximize overall machine throughput.

There are five types of memory conflicts: section, subsection, simultaneous subsection access, bank busy, and write bank busy. The following paragraphs explain each type of conflict and how the conflict is resolved.

Section Conflict

A section conflict occurs when two or more ports in the same CPU simultaneously attempt to access the same memory section. A section conflict occurs because there is only one path from each CPU to each memory section. The port with the highest priority level and no subsection conflict is allowed to begin its reference. All other conflicting ports hold reference for 1 CP. The following rules determine priorities between conflicting ports:

- Port D has priority over ports A, B, and C when it is used for an instruction fetch sequence.

- Port D normally has a lower priority than ports A, B, and C when it is used for an I/O transfer. However, if a port D I/O memory reference is forced to hold for 32 CPs, port D is temporarily given top priority so that one memory reference can proceed. Port D returns to a low-priority status after the reference begins.

- Among ports A, B, and C, any port that has an odd memory address increment has priority over ports that have an even increment. The following rules determine the type of increment (even or odd) for each port:

  - A port used by a block reference instruction has an address increment of 1, which is odd.

  - A port used by a stride reference instruction can have any constant increment (even or odd).

  - A port used by a gather or scatter instruction can have an increment that changes after each reference. For the purpose of conflict resolution, a gather or scatter instruction is always considered to have an odd increment.

- Among ports A, B, and C with the same type of memory increment, priority is determined by the relative time of instruction issue. The port used by the instruction first issued has the highest priority.

Subsection Conflict

Subsection conflicts occur because each memory reference by a CPU makes an entire memory subsection unavailable to all ports in the same CPU for 7 CPs. A subsection conflict occurs if any port in the same CPU attempts to make a reference to the same subsection during this interval. The new reference holds for 1 to 6 CPs until the old reference no longer needs the subsection. Subsection conflicts usually involve two or more ports, but may involve two references from the same port.

If two or more references are holding issue because of the same subsection conflict, a section conflict occurs immediately following the resolution of the subsection conflict. Another subsection conflict occurs 1 CP after the section conflict. For example, if port A is using a subsection and ports B and C attempt to use the same subsection while it is busy, ports B and C hold issue because of the subsection conflict. When the reference from port A no longer needs the subsection, the subsection conflicts disappear. Ports B and C are involved in a section conflict, which is resolved according to the priority rules previously described. The port with the higher priority makes its reference, and the port with the lower priority encounters a subsection conflict.

Simultaneous Subsection Access Conflict

Simultaneous subsection access conflicts occur when two or more ports in different CPUs attempt to access the same memory bank group at the same time. The CPU with the highest priority is allowed to make its reference. All other CPUs attempting to access the same bank group hold their references for 1 CP. Relative priorities between CPUs are determined by the value stored in a priority counter. This value increments by 1 each CP. For a given value of the priority counter, each of the CPUs is assigned the priority shown in Table 2-3. For example, if the priority count is 5, then CPU 5 has the highest priority and CPU 12 has the lowest priority. Following a simultaneous subsection access conflict, each CPU port forced to hold a reference encounters a bank-busy conflict.

Table 2-3. CPU Priority Matrix

Priority Count	Highest Priority ←															→ Lowest Priority
0	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
1	1	0	3	2	5	4	7	6	11	10	13	12	15	14	17	16
2	2	3	0	1	6	7	4	5	12	13	10	11	16	12	14	15
3	3	2	1	0	7	6	5	4	13	12	11	10	17	16	15	14
4	4	5	6	7	0	1	2	3	14	15	16	17	10	11	12	13
5	5	4	7	6	1	0	3	2	15	14	17	16	11	10	13	12
6	6	7	4	5	2	3	0	1	16	17	14	15	12	13	10	11
7	7	6	5	4	3	2	1	0	17	16	15	14	13	12	11	10
10	10	11	12	13	14	15	16	17	0	1	2	3	4	5	6	7
11	11	10	13	12	15	14	17	16	1	0	3	2	5	4	7	6
12	12	13	10	11	16	17	14	15	2	3	0	1	6	7	4	5
13	13	12	11	10	17	16	15	14	3	2	1	0	7	6	5	4
14	14	15	16	17	10	11	12	13	4	5	6	7	0	1	2	3
15	15	14	17	16	11	10	13	12	5	4	7	6	1	0	3	2
16	16	17	14	15	12	13	10	11	6	7	4	5	2	3	0	1
17	17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0

Bank-busy Conflict

Bank-busy conflicts occur because each memory reference by a CPU makes the referenced memory bank unavailable to all ports in all other CPUs for 6 CPs. A bank-busy conflict occurs if any port in a different CPU attempts to make a reference to the same bank during this interval. The new reference holds from 1 to 5 CPs until the old reference no longer needs the bank. If two or more CPUs are holding because of the same bank-busy conflict, a simultaneous subsection access conflict occurs immediately following resolution of the bank-busy conflict.

Write Bank-busy Conflict

Write bank-busy conflicts occur because within a subsection each write data path is shared between two memory banks. Banks 0 and 4, 1 and 5, 2 and 6, and 3 and 7 share write data paths. When a write reference is made by a CPU to a memory bank, both memory banks in the pair are unavailable to all ports in all other CPUs for 6 CPs. A write bank-busy conflict occurs if any port in a different CPU attempts to make a reference to either bank during this interval. The new reference holds from 1 to 5 CPs until the old write reference no longer needs the bank. If two or more CPUs are holding because of the same write bank-busy conflict, a simultaneous subsection access conflict occurs immediately following resolution of the write bank-busy conflict. Table 2-4 summarizes the five types of memory conflicts.

Table 2-4. Memory Conflicts

Conflict	Type	Duration	Resolution	Comment
Section	Intra-CPU	1 CP	The highest-priority port with no subsection conflict makes reference. Other ports hold reference.	Followed by a subsection conflict if references are made to the same subsection.
Subsection	Intra-CPU	1 to 6 CPs	Memory references for the port hold until the reference in progress is complete.	Followed by a section conflict if two or more references are forced to hold.
Simultaneous Subsection Access	Inter-CPU	1CP	The highest-priority CPU makes reference. Other CPUs hold reference.	Followed by a bank-busy conflict in each CPU that was forced to hold.
Bank-busy	Inter-CPU	1 to 5 CPs	Memory references for the port hold until the reference in progress is complete.	Followed by a simultaneous subsection access conflict if two or more references are forced to hold.
Write Bank-busy	Inter-CPU	1 to 5 CPs	Memory references for the port hold until the write reference in progress is complete.	Followed by a simultaneous subsection access conflict if two or more references are forced to hold.

Memory Addressing

Memory addresses are 29 bits long, allowing up to 512 Mwords of storage to be referenced. Figure 2-2 shows the function of each address bit. In each of the section, subsection, and bank-select fields, the highest-numbered bit is most significant. With this arrangement, if the memory address advances sequentially, all memory sections are stepped through in turn. Next, all subsections are stepped through, then each bank group, and finally all banks are stepped through.

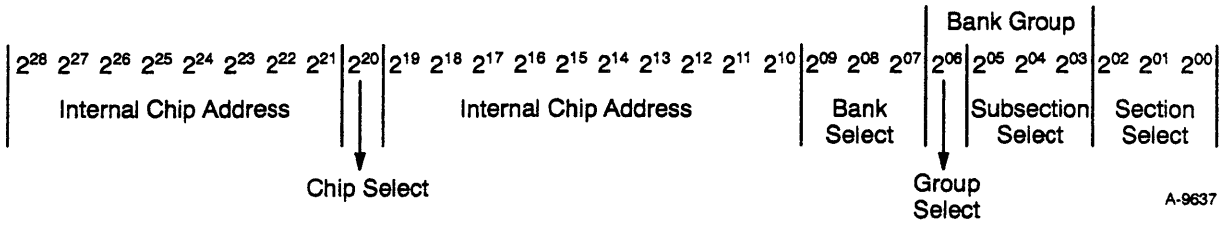


Figure 2-2. Memory Addressing

Address bits 2^6 through 2^3 collectively make up the bank group number. Thus, subsection 5 (101_2) in group 1 has a bank group number of 15. The bank group numbers run from 00 through 17_8.

The chip select bit is used to select one of two physical groups of chips on the memory module. The internal chip address selects a group of four bits on a 256 Kbyte x 4 bit memory chip.

Absolute Memory Address Calculating

Memory reference instructions listed in Table 2-1 calculate absolute memory addresses by adding combinations of the following values.

- A register contents
- V register contents
- DBA register contents
- Three-parcel instruction *nm* field contents

Each time an instruction makes a memory reference, the memory address generated by the instruction is added to the contents of the DBA register to form the absolute memory address. The use of the DBA register is further explained in the following subsection.

Address Range Checking

Four registers in the exchange package place program data and instruction areas in specific locations in memory and allocate specific amounts of memory to the areas. These registers allow all programs to be relocated. When a program is written, the programmer does not need to know where the instruction and data areas are located in memory. These registers also enable the programmer to restrict certain parts of memory from any program. A program can be halted if it tries to perform an instruction outside of its allowed instruction area or if it tries to read or write data outside of its allowed data area. When more than one program occupies memory at the same time, programs can be prevented from performing instructions or operating on data that belongs to other programs.

DBA Register

The data base address (DBA) register determines where the data area of a program begins in memory. Data addresses generated by memory reference instructions are relative to the DBA. The absolute address of any memory location is determined by adding the DBA to the address generated by the memory reference instruction. Refer again to Table 2-1 for a list of memory reference instructions.

DLA Register

The data limit address (DLA) register determines the highest absolute memory address the program can use for reading or writing data. Each time an instruction makes a memory reference, the absolute memory address generated is compared to the values stored in the DBA and DLA registers. If the absolute memory address is between the DBA and DLA, the reference is allowed to proceed. Otherwise, an out-of-range condition exists and the memory reference is aborted by disabling all chip selects and write enables in the referenced memory bank. For a memory write reference, no write operation is performed. For a memory read reference, all bits are set to 0.

If the interrupt-on-operand range (IOR) interrupt mode is set in the exchange package, an out-of-range condition sets the operand range error (ORE) interrupt flag and causes an exchange sequence to begin. If the IOR interrupt mode is not set, the program continues to run.

IBA Register

The instruction base address (IBA) register functions similarly to the DBA register, except that it operates on the instruction area of a program. Each time an instruction fetch sequence takes place, absolute memory addresses are formed by adding the relative addresses generated by the fetch control logic to the contents of the IBA register.

ILA Register

The instruction limit address (ILA) register functions similarly to the DLA register, except that it operates on the instruction area of a program and does not provide for continuing program execution when an out-of-range condition occurs. If an absolute memory address generated by an instruction fetch sequence is between the IBA and ILA, the fetch sequence is allowed to proceed. Otherwise, an out-of-range condition exists. An out-of-range condition sets the program range error (PRE) interrupt flag in the exchange package and causes an exchange sequence to begin.

The DBA, DLA, IBA, and ILA registers contain only address bits 2^{10} and above. Bits 2^0 through 2^9 are always 0; therefore, the content of these registers is always a multiple of 2000_8 ($1{,}024_{10}$). Adding the contents of the DBA or IBA register to a relative memory address does not change the section, subsection, or bank number. Therefore, memory conflicts can be determined from the relative addresses generated by instructions and the fetch control logic. It is not necessary to use absolute memory addresses to determine whether conflicts exist.

Address range checking is not performed during exchange sequences and I/O transfers. Memory addresses generated by these operations are absolute memory addresses.

Error Detection and Correction

Single-byte correction/double-byte detection (SBCDBD) monitors central memory for data errors. Memory errors involving only one 4-bit byte in each data word (single-byte errors) can be detected and corrected by the hardware. Double-byte errors can be detected but cannot be corrected. Errors involving more than 2 bytes cannot be reliably detected.

When a 64-bit word (bits 2^0 through 2^{63}) is written to memory, a 16-bit checkbyte is generated and stored in memory with the data word. (The check bits are numbered 0 through 15 and are stored as data bits 2^{64} through 2^{79}.) When the word is read from memory, a checkbyte is again generated and compared with the original checkbyte, using an exclusive

OR (XOR) operation. The resulting comparison is called a syndrome code. If all the bits in the syndrome code are 0, the 2 checkbytes are identical and no memory error occurred.

If there are one or more 1 bits in the syndrome code, some type of memory error occurred. The type of memory error (single-byte or double-byte) can be determined by interpreting the syndrome code. If a single-byte error occurs, the syndrome indicates the bit or bits within that byte that are in error. The SBCDBD logic toggles the incorrect bit or bits to the correct value. If a double-byte error occurs, the syndrome code indicates that there is an error, but it cannot pinpoint the incorrect bits. Errors involving more than 2 bytes produce unpredictable results. In some cases, errors produce unique syndrome codes that can be detected by the SBCDBD logic. In other cases, the syndrome code appears to be a no-error condition or a single- or double-byte error.

Table 2-5 shows the data bits used to generate each bit in the checkbyte. All data bits marked with an X in a row contribute to the corresponding check bit. The parity of all data bits marked with an X determines the state of the check bit. If the parity is even, the check bit is set to 0. If it is odd, the check bit is set to 1. For example, the data bits used to generate check bit 3 are bits 2^3, 2^7, 2^{11}, 2^{15}, 2^{33}, 2^{38}, 2^{43}, 2^{44}, 2^{48}, 2^{55}, 2^{58}, and 2^{61}. If an even number of these bits is 1, check bit 3 is set to logic 0; otherwise, it is set to logic 1.

If a syndrome code other than all 0's is generated, memory error information is sent to the error channel to help pinpoint the hardware failure. A nonzero syndrome code may also initiate an exchange sequence, depending on the state of two of the interrupt modes in the exchange package. If the interrupt-on-correctable memory error (ICM) interrupt mode is set, a single-byte (correctable) memory error sets the memory error – correctable (MEC) interrupt flag in the exchange package and starts an exchange sequence. If the interrupt-on-uncorrectable memory error (IUM) interrupt mode is set, a double-byte or detectable multiple-byte (uncorrectable) error sets the memory error – uncorrectable (MEU) interrupt flag and starts an exchange sequence. If either the ICM or the IUM interrupt mode is not set, the corresponding memory error does not start an exchange sequence and does not set an interrupt flag.

Table 2-5. Check Bit Generation

Check Bit	Data Bits																							
	Byte 15				Byte 14				Byte 13				Byte 12				Byte 11				Byte 10				Byte 9				Byte 8			
	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
0		X			X							X			X	X			X	X				X	X				X		
1	X	X			X			X			X			X				X					X		X			X	X			
2	X			X			X			X			X				X					X					X				X
3			X			X			X							X				X	X				X					X		
4			X	X				X	X					X																		
5		X					X		X			X	X	X																		
6	X					X					X		X			X																
7				X	X					X					X																	
8																				X				X				X				X
9																			X				X				X				X		
10																		X				X				X				X			
11																	X				X				X				X				
12				X				X				X				X		X			X							X			X	X	
13			X				X				X				X		X	X			X			X			X			X		
14		X				X				X				X			X			X			X			X			X			
15	X				X				X				X						X			X			X							X

Table 2-5. Check Bit Generation (continued)

Check Bit	Data Bits																											
	Byte 7				Byte 6				Byte 5				Byte 4				Byte 3				Byte 2				Byte 1				Byte 0			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																				X				X				X				X
1																			X				X				X				X	
2																		X				X				X				X		
3																	X				X				X				X			
4				X				X				X				X		X			X							X			X	X
5			X				X				X				X		X	X			X			X			X			X		
6		X				X				X				X			X			X			X			X			X			
7	X				X				X				X						X			X			X							X
8		X			X							X			X	X			X	X				X	X					X		
9	X	X			X			X			X			X				X					X		X			X	X	X		
10	X			X			X			X			X				X				X					X			X			X
11			X			X			X							X				X	X					X					X	
12			X	X				X	X					X																		
13		X					X		X			X	X	X																		
14	X					X					X		X			X																
15				X	X					X					X																	

Central Memory Performance Summary

Central memory has an intra-CPU subsection cycle time of 7 CPs and an inter-CPU bank cycle time of 6 CPs. That is, when a port in a CPU makes a memory reference, it reserves a subsection and a bank within the subsection. The reserved subsection is unavailable to all ports in that CPU for 7 CPs. The reserved bank is unavailable to all ports in all CPUs for 6 CPs.

Access time is the time required for an instruction to transfer one or more operands from central memory to an operating register. Access time depends on the type of register receiving the operand(s) and the number of operands being transferred. If no memory conflicts are encountered, the following access times apply for each register type:

- 24 CPs for A registers.
- 23 CPs for S registers.
- 26 plus (block length/2) CPs for B and T registers.
- 26 plus (vector length/2) CPs for V register stride references.
- 30 plus (vector length/2) CPs for V register gather references.

The maximum central memory data transfer rate equals the number of CPUs x 4 ports per CPU x 2 words per port per CP. The following are maximum data transfer rates within a CPU:

- 1 word (read or write) per 2 CPs for A and S registers.
- 6 words (4 read and 2 write) per CP for B, T, and V registers.
- 2 words (read or write) per CP for an instruction fetch or an I/O transfer.
- 1 word (read or write) per CP for an exchange sequence.

If memory conflicts occur, access times increase and data transfer rates decrease, causing program performance to degrade. The amount of performance degradation that a program encounters depends on many factors and is difficult to predict. However, performance degradation can be analyzed for vector stride instructions.

Normally, a vector stride instruction makes two memory references each CP until the instruction completes. However, if the stride is a multiple of 64 (that is, stride mod 64 = 0), the instruction attempts to access the same subsection each CP. Because a reference reserves the subsection for 7 CPs, the next reference encounters a subsection busy conflict and is forced to delay 6 CPs. Therefore, only one memory reference for each 7-CP interval is completed, compared to the twelve references that occur

in the absence of the subsection conflicts, making the relative performance equal to 1/12. In general, performance degradation results whenever the stride is a multiple of 16.

I/O Section

The I/O section of the mainframe is shared by all CPUs. The mainframe supports three channel types identified by their maximum transfer rates:

- Low-speed (LOSP) channels – 6 Mbytes/s or 20 Mbytes/s
- High-speed (HISP) channels – 200 Mbytes/s
- Very high-speed (VHISP) channels – 1,800 Mbytes/s

The I/O section uses ports D and D' (also referred to as pipes 0 and 1 of port D) in each CPU to transfer data between central memory and the I/O channels. Table 2-6 shows the assignment of I/O channels to each CPU. All numbers used in the table are octal.

Table 2-6. CPU I/O Channel Assignments

CPU	LOSP Channel		HISP Channel	VHISP Channel
	Input	Output	(Input and Output)	(Input and Output)
0	40	41	0, 1	
1	42	43		3
2	44	45	2, 3	
3	46	47		7
4	50	51	4, 5	
5	52	53		13
6	54	55	6, 7	
7	56	57		17
10	60	61	10, 11	
11	62	63		23
12	64	65	12, 13	
13	66	67		27
14	70	71	14, 15	
15	72	73		33
16	74	75	16, 17	
17	76	77		37

Each CPU provides access to central memory for one LOSP channel pair (one input and one output) and either two HISP channels or one bidirectional VHISP channel. The LOSP channels are normally used to transfer control information between the mainframe and I/O subsystem (IOS). Each of these channels can be programmed from any CPU in the mainframe. The HISP channels are used to transmit data between the mainframe and an IOS and are programmed from the IOS. The VHISP channels transfer data between the mainframe and a solid-state storage device (SSD). Like the LOSP channels, the VHISP channels can be programmed from any CPU in the mainframe.

The following subsections describe each of the I/O channels and provide programming information for the LOSP and VHISP channels. For information on programming the HISP channels, refer to the *IOS Model E System Programmer Reference Manual*, publication number CSM-1010-000.

LOSP Channels

One LOSP (6 Mbytes/s) channel pair or one enhanced LOSP (20 Mbytes/s) channel pair is provided for each CPU; the user-selectable option is installed with the machine. Unless specifically stated otherwise, the term *LOSP channel* used in this subsection refers to both types of LOSP channel. These channel pairs transmit data between central memory and an external 16-bit asynchronous device, normally an IOS. Each channel pair consists of an input and an output channel. Each channel in the pair sends data in 16-bit parcels. An input channel assembles 4 parcels to make a 64-bit word. An output channel disassembles a 64-bit word into 4 parcels. Each channel provides data-error detection (but not correction) by sending 4 parity bits with each parcel.

Control for each channel is provided by three signals: Ready, Resume, and Disconnect. On an input channel, the external device transmits Ready and Disconnect signals to the mainframe, and the mainframe transmits a Resume signal to the external device.

The following steps show the normal control sequence for a LOSP input channel:

1. The external device places a parcel of data and 4 parity bits on the channel.

2. The external device then activates a Ready signal to inform the mainframe that data is waiting on the channel.

3. The mainframe reads the data and parity bits from the channel and checks for parity errors.

4. The mainframe then activates a Resume signal to indicate that it has received the data.

5. Steps 1 through 4 are repeated until all data is transferred.

6. The external device activates a Disconnect signal to indicate that the transfer is complete.

7. The mainframe activates a Disconnecting Resume signal, which acts as a Master Clear signal to both the mainframe and external device.

The normal control sequence for an enhanced LOSP input channel is similar to the above sequence. The only difference is that the Resume signal mentioned in Step 4 is activated only once for every four Ready signals. After receipt of parcel 0 of each word, the mainframe activates a Resume signal to indicate that it is ready for the next 4 parcels of data together with their Ready signals. The final Resume signal, however, is sent in response to a Disconnect signal, and it can be sent only after the final word transferred is stored.

An output channel uses the same control signals, but the signal directions are reversed. That is, the mainframe transmits Ready and Disconnect signals to the external device, and the external device transmits Resume signals to the mainframe. Each output channel also sends a Master Clear signal to the external device. The Master Clear signal can be set or cleared under program control from any CPU.

Each LOSP channel has two registers that can be loaded from any CPU. The channel address (CA) register contains the address of the next word in central memory to be transferred. When an I/O transfer begins, the CA register contains the address of the first word to be transferred. After the first word is transferred, the CA register increments. The next word is transferred and the CA register again increments. This process continues until all words are transferred.

The contents of the channel limit (CL) register determine the address of the last word in central memory to be transferred. An I/O transfer completes when the address contained in the CA register equals the address stored in the CL register. All words between (CA) and (CL) – 1 are transferred; that is, all words starting at the initial address stored in the CA register through 1 less than the address stored in the CL register.

Channel Programming

Data transfers through a LOSP channel can be initiated by any CPU in monitor mode. The CPU does not need to take any further action after the transfer is initiated. The transfer operates as a background activity and the CPU may resume other processing. When the transfer

completes, the channel sets an I/O interrupt (IOI) flag in a CPU if the system I/O interrupts enabled (SIE) flag is set. The CPU that receives the interrupt request is not necessarily the CPU that initiated the transfer. Refer to "I/O Interrupts" later in this section for additional information.

Table 2-7 lists all instructions applicable to LOSP channels. Instructions 0010*jk* through 0012*j*3 perform channel control and can be executed only by a CPU in monitor mode. There is no hardware interlock between CPUs; the programmer must ensure that two CPUs do not try to control the same channel at the same time. Instructions 033*ij*0 through 033*ij*1 transmit I/O status information to register A*i*. These instructions are not limited to monitor mode and can be simultaneously executed by any number of CPUs.

Table 2-7. LOSP Channel Instructions

Machine Instruction	CAL Syntax	Description
0010*jk*	CA,A*j* A*k*	Set the CA register for channel (A*j*) to (A*k*) and begin I/O sequence.
0011*jk*	CL,A*j* A*k*	Set the CL register for channel (A*j*) to (A*k*).
0012*j*0	CL,A*j*	Clear the interrupt and error flags for channel (A*j*); clear device master-clear (output channels only); enable channel interrupt.
0012*j*1	MC,A*j*	Clear the interrupt and error flags for channel (A*j*); set device master-clear (output channels only); clear device ready-held (input channels only).
0012*j*2	DI,A*j*	Disable channel (A*j*) interrupts.
0012*j*3	EI,A*j*	Enable channel (A*j*) interrupts.
033*ij*0	A*i* CA,A*j*	Transmit the current address of channel (A*j*) to A*i* ($j \neq 0$).
033*i*00	A*i* CI	Transmit to A*i* the channel number of the highest priority channel requesting an interrupt.
033*ij*1	A*i* CE,A*j*	Transmit channel status word for channel (A*j*) to A*i* ($j \neq 0$).

The following sequence of instructions initiates a data transfer across the LOSP channel specified by register A*j*.

Step	Machine Instruction	CAL	Comment
1	0011*jk*	CL,A*j* A*k*	Sets the CL register to (A*k*), where A*k* contains 1 + the address of the last word to be transferred.
2	0010*jk*	CA,A*j* A*k*	Sets the CA register to (A*k*), where A*k* contains the address of the first word to be transferred.

This sequence starts the I/O transfer and increments the CA register after each data word is transferred to or from the mainframe. On an output channel, the transfer stops when (CA) = (CL). On an input channel, the transfer stops when (CA) = (CL) or when the mainframe receives a Disconnect signal, whichever comes first.

Two important characteristics of LOSP channels must be kept in mind when programming an I/O transfer. First, load the CL register before the CA register; the transfer begins when the CA register is loaded, regardless of the contents of the CL register. Second, load the CA register with a value less than the contents of the CL register. Unpredictable results occur if the CA register is loaded with a value equal to or greater than the CL register.

Two auxiliary operations can also be programmed to a LOSP channel. These operations are usually used to initialize a channel after a deadstart or to resynchronize a channel after an error. The first operation involves a Ready signal received by an input channel.

When the Ready signal is received, it is held (latched) until the channel is ready to receive the data. It is sometimes useful to clear the ready-held condition because a Ready signal can be received when the channel is not active. Instruction 0012*j*1 performs this function.

The second auxiliary operation performs a master clear sequence on an external device through an output channel.

The following instructions perform the external master clear sequence:

Step	Machine Instruction	CAL	Comment
1	0012*j*0	CL,A*j*	Clears the input channel to ensure any external activity on the channel pair has stopped.
2	0012*j*1	MC,A*j*	Clears the output channel to ensure any external activity on the channel pair has stopped. Sets the device Master Clear signal.
	–	–	Delay. The required delay time is determined by the external device.
3	0012*j*0	MC,A*j*	Clears the output channel. Clears the device Master Clear signal.
	–	–	Delay. The required delay time is determined by the external device.

The 0012*j*0 and 0012*j*1 instructions used in the auxiliary functions also clear the channel interrupt and error flags. Refer to "I/O Interrupts" later in this section for more information on clearing the channel interrupt and error flags.

Channel Errors

LOSP channels detect two specific errors. Input channels detect parity errors and output channels (on enhanced LOSP channels only) detect Disconnecting Resume signals received from external devices. Either type of channel error sets the parity/disconnecting resume error flag (bit 2^{29}) and the channel error flag (bit 2^{30}) in the channel status word. Several other errors also set the channel error flag, as shown in Table 2-8. Bit 2^{31} of the channel status word is a done flag, indicating that the data transmission is complete. These are the only three bits of the channel status word used for LOSP error reporting. Instruction 033*ij*1 transfers the 32 bits of the channel status word to register A*i* and clears all other bits of the register.

Table 2-8. LOSP Channel Error Flag Settings

Input Errors	Channel Status Word Bits		
	2^{31}	2^{30}	2^{29}
Parity error	0	1	1
Transmission terminated on nonword boundary	1	1	0
Aborted reference	0	1†	0
Sequence error (Ready signal at wrong time)	0	1	0
Sequence error (Disconnect signal at wrong time)	1	1	0
Output Errors			
MISP Disconnecting Resume signal detected	1	1	1
Transmission terminated on nonword boundary	1	1	0
Aborted reference	0	1†	0
Sequence error (Resume signal when channel active)	0	1	0
Sequence error (Resume signal at wrong time)	0	1	0

† This bit is set only until the reference is completed.

When an input channel detects a parity error, it sets the parity error flag but does not interrupt the data transfer. The word in error is written into central memory. All data received after the parity error is cleared before being written into central memory. It is not possible to determine which parcel of the word caused the parity error. When the transfer is completed, the parity error flag sets the channel error flag. There is no way to inform the external device of a parity error.

Instruction 0012*j*1 issued to an enhanced LOSP input channel generates a Disconnecting Resume signal to the complementary output channel. This signal is used to request that the output channel restart or resynchronize with the input channel. When the signal is detected by the enhanced LOSP output channel, it sets the disconnecting resume error flag as well as the channel error flag.

The LOSP input channels accept full-word (4-parcel) transfers only. A channel error is reported if the data transmission terminates on a nonword boundary. The last full word of a transfer is always written to memory, whether or not all of the parcels contain valid data.

An aborted reference error occurs when a clear channel command is issued to any active channel. The channel error flag sets and holds as long as any memory references are pending. When all memory

references are completed, the channel error flag clears. The done flag remains clear. The program should issue another clear channel command after the aborted reference error clears and before restarting the channel.

When either an input or output channel receives a control signal (Ready, Resume, or Disconnect) when it is not expected, the channel error flag is set. Refer to "I/O Interrupts" in this section for more information on the channel error flags and interrupts.

HISP Channels

Two HISP channels are provided for each even-numbered CPU; these channels transfer data between central memory and an external device, normally an IOS. Each channel uses 64-bit data words with 8 check bits for error detection and correction using a single-error correction/double-error detection (SECDED †) scheme. Data is transmitted in 16-word blocks.

The HISP channels are under control of the external devices; their operation is transparent to the CPUs in the mainframe. There are no CPU instructions to control or monitor channel operations, and no CPU interrupt requests are generated. Channel errors can be detected at either end of the channel but are reported to the external device.

VHISP Channels

One VHISP channel is provided for each odd-numbered CPU for a maximum of eight VHISP channels per mainframe. The VHISP channels transfer data between central memory and an SSD.

Each VHISP channel is 128 bits wide. Two parallel 64-bit channels are used; each channel has 8 check bits for SECDED. Data is transmitted in blocks with each block containing sixty-four 64-bit words. Unlike the LOSP channels, the VHISP channels are bidirectional. One channel number applies to both an input and an output channel; the channel can be active in only one direction at a time.

Two registers for each channel can be loaded from any CPU. The channel address (CA) register contains the address of the next word in central memory to be transferred. When an I/O transfer begins, the CA register contains the address of the first word. As each word is written to or read from central memory the CA register increments. The next block of words is transferred and the CA register again increments. This process continues until all words are transferred.

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

The block length (BL) register determines the number of 64-word blocks to be transferred. The BL register decrements after each block is transferred. An I/O transfer is complete when the content of the BL register is equal to 0.

Channel Programming

Programming a VHISP channel is similar to programming a LOSP channel. Most of the same instructions (privileged to monitor mode) are used, but instructions 0010*jk*, 0011*jk*, and 0012*j*0 operate differently than they do for the LOSP channels. These differences are explained in the following paragraphs. Table 2-9 lists the instructions applicable to the VHISP channel.

Instruction 0010*jk* performs two functions. The first time it executes, it determines the starting block address in the SSD. The second time it executes, it loads the channel's CA register, which determines the starting address in central memory.

Instruction 0011*jk* loads the channel's BL register and determines whether to do an input or an output transfer. The A*k* register bits 2^0 through 2^{17} are loaded into the BL register. The A*k* register bit 2^{23} determines the transfer direction. If this bit equals 0, data is transferred from the SSD to the mainframe. If this bit equals 1, data is transferred from the mainframe to the SSD. Instruction 0011*jk* also initiates the VHISP I/O sequence.

Table 2-9. VHISP Channel Instructions

Machine Instruction	CAL Syntax	Description
0010*jk*	CA,A*j* A*k*	First occurrence: set SSD starting block address for channel (A*j*) to (A*k*). Second occurrence: set CA register for channel (A*j*) to (A*k*).
0011*jk*	CL,A*j* A*k*	Set the BL register for channel (A*j*) to (A*k*), select input or output transfer, and begin I/O sequence.
0012*j*0	CL,A*j*	Clear the interrupt and error flags for channel (A*j*).
0012*j*2	DI,A*j*	Disable channel (A*j*) interrupts.
0012*j*3	EI,A*j*	Enable channel (A*j*) interrupts.
033*i*00	A*i* CI	Transmit to A*i* the channel number of the highest priority channel requesting an interrupt.
033*ij*1	A*i* CE,A*j*	Transmit channel status word for channel (A*j*) to A*i* ($j \neq 0$).

Instruction 0012*j*0 clears the channel interrupt flag and the channel status word. It does not perform additional functions when used for a VHISP channel.

Instruction 033*ij*1 transmits the channel status word to register A*i*. Table 2-10 lists the bits of the VHISP channel status word.

Table 2-10. VHISP Channel Status Word

Bit Position	Description
$2^0 - 2^{23}$	Block length (BL) register bits $2^0 - 2^{17}$.
$2^{24} - 2^{25}$	Not used (forced to 0).
2^{26}	Channel transfer in progress.
2^{27}	Block length error.
2^{28}	Uncorrectable (double-bit) error in SSD.
2^{29}	Uncorrectable (double-bit) error in mainframe.
2^{30}	Fatal error.
2^{31}	Complement of done flag.

The following sequence of instructions initiates a transfer across a VHISP channel.

Step	Machine Instruction	CAL	Comment
1	0012*j*0	CL,A*j*	Clears channel (A*j*).
2	0010*jk*	CA,A*j* A*k*	Sets the SSD starting block address for channel (A*j*) to (A*k*).
3	0010*jk*	CA,A*j* A*k*	Sets the CA register for channel (A*j*) to (A*k*).
4	0010*jk*	CL,A*j* A*k*	Sets the BL register for channel (A*j*) to (A*k*), selects the transfer direction, and begins the I/O sequence.

I/O Interrupts

I/O interrupts are generated by the LOSP and VHISP channels to indicate a completed data transfer or unexpected error. Parity errors (LOSP input channels) and correctable data errors (VHISP channels) do not cause interrupts.

The system I/O interrupts enabled (SIE) flag determines whether or not an I/O interrupt request is allowed to interrupt a CPU. If the SIE flag is clear, no CPU can receive an I/O interrupt request. If the SIE flag is set, all I/O interrupts are directed to the lowest-numbered CPU that has both the interrupt on I/O (IIO) interrupt mode and the enable interrupt modes (EIM) flag set. The SIE flag is cleared automatically when an I/O interrupt request is sent to a CPU. The flag should be reset only by instruction 001600 issued by the CPU that received the I/O interrupt.

The IOI interrupt flag sets in the CPU that receives the I/O interrupt, initiating an exchange sequence. The following steps should be performed by the CPU after the exchange sequence is finished:

1. Issue instruction 033*i*00 to determine which channel generated the interrupt.
2. Issue instruction 033*ij*1 to read the channel status word.
3. Issue instruction 0012*j*0 to clear the interrupt and error flags or the status word.
4. Retransmit the data if necessary.
5. Issue instruction 001600 to reset the SIE flag.

If two or more I/O channels generate interrupt requests to the same CPU, instruction 033*i*00 returns the lowest-numbered I/O channel requesting service. The instruction returns the next-lowest I/O channel number when that channel's interrupt flag is cleared. Instruction 033*i*00 returns a value of 0 after all interrupt flags are cleared.

Interprocessor Communication Section

The interprocessor communication section of the mainframe has three features that pass data and control information between CPUs:

- Shared registers to pass data between CPUs.
- Semaphore registers to allow synchronization of programs operating in different CPUs.

- Interprocessor interrupts to allow one CPU to initiate an exchange sequence in other CPUs.

These features are especially useful in multitasking environments.

The following paragraphs explain clusters, shared and semaphore registers, deadlock conditions, and interprocessor interrupts.

Clusters

The shared and semaphore registers are divided into 17 identical groups called clusters, as shown in Figure 2-3. However, only n+1 clusters are supported for an n-processor mainframe. Thus, a four-processor mainframe can support five clusters, and an eight-processor system can support nine clusters, etc. Each cluster contains eight 32-bit shared address (SB) registers, eight 64-bit shared scalar (ST) registers, and thirty-two 1-bit semaphore (SM) registers. These registers are described in the next two subsections.

Each CPU can be assigned to only one cluster at a time, giving it access to the registers in that cluster. The cluster number (CLN) field in the exchange package determines the cluster to which a CPU is assigned for a particular program. CPUs with the same cluster number share a common set of shared and semaphore registers.

Acceptable values for the 5-bit CLN field are 0 through n+1 for an n-processor mainframe. Thus, the CLN field in a 16-CPU system may be set to any value from 0 through 17 (0 through 21_8). Setting the CLN field to values outside its acceptable range causes unpredictable results. A CLN value of 0 prevents a CPU from accessing any shared or semaphore registers and causes the shared register instructions to perform no operation or to return a value of 0 to the destination register.

There are two ways to enter data into the CLN field: automatically during an exchange sequence or by issuing instruction 0014*j*3 with the CPU in monitor mode.

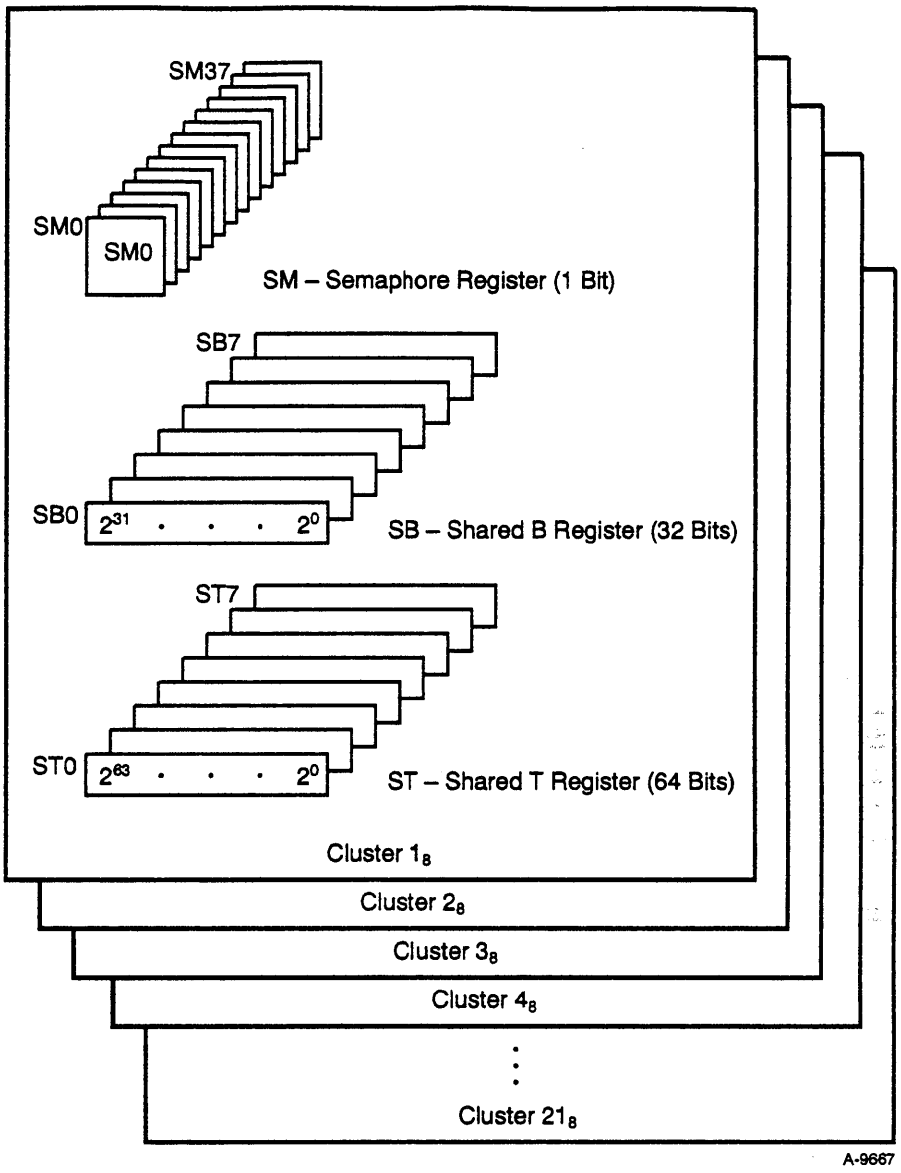


Figure 2-3. Shared Registers

Shared Registers

There are two types of shared registers: shared address (SB) and shared scalar (ST) registers. These registers function as intermediate storage between CPUs and provide a way to transfer data between operating registers in different CPUs. One CPU loads a shared register from its A or S registers; other CPUs assigned to the same cluster can then transfer the data from the shared register to their own A or S registers. Within a CPU, data is transmitted between the SB and A registers and between the ST and S registers. For data transfer between CPUs, the shared registers use the shared paths. Refer to "Shared Paths Access Priority" in this section for more information.

The SB and ST registers report parity errors to status register 7 (SR7). Refer to "Status Registers" in Section 3 of this manual for more information.

Table 2-11 lists all instructions that transmit data to or from the shared registers. In a CPU where the contents of the CLN register equal 0, the listed variations of instructions 026*ijk* and 072*ijk* return a value of 0, and the variations of instructions 027*ijk* and 073*ijk* perform no operation.

Table 2-11. Shared Register Instructions

Machine Instruction	CAL Syntax	Description
026*ij*4	A*i* SB,A*j*,+1	Transmit (SB) designated by (A*j*) to A*i*, and increment (SB,A*j*) by 1.
026*ij*5	A*i* SB*j*,+1	Transmit (SB*j*) to A*i*, and increment (SB*j*) by 1.
026*ij*6	A*i* SB,A*j*	Transmit (SB) designated by (A*j*) to A*i*.
026*ij*7	A*i* SB*j*	Transmit (SB*j*) to A*i*.
027*ij*6	SB,A*j* A*i*	Transmit (A*i*) to SB designated by (A*j*).
027*ij*7	SB*j* A*i*	Transmit (A*i*) to SB*j*.
072*ij*3	S*i* ST*j*	Transmit (ST*j*) to S*i*.
072*ij*6	S*i* ST,A*j*	Transmit (ST) designated by (A*j*) to S*i*.
073*ij*3	ST*j* S*i*	Transmit (S*i*) to ST*j*.
073*ij*6	ST,A*j* S*i*	Transmit (S*i*) to ST designated by (A*j*).

Semaphore Registers

Semaphore (SM) registers allow a CPU to temporarily suspend program operation in order to synchronize operation with other CPUs. Each CPU assigned to a particular cluster can set or clear each SM register in the cluster and can perform a test and set instruction, as explained in the next paragraph. Each CPU in the cluster can also transmit all 32 SM registers to or from an S register. CPUs use the shared paths to set and clear semaphore registers. Refer to "Shared Paths Access Priority" in this section for more information.

Table 2-12 lists all instructions that use the SM registers. The 0034*jk* test and set instruction tests the state of the SM*jk* or SM,A*k* register. If the content of the designated SM register is 0, the 0034*jk* instruction

executes immediately. If the content of the designated SM register is 1, the 0034*jk* instruction holds issue until another CPU assigned to the same cluster clears the SM register. When the instruction issues, it sets the designated SM register. Instructions 0036*jk* and 0037*jk* clear and set the SM*jk* or SM,A*k* register.

Table 2-12. SM Register Instructions

Machine Instruction	CAL Syntax	Description
0034*jk*	SM*jk* 1,TS	Test and set semaphore *jk*, $0 < jk < 31_{10}$ (*j*2 = 0).
0034*jk*	SM,A*k* 1,TS	Test and set semaphore (A*k*), $0 < (Ak) < 31_{10}$ (*j*2 = 1).
0036*jk*	SM*jk* 0	Clear semaphore *jk*, $0 < jk < 31_{10}$ (*j*2 = 0).
0036*jk*	SM,A*k* 0	Clear semaphore (A*k*), $0 < (Ak) < 31_{10}$ (*j*2 = 1).
0037*jk*	SM*jk* 1	Set semaphore *jk*, $0 < jk < 31_{10}$ (*j*2 = 0).
0037*jk*	SM,A*k* 1	Set semaphore (A*k*), $0 < (Ak) < 31_{10}$ (*j*2 = 1).
072*i*02	S*i* SM	Transmit (SM) to S*i*.
073*i*02	SM S*i*	Transmit (S*i*) to SM.
0064*jk nm*	JTS*jk exp*	Branch to *exp* if (SM*jk*) = 1; else set SM*jk* (*j*2 = 0).
0064*jk nm*	JTS,A*k exp*	Branch to *exp* if (SM,(A*k*)) = 1; else set SM,(A*k*) (*j*2 = 1).

Instructions 072*i*02 and 073*i*02 transmit the SM register contents to or from the upper half of the S register (the lower half of the S register is not used). Figure 2-4 shows the relation between the SM registers and the bits of an S register.

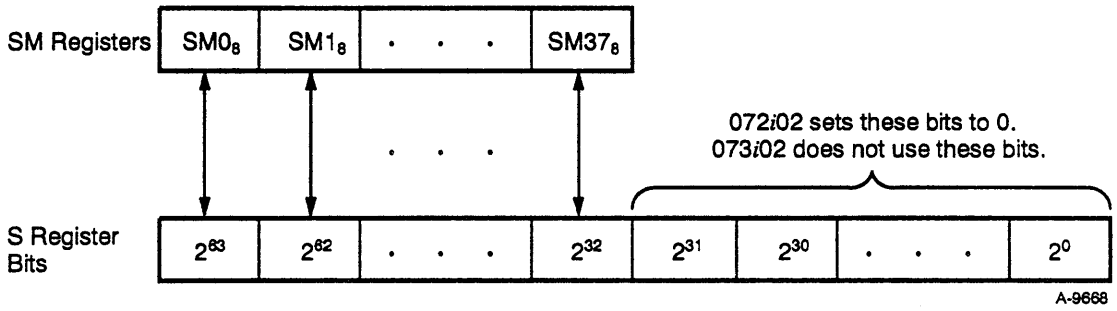


Figure 2-4. Relation between SM Registers and S Register Bits

If a CPU is not assigned to any cluster (CLN = 0), instructions 0034*jk*, 0036*jk*, 0037*jk*, and 073*i*02 perform no operation. Instruction 072*i*02 sets register S*i* to 0.

The following example shows how an SM register is used to synchronize the operation of two CPUs in a multitasking program. Both CPUs must be assigned to the same cluster number. In this example, CPU 0 computes a partial result needed by CPU 1, while CPU 1 computes a second partial result. CPU 1 then uses the two partial results as operands for further processing.

CPU 0	CPU 1
1. SM0 1 (003700)	
2. Compute partial result • • • • • • Place partial result in S1	3. Compute partial result • • Place partial result in S1 4. SM0 1,TS (003400)
5. ST0 S1 (073103)	
6. SM0 0 (003600)	
7. Continue processing • • • •	8. S2 ST0 (072203) 9. Continue processing • •

In Step 1, CPU 0 begins processing by setting register SM0, which indicates that it has not yet computed its partial result. In Steps 2 and 3, CPUs 0 and 1 begin computing the partial results. At the end of Step 3, CPU 1 places its partial result in register S1. CPU 1 now needs CPU 0's partial result before it can proceed. CPU 1 performs a test and set instruction (Step 4) on register SM0. Because register SM0 is already set, CPU 1 holds issue.

CPU 0 continues its computations and transfers its partial result to the S1 register. CPU 0 then transfers the partial result from S1 to register ST0 (Step 5). In Step 6, CPU 0 clears register SM0, which indicates that the partial result is ready in register ST0. CPU 0 can now continue with other processing (Step 7). SM0 is now cleared and the test and set instruction in CPU 1 issues, setting register SM0. CPU 1 then transfers CPU 0's partial result from register ST0 to register S2 (Step 8). CPU 1 now has its own partial result in register S1 and CPU 0's partial result in register S2 and can continue processing (Step 9).

Deadlock

A deadlock condition occurs when all CPUs assigned to a cluster are holding issue on a test and set (0034*jk*) instruction; that is, each CPU within the cluster is waiting for another CPU to clear an SM register. When this condition occurs, no instructions can execute in any of the CPUs assigned to the cluster.

There are two situations in which a deadlock occurs:

- All CPUs in the same cluster are holding issue on a test and set instruction.
- A single CPU is holding issue on a test and set instruction and there are no other CPUs in the same cluster. This situation can occur in one of two ways:
  - There is only one CPU assigned to a particular cluster, and that CPU issues a test and set instruction for an SM register currently set.
  - There are initially several CPUs assigned to the same cluster, one of which is holding issue on a test and set instruction. Then, all the other CPUs exchange to new programs with different cluster numbers.

In order to resolve the deadlock condition, a deadlock interrupt occurs. This interrupt sets the deadlock (DL) flag in the current exchange package of each CPU assigned to the cluster in which the deadlock occurred, which causes each affected CPU not in monitor mode to perform an exchange sequence.

Interprocessor Interrupts

Interprocessor interrupts allow a CPU to interrupt program execution in other CPUs. Table 2-13 shows the two instructions that involve interprocessor interrupts. These instructions can be executed only by a CPU in monitor mode.

Table 2-13. Interprocessor Interrupt Instructions

Machine Instruction	CAL Syntax	Description
0014*j*1	SIPI A*j*	Send an interprocessor interrupt request to CPU (A*j*).
001402	CIPI	Clear the interprocessor interrupt request.

When a CPU executes a 0014*j*1 instruction, the interprocessor interrupt (ICP) flag is set in the CPU designated by the contents of register A*j*, provided the designated CPU has its interrupt-on-interprocessor (IIP) interrupt mode set and enabled. If this CPU does not have IIP interrupt mode set and enabled, the interrupt request is held, and the flag is not set.

When the ICP flag sets, it initiates an exchange sequence. The program that begins running as the result of the exchange sequence should be in monitor mode and should execute instruction 001402 to clear the ICP flag. If this instruction is not executed, the ICP flag initiates another exchange sequence when the monitor mode program exits to a nonmonitor mode program.

If instruction 0014*j*1 is executed with the contents of register A*j* equal to the number of the CPU executing the instruction (if a CPU tries to interrupt itself), the instruction performs no operation.

Real-time Clock

The CRAY Y-MP C90 mainframe has a real-time clock (RTC) that increments synchronously with program execution and may be used to time the number of CPs for a program. The RTC consists of local clocks: one clock on each CPU. All local clocks run synchronously and appear to the programmer as a single clock. The RTC is a 64-bit counter that increments each CP except when being written to or read from. Table 2-14 shows the two instructions that write data to and read data from the RTC.

Table 2-14. RTC Instructions

Machine Instruction	CAL Syntax	Description
0014*j0*	RT S*j*	Transmit (S*j*) to the RTC register.
072*i*00	S*i* RT	Transmit (RTC) to S*i*.

The 0014*j*0 instruction can be issued only by a CPU in monitor mode; the CPU issuing this instruction updates the count of the local clocks on all other CPUs. Two or more CPUs should not issue this instruction simultaneously because there is no hardware to detect this condition and unpredictable results occur. It is the programmer's responsibility to avoid this situation. The 072*i*00 instruction may be simultaneously issued by any number of CPUs.

The RTC is normally used to determine the running time of a program or a segment of program code. The following sample instruction sequence can be used to determine the running time of a program.

Step	Machine Instruction	CAL	Comment
1	072100	S1 RT	Load S1 with starting time.
	–	–	Insert code to be timed here. Code must not use S1.
2	072200	S2 RT	Load S2 with ending time.
3	061121	S1 S2–S1	Load S1 with the difference between the starting and ending times.

At the end of this sequence, assuming no interrupts occur, register S1 equals 1 plus the number of CPs required to execute the code inserted between Steps 1 and 2.

Shared Paths Access Priority

Shared paths are used to pass data and control information between CPUs. Data contained in the SB, ST, and SM registers is transferred between CPUs along these paths. The shared paths are also used to transfer data from the S registers to the real-time clock and to transmit interprocessor interrupt signals and I/O instructions.

Only one CPU at a time can access the shared paths. An access conflict occurs when two or more CPUs attempt to use the shared paths simultaneously. These conflicts are resolved according to a two-level priority arbitration scheme, which is explained in the following paragraphs. Before an instruction using the shared paths issues, the CPU sends a Shared Register (SR) Request signal to check whether other CPUs are using the shared paths. This process takes 1 CP. The SR Request signal must then gain first arbitration level priority before it is latched. The CPU's latched SR Request signal must then gain second arbitration level priority before the CPU can latch an SR In Progress signal and issue the instruction. The SR In Progress signal is cleared when the CPU finishes executing the instruction. Shared and semaphore instructions require a minimum of 3 CPs to issue.

The shared paths access priority arbitration scheme is described in the following diagram and text.

	Highest Priority CPU											Lowest Priority CPU				
First Arbitration Level	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Second Arbitration Level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- First Level – Assuming no requests already exist in the first level, simultaneous SR Request signals from all CPUs are latched. The CPU with the highest physical processor number (PPNO) has the highest priority and latches an SR In Progress signal at the second level. Subsequent SR Request signals from CPUs with lower PPNOs than those already latched will hold issue until the CPUs with higher PPNOs complete their instructions (clear their SR In Progress signals).

- Second Level – All shared and semaphore instructions are issued while the CPU is in the second arbitration level. CPUs with the lowest PPNO have the highest priority in this level. While the second arbitration level is handling requests from the first level, any CPU with a higher PPNO can latch an SR Request in the first level.

This two-level arbitration enables all CPUs to have equal access to the shared paths. The following examples explain how the shared paths access priority arbitration scheme resolves access conflicts.

- If CPU 12 makes and latches an SR Request signal in the first level, CPUs 0 through 11 can send SR Request signals, but the requests are held in the CIP registers and do not latch. However, CPUs 13 through 15 can latch an SR Request signal while CPU 12 has an SR Request signal latched in the first level. CPUs 13 through 15 must wait for CPU 12 to clear its SR In Progress signal before one of them can issue its instruction. CPU 13 then has the highest priority and is processed next.

- If CPU 4 has an SR In Progress signal latched and CPUs 2 and 5 have SR Request signals latched, CPU 2 will issue before CPU 5 because CPU 2 has a higher priority than CPU 5 in the second level. (CPU 4 must clear its SR In Progress signal before CPU 2 can latch an SR In Progress signal and issue its instruction.)

Shared Register and Real-time Clock Troubleshooting

To assist in troubleshooting problems with the shared registers and real-time clock, Figure 2-5 is a block diagram of these registers showing the options involved and the signals that pass between them. For a more detailed description of this diagram, refer to the *CRAY Y-MP C90 Computer System Hardware Maintenance Manual*, publication number CMM-0502-000.

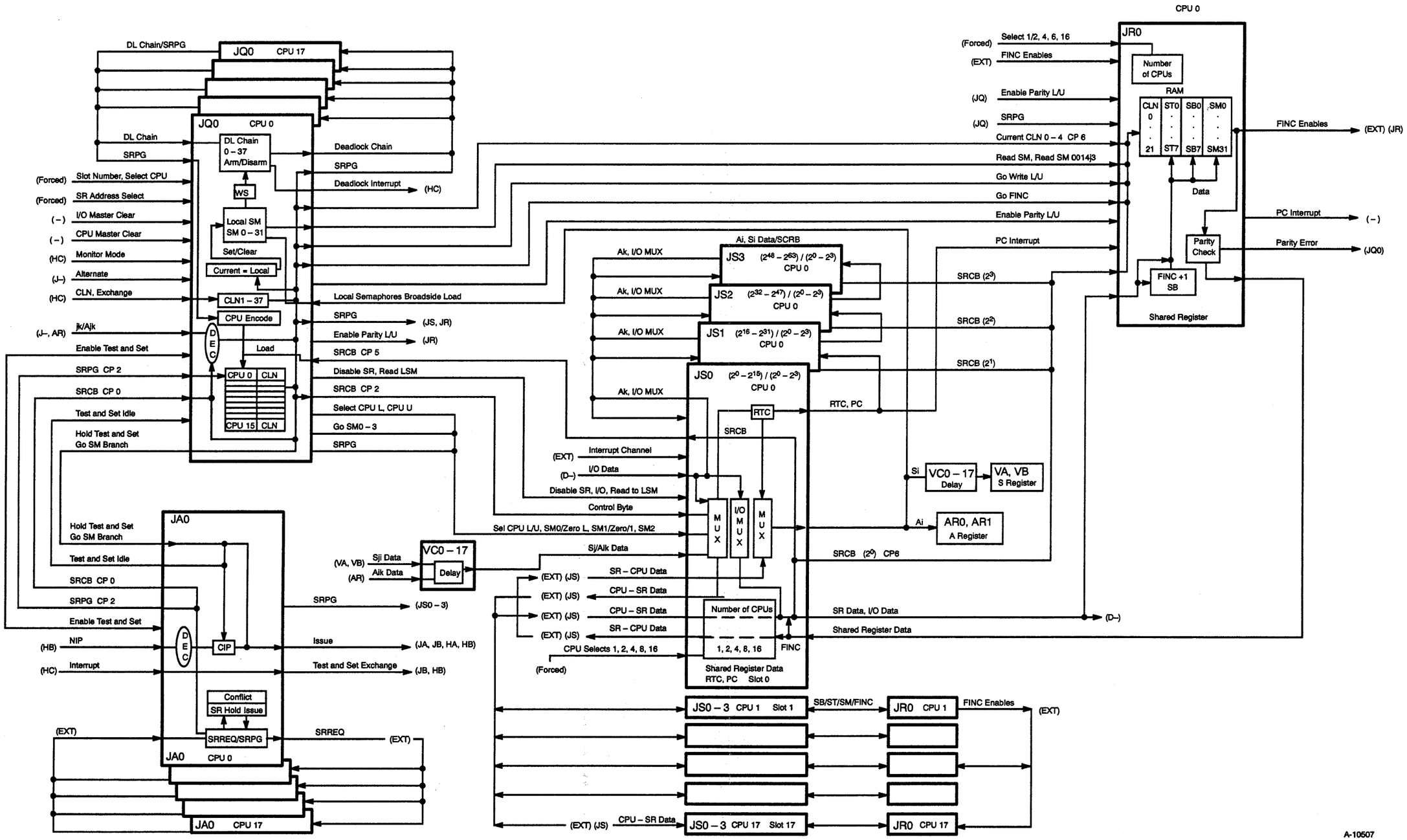


Figure 2-5. Shared Registers Block Diagram

3 CPU CONTROL

Each central processing unit (CPU) is assigned tasks and is controlled in the execution of those tasks through exchange sequences, fetch sequences, and issue sequences. These three sequences are closely related. For an initial deadstart program or a new program to run, an exchange sequence must occur. This sequence of steps sets several important parameters of the program in the CPU and may initialize some of the CPU's operating registers. A fetch sequence begins immediately after the exchange sequence and transfers a block of instructions from memory to an instruction buffer. The issue sequence then 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, allows the instruction to be executed.

As the instruction executes, the P register increments, causing new instructions to be selected from an instruction buffer and to move through the issue sequence. When a desired instruction is not in an instruction buffer, another fetch sequence occurs, retrieving another block of instructions from memory. This overall process continues until either the program terminates or is interrupted, at which time another exchange sequence occurs and the whole process starts over.

This section describes the exchange mechanism, the instruction fetch sequence, and the instruction issue sequence unique to each CPU. The programmable clock, the status registers, and the performance monitor are also briefly described.

Exchange Mechanism

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

The following subsections describe the contents of the exchange package and explain the exchange sequence in more detail.

Exchange Package

An exchange package is a 16-word block of data stored in a reserved area of memory that contains the initial parameters for a particular computer program. In addition to initializing the program, these parameters are also used to provide continuity if a program stops and restarts from one section of the program to the next.

The exchange package includes the contents of the address (A) and scalar (S) registers. The contents of the intermediate address (B), intermediate scalar (T), vector (V), vector mask (VM), shared B (SB), shared T (ST), and semaphore (SM) registers are not saved in the exchange package. Data in these registers must be stored and replaced as required by the program supervising the object program or by any program that needs this data.

Figure 3-1 shows the format of a portion of the exchange package. The 32 bits of words 0 through 7 that are not shown hold the contents of the A registers, and words 10_8 through 17_8 hold the contents of the S registers. The following subsections define and explain the fields of the exchange package.

NOTE: The exchange package bits are numbered from left to right with bit 0 assigned to bit position 2^{63}.

Program Address Register Field

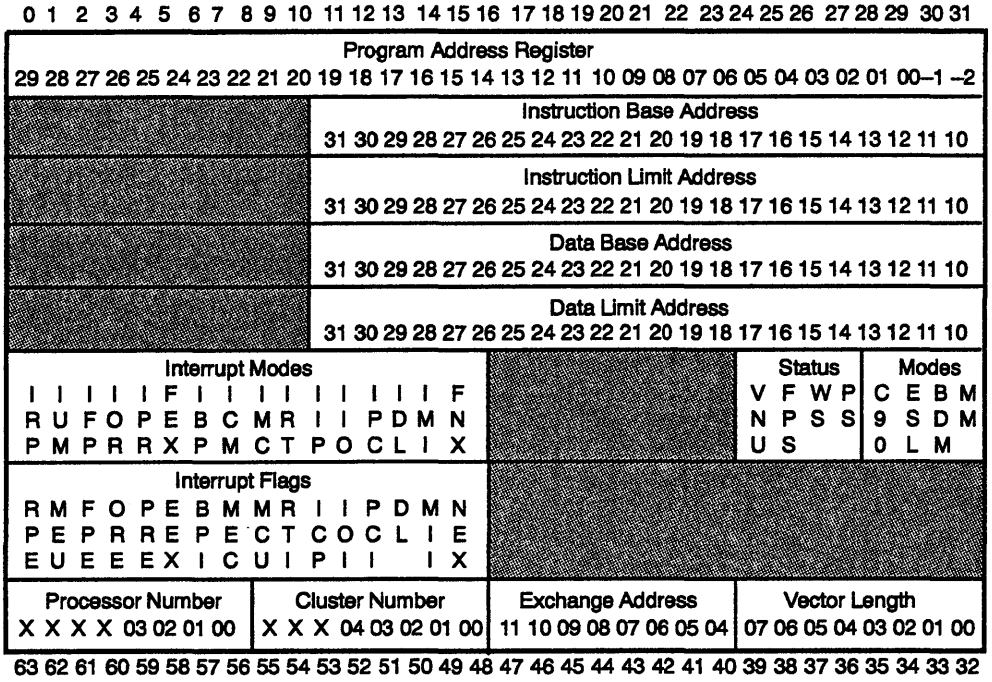
The program address (P) register contents are stored in the program address register field of the exchange package. There are 32 bits in the P register, the lower 2 of which are used to select a particular 16-bit parcel of a memory word. The P register is wide enough to address 1 gigaword of memory.

The address stored in the P register field is that of the first instruction that issues when the program corresponding to this exchange package begins execution.

Instruction Base Address Register Field

The instruction base address (IBA) register holds the base address of the user's instruction area (the location in memory where a program's instruction area begins). The absolute memory address for an instruction fetch sequence is formed by adding the contents of the IBA register to the 30 high-order bits of the contents of the P register.

The IBA register field stores bits 2^{10} through 2^{31} of the IBA; bits 2^0 through 2^9 are always 0. Therefore, the IBA is always a multiple of 2000_8 ($1{,}024_{10}$).



Interrupt Modes (Enabled by EIM Flag)	
IRP	Interrupt on Register Parity Error
IUM	Interrupt on Uncorrectable Memory Error
IFP	Interrupt on Floating-point Error (002100, 002200)
IOR	Interrupt on Operand Range Error (002300, 002400)
IPR	Interrupt on Program Range Error (Does not disable exchange)
FEX	Enable Flag on Error Exit (Does not disable exchange)
IBP	Interrupt on Breakpoint (002340, 002440)
ICM	Interrupt on Correctable Memory Error
IMC	Interrupt on MCU Interrupt
IRT	Interrupt on Real-time Interrupt
IIP	Interrupt on Inter-processor Interrupt
IIO	Interrupt on I/O
IPC	Interrupt on Programmable Clock (001406, 001407)
IDL	Interrupt on Deadlock
IMI	Interrupt on 001*ijk*;*j*≠0
FNX	Enable Flag on Normal Exit (Does not disable exchange)

Interrupt Flags	
RPE	Register Parity Error
MEU	Memory Error – Uncorrectable
FPE	Floating-point Error
ORE	Operand Range Error
PRE	Program Range Error
EEX	Error Exit (000 Issued)
BPI	Breakpoint Interrupt
MEC	Memory Error – Correctable
MCU	MCU Interrupt
RTI	Real-time Interrupt
ICP	Interrupt from Internal CPU
IOI	I/O Interrupt (If IIO and SIE)
PCI	Programmable Clock Interrupt
DL	Deadlock (If IDL and Not MM)
MII	001*ijk*;*j*≠0 (If IMI and Not MM)
NEX	Normal Exit (004 Issued)

Status	
PS	Program State
WS	Waiting on Semaphore
FPS	Floating-point Status (Cleared by 002100, 002200)
VNU	Vectors Not Used

Modes	
MM	Monitor Mode
BDM	Enable Bidirectional Memory (002500, 002600)
ESL	Enable Second Vector Logical
C90	C90 Mode

IPC must be set in the user exchange package. (Instructions 001406, 001407 only affect monitor mode IPC.)
SIE = System I/O Interrupts Enabled (Cleared on I/O Interrupt, set by 001600.)
EIM = Enable Interrupt Modes.

An exchange to nonmonitor mode sets the EIM flag. An exchange to monitor mode clears the EIM flag. While in MM, 001302 sets EIM, and 001303 clears EIM. After 001303, 13 CPs must elapse to take effect.
The following interrupt modes are not affected by EIM: FNX, FEX, IPR.
The following interrupts are held if EIM is clear: PCI, ICP, RTI, MCU, MEC, BPI, ORE, FPE, MEU, RPE.
PCI and ICP are held pending until cleared by instructions 001405 and 001402, respectively.
If EIM is set, interrupts or held interrupts corresponding to set interrupt modes are allowed; held interrupts, except PCI and ICP, are cleared on any exchange.

A-8849

Figure 3-1. CRAY Y-MP C90 Exchange Package

Instruction Limit Address Register Field

The instruction limit address (ILA) register holds the limit address of the user's instruction area, which is used to determine the highest absolute memory address that can be accessed during an instruction fetch sequence.

The absolute memory address used in an instruction fetch sequence must be an address between the IBA and ILA specified for the program being executed, or a program range error occurs. If the interrupt-on-program-range-error (IPR) mode is set in the exchange package, this error sets the program-range-error (PRE) interrupt flag, causing a CPU interrupt.

The ILA register field stores bits 2^{10} through 2^{31} of the ILA; bits 2^{0} through 2^{9} are always 0. Therefore, the ILA is always a multiple of 2000_8 ($1{,}024_{10}$). The highest absolute instruction address of a program is defined by $[(\text{ILA}) \times 2^{10}] - 1$.

Data Base Address Register Field

The data base address (DBA) register holds the base address of the user's data area (the location in memory where a program's data area begins). Each time an instruction in the program makes a memory reference, the memory address generated by the instruction is added to the DBA to form the absolute memory address.

The DBA register field stores bits 2^{10} through 2^{31} of the DBA; bits 2^{0} through 2^{9} are always 0. Therefore, the DBA is always a multiple of 2000_8 ($1{,}024_{10}$).

Data Limit Address Register Field

The data limit address (DLA) register holds the limit address of the user's data area, which is used to determine the highest absolute memory address the program can use for reading or writing data.

Each time an instruction makes a memory reference, the absolute memory address generated is compared to the DBA and the DLA. The absolute memory address must be between the DBA and the DLA, or an operand range error occurs. If the interrupt-on-operand-range-error (IOR) mode is set in the exchange package, this error sets the operand-range-error (ORE) interrupt flag, causing a CPU interrupt.

An instruction that attempts to read from a memory address beyond the DLA still issues and completes, but a zero value is transferred from memory. An instruction that attempts to write to a memory address beyond the DLA issues, but no write operation occurs.

The DLA register field stores bits 2^{10} through 2^{31} of the DLA; bits 2^0 through 2^9 are always 0. Therefore, the DLA is always a multiple of 2000_8 ($1{,}024_{10}$). The highest absolute memory address that can be referenced for data by a program is defined by [(DLA) x 2^{10}] – 1.

Interrupt Modes Field

There are 16 user-selectable interrupt modes, which allow the programmer to select the conditions under which the active program can be interrupted. These modes are usually selected in the exchange package, and except for IPR, FEX, and FNX, they must be enabled by setting the EIM (enable interrupt modes) flag. The EIM flag sets automatically on an exchange to nonmonitor mode and clears on an exchange back to monitor mode. While in monitor mode, the EIM flag can be set or cleared by instructions 001302 or 001303, respectively.

The interrupt modes are listed in concise form in Figure 3-1 and are explained briefly in Table 3-1 below.

Table 3-1. CRAY Y-MP C90 Interrupt Modes

Bit Position	Mode	Description
2^0	IRP	Allows an interrupt if a register parity error is detected while data is being read from a register.
2^1	IUM	Allows an interrupt if an uncorrectable memory error is detected while data is being read from memory.
2^2	IFP	Allows an interrupt if a floating-point error occurs. This mode can also be set by instruction 002100 or 073*i*05 (with S*i* bit 2^{50} = 1); it can be cleared by instruction 002200 or 073*i*05 (with S*i* bit 2^{50} = 0).
2^3	IOR	Allows an interrupt if an operand range error occurs. This mode can also be set by instructions 002300 or 073*i*05 (with S*i* bit 2^{49} = 1); it can be cleared by instructions 002400 or 073*i*05 (with S*i* bit 2^{49} = 0).
2^4	IPR	Allows the PRE interrupt flag to set if a program range error occurs. A program range error always causes an exchange, regardless of the IPR state. This mode is not affected by the EIM flag.
2^5	FEX	Allows the EEX interrupt flag to set if an error exit instruction (000000) issues. Issuing an error exit instruction always causes an exchange, regardless of the FEX state. This mode is not affected by the EIM flag.

Table 3-1. CRAY Y-MP C90 Interrupt Modes (continued)

Bit Position	Mode	Description
2^6	IBP	Allows an interrupt if a breakpoint occurs. This mode can also be set by instruction 002301 or 073*i*05 (with S*i* bit 2^{52} = 1); it can be cleared by instruction 002401 or 073*i*05 (with S*i* bit 2^{52} = 0).
2^7	ICM	Allows an interrupt if a correctable memory error is detected while data is being read from memory.
2^8	IMC	Allows an interrupt if one is requested by the MCU.
2^9	IRT	Allows an interrupt if one is requested by the real-time clock.
2^{10}	IIP	Allows an interprocessor interrupt if one is requested by another CPU.
2^{11}	IIO	Allows an I/O interrupt if SIE is set and this CPU is the lowest-numbered CPU with IIO=1 and EIM=1.
2^{12}	IPC	Allows an interrupt if one is requested by the programmable clock. While in monitor mode, this interrupt mode can be set by instruction 001406 or cleared by instruction 001407. Setting or clearing IPC by these instructions is only valid while the program remains in monitor mode. To set IPC in user mode, it must be set in the exchange package.
2^{13}	IDL	Allows an interrupt if a deadlock occurs while the program is not in monitor mode. IDL has no effect in monitor mode.
2^{14}	IMI	Allows an interrupt if a monitor mode instruction (001*ijk*; $j \neq 0$) issues while the program is not in monitor mode. IMI has no effect in monitor mode.
2^{15}	FNX	Allows the NEX interrupt flag to set if a normal exit instruction (004) issues. Issuing a normal exit instruction always causes an exchange, regardless of the FNX state. This mode is not affected by the EIM flag.

Interrupt Flags Field

There are 16 interrupt flags, with one flag corresponding to each of the 16 user-selectable interrupt modes. If a particular interrupt mode (except IPR, FEX, or FNX) is set and enabled and the specified error occurs, the corresponding interrupt flag is set, forcing an exchange. If the error occurs while the appropriate interrupt mode is set but not enabled, the interrupt is held. This condition can occur only while the program is in monitor mode. Enabling the interrupt modes, either by exchanging to user mode or by issuing instruction 001302, allows the held interrupt to be processed, at which time it sets the corresponding interrupt flag and force an exchange.

All interrupts or held interrupts except PCI and ICP are cleared on any exchange. PCI and ICP interrupts are held until they are cleared by instruction 001405 or 001402, respectively.

Two interrupt flags, deadlock (DL) and monitor instruction interrupt (MII), set only if the corresponding interrupt modes are set and if the program is in nonmonitor mode when the error occurs.

The I/O interrupt (IOI) flag sets only if the system I/O interrupts enabled (SIE) flag is set and if the CPU to be interrupted is the lowest-numbered CPU with IIO interrupt mode set and enabled. The SIE flag can be set by any CPU issuing instruction 001600. After any CPU is interrupted by an I/O interrupt, this flag is cleared, disabling all I/O interrupts. The interrupted CPU should reset the SIE flag by issuing instruction 001600 after it has serviced the I/O interrupt.

There are three errors that always cause an exchange, regardless of the status of the EIM flag: a program range error, issuing instruction 000000, or issuing instruction 004000. The interrupt modes specifying these errors (IPR, FEX, and FNX) are used solely to enable the corresponding interrupt flags (PRE, EEX, and NEX, respectively) to set should the appropriate error occur. Setting an interrupt flag in these cases makes it easier to track down the source of the error.

The errors causing interrupt flags to set are explained briefly in Table 3-2.

Table 3-2. CRAY Y-MP C90 Interrupt Flags

Bit Position	Flag	Description
2^0	RPE	The register parity error flag sets if IRP interrupt mode is set and enabled and if a parity error occurs during a read operation from a B, T, V, SB, or ST register or from an instruction buffer.
2^1	MEU	The memory error – uncorrectable flag sets if IUM interrupt mode is set and enabled and if an uncorrectable memory error occurs while data is being read from memory.
2^2	FPE	The floating-point error flag sets if IFP interrupt mode is set and enabled and if a floating-point range error occurs in any of the floating-point functional units.
2^3	ORE	The operand range error flag sets if IOR interrupt mode is set and enabled and if a data reference is made outside the address boundaries specified in the DBA and DLA registers.

Table 3-2. CRAY Y-MP C90 Interrupt Flags (continued)

Bit Position	Flag	Description
2^4	PRE	The program range error flag sets if IPR interrupt mode is set and enabled and if an instruction fetch is made outside the address boundaries specified in the IBA and ILA registers. A program range error always causes an exchange, regardless of the IPR state.
2^5	EEX	The error exit flag sets if FEX interrupt mode is set and enabled and if an error exit instruction (000000) issues. Issuing an error exit instruction always causes an exchange, regardless of the FEX state.
2^6	BPI	The breakpoint interrupt flag sets if IBP interrupt mode is set and enabled and if a write reference is made to an address within the breakpoint range.
2^7	MEC	The memory error – correctable flag sets if ICM interrupt mode is set and enabled and if a correctable memory error occurs while data is being read from memory.
2^8	MCU	The MCU interrupt flag sets if IMC interrupt mode is set and enabled and if the MCU interrupt signal becomes active on I/O channel 40.
2^9	RTI	The real-time interrupt flag sets if IRT interrupt mode is set and enabled and if a real-time interrupt request is received.
2^{10}	ICP	The interrupt from internal CPU flag sets if IIP interrupt mode is set and enabled and if another CPU requests an interrupt of this CPU by issuing instruction 0014*j*1.
2^{11}	IOI	The I/O interrupt flag sets if SIE is set and this CPU is the lowest-numbered CPU with IIO interrupt mode set and enabled when a LOSP or VHISP channel completes a transfer.
2^{12}	PCI	The programmable clock interrupt flag sets if IPC interrupt mode is set and enabled and if the counter in the programmable clock equals 0.
2^{13}	DL	The deadlock interrupt flag sets if IDL interrupt mode is set, if the program is not in monitor mode, and if a deadlock occurs because all CPUs in a cluster are holding issue on a test and set instruction.
2^{14}	MII	The monitor instruction interrupt flag sets if IMI interrupt mode is set and if a monitor mode instruction (001*ijk*; $j \neq 0$) issues while the program is not in monitor mode.
2^{15}	NEX	The normal exit flag sets if FNX interrupt mode is set and enabled and if a normal exit instruction (004000) issues. Issuing a normal exit instruction always causes an exchange, regardless of the FNX state.

Status Field

The status field contains 4 bits used to indicate the state of the CPU at the time an exchange occurs. These status bits are set during program execution and therefore are not user-selectable. Table 3-3 briefly describes each of the status bits used.

Table 3-3. CRAY Y-MP C90 Status Field Bit Assignments

Bit Position	Status	Description
2^{24}	VNU	The vectors not used bit sets if no vector instructions (077*ijk* or 140*ijk* through 177*ijk*) were issued during the execution interval.
2^{25}	FPS	The floating-point status bit sets if a floating-point error occurred during the execution interval. This bit can also be set by instruction 073*i*05 (with S*i* bit 2^{51} = 1); it can be cleared by instruction 002100, 002200, or 073*i*05 (with S*i* bit 2^{51} = 0). It can also be read to an S register by instruction 073*i*01.
2^{26}	WS	The waiting on semaphore bit sets if a test and set instruction (0034*jk*) is holding issue in the CIP register.
2^{27}	PS	The program state bit is set by the operating system to denote whether a CPU concurrently processing a program with another CPU is the master or slave in a multitasking situation.

Modes Field

There are four user-selectable operating modes. These modes are described briefly in Table 3-4.

Processor Number Field

The contents of the 4-bit processor number field indicate the logical number of the CPU that performed the exchange sequence. This value is not initially stored in the exchange package before the program starts; it is a constant value inserted into the exchange package after the program runs and exchanges out.

Table 3-4. CRAY Y-MP C90 Operating Modes

Bit Position	Mode	Description
2^{28}	C90	If C90 mode is set, the program can use the full CRAY Y-MP C90 instruction set; otherwise, only CRAY Y-MP instructions can be executed.
2^{29}	ESL	If enable second vector logical mode is set, the second vector logical functional unit is enabled, and if it is not busy, it has first priority to execute instructions 140*ijk* through 145*ijk*.
2^{30}	BDM	If bidirectional memory mode is set, block read and write operations can operate concurrently. BDM mode can also be set by instruction 002500 or 073*i*05 (with S*i* bit 2^{48} = 1); it can be cleared by instruction 002600 or 073*i*05 (with S*i* bit 2^{48} = 0). This bit can also be read to an S register by instruction 073*i*01.
2^{31}	MM	If monitor mode is set, the program can execute instructions privileged to monitor mode.

Cluster Number Field

The 5-bit cluster number (CLN) field contains the number to be loaded into the CLN register. This number selects one of 17_{10} available clusters of shared registers the CPU can access. If the contents of the CLN register are 0, the CPU does not have access to any shared registers. The contents of the CLN registers in all CPUs are also used to determine a deadlock interrupt condition.

Exchange Address Register Field

The 8-bit exchange address (XA) register field specifies the address of the first word of a 16-word exchange package loaded by an exchange sequence. The XA register contains only the 8 high-order bits of a 12-bit absolute memory address. The low-order bits of the address are always 0, since an exchange package must begin on a 16-word boundary. The 12-bit limit on the absolute memory address means that all exchange packages are located in the lower 4,096 (10000_8) words of memory.

Vector Length Register Field

The 8-bit vector length (VL) register field specifies the length of all vector operations performed by vector instructions and the number of elements held in the V registers. The value in the VL register can be changed during program execution by using instruction 00200*k*.

A Register Fields

The contents of all A registers are stored in bit positions 2^{32} through 2^{63} of words 0 through 7 during an exchange sequence.

S Register Fields

The contents of all S registers are stored in bit positions 2^{0} through 2^{63} of words 8 through 15 during an exchange sequence.

Exchange Sequence

The exchange sequence moves the contents of an inactive exchange package from memory into the operating registers. Simultaneously, the exchange sequence retrieves data from the operating registers, uses it to construct the active exchange package, and then moves this exchange package back into memory. This swapping operation occurs in a fixed sequence when all computational activity associated with the active exchange package stops.

The exchange sequence involves 16 memory read references and 16 memory write references. A single 16-word block of memory is used as the source of the inactive exchange package and the destination of the active exchange package. Word 0 of the active exchange package is swapped with word 0 of the inactive exchange package. The location of this block of data is specified by the contents of the XA register and is a part of the active exchange package.

Exchange Sequence Timing

The following subsections define the hold conditions, execution time, and special case conditions for an exchange sequence.

Hold Conditions

The following conditions can delay the start of an exchange sequence:

- Incomplete memory references.
- Any active A, S, or V registers within the CPU.

Execution Time

An exchange takes a minimum total of 62 CPs: 35 CPs for the exchange sequence and 27 CPs for a fetch operation. This time applies if there are no memory conflicts. Memory conflicts are possible during both the exchange sequence and the fetch operation.

Special Case Conditions

If a test and set instruction is holding in the CIP register, both the CIP register and the next instruction parcel (NIP) register are cleared. The exchange occurs with the waiting on semaphore (WS) status bit set and the P register pointing to the address of the test and set instruction.

Initiating an Exchange Sequence

An exchange sequence can be initiated by a deadstart sequence, a program exit, or an interrupt. The following subsections describe the conditions that cause an exchange sequence.

Deadstart Sequence

The deadstart sequence starts a program running in the mainframe after a power-off/power-on operation or whenever the operating system is reinitialized in the mainframe. Consider all control latches, words in memory, and the contents of all registers invalid after a power-off/power-on operation.

There are two ways to deadstart the mainframe: through an external device connected to I/O channel 40 or through the maintenance channel. With the first method, the external device used is usually an IOS or the maintenance workstation (MWS). The external device deadstarts one CPU in the mainframe with the following sequence:

1. Activates the CPU Master Clear signal.
2. Activates the I/O Master Clear signal.
3. Deactivates the I/O Master Clear signal.
4. Loads 20000_8 words of data into memory through I/O channel 40.
5. Deactivates the CPU Master Clear signal.

The CPU Master Clear signal halts all internal computation and forces critical control latches to predetermined states. The I/O Master Clear signal clears the input channel address (CA) register of I/O channel 40 and activates the channel; all other I/O channels remain inactive. The external device then loads a deadstart exchange package and monitor

program. Because the CPU Master Clear signal forces the contents of the XA register to 0, the deadstart exchange package must be located at memory address 0.

Deactivating the Master Clear signal initiates the exchange sequence and starts the monitor program running in logical processor 0 (LPN 0). A switch on the mainframe control panel determines which CPU is assigned as LPN 0.

Because the mainframe's operating registers at the time of the deadstart sequence contain indeterminate data, the initial exchange sequence causes indeterminate data to be written into the exchange package at memory address 0. The monitor program must rebuild this exchange package with accurate data in preparation for deadstarting subsequent CPUs with an interprocessor interrupt. LPN 0 can then issue instruction 0014*j*1, which causes an interrupt and subsequent deadstart exchange in another CPU (LPN 1). LPN 0's monitor program again rebuilds the deadstart exchange package and issues an interrupt to LPN 2. This sequence continues until all CPUs are deadstarted.

The maintenance channel can also be used to deadstart one CPU in the mainframe by transmitting data over the channel in the following sequence:

1. Sends function code 004500.
2. Sends the deadstart exchange package and the monitor program.
3. Sends function code 005000.

Function code 004500 activates the Master Clear signal, halting all internal computation and clearing key registers. The deadstart exchange package is then written to memory address 0, and the monitor program is loaded elsewhere in memory. Function code 005000 then deadstarts the CPU specified in its accompanying ID code.

For one CPU to deadstart the other CPUs through the maintenance channel, the previous sequence must be modified as follows:

1. Sends function code 004500.
2. Sends the deadstart exchange package and the monitor program.
3. Sends function code 004000.
4. Sends function code 005000.

Function code 004000 deactivates the Master Clear signal. Then, after the first CPU is deadstarted, its monitor program can rebuild the deadstart exchange package and issue instruction 0014*j*1 to deadstart the other CPUs in the same manner as explained above.

The exchange package for each program resides in an area defined during system deadstart. The defined area must be located in the lower 4,096 (10000_8) words of memory. The exchange package at memory address 0 is the deadstart monitor's exchange package. Only the monitor program has a data area defined so that it can access all of memory, including exchange package areas. This capability allows the monitor program to define or alter all exchange packages other than its own when it is active. Other exchange packages provide for object programs and other monitor tasks and are located outside of the program's instruction and data areas.

Program Exit Instructions

There are two program exit instructions that initiate an exchange sequence: the error exit (000000) and the normal exit (004000). These two instructions enable a program to request its own termination. A program usually uses the normal exit instruction to exchange back to the monitor program after it has finished its programmed task. The error exit instruction allows for termination of an object program if an error condition occurs; the exchange address selected is the same as for a normal exit instruction.

Issuing either of these two instructions may also cause an interrupt flag to set, depending on whether the appropriate interrupt mode is set and enabled. These interrupt modes and flags were explained earlier in this section.

Interrupts

An exchange sequence can also be initiated by setting any of the interrupt flags described earlier in this section or by a program range error.

Exchange Package Management

Exchange package management dictates that a user program should always exchange back to the monitor that caused the user program to start executing. This exchange back to the monitor ensures that the program information is always exchanged into its proper exchange package. The following paragraphs illustrate the dynamics of exchange package management.

A monitor begins an execution interval following a deadstart sequence. Assuming the interrupt modes are not enabled, no interrupts (except a program range error or instructions 000000 or 004000) can terminate its execution interval because it is in monitor mode. Before the monitor program exits, the monitor sets the contents of the XA register to point to

a user program's exchange package so that a user program runs next. The monitor then voluntarily exits by issuing a normal exit instruction (004000).

The exchange sequence moves the inactive exchange package (in this case, the user program's) from memory into the operating registers. At the same time, the exchange sequence retrieves data from the operating registers, uses it to construct the active exchange package (in this case, the monitor's), and then moves this exchange package back into memory at the location previously occupied by the user program's exchange package. The XA register in the user program's exchange package contains its previous storage address, which is now the address of the monitor's exchange package. When the exchange is complete, the user program begins to run.

If an interrupt occurs while the user program is running, an exchange sequence is initiated. Because the contents of the XA register in the user's program exchange package is pointing to the monitor, the exchange is back to the monitor. (Note that a user program cannot alter the contents of the XA register.)

When the exchange back to the monitor is complete, the monitor determines which interrupt caused the exchange and sets the contents of the XA register to call the proper interrupt-processing program to run. To do this, the monitor sets the XA register to point to the exchange package for the relevant interrupt processing program. The monitor then clears the interrupt and executes a normal exit (004000) instruction causing the interrupt-processing program to run. Depending on the operating task, the interrupt-processing program can run in monitor mode or user mode.

NOTE: There is no interlock between an exchange sequence in a CPU and memory transfers in another CPU; therefore, avoid modifying exchange packages used by other CPUs except under software-controlled situations.

Instruction Fetch Sequence

An instruction fetch sequence retrieves program code from memory and places it in an instruction buffer. The program code is held in the instruction buffer before being delivered to the instruction issue registers. The following subsections describe the hardware associated with the instruction fetch sequence and define the fetch operation.

Instruction Fetch Hardware

The CRAY Y-MP C90 computer system uses the P register to initiate an instruction fetch sequence and uses eight instruction buffers to store the instructions retrieved from central memory. Figure 3-2 shows the P register and instruction buffers.

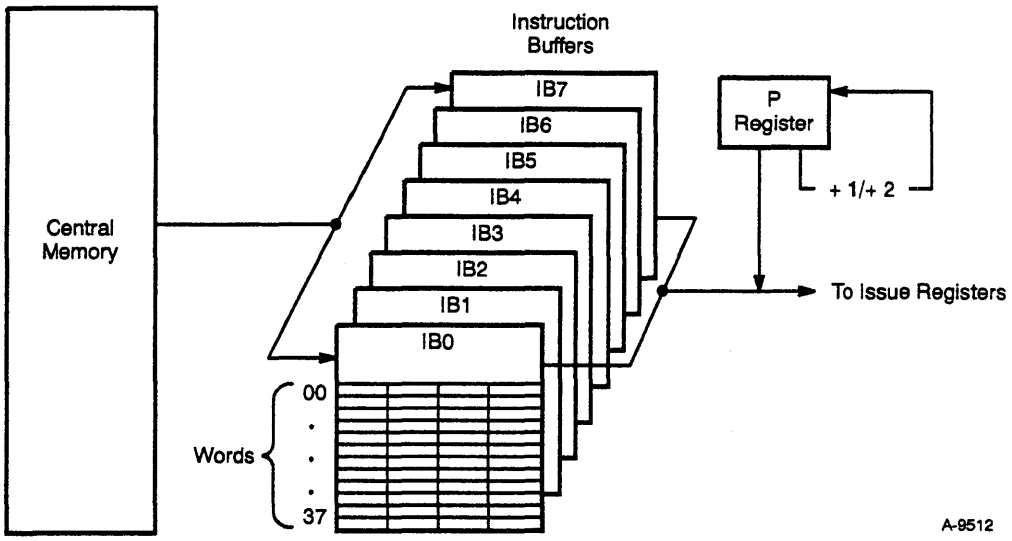


Figure 3-2. Instruction Fetch Block Diagram

Instruction Buffers

Each of the eight instruction buffers (IB0 through IB7) holds 32 (40_8) words, numbered 00_8 through 37_8. Each word contains four 16-bit instruction parcels; therefore, each buffer holds 128 parcels. Instruction parcels are held in the buffers before being delivered to the issue registers.

The first instruction parcel in a buffer always has a memory word address that is a multiple of 40_8. This word address allows the entire area of addresses for instructions in a buffer to be defined by the 25 high-order bits of the P register.

Each instruction buffer has an associated instruction buffer address register (IBAR). The IBAR contains the upper 25 bits of the P register and an IBAR valid bit. When set, the IBAR valid bit indicates that the buffer contains valid data. During an exchange sequence, all the IBAR valid bits are cleared to invalidate the previous program's instructions and to force the CPU to fetch new instructions. Once the fetch operation begins, the appropriate IBAR is loaded with the upper 25 bits of the P register, and its valid bit is set. Figure 3-3 shows the IBAR register.

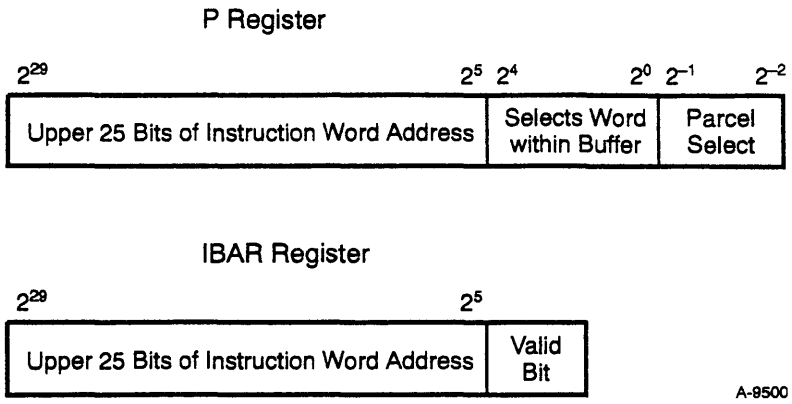


Figure 3-3. P Register and IBAR Register Address Formats

The instruction buffers are protected with parity bits. When a word is written into an instruction buffer, 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 instruction buffer. A parity error is indicated when the two sets do not match and is reported to status register 7 (SR7). The status registers are described later in this section.

Program Address Register

The 32-bit P register indicates the next parcel of program code to enter the NIP register. As shown in Figure 3-3, the 25 high-order bits of the P register indicate the word address of the program code in memory relative to the base address. The 2 low-order bits indicate the parcel within the word. Because only 25 bits specify the word address, the maximum program length is 32 Mwords.

Under normal circumstances, the P register increments sequentially as instructions issue. For 1- and 2-parcel instructions, the P register increments by one; for 3-parcel instructions, the P register increments by two. These increments allow both 2- and 3-parcel instructions to issue in 2 clock periods (CPs). Branch instructions can load the P register with any value. When the program exchanges out, the saved P register contains the address of the instruction immediately following the last instruction that executed.

Instruction Fetch Operation

An instruction fetch operation refers to the series of steps performed to retrieve program code from memory to an instruction buffer.

The fetch operation is initiated after a comparison check of the P register against the values held in the eight IBAR registers; this comparison is made each CP. The P register always contains the parcel address of the next instruction to be decoded. If the contents of one of the IBAR registers is equal to the upper 25 bits in the P register, and the IBAR valid bit is set, an in-buffer (or coincidence) condition exists. In this case, the next instruction to be decoded is already contained in an instruction buffer, and no fetch sequence is needed. If the 25 high-order bits of the P register do not match the contents of any IBAR, or the valid bit is not set, an out-of-buffer (or no-coincidence) condition exists, and the instruction fetch sequence starts.

The instruction buffers are filled circularly, one at a time. Each buffer stores 128 parcels of instruction code. Each time a no-coincidence condition occurs, a fetch sequence is initiated and another instruction buffer is filled. This process continues until all eight buffers are filled. If the program code exceeds 1,024 parcels, the ninth fetch reloads the first instruction buffer.

The instruction fetch sequence uses memory ports D and D' to transfer 32 words (128 parcels) from memory into the instruction buffer (refer to "Logical Organization" in Section 2 for more information on memory ports). Two words are transferred each CP.

One of the first two words delivered to the instruction buffer always contains the next instruction required for execution. For example, if the P register contains the address 124-2 (parcel 2 of word 124) when the fetch operation begins, the first two words delivered to the instruction buffer are from memory addresses 124 and 125. During each succeeding CP, two additional words arrive at the instruction buffer, filling the buffer in order with words 124 through 137 and then words 100 through 123.

Once the instruction buffers are loaded, or if the comparison between the P and IBAR registers produced a coincidence condition, the proper instruction parcel is selected from the instruction buffer. The instruction parcel is sent to the NIP register and then to the CIP register, from which the instruction issues. Instruction issue is explained later in this section.

Although optimizing code segment lengths for instruction buffers is not a prime consideration when programming a CPU, the number and size of the buffers and the capability for forward and backward branching can be used to minimize fetches. Large loops containing up to 1,024 consecutive instruction parcels can be maintained in the eight buffers. An alternative way to handle a large loop is for a main program sequence in one or two of the buffers to make repeated calls to short subroutines maintained in the other buffers. The program and subroutines remain undisturbed in the buffers as long as no out-of-buffer condition or exchange causes reloading of a buffer.

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

Because instructions are held in instruction buffers before issue and until the buffer is reloaded, self-modifying code should not be used. Also, because of independent data and instruction memory protection, self-modifying code may be impossible. As long as the address of the unmodified instruction is in an instruction buffer, the modified instruction in memory is not loaded into an instruction buffer.

Instruction Fetch Timing

During an instruction fetch sequence, instructions are retrieved from memory to an instruction buffer at the rate of 2 words per CP. It takes 25 CPs for the first word to arrive at the instruction buffer and an additional 3 CPs for the first instruction to get to the CIP register. Instruction issue can run concurrently with the fetch operation as long as the required instruction parcel is in the instruction buffer. If no memory conflicts are encountered, the instruction buffer is filled in 40 CPs (25 CPs for the first two words and 15 CPs for the remaining 30 words). Memory conflicts can lengthen the fetch sequence timing.

Instruction Issue

An instruction issue sequence is the series of steps performed to get an instruction from an instruction buffer through the issue registers and into execution.

Instruction Issue Hardware

The CRAY Y-MP C90 computer systems use four different registers to issue instructions. Figure 3-4 shows the registers and buffers and the general flow of the instruction parcels through them. CPU instructions are 1-, 2-, or 3-parcel instructions; refer to "Instruction Formats" in Section 7 for information on instruction parcels.

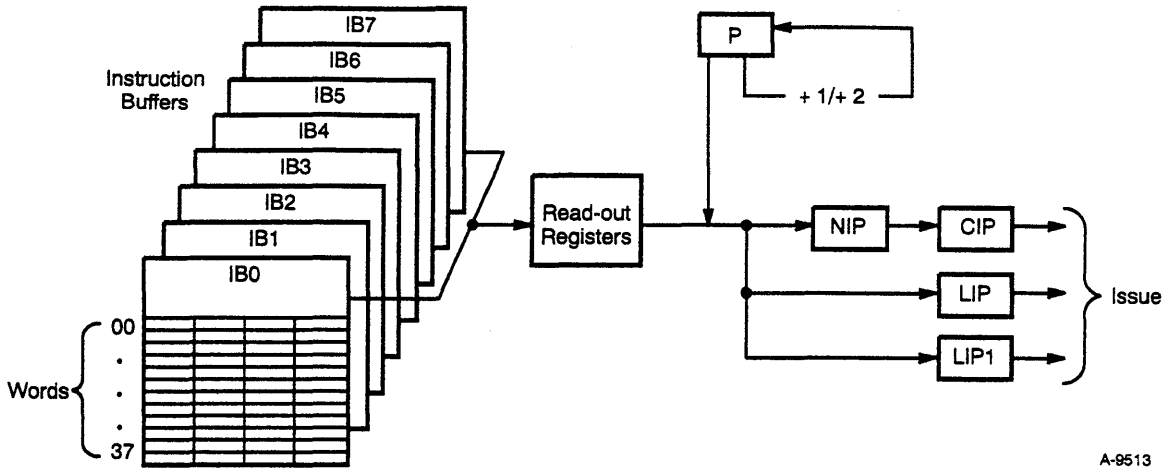


Figure 3-4. Instruction Issue Block Diagram

Instruction Buffers

The instruction buffers hold the program code after it is retrieved from memory and before it is passed on to the issue registers. The instruction buffers have two associated read-out registers to streamline the flow of instructions from the buffers to the NIP register. Even-numbered words are loaded into the even read-out register, while odd-numbered words are loaded into the odd read-out register. Bit 2^0 of the P register determines which read-out register is used, and bits 2^{-1} and 2^{-2} of the P register select the parcel to be sent to the NIP register.

Program Address Register

The 32-bit P register indicates the next parcel of program code to enter the NIP register. The 25 high-order bits of the P register indicate the word address for the program code in memory relative to the base address. The 2 low-order bits indicate the parcel within the word. Under normal circumstances, the P register increments sequentially as instructions issue. For 1- and 2-parcel instructions, the P register increments by one; for 3-parcel instructions, it increments by two. This allows both 2- and 3-parcel instructions to issue in 2 CPs. Branch instructions and exchange sequences can load the P register with any value.

Next Instruction Parcel Register

The 16-bit NIP register receives an instruction parcel from one of the instruction buffer read-out registers. While the parcel of program code is held in the NIP register, it is decoded to determine whether the instruction is a 1-, 2-, or 3-parcel instruction. The parcel is then passed on to the CIP register.

The NIP register cannot be master cleared. An undetermined instruction can issue during the master clear sequence before an interrupt condition blocks data entry into the NIP register.

Current Instruction Parcel Register

The 16-bit CIP register receives the parcel of program code from the NIP register and holds the instruction until it issues. Issue of an instruction held in the CIP register can be delayed until conflicting operations are completed (refer to "Reservations and Hold Issue Conditions" later in this section).

The issue control associated with the CIP register can be master cleared; the register itself cannot. An undetermined instruction can issue during the master clear sequence.

Lower Instruction Parcel and Lower Instruction Parcel 1 Registers

The 16-bit lower instruction parcel (LIP) register holds the second parcel of a 2-parcel instruction (the first parcel of this instruction is always held in the CIP register). The 16-bit LIP1 register holds the third parcel of a 3-parcel instruction (again, the first parcel is held in the CIP register, while the second parcel of this instruction is held in the LIP register).

Instruction Issue Operation

Control logic associated with the NIP register determines whether the instruction is a 1-, 2-, or 3-parcel instruction and steers subsequent parcels to the correct register. The general sequences for the three types of instructions are described in the following paragraphs; specific examples of 1-, 2-, and 3-parcel instructions are given on the following pages.

For 1-parcel instructions, the P register sends the instruction parcel to the NIP register. From the NIP register, the instruction moves to the CIP register. If there are no conflicts, the instruction executes.

For a 2-parcel instruction, the P register sends the first parcel to the NIP register. Then the first parcel is sent to the CIP register, while the second parcel goes directly to the LIP register. When the two registers are properly loaded with the correct parcels, and there are no conflicts, the first parcel issues from the CIP register and the second parcel issues from the LIP register at the same time. When the parcels of the 2-parcel instruction move from the CIP and LIP registers to execution, the NIP register sends a blank parcel to the CIP register. The control logic decodes this blank as a no operation instruction when it issues from the CIP register. While this blank parcel is loaded into the CIP register, a new parcel is loaded into the NIP register and the control logic determines if the instruction is a multi-parcel instruction. During this sequence, a delay can occur if the new instruction is in a different buffer than the previous instruction or if a fetch operation is required.

For a 3-parcel instruction, the P register sends the first parcel to the NIP register. Then the first parcel is sent to the CIP register, while the second parcel goes directly to the LIP register and the third parcel goes directly to the LIP1 register. When the three registers are loaded with the correct parcels, and there are no conflicts, the first parcel issues from the CIP register, the second parcel issues from the LIP register, and the third parcel issues from the LIP1 register, all at the same time. When the parcels of the 3-parcel instruction move from the CIP and LIP registers to execution, the NIP register sends a blank parcel to the CIP register. The control logic decodes this blank as a no operation instruction when it issues from the CIP register. While this blank parcel is loaded into the CIP register, a new parcel is loaded into the NIP register and the control logic determines if it is a multi-parcel instruction. Delays can occur if the new instruction is in a different buffer than the previous instruction or if a fetch operation is required.

Figure 3-5 through Figure 3-14 and the following paragraphs show the steps that occur as 1-, 2-, and 3-parcel instructions are steered in sequence through the issue registers. The sequence assumes a 1-CP delay for all conflicts and is numbered CPn through CPn+9. An instruction buffer with its two read-out registers, the P register, and the relevant issue registers are shown for each CP.

Figure 3-5 shows parcels 20-0 through 21-3 being held in an instruction buffer and read-out registers. The P register is pointing to parcel 20-0 as the next parcel to be read into the NIP register.

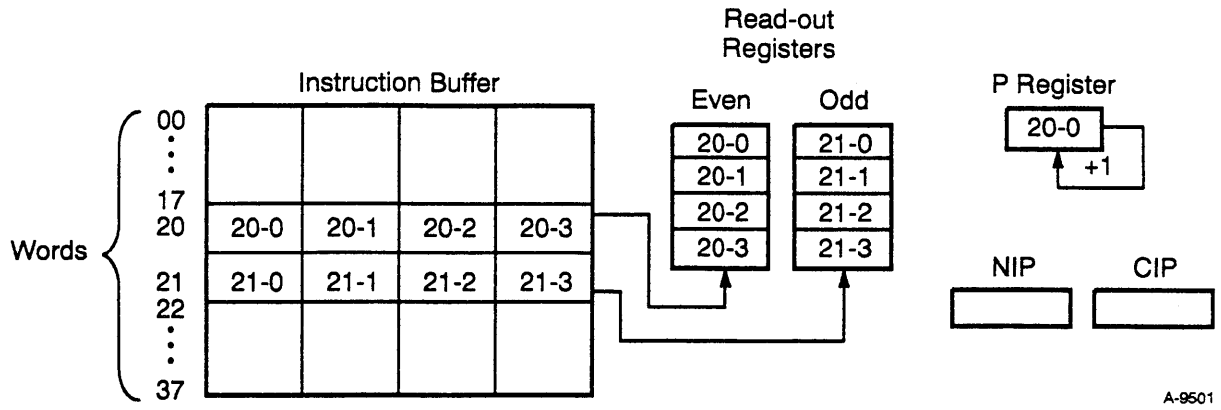


Figure 3-5. Instruction Flow through Issue Registers (CPn)

Figure 3-6 shows parcel 20-0 in the NIP register. The P register incremented by 1 and is pointing to parcel 20-1 to read out as the next parcel. While parcel 20-0 is in the NIP register, hardware determines whether it is a 1-, 2-, or 3-parcel instruction.

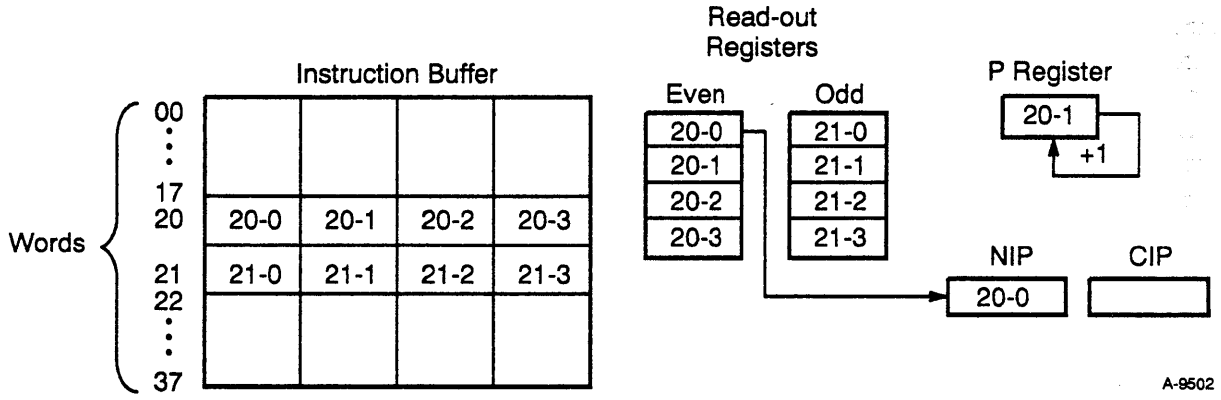


Figure 3-6. Instruction Flow through Issue Registers (CPn + 1)

Since parcel 20-0 is a 1-parcel instruction, the logic steers this parcel into the CIP register and parcel 20-1 into the NIP register. The P register increments by 1 (refer to Figure 3-7).

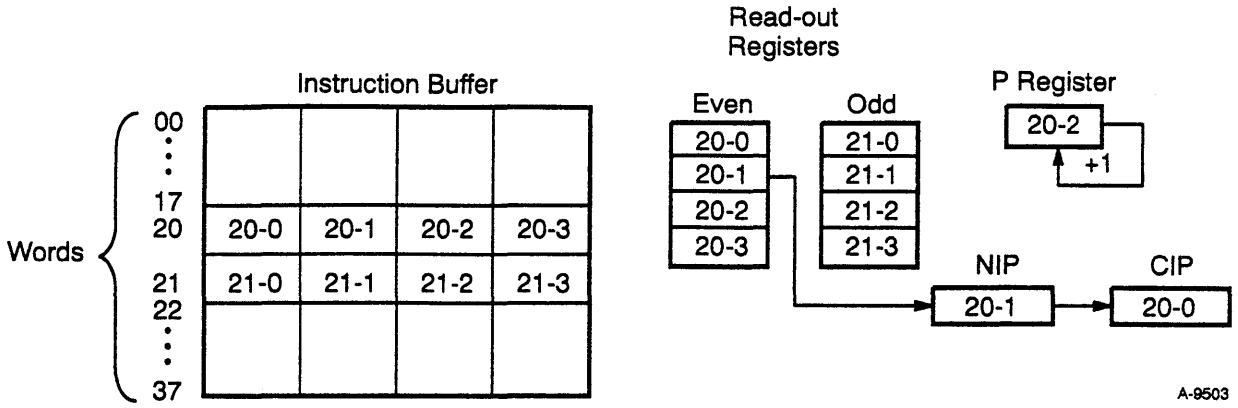


Figure 3-7. Instruction Flow through Issue Registers (CPn + 2)

While the parcel in the NIP register is decoded to determine whether it is a 1-, 2-, or 3-parcel instruction, the issue hardware checks for any conflicts that might prevent the instruction in the CIP register from issuing. If there are conflicts, both the CIP and NIP registers hold their parcels, and the P register does not increment (refer to Figure 3-8).

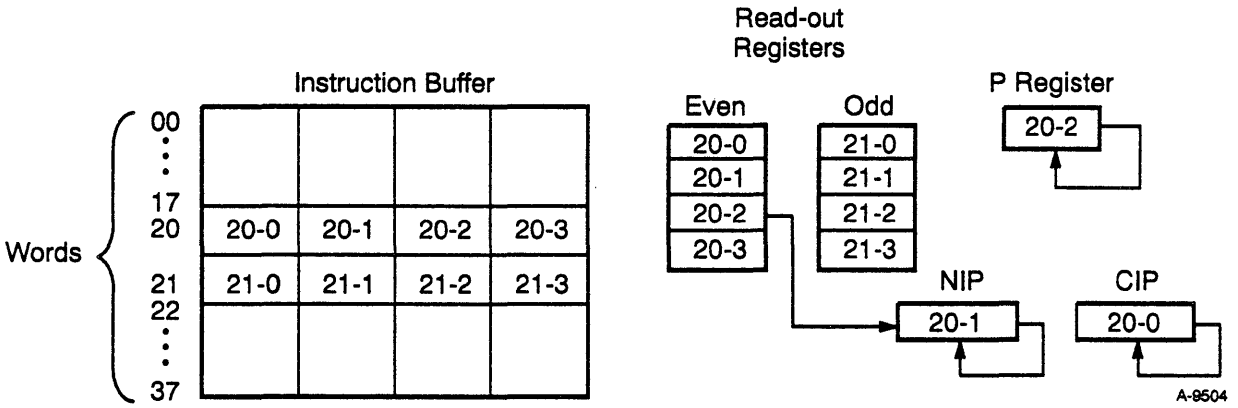


Figure 3-8. 1-parcel Instruction Holding 1 CP for Conflict (CPn + 3)

This holding state is maintained until the conflict is resolved. When the conflict is resolved, or if there are no conflicts, parcel 20-0 issues from the CIP register (refer to Figure 3-9).

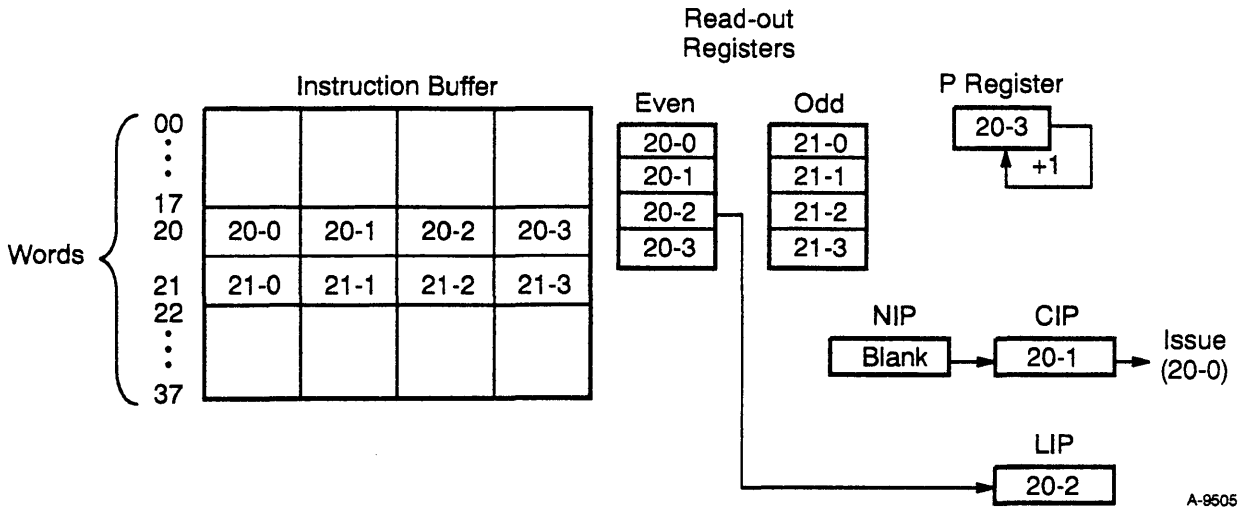


Figure 3-9. Instruction Flow through Issue Registers (CPn + 4)

Because parcel 20-1 is the first parcel of a 2-parcel instruction, the logic steers parcel 20-2 into the LIP register and parcel 20-1 into the CIP register. Also, a blank parcel is generated in the NIP register. The P register increments by 1 to point to the next parcel (in this case, parcel 20-3). Issue hardware checks for conflicts. If any conflicts are found, the CIP, LIP, and NIP registers hold their parcels and the P register does not increment (refer to Figure 3-10).

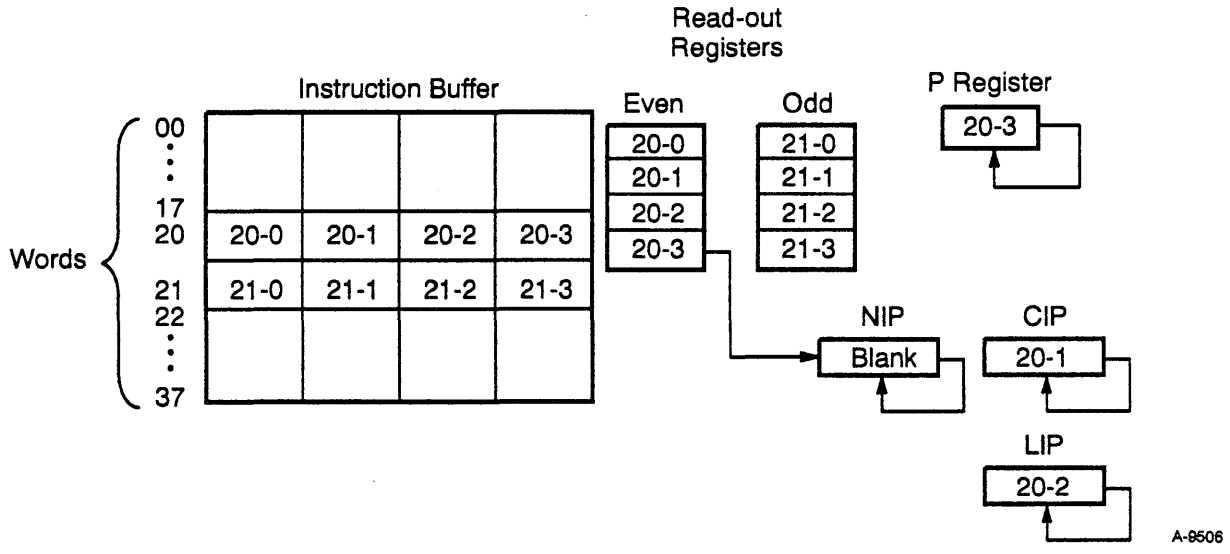


Figure 3-10. 2-parcel Instruction Holding 1 CP for Conflict (CPn + 5)

This holding state is maintained until the conflict is resolved. When the conflict is resolved, or if there are no conflicts, parcels 20-1 and 20-2 issue together in the next CP (refer to Figure 3-11).

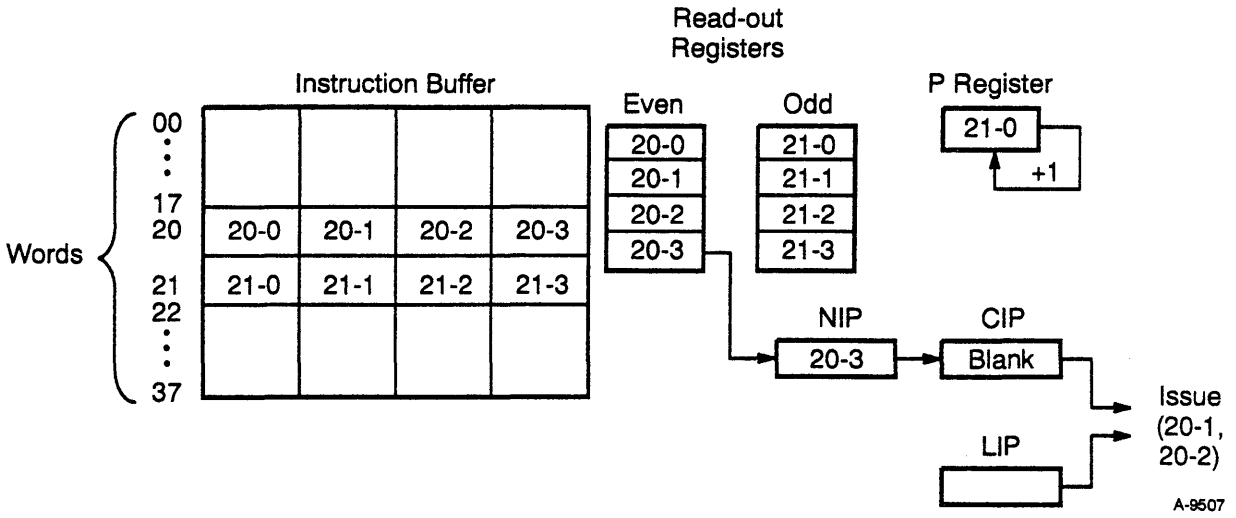


Figure 3-11. Instruction Flow through Issue Registers (CPn + 6)

As the 2 parcels move from the CIP and LIP registers to issue, parcel 20-3 is loaded into the NIP register and a blank parcel is loaded into the CIP register. The P register increments by 1 and points to the next parcel (in this case, parcel 21-0). Since the P register no longer points to a parcel in word 20, a new word is loaded into the even read-out register during the next CP. The blank parcel in the CIP register is decoded as a no operation instruction when it issues during CPn+7 (refer to Figure 3-12).

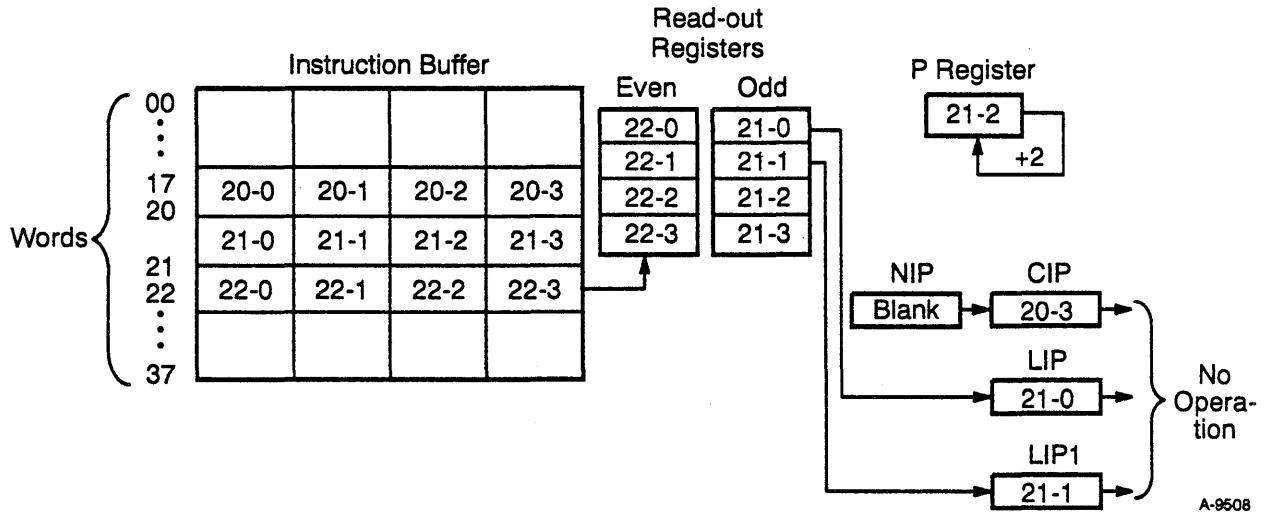


Figure 3-12. Instruction Flow through Issue Registers (CPn + 7)

Because parcel 20-3 is the first parcel of a 3-parcel instruction, the logic steers parcel 21-1 into the LIP1 register, parcel 21-0 into the LIP register, and parcel 20-3 into the CIP register. A blank parcel is generated in the NIP register. The P register increments by 2 and points to the next parcel (in this case, Parcel 21-2). Issue hardware checks for conflicts. If any conflicts are found, and resolved, the issue registers hold their parcels, and the P register does not increment (refer to Figure 3-13).

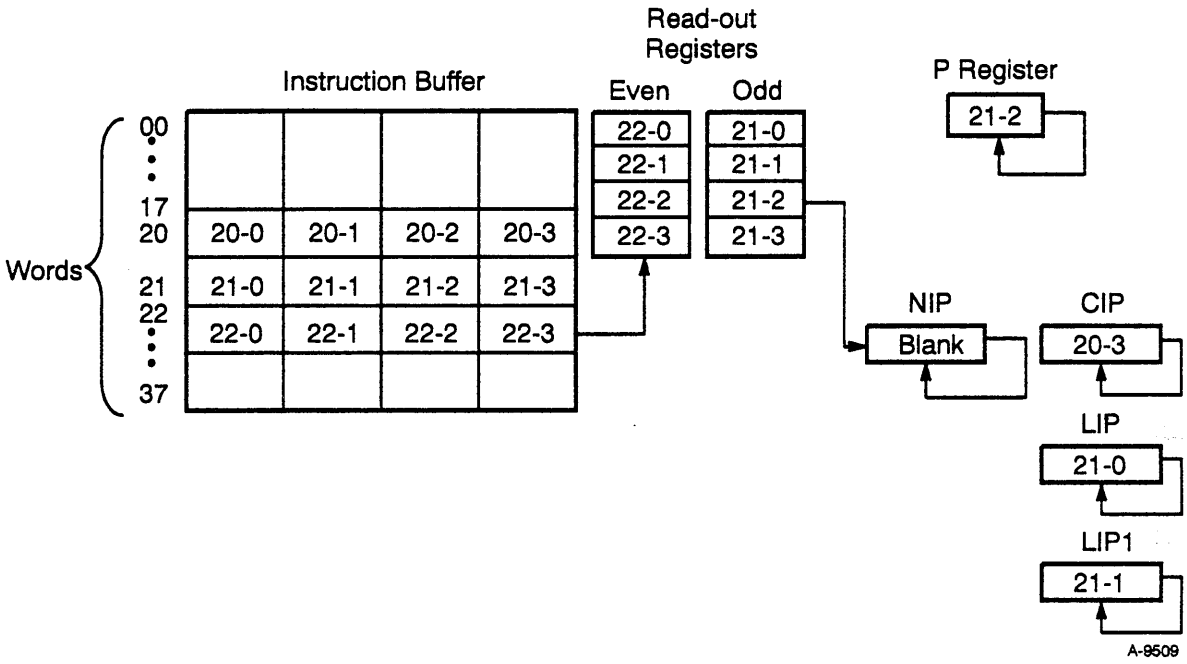


Figure 3-13. 3-parcel Instruction Holding 1 CP for Conflict (CPn + 8)

This holding state is maintained until the conflict is resolved. If there are no conflicts, parcels 20-3, 21-0, and 21-1 issue together in the next CP (refer to Figure 3-14).

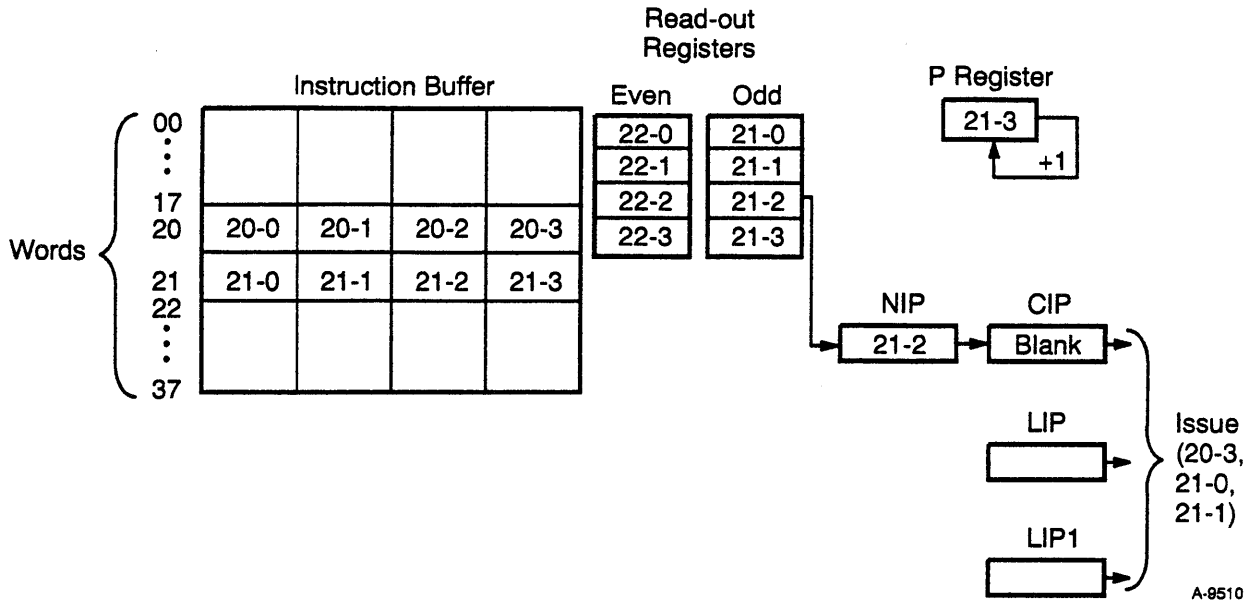


Figure 3-14. Instruction Flow through Issue Registers (CPn + 9)

As the 3 parcels move from the CIP, LIP, and LIP1 registers to execution, parcel 21-2 enters the NIP register, and a blank parcel enters the CIP register. The P register increments by 1 to point to the next parcel (in this case, parcel 21-3).

Instructions continue to flow through the issue registers until the program code exits normally or is interrupted. In either case, an exchange and fetch operation bring new code into the instruction buffers and a new value into the P register, and the issue sequence starts over again.

The issue sequence explained and illustrated in the previous paragraphs is summarized in Table 3-5. This chart shows the movement of the instruction parcels at each CP as they pass through the issue registers.

Table 3-5. Instruction Issue Sequence

Registers	CPn	n + 1	n + 2	n + 3	n + 4	n + 5	n + 6	n + 7	n + 8	n + 9
P	20-0	20-1	20-2	20-2	20-3	20-3	21-0	21-2	21-2	21-3
NIP		20-0	20-1	20-1	Blank	Blank	30-3	Blank	Blank	21-2
CIP			20-0	20-0	20-1	20-1	Blank	20-3	20-3	Blank
LIP					20-2	20-2		21-0	21-0	
LIP1								21-1	21-1	

Reservations and Hold Issue Conditions

When the first parcel of an instruction is in the CIP register, hardware is used to determine if there are any conflicts preventing the instruction from executing. These conflicts are referred to as hold issue conditions and cause the instruction to be held in the issue registers until the conflict is resolved. Once the instruction issues, reservations are immediately placed on the appropriate registers, paths, ports, or functional units as needed. These reservations are usually held until a few CPs before the instruction finishes execution; the exact timing depends on the type of instruction.

Register reservations are placed in the following cases:

- When A and S registers are reserved as result registers but not as operand registers.
- When access to the B or T registers is reserved during block transfers.
- When individual V registers are reserved specifically as either operand or result registers.
- When input paths are reserved for the CP during which the data is expected to enter the A or S registers.

Port reservations are placed when the following conditions occur:

- Port A is reserved for memory reads to the B registers.
- Port B is reserved for memory reads to the T registers.
- Port A or port B is reserved for memory reads to the V registers.
- Port C is reserved for B register, T register, or V register writes to memory.

Functional unit reservations are placed on functional units processing vector operands.

Conflicts also occur when more than one CPU tries to access the shared path at the same time. The shared path is used by all shared and semaphore registers, and by I/O instructions, interprocessor interrupt signals, and the real-time and programmable clocks. Refer to "Shared Paths Access Priority" in Section 2 for more information on the shared path.

For a detailed description of the hold issue conditions for each instruction, refer to "CPU Instruction Descriptions" in Section 7. In several cases, these conditions are limited to a specific instruction or instruction sequence. The following conditions are a few generalized hold issue conditions.

Scalar instructions hold issue if the following conditions occur:

- The A or S register needed for a result is reserved.
- The input path is reserved for the CP during which incoming data enters the register.
- The instruction calls for a floating-point operation, and the floating-point functional unit is reserved.
- The instruction references memory, and port A, B, or C is reserved.

Vector instructions hold issue if the following conditions occur:

- The V register needed for an operand is reserved as an operand.
- The V register needed for a result is reserved as either an operand or a result.
- The functional unit needed is reserved.
- The instruction references memory and the needed port is reserved.

For B and T register block transfers, a hold issue condition exists if the needed port is reserved. For multi-parcel instructions, a hold issue condition exists if the second or third parcel of the instruction is in a different buffer (3-CP delay) or not in any buffer.

Programmable Clock

Each CPU has a programmable clock that generates interrupts at user-specified intervals. Available intervals range from 9 to $2^{32}-1$ CPs. Intervals shorter than 100 μs are not practical because of the time required by the monitor to process the interrupt. The instructions listed in Table 3-6 are used to enable and disable the programmable clock. These instructions are privileged to monitor mode.

Table 3-6. Programmable Clock Instructions

Machine Instruction	CAL Syntax	Description
0014*j*4	PCI S*j*	Transmit (S*j*) to the II register.
001405	CCI	Clear the programmable clock interrupt request.
001406	ECI	Enable the programmable clock interrupt request.
001407	DCI	Disable the programmable clock interrupt request.

The interrupt interval (II) register supports the programmable clock and is explained in the following subsection.

Interrupt Interval Register

The 32-bit II register is loaded with the number of CPs desired between programmable clock interrupt requests. Instruction 0014*j*4 transmits the 32 low-order bits of the S*j* register to the II register, with bit 2^3 always forced to a logical 1. The interval used is actually one CP more than the value stored in the II register. For example, if S*j* equals 0, the II register contains the value 8 (because bit 2^3 is always set), and the interval equals 9 CPs.

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

Operation

The 32-bit programmable clock is preset to the value contained in the II register when instruction 0014*j*4 executes. This clock runs continuously and decrements by 1 each CP until the contents of the clock is 0. The programmable clock then sets the programmable clock interrupt (PCI) request and reads the interval value held in the II register. The programmable clock repeats the countdown cycle and sets the PCI request at the intervals determined by the contents of the II register.

A PCI request can set the PCI interrupt flag only if IPC interrupt mode is set and enabled. If this mode is set, but not enabled, the PCI request is held. When the mode is enabled, the PCI request sets the PCI interrupt flag and causes an exchange. A held interrupt request or a set interrupt flag remains set until instruction 001405 executes and clears the request or flag.

Normally, IPC interrupt mode is not enabled while the program is in monitor mode. However, instruction 001302 can enable IPC interrupt mode if it is already set. If IPC interrupt mode is not set, it can be set while in monitor mode by instruction 001406. It can also be cleared while in monitor mode by instruction 001407 or disabled by instruction 001303.

Following a deadstart sequence, the monitor program ensures the state of the PCI request by issuing instructions 001405 and 001407 to clear and disable the PCI request.

Status Registers

There are eight status registers, numbered SR0 through SR7, in the CRAY Y-MP C90 mainframe. The organization of these registers is shown in Figure 3-15. Instruction 073*ij*1 transmits the contents of status register SR*j* to S*i*. Instruction 073*ij*5 transmits the contents of the S*i* register to SR*j*. Bits 2^{48} through 2^{52} of SR0 are the only bits that can be written to while in user mode, which is done with instruction 073*i*05.

SR0 contains information on the status of several bits in the active exchange package. The data fields of SR0 are described in Table 3-7.

Table 3-7. SR0 Data Fields

Bit Position	Status Bit	Description
63	CLN ≠ 0	Cluster number ≠ 0.
57	PS	Program state.
52	IBP	IBP interrupt mode.
51	FPS	Floating-point status (Sets if a floating-point error occurs).
50	IFP	IFP interrupt mode.
49	IOR	IOR interrupt mode.
48	BDM	Bidirectional memory mode.
47	PMBY	Performance monitor is busy.
43 – 40	Processor	CPU number.
36 – 32	Cluster Number	Cluster number.

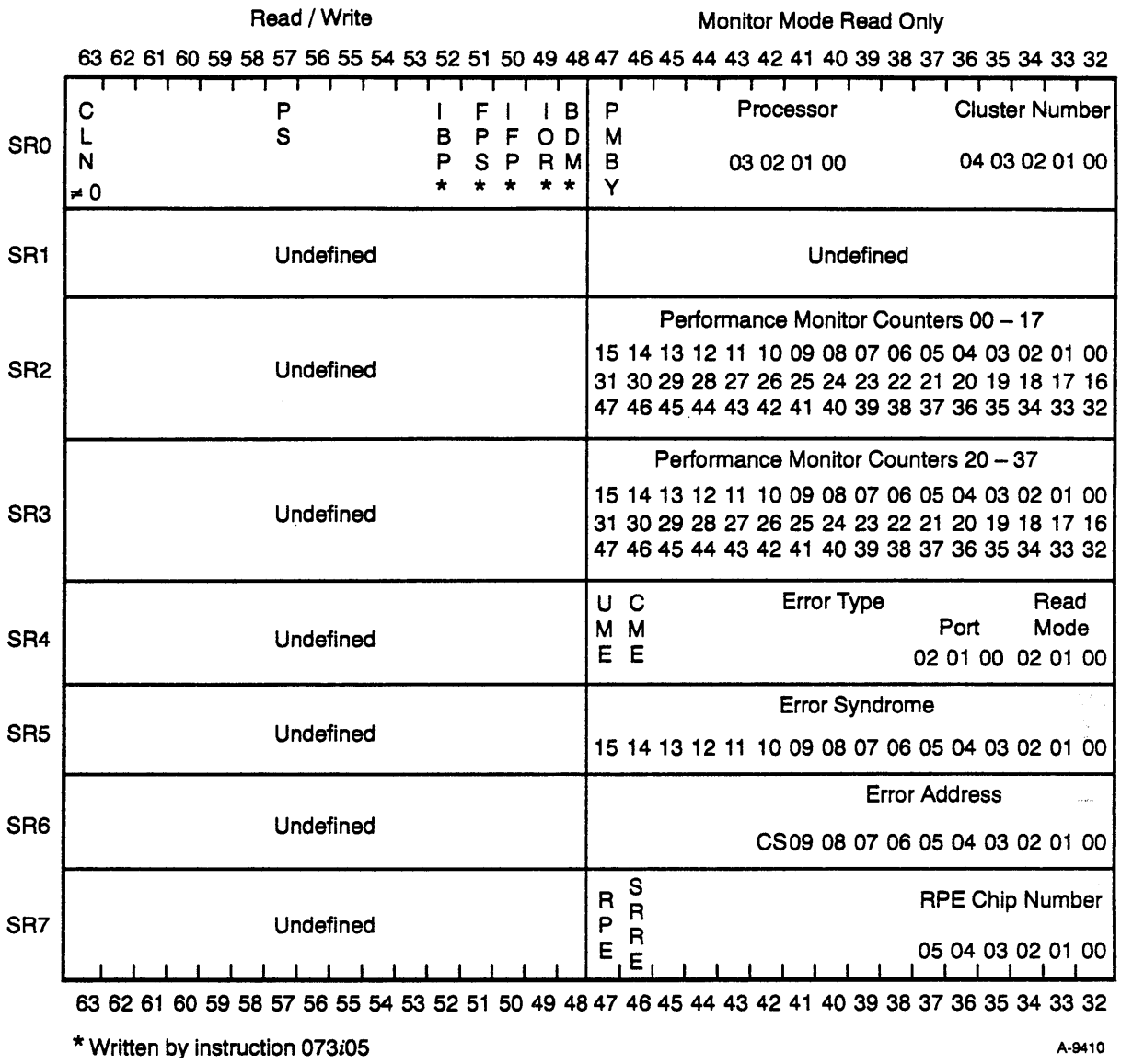


Figure 3-15. Status Registers

SR2 and SR3 contain the performance monitor counters. The operation of the performance monitor is explained in the next section.

SR4 through SR6 contain memory error information. Bits 2^{47} and 2^{46} of SR4 identify the error as either an uncorrectable or a correctable memory error, respectively. The read mode bits (2^{34} through 2^{32}) specify the type of read operation, and the port designator bits (2^{37} through 2^{35}) specify the port associated with the memory read error. Table 3-8 and Table 3-9 show these bits decoded.

Table 3-8. Read Mode Bits

2^2	2^1	2^0	Port A0	Port A1	Port B0	Port B1	Port D0	Port D1
0	0	0	Vector	Vector	Vector	Vector	VHISP/HISP 0	VHISP/HISP 1
0	0	1	B register	B register	T register	T register	LOSP	Maintenance
0	1	0	–	–	–	–	Fetch	Fetch
0	1	1	S register	–	–	–	Fetch	Fetch
1	1	1	A register	–	–	–	–	–

Table 3-9. Port Designator Bits

2^2	2^1	2^0	Port
0	0	0	Not valid
0	0	1	A0
0	1	0	A1
0	1	1	B0
1	0	0	B1
1	0	1	D0
1	1	0	D1
1	1	1	Not valid

SR5 contains the 16-bit error syndrome code of SBCDBD. This error correction and detection scheme is explained fully in Section 2.

SR6 contains bits 2^0 through 2^9 and bit 2^{20} of the memory error address. These bits can be decoded more specifically as shown in Table 3-10.

Table 3-10. Memory Error Address Bits

Bits	Memory Location
2^2 2^1 2^0	Section 0 – 7
2^5 2^4 2^3	Subsection 0 – 7
2^6	Bank group 0/1
2^9 2^8 2^7	Bank 0 – 7
2^{20}	Chip select

SR7 contains register parity error information. Bit 2^{47} indicates that a register parity error (RPE) has occurred. Bits 2^{37} through 2^{32} can be decoded as shown in Table 3-11 to determine exactly where the error occurred.

Table 3-11. Register Parity Error Bits

Octal	Description
00	Vector register V0 – V3, bits $2^0 - 2^{15}$, pipe 0.
01	Vector register V4 – V7, bits $2^0 - 2^{15}$, pipe 0.
02	Vector register V0 – V3, bits $2^{16} - 2^{31}$, pipe 0.
03	Vector register V4 – V7, bits $2^{16} - 2^{31}$, pipe 0.
04	Vector register V0 – V3, bits $2^{32} - 2^{47}$, pipe 0.
05	Vector register V4 – V7, bits $2^{32} - 2^{47}$, pipe 0.
06	Vector register V0 – V3, bits $2^{48} - 2^{63}$, pipe 0.
07	Vector register V4 – V7, bits $2^{48} - 2^{63}$, pipe 0.
10	Vector register V0 – V3, bits $2^0 - 2^{15}$, pipe 1.
11	Vector register V4 – V7, bits $2^0 - 2^{15}$, pipe 1.
12	Vector register V0 – V3, bits $2^{16} - 2^{31}$, pipe 1.
13	Vector register V4 – V7, bits $2^{16} - 2^{31}$, pipe 1.
14	Vector register V0 – V3, bits $2^{32} - 2^{47}$, pipe 1.
15	Vector register V4 – V7, bits $2^{32} - 2^{47}$, pipe 1.
16	Vector register V0 – V3, bits $2^{48} - 2^{63}$, pipe 1.
17	Vector register V4 – V7, bits $2^{48} - 2^{63}$, pipe 1.
20	T register, bits $2^0 - 2^{15}$, pipe 0/pipe 1.
21	T register, bits $2^{16} - 2^{31}$, pipe 0/pipe 1.
22	T register, bits $2^{32} - 2^{47}$, pipe 0/pipe 1.
23	T register, bits $2^{49} - 2^{63}$, pipe 0/pipe 1.
24	B register, bits $2^0 - 2^{15}$, pipe 0/pipe 1.
25	B register, bits $2^{16} - 2^{31}$, pipe 0/pipe 1.
26	Instruction buffers, bits $2^0 - 2^{15}$, pipe 0/pipe1.
27	Instruction buffers, bits $2^{16} - 2^{31}$, pipe 0/pipe 1.
30	Instruction buffers, bits $2^0 - 2^{15}$, pipe 0/pipe 1.
31	Instruction buffers, bits $2^{16} - 2^{31}$, pipe 0/pipe 1.
32	Shared registers SB/ST/SM.
33	Port A, A′ read address, modes delay.
34	Port B, B′ read address, modes delay.
35	Port D, D′ read address, modes delay.
36	Performance monitor counter 0 – 37.
37	Multiple errors.

Performance Monitor

The performance monitor tracks groups of hardware-related events. The results can be used to indicate the relative performance of a program. The performance monitor contains thirty-two 48-bit performance counters, which monitor the events shown in Table 3-12.

Table 3-12. Performance Monitor

Counter	Event Monitored	Instructions	Max. Increment per CP
	Number of:		
0	Clock periods monitored		+1
1	Instruction issued		+1
2	Clock periods holding issue		+1
3	Instruction fetches		+1
4	CPU memory references		+6
5	CPU memory conflicts		+6
6	I/O memory references		+2
7	I/O memory conflicts		+2
	Holding issue on:		
10	A registers and access conflicts		+1
11	S registers and access conflicts		+1
12	V registers		+1
13	B/T registers		+1
14	Functional units		+1
15	Shared registers		+1
16	Memory ports		+1
17	Miscellaneous		+1
	Number of:		
20	Instructions 000 through 004	000 – 004	+1
21	Branches	005 – 017	+1
22	Address instructions	02x, 030 – 033	+1
23	B/T memory instructions	034 – 037	+1
24	Scalar instructions	040 – 043, 071 – 077	+1
25	Scalar integer instructions	044 – 061	+1
26	Scalar floating-point instructions	062 – 070	+1
27	S/A memory instructions	10x – 13x	+1
	Number of:		
30	Vector logicals	14x, 175	+VL
31	Vector shifts, pop., leading zero	150 – 153, 174xx (1 – 7)	+VL
32	Vector integer adds	154 – 157	+VL
33	Vector floating-point multiplies	160 – 167	+VL
34	Vector floating-point adds	170 – 173	+VL
35	Vector floating-point reciprocals	174xx0	+VL
36	Vector memory reads	176	+VL
37	Vector memory writes	177	+VL

Performance events are monitored only when the system is operating in nonmonitor mode. Entering monitor mode disables the performance counters.

There are two types of instructions used with the performance monitor: user instructions and maintenance instructions. The user instructions allow the user to select and read the performance monitor. The maintenance instructions test the logic of the performance monitor. The following subsections explain how these instructions are used with the performance monitor.

Selecting and Reading Performance Events

The performance counters can continuously monitor events for approximately 156 hours before they must be reset. Fifty CPs must elapse before you can issue another performance monitor instruction.

Instruction 073*i*21 reads consecutive 16-bit segments of performance monitor (PM) counters 0 through 17_8 into bits 2^{32} through 2^{47} of the S*i* register. Instruction 073*i*31 reads consecutive 16-bit segments of PM counters 20_8 through 37_8 into bits 2^{32} through 2^{47} of the S*i* register. Neither of these instructions should be issued if the PM is busy (bit 2^{47} of status register 0 set).

Each PM counter is 48 bits wide and is divided into three 16-bit segments. A performance counter pointer selects the 16-bit segment to be read into the S register. This pointer is cleared on entry to or exit from monitor mode or by instruction 001500. Each successive execution of instruction 073*i*21 or 073*i*31 advances the pointer, enabling the next instruction of the same type to read the next 16-bit segment of the appropriate PM counter. A 3-CP delay must occur between successive PM reads for the data to be valid. The read sequences for the PM counters are shown below.

Instruction 073*i*21 reads PM counters 0 through 17_8:

- First read returns counter 00 (bits $2^0 - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$).
- Second read returns counter 00 (bits $2^{16} - 2^{31}$) to S*i* (bits $2^{32} - 2^{47}$).
- Third read returns counter 00 (bits $2^{32} - 2^{47}$) to S*i* (bits $2^{32} - 2^{47}$).
- Fourth read returns counter 01 (bits $2^0 - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$) (48 reads returns the pointer to counter 00).

Instruction 073*i*31 reads PM counters 20_8 through 37_8:

- First read returns counter 20 (bits $2^0 - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$).

- Second read returns counter 20 (bits $2^{16} - 2^{31}$) to S*i* (bits $2^{32} - 2^{47}$).
- Third read returns counter 20 (bits $2^{32} - 2^{47}$) to S*i* (bits $2^{32} - 2^{47}$).
- Fourth read returns counter 21 (bits $2^{0} - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$) (48 reads returns the pointer to counter 20).

Testing Performance Counters

Instructions 073*i*75 and 073*i*25 are used to test the operation of the performance counters. This testing can be performed only when the system is in performance monitor (PM) maintenance mode. The procedure for performing this test is explained fully in the *CRAY Y-MP C90 Computer System Hardware Maintenance Manual*, publication number CMM-0502-000.

4 CPU COMPUTATION SECTION

Each central processing unit (CPU) is an identical, independent computation section consisting of operating registers, functional units, and an instruction control network (refer again to Figure 1-2). The operating registers and functional units store and process three types of data: address, scalar, and vector.

Address data is used to control internal operations and consists of information such as memory addresses, register designators, and indexes. Address data is stored in the address (A) registers and intermediate address (B) registers, and is processed in two dedicated functional units.

Scalar data is any discrete numerical quantity that can be processed in functional units either singly or in operand pairs to produce a single scalar result. Scalar data is stored in the scalar (S) registers and the intermediate scalar (T) registers, and is processed in four dedicated functional units. Scalar floating-point data is processed in one of three floating-point functional units; these functional units are also used to process vector floating-point data.

Vector data refers to a set (or vector) of discrete numerical quantities that can be referenced by a single name. Vector data can be processed either singly or in operand pairs in special functional units to produce a vector result. Practically speaking, this means that a single instruction can result in the same operation being performed sequentially on a whole set of operands to produce a set of results. Vector data is stored in the vector (V) registers and is processed in five dedicated functional units. Vector floating-point data is processed in one of three floating-point functional units; these functional units are also used to process scalar floating-point data.

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. Depending on the instruction sequence, data flows along either the scalar or vector path with two exceptions. In some cases, the scalar registers may provide one of the required operands for some vector operations performed in the vector functional units. Also, some scalar functional units return their results to an address register.

The computation section performs integer or floating-point arithmetic operations. Integer arithmetic is performed in two's complement mode; floating-point quantities have signed magnitude representation.

Integer (or fixed-point) operations are integer addition, integer subtraction, and integer multiplication. No integer division instruction is provided; the operation is accomplished through a software algorithm using floating-point hardware.

Floating-point instructions allow addition, subtraction, multiplication, and reciprocal approximation operations. The reciprocal approximation instructions used in conjunction with other instructions allow for a floating-point division operation.

The instruction set includes logical operations for AND, inclusive OR, exclusive OR, exclusive NOR, and mask-controlled merge operations. Shift operations allow the manipulation of either 64-bit or 128-bit operands to produce 64-bit results. With the exception of 32-bit integer arithmetic performed in the A register functional units, most operations are used in vector or scalar instructions.

The 32-bit integer product is a vector instruction designed for index calculation. A full indexing capability is possible throughout central memory in either scalar or vector modes. The index can be positive or negative in either mode. Indexing allows matrix operations in vector mode to be performed on rows or on the diagonal as well as allowing conventional column-oriented operations.

The following subsections describe each of the operating registers and their associated functional units.

Operating Registers

Each CPU has three primary and two intermediate sets of operating registers. The primary sets of operating registers are the A, S, and V registers. These registers are considered primary because functional units and central memory can access them directly.

For the A and S registers, an intermediate level of registers exists; they are not accessible to the functional units and serve mainly as a memory buffer for the primary registers. To reduce the number of memory reference instructions for scalar and address operations, block transfers are possible between these intermediate registers and central memory. The A registers are supported by the B registers, while the S registers are supported by the T registers. The V registers do not have associated intermediate registers.

Address (A) Registers

Figure 4-1 shows the eight A registers and their associated CPU hardware. The A registers are designated A0 through A7. The following subsections explain A register functions, special uses, and instructions.

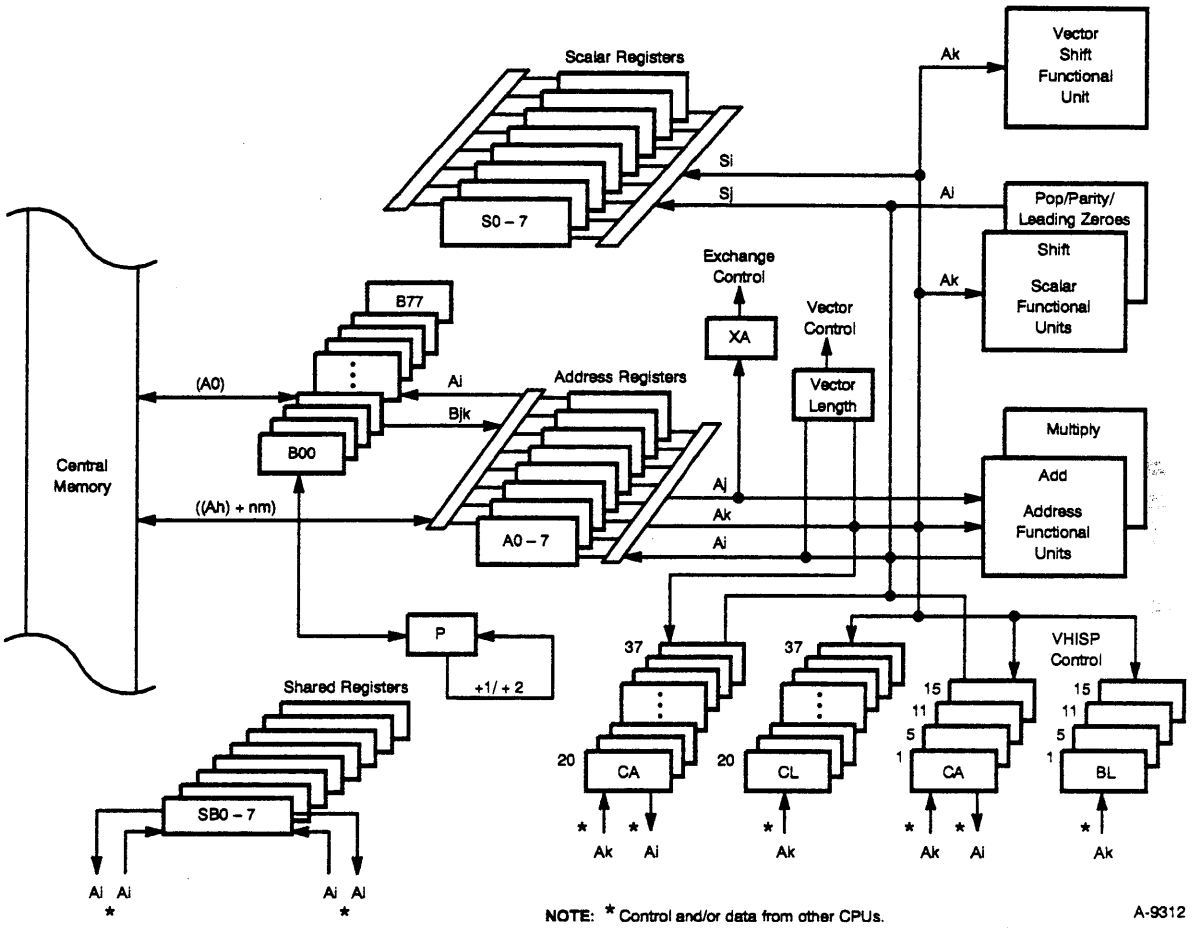


Figure 4-1. A Register Block Diagram

A Register Functions

The A registers are used primarily as address registers for memory references and as index registers. They are also used to transfer data to and from various internal registers. Each A register can store up to 32 bits of data. While all 32 bits enter or leave the memory modules, the upper five bits, 2^{27} through 2^{31}, are not used in memory referencing instructions on the 128-Mword CRAY Y-MP C90 system. Refer to "Absolute Memory Address Calculating" in Section 2 for additional information.

The A registers index the base address for scalar memory references and provide both a base address and an address increment or block count for block memory references. The A registers also provide values for shift counts and I/O channel operations and serve as result registers for the scalar population/parity/leading zero functional unit.

The A registers are connected to the vector length (VL) and exchange address (XA) registers. The VL register is loaded by the 00200*k* instruction. The XA register is loaded by the 0013*j*0 instruction only while the system is operating in monitor mode. Refer to "Vector Length Register" later in this section for more information on the VL register, and refer to "Exchange Address Register Field" in Section 3 for more information on the XA register.

Data either moves directly between central memory and the A registers or it is placed in the B registers. Placing data in the B registers allows buffering of the data between A registers and central memory. Data can also be transferred between A and S registers and between A and shared address (SB) registers.

The following list summarizes the functions of the A registers:

- Generate addresses for memory references.
- Serve as index registers or store address increments or block counts for memory references.
- Set the CA and CL registers and enable and disable channel interrupts (I/O control).
- Provide values for shift counts and loop controls.
- Serve as result registers for the scalar population/parity/leading zero functional unit.
- Set the XA register (exchange control).
- Set and read the VL register (vector control).
- Transfer data between the A and S registers.
- Transfer data between the A and SB registers.
- Provide indirect addresses for referencing semaphore registers and for data transfers between A and SB registers, between S and ST registers, and between S and V registers.

The address functional units support address and index generation by performing 32-bit integer arithmetic on operands obtained from A registers and by delivering the results to A registers. Refer to "Address Functional Units" in this section for more information on the address functional units.

Special A Register Values

If register A0 is referenced in the *h*, *j*, or *k* fields of an instruction, the contents of the register are not used; instead, a special operand is generated. This special value is available immediately regardless of existing A0 register reservations (they are not checked in this instance), and this value does not alter the data stored in the A0 register. Table 4-1 shows the special A0 register values.

Table 4-1. Special A0 Register Values

Field	Operand Value
A*h*, *h*=0	0
A*j*, *j*=0	0
A*k*, *k*=0	1

If the *i* field equals 0, then the contents of register A0 are used. The *i* field is not used as a special case.

Bypass Path

In addition to being routed to the A registers, write data can simultaneously be routed along a bypass path. This bypass path is used when a read operation follows a write operation to the same A register. This path makes read data available 1 clock period (CP) sooner than would otherwise be possible. Figure 4-2 shows the timing of two instructions that cause data to use the bypass path. A bypass occurs because instruction 030415 is trying to read data from A1 before the data in A1 is valid. The bypass operation saves 1 CP by allowing result data to go directly into a read-out register without having to travel first through the A1 register.

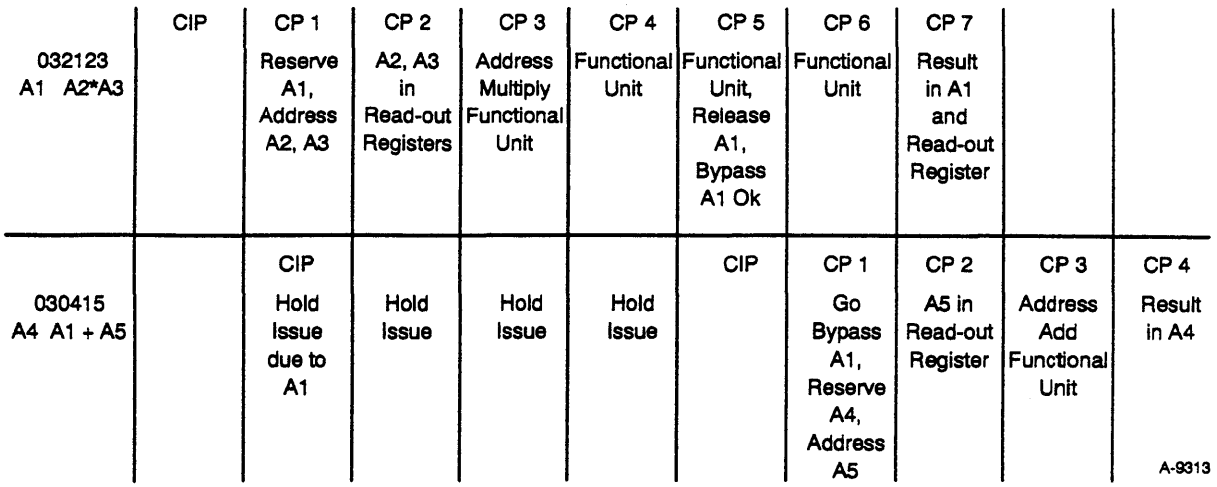


Figure 4-2. Instruction Timing for a Bypass Operation

A Register Instructions

Only one result per CP can be transferred to an A register. When an instruction delivering new data to an A register issues, a reservation is set for that register. The reservation prevents issue of instructions that use the specified register until the new data is on the input path to the register. Instructions reference A registers by specifying the register number as the *h*, *i*, *j*, or *k* designator (refer to "Instruction Formats" in Section 7 for more information on instruction fields). A0 is the only A register that can be referenced when it is not specified in one of the instruction fields.

Table 4-2 lists the A register instructions and provides octal and CAL codes. Refer to "CPU Instruction Descriptions" in Section 7 for complete information on these instructions.

Table 4-2. A Register Instructions

Machine Instruction	CAL Syntax	Description	Type of Instruction
10*hi*00 *nm*	A*i* *exp*,A*h*	Read from ((A*h*) + *exp* + (DBA)) to A*i*.	Memory Transfer
100*i*00 *nm*	A*i* *exp*,0	Read from (*exp* + (DBA)) to A*i*.	
10*hi*00 00	A*i* ,A*h*	Read from ((A*h*) + (DBA)) to A*i*.	
11*hi*00 *nm*	*exp*,A*h* A*i*	Write (A*i*) to ((A*h*) + *exp* + (DBA).	
110*i*00 *nm*	*exp*,0 A*i*	Write (A*i*) to (*exp* + (DBA).	

Table 4-2. A Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
11*hi*00 00	,A*h* A*i*	Write (A*i*) to ((A*h*) + (DBA).	Memory Transfer
12*hi*00 *nm*	*Si exp*,A*h*	Read from ((A*h*) + *exp* + (DBA)) to *Si*.	Index for Memory Transfer
13*hi*00 *nm*	*exp*,A*h* *Si*	Write (S*i*) to ((A*h*) + *exp* + (DBA).	
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words starting at address (A0) + (DBA) to B registers starting at register *jk*.	Memory Block Transfer
035*ijk*	,A0 B*jk*,A*i*	Write (A*i*) words from B registers starting at register *jk* to memory starting at (A0) + (DBA).	
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words starting at address (A0) + (DBA) to T registers starting at register *jk*.	
037*ijk*	,A0 T*jk*,A*i*	Write (A*i*) words from T registers starting at register *jk* to memory starting at (A0) + (DBA).	
176*i*0*k*	V*i* ,A0,A*k*	Read (VL) words to V*i* starting at address (A0) + (DBA), incrementing by (A*k*).	
176*i*1*k*	V*i* ,A0,V*k*	Read (VL) words to V*i* using memory addresses ((A0) + (V*k*) + (DBA)).	
1770*jk*	,A0,A*k* V*j*	Write (VL) words from (V*j*) to memory starting at (A0) + (DBA), incrementing by (A*k*).	
1771*jk*	,A0,V*k* V*j*	Write (VL) words from (V*j*) to memory using memory addresses ((A0) + (V*k*) + (DBA)).	
0013*j*0	XA A*j*	Transmit (A*j*) to the XA register.	Interregister Transfer
0014*j*3	CLN A*j*	Transmit (A*j*) to the CLN register.	
0017*jk*	BP,*k* A*j*	Transmit (A*j*) to breakpoint address *k*.	
00200*k*	VL A*k*	Transmit (A*k*) to the VL register.	
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*.	
025*ijk*	B*jk* A*i*	Transmit (A*i*) to B*jk*.	

Table 4-2. A Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
026*ij*4	A*i* SB,A*j*,+1	Transmit (SB) designated by (A*j*) to A*i*, and increment (SB,A*j*) by 1.	Interregister Transfer
026*ij*5	A*i* SB*j*,+1	Transmit (SB*j*) to A*i*, and increment (SB*j*) by 1.	
026*ij*6	A*i* SB,A*j*	Transmit (SB) designated by (A*j*) to A*i*.	
026*ij*7	A*i* SB*j*	Transmit (SB*j*) to A*i*.	
027*ij*6	SB,A*j* A*i*	Transmit (A*i*) to SB designated by (A*j*).	
027*ij*7	SB*j* A*i*	Transmit (A*i*) to SB*j*.	
030*i*0*k*	A*i* A*k*	Transmit (A*k*) to A*i*.	
031*i*0*k*	A*i* – A*k*	Transmit the negative of (A*k*) to A*i*.	
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 an unnormalized floating-point number.	
020*i*00 *nm*	A*i* *exp*	Transmit *exp* (*nm*) to A*i*.	Constant Entry
021*i*00 *nm*	A*i* *exp*	Transmit one's complement of *exp* (*nm*) to A*i*.	
022*ijk*	A*i* *exp*	Transmit *exp* (*jk*) to A*i*.	
031*i*00	A*i* –1	Transmit –1 to A*i*.	
0010*jk*	CA,A*j* A*k*	Set the CA register for channel (A*j*) to (A*k*) and begin I/O sequence.	I/O Channels
0011*jk*	CL,A*j* A*k*	Set the CL register for channel (A*j*) to (A*k*).	
0012*j*0	CL,A*j*	Clear the interrupt and error flags for channel (A*j*); clear device master clear (output channels only); enable channel interrupt.	
0012*j*1	MC,A*j*	Clear the interrupt and error flags for channel (A*j*); set device master clear (output channels only); clear device ready held (input channels only).	
0012*j*2	DI,A*j*	Disable channel (A*j*) interrupts.	
0012*j*3	EI,A*j*	Enable channel (A*j*) interrupts.	

Table 4-2. A Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
033*i*00	A*i* CI	Transmit to A*i* the channel number of the highest priority channel requesting an interrupt.	I/O Channels
033*ij*0	A*i* CA,A*j*	Transmit the current address of channel (A*j*) to A*i* (*j* ≠ 0).	
033*ij*1	A*i* CE,A*j*	Transmit channel status word for channel (A*j*) to A*i* (*j* ≠ 0).	
030*ijk*	A*i* A*j* + A*k*	Transmit the integer sum of (A*j*) and (A*k*) to A*i*.	Integer Operation
030*ij*0	A*i* A*j* + 1	Transmit the integer sum of (A*j*) and 1 to A*i*.	
031*ijk*	A*i* A*j* - A*k*	Transmit the integer difference of (A*j*) and (A*k*) to A*i*.	
031*ij*0	A*i* A*j* - 1	Transmit the integer difference of (A*j*) and 1 to A*i*.	
032*ijk*	A*i* A*j* * A*k*	Transmit the integer product of (A*j*) and (A*k*) to A*i*.	
010000 *nm*	JAZ *exp*	Jump to *exp* if (A0) = 0.	Conditional Jump
011000 *nm*	JAN *exp*	Jump to *exp* if (A0) ≠ 0.	
012000 *nm*	JAP *exp*	Jump to *exp* if (A0) is positive; (A0) ≥ 0.	
013000 *nm*	JAM *exp*	Jump to *exp* if (A0) is negative.	
026*ij*0	A*i* PS*j*	Transmit the population count of (S*j*) to A*i*.	Bit Count
026*ij*1	A*i* QS*j*	Transmit the population count parity of (S*j*) to A*i*.	
027*ij*0	A*i* ZS*j*	Transmit leading zero count of (S*j*) to A*i*.	
0014*j*1	SIPI A*j*	Send an interprocessor interrupt request to CPU (A*j*).	Interrupt
0034*jk*	SM,A*k* 1,TS	Test and set semaphore A*k* (*j*2 = 1).	Indirect Address for Semaphore
0036*jk*	SM,A*k* 0	Clear semaphore A*k* (*j*2 = 1).	
0037*jk*	SM,A*k* 1	Set semaphore A*k* (*j*2 = 1).	

Table 4-2. A Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
0064*jk nm*	JTS,A*k* *exp*	Branch to *exp* if SM,A*k* = 1; else set SM,A*k* (*j*2 = 1).	Indirect Address for Semaphore
076*ijk*	S*i* V*j*,A*k*	Transmit (V*j* element (A*k*)) to S*i*.	Indirect Address for Vector
077*ijk*	V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*).	
056*ijk*	S*i* S*i*,S*j* < A*k*	Shift (S*i*) and (S*j*) left (A*k*) places to S*i*.	Indirect Address for Shift Count
057*ijk*	S*i* S*j*,S*i* > A*k*	Shift (S*j*) and (S*i*) right (A*k*) places to S*i*.	
150*ijk*	V*i* V*j* < A*k*	Shift (V*j* elements) left (A*k*) places to V*i* elements.	
151*ijk*	V*i* V*j* > A*k*	Shift (V*j* elements) right (A*k*) places to V*i* elements.	
152*ijk*	V*i* V*j*,V*j* < A*k*	Double shift (V*j* elements) left (A*k*) places to V*i* elements.	
005400 152*ijk*	V*i* V*j*,A*k*	Transfer (V*j* elements) to V*i* elements starting at element (A*k*).	
153*ijk*	V*i* V*j*,V*j* > A*k*	Double shift (V*j* elements) right (A*k*) places to V*i* elements.	

Intermediate Address (B) Registers

Sixty-four 32-bit B registers are designated $B00_8$ through $B77_8$. The B registers are used as intermediate storage registers for the A registers. Typically, B registers contain data to be referenced repeatedly over a long time period, making it inefficient to retain the data in either A registers or in central memory. Data transfers between an A and B register take 1 CP. Examples of data stored in B registers are loop counts, variable array base addresses, and array dimensions.

Instructions reference B registers by specifying the B register number in the *jk* field. Refer to "Instruction Formats" in Section 7 for more information on instruction fields.

B register data transfers to or from central memory at a maximum rate of two words per CP. The 7 low-order bits of the contents of register A*i* specify the number of words to be transmitted. The first register

involved in a block transfer is specified by the *jk* fields of the instruction; successive data words transferred involve successive B registers until B77 is reached. Register B00 is processed after register B77 if the count in register A*i* is not exhausted. During these block transfers, a reservation is placed on all B registers. Other instructions can issue while B register data is being transferred to or from central memory.

The B registers are protected with parity bits. When a word is written into a B 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 B register. An error is indicated when the two sets do not match. Parity errors are reported to the register parity error (RPE) field of the exchange package. Refer to "Status Registers" in Section 3 for additional information on parity errors.

B00 is the only B register that can be referenced when it is not specified in one of the instruction fields. Instruction 007*ijkm* or 007000 *nm* automatically stores the return address for a subroutine jump in register B00. Table 4-3 lists the B register instructions.

Table 4-3. B Register Instructions

Machine Instruction	CAL Syntax	Description	Type of Instruction
024*ijk*	A*i* B*jk*	Transmit (B*jk*) to A*i*.	Interregister Transfer
025*ijk*	B*jk* A*i*	Transmit (A*i*) to B*jk*.	
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words starting at address (A0) + (DBA) to B registers starting at register *jk*.	Block Transfer
035*ijk*	,A0 B*jk*,A*i*	Write (A*i*) words from B registers starting at register *jk* to memory starting at (A0) + (DBA).	
0050*jk*	J B*jk*	Jump to (B*jk*).	Jump
0051*jk*	J B*jk*	Jump to (B*jk*). (Maintenance only: invalidates instruction buffers.)	
007*ijkm*	R *exp*	Return jump to *exp* and set register B00 to (P) + 2.	
007000 *nm*	R *exp*	Return jump to *exp* and set register B00 to (P) + 3.	

A and B Register Troubleshooting

For assistance in troubleshooting problems with the A and B registers, refer to Figure 4-3, a block diagram of these registers showing the options involved and the signals that pass between them. For a more detailed description of this diagram, refer to the *CRAY Y-MP C90 Computer System Hardware Maintenance Manual*, publication number CMM-0502-000.

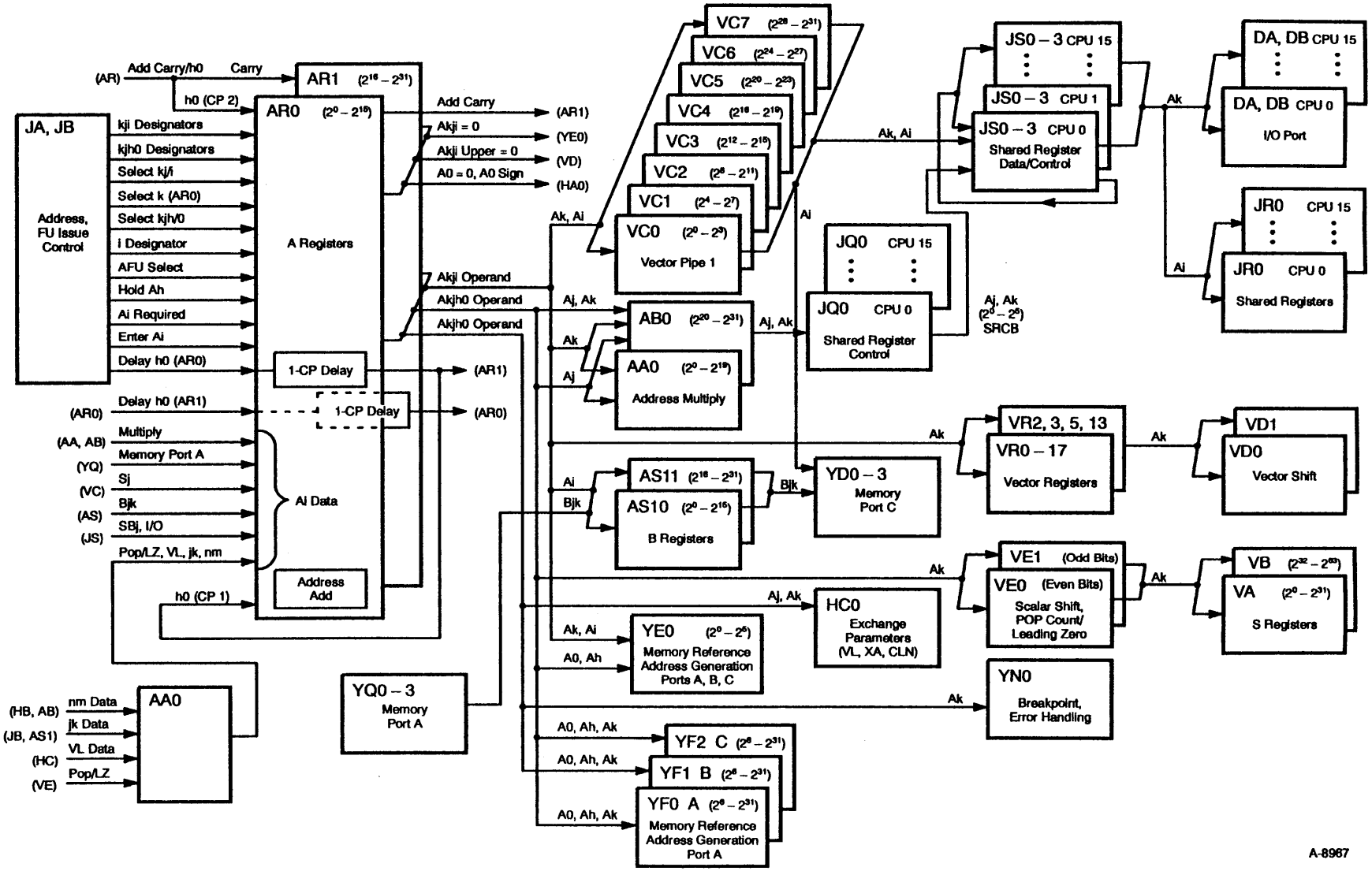


Figure 4-3. A and B Registers Troubleshooting Block Diagram

Scalar (S) Registers

Figure 4-4 shows the eight S registers and their associated hardware. The S registers are designated $S0_8$ through $S7_8$, and each one can store up to 64 bits of data. The following subsections explain S register functions, special uses, and instructions.

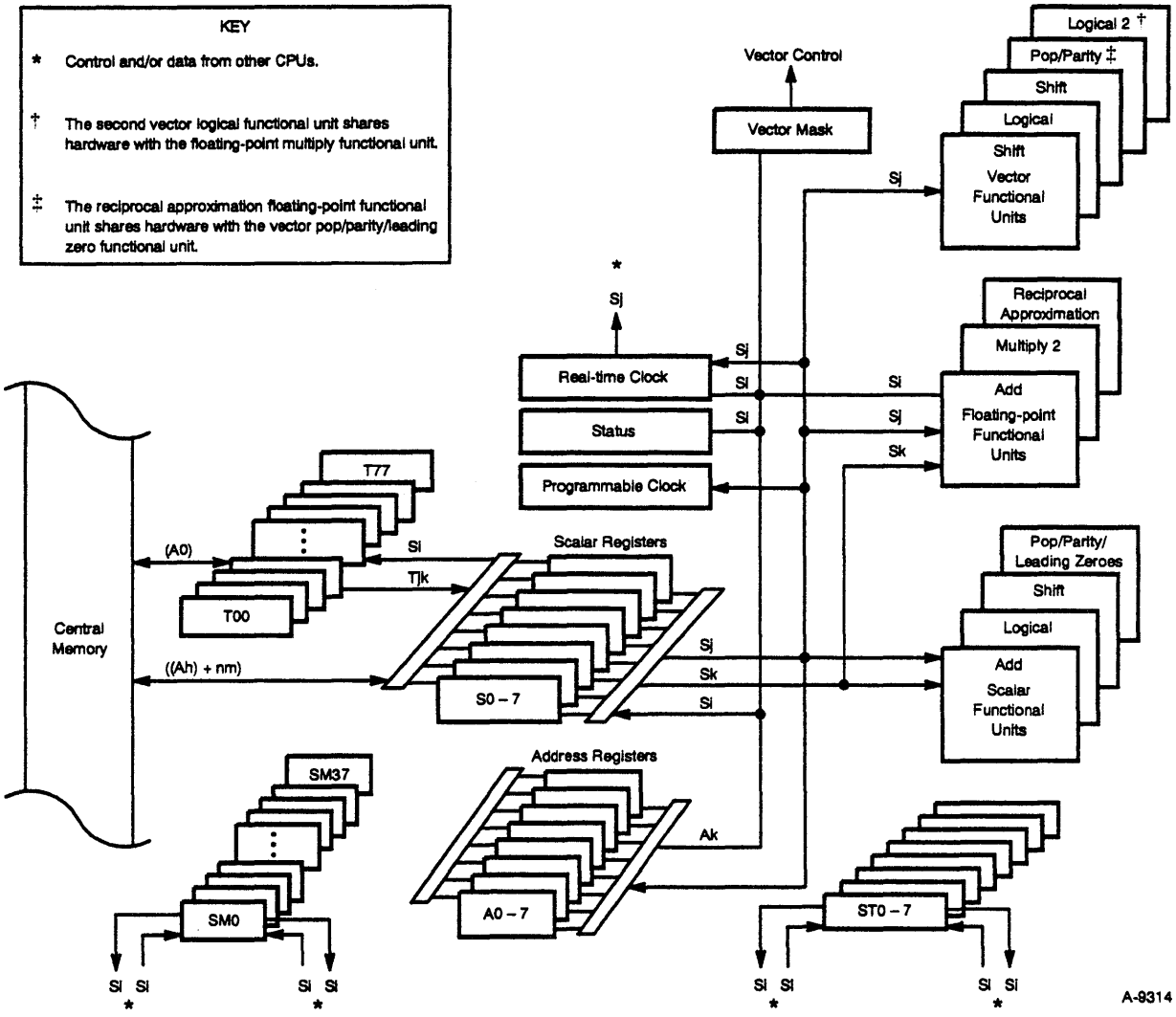


Figure 4-4. Scalar Register Block Diagram

S Register Functions

The S registers are the principal scalar registers for a CPU, serving as the source and destination for operands performing scalar arithmetic and logical operations. The scalar functional units support the S registers by

performing both integer and floating-point arithmetic operations. Refer to "Scalar Functional Units" in this section for more information on the scalar functional units.

Data moves directly between central memory and S registers or is placed in the T registers. Placing data in T registers allows buffering of scalar operands between S registers and central memory. Data can also be transferred between S and A registers, between S registers and elements of V registers, between S and shared scalar (ST) registers, and between S and semaphore (SM) registers.

Data from the status registers (SR0 through SR7) can be transferred to the S registers. The SR registers include the performance monitor (PM) counters and the maintenance modes (MM) register. S register data can be written to SR0 and to the MM register.

The contents of the vector mask (VM) register or real-time clock (RTC) register can be transferred to an S register. Data from an S registers can be transferred to either of these two registers, as well as to the interrupt interval (II) register in the programmable clock.

Instructions 071*i*30 through 071*i*70 generate frequently used floating-point constants and store them in an S register. These constants are used for special floating-point operations, such as conversions of numbers from integer to floating-point format.

The following list summarizes the functions of the S registers:

- Perform scalar arithmetic and logical operations.
- Transfer data between S registers and A, V, ST, or SM registers.
- Read the contents of the status registers or write data to SR0 or the MM register.
- Set/read the RTC and VM registers.
- Set the II register.
- Provide constant values for vector floating-point operations.

Special S Register Values

If register S0 is referenced in the *j* or *k* fields of an instruction, the contents of the register are not used; instead, a special operand is generated. This special value is available immediately regardless of

existing S0 register reservations (they are not checked in this instance). This usage does not alter the data stored in the S0 register. Table 4-4 shows the special S0 register values.

Table 4-4. Special S0 Register Values

Field	Operand Value
S*j*, *j* = 0	0
S*k*, *k* = 0	2^{63}

If the *i* field equals 0, then the contents of the S0 register are used. The *i* field is not used as a special case.

S Register Instructions

Only one result per CP can be transferred to the S registers. When an instruction delivering new data to an S register issues, the register is reserved. This prevents subsequent instructions from reading data from the register until the new data is delivered. Instructions reference S registers by specifying the register number as the *i*, *j*, or *k* designator. Refer to "Instruction Formats" in Section 7 for more information on instruction fields. S0 is the only S register that can be referenced when it is not specified in one of the instruction fields.

Table 4-5 lists S register instructions and provides the octal and CAL codes. Refer to "CPU Instruction Descriptions" in Section 7 for complete information on these instructions. The contents of the DBA register are added to instruction-generated memory addresses to form physical memory addresses. Refer to "Address Range Checking" in Section 2.

There is only one input path to the S registers; therefore, all instructions that write data into the S registers must reserve the path for the CP when the data is expected. Because each instruction takes a predetermined amount of time to complete, the decode of the instruction in the issue hardware automatically selects the proper CP at which to reserve the input path. If the path is already reserved for that CP, the instruction holds issue. The instruction continues to hold issue until the S register input path is available at the necessary CP. The instruction then issues and reserves the path for that CP.

Table 4-5. S Register Instructions

Machine Instruction	CAL Syntax	Description	Type of Instruction
040*i*00 *nm*	S*i* *exp*	Transmit *nm* to S*i*, bits $2^0 - 2^{31}$ (bits $2^{32} - 2^{63}$ are set to 0).	Register Entry
040*i*20 *nm*	S*i* S*i*:*exp*	Transmit *nm* to S*i*, bits $2^0 - 2^{31}$ (bits $2^{32} - 2^{63}$ unchanged).	
040*i*40 *nm*	S*i* *exp*:S*i*	Transmit *nm* to S*i*, bits $2^{32} - 2^{63}$ (bits $2^0 - 2^{31}$ unchanged).	
041*i*00 *nm*	S*i* *exp*	Transmit one's complement of *nm* to S*i*.	
042*ijk*	S*i* < *exp*	Form ones mask in S*i* *exp* bits from the right; the *jk* field contains the value 100_8 - *exp*.	
042*ijk*	S*i* #> *exp*	Form zeroes mask in S*i* *exp* bits from the left; the *jk* field contains the value *exp*.	
042*i*77	S*i* 1	Transmit 1 to the S*i* register.	
042*i*00	S*i* –1	Transmit –1 to the S*i* register.	
043*ijk*	S*i* #< *exp*	Form zeroes mask in S*i* *exp* bits from the right; the *jk* field contains the value 100_8 – *exp*.	
043*i*00	S*i* 0	Clear the S*i* register.	
047*i*00	S*i* #SB	Transmit the one's complement of the sign bit to the S*i* register.	
071*i*30	S*i* 0.6	Transmit 0.75 x 2^{48} to S*i* as a normalized floating-point constant.	
071*i*40	S*i* 0.4	Transmit 0.4 to S*i* as a normalized floating-point constant.	
071*i*50	S*i* 1.0	Transmit 1.0 to S*i* as a normalized floating-point constant.	
071*i*60	S*i* 2.0	Transmit 2.0 to S*i* as a normalized floating-point constant.	
071*i*70	S*i* 4.0	Transmit 4.0 to S*i* as a normalized floating-point constant.	
12*hi*00 *nm*	S*i* *exp*,A*h*	Read from ((A*h*) + *exp* + (DBA)) to S*i*.	Memory Transfer
120*i*00 *nm*	S*i* *exp*,0	Read from (*exp* + (DBA)) to S*i*.	
12*hi*00 00	S*i* ,A*h*	Read from ((A*h*) + (DBA)) to S*i*.	

Table 4-5. S Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
13*hi*00 *nm*	*exp*,A*h* S*i*	Write (S*i*) to ((A*h*) + *exp* + (DBA).	Memory Transfer
130*i*00 *nm*	*exp*,0 S*i*	Write (S*i*) to (*exp* + (DBA).	
13*hi*00 00	,A*h* S*i*	Write (S*i*) to ((A*h*) + (DBA).	
0030*j*0	VM S*j*	Transmit (S*j*) to VM lower register.	Interregister Transfer
0030*j*1	VM1 S*j*	Transmit (S*j*) to VM upper register.	
023*ij*0	A*i* S*j*	Transmit (S*j*) to A*i*.	
047*i*0*k*	S*i* #S*k*	Transmit the one's complement of (S*k*) to S*i*.	
051*i*0*k*	S*i* S*k*	Transmit (S*k*) to S*i*.	
072*i*00	S*i* RT	Transmit (RTC) to S*i*.	
072*i*02	S*i* ST*j*	Transmit (ST*j*) to S*i*.	
072*ij*3	S*i* ST*j*	Transmit (ST*j*) to S*i*.	
072*ij*6	S*i* ST,A*j*	Transmit (ST) designated by (A*j*) to S*i*.	
073*i*00	S*i* VM	Transmit (VM) to S*i*.	
073*i*02	SM S*i*	Transmit (S*i*) to SM.	
073*i*10	S*i* VM1	Transmit (VM1) to S*i*.	
073*i*21	S*i* SR2	Read PM counters 00 – 17 and increment pointer.	
073*i*25	SR2 S*i*	Issue PM maintenance advance.	
073*i*31	S*i* SR3	Read PM counters 20 – 37 and increment pointer.	
073*i*75	SR7 S*i*	Transmit (S*i*) to maintenance mode register.	
073*ij*1	S*i* SR*j*	Transmit (SR*j*) to S*i*.	
073*ij*3	ST*j* S*i*	Transmit (S*i*) to ST*j*.	
073*ij*5	SR*j* S*i*	Transmit (S*i*) to SR*j*.	
073*ij*6	ST,A*j* S*i*	Transmit (S*i*) to ST designated by (A*j*).	

Table 4-5. S Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
074*ijk*	S*i* T*jk*	Transmit (T*jk*) to S*i*.	Interregister Transfer
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*.	
077*ijk*	V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*).	
146*ijk*	V*i* S*j*|V*k*&VM	Transmit (S*j*) if VM bit = 1, or (V*k* element) if VM bit = 0, to V*i* elements.	
060*ijk*	S*i* S*j* + S*k*	Transmit the integer sum of (S*j*) and (S*k*) to S*i*.	Integer Operation
061*ijk*	S*i* S*j* – S*k*	Transmit the integer difference of (S*j*) and (S*k*) to S*i*.	
061*i*0*k*	S*i* –S*k*	Transmit the negative of (S*k*) to S*i*.	
154*ijk*	V*i* S*j* + V*k*	Transmit the integer sums of (S*j*) and (V*k* elements) to V*i* elements.	
156*ijk*	V*i* S*j* – V*k*	Transmit the integer differences of (S*j*) and (V*k* elements) to V*i* elements.	
166*ijk*	V*i* S*j**V*k*	Transmit the 32-bit integer products of (S*j*) and (V*k* elements) to V*i* elements.	
062*ijk*	S*i* S*j* + FS*k*	Transmit the floating-point sum of (S*j*) and (S*k*) to S*i*.	Floating-point Operation
062*i*0*k*	S*i* + FS*k*	Transmit the normalized (S*k*) to S*i*.	
063*ijk*	S*i* S*j* – FS*k*	Transmit the floating-point difference of (S*j*) and (S*k*) to S*i*.	
063*i*0*k*	S*i* – FS*k*	Transmit the normalized negative of (S*k*) to S*i*.	
064*ijk*	S*i* S*j* * FS*k*	Transmit the floating-point product of (S*j*) and (S*k*) to S*i*.	
065*ijk*	S*i* S*j* * HS*k*	Transmit the half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*.	
066*ijk*	S*i* S*j* * RS*k*	Transmit the rounded floating-point product of (S*j*) and (S*k*) to S*i*.	
067*ijk*	S*i* S*j* * IS*k*	Transmit the reciprocal iteration 2 – (S*j*) * (S*k*) to S*i*.	

Table 4-5. S Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
070*ij*0	S*i* /HS*j*	Transmit the floating-point reciprocal approximation of (S*j*) to S*i*.	Floating-point Operation
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 an unnormalized floating-point number.	
160*ijk*	V*i* S*j**FV*k*	Transmit the floating-point products of (S*j*) and (V*k* elements) to V*i* elements.	
162*ijk*	V*i* S*j**HV*k*	Transmit the half-precision rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements.	
170*ijk*	V*i* S*j* + FV*k*	Transmit the floating-point sums of (S*j*) and (V*k* elements) to V*i* elements.	
172*ijk*	V*i* S*j* – FV*k*	Transmit the floating-point differences of (S*j*) and (V*k* elements) to V*i* elements.	
044*ijk*	S*i* S*j*&S*k*	Transmit the logical product of (S*j*) and (S*k*) to S*i*.	Logical Operation
044*ij*0	S*i* S*j*&SB	Transmit the sign bit (bit 2^{63}) of (S*j*) to S*i*.	
045*ijk*	S*i* #S*k*&S*j*	Transmit the logical product of (S*j*) and the complement of (S*k*) to S*i*.	
045*ij*0	S*i* #SB&S*j*	Transmit (S*j*) with sign bit cleared to S*i*.	
046*ijk*	S*i* S*j*\S*k*	Transmit the exclusive OR of (S*j*) and (S*k*) to S*i*.	
046*ij*0	S*i* S*j*\SB	Toggle the sign bit of (S*j*), and transmit the result to S*i*.	
047*ijk*	S*i* #S*j*\S*k*	Transmit the logical equivalence of (S*j*) and (S*k*) to S*i*.	
047*ij*0	S*i* #S*j*\SB	Transmit the logical equivalence of (S*j*) and the sign bit to S*i*.	
050*ijk*	S*i* S*j*!S*i*&S*k*	Transmit the logical product of (S*i*) and (S*k*) complement ORed with the logical product of (S*j*) and (S*k*) to S*i*; merge (S*i*) and (S*j*) into S*i* using (S*k*) as the mask.	

Table 4-5. S Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
050*ij*0	S*i* S*j*!S*i*&SB	Transmit the scalar merge of (S*i*) and the sign bit of (S*j*) to S*i*.	Logical Operation
051*ijk*	S*i* S*j*!S*k*	Transmit the logical sum of (S*j*) and (S*k*) to S*i*.	
051*ij*0	S*i* S*j* !SB	Transmit the logical sum of (S*j*) and the sign bit to S*i*.	
140*ijk*	V*i* S*j*&V*k*	Transmit the logical products of (S*j*) and (V*k* elements) to V*i* elements.	
142*ijk*	V*i* S*j*!V*k*	Transmit the logical sums of (S*j*) and (V*k* elements) to V*i* elements.	
144*ijk*	V*i* S*j*\V*k*	Transmit the exclusive ORs of (S*j*) and (V*k* elements) to V*i* elements.	
052*ijk*	S0 S*i* < *exp*	Shift (S*i*) left *exp* places to S0; *exp* = *jk*.	Register Shift
053*ijk*	S0 S*i* > *exp*	Shift (S*i*) right *exp* places to S0; *exp* = $100_8 - jk$.	
054*ijk*	S*i* S*i* < *exp*	Shift (S*i*) left *exp* places to S*i*; *exp* = *jk*.	
055*ijk*	S*i* S*i* > *exp*	Shift (S*i*) right *exp* places to S*i*; *exp* = $100_8 - jk$.	
056*ijk*	S*i* S*i*,S*j* < A*k*	Shift (S*i*) and (S*j*) left (A*k*) places to S*i*.	
056*ij*0	S*i* S*i*,S*j* < 1	Shift (S*i*) and (S*j*) left one place to S*i*.	
056*i*0*k*	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 (A*k*) places to S*i*.	
057*ij*0	S*i* S*j*,S*i* > 1	Shift (S*j*) and (S*i*) right one place to S*i*.	
057*i*0*k*	S*i* S*i* > A*k*	Shift (S*i*) right (A*k*) places to S*i*.	
014*ijkm*	JSZ *exp*	Jump to *exp* if (S0) = 0 (*i*2 = 0). (Y-MP mode)	Conditional Jump
014000 *nm*	JSZ *exp*	Jump to *exp* if (S0) = 0. (C90 mode)	
015*ijkm*	JSN *exp*	Jump to *exp* if (S0) ≠ 0 (*i*2 = 0). (Y-MP mode)	
015000 *nm*	JSN *exp*	Jump to *exp* if (S0) ≠ 0. (C90 mode)	
016*ijkm*	JSP *exp*	Jump to *exp* if (S0) is positive; (S0) ≥ 0 (*i*2 = 0). (Y-MP mode)	

Table 4-5. S Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
016000 *nm*	JSP *exp*	Jump to *exp* if (S0) is positive; (S0) ≥ 0. (C90 mode)	Conditional Jump
017*ijkm*	JSM *exp*	Jump to *exp* if (S0) is negative (*i*2 = 0). (Y-MP mode)	
017000 *nm*	JSM *exp*	Jump to *exp* if (S0) is negative. (C90 mode)	

Intermediate Scalar (T) Registers

Sixty-four 64-bit T registers are designated $T0_8$ through $T77_8$. The T registers are used as intermediate storage registers for the S registers. Typically, T registers contain data to be referenced repeatedly over a long time period, making it inefficient to retain the data in either S registers or in central memory. Data transfers between a T register and an S register take 1 CP.

T register data transfers to or from central memory at a maximum rate of two words per CP. The 7 low-order bits of the contents of register A*i* specify the number of words to be transmitted. The first T register involved in the block transfer is specified by the *jk* fields of the instruction; successive data words transferred are to or from successive T registers until T77 is reached. Register T00 is processed after register T77 if more words need to be transferred. During these block transfers, a reservation is made on all T registers used in the block transfer. Other instructions can issue while T register data is being transferred to or from central memory.

T register data is protected with parity bits. When a word is written into a T register, a set of parity bits is generated and stored with the data bits. This set of parity bits is compared to another set generated when the word is read out of the T 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. Refer to "Status Registers" in Section 3 for additional information on parity errors.

Instructions reference T registers by specifying the T register number in the *jk* designator. Refer to "Instruction Formats" in Section 7 for more information on instruction fields. Table 4-6 summarizes the T register instructions.

Table 4-6. T Register Instructions

Machine Instruction	CAL Syntax	Description	Type of Instruction
074*ijk*	S*i* T*jk*	Transmit (T*jk*) to S*i*.	Interregister Transfer
075*ijk*	T*jk* S*i*	Transmit (S*i*) to T*jk*.	
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words from memory starting at address (A0) + (DBA) to T registers starting at register *jk*.	Block Transfer
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).	

S and T Register Troubleshooting

To assist in troubleshooting problems with the S and T registers, Figure 4-5 provides a block diagram of these registers showing the options involved and the signals that pass between them. For a more detailed description of this diagram, refer to the *CRAY Y-MP C90 Computer System Hardware Maintenance Manual*, publication number CMM-0502-000.

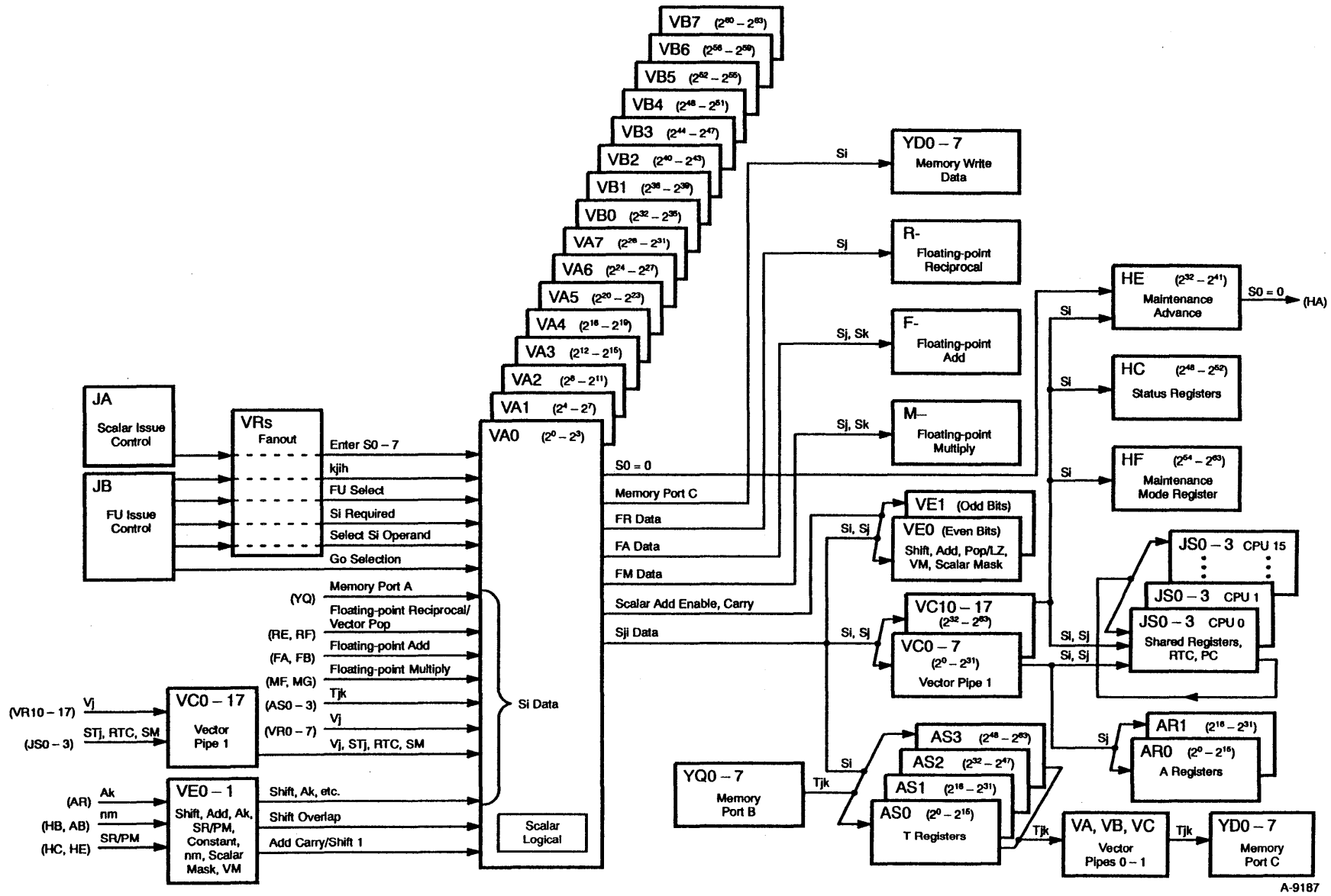


Figure 4-5. S and T Registers Troubleshooting Block Diagram

Vector (V) Registers

Figure 4-6 shows the eight V registers and their associated hardware. The V registers are designated V0 through V7. Each V register contains 128_{10} elements; each element can store 64 bits of data. The 128 elements are divided into two groups, called pipes, with one pipe processing the even-numbered elements and the other pipe processing the odd-numbered elements. Each pipe is supported by an identical set of functional units.

V register data is protected with parity bits. When a word is written into a V register, a set of parity bits is generated and stored with the data bits. This set of parity bits is compared to another set generated when the word is read out of the V 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. Refer to "Status Registers" in Section 3 for additional information on parity errors.

The V registers are used for vector processing. The following subsections explain vector processing, the V register functions, the V register instructions, and vector chaining.

Vector Processing

Vector processing 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.

A vector is an ordered set of elements; each element 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, two successive pairs of elements are processed each CP. The dual vector pipes and the dual sets of vector functional units allow a pair of even-numbered elements and a pair of odd-numbered elements to be processed during the same CP. As each pair of operations is completed, the results are delivered to successive even- or odd-numbered elements of the result register. The vector operation continues until the number of elements processed by the instruction equals the count specified by the vector length (VL) register.

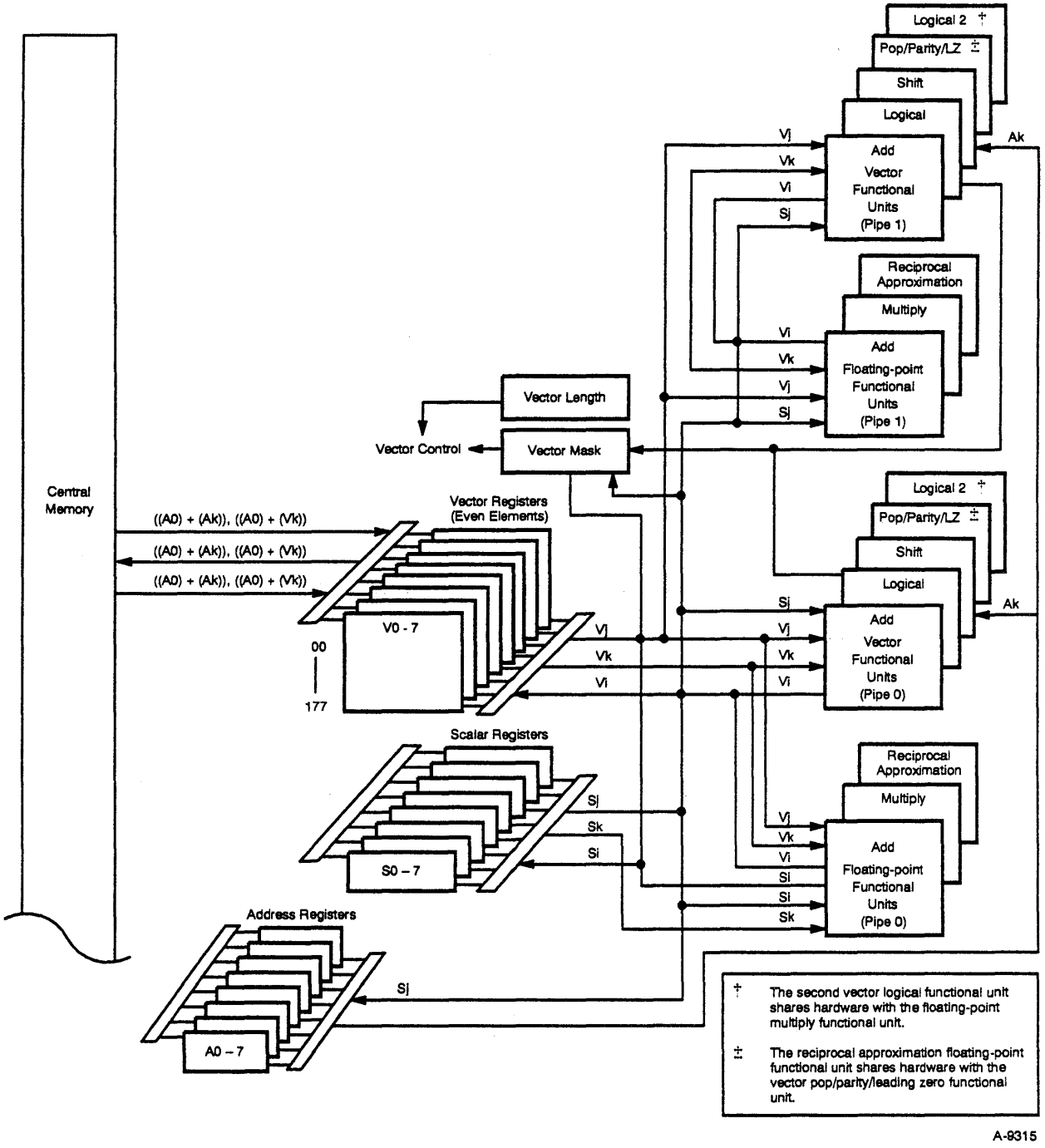


Figure 4-6. V Register Block Diagram

Parallel vector operations allow the generation of more than two results per CP. Parallel vector operations occur automatically in the following situations:

- When successive vector instructions use different functional units and different V registers.
- When successive vector instructions use the result stream from one vector register as the operand of another operation using a different functional unit. This process is known as *chaining* and is explained later in this subsection.

Advantages of Vector Processing

In general, vector processing is faster and more efficient than scalar processing. Vector processing reduces the extra time and storage space associated with maintenance of the loop-control variable (for example, incrementing and checking the count). In many cases, loops processed as vectors are reduced to a simple sequence of instructions without branching backwards. Central memory access conflicts are reduced, and finally, functional unit segmentation is exploited through vector processing because results from the units can then be obtained at the rate of two results per CP.

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

V Register Functions

The V registers are used solely for vector processing, unlike the A and S registers, which are used for many secondary functions. Vector processing allows a single instruction to perform a specified operation sequentially on a set (vector) of operands, to produce a vector of results. Examples of these sets or vectors may be rows or columns of a matrix or elements of a table.

The contents of a V register are transferred to or from central memory through a block transfer. A vector block transfer is accomplished by specifying a first word address in central memory, an increment or decrement value for the central memory address, and a vector length. The transfer begins with the first element of the V register and proceeds at a maximum rate of two words per clock period (CP); this rate can be affected by central memory conflicts. Central memory conflicts interrupt

the vector data stream and can occur in chained operations (although they do not inhibit chaining). Any interruption in the vector data stream adds proportionally to the total execution time of vector operations.

Single-word data transfers can also be made between an S register and an element of a V register.

Vector Instructions

Vector instructions reference V registers by specifying the register number in the *i*, *j*, or *k* field of the instruction. Refer to "Instruction Formats" in Section 7 for information on instruction fields. Operations on vector registers always start with element 0. Individual elements of a V register are designated by octal numbers ranging from 00 through 177. These numbers appear as subscripts to vector register references. For example, $V6_{27}$ refers to element 27 of V register 6.

Vector instructions reserve V registers as either operands or results. If the register is reserved as an operand, it cannot be used as an operand or result until the operand reservation clears. A vector register can be used as both an operand and result register for the same vector instruction. If a register is reserved as a result, it can be used as an operand through a process called chaining. Refer to "Vector Chaining" in this section for more information on chaining.

No reservation is placed on the VL register during vector processing. If a vector instruction uses an S register as an operand, no reservation is placed on the S register. Conflicts can occur between vector and scalar operations involving floating-point operations and memory access. With the exception of these operations, the floating-point functional units are always available for scalar operations. The S and VL registers can be modified after the vector instruction issues without affecting the vector operation. The A0 and A*k* registers in a vector memory reference can also be modified after the instruction issues.

Because most transfers to or from registers are done in blocks of data, instructions that transfer data between V registers and central memory reserve a port, and functional unit instructions reserve the appropriate functional unit.

Table 4-7 summarizes the types of V register instructions and shows the machine instruction, the CAL code, a description of the instruction, and the type of instruction. Refer to "CPU Instruction Descriptions" in Section 7 for a detailed description of these instructions.

Table 4-7. V Register Instructions

Machine Instruction	CAL Syntax	Description	Type of Instruction
076*ijk*	S*i* V*j*,A*k*	Transmit (V*j* element (A*k*)) to S*i*.	Register Entry
077*ijk*	V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*).	
077*i*0*k*	V*i*,A*k* 0	Clear element (A*k*) of register V*i*.	
176*i*0*k*	V*i* ,A0,A*k*	Read (VL) words to V*i* starting at address (A0) + (DBA), incrementing by (A*k*).	Memory Transfer
176*i*00	V*i* ,A0,1	Read (VL) words to V*i* starting at address (A0) + (DBA), incrementing by 1.	
176*i*1*k*	V*i* ,A0,V*k*	Read (VL) words to V*i* using memory addresses ((A0) + (V*k*) + (DBA)).	
1770*jk*	,A0,A*k* V*j*	Write (VL) words from (V*j*) to memory starting at (A0) + (DBA), incrementing by (A*k*).	
1770*j*0	,A0,1 V*j*	Write (VL) words from (V*j*) to memory starting at address (A0) + (DBA), incrementing by 1.	
1771*jk*	,A0,V*k* V*j*	Write (VL) words from (V*j*) to memory using memory addresses ((A0) + (V*k*) + (DBA)).	
154*ijk*	V*i* S*j* + V*k*	Transmit the integer sums of (S*j*) and (V*k* elements) to V*i* elements.	Integer Operation
155*ijk*	V*i* V*j* + V*k*	Transmit the integer sums of (V*j* elements) and (V*k* elements) to V*i* elements.	
156*ijk*	V*i* S*j* – V*k*	Transmit the integer differences of (S*j*) and (V*k* elements) to V*i* elements.	
157*ijk*	V*i* V*j* – V*k*	Transmit the integer differences of (V*j* elements) and (V*k* elements) to V*i* elements.	
174*ij*3	V*i* ZV*j*	Transmit the leading zero count of (V*j* elements) to V*i* elements.	
160*ijk*	V*i* S*j**FV*k*	Transmit the floating-point products of (S*j*) and (V*k* elements) to V*i* elements.	Floating-point Operation
161*ijk*	V*i* V*j**FV*k*	Transmit the floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements.	
162*ijk*	V*i* S*j**HV*k*	Transmit the half-precision rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements.	

Table 4-7. V Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
163*ijk*	V*i* V*j**HV*k*	Transmit the half-precision rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements.	Floating-point Operation
164*ijk*	V*i* S*j**RV*k*	Transmit the rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements.	
165*ijk*	V*i* V*j**RV*k*	Transmit the rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements.	
166*ijk*	V*i* S*j**V*k*	Transmit the 32-bit integer products of (S*j*) and (V*k* elements) to V*i* elements.	
167*ijk*	V*i* V*j**V*k*	Transmit the reciprocal iterations 2 – (V*j* elements) * (V*k* elements) to V*i* elements.	
170*ijk*	V*i* S*j* + FV*k*	Transmit the floating-point sums of (S*j*) and (V*k* elements) to V*i* elements.	
170*i*0*k*	V*i* + FV*k*	Transmit the normalized (V*k* elements) to V*i* elements.	
171*ijk*	V*i* V*j* + FV*k*	Transmit the floating-point sums of (V*j* elements) and (V*k* elements) to V*i* elements.	
172*ijk*	V*i* S*j* – FV*k*	Transmit the floating-point differences of (S*j*) and (V*k* elements) to V*i* elements.	
172*i*0*k*	V*i* – FV*k*	Transmit the normalized negatives of (V*k* elements) to V*i* elements.	
173*ijk*	V*i* V*j* – FV*k*	Transmit the floating-point differences of (V*j* elements) and (V*k* elements) to V*i* elements.	
174*ij*0	V*i* /HV*j*	Transmit the floating-point reciprocal approximation of (V*j* elements) to V*i* elements.	
140*ijk*	V*i* S*j*&V*k*	Transmit the logical products of (S*j*) and (V*k* elements) to V*i* elements.	Logical Operation
141*ijk*	V*i* V*j*&V*k*	Transmit the logical products of (V*j* elements) and (V*k* elements) to V*i* elements.	
142*ijk*	V*i* S*j*!V*k*	Transmit the logical sums of (S*j*) and (V*k* elements) to V*i* elements.	
142*i*0*k*	V*i* V*k*	Transmit (V*k* elements) to V*i* elements.	

Table 4-7. V Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
143*ijk*	V*i* V*j*!V*k*	Transmit the logical sums of (V*j* elements) and (V*k* elements) to V*i* elements.	Logical Operation
144*ijk*	V*i* S*j*\V*k*	Transmit the exclusive ORs of (S*j*) and (V*k* elements) to V*i* elements.	
145*ijk*	V*i* V*j*!Vk	Transmit the exclusive ORs of (V*j* elements) and (V*k* elements) to V*i* elements.	
146*ijk*	V*i* S*j*!V*k*&VM	Transmit (S*j*) if VM bit = 1, or (V*k* element) if VM bit = 0, to V*i* elements.	
146*i*0*k*	V*i* #VM&V*k*	Transmit the vector merge of (V*k* elements) and 0 to V*i* elements.	
147*ijk*	V*i* V*j*!V*k*&VM	Transmit (V*j* elements) if VM bit = 1, or (V*k* elements) if VM bit = 0, to V*i* elements.	
150*ijk*	V*i* V*j* < A*k*	Shift (V*j* elements) left (A*k*) places to V*i* elements.	Register Shift
150*ij*0	V*i* V*j* < 1	Shift (V*j* elements) left one place to Vi elements.	
005400 150*ij*0	V*i* V*j* < V0	Shift (V*j* elements) left (V0) places to V*i* elements.	
151*ijk*	V*i* V*j* > A*k*	Shift (V*j* elements) right (A*k*) places to V*i* elements.	
005400 151*ij*0	V*i* V*j* > V0	Shift (V*j* elements) right (V0) places to V*i* elements.	
151*ij*0	V*i* V*j* > 1	Shift (V*j* elements) right one place to Vi elements.	
152*ijk*	V*i* V*j*,V*j* < A*k*	Double shift (V*j* elements) left (A*k*) places to V*i* elements.	
152*ij*0	V*i* V*j*,V*j* < 1	Double shift (V*j* elements) left one place to V*i* elements.	
005400 152*ijk*	V*i* V*j*,A*k*	Transfer (V*j* elements) to V*i* elements starting at element (A*k*).	

Table 4-7. V Register Instructions (continued)

Machine Instruction	CAL Syntax	Description	Type of Instruction
153*ijk*	V*i* V*j*,V*j* > A*k*	Double shift (V*j* elements) right (A*k*) places to V*i* elements.	Register Shift
153*ij*0	V*i* V*j*,V*j* > 1	Double shift (V*j* elements) right one place to V*i* elements.	

Vector Chaining

The CRAY Y-MP C90 computer system 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 because of 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 arrival time of the result data stream. For full chaining to occur, the chaining instruction must issue 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 of the result vector data stream.

Figure 4-7 shows how the results of four instructions are chained together. The instruction chaining sequence comprises the following operations:

1. Reads a vector of integers from central memory to register V0.
2. Adds the contents of register V0 to the contents of register V1 and sends the results to register V2.
3. Shifts the results obtained in Step 2 and sends them to register V3.
4. Forms the logical product of the shifted sum obtained in Step 3 with the contents of register V4 and sends the results to register V5.

As soon as the first two elements from central memory arrive at register V0, they are added to the first two elements of vector register V1. Subsequent pairs of elements are pipelined through the segmented functional unit, so a continuous stream of results is sent to the destination register, which is register V2. As soon as the first two elements arrive at

register V2, they are used as operands for the shift operation. The results are sent to register V3, which immediately becomes the source of one of the operands necessary for the logical operation between registers V3 and V4. The results of the logical operation are then sent to register V5.

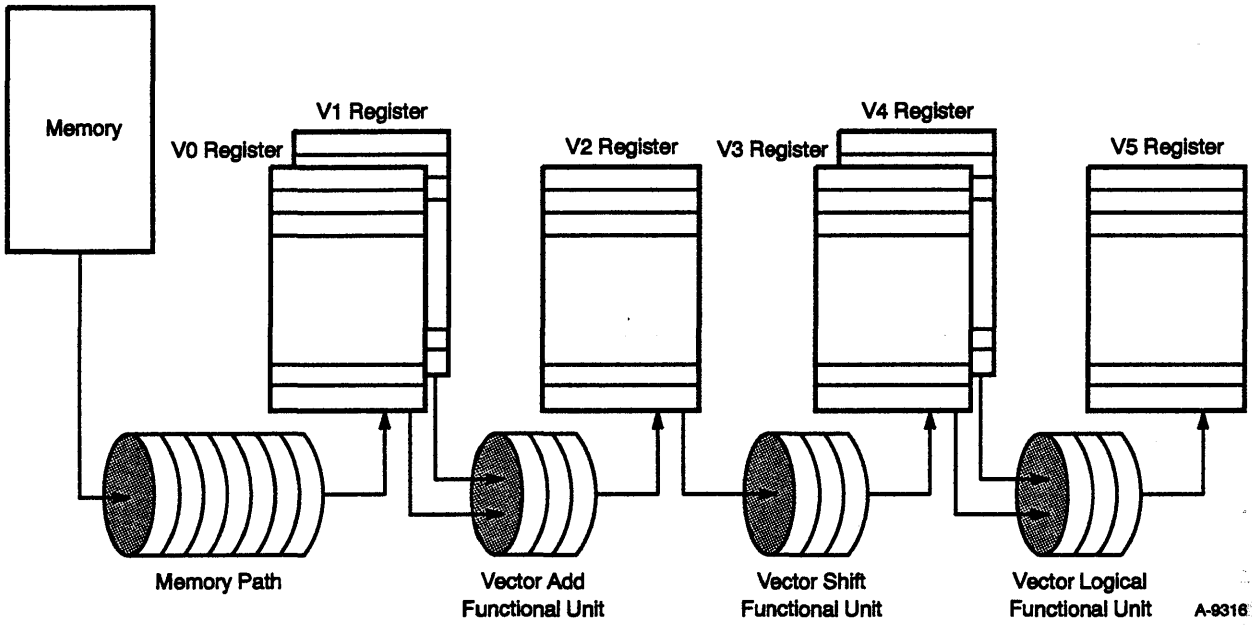


Figure 4-7. Vector Chaining Example

Vector Control Registers

The vector length (VL) register and vector mask (VM) register provide control information needed in the performance of vector operations and are described in the following subsections.

Vector Length Register

The 8-bit VL register can be set to any value from 1_8 through 200_8 (a value of 0 results in VL = 200_8). The value stored in this register specifies the number of vector operations performed by the vector instructions, 140*ijk* through 177*ijk*. The VL register is loaded and saved during an exchange sequence. The VL register can also be loaded by instruction 00200*k* or read by instruction 023*i*01.

Vector Mask Register

The VM register can store 128 bits; each bit corresponds to an element of a vector register. Bit 2^{127} corresponds to element 0, bit 2^0 to element 127. The mask is used to allow operations to be performed on selected

vector elements in a vector merge and to store the results of vector test instructions. Table 4-8 lists the VM register instructions and shows octal and CAL codes. Refer to "CPU Instruction Summary" in Section 7 for complete information on these instructions.

Table 4-8. Vector Mask Instructions

Machine Instruction	CAL Syntax	Description	Type of Instruction
0030*j*0	VM S*j*	Transmit (S*j*) to VM lower register.	Register Entry
0030*j*1	VM1 S*j*	Transmit (S*j*) to VM upper register.	
003000	VM 0	Clear VM register.	
073*i*00	S*i* VM	Transmit (VM) to S*i*.	
146*ijk*	V*i* S*j*!V*k*&VM	Transmit (S*j*) if VM bit = 1, or (V*k* element) if VM bit = 0, to V*i* elements.	Logical Operation
146*i*0*k*	V*i* #VM&V*k*	Transmit the vector merge of (V*k* elements) and 0 to V*i* elements.	
147*ijk*	V*i* V*j*!V*k*&VM	Transmit (V*j* elements) if VM bit = 1, or (V*k* elements) if VM bit = 0, to V*i* elements.	
1750*j*0	VM V*j*,Z	Set VM bit if (V*j* element) = 0.	
1750*j*1	VM V*j*,N	Set VM bit if (V*j* element) ≠ 0.	
1750*j*2	VM V*j*,P	Set VM bit if (V*j* element) ≥ 0.	
1750*j*3	VM V*j*,M	Set VM bit if (V*j* element) < 0.	
175*ij*4	V*i*,VM V*j*,Z	Set VM bit if (V*j* element) = 0; also, store the compressed indices of the V*j* elements = 0 in the V*i* elements.	
175*ij*5	V*i*,VM V*j*,N	Set VM bit if (V*j* element) ≠ 0; also, store the compressed indices of the V*j* elements ≠ 0 in the V*i* elements.	
175*ij*6	V*i*,VM V*j*,P	Set VM bit if (V*j* element) ≥ 0; also, store the compressed indices of the V*j* elements ≥ 0 in the V*i* elements.	
175*ij*7	V*i*,VM V*j*,M	Set VM bit if (V*j* element) < 0; also, store the compressed indices of the V*j* elements < 0 in the V*i* elements.	

The contents of the VM register can be set from an S register through instructions 0030*j*0 and 0030*j*1, or they can be created by testing the elements of a vector register for a specified condition using instruction

175*ijk*. The mask controls element selection in the vector merge instructions (146*ijk* and 147*ijk*). Instruction 073*i*00 reads the contents of the VM register to an S register.

V Register Troubleshooting

To assist in troubleshooting problems with the V registers, Figure 4-8 provides a block diagram of these registers showing the options involved and the signals that pass between them. For a more detailed description of this diagram, refer to the *CRAY Y-MP C90 Computer System Hardware Maintenance Manual*, publication number CMM-0502-000.

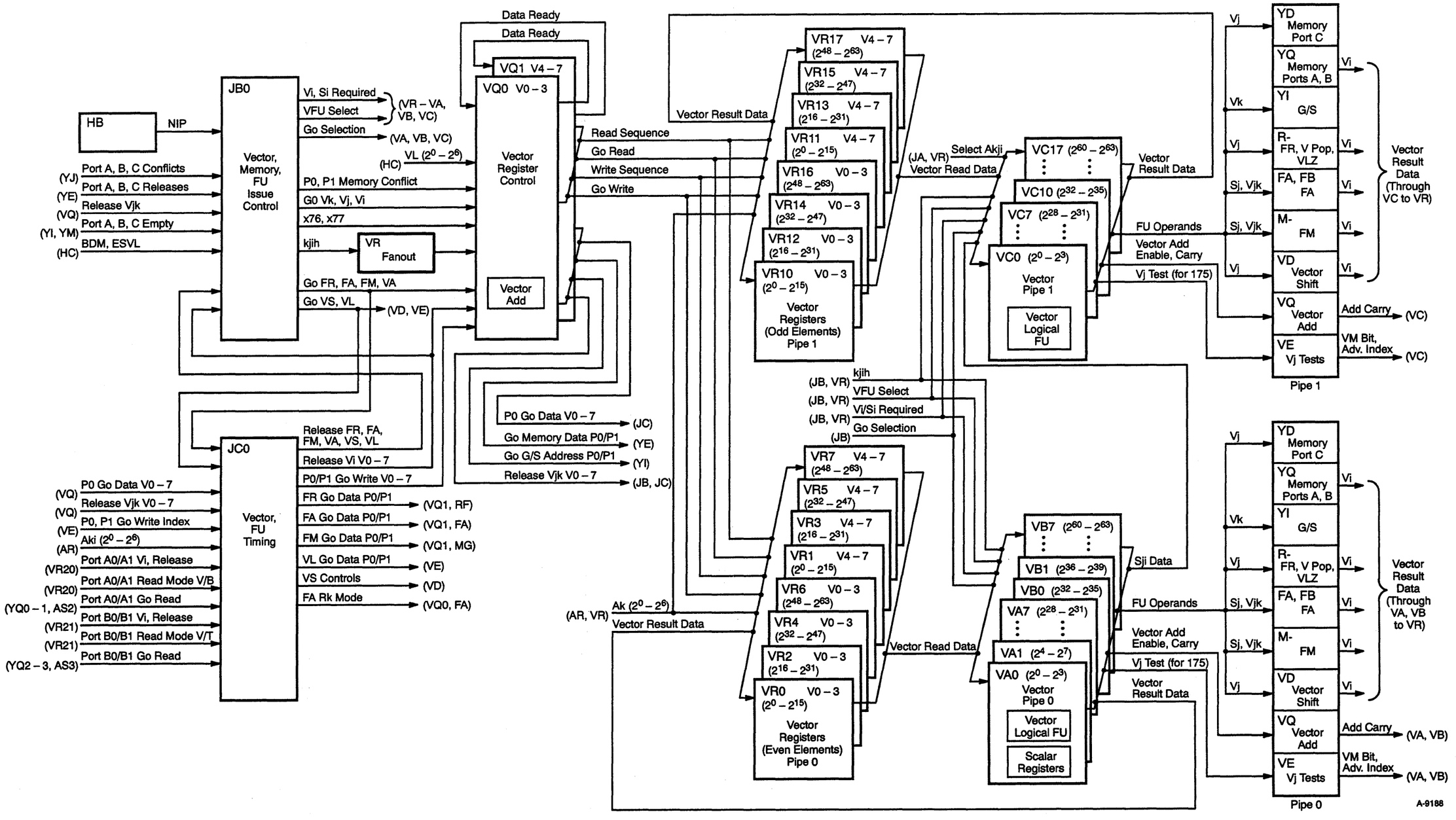


Figure 4-8. Vector Registers Troubleshooting Block Diagram

Functional Units

Functional units perform instructions other than simple transfers or control operations. Functional units have independent logic except for the reciprocal approximation, vector population count, and vector leading zero units, and the floating-point multiply and second vector logical units, which share some logic (described later in this section). All functional units can operate simultaneously. For more information, refer to "Functional Unit Independence" in Section 5.

A functional unit receives operands from registers, performs a specific operation on them, and delivers the result to a register when the operation is completed. Functional units operate generally in three-address mode with source and destination addressing limited to register designators.

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

Functional units are fully segmented, which means a new set of operands for unrelated computation can enter a functional unit each CP, even though the functional unit time is more than 1 CP. Refer to "Pipelining and Segmentation" in Section 5 for more information on segmentation.

There are four groups of functional units: address, scalar, vector, and floating-point. The address, scalar, and vector functional units operate with one of the primary register types (A, S, and V) to support address, scalar, and vector processing. The floating-point functional units support either scalar or vector operations and accept operands from or deliver results to S or V registers. For timing purposes, central memory can also act as a functional unit for vector operations.

The following subsections define the functions, the functional unit times, and the instructions executed by each functional unit. Refer to the following headings for additional information on functional units:

- "Pipelining and Segmentation" and "Functional Unit Independence" in Section 5 contain detailed information on functional unit segmentation and independence.
- "Functional Unit Operations" in this section contains detailed information on integer arithmetic, floating-point arithmetic, normalized floating-point numbers, floating-point range errors, addition algorithm, multiply algorithm, and the division algorithm.
- "CPU Instruction Descriptions" in Section 7 contains detailed information on the instructions and instruction formats.

Address Functional Units

Address functional units perform integer arithmetic on operands obtained from A registers and deliver the results to an A register.

Address Add Functional Unit

The address add functional unit performs 32-bit integer addition and subtraction. The unit executes instructions 030*ijk* (addition) and 031*ijk* (subtraction). The subtraction operation uses two's complement arithmetic. The A*k* operand is complemented then added to the A*j* operand. A 1 is added to the low-order bit position of the result. The address add functional unit does not detect overflow.

The address add functional unit time is 1 CP. Two CPs elapse from instruction issue to the time the result register can be referenced again.

Address Multiply Functional Unit

The address multiply functional unit performs 32-bit integer multiplication. The unit executes instruction 032*ijk* forming a 32-bit integer product from two operands. No rounding is performed. The result consists of the least significant 32 bits of the product. The address multiply functional unit does not detect overflow.

The address multiply functional unit time is 3 CPs. Four CPs elapse from instruction issue to the time the result register can be referenced again.

Scalar Functional Units

Scalar functional units perform operations on 64-bit operands obtained from S registers and usually deliver the 64-bit results to an S register. The exception is the population/parity/leading zero count functional units that deliver a 7-bit result to an A register.

Four functional units are exclusively associated with scalar operations and are described below. Three floating-point functional units are used for both scalar and vector operations. When a scalar instruction uses a floating-point functional unit, it must determine whether that functional unit is reserved by a vector operation. If the functional unit is reserved, the scalar instruction holds issue until the functional unit reservation is cleared. Refer to "Floating-point Functional Units" in this section for more information on these functional units.

Scalar Add Functional Unit

The scalar add functional unit performs 64-bit integer addition and subtraction. It executes instructions 060*ijk* (addition) and 061*ijk* (subtraction). The subtraction operation uses two's complement arithmetic. The S*k* operand is complemented and added to the S*j* operand, and a 1 is added to the low-order bit position of the result. The scalar add functional unit does not detect overflow.

The scalar add functional unit time is 2 CPs. Three CPs elapse from instruction issue to the time the result register can be referenced again.

Scalar Shift Functional Unit

The scalar shift functional unit shifts the entire 64-bit contents of an S register (single shift) or shifts the 128-bit contents of two concatenated S registers (double shift). For a single shift (instructions 052*ijk* through 055*ijk*), the shift count is specified by the *jk* field. For a double shift (instructions 056*ijk* and 057*ijk*), the A*k* register must contain a valid shift count in the 7 lower bit positions. Any bits set in the upper 25 positions cause the result register S*i* to be zeroed out.

All single and double shifts are end-off with zero fill. A circular shift is performed by the double shift instructions if the shift count does not exceed 64 and if the *i* and *j* designators are equal and nonzero.

Single-shift instructions have a functional unit time of 2 CPs, and the result register is available in 3 CPs. Double-shift instructions have a functional unit time of 3 CPs, and the result register is available in 4 CPs.

Scalar Logical Functional Unit

The scalar logical functional unit performs bit-by-bit manipulation of 64-bit quantities obtained from S registers. It executes instructions 042*ijk* through 043*ijk* (mask) and 044*ijk* through 051*ijk* (logical operations).

The scalar logical functional unit time is less than 1 CP. One CP elapses from instruction issue to the time the result register can be referenced again.

Scalar Population/Parity/Leading Zero Functional Unit

This functional unit performs instructions 026*ij*0 and 026*ij*1 (population count and population count parity) and 027*ij*0 (leading zero count). Instruction 026*ij*0 counts the number of bits in the S*j* register having a value of 1 and returns the 7-bit result to the A*i* register; the maximum count is 100_8 (64_{10}), while the minimum count is 0.

Instruction 026*ij*1 counts the number of bits in the S*j* operand having a value of 1, but returns only a 1-bit parity count to the A*i* register. If the S*j* operand has an even number of bits set, a 0 is returned to the A*i* register. If the S*j* operand has an odd number of bits set, a 1 is returned to the A*i* register.

The functional unit time for the population count and population count parity operations is 4 CPs. Six CPs elapse from instruction issue to the time the result register can be referenced again.

Instruction 027*ij*0 counts the number of 0 bits preceding the first 1 bit in the operand. For these instructions, the 64-bit operand is obtained from an S register and the 7-bit result is delivered to an A register.

The functional unit time for the leading zero count operation is 4 CPs. Six CPs elapse from instruction issue to the time the result register can be referenced again.

Vector Functional Units

There are two parallel sets of vector functional units referred to as pipe 0 and pipe 1. Pipe 0 processes the even-numbered elements of a vector, while pipe 1 processes the odd-numbered elements. This duplication of functional units allows two pairs of elements to be processed at the same time and increases the efficiency of the vector processing operations.

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

Successive even- and odd-numbered 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 fixed functional unit time. The VL register determines the number of operand pairs to be processed by a functional unit.

The functional units described in this section are exclusively associated with vector operations. Three functional units are associated with both vector operations and scalar operations. Refer to "Floating-point Functional Units" later in this section for more information.

Vector Add Functional Unit

The vector add functional unit performs 64-bit integer addition and subtraction for a vector operation and delivers the results to elements of a V register. The unit executes instructions 154*ijk* and 155*ijk* (addition) and 156*ijk* and 157*ijk* (subtraction). Instructions 154*ijk* and 156*ijk* use scalar register operands. The subtraction operation uses two's complement arithmetic. The V*k* operand is complemented and added to the V*j* operand, and a 1 is added to the low-order bit position of the result. The vector add functional unit does not detect overflow.

The vector add functional unit time is 4 CPs. If no conflicts cause a delay in the operation, the result register can be referenced as an operand in 6 CPs and as a result register in (VL)/2 + 7 CPs.

Vector Shift Functional Unit

The vector shift functional unit shifts the entire 64-bit contents of a vector register element (single-shift) or the 128-bit value formed from two consecutive elements of a V register (double shift). Shift counts are obtained from an A register and are end-off with zero fill. All shift counts are considered positive unsigned integers. If any bit of the A register higher than bit 2^6 is set, the shifted result is all 0's.

The vector shift functional unit executes instructions 150*ijk* and 151*ijk* (single shift) and instructions 152*ijk* and 153*ijk* (double shift). The functional unit time is 4 CPs for instructions 150*ijk*, 151*ijk*, and 153*ijk*, and it is 5 CPs for instruction 152*ijk*. For instructions 150*ijk*, 151*ijk*, and 153*ijk*, if no conflicts cause a delay in the operation, the result register can be referenced as an operand in 6 CPs and as a result register in (VL)/2 + 7 CPs. For instruction 152*ijk*, if no conflicts cause a delay in the operation, the result register can be referenced as an operand in 7 CPs and as a result register in (VL)/2 + 8 CPs.

Full Vector Logical Functional Unit

The full vector logical functional unit performs a bit-by-bit manipulation of the 64-bit quantities for instructions 140*ijk* through 145*ijk*. The full vector logical functional unit also performs the logical operations associated with the vector mask instructions 146*ijk*, 147*ijk*, and 175*ijk*.

The full vector logical functional unit time is 2 CPs for instructions 140*ijk* through 147*ijk* and 5 CPs for instruction 175*ijk*. For instructions 140*ijk* through 147*ijk*, if no conflicts cause a delay in the operation, the result register can be referenced as an operand in 4 CPs and as a result register in (VL)/2 + 5 CPs. For instruction 175*ijk*, if no conflicts cause a delay in the operation, the result register can be referenced as an operand in 8 CPs and as a result register in (VL)/2 + 9 CPs.

Second Vector Logical Functional Unit

The second vector logical functional unit, when enabled, performs the same bit-by-bit manipulations as the full vector logical functional unit for instructions 140*ijk* through 145*ijk*. However, instructions 146*ijk*, 147*ijk*, and 175*ijk* cannot execute in this functional unit. This functional unit can be enabled by setting the enable second vector logical (ESL) bit in the modes section of the exchange package. When the second vector logical functional unit is enabled, it has priority for the processing of instructions 140*ijk* through 145*ijk*. When the second vector logical functional unit is busy or is disabled, instructions 140*ijk* through 145*ijk* are processed in the full vector logical functional unit.

The second vector logical and floating-point multiply functional units cannot be used simultaneously because they share input and output data paths. Also, because these two units share paths, some codes that rely on floating-point products may run slower if the second vector logical functional unit is enabled. If the floating-point multiply unit is busy, and the full vector logical unit is not busy, a vector logical instruction uses the full vector logical functional unit.

The second vector logical functional unit time is 2 CPs. If no conflicts cause a delay in the operation, the result register can be referenced as an operand in 4 CPs and as a result register in (VL)/2 + 5 CPs.

Vector Population/Parity/Leading Zero Functional Unit

The vector population/parity/leading zero functional unit performs population counts, parity checks, and leading zero counts for vector operations. It executes instructions 174*ij*1 (vector population count), 174*ij*2 (vector population count parity), and 174*ij*3 (vector leading zero count). This functional unit shares some logic with the reciprocal approximation functional unit. Therefore, the *k* field must be nonzero for the instructions to be recognized as population/parity/leading zero instructions.

Instruction 174*ij*1 counts the 1 bits in each element of the V*j* register and returns this number to the respective elements of the V*i* register; the total number of 1 bits is the population count. This population count can be an odd or an even number, as indicated by its low-order bit.

Instruction 174*ij*2 counts the number of 1 bits in each element of the V*j* register and returns a 1-bit parity result to the respective elements of the V*i* register. Parity can be odd (signified by a 1) or even (signified by a 0).

Instruction 174*ij*3 counts the number of 0 bits preceding the first 1 bit in each element of the V*j* register and returns this number to the respective elements of the V*i* register.

The vector population/parity/leading zero functional unit time is 5 CPs. If no conflicts cause a delay in the operation, the result register can be referenced as an operand in 7 CPs and as a result register in (VL)/2 + 8 CPs.

Floating-point Functional Units

There are two parallel sets of floating-point functional units, with each set containing three functional units. These floating-point functional units perform floating-point arithmetic for both scalar and vector operations. The vector registers use both sets of functional units, with one set processing the even-numbered elements while the other set processes the odd-numbered elements. For an operation involving only scalar operands, only one set of floating-point functional units is used.

When executing vector instructions, operands are obtained from V registers, or from an S register and a V register, and results are delivered to a vector register. When a floating-point functional unit is used for a vector operation, the general description of vector functional units applies. When executing a scalar instruction, operands are obtained from S registers, and results are delivered to an S register.

Floating-point Add Functional Unit

The floating-point add functional unit performs addition and subtraction of 64-bit operands in floating-point format. It executes instructions 062*ijk* (scalar add), 063*ijk* (scalar subtract), and 170*ijk* through 173*ijk* (vector add and subtract). The result is normalized even when operands are not normalized. The floating-point add functional unit detects overflow and underflow conditions; only overflow conditions are flagged.

The floating-point add functional unit time is 6 CPs. For the scalar instructions (062*ijk* and 063*ijk*), 6 CPs elapse from instruction issue to the time the result register can be referenced again. For the vector instructions (170*ijk* through 173*ijk*), if no conflicts cause a delay in the operation, the result register can be referenced as an operand in 9 CPs and as a result register in (VL)/2 + 10 CPs.

Floating-point Multiply Functional Unit

The floating-point multiply functional unit performs full- and half-precision multiplication of 64-bit operands in floating-point format. It executes instructions 064*ijk* through 067*ijk* (scalar multiplication) and instructions 160*ijk* through 167*ijk* (vector multiplication). The half-precision product is rounded; the full-precision product can be rounded or not.

Instruction 166*ijk* computes the 32-bit integer product of the contents of the S*j* register and the elements of the V*k* register and transmits the results to the elements of the V*i* register.

The floating-point multiply and second vector logical functional units cannot be used simultaneously because they share control hardware. If one of these functional units is reserved, the other functional unit is also reserved.

Input operands must be normalized; the floating-point multiply functional unit only delivers a normalized result if both input operands are normalized. The floating-point multiply functional unit detects overflow and underflow conditions; only overflow conditions are flagged.

The floating-point multiply functional unit time is 6 CPs. For the scalar instructions (064*ijk* through 067*ijk*), six CPs elapse from instruction issue to the time the result register can be referenced again. For the vector instructions (160*ijk* through 167*ijk*), if no conflicts cause a delay in the operation, the result register can be referenced as an operand in 9 CPs and as a result register in (VL)/2 + 10 CPs.

The floating-point multiply functional unit recognizes both operands having zero exponents as a special case, and performs an integer multiply operation. The result is considered as an integer product, is not normalized, and is not considered out of range. This case illustrates a fast method of computing a 48-bit integer product, although the operands in this case must be shifted before the multiply operation. Refer to "Integer Arithmetic" in this section for more information on integer multiplication.

Reciprocal Approximation Functional Unit

The reciprocal approximation functional unit finds the approximate reciprocal of a 64-bit operand in floating-point format. The unit executes instructions 070*ij*0 and 174*ij*0. Because the vector population/parity/leading zero functional unit shares some logic with this unit, the *k* field must be 0 for the instruction to be recognized as a reciprocal approximation instruction.

The input operand must be normalized; the floating-point reciprocal approximation functional unit delivers a correct result only if the input operand is normalized. The high-order bit of the coefficient is not tested but is assumed to be a 1. The floating-point reciprocal approximation functional unit detects overflow and underflow conditions; both conditions are flagged.

The reciprocal approximation functional unit time is 10 CPs. For the scalar instruction (070*ij*0), 10 CPs elapse from instruction issue to the time the result register can be referenced again. For the vector instruction (174*ij*0), if no conflicts cause a delay in the operation, the result register can be referenced as an operand in 13 CPs and as a result register in (VL)/2 + 14 CPs.

Functional Unit Operations

Functional units in a CPU perform logical operations, integer arithmetic, and floating-point arithmetic. Integer and floating-point arithmetic are performed in two's complement. The following subsections explain the logical operations, the integer arithmetic, and the floating-point arithmetic used by the CRAY Y-MP C90 computer system.

Logical Operations

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

A logical product is the AND function, which is shown in the following example:

```
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, which is shown in the following example:

```
Operand 1: 1 0 1 0
Operand 2: 1 1 0 0
Result :   1 1 1 0
```

A logical exclusive OR function is shown in the following example:

```
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, which is shown in the following example:

```
Operand 1: 1 0 1 0
Operand 2: 1 1 0 0
Result:    1 0 0 1
```

The scalar merge operation uses two scalar operands and a mask to produce results. 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. The following example shows a scalar merge operation:

```
Operand 1: 1 0 1 0 1 0 1 0
Operand 2: 1 1 0 0 1 1 0 0
Mask:      1 1 1 1 0 0 0 0
Result:    1 0 1 0 1 1 0 0
```

The vector merge operation uses two vector operands and a mask to produce results. The operation is similar to the scalar merge except that the elements of vector operands replace the bits of the scalar operands. The following example shows a vector merge operation:

```
Elements of   (00) = 1        Elements of   (00) = –1
Operand 1:    (01) = 2        Operand 2:    (01) = –2
              (02) = 3                      (02) = –3
              (03) = 4                      (03) = –4
Mask: 0110
Elements of result operand:   (00) = –1
                              (01) = 2
                              (02) = 3
                              (03) = –4
```

Integer Arithmetic

All integers, whether 32 or 64 bits long, are represented in the registers as shown in Figure 4-9. The address add and address multiply functional units perform 32-bit arithmetic. The scalar add and vector add functional units perform 64-bit arithmetic.

Two scalar (64-bit) integer operands are multiplied using the floating-point multiply instruction and one of two 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 multiplication operation and the method used for operands greater than 24 bits.

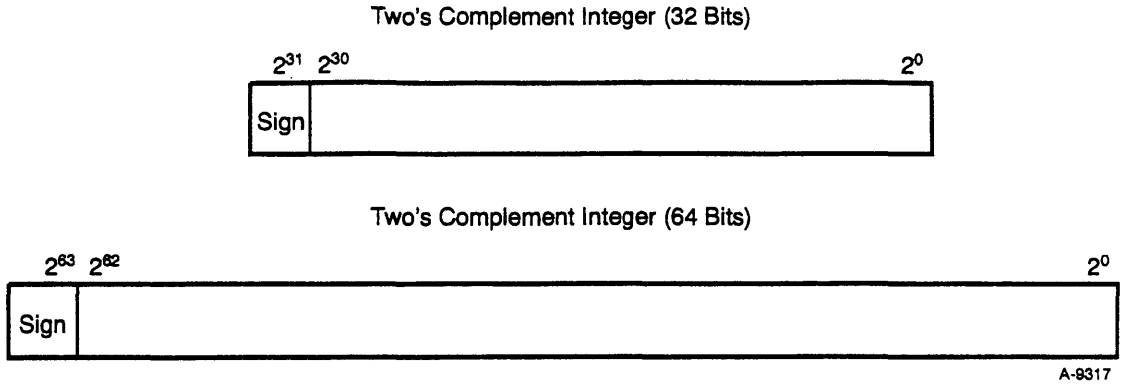


Figure 4-9. Integer Data Formats

The floating-point multiply functional unit detects a condition in which both operands have zero exponents as a special case; this condition is treated as an integer multiplication operation, and a complete multiplication operation 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 integer numbers together, set each operand's exponent (bits 2^{62} through 2^{48}) equal to 0 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 24 least significant bits must be 0.

When the floating-point multiply functional unit performs the operation, it returns the 48 high-order 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. If the 24 least significant bits of the operand coefficients were nonzero, the 48 low-order bits of the product could be nonzero and could generate a carry into the least significant of the 48 high-order bits returned, causing the result to be one larger than expected.

As shown in Figure 4-10, 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 rules for multiplying signs and the result is a signed-magnitude integer. Bits 2^{63} of operands 1 and 2 are XORed to derive the sign of the result. The format of integers expected by both the hardware and software is two's complement, not signed-magnitude; therefore, negative products must be converted to two's complement form.

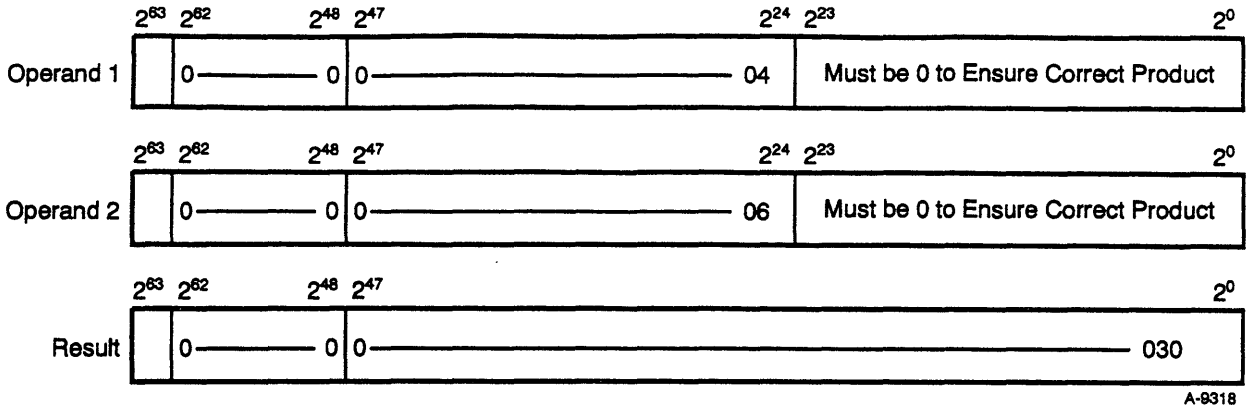


Figure 4-10. 24-bit Integer Multiply Performed in a Floating-point Multiply Functional Unit

The second multiplication method is used when the operands are greater than 24 bits in length; multiplication is done by software, which forms multiple partial products and then shifts and adds the partial products.

A second integer multiplication operation performs a 32-bit multiplication operation on the S*j* operand and the V*k* operand and puts the result in the V*i* register. The operands must be shifted left before the operation begins. The S*j* operand must be shifted left 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 4-11). 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. Bits 2^{63} through 2^{48} are zero filled. The result of the multiply is right justified into positions 2^{31} through 2^{0}, while positions 2^{63} through 2^{32} are zero filled.

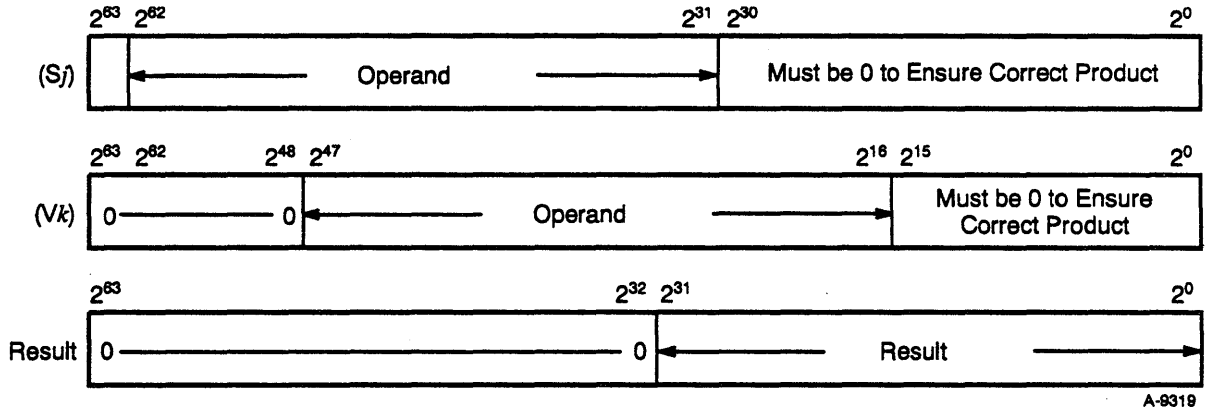


Figure 4-11. 32-bit Integer Multiply Performed in a Floating-point Multiply Functional Unit

Although no integer division 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. For more information on integer division, refer to "Floating-point Division Algorithm" later in this section.

Floating-point Arithmetic

Floating-point arithmetic is used by the scalar and vector instructions. Information under the following headings explains floating-point arithmetic:

- Floating-point data format
- Exponent ranges
- Normalized floating-point numbers
- Floating-point range errors
- Floating-point addition
- Multiplication and division algorithms
- 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 4-12. The format has three fields: coefficient sign, exponent, and coefficient.

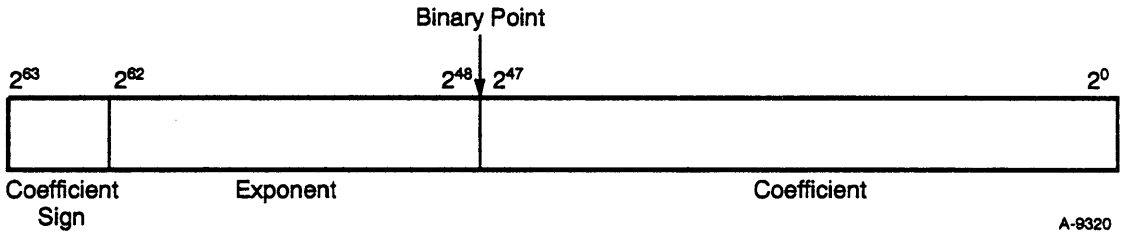


Figure 4-12. Floating-point Data Format

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, and a 1 indicates a negative exponent. Bit 2^{62} is the bias of the exponent.

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 signed-magnitude format, it is not complemented for negative values. A normalized floating-point number has a 1 in the 2^{47} bit position, and an unnormalized floating-point number has a 0 in this bit position (normalized numbers are discussed in more detail later in this section).

Figure 4-13 and the following steps show the relation between the biased exponent and the coefficient. The following steps convert a floating-point number to its decimal equivalent.

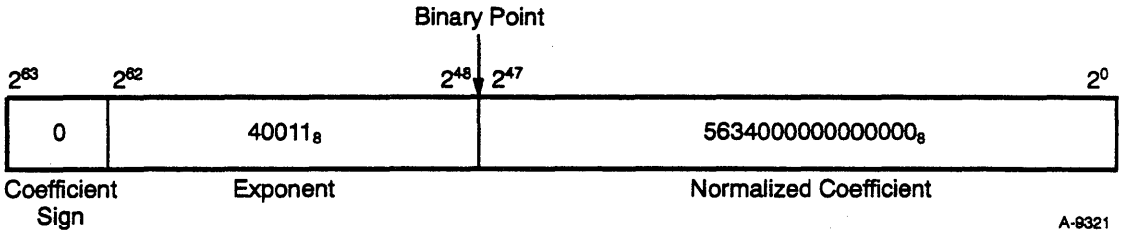


Figure 4-13. Internal Representation of a Floating-point Number

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 a negative 0 is one operand going into the floating-point multiply or floating-point add functional unit.

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 added to the exponents is 40000_8, which represents an exponent of 0. Figure 4-14 shows the biased and unbiased exponent ranges.

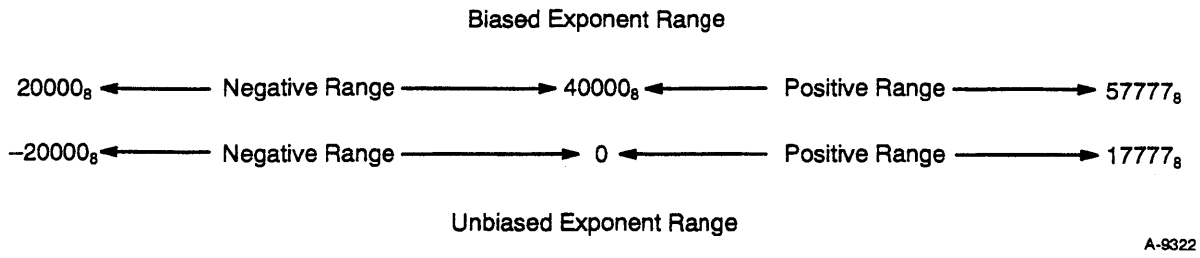


Figure 4-14. Biased and Unbiased Exponent Ranges

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

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 is shifted as far left as possible and that the exponent is 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 40060_8 into the exponent and a 48-bit integer into the coefficient, normalize the result before using it in a floating-point operation; you can do so by adding the unnormalized floating-point operand to 0. Because S0 provides a 64-bit zero when used in the S*j* field of an instruction, an operand in S*k* is normalized with the 062*i*0*k* instruction. S*i*, which can be the same register as S*k*, contains the normalized result.

The reciprocal approximation functional unit must have normalized numbers to produce correct results; using unnormalized numbers produces inaccurate results. The floating-point multiply functional unit does not require 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 ensure that the limits of the functional units are not exceeded, a range check is made on the exponent of each floating-point number coming into the functional unit for overflow and underflow conditions. In the floating-point add and floating-point multiply functional units, bits 2^{61} and 2^{62} are checked. If both are equal to 1, the exponent is equal to or greater than 60000_8, and an overflow condition is detected. If both are equal to 0, the exponent is less than or equal to 17777_8, and an underflow condition is detected.

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, an incoming operand with an exponent less than or equal to 20001_8, or greater than or equal to 60000_8, causes a floating-point range error.

When an overflow condition is detected in the floating-point add or multiply functional unit, or a floating-point range error is detected in the reciprocal approximation functional unit, an interrupt may occur. An interrupt occurs only if the interrupt-on-floating-point error (IFP) mode is set and enabled and the system is not in monitor mode. The IFP bit can be set or cleared by a user mode program.

When an underflow condition is detected in the floating-point add or multiply functional unit, no flag is set, but the exponent and coefficient are both set to 0. When an underflow condition is detected in the reciprocal approximation functional unit, the coefficient is set to 0, the exponent is set to 60000_8, and the floating-point error (FPE) flag is set if the IFP mode is set and enabled.

Floating-point Add Functional Unit Range Errors

A floating-point add range error condition occurs when the exponent of either of the incoming operands is greater than or equal to 60000_8. This condition is referred to as *overflow*. It sets the FPE interrupt flag (if the IFP interrupt mode is set and enabled), and sends an exponent of 60000_8, together with the computed coefficient, to the result register (refer to Figure 4-15).

If a floating-point addition or floating-point subtraction operation generates an unnormalized result with an exponent of less than 20000_8 or a coefficient of 0, a condition referred to as *underflow* results. No fault is generated, and the word returned from the functional unit is all 0 bits (refer to Figure 4-15).

Because the underflow condition of the result is tested before the result is normalized, the normalized result can have a valid exponent as low as 17721_8. This would occur if the unnormalized result had an exponent of 20000_8 and a coefficient of 1. In this case, no underflow is detected, and the calculated result is sent to the result register.

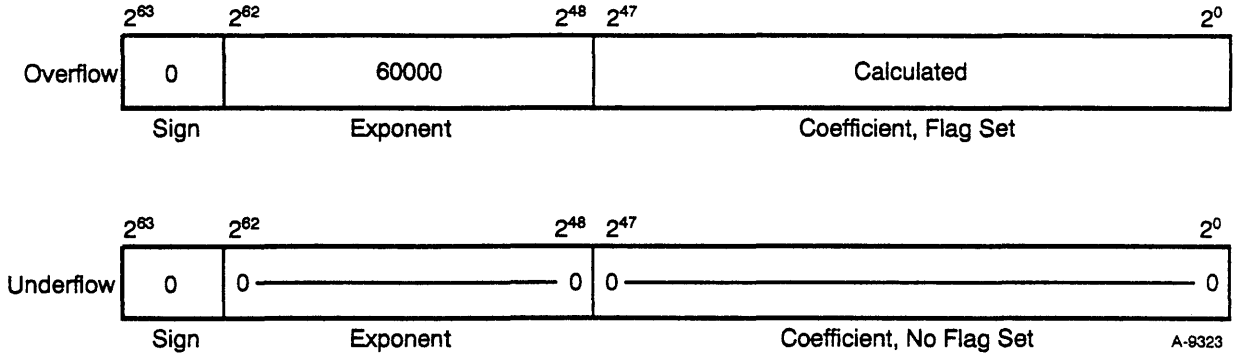


Figure 4-15. Floating-point Add and Floating-point Multiply Range Errors

Floating-point Multiply Functional Unit Range Errors

The floating-point multiply functional unit has the same range error conditions as the floating-point add functional unit (refer to Figure 4-16). The only exception is when both exponents are equal to 0; the operation is allowed to proceed as an integer multiply, leaving the exponent and sign bits equal to 0.

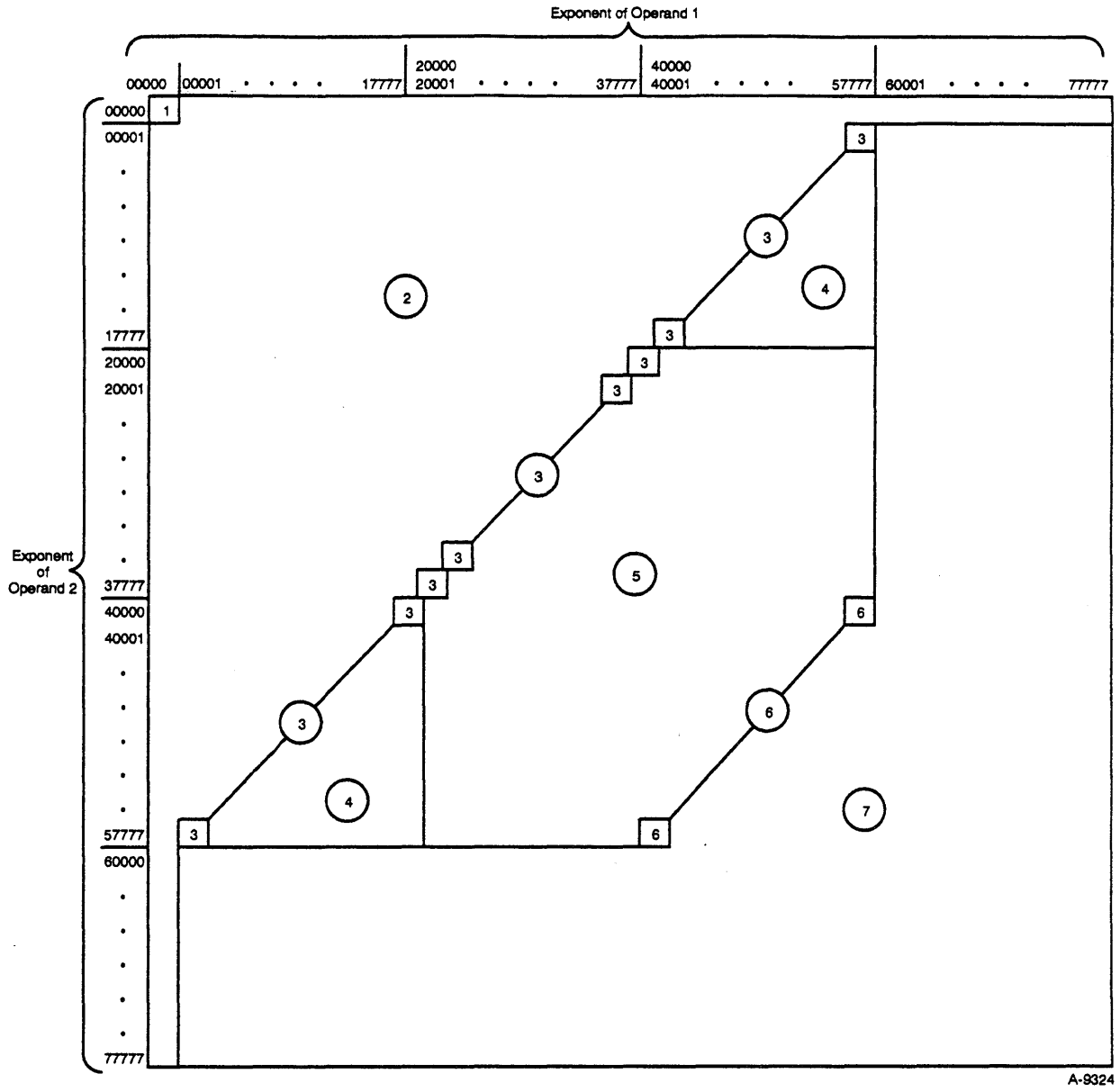


Figure 4-16. Exponent Matrix for a Floating-point Multiply Functional Unit

Out-of-range conditions are tested before the operands are normalized in the floating-point multiply functional unit. The way the out-of-range conditions are handled can be determined by using the exponent matrix shown in Figure 4-16. The exponent of the result, for any set of exponents, falls into one of the following seven zones. Only zones 6 and 7 generate floating-point errors.

Zone	Description
1	Zone 1 indicates a simple integer multiply; no fault is possible.
2	Exponents in zone 2 result in an underflow condition; the result is set to +0. (Multiplication by 0 is in this group.)
3	An underflow condition may occur on this boundary in zone 3. When a normalized shift is required, the underflow is not detected, and the coefficient and the exponent are not zeroed out. The exponent used before the shift is 20000_8; the exponent used after the shift is 17777_8. An underflow condition is only detected on the exponent used for an unshifted product coefficient.
4	The use of an operand with an underflow exponent in zone 4 is allowed if the final result ranges from 20000_8 to 57777_8.
5	Zone 5 is the normal operand range; normal results are produced.
6	An overflow condition is flagged on this boundary in zone 6. If a normalized shift is required, the value should be within bounds if the exponent is 57777_8. Because overflow is detected, 60000_8 is inserted in the product as the final exponent when the exponent for the unnormalized shift condition is used.
7	Within zone 7 an overflow fault is flagged and the product exponent is set to 60000_8.

Floating-point Reciprocal Approximation Functional Unit Range Errors

For the floating-point reciprocal approximation functional unit, an incoming operand with an exponent less than or equal to 20001_8, or greater than or equal to 60000_8, causes a floating-point range error. If the IFP interrupt mode is set and enabled, the FPE interrupt flag is set, and the following value is returned to the result register (refer to Figure 4-17):

- An exponent of 60000_8.
- The computed coefficient with bit 2^{47} set to 0.

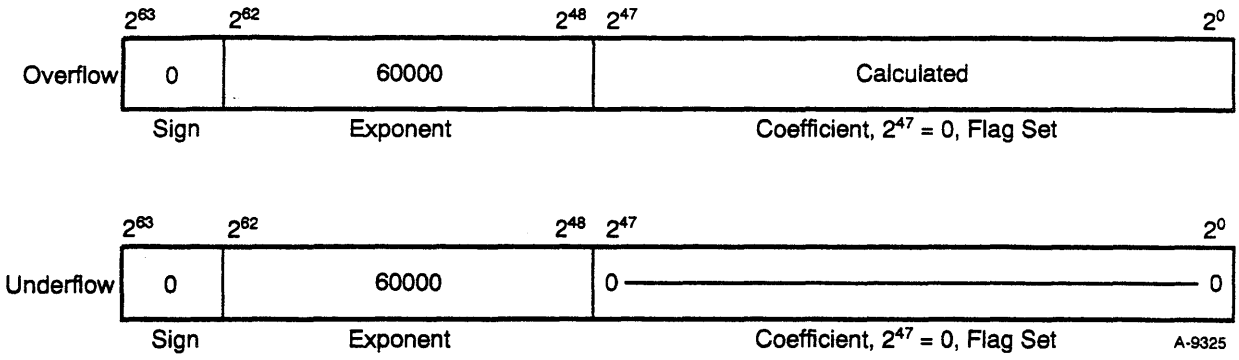


Figure 4-17. Floating-point Reciprocal Approximation Range Errors

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 results.

To equalize the exponents, the larger of the two exponents is retained. The coefficient of the smaller exponent is shifted right by the difference of the two exponents. Bits shifted out of the register are lost; no roundup occurs. Because the coefficient is only 48 bits, any shift beyond 48 bits causes the smaller coefficient to become 0's.

After the two coefficients are equalized, 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 (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 addition 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 subtraction operation is performed.
- If the sign bits are not equal and a subtract instruction is issued, an addition 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 be

decremented accordingly. If there is a carry across the binary point during an addition operation, the coefficient is shifted right by 1 and the exponent increases by 1.

The normalization feature of the floating-point add functional unit is used to normalize any floating-point number. Simply pair the number with a zero operand and send them both through the floating-point add functional unit.

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

Floating-point Multiplication Algorithm

The floating-point multiply functional unit receives two 48-bit floating-point operands from either an S or V register as input into a multiply pyramid (refer to Figure 4-18). Multiplication is commutative; that is, A x B = B x A. The signs of the two operands are exclusively ORed, 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 it is generated, while a 95-bit product requires a left shift of 1 to generate the final coefficient. If the shift is done, 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. The process assumes no shift was required to generate the final product; power of two designators are used.

The floating-point multiply 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}. This value was determined by adding all carries produced by all possible combinations that could be truncated and dividing the sum by the number of possible combinations. Nine carries are inserted at the 2^{-56} position to compensate for the truncated bits.

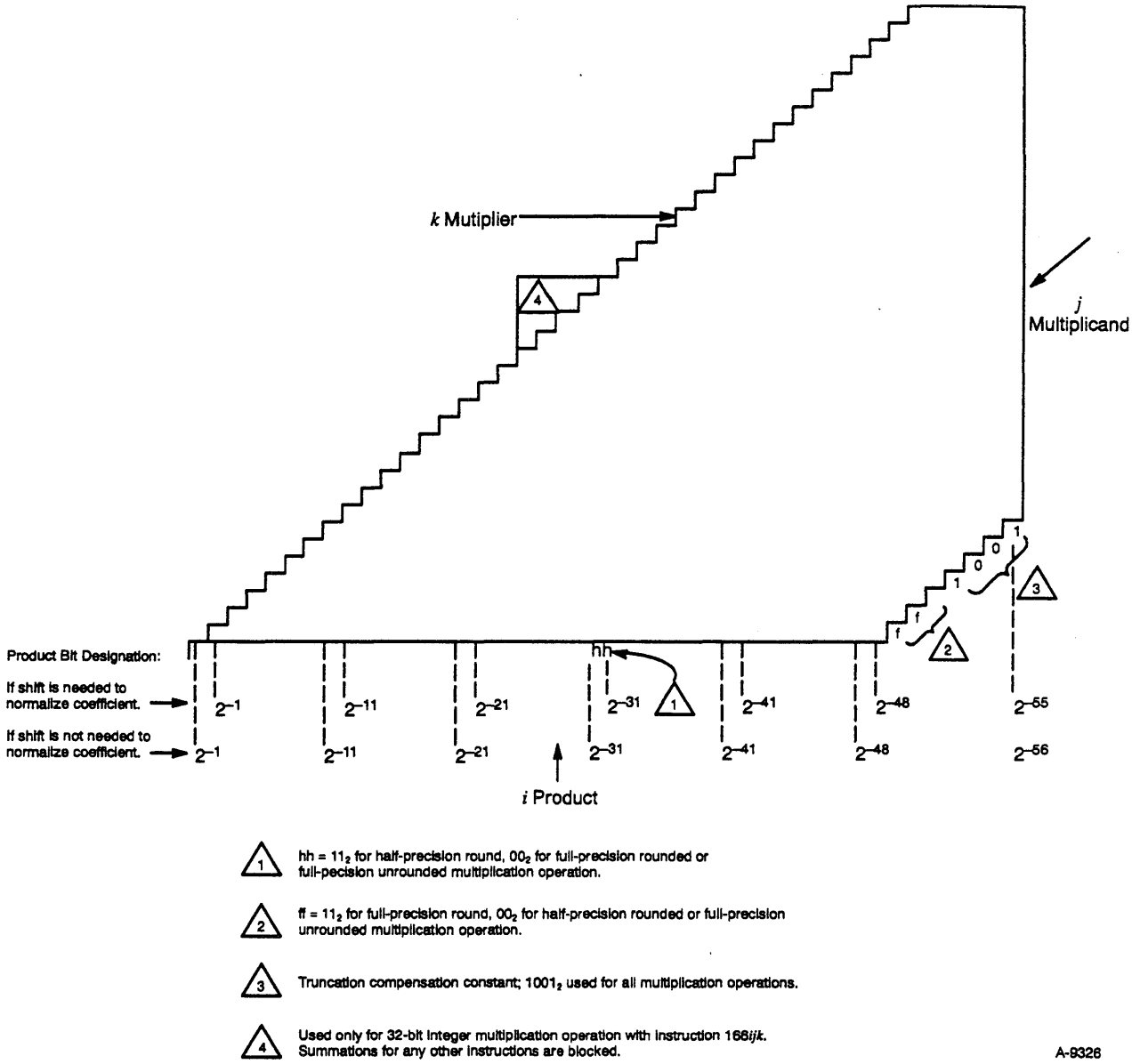


Figure 4-18. Floating-point Multiply Partial-product Sums Pyramid

The effect of the truncation without compensation is at most a result coefficient 1 smaller than expected in bit position 2^{-48}. This is because bit position 2^{-48} is set to 0 if no compensation is used. With compensation, the results range from 1 too large to 1 too small in bit position 2^{-48}, reflecting the fact that this bit is set to either 0 or 1. With compensation, approximately 99% of the values have zero deviation from what would have been generated had a full 96-bit product been present. Rounding is optional, but 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 multiplication operation, 2 round bits are entered into the summation at bit positions 2^{-50} and 2^{-51} and allowed to propagate.

For a half-precision multiplication operation, 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 upward and the 29 most significant bits of the normalized result are transmitted to the result register.

The result variations caused by 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 result variations are in the following range:

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

Floating-point Division Algorithm

The CRAY Y-MP C90 computer system does not have a single functional unit dedicated to the division operation; rather, the floating-point multiply and reciprocal approximation functional units together carry out the algorithm. The following paragraphs explain the algorithm and how it is used in the functional units.

Finding the quotient of two floating-point numbers involves two steps, as shown below in the example to find the quotient A/B.

Step	Operation
1	The B operand is sent through the reciprocal approximation functional unit to obtain its reciprocal, 1/B.
2	The result from Step 1 along with the A operand is sent to the floating-point multiply functional unit to obtain the product 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 $F(x) = 1/x - B = 0$ must be solved. To do this, 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 (shown as x_0 in Figure 4-19), sufficiently close to the true root (shown as x_t in Figure 4-19). x_0 is then used to obtain a better approximation; this is done by drawing a tangent line (line 1 in Figure 4-19) to the graph of $y = F(x)$ at the point $[x_0, F(x_0)]$. The x-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.

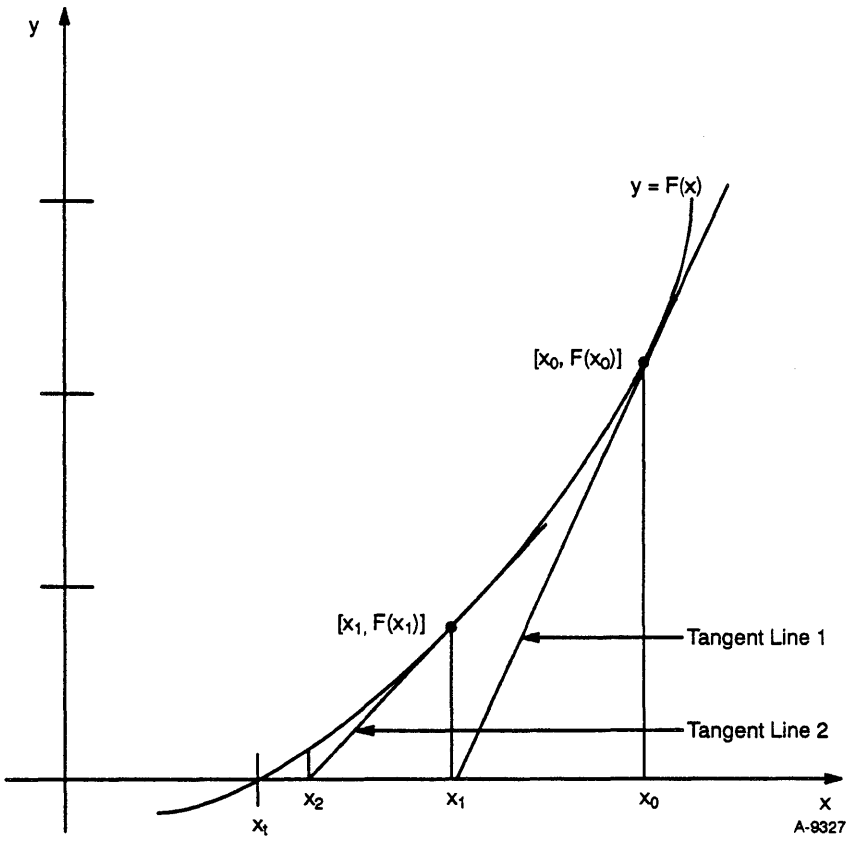


Figure 4-19. Newton's Method for Approximating Roots

The following iteration equation is derived from the above process:

$$x_{(i+1)} = 2x_i - x_i^2B = x_i\,(2 - x_iB)$$

In the equation, $x_{(i+1)}$ 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.

The CRAY Y-MP C90 mainframe uses this approximation technique based on Newton's method. A hardware look-up table provides an initial guess, x_0, which is accurate to 8 bits. 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 16 bits.
2	$x_2 = x_1(2 - x_1B)$	The second approximation is done in the reciprocal approximation functional unit and is accurate to 30 bits.
3	$x_3 = x_2(2 - x_2B)$	The third approximation is done in the floating-point multiply functional unit to calculate the correction term.

The reciprocal approximation functional unit calculates the first two iterations, while the floating-point multiply functional unit calculates the third iteration. The third 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).

The reciprocal iteration is designed for use once with each half-precision reciprocal generated. If the third iteration (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. A fourth iteration should not be done.

The following example shows how the floating-point multiply functional unit provides a full-precision result, computing the value of S1/S2.

Step	Operation	Unit
1	S3 = 1/S2	Reciprocal approximation functional unit
2	S4 = [2 – (S3 * S2)]	Floating-point multiply functional unit

Step	Operation	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 rounding

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 multiplication 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 multiplication operation to produce a half-precision quotient in only two operations, as shown in the following example.

Step	Operation	Unit
1	S3 = 1/S2	Reciprocal approximation functional unit
2	S6 = S1 * S3	Floating-point multiply functional unit in half-precision mode

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

The following is another method of computing division:

Step	Operation	Unit
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

With this method, the correction to reach a full-precision reciprocal is done after the numerator 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 the final quotient is produced. Therefore, compare the reciprocals using the same method each time they are generated. Cray Fortran CFT and CFT77 use a consistent method to ensure that the reciprocals of numbers are always the same.

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, Inc.

5 PARALLEL PROCESSING FEATURES

The CRAY Y-MP C90 computer system has several special features that enhance the parallel processing capabilities inherent in the system. Parallel processing can mean different things in different environments; the following subsections discuss two types of parallel processing found in the CRAY Y-MP C90 computer system:

- Parallel processing within a single central processing unit (CPU) of a CRAY Y-MP C90 mainframe.
- Parallel processing between two or more CPUs of a CRAY Y-MP C90 mainframe.

Parallel processing features within a single CPU include instruction pipelining and segmentation, functional unit independence, and vector processing (vectorization). The first two features are inherent hardware features of the CRAY Y-MP C90 computer system; a programmer has little control over these features. Vector processing is the feature that can be manipulated by the programmer to provide optimum throughput. Refer to "Vector Processing" in Section 4 for more information on vector processing.

Parallel processing between two or more CPUs is called *multiprocessing*: the ability for several programs to run concurrently on multiple CPUs of a single mainframe. Included in this category are *multitasking* and *Autotasking*. Multitasking is the ability to run two or more parts (or tasks) of a single program in parallel on different CPUs within a mainframe. Autotasking is a feature of the CF77 Fortran compiling system that allows user programs to be automatically partitioned over multiple CPUs without a user interface.

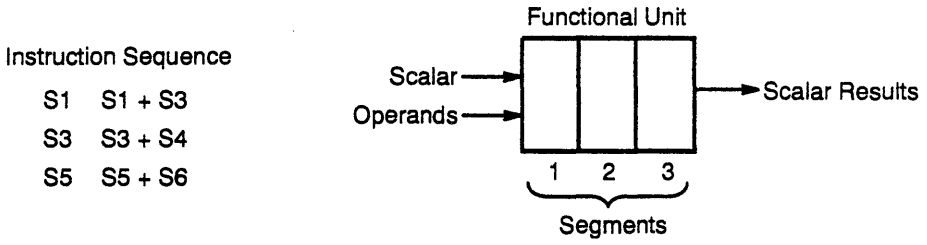
Because the intent of this manual is to present programmers with system hardware information, the following subsections focus on the parallel processing features most closely related to the hardware (the parallel processing features that execute within a single CPU of a mainframe). A basic definition and explanation of multiprocessing, multitasking, and Autotasking is included. Refer to the bibliography of this manual for a list of publications containing more detailed information on optimizing code and more of the software features of the CRAY Y-MP C90 computer system.

Pipelining and Segmentation

Pipelining is defined as an operation or instruction beginning before a previous operation or instruction completes. Pipelining is accomplished using 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 uses this segmentation by performing one segment of the operation during a single clock period (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 processes the next operation. If segmented hardware is not used, the entire operation or instruction must complete before another operation or instruction starts.

In CRAY Y-MP computer systems, all hardware is fully segmented. Therefore, pipelining occurs during all hardware operations such as 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 5-1 shows how the execution of the sequence of scalar instructions indicated results in the pipelining of three sets of operands through a segmented functional unit.



CP	Functional Unit Segment 1	2	3
1	S1 Partial Result		
2	S3 Partial Result	S1 Partial Result	
3	S5 Partial Result	S3 Partial Result	S1 Partial Result
4		S5 Partial Result	S3 Partial Result

A-9444

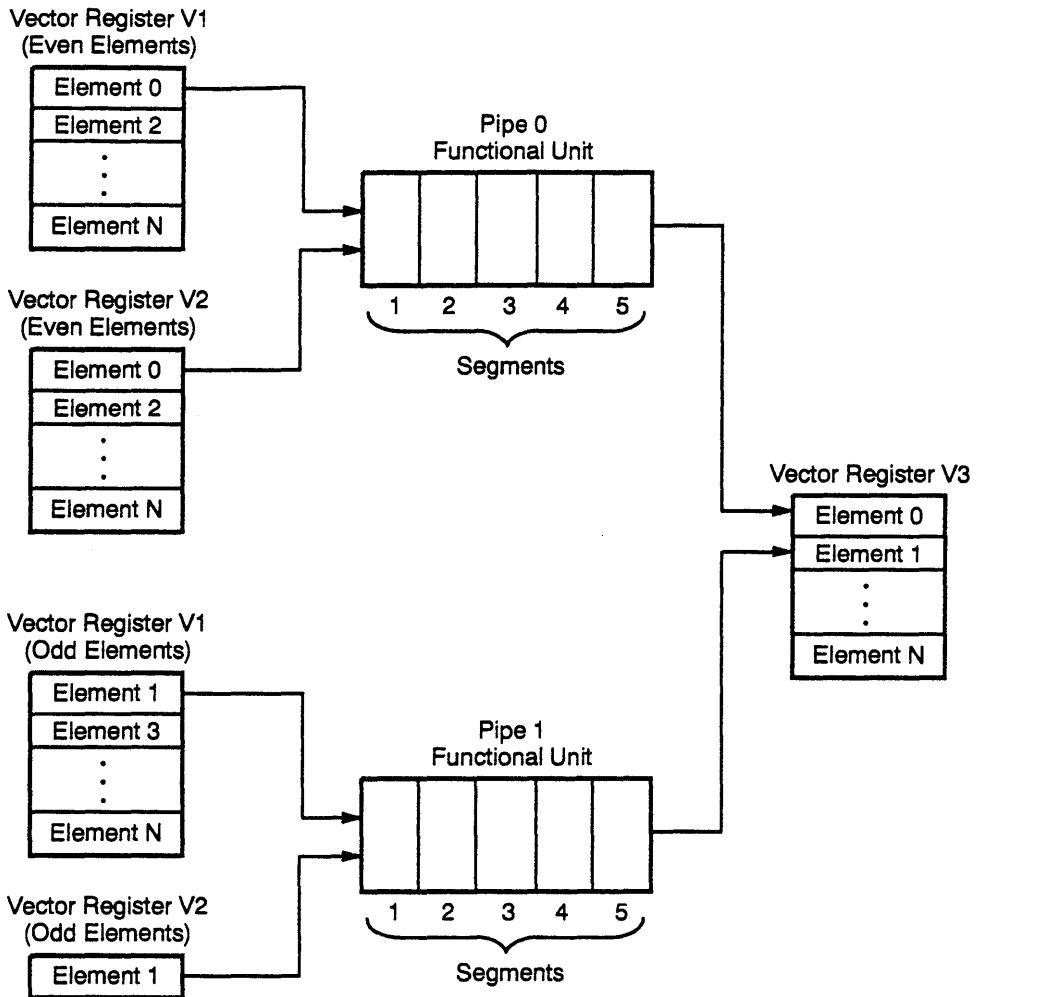
Figure 5-1. Scalar Segmentation and Pipelining Example

In the first CP, the first set of operands 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 the second pair of operands enters the first segment. This process continues each CP until the three operand pairs are completely processed. After 3 CPs, the first result leaves the functional unit and enters scalar register S1; the S3 and S5 results are available in successive CPs.

The CRAY Y-MP C90 computer system contains two sets of vector functional units, one for processing even-numbered elements and one for processing odd-numbered elements. This dual set of functional units allows two pairs of elements to be processed in a single CP, almost doubling the vector processing rate. Figure 5-2 shows how a set of vector elements is pipelined through a dual vector functional unit.

In the first CP, element 0 of register V1 and element 0 of register V2 enter the first segment of the pipe 0 functional unit, while element 1 of each register enters the pipe 1 functional unit. During the next CP, the partial results are moved to the second segments of each functional unit, while element 2 of both vector registers enters the first segment of the pipe 0 functional unit, and element 3 of both vector registers enters the first segment of the pipe 1 functional unit. This process continues each CP until the number of elements specified by the vector length (VL) register is processed.

In this example, the functional units are divided into five segments; the dual functional units can process up to ten different pairs of elements simultaneously. After 5 CPs, the first results leave the functional units and enter vector register V3; subsequent results are available at the rate of two per CP.



CP	Functional Unit Segment 1	2	3	4	5
1	0, 1				
2	2, 3	0, 1			
3	4, 5	2, 3	0, 1		
4	6, 7	4, 5	2, 3	0, 1	
5	8, 9	6, 7	4, 5	2, 3	0, 1
6	10, 11	8, 9	6, 7	4, 5	2, 3

Elements in each Segment during Successive CPs

A-9445

Figure 5-2. Vector Segmentation and Pipelining Example

Functional Unit Independence

The specialized functional units in the CRAY Y-MP C90 computer system handle the arithmetic, logical, and shift operations. Most functional units are fully independent; any number of functional units can process instructions concurrently. Functional unit independence allows different operations such as multiplications and additions to proceed in parallel.

For example, the equation A = (B + C) x D x E could be run as follows. If operands B, C, D, and E are 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; because the addition takes less time to run than the multiplication, they complete at the same time. The addition operation is essentially hidden, in that it occurs during the same time interval as the multiplication operation. The results of these two operations are then multiplied to obtain the final result.

Multiprocessing and Multitasking

Users of CRAY Y-MP C90 computer systems can take advantage of parallel processing features known as multiprocessing and multitasking, including microtasking.

Parallel processing between two or more CPUs is called multiprocessing: the ability for several programs to be run concurrently on multiple CPUs of a single mainframe. Up to *n* programs can run simultaneously on a machine with *n* CPUs.

Multitasking is a more recent and complex enhancement than vectorization. Multitasking is the ability to run two or more parts, or tasks, of a single program in parallel on different CPUs within a mainframe. To take advantage of this feature, a program must be logically or functionally divided to allow two or more tasks to run simultaneously (that is, in parallel). An example of this is a weather modeling job in which the northern hemisphere calculation is one part and the southern hemisphere another part. Distinct code segments are not needed; the same code could run on multiple processors simultaneously, with each processor acting on different data.

The theoretical gain that can be achieved from multitasking is that a program running on a dedicated system in wall clock time, *t*, could run in a time as short as *t/n*, if multitasked, or if modified to use *n* or more parallel tasks on a machine with *n* CPUs.

In practice, however, a speedup factor of *n* is not quite attainable because of the overhead needed to implement multitasking. In some instances, multitasking can actually increase a program's execution time if the multitasking overhead decreases performance more than parallel execution time improves it. Investigate this situation before you invest too much time and effort. There are some factors that limit the maximum improvement of a program:

- Not all parts of a program can be divided into parallel tasks.
- Those parts that can be multitasked may depend on one another resulting, at running time, in one or more tasks having to wait until others complete some operation.
- Use of the multitasking features incurs a certain amount of overhead.

The CFT77 compiler on the CRAY Y-MP C90 computer system automatically uses the vector hardware to perform operations on inner DO loops that have no data dependencies. Once such optimizing is complete, a single processor can work no faster, but more than one processor could operate on separate parts of the data simultaneously to achieve results faster. Microtasking permits multiple processors to work on a Fortran program at the DO-loop level. The name *microtasking* was chosen because multiprocessing is efficient even at a DO-loop level where the task size, or granularity, may be small.

Microtasking also works well when the number of processors available is unknown or may vary during the program's execution, which means that microtasked jobs do not require a dedicated system, although they perform best in a dedicated environment with no competing jobs.

As stated before, advanced programming skills and tools are needed to successfully use multiprocessing, multitasking, and microtasking concepts in order to promote more efficient programs. These features are thoroughly discussed and explained in CRI software publications; refer to the bibliography.

Autotasking

CRAY Y-MP C90 computer system analysts and programmers can use the Autotasking feature of the CF77 Fortran compiling system to automatically detect whether portions of their programs can be run in parallel. Autotasking is an extension of multiprocessing and microtasking and is designed to make parallel processing easier to use. Autotasking alters a Fortran program to allow it to run simultaneously on multiple CPUs.

Autotasking is available on CRAY Y-MP computer systems beginning with UNICOS release 4.0 and CF77 release 3.0. Refer to the *CF77 Compiling System, Volume 4: Parallel Processing Guide*, CRI publication number SG-3074, for more detailed information on Autotasking.

6 MAINTENANCE CHANNEL

The maintenance channel of the CRAY Y-MP C90 computer system provides extensive control over the system and over individual CPUs for maintenance activities and problem diagnosis. The maintenance channel supports a concurrent maintenance philosophy if the operating system is configured appropriately.

The maintenance channel performs basic functions that offer two levels of control: system level and individual CPU level. The functions available at each level are listed below:

- System level functions:
  - I/O master clear
  - CPU master clear
  - Set memory priority counter
  - Master CPU selection
  - External control cable selection
  - System-level status reporting

- Individual CPU level functions:
  - I/O master clear
  - CPU master clear/exchange
  - Memory modes (one-half memory and 256K memory)
  - Control of CPU soft switches
  - Read from and write to memory
  - CPU-level status reporting
  - Loopback
  - Interface to diagnostic monitor (DM)

Theory of Operation

The maintenance channel is a standard low-speed (LOSP) channel connecting the mainframe to a maintenance workstation (MWS). One pair of LOSP channel cables supports the entire system. Refer to Figure 1-3.

The LOSP channel connecting the MWS to the system communicates through a 16-bit asynchronous maintenance channel interface located on the clock module in the mainframe. Each CPU in the system is connected individually to this interface. This arrangement allows for data to be broadcast simultaneously to all CPUs and avoids problems inherent in daisy chain organization, such as missing CPUs.

To operate the maintenance channel, transmit a specific function code from the MWS to the system through the maintenance channel interface. This function code specifies one of three types of commands: an individual CPU command, a broadcast command to all CPUs, or a system command. These three types of commands are described briefly in the following subsections. A special write/read operation, called *loopback*, is also described.

Individual CPU Commands

Commands directed to a specific CPU must include the physical number of the CPU in the CPU ID field. After the function code is decoded and checked for errors by the maintenance channel interface, the code is transmitted to all CPUs in the system. However, only the specified CPU acts on the command.

The selected CPU must be able to respond to the command. If the maintenance channel does not receive a response, a Resume signal is not transmitted after the fourth parcel of the function word, which effectively deactivates the input channel. This, in turn, forces a write hang on the MWS LOSP channel. If you attempt to use an unavailable CPU or one that cannot respond, allow the hang condition to time out on the LOSP channel connection; clear the channel; select another prospective CPU; and try again.

Broadcast Commands

A function used as a broadcast command is sent to all CPUs in the system. Broadcast commands are designated by setting bit 2^7 in parcel 0 of the function word. In addition, the CPU ID field must contain the number of a valid CPU that can respond to the function. If the maintenance channel does not receive a response, a Resume signal is not transmitted after the fourth parcel of the function word, which effectively deactivates the input channel. This, in turn, forces a write hang on the MWS LOSP channel. If you attempt to use an unavailable CPU or one that cannot respond, allow the hang condition to time out on the LOSP channel connection; clear the channel; select another prospective CPU; and try again.

System Commands

System commands are decoded and executed by the maintenance channel interface. These commands are designated by setting bit 2^6 in parcel 0 of the function word. The other fields in parcel 0 are not used except for the function code, and for one command, the CPU ID field.

Loopback

Loopback is a special case write and read operation in which a single word is transferred from the MWS to the system and back again. The loop-back operation is selected by setting bit 2^{15} in parcel 0 of the function word. The 7-bit function code is not decoded, nor are the broadcast or system bits. The CPU ID field, however, must contain the number of a valid CPU through which the loop-back data can pass.

The data word returned from the system in the loop-back operation contains parcel 0 of the transmitted word in the parcel 0 position and parcel 1 of the transmitted word in parcel locations 1, 2, and 3. The loop-back operation completes by transmitting a disconnect signal to the LOSP channel connection.

Write Hang

If a write hang occurs on the MWS LOSP channel, transmit a disconnect command from the MWS to the system before attempting to transmit another function code. If the disconnect attempt fails, clear the LOSP channel. Clearing the channel will also clear the error flags.

Normally, the disconnect is followed by a system status read command to determine if the cause of the hang is a function error. A function error results if there is a parcel 0 parity error, or if the upper 8 bits of parcel 1 are not the complement of the upper 8 bits of parcel 0.

Maintenance Channel Functions

Table 6-1 contains a summary of all available functions, and Table 6-2 explains the functions in detail. In Table 6-1, the only functions available in restricted mode of the maintenance channel are identified by an R designator in the last column. Some functions also require that the CPU be in 256K memory mode.

Table 6-2 includes the octal coding for the entire first parcel of the function word in parentheses beneath the binary function code.

Table 6-1. Maintenance Channel Functions

Function	Description	Function Type		
		Broadcast	System	Individual
0 000 000	Initialize CPU soft switch to hard switch settings.	X		X
0 000 000	Initialize control cable enables to hard switch.		X	
0 000 100	Clear control cable 0 enabled soft switch.		X	
0 000 101	Set control cable 0 enabled soft switch.		X	
0 000 110	Clear control cable 1 enabled soft switch.		X	
0 000 111	Set control cable 1 enabled soft switch.		X	
0 001 000	Clear CPU master clear.		X	
0 001 000	Clear CPU master clear/clear idle CPU.			X
0 001 001	Set CPU master clear.		X	
0 001 001	Set CPU master clear/set idle CPU.			X-R
0 001 010	Clear CPU master clear/clear idle CPU/exchange.			X-R
0 010 000	Clear I/O master clear.		X	X
0 010 001	Set I/O master clear.		X	X
0 010 010	Clear I/O master clear, memory priority counter = 0 and hold.		X	
0 010 011	Set I/O master clear, memory priority counter = 0 and hold.		X	
0 010 100	Release memory priority hold (advance each bank busy time).		X	
0 010 101	Advance memory priority (hold must be set).		X	
0 010 110	Set highest priority CPU = ID field and hold.		X	
0 011 000	Clear half-memory mode (soft switch).	X		X
0 011 010	Set half-memory size lower, CPU and I/O (soft switch).	X		X
0 011 011	Set half-memory size upper, CPU and I/O (soft switch).	X		X
0 100 000	Clear 256K memory mode (online tests).			X-R
0 100 010	Set 256K memory mode CPU and maintenance (online tests).			X-R
0 100 011	Set 256K memory mode CPU, I/O, and maintenance (online tests).			X-R
0 101 000	Allow full shared register and I/O access.			X
0 101 001	Cluster number = max, only I/O on this CPU.			X
0 101 010	Cluster number = max, no I/O.			X
0 101 011	No shared register or I/O (forced when restricted 256K).			X-R
0 110 000	Write memory using CA and CL.			X-(R if 256K)
0 110 001	Read memory using CA and CL.			X-(R if 256K)
0 110 011	Kill read memory (forces disconnect on maintenance channel).			X-(R if 256K)
0 111 000	Write block data to diagnostic monitor using CA and CL.	X		X
0 111 001	Read 256 parcels from diagnostic monitor.			X
0 111 011	Kill read of diagnostic monitor (forces maintenance disconnect).			X
1 000 000	Select master CPU = ID field (soft switch).	X		
1 010 000	Disable I/O ECC (soft switch).	X		X
1 010 001	Enable I/O ECC (soft switch).	X		X
1 011 000	CPU test mode off (soft switch control CPU diagnostic modes).	X		X
1 011 001	CPU test mode on (soft switch control CPU diagnoistic modes).	X		X
1 110 001	Read status/soft switches.		X-R	X-R
1 110 011	Kill read status or loopback.			X-R
1 111 000	Reset CPU diagnostic monitor (sets default values).	X		X
1 111 001	Reset CPU diagnostic monitor time stamp.	X		X
1 111 010	Stop CPU diagnostic monitor recording.	X		X
1 111 011	Stop CPU diagnostic monitor recording/hold issue next CIP.	X		X
1 111 100	Reset CPU diagnostic monitor trigger/activate/release issue.	X		X

Table 6-2. Maintenance Channel Functions in Detail

Function	Description	Requirements	Type
0 000 000 (0000id)	Initialize soft switch settings to defaults set by the current hard switch settings. Sets shared register access to full.	Maintenance channel on, restricted off, system bit clear	Broadcast, Individual CPUs
0 000 000 (000100)	Initialize external control cable enables to current switch settings (soft switch) [1].	Maintenance channel on, restricted off, system bit set	System
0 000 100 (002100)	Clear external control cable 0 enable (soft switch).	Maintenance channel on, restricted off, system bit set	System
0 000 101 (002500)	Set enable external control cable 0 (soft switch).	Maintenance channel on, restricted off, system bit set	System
0 000 110 (003100)	Clear external control cable 1 enable (soft switch).	Maintenance channel on, restricted off, system bit set	System
0 000 111 (003500)	Set enable external control cable 1 (soft switch).	Maintenance channel on, restricted off, system bit set	System
0 001 000 (004100)	Clear system CPU master clear (all CPUs).	Maintenance channel on, restricted off, system bit set	System
0 001 000 (0040id)	Clear CPU master clear/clear idle CPU (soft switch).	Maintenance channel on, restricted off, system bit clear	Individual CPUs
0 001 001 (004500)	Set system CPU master clear (all CPUs).	Maintenance channel on, restricted off, system bit set	System
0 001 001 (0044id)	Set CPU master clear/set idle CPU (soft switch).	Maintenance channel on, system bit set	Individual CPUs
0 001 010 (0050id)	Clear CPU master clear/clear idle CPU (soft switch)/exchange, start CPU.	Maintenance channel on, system bit set	Individual CPUs
0 010 000 (010100)	Clear system I/O master clear (all CPUs).	Maintenance channel on, restricted off, system bit set	System [2]
0 010 000 (0100id)	Clear I/O master clear.	Maintenance channel on, restricted off, system bit set	Individual CPUs
0 010 001 (010500)	Set system I/O master clear.	Maintenance channel on, restricted off, system bit set	System [3]
0 010 001 (0104id)	Set I/O master clear.	Maintenance channel on, restricted off, system bit set	Individual CPUs
0 010 010 (011100)	Clear system I/O master clear/set memory priority = 0 and hold.	Maintenance channel on, restricted off, system bit set	System [4]
0 010 011 (011500)	Set system I/O master clear/set memory priority = 0 and hold.	Maintenance channel on, restricted off, system bit set	System [5]
0 010 100 (012100)	Release memory priority hold (allow advance of priority each bank busy time).	Maintenance channel on, restricted off, system bit set	System [6]
0 010 101 (012500)	Advance memory priority (hold must be set; counter advances once per function).	Maintenance channel on, restricted off, system bit set	System

Table 6-2. Maintenance Channel Functions in Detail (continued)

Function	Description	Requirements	Type
0 010 110 (0131id)	Set highest priority CPU = ID field and hold (counter advances until = CPU, then holds).	Maintenance channel on, restricted off, system bit set	System
0 011 000 (0140id)	Clear half-memory mode (soft switch).	Maintenance channel on, system bit set	Broadcast, Individual CPUs
0 011 010 (0150id)	Set half-memory mode lower (soft switch) CPU, I/O, maintenance addresses are affected.	Maintenance channel on, system bit set	Broadcast, Individual CPUs
0 011 011 (0154id)	Set half-memory mode upper (soft switch) CPU, I/O, maintenance addresses are affected.	Maintenance channel on, system bit set	Broadcast, Individual CPUs
0 100 000 (0200id)	Clear 256K memory mode .	Maintenance channel on	Individual CPUs [7]
0 100 010 (0210id)	Set 256K memory mode CPU, maintenance addresses are affected.	Maintenance channel on	Individual CPUs [8]
0 100 011 (0214id)	Set 256K memory mode CPU, I/O, and maintenance addresses are affected.	Maintenance channel on	Individual CPUs [9]
0 101 000 (0240id)	Allow full shared register and I/O access.	Maintenance channel on, restricted off	Individual CPUs
0 101 001 (0244id)	Cluster = max, I/O allowed from this CPU.	Maintenance channel on, restricted off	Individual CPUs
0 101 010 (0250id)	Cluster = max, I/O not allowed from this CPU.	Maintenance channel on, restricted off	Individual CPUs
0 101 011 (0254id)	No shared register or I/O allowed from this CPU.	Maintenance channel on	Individual CPUs [10]
0 110 000 (0300id)	Write memory using CA and CL (12 or more parcels).	Maintenance channel on	Individual CPUs [11]
0 110 001 (0304id)	Read memory using CA and CL (8 parcels).	Maintenance channel on	Individual CPUs [12]
0 110 011 (0314id)	Kill read memory.	Maintenance channel on	Individual CPUs [13]
0 111 000 (0340id)	Write to diagnostic monitor using CA and CL/stop DM recording (12+ parcels).	Maintenance channel on, restricted off	Broadcast, Individual CPUs
0 111 001 (0344id)	Read 256 parcels from diagnostic monitor/stops DM recording.	Maintenance channel on, restricted off	Individual CPUs
0 111 011 (0354id)	Kill read of diagnostic monitor/stops DM.	Maintenance channel on, restricted off	Individual CPUs [14]
1 000 000 (0402id)	Select Master CPU = ID field (soft switches).	Maintenance channel on, restricted off, broadcast set	Broadcast
1 010 000 (0500id)	Disable I/O ECC (soft switch).	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 010 001 (0504id)	Enable I/O ECC (soft switch).	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 011 000 (0540id)	CPU test mode off (soft switch control CPU diagnostic modes).	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 011 001 (0544id)	CPU test mode on (soft switch control CPU diagnostic modes).	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 110 001 (070500)	Read status/system soft switch settings.	System bit set	System, no CPUs [15]
1 110 001 (0704id)	Read status/soft switch settings.	Maintenance channel on, system bit clear	Individual CPUs [16]

Table 6-2. Maintenance Channel Functions in Detail (continued)

Function	Description	Requirements	Type
1 110 011 (0714id)	Kill status/loopback mode.	Maintenance channel on	Individual CPUs [17]
1 111 000 (0740id)	Reset CPU diagnostic monitor (sets default values does not activate DM).	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 111 001 (0744id)	Reset CPU diagnostic monitor time stamp.	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 111 010 (0750id)	Stop CPU diagnostic monitor recording.	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 111 011 (0754id)	Stop CPU diagnostic monitor recording/hold issue next valid CIP.	Maintenance channel on, restricted off	Broadcast, Individual CPUs
1 111 100 (0760id)	Reset CPU diagnostic monitor trigger/write "ones"/release issue/activate DM.	Maintenance channel on, restricted off	Broadcast, Individual CPUs

NOTES

1. The control cables contain CPU, MC, I/O, deaddump, real-time interrupts, and an MCU interrupt from the external OWS or IOS. These controls can come from two separate cables.
2. System use also releases memory master clear and releases any hold of memory priority.
3. Individual CPU use does not affect the memory master clear.
4. System use also sets memory master clear, resets the memory priority counter, and releases any hold of memory priority. Any 256K memory mode is cleared. System I/O master clear will also prevent any maintenance channel functions sent to a CPU from completing.
5. Individual CPU use does not affect the memory master clear or priority.
6. System I/O master clear also prevents any maintenance channel functions sent to a CPU from completing. Any 256K memory mode is cleared.
7. If restricted mode is on, share access is set to full.
8. When this mode is set with a restricted bit on a 0 100 010, the function code is forced, limiting this CPU so it cannot issue shared register or I/O instructions.
9. When this mode is set with a restricted bit on a 0 101 011, the function code is forced, limiting this CPU so it cannot issue shared register or I/O instructions.
10. Programmable clock interrupt and read of real-time clock is still available to this CPU.
11. If restricted mode is on, 256K mode must be enabled, or the channel hangs and times out.
12. If restricted mode is on, 256K mode must be enabled, or the data and the Disconnect signal is not sent. The channel times out.
13. This function sends a Disconnect signal after any current memory reference is complete.
14. This function sends an Unconditional Disconnect signal.
15. A read of system status contains system status error accumulators for all CPUs. This function clears these errors. A valid CPU is not required, because this function does not access any CPU. This function is used for obtaining the valid CPUs of this system, as well as the current system-level soft switches (control cable enables).
16. A read of a CPU status also contains system status and error accumulators for all CPUs. This function clears these errors. The soft switch status from the CPU indicates that CPU's switch setting. The control cable enables are exceptions. They are system-level soft switches only. The current valid CPUs are also shown.
17. This function should be used only when the status function or loop-back mode is in a hung condition. Normally status and loopback finish by transmitting their 4 parcels and sending a Disconnect signal. This function sends an Unconditional Disconnect signal.

Data Formats

Data transmitted from the MWS to the system must be arranged in a specific format if it is to be interpreted properly. Also, status read data transmitted from the system to the MWS follows a definite format. These two formats are described in the following subsections.

MWS Write Data

The format of the data transmitted from the MWS to the system depends on the function selected. Three different transfer lengths are used. Most functions require only 4 parcels (one word); however, memory read functions require 8 parcels (two words), and memory write or diagnostic monitor parameter write functions use more than 12 parcels. Figure 6-1 shows the content and data format of the parcels used in these operations.

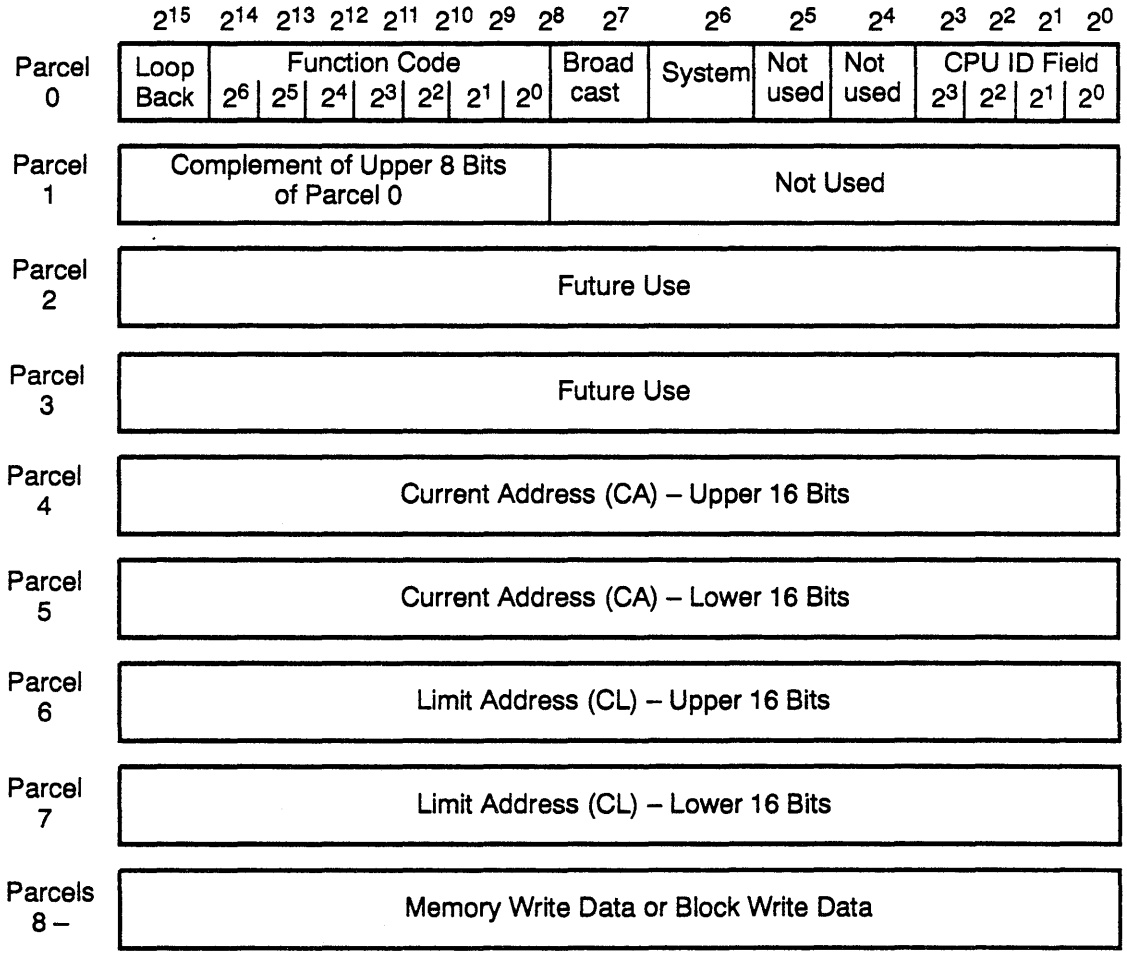


Figure 6-1. MWS Write Data Format

Parcel 0 of the function word (parcels 0 through 3) includes the function code and type of command. These codes and commands were explained in a previous subsection. All references to CPU numbers in the ID field are based on the physical CPU, not the logical CPU. The upper 8 bits of parcel 1 of the function word must contain the complement of the upper 8 bits of parcel 0. The remaining bits in the function word are not used and may be set to any value.

A second word (parcels 4 through 7), which contains beginning and ending channel addresses of the data to be transferred, is required only if data is to be read or written over the channel. Additional words containing write data are required only for memory or diagnostic monitor write functions.

For memory write operations, successful completion of the data transfer is indicated by the receipt of the Resume signal at the LOSP channel connection transmitted after the fourth parcel of the last word of the transfer.

For all read operations, including memory, diagnostic monitor, status, and loop-back operations, the CPU transmits serial data to the maintenance channel interface. Here the data is converted to the 16-bit data and 4-bit parity format required by the LOSP channel for transmission to the MWS. The end of the transfer is indicated by a Disconnect signal sent to the LOSP channel connection.

Status Read Data

Function 1 110 001 transmits selected status information from the mainframe to the MWS. This function can be used as either a system command or as an individual CPU command. Either command returns a single word of status data from the system. However, the contents and format of parcel 0 are unique in each case. Figure 6-2 shows the contents of parcel 0 of the status word for a system read, and Table 6-3 shows the contents of the same parcel for an individual CPU read.

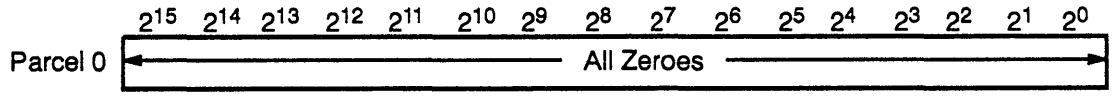


Figure 6-2. System Status Read Format (Parcel 0)

Table 6-3. Individual CPU Status Read Format (Parcel 0)

Bit Position	Description
2^{15}	Always 1
2^{14}	Maintenance restrict
2^{13}	Diagnostic monitor active
2^{12}	Diagnostic monitor parity error
2^{11}	256K memory mode (not a soft switch)
2^{10}	Shared register select 1
2^{9}	Shared register select 0
2^{8}	CPU test mode
2^{7}	Half-memory mode
2^{6}	Half-memory upper select
2^{5}	I/O ECC enabled
2^{4}	Idle
2^{3}	Master CPU select 3
2^{2}	Master CPU select 2
2^{1}	Master CPU select 1
2^{0}	Master CPU select 0

Parcels 1 through 3 of the status word contain the same data and format for both the system and individual CPU command. This format is shown in Figure 6-3. Bits 2^7 through 2^0 of parcel 1 are explained below:

- Bits 2^7 and 2^6, CTRL1 EN and CTRL0 EN, are soft switch values.
- Bit 2^5, FNCT ERR, sets if there is a mismatch in parcel 0 or parcel 1 of the maintenance channel (command) header.
- Bit 2^4, SYS ERR, sets if there is a parcel parity error.
- Bits 2^3 through 2^0 specify the current memory priority.

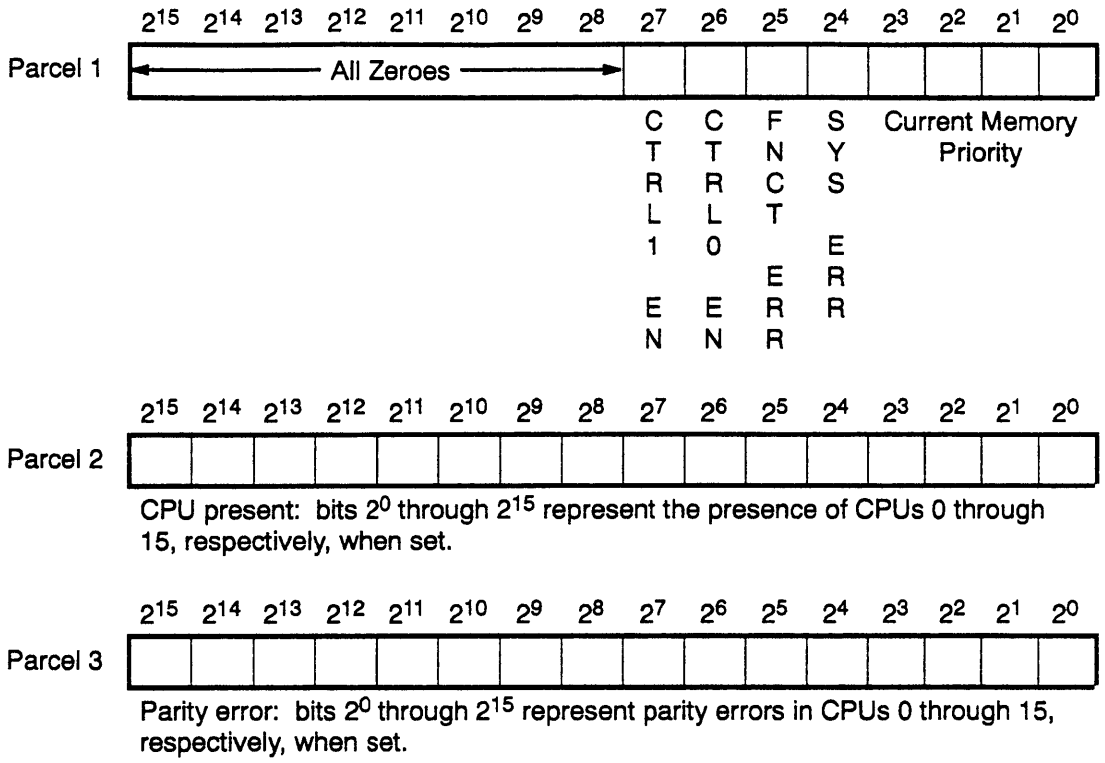


Figure 6-3. System and Individual CPU Status Read Formats (Parcels 1 through 3)

Diagnostic Monitor

The diagnostic monitor is used to record testpoint and control information at specified times to indicate the state of the CPU. The monitor is most useful primarily as a diagnostic tool in system troubleshooting. For a complete description of the diagnostic monitor and how it operates, refer to the *CRAY Y-MP C90 Computer System Hardware Maintenance Manual*, publication number CMM-0502-000.

7 CPU INSTRUCTIONS

The following subsections explain the instruction formats, special register values, special CAL syntax forms, and monitor mode instructions, as well as the instruction differences between Y-MP mode and C90 mode. The remainder of the section contains a detailed description of all CRAY Y-MP C90 CPU instructions.

Notational Conventions

The following conventions are used throughout this section:

- All numbers are decimal numbers unless otherwise indicated.
- The letter x represents an unused value.
- Register bits are numbered from right to left as powers of 2.
- The letter *n* represents a specified value.
- The notation (value) specifies the contents of a register or memory location as designated by value.
- Variable parameters are in *italic* type.
- The vector mask (VM) bits are contained in the VM and VM1 registers. The bits of the VM register correspond to vector elements 0 through 63, and the bits of the VM1 register correspond to vector elements 64 through 127, as shown in Figure 7-1.

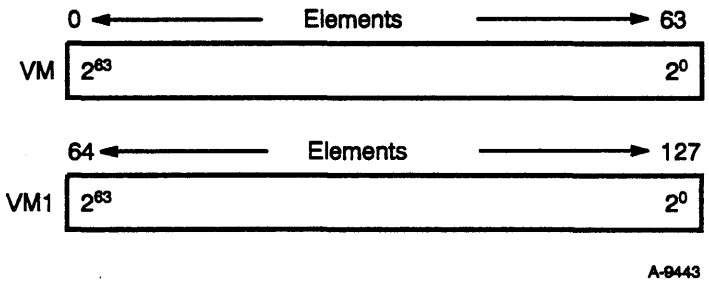


Figure 7-1. Vector Mask Bits

Instruction Formats

Instructions can be 1 parcel (16 bits), 2 parcels (32 bits), or 3 parcels (48 bits) 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- or 3-parcel instruction can begin 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 of word boundaries is required. Figure 7-2 shows the general instruction format.

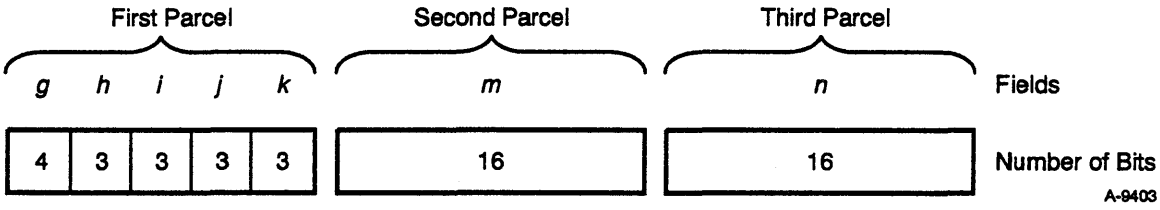


Figure 7-2. General Instruction Format

The first parcel is divided into five fields, and the second and third parcels each contain a single field. The four variations of this general format are listed below:

- 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 *i*, *j*, *k*, and *m* fields (Y-MP mode only).
- 3-parcel instruction format with combined *m* and *n* fields.

Each format uses the fields differently and is described in detail in the following subsections.

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 7-3). The *g* and *h* fields define the operation code, the *i* field designates a result register, and the *j* and *k* fields designate operand registers. Some instructions ignore one or more of the *i*, *j*, and *k* fields.

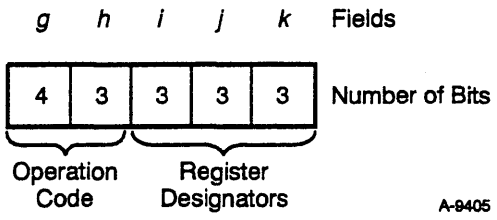


Figure 7-3. 1-parcel Instruction Format with Discrete *j* and *k* Fields

The following types of instructions use this format:

- Arithmetic
- Logical
- Vector shift
- Scalar double-shift
- Floating-point constant

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 7-4). The *g* and *h* fields contain the operation code, and the *i* field usually designates a result register. The combined *j* and *k* fields contain a constant, an intermediate address (B) register designator, or an intermediate scalar (T) register designator. The 005 branch instructions and the following types of instructions use the 1-parcel instruction format with combined *j* and *k* fields:

- 6-bit constant
- B or T register block memory transfer
- B or T register data transfer with address (A) or scalar (S) register
- Scalar single-shift
- Scalar mask

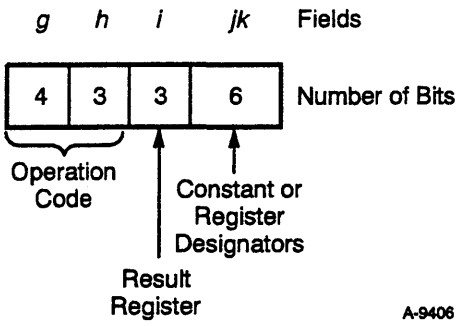


Figure 7-4. 1-parcel Instruction Format with Combined *j* and *k* Fields

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

The 2-parcel instruction 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 7-5). A 7-bit operation code (*gh*) is followed by an *ijkm* field. The high-order bit of the *i* field (*i2*) is equal to 0.

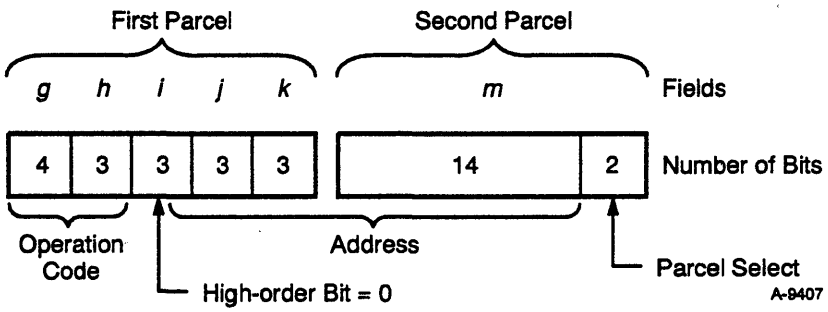


Figure 7-5. 2-parcel Instruction Format with Combined *i, j, k,* and *m* Fields

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

There are three distinct 3-parcel instruction formats using the 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 instructions using this format transfer 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 for a C90-mode branch instruction uses the combined *m* and *n* fields to hold the memory branch address. C90 mode is explained in the next subsection. The 7-bit *g* and *h* fields (and, in one case, bit 2^2 of the *i* field) contain an operation code.

The format for A or S register memory references uses the combined *m* and *n* fields to hold the memory reference address. 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. Refer to "Absolute Memory Address Calculating" in Section 2 for additional information.

Figure 7-6 shows the three applications for the 3-parcel instruction format with combined *m* and *n* fields. Remember that the *m* and *n* fields are reversed when a 3-parcel instruction is assembled.

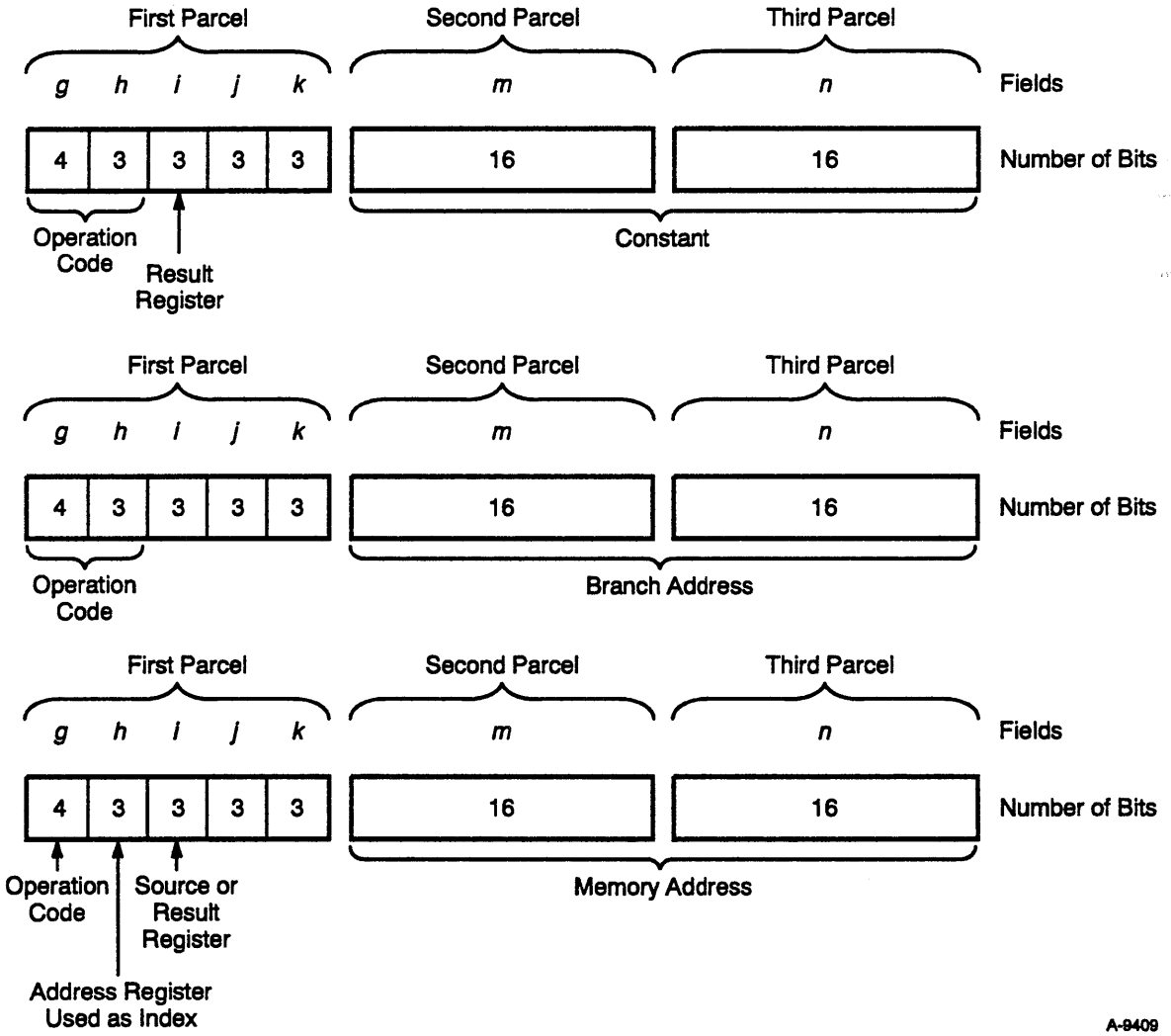


Figure 7-6. 3-parcel Instruction Format with Combined *m* and *n* Fields

Y-MP Mode and C90 Mode Instruction Differences

The CRAY Y-MP C90 computer system offers two modes of operation: a Y-MP compatibility mode and C90 mode. These two modes are referred to as as Y-MP mode and C90 mode, respectively. In Y-MP mode, all instructions defined within the Y-MP mode of the CRAY Y-MP computer system function as specified for that mode. In addition, many new C90 mode instructions are supported in the Y-MP mode. Table 7-1 lists the new instructions for the CRAY Y-MP C90 computer system or that function differently in the CRAY Y-MP C90 and CRAY Y-MP computer systems.

Instructions supported only in C90 mode are indicated by the letter A in the mode column in "CPU Instruction Descriptions" later in this section, and instructions supported only in Y-MP mode are indicated by the letter B in the same column.

The program range is 4 Mwords in Y-MP mode and 1 Gword in C90 mode. An instruction outside these ranges produces an undefined result.

Table 7-1. CRAY Y-MP C90 and CRAY Y-MP Instruction Comparison

Machine Instruction	CAL Syntax	CRAY Y-MP C90 Function	CRAY Y-MP Function
001000	PASS	This is a no-operation instruction.	This is a no-operation instruction.
0012*j*2	DI,A*j*	Disable channel (A*j*) interrupts.	Disable channel (A*j*) interrupts.
0012*j*3	EI,A*j*	Enable channel (A*j*) interrupts.	Enable channel (A*j*) interrupts.
001302	EMI	Enable monitor mode interrupt modes.	Enable monitor mode interrupt modes.
001303	DMI	Disable monitor mode interrupt modes.	Disable monitor mode interrupt modes.
001406	ECI	Enable the programmable clock interrupt request.	Enable the programmable clock interrupt request.
001407	DCI	Disable the programmable clock interrupt request.	Disable the programmable clock interrupt request.
001600	ESI	Enable system I/O interrupts.	Enable system I/O interrupts.
0017*jk*	BP,*k* A*j*	Transmit (A*j*) to breakpoint address *k*.	N/A
00200*k*	VL A*k*	Transmit (A*k*) to VL register. (Maximum VL = 128.)	Transmit (A*k*) to VL register. (Maximum VL = 64.)

Table 7-1. CRAY Y-MP C90 and CRAY Y-MP Instruction Comparison (continued)

Machine Instruction	CAL Syntax	CRAY Y-MP C90 Function	CRAY Y-MP Function
002301	EBP	Enable interrupt on breakpoint.	N/A
002401	DBP	Disable interrupt on breakpoint.	N/A
002704	CPA	Complete port reads and writes.	N/A
002705	CPR	Complete port reads.	N/A
002706	CPW	Complete port writes.	N/A
0030*j*1	VM1 S*j*	Transmit (S*j*) to VM upper register.	N/A
0034*jk*	SM,A*k* 1,TS	Test and set semaphore (A*k*) (*j*2 = 1).	N/A
0036*jk*	SM,A*k* 0	Clear semaphore (A*k*) (*j*2 = 1).	N/A
0037*jk*	SM,A*k* 1	Set semaphore (A*k*) (*j*2 = 1).	N/A
0051*jk*	J B*jk*	Jump to (B*jk*). (Maintenance only: invalidates instruction buffers.)	N/A
006000 *nm*	J *exp*	Jump to *nm*.	N/A
0064*jk nm*	JTS*jk exp*	Branch to *nm* if SM*jk* = 1; else set SM*jk* = 1 (*j*2 = 0).	N/A
0064*jk nm*	JTS,A*k exp*	Branch to *nm* if SM,A*k* = 1; else set SM,A*k* = 1 (*j*2 = 1).	N/A
007000 *nm*	R *exp*	Return jump to *nm*; set B00 to (P) + 3.	N/A
010000 *nm*	JAZ *exp*	Jump to *nm* if (A0) = 0.	N/A
011000 *nm*	JAN *exp*	Jump to *nm* if (A0) ≠ 0.	N/A
012000 *nm*	JAP *exp*	Jump to *nm* if (A0) ≥ 0.	N/A
013000 *nm*	JAM *exp*	Jump to *nm* if (A0) < 0.	N/A
014000 *nm*	JSZ *exp*	Jump to *nm* if (S0) = 0.	N/A
015000 *nm*	JSN *exp*	Jump to *nm* if (S0) ≠ 0.	N/A
016000 *nm*	JSP *exp*	Jump to *nm* if (S0) ≥ 0.	N/A

Table 7-1. CRAY Y-MP C90 and CRAY Y-MP Instruction Comparison (continued)

Machine Instruction	CAL Syntax	CRAY Y-MP C90 Function	CRAY Y-MP Function
017000 *nm*	JSM *exp*	Jump to *nm* if (S0) < 0.	N/A
023*i*01	A*i* VL	Transmit (VL) to A*i*. (Maximum VL = 128.)	Transmit (VL) to A*i*. (Maximum VL = 64.)
026*ij*4	A*i* SB,A*j*,+1	Transmit (SB) designated by (A*j*) to A*i*; increment by 1.	N/A
026*ij*5	A*i* SB*j*,+1	Transmit (SB*j*) to A*i*; increment by 1.	N/A
026*ij*6	A*i* SB,A*j*	Transmit (SB) designated by (A*j*) to A*i*.	N/A
027*ij*6	SB,A*j* A*i*	Transmit (A*i*) to SB designated by (A*j*).	N/A
033*ij*1	A*i* CE,A*j*	Transmit error flag of channel (A*j*) to A*i* ($j \neq 0$); include done flag.	Transmit error flag of channel (A*j*) to A*i* ($j \neq 0$).
040*i*20 *nm*	S*i* *exp*	Transmit *nm* to S*i* bits $2^0 - 2^{31}$. (Bits $2^{32} - 2^{63}$ are unchanged.)	N/A
040*i*40 *nm*	S*i* *exp*	Transmit *nm* to S*i* bits $2^{32} - 2^{63}$. (Bits $2^0 - 2^{31}$ are unchanged.)	N/A
072*ij*6	S*i* ST,A*j*	Transmit (ST) designated by (A*j*) to S*i*.	N/A
073*i*10	S*i* VM1	Transmit VM1 to S*i*.	N/A
073*i*21	S*i* SR2	Read PM counters 00 – 17 and increment pointer.	Increment performance counter.
073*i*25	SR2 S*i*	Issue PM maintenance advance.	N/A
073*i*31	S*i* SR3	Read PM counters 20 – 37 and increment pointer.	Clear all maintenance modes.
073*i*75	SR7 S*i*	Transmit (S*i*) to maintenance mode register.	N/A
073*ij*1	S*i* SR*j*	Transmit (SR*j*) to S*i*.	Transmit (SR*j*) to S*i*.
073*ij*6	ST,A*j* S*i*	Transmit (S*i*) to ST designated by (A*j*).	N/A

Table 7-1. CRAY Y-MP C90 and CRAY Y-MP Instruction Comparison (continued)

Machine Instruction	CAL Syntax	CRAY Y-MP C90 Function	CRAY Y-MP Function
005400 150*ij*0	V*i* V*j* < V0	Shift (V*j*) left (V0) places to V*i*.	N/A
005400 151*ij*0	V*i* V*j* > V0	Shift (V*j*) right (V0) places to V*i*.	N/A
005400 152*ijk*	V*i* V*j*,A*k*	Transfer (V*j*) to (V*i*) starting at element (A*k*).	N/A
174*ij*3	V*i* ZV*j*	Transmit leading zero count of (V*j*) to V*i*.	N/A
073*ij*5	SR*j* S*i*	Transmit (S*i*) to SR*j*.	N/A

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. This special operand is always available regardless of existing A0 or S0 reservations, because data from these registers is not used. This special operand does not alter the actual value of the S0 or A0 register. If register S0 or A0 is referenced in the *i* field as an operand, the value stored in the register is used. Cray Assembly Language (CAL) issues a caution-level error message for A0 or S0 when 0 does not apply to the *i* field. Table 7-2 lists the special register values.

Table 7-2. Special Register Values

Field	Operand Value
A*h*, *h* = 0	0
A*j*, *j* = 0	0
A*k*, *k* = 0	1
S*j*, *j* = 0	0
S*k*, *k* = 0	$2^{63} = 1$

Monitor Mode Instructions

The monitor mode instructions (channel control, set real-time clock, programmable clock interrupts, and so on) perform specialized functions 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.

Monitor mode instructions are indicated by the letter C in the mode column in "CPU Instruction Descriptions" later in this section.

Special CAL Syntax Forms

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 CAL instruction set.

For example, the following CAL instructions generate machine instruction 002000, which enters a 1 into the vector length (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 instruction 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.

CAL instructions with a special syntax form are identified by the § symbol in the following subsection.

CPU Instruction Descriptions

This subsection describes all instructions used by the CRAY Y-MP C90 mainframe. The instruction descriptions use acronyms and abbreviations that were defined in previous sections.

The following information is included with each instruction description:

- Special cases
- Hold issue conditions
- Execution time
- Description

In some instructions, register designators are prefixed by the following letters, which 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

Instructions pertaining to functional units use the following characters to indicate the functional operations:

Character	Operation
+	Arithmetic sum of specified registers
–	Arithmetic difference of specified registers
*	Arithmetic product of specified registers
/	Reciprocal 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 exclusive OR of specified registers

An expression (*exp*) occupies the *jk*, *ijkm*, or *mn* 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.

CAL instructions with a special syntax form are followed by the § symbol in the tables listing the instructions.

The following letter codes are used in the mode column of the tables listing the instructions:

Letter	Description
A	Instruction supported only in C90 mode
B	Instruction supported only in Y-MP mode
C	Instruction supported only in monitor mode

Functional Units Instruction Summary

Instructions other than simple transmit or control operations are performed by specialized hardware components known as functional units. Listed below are the machine 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 057
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 through 153
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)	10*h* through 13*h*
Memory (vector)	176, 177

Instruction 000000			
Mode	Machine Instruction	CAL Syntax	Description
	000000	ERR	Error exit

Special Cases

If the FEX mode (enable flag on error exit) is not set, instruction 000000 does not perform any operation.

Hold Issue Conditions

The instruction holds issue if any A, S, or V register is reserved or if an instruction fetch operation is in progress.

Execution Time

Instruction 000000 issues in 1 CP. Following the instruction issue, an additional 62 CPs are required: 35 CPs for an exchange sequence and 27 CPs for a fetch operation. Memory conflicts during the exchange sequence cause additional delays.

Description

If the FEX mode is set when instruction 000000 issues, the error exit (EEX) interrupt flag is set. An exchange sequence is initiated, invalidating the contents of the instruction buffers. All instructions issued before the 004000 instruction, however, run to completion.

When the results of previously issued instructions arrive at the operating registers, an exchange occurs, with control being shifted to the exchange package located at the address designated by the contents of the XA register. The program address stored during the final exchange sequence is obtained by adding 1 to the address contained in the P register. This derived address is the address of the instruction immediately following the error exit instruction.

Instruction 000000 is not generally used in program code. This instruction stops execution of an incorrectly coded program that branches to an unused area of memory (if memory was backgrounded with 0's) or into a data area (if the data is positive integers, right justified ASCII, or floating-point 0's).

Instructions 0010 through 0012			
Mode	Machine Instruction	CAL Syntax	Description
C	0010*jk*	CA,A*j* A*k*	Set the CA register for channel (A*j*) to (A*k*) and begin I/O sequence.
	001000	PASS	This is a no-operation instruction. It does not cause an MII interrupt.
C	0011*jk*	CL,A*j* A*k*	Set the CL register for channel (A*j*) to (A*k*).
C	0012*j*0	CL,A*j*	Clear the interrupt and error flags for channel (A*j*); clear device master clear (output channels only); enable channel interrupt.
C	0012*j*1	MC,A*j*	Clear the interrupt and error flags for channel (A*j*); set device master clear (output channels only); clear device ready held (input channels only).
C	0012*j*2	DI,A*j*	Disable channel (A*j*) interrupts.
C	0012*j*3	EI,A*j*	Enable channel (A*j*) interrupts.

Special Cases

If the program is not in monitor mode, and IMI mode (interrupt on 001*ijk*; $k \neq 0$) is not set, these instructions become no-operation instructions with all hold issue conditions remaining effective.

If the program is not in monitor mode, and IMI mode is set, these instructions, with the exception of instruction 001000, cause an exchange to occur. Registers are not updated, and the P register contains the address of the parcel following the instruction.

Special cases for instructions 0010, 0011, and 0012 are as follows:

- If $j = 0$, the instruction performs no operation.
- If $k = 0$, the CA or CL register is set to 1.
- Valid channel numbers are 3_8, 7_8, 13_8, 17_8, 23_8, 27_8, 33_8, and 37_8 for the VHISP (very high speed) channels and 40_8 through 77_8 for the LOSP (low speed) channels.

When interrupts occur, the channel number of the highest priority interrupting channel cannot be read by instruction 033 until 1 CP after issue of a 0012 instruction.

Hold Issue Conditions

Instructions 0010 through 0012 hold issue for 3 CPs and continue to hold issue if a shared register access conflict occurs with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information on shared register access conflicts.

The 0010*jk* and 0011*jk* instructions hold issue if the A*j* or the A*k* register is reserved (except A0).

Instructions 0010 and 0011 hold issue if a 033 instruction is in CP 1 through 15.

The 0012 instructions hold issue if the A*j* register is reserved (except A0).

Execution Time

The instruction issue time for instructions 0010 through 0012 is 1 CP.

Description

Instructions 0010 through 0012 are privileged to monitor mode and provide operations useful to the operating system. Functions are selected through the *i* designator. Instructions are treated as pass instructions if the monitor mode bit is not set. A monitor program activates a user job by initializing the XA register to point to the user job's exchange package and then by executing a normal exit instruction.

When the *j* designator is 0, the functions are executed as pass instructions. When the *k* designator is 0, the CA register or the CL register is set to 1. In the CRAY Y-MP C90 computer system, valid channel numbers are 3_8, 7_8, 13_8, 17_8, 23_8, 27_8, 33_8, 37_8 (VHISP, or 1800-Mbyte/s channel), and 40_8 through 77_8 (LOSP, or 6- or 20-Mbyte/s channel).

Instructions 0010, 0011, and 0012 control operation of the I/O channels. Each LOSP channel has a CA and a CL register to direct channel activity. The CA register contains the address of the current channel word; the CL register specifies the limit address. When programming the channel, the CL register should be initialized first, and then the CA register should be set. Setting the CA register activates the channel and begins the data transfer. During the transfer, the CA register increments by 1 after each word is transferred. When the contents of the CA register are equal to the contents of the CL register, the transfer is complete. All words

between (CA) and (CL) – 1 are transferred; that is, all words starting at the initial address stored in the CA register through 1 less than the address stored in the CL register are transferred.

The 001000 instruction functions as a pass instruction; it does not perform any operations. This instruction does not cause an MII interrupt because the *k* field of the instruction is 0.

The 0010*jk* instruction sets the CA register for the channel indicated by the contents of the A*j* register to the address specified in the A*k* register. The 0011*jk* instruction sets the CL register for the channel indicated by the contents of the A*j* register to the address specified in the A*k* register. The 0011*jk* instruction is usually issued before the 0010*jk* instruction is issued.

Instruction 0012*j*0 clears the interrupt and error flags for the channel indicated by the contents of the A*j* register; if the contents of the A*j* register specify an output channel, the device master clear is cleared.

Instruction 0012*j*1 also clears the interrupt and error flags for the channel indicated by the contents of the A*j* register; if the contents of the A*j* register specify an output channel, the device master clear is set; if the contents of the A*j* register specify an input channel, the device ready flag is cleared.

Each LOSP and VHISP channel has a channel interrupt enable mode bit, which is toggled with the 0012*j*2 and 0012*j*3 instructions.

The 0012*j*2 instruction disables the interrupts from the channel specified by the contents of the A*j* register.

The 0012*j*3 instruction enables the interrupts from the channel specified by the contents of the A*j* register.

Instruction 0013			
Mode	Machine Instruction	CAL Syntax	Description
C	0013*j*0	XA A*j*	Transmit (A*j*) to the XA register.
C	001302	EMI	Enable monitor mode interrupt modes.
C	001303	DMI	Disable monitor mode interrupt modes.

Special Cases

If the program is not in monitor mode, and IMI mode is not set, these instructions become no-operation instructions with all hold issue conditions remaining effective.

If the program is not in monitor mode, and IMI mode is set, these instructions cause an exchange to occur. Registers are not updated, and the P register contains the address of the parcel following the instruction.

For instruction 0013*j*0, if $j = 0$, the XA register is cleared.

Hold Issue Conditions

There are none.

Execution Time

Instruction 0013 issues in 1 CP.

For instruction 001302, the interrupt modes are enabled in 3 CPs.

For instruction 001303, the interrupt modes are disabled in 13 CPs.

NOTE: In monitor mode, the software must ensure that only one CPU at a time is servicing an I/O channel.

Description

Instruction 0013*j*0 transmits bits 2^{11} through 2^4 of the A*j* register to the XA register. The XA register is cleared when the *j* designator is 0.

Each CPU has an EIM (enable interrupt modes) bit. The 001302 instruction sets the EIM flag (EIM = 1), and the 001303 instruction clears the EIM flag (EIM = 0). An exchange to monitor mode clears the EIM flag, providing a stable environment within monitor mode immediately following an exchange. An exchange to nonmonitor mode sets the EIM flag.

- If EIM = 0, all interrupt modes except FNX, FEX, and IPR are disabled in monitor mode.
- If EIM = 0, the following interrupts are held in monitor mode: RTI, MCU, MEC, BPI, ORE, FPE, RPE, and MEU.
- If EIM = 1, then all interrupts (or held interrupts) corresponding to set interrupt modes are allowed.
- I/O interrupts are allowed only if EIM is set, IIO interrupt mode is set, and the SIE flag is set.

Instruction 0014			
Mode	Machine Instruction	CAL Syntax	Description
C	0014*j*0	RT S*j*	Transmit (S*j*) to the RTC register.
C	0014*j*1	SIPI A*j*	Send an interprocessor interrupt request to CPU (A*j*).
C	001401	SIPI	Send an interprocessor interrupt request to CPU 0.
C	001402	CIPI	Clear the interprocessor interrupt request.
C	0014*j*3	CLN A*j*	Transmit (A*j*) to the CLN register.
C	0014*j*4	PCI S*j*	Transmit (S*j*) to the II register.
C	001405	CCI	Clear the programmable clock interrupt request.
C	001406	ECI	Enable the programmable clock interrupt request.
C	001407	DCI	Disable the programmable clock interrupt request.

Special Cases

If the program is not in monitor mode, and IMI mode is not set, these instructions become no-operation instructions with all hold issue conditions remaining effective.

If the program is not in monitor mode, and IMI mode is set, these instructions cause an exchange to occur. Registers are not updated, and the P register contains the address of the parcel following the instruction.

Hold Issue Conditions

Instructions 0014*j*0 and 0014*j*1 hold issue if the S*j* register is reserved (except S0).

Instructions 0014*j*0, 0014*j*1, 0014*j*3, and 0014*j*4 hold issue if the A*j* register is reserved (except A0).

Instructions 0014*j*0 through 0014*j*3 hold issue for 3 CPs and continue to hold if a shared paths access conflict exists with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Execution Time

Instruction 0014 issues in 1 CP.

The RTC register does not contain valid data until 8 CPs after instruction 0014*j*0 has issued.

Description

The 0014 instructions perform specialized functions for managing the real-time and programmable clocks. These functions process interprocessor interrupt requests and cluster number operations. The 0014 instructions are privileged to monitor mode and are treated as pass instructions if the monitor mode bit is not set.

The 0014*j*0 instruction loads the contents of the S*j* register into the RTC register; the lower 4 bits of the S*j* register are forced to 0 when loaded into the RTC register. The RTC register is set to 0 when the *j* designator is 0.

The 0014*j*1 instruction sets the CPU interrupt request in the CPU specified by the contents of the A*j* register. If the CPU named in the contents of the A*j* register is the CPU issuing the instruction (if a CPU attempts to interrupt itself) the instruction performs no operation. If the named CPU has IIP mode set and enabled, the interrupt-from-internal CPU (ICP) flag sets in that CPU, causing an interrupt. The request remains until the receiving CPU issues instruction 001402, which clears the request. Instruction 001401 performs the same function as instruction 0014*j*1, except that it sets the internal CPU interrupt request in CPU 0.

Instruction 001402 clears the internal CPU interrupt request set by another CPU.

The 0014*j*3 instruction sets the cluster number to the contents of the A*j* register to make 1 of 18 cluster selections. The cluster number 0 causes all shared and semaphore register operations to be no-operation instructions (except SB, ST, or SM register reads, which return a zero value to the A*i* or S*i* register). A nonzero cluster number allows access to a separate set of SM, SB, and ST registers. A cluster number larger than 18_{10} produces undefined results.

The 0014*j*4 instruction loads the 32 low-order bits from the S*j* register into the interrupt interval (II) register and programmable clock. The programmable clock is a 32-bit counter, the contents of which are decremented by 1 each CP until equal to 0, which sets the programmable clock interrupt request. The counter is then reset to the interval value held in the II register, and the counter repeats the countdown to 0. When

a programmable clock interrupt request is set, it remains set until a 001405 instruction is executed. Refer to "Interrupt Interval Register" in Section 3 for more information on the II register.

The 001405 instruction clears the programmable clock interrupt request if the request is set previously by the programmable clock counting down to 0.

The 001406 instruction enables repeated programmable clock interrupt (PCI) requests by setting the IPC interrupt mode in the monitor mode exchange package.

The 001407 instruction disables repeated PCI requests by clearing the IPC interrupt mode in the monitor mode exchange package.

Instruction 0015			
Mode	Machine Instruction	CAL Syntax	Description
C	001500		Clear all performance monitor counters.

Special Cases

If the program is not in monitor mode, and IMI mode is not set, instruction 0015 becomes a no-operation instruction with all hold issue conditions remaining effective.

If the program is not in monitor mode, and IMI mode is set, instruction 0015 causes an exchange to occur. Registers are not updated, and the P register contains the address of the parcel following the instruction.

Instruction 0015 should not be issued while the performance monitor is busy. Bit 2^{47} of status register 0 is set when the performance monitor is busy.

Hold Issue Conditions

There are no hold issue conditions.

Execution Time

Instruction 0015 issues in 1 CP.

After instruction 0015 issues, the performance monitor is busy for 71 CPs.

Description

Instruction 0015 clears all the performance monitor counters.

Instruction 001600			
Mode	Machine Instruction	CAL Syntax	Description
C	001600	ESI	Enable system I/O interrupts (SIE = 1).

Special Cases

If the program is not in monitor mode, and IMI mode is not set, this instruction becomes a no-operation instruction with all hold issue conditions remaining effective.

If the program is not in monitor mode, and IMI mode is set, this instruction causes an exchange to occur. Registers are not updated, and the P register contains the address of the parcel following the instruction.

Hold Issue Conditions

Instruction 001600 holds issue for 3 CPs and continues to hold if a shared paths access conflict exists with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Execution Time

Instruction 001600 issues in 1 CP.

Description

Instruction 001600 is designed to enhance I/O throughput by eliminating I/O interrupt blocks that arise when monitor mode is used simultaneously in several CPUs. Once an I/O interrupt occurs, no further I/O interrupts are allowed in any CPU regardless of the state of any CPU's IIO flag. The CPU that receives the I/O interrupt can issue instruction 001600 to re-enable system I/O interrupts. These interrupts are directed to the lowest-numbered CPU with an IIO interrupt mode set and enabled, whether or not the CPU is in monitor mode. Instruction 001600 should be issued only after the interrupting channel is serviced; otherwise, the channel interrupts another CPU.

Instruction 0017			
Mode	Machine Instruction	CAL Syntax	Description
C	0017*jk*	BP,*k* A*j*	Transmit (A*j*) to breakpoint address *k*.

Special Cases

If the program is not in monitor mode, and IMI mode is not set, this instruction becomes a no-operation instruction with all hold issue conditions remaining effective.

If the program is not in monitor mode, and IMI mode is set, this instruction causes an exchange to occur. Registers are not updated, and the P register contains the address of the parcel following the instruction.

If $k = 0$, the breakpoint base address = (A*j*).

If $k = 1$, the breakpoint limit address = (A*j*).

Hold Issue Conditions

Instruction 0017 holds issue if A*j* is reserved (except A0).

Execution Time

Instruction 0017 issues in 1 CP.

Description

This instruction transmits the contents of register A*j* to breakpoint address *k*. A breakpoint interrupt occurs if the breakpoint interrupt mode is enabled (IBP = 1) and if a write reference is made within the breakpoint range.

The breakpoint range is saved and restored by the operating system during an exchange.

Instruction 0020			
Mode	Machine Instruction	CAL Syntax	Description
	00200*k*	VL A*k*	Transmit (A*k*) to the VL register.
	002000	VL 1 §	Transmit 1 to the VL register.

Special Cases

If $k = 0$, the instruction transmits a 1 to the VL register.

The following conditions apply in C90 mode:

- The maximum vector length is 128.
- If $k \neq 0$ and (A*k*) = 0 or a multiple of 200_8, the instruction transmits 200_8 to the VL register.

The following conditions apply in Y-MP mode:

- The maximum vector length is 64.
- If $k \neq 0$ and (A*k*) = 0 or a multiple of 100_8, the instruction transmits 100_8 to the VL register.

Hold Issue Conditions

Instruction 0020 holds issue if the A*k* register is reserved (except A0).

Execution Time

Instruction 0020 issues in 1 CP.

The VL register is ready 2 CPs after instruction issue.

Description

Instruction 00200*k* transmits the contents of the 6 or 7 lowest-order bits of A*k* to the VL register.

In C90 mode, the 7 low-order bits of the A*k* register are entered into the VL register; the eighth bit of the VL register is set if the 7 low-order bits of the A*k* register equal 0; if the contents of the A*k* register equal 0 or a multiple of 200_8, then VL = 200_8. The contents of the VL register are always between 1 and 200_8.

In Y-MP mode, the 6 low-order bits of the A*k* register are entered into the VL register; the seventh bit of the VL register is set if the 6 low-order bits of the A*k* register equal 0; if the contents of the A*k* register equal 0 or a multiple of 100_8, then VL = 100_8. The contents of the VL register are always between 1 and 100_8.

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

Instructions 0021 through 0026			
Mode	Machine Instruction	CAL Syntax	Description
	002100	EFI	Enable interrupt on floating-point error.
	002200	DFI	Disable interrupt on floating-point error.
	002300	ERI	Enable interrupt on operand range error.
	002301	EBP	Enable interrupt on breakpoint.
	002400	DRI	Disable interrupt on operand range error.
	002401	DBP	Disable interrupt on breakpoint.
	002500	DBM	Disable bidirectional memory transfers.
	002600	EBM	Enable bidirectional memory transfers.

Special Cases

There are no special cases.

Hold Issue Conditions

Instructions 0021 through 0026 hold issue if the status register is busy.

Execution Time

These instructions issue in 1 CP.

The status register remains busy for 5 CPs plus 3 CPs after the following conditions are satisfied:

- There are no scalar memory references in CPs 1 through 3.
- Ports A, B, and C are not busy.
- No floating-point instructions were issued in the preceding CP.

Description

Instructions 002100 and 002200 set and clear the interrupt-on-floating-point-error (IFP) interrupt mode. When this interrupt mode is set and enabled, it allows interrupts on floating-point

range errors. These two instructions do not check the previous state of the IFP interrupt mode. Issuing either of these instructions also clears the floating-point error status (FPS) bit.

Instructions 002300 and 002400 set and clear the interrupt-on-operand-range-error (IOR) interrupt mode. When this interrupt mode is set and enabled, it allows interrupts on operand range errors. These two instructions do not check the previous state of the IOR interrupt mode.

Instructions 002301 and 002401 set and clear the interrupt-on-breakpoint (IBP) interrupt mode. When this interrupt mode is set and enabled, it allows interrupts on write references within the breakpoint range, which should be set previously by instruction 0017*jk*. The enabled mode becomes effective after previously issued memory or floating-point operations can no longer cause interrupts.

Instructions 002500 and 002600 disable and enable the bidirectional memory mode. When this mode is enabled, block read and write operations can operate concurrently. When disabled, only block read operations can operate concurrently.

Instruction 0027			
Mode	Machine Instruction	CAL Syntax	Description
	002700	CMR	Complete memory references.
	002704	CPA	Complete port reads and writes.
	002705	CPR	Complete port reads.
	002706	CPW	Complete port writes.

Special Cases

There are no special cases.

Hold Issue Conditions

Instruction 002700 holds issue for any of the following conditions:

- Ports A, B, and C are busy.
- There is a scalar instruction in CPs 1 through 12.
- There is a block instruction in CPs 1 through 12.

Instruction 002704 holds issue for any of the following conditions:

- Ports A and B are busy.
- Port C is busy.
- There is a scalar instruction in CPs 1 through 6.

Instruction 002705 holds issue if ports A and B are busy or if there is a scalar instruction in CPs 1 through 6.

Instruction 002706 holds issue if port C is busy or if there is a scalar instruction in CPs 1 through 6.

Execution Time

Instruction 027 issues in 1 CP.

Description

Instruction 002700 ensures completion of all memory references within the particular CPU issuing the instruction. Instruction 002700 does not issue until all previous memory references can complete in a fixed number of CPs. For example, a CPU is assured of receiving updated data when it issues a data load instruction after a 002700 instruction. Used in conjunction with semaphore instructions, this instruction synchronizes memory references between processors.

Instructions 002704 through 002706 can be used to ensure sequential memory referencing within a CPU. These instructions do not issue until all previous memory references are at a stage of execution such that they can run to completion before any subsequent memory references. Instruction 002704 ensures that all read and write operations are at this stage. Instruction 002705 ensures that all read operations are at this stage, while instruction 002706 ensures that all write operations are at this stage.

Instruction 0030			
Mode	Machine Instruction	CAL Syntax	Description
	0030*j*0	VM S*j*	Transmit (S*j*) to VM lower register.
	003000	VM 0 §	Clear VM register.
A	0030*j*1	VM1 S*j*	Transmit (S*j*) to VM upper register.
A	003001	VM1 0 §	Clear VM1 register.

Special Cases

If $j = 0$, then (Sj) = 0.

Hold Issue Conditions

The 0030 instructions hold issue for any of the following conditions:

- The S*j* register is reserved (except S0).
- Instruction 14x is in progress in the full vector logical functional unit; the functional unit is busy for (VL/2) + 4 CPs.
- Instruction 175 is in progress; the functional unit is busy for (VL/2) + 4 CPs.

Execution Time

Instruction 0030 issues in 1 CP.

The full vector logical functional unit remains busy for 2 CPs when 14x and 175 instructions attempt to issue, and it remains busy for 3 CPs when 073*i*00 and 073*i*10 instructions attempt to issue.

Description

Instruction 0030 assigns values to the vector mask registers. The vector merge instructions (146 and 147) then use these values to perform vector and scalar merge operations.

Instruction 0030*j*0 transmits the contents of the S register specified by *j* to the VM register. Bits 2^{63} through 2^{0} of the VM register correspond to elements 0 through 63 of a vector.

Instruction 003000 clears the VM register.

Instruction 0030*j*1 transmits the contents of the S register specified by *j* to the VM1 register. Bits 2^{63} through 2^{0} of the VM1 register correspond to elements 64 through 127 of a vector.

Instruction 003001 clears the VM1 register.

Instructions 0034, 0036, and 0037			
Mode	Machine Instruction	CAL Syntax	Description
	0034*jk*	SM*jk* 1,TS	Test and set semaphore *jk*, 0 < *jk* < 31_{10} (*j*2 = 0).
	0034*jk*	SM,A*k* 1,TS	Test and set semaphore (A*k*), 0 < (A*k*) < 31_{10} (*j*2 = 1).
	0036*jk*	SM*jk* 0	Clear semaphore *jk*, 0 < *jk* < 31_{10} (*j*2 = 0).
	0036*jk*	SM,A*k* 0	Clear semaphore (A*k*), 0 < (A*k*) < 31_{10} (*j*2 = 1).
	0037*jk*	SM*jk* 1	Set semaphore *jk*, 0 < *jk* < 31_{10} (*j*2 = 0).
	0037*jk*	SM,A*k* 1	Set semaphore (A*k*), 0 < (A*k*) < 31_{10} (*j*2 = 1).

Special Cases

Instructions 0034*jk*, 0036*jk*, and 0037*jk* perform no operation if CLN = 0.

Hold Issue Conditions

Instruction 0034*jk* has the following hold issue conditions:

- If *j*2 = 1 and A*k* is reserved (except A0), the instruction holds issue.
- If the current cluster number ≠ 0, and SM*jk* is set (*j*2 = 0) or SM,A*k* is set (*j*2 = 1), the instruction holds issue until a CPU in the same cluster clears the semaphore register.
- This instruction holds issue for 5 CPs and continues to hold issue if a shared register access conflict occurs with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Instructions 0036*jk* and 0037*jk* have the following hold issue conditions:

- If *j*2 = 1 and A*k* is reserved (except A0), these instructions hold issue.
- These instructions hold issue for 3 CPs and continue to hold issue if a shared paths access conflict occurs with another CPU.

Execution Time

Instructions 0034 through 0037 each issue in 1 CP.

Description

There are thirty-two 1-bit semaphore (SM) registers, numbered SM0 through $SM37_8$; SM0 is the most significant semaphore register. Instructions 0034*jk* through 0037*jk* designate a particular SM register using either the combined *jk* fields of the instruction or (A*k*). If bit 2^2 of the *j* field = 0, then the combined *jk* fields select the semaphore register; if bit 2^2 of the *j* field = 1, then the contents of the A*k* register select the semaphore register. The highest SM register that can be selected by the *jk* fields of the instruction is 37_8. A number higher than 37_8 entered into the *jk* fields sets bit 2^2 of the *j* field, causing (A*k*) to select the semaphore register.

Instruction 0034*jk* tests and sets the SM register designated either by the combined *jk* fields of the instruction or by the contents of A*k*. If the designated SM register is clear, instruction 0034*jk* issues and sets the SM register. If the designated SM register is set, the instruction holds issue until another CPU clears that SM register; the instruction then issues and sets the SM register. If all CPUs in a given cluster are holding issue on a test and set instruction, the deadlock (DL) flag is set in the exchange package (if not in monitor mode), and an exchange occurs.

While a 0034*jk* instruction is holding in the CIP register, the waiting-on-semaphore (WS) bit in the status field of the exchange package is set. If an interrupt occurs, the CIP and NIP registers are cleared, the P register is set to the address of the 0034*jk* instruction, and an exchange occurs.

Instruction 0036*jk* clears the SM register designated by either the combined *jk* fields of the instruction or by the contents of A*k*.

Instruction 0037*jk* sets the SM register designated by either the combined *jk* fields of the instruction or by the contents of A*k*.

Instruction 004000			
Mode	Machine Instruction	CAL Syntax	Description
	004000	EX	Normal exit from the operating system.

Special Cases

There are no special cases.

Hold Issue Conditions

Instruction 004000 holds issue if any A, S, or V register is reserved or if an instruction fetch is in progress.

Execution Time

Instruction 004000 issues in 1 CP. Following the instruction issue, 62 CPs are required for an exchange sequence (35 CPs) and a fetch operation (27 CPs). Memory conflicts during the exchange sequence or fetch operation cause additional delays.

Description

Instruction 004000 is used to call a monitor program from a user program or to transfer control from a monitor program to another program. If the FNX mode is set when instruction 004000 issues, the normal exit (NEX) interrupt flag is set. An exchange sequence begins, invalidating the contents of the instruction buffers. All instructions issued before instruction 004000, however, run to completion.

Instructions 0050 and 0051			
Mode	Machine Instruction	CAL Syntax	Description
	0050*jk*	J B*jk*	Jump to (B*jk*).
	0051*jk*	Jinv B*jk*	Jump to (B*jk*). (Maintenance only: invalidates instruction buffers.)

Special Cases

Instructions 0050 and 0051 execute as 2-parcel instructions. The parcel following the single parcel of these instructions is not used; however, a delay occurs if this second parcel is not in the instruction buffer.

Hold Issue Conditions

Instruction 0050 and 0051 hold issue for any of the following conditions:

- Instruction 034 or 035 is in progress.
- Instruction 025 was issued in the preceding CP.
- The second parcel of the instruction is in another buffer (a 3-CP delay occurs).
- The second parcel of the instruction is not in an instruction buffer (a 26-CP delay occurs).

Instruction 0051 also holds issue if a fetch is active.

Execution Time

If the instruction parcel and the following parcel are in the same buffer, and the branch address is in a buffer, the issue time is 8 CPs.

If the instruction parcel and the following parcel are both in a buffer, and the branch address is not in a buffer, the issue time is 31 CPs. Additional time is required if a memory conflict exists.

Description

Instructions 0050 and 0051 set the P register to the 32-bit (24-bit if in Y-MP mode) parcel address contained in the B register specified by *jk*, causing the program to continue at that address. These instructions are used to exit a subroutine and return to the calling program.

Instruction 0051 also clears the buffer valid bits, invalidating any data stored in the buffers and forcing a fetch to occur.

Instruction 006			
Mode	Machine Instruction	CAL Syntax	Description
B	006*ijkm*	J *exp*	Jump to *exp*.
A	006000 *nm*	J *exp*	Jump to *exp*.
A	0064*jk nm*	JTS*jk exp*	Branch to *exp* if (SM*jk*) = 1; else set SM*jk* (*j*2 = 0).
A	0064*jk nm*	JTS,A*k exp*	Branch to *exp* if (SM,(A*k*)) = 1; else set SM,(A*k*) (*j*2 = 1).

Special Cases

For instruction 006*ijkm*, the high-order bit of the *i*-designator (*i*2) must be 0.

Hold Issue Conditions

Instruction 006 holds issue for any of the following conditions:

- The second parcel of the instruction is in another buffer (a 3-CP delay occurs).
- The third parcel (of instructions 006000 *nm* through 0064) is in another buffer (a 4-CP delay occurs).
- The second and/or third parcel of the instruction is not in an instruction buffer (a 26-CP delay occurs).

Instruction 0064 holds issue for 5 CPs and continues to hold if a shared paths access conflict exists with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Instruction 0064*jk nm* with *j*2 = 1 holds issue if A*k* is reserved.

Execution Time

The issue times for instruction 006 are as follows:

- If all parcels of the instruction are in the same buffer, and the branch address is in a buffer, the issue time is 6 CPs.

- If all parcels of the instruction are in the same buffer, and the branch address is not in a buffer, the issue time is 29 CPs. Additional time is required if a memory conflict exists.

Description

Instruction 006*ijkm* is a 2-parcel unconditional jump instruction used in Y-MP mode. It sets the P register to the parcel address specified by the 24 low-order bits of the *exp* (*ijkm*) field. Program execution continues at that address.

Instruction 006000 *nm* is a 3-parcel unconditional jump instruction used in C90 mode. It sets the P register to the parcel address specified by the 32 low-order bits of the *exp* (*nm*) field. Program execution continues at that address.

Instruction 0064*jk nm* is a 3-parcel instruction that causes program execution to branch to the address specified in parcels 2 and 3 (*nm* field) if the semaphore register specified by the *jk* field is set to 1. If the selected semaphore register contains a 0, the semaphore register is set to 1, and the next instruction is executed. This instruction performs in this manner if the contents of its combined *jk* field range from 0 through 37_8.

If bit 2^2 of the *j* field is set (the combined *jk* fields contain a number greater than 37_8), the 0064*jk nm* instruction uses the value contained in the A register specified by *k* to select a semaphore register.

Instruction 007			
Mode	Machine Instruction	CAL Syntax	Description
B	007*ijkm*	R *exp*	Return jump to *exp* and set register B00 to (P) + 2.
A	007000 *nm*	R *exp*	Return jump to *exp* and set register B00 to (P) + 3.

Special Cases

There are no special cases.

Hold Issue Conditions

Instruction 007 holds issue for any of the following conditions:

- Instruction 034 or 035 is in progress.
- Instruction 025 was issued in the previous 2 CPs.
- The second parcel of the instruction is in another buffer (a 3-CP delay occurs).
- The second and/or third parcel of the instruction is not in an instruction buffer (a 26-CP delay occurs).

Execution Time

The issue times for instruction 007 are as follows:

- If all parcels of the instruction are in the same buffer, and the branch address is in a buffer, issue time is 6 CPs.
- If all parcels of the instruction are in the same buffer, and the branch address is not in a buffer, issue time is 29 CPs. Additional time is needed if a memory conflict exists.

Description

Instruction 007*ijkm* is a 2-parcel return jump instruction used in Y-MP mode. It sets register B00 to the address of the parcel following the second parcel of the instruction, and sets the P register to the parcel address specified by the 24 low-order bits of the *exp* (*ijkm*) field. Program execution continues at that address.

Instruction 007000 *nm* is a 3-parcel return jump instruction used in C90 mode. It sets register B00 to the address of the parcel following the third parcel of the instruction and sets the P register to the parcel address specified by the 32 bits of the *exp* (*nm*) field. Program execution continues at that address.

This instruction provides return links for subroutine calls. The subroutine is entered through a return jump. The subroutine can return to the caller at the instruction following the call by executing a jump to the contents of register B00 (005000).

Instructions 010 through 013			
Mode	Machine Instruction	CAL Syntax	Description
B	010*ijkm*	JAZ *exp*	Jump to *exp* if (A0) = 0 (*i*2 = 0).
A	010000 *nm*	JAZ *exp*	Jump to *exp* if (A0) = 0.
B	011*ijkm*	JAN *exp*	Jump to *exp* if (A0) ≠ 0 (*i*2 = 0).
A	011000 *nm*	JAN *exp*	Jump to *exp* if (A0) ≠ 0.
B	012*ijkm*	JAP *exp*	Jump to *exp* if (A0) is positive; (A0) ≥ 0 (*i*2 = 0).
A	012000 *nm*	JAP *exp*	Jump to *exp* if (A0) is positive; (A0) ≥ 0.
B	013*ijkm*	JAM *exp*	Jump to *exp* if (A0) is negative (*i*2 = 0).
A	013000 *nm*	JAM *exp*	Jump to *exp* if (A0) is negative.

Special Cases

Special cases for instructions 010 through 013 are as follows:

- (A0) = 0 is interpreted as (A0) positive.
- The high-order bit of the *i* designator (*i*2) must be 0.
- Register A0 is 32 bits wide and 2^{31} is the sign bit.

Hold Issue Conditions

Instructions 010 through 013 hold issue for any of the following conditions:

- Register A0 is busy in any one of the previous 3 CPs.
- The second and/or third parcel of the instruction is in another buffer (a 3-CP delay occurs).
- The second and/or third parcel of the instruction is not in an instruction buffer (a 26-CP delay occurs).

Execution Time

The following issue times are for instructions 010 through 013 if the branch is taken (if the jump conditions are satisfied):

- If all parcels of the instruction are in the same buffer, if the branch is taken, and if the branch address is in a buffer, the issue time is 6 CPs.
- If all parcels of the instruction are in the same buffer, if the branch is taken, and if the branch address is not in a buffer, the issue time is 29 CPs.
- If the second or third parcel of the instruction is in another buffer, if the branch is taken, and if the branch address is in a buffer, the issue time is 9 CPs.
- If the second or third parcel of the instruction is in another buffer, if the branch is taken, and if the branch address is not in a buffer, the issue time is 32 CPs.
- If the second or third parcel of the instruction is not in a buffer, if the branch is taken, and if the branch address is in a buffer, the issue time is 32 CPs.
- If the second or third parcel of the instruction is not in a buffer, if the branch is taken, and if the branch address is not in a buffer, the issue time is 53 CPs.

The following instruction issue times are for instructions 010 through 013 if the branch is not taken (if the jump conditions are not satisfied):

- If all parcels of the instruction are in the same buffer, if the branch is not taken, and if the next instruction is in the same buffer, the issue time is 2 CPs.
- If all parcels of the instruction are in the same buffer, if the branch is not taken, and if the next instruction is in another buffer, the issue time is 5 CPs.
- If all parcels of the instruction are in the same buffer, if the branch is not taken, and if the next instruction is not in a buffer, the issue time is 29 CPs.
- If the second or third parcel of the instruction is in another buffer, and if the branch is not taken, the issue time is 5 CPs.

- If the second or third parcel of the instruction is not in a buffer, and if the branch is not taken, the issue time is 27 CPs.

NOTE: Memory conflicts may cause a delay whenever a fetch operation occurs.

Description

In Y-MP mode, the 2-parcel instructions 010 through 013 test the contents of the A0 register for the condition specified by the *h* field. If the condition is satisfied, the P register is set to the parcel address specified by the 24 low-order bits of the *exp* (*ijkm*) field, and execution continues at that address. The high-order bit (*i2*) of the *ijkm* field must be 0. If the condition is not satisfied, execution continues with the instruction that follows the branch instruction.

In C90 mode, the 3-parcel instructions 010 through 013 test the contents of the A0 register for the condition specified by the *h* field. If the condition is satisfied, the P register is set to the parcel address specified by the 32 bits of the *exp* (*nm*) field, and execution continues at that address. If the condition is not satisfied, execution continues with the instruction that follows the branch instruction.

Instructions 014 through 017			
Mode	Machine Instruction	CAL Syntax	Description
B	014*ijkm*	JSZ *exp*	Jump to *exp* if (S0) = 0 (*i*2 = 0).
A	014000 *nm*	JSZ *exp*	Jump to *exp* if (S0) = 0.
B	015*ijkm*	JSN *exp*	Jump to *exp* if (S0) ≠ 0 (*i*2 = 0).
A	015000 *nm*	JSN *exp*	Jump to *exp* if (S0) ≠ 0.
B	016*ijkm*	JSP *exp*	Jump to *exp* if (S0) is positive; (S0) ≥ 0 (*i*2 = 0).
A	016000 *nm*	JSP *exp*	Jump to *exp* if (S0) is positive; (S0) ≥ 0.
B	017*ijkm*	JSM *exp*	Jump to *exp* if (S0) is negative (*i*2 = 0).
A	017000 *nm*	JSM *exp*	Jump to *exp* if (S0) is negative.

Special Cases

Special cases for instructions 014 through 017 are as follows:

- (S0) = 0 is interpreted as (S0) positive.
- The high-order bit of the *i* designator (*i*2) must be 0.

Hold Issue Conditions

Instructions 014 through 017 hold issue for any of the following conditions:

- Register S0 is busy in any one of the previous 5 CPs.
- The second and/or third parcel of the instruction is in another buffer (a 3-CP delay occurs).
- The second and/or third parcel of the instruction is not in an instruction buffer (a 26-CP delay occurs).

Execution Time

The following issue times are for instructions 014 through 017 if the branch is taken (if the jump conditions are satisfied):

- If all parcels of the instruction are in the same buffer, if the branch is taken, and if the branch address is in a buffer, the issue time is 6 CPs.
- If all parcels of the instruction are in the same buffer, if the branch is taken, and if the branch address is not in a buffer, the issue time is 29 CPs.
- If the second or third parcel of the instruction is in another buffer, if the branch is taken, and if the branch address is in a buffer, the issue time is 9 CPs.
- If the second or third parcel of the instruction is in another buffer, if the branch is taken, and if the branch address is not in a buffer, the issue time is 32 CPs.
- If the second or third parcel of the instruction is not in a buffer, if the branch is taken, and if the branch address is in a buffer, the issue time is 32 CPs.
- If the second or third parcel of the instruction is not in a buffer, if the branch is taken, and if the branch address is not in a buffer, the issue time is 53 CPs.

The following issue times are for instructions 014 through 017 if the branch is not taken (if the jump conditions are not satisfied):

- If all parcels of the instruction are in the same buffer, if the branch is not taken, and if the next instruction is in the same buffer, the issue time is 2 CPs.
- If all parcels of the instruction are in the same buffer, if the branch is not taken, and if the next instruction is in another buffer, the issue time is 5 CPs.
- If all parcels of the instruction are in the same buffer, if the branch is not taken, and if the next instruction is not in a buffer, the issue time is 29 CPs.
- If the second or third parcel of the instruction is in another buffer, and if the branch is not taken, the issue time is 5 CPs.

- If the second or third parcel of the instruction is not in a buffer, and if the branch is not taken, the issue time is 27 CPs.

NOTE: Memory conflicts may cause a delay whenever a fetch operation occurs.

Description

In Y-MP mode, the 2-parcel instructions 014 through 017 test the contents of the S0 register for the condition specified by the *h* field. If the condition is satisfied, the P register is set to the parcel address specified by the 24 low-order bits of the *exp* (*ijkm*) field, and execution continues at that address. The high-order bit (*i2*) of the *ijkm* field must be 0. If the condition is not satisfied, execution continues with the instruction that follows the branch instruction.

In C90 mode, the 3-parcel instructions 014 through 017 test the contents of the S0 register for the condition specified by the *h* field. If the condition is satisfied, the P register is set to the parcel address specified by the 32 bits of the *exp* (*nm*) field, and execution continues at that address. If the condition is not satisfied, execution continues with the instruction that follows the branch instruction.

Instructions 020 through 022			
Mode	Machine Instruction	CAL Syntax	Description
	020*i*00 *nm*	A*i* *exp*	Transmit *exp* (*nm*) to A*i*.
	021*i*00 *nm*	A*i* *exp*	Transmit one's complement of *exp* (*nm*) to A*i*.
	022*ijk*	A*i* *exp*	Transmit *exp* (*jk*) to A*i*.

Special Cases

There are no special cases.

Hold Issue Conditions

Instructions 020 through 022 hold issue for any of the following conditions:

- The A*i* register is reserved.
- The second and/or third parcel of the instruction is not in an instruction buffer (a 26-CP delay occurs).
- An A register access conflict exists; therefore, one of the following is true:
  - Instruction 030 or 031 is in CP 1.
  - Instruction 026 or 027, with $k = 0$ or 1, is in CP 5.
  - Instruction 026, with $k = 4$ through 7, is in CP 7.
  - Instruction 033 is in CP 18.

Execution Time

The following instruction issue times apply to instructions 020 through 022:

- Register A*i* is ready in 1 CP.
- For instructions 020 and 021, issue time is 2 CPs.
- For instruction 022, issue time is 1 CP.

- If parcel 0 is in a different buffer than parcels 1 and 2, issue time is 5 CPs.

- If parcel 2 is in a different buffer than parcels 0 and 1, issue time is 6 CPs.

Description

Instructions 020 through 022 transmit a value determined by *exp* into the A*i* register. The syntax of these instructions differs from most CAL symbolic instructions in that the assembler generates one of these three Cray Research machine instructions depending on the form, value, and attributes of the *exp*.

The assembler generates instruction 022*ijk* if all of the following conditions are true, and if the *jk* fields contain the 6-bit value of *exp*.

- The value of the expression is positive and less than 100_8.
- All symbols (if any) within the expression are previously defined.
- The expression has an absolute relative attribute.

If any of the previous three conditions is not true, the assembler generates one of the 3-parcel instructions 020*i*00 *nm* or 021*i*00 *nm* according to the criteria listed below. The *nm* fields contain the 32-bit value of *exp*.

- If the *exp* value is positive and greater than 77_8 or has either a relocatable or external relative attribute, instruction 020*i*00 *nm* is generated.

- If the *exp* value is negative (not explicitly –1) and has an absolute relative attribute, instruction 021*i*00 *nm* is generated, with the one's complement of *exp* entered into the *nm* field. If the value of *exp* is explicitly –1, instruction 031*i*00 is generated.

Instruction 023			
Mode	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*.

Special Cases

For instruction 023*ij*0, if $j = 0$ then a value of 0 is transmitted to A*i*.

For instruction 023*i*01, two special cases occur:

- In C90 mode, if the 7 low-order bits of the VL register are 0, bit 2^7 in the VL register is set to 1. If any of the 7 low-order bits of the VL register are not 0, bit 2^7 is forced to 0.
- In Y-MP mode, if the 6 low-order bits of the VL register are 0, bit 2^6 in the VL register is set to 1. If any of the 6 low-order bits of the VL register are not 0, bit 2^6 is forced to 0.

The following examples illustrate the special case for C90 mode.

- In C90 mode, if (A1) = 0, the following CAL sequence results in (A2) = 200_8.

```
VL A1
A2 VL
```

- In C90 mode, if (A1) = 723_8, the above CAL sequence results in (A2) = 123_8.

Hold Issue Conditions

Instruction 023 holds issue for any of the following conditions:

- The A*i* register is reserved.
- An A register access conflict exists; therefore, one of the following is true:
  - Instruction 030 or 031 is in CP1.
  - Instruction 026 or 027, with $k = 0$ or 1, is in CP 5.

- Instruction 026, with k = 4 through 7, is in CP 7.
- Instruction 033 is in CP 18.

- Instruction 0020xx was issued in the preceding CP.

Instruction 023*ij*0 holds issue if the S*j* register is reserved (except S0).

Execution Time

Instruction 023 issues in 1 CP.

For instruction 023*ij*0, the A*i* register is ready in 3 CPs.

For instruction 023*i*01, the A*i* register is ready in 1 CP.

Description

Instruction 023*ij*0 transmits the 32 low-order bits of the contents of the S*j* register to the A*i* register. The high-order bits of the S*j* register are ignored. Register A*i* receives the value of 0 if the *j* designator is 0.

Instruction 023*i*01 transmits the contents of the VL register to the A*i* register.

Instructions 024 and 025			
Mode	Machine Instruction	CAL Syntax	Description
	024*ijk*	A*i* B*jk*	Transmit (B*jk*) to A*i*.
	025*ijk*	B*jk* A*i*	Transmit (A*i*) to B*jk*.

Special Cases

There are no special cases.

Hold Issue Conditions

Instructions 024 and 025 hold issue for any of the following conditions:

- Instruction 034 or 035 is in progress.
- Register A*i* is reserved.

Instruction 024 holds issue if instruction 025 was issued in the preceding CP, or if an A register access conflict exists; therefore, one of the following is true:

- Instruction 030 or 031 is in CP1.
- Instruction 026 or 027, with $k = 0$ or 1, is in CP 5.
- Instruction 026, with $k = 4$ through 7, is in CP 7.
- Instruction 033 is in CP 18.

Execution Time

Instructions 024 and 025 issue in 1 CP.

For instruction 024, the A*i* register is ready in 1 CP.

Description

Instruction 024*ijk* transmits the contents of the B register specified by *jk* to the A*i* register.

Instruction 025*ijk* transmits the contents of the A*i* register to the B register specified by *jk*.

Instruction 026			
Mode	Machine Instruction	CAL Syntax	Description
	026*ij*0	A*i* PS*j*	Transmit the population count of (S*j*) to A*i*.
	026*ij*1	A*i* QS*j*	Transmit the population count parity of (S*j*) to A*i*.
	026*ij*4	A*i* SB,A*j*,+1	Transmit (SB) designated by (A*j*) to A*i*, and increment (SB,A*j*) by 1.
	026*ij*5	A*i* SB*j*,+1	Transmit (SB*j*) to A*i*, and increment (SB*j*) by 1.
	026*ij*6	A*i* SB,A*j*	Transmit (SB) designated by (A*j*) to A*i*.
	026*ij*7	A*i* SB*j*	Transmit (SB*j*) to A*i*.

Special Cases

For instructions 026*ij*0 and 026*ij*1, if $j = 0$, then (A*i*) = 0.

For instructions 026*ij*4 through 026*ij*7, if the CLN = 0, then (A*i*) = 0.

For instructions 026*ij*4 and 026*ij*6, if $j = 0$, then (SB0) is transmitted to A*i*.

Hold Issue Conditions

Instruction 026 holds issue if the A*i* register is reserved.

Instructions 026*ij*0 and 026*ij*1 hold issue if the S*j* register is reserved (except S0) or if there is an A register access conflict; therefore, one of the following is true:

- One of the instructions from 026*ij*4 through 026*ij*7 is in CP 2.
- Instruction 033 is in CP 13.

Instructions 026*ij*4 through 026*ij*7 hold issue for 3 CPs and continue to hold if a shared paths access conflict occurs with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Instructions 026*ij*4 and 026*ij*6 hold issue if the A*j* register is reserved.

Execution Time

Instruction 026 issues in 1 CP.

For instructions 026*ij*0 and 026*ij*1, the A*i* register is ready in 6 CPs.

For instructions 026*ij*4 through 026*ij*7, the A*i* register is ready in 8 CPs.

Description

Instructions 026*ij*0 and 026*ij*1 are executed in the population/leading zero count functional unit.

Instruction 026*ij*0 counts the number of 1 bits in the (S*j*) register and enters the result into the 7 low-order bits of the A*i* register. The high-order bits of the A*i* register are cleared. If the S*j* register equals 0, then the value in the A*i* register is cleared to 0.

Instruction 026*ij*1 enters a 0 into the A*i* register if the (S*j*) register has an even number of 1 bits. If the (S*j*) register has an odd number of 1 bits, a 1 is entered into the A*i* register. The high-order bits of the A*i* register are cleared. The actual population count is not transferred.

Instruction 026*ij*4 reads from A*j* the address of a specific SB register. It transmits the contents of that SB register to A*i*, and then increments the contents of that SB register by 1. The A*j* register must be set to a value corresponding to the number of the desired SB register before instruction 026*ij*4 is issued.

Instruction 026*ij*5 transmits the contents of the SB*j* register to A*i*, and then increments the contents of that SB register by 1.

Instruction 026*ij*6 reads from A*j* the address of a specific SB register. It transmits the contents of that SB register to A*i*. No incrementing occurs. The A*j* register must be set to a value corresponding to the number of the desired SB register before instruction 026*ij*6 is issued.

Instruction 026*ij*7 transmits the contents of the SB*j* register to the A*i* register. No incrementing occurs.

Instruction 027			
Mode	Machine Instruction	CAL Syntax	Description
	027*ij*0	A*i* ZS*j*	Transmit leading zero count of (S*j*) to A*i*.
	027*ij*6	SB,A*j* A*i*	Transmit (A*i*) to SB designated by (A*j*).
	027*ij*7	SB*j* A*i*	Transmit (A*i*) to SB*j*.

Special Cases

For instruction 027*ij*0, there are two special cases:

- If *j* = 0, the A*i* register receives the value 64.
- If S*j* is negative, the A*i* register is cleared to 0.

For instruction 027*ij*6, if *j* = 0, (A*i*) is transmitted to SB0.

For instructions 027*ij*6 and 027*ij*7, if CLN = 0, the instructions perform no operation.

Hold Issue Conditions

Instruction 027 holds issue if the A*i* register is reserved.

Instruction 027*ij*0 holds issue if the S*j* register is reserved (except S0) or if an A register access conflict occurs; therefore, one of the following is true:

- One of the instructions from 026*ij*4 through 026*ij*7 is in CP 2.
- Instruction 033 is in CP 13.

Instructions 027*ij*6 and 027*ij*7 hold issue for 3 CPs and continue to hold if a shared paths access conflict occurs with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Instruction 027*ij*6 holds issue if the A*j* register is reserved.

Execution Time

Instruction 027 issues in 1 CP.

For instruction 027*ij*0, the A*i* register is ready in 6 CPs.

For instructions 027*ij*6 and 027*ij*7, the SB*j* register is ready in 1 CP.

Description

Instruction 027*ij*0 counts the number of leading 0's in the S*j* register and enters the result into the 7 low-order bits of the A*i* register. All bits above bit 28 in the A*i* register are cleared. The A*i* register is set to 64 if the *j* designator is 0 or if the contents of the S*j* register are 0. Instruction 027*ij*0 executes in the population/leading zero count functional unit.

Instruction 027*ij*6 transmits the contents of the A*i* register to the SB register designated by the contents of A*j*.

Instruction 027*ij*7 transmits the contents of the A*i* register to the SB*j* register.

Instructions 030 and 031			
Mode	Machine Instruction	CAL Syntax	Description
	030*ijk*	A*i* A*j* + A*k*	Transmit the integer sum of (A*j*) and (A*k*) to A*i*.
	030*i*0*k*	A*i* A*k* §	Transmit (A*k*) to A*i*.
	030*ij*0	A*i* A*j* + 1 §	Transmit the integer sum of (A*j*) and 1 to A*i*.
	031*ijk*	A*i* A*j* – A*k*	Transmit the integer difference of (A*j*) and (A*k*) to A*i*.
	031*i*00	A*i* –1 §	Transmit –1 to A*i*.
	031*i*0*k*	A*i* – A*k* §	Transmit the negative of (A*k*) to A*i*.
	031*ij*0	A*i* A*j* – 1 §	Transmit the integer difference of (A*j*) and 1 to A*i*.

Special Cases

The special cases for instruction 030 are as follows:

- If $j = 0$ and $k \neq 0$, then (A*i*) = (A*k*).
- If $j = 0$ and $k = 0$, then (A*i*) = 1.
- If $j \neq 0$ and $k = 0$, then (A*i*) = (A*j*) + 1.

The special cases for instruction 031 are as follows:

- If $j = 0$ and $k \neq 0$, then (A*i*) = –(A*k*).
- If $j = 0$ and $k = 0$, then (A*i*) = –1.
- If $j \neq 0$ and $k = 0$, then (A*i*) = (A*j*) – 1.

Hold Issue Conditions

Instructions 030 and 031 hold issue for any of the following conditions:

- The A*i* register is reserved.
- The A*j* or A*k* register is reserved (except A0).
- An A register access conflict occurs; therefore, one of the following is true:
  - Instruction 026*ij*0, 026*ij*1, or 027*ij*0 is in CP 4.
  - One of the instructions from 026*ij*4 through 026*ij*7 is in CP 6.
  - Instruction 033 is in CP 17.

Execution Time

Instructions 030 and 031 issue in 1 CP.

The A*i* register is ready in 2 CPs.

Description

Instructions 030 and 031 execute in the address add functional unit. Overflow is not detected by either instruction.

Instruction 030 forms the integer sum of the contents of the A*j* and A*k* registers and enters the result into the A*i* register.

Instruction 031 forms the integer difference of the contents of the A*j* and A*k* registers and enters the result into the A*i* register. Instruction 031*i*00 is generated in place of instruction 020 if the operand is explicitly –1.

Instruction 032			
Mode	Machine Instruction	CAL Syntax	Description
	032*ijk*	A*i* A*j* * A*k*	Transmit the integer product of (A*j*) and (A*k*) to A*i*.

Special Cases

The two special cases for instruction 032 are as follows:

- If $j = 0$, then (A*i*) = 0.
- If $j \neq 0$ and $k = 0$, then (A*i*) = (A*j*).

Hold Issue Conditions

Instruction 032 holds issue if the A*i* register is reserved or if either the A*j* or the A*k* (except A0) register is reserved.

Execution Time

Instruction 032 issues in 1 CP.

The A*i* register is ready in 4 CPs.

Description

Instruction 032 forms the integer product of the contents of the A*j* and A*k* registers and enters the 32 low-order bits of the result into the A*i* register. Instruction 032 executes in the address multiply functional unit with no overflow detected.

Instruction 033			
Mode	Machine Instruction	CAL Syntax	Description
	033*i*00	A*i* CI	Transmit to A*i* the channel number of the highest priority channel requesting an interrupt.
	033*ij*0	A*i* CA,A*j*	Transmit the current address of channel (A*j*) to A*i* ($j \neq 0$).
	033*ij*1	A*i* CE,A*j*	Transmit channel status word for channel (A*j*) to A*i* ($j \neq 0$).

Special Cases

The special cases for instruction 033 are as follows:

- If the program is not in monitor mode and IMI mode is set, this instruction causes an exchange.
- If $j = 0$, then (A*i*) = the channel number of the highest priority channel causing an interrupt.
- Valid LOSP channel numbers for instructions 033*ij*0 and 033*ij*1 are 3_8, 7_8, 13_8, 17_8, 23_8, 27_8, 33_8, and 37_8.
- If $j \neq 0$ and $k = 0$, then (A*i*) = current address of channel (A*j*).
- If $j \neq 0$ and $k = 1$, then (A*i*) = I/O error flag of channel (A*j*).
- After instruction 0012*j*0 issues, 1 CP must elapse before the correct channel number of the interrupting channel can be read by instruction 033*i*00.

Hold Issue Conditions

Instruction 033 holds issue if the A*i* or A*j* (except A0) register is reserved.

Instruction 033 holds issue for 3 CPs and continues to hold issue if a shared register access conflict occurs with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Execution Time

Instruction 033 issues in 1 CP.

The A*i* register is ready in 19 CPs.

Description

Instruction 033 enters channel status information into the A*i* register. The *j* and *k* designators and the contents of register A*j* determine the information transmitted. Instruction 033 does not interfere with normal channel operation and is not restricted to monitor mode.

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

Instruction 033*ij*0 reads from A*j* the address of a specific channel. It then transmits the contents of the CA register for that channel to the A*i* register. The A*j* register must be set to a value corresponding to the number of the desired channel before instruction 033*ij*0 is issued.

Instruction 033*ij*1 reads from A*j* the address of a specific channel. It then transmits a 32-bit channel status word from that channel to the A*i* register. Bit assignments for the status word are listed in Table 7-3. The A*j* register must be set to a value corresponding to the number of the desired channel before instruction 033*ij*1 is issued.

Table 7-3. Channel Status Word

Bit Position	Description
$2^0 - 2^{23}$	Block length (BL) register bits $2^0 - 2^{17}$.
$2^{24} - 2^{25}$	Not used (forced to 0).
2^{26}	Channel transfer in progress.
2^{27}	Block length error.
2^{28}	Uncorrectable (double-bit) error in SSD.
2^{29}	Uncorrectable (double-bit) error in mainframe.
2^{30}	Fatal error.
2^{31}	Complement of done flag.

Bit 2^{31} of the status word is the complement of the done flag. Incorporating the complement into the hardware allows software to test for the done condition by using a jump on positive instruction (012). This test works because $2^{31} = 0$ (a positive number) means the data

transmission is done, and $2^{31} = 1$ (denoting a negative number) means the data transmission is not done. Software can also test for error conditions by using a jump on nonzero instruction (011) since all bits of the status word are 0 if the channel is done and no errors have occurred. Testing for either condition can be done without manipulating the status word.

Once the internal channel error flag is set, it can only be cleared in monitor mode by instruction 0012.

Instructions 034 through 037			
Mode	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 register *jk*.
	034*ijk*	B*jk*,A*i* 0,A0 §	Read (A*i*) words from memory starting at address (A0) + (DBA) to B registers starting at register *jk*.
	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*	0,A0 B*jk*,A*i* §	Write (A*i*) words from B registers starting at register *jk* to memory starting at address (A0) + (DBA).
	036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words from memory starting at address (A0) + (DBA) to T registers starting at register *jk*.
	036*ijk*	T*jk*,A*i* 0,A0 §	Read (A*i*) words from memory starting at address (A0) + (DBA) to T registers starting at register *jk*.
	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*	0,A0 T*jk*,A*i* §	Write (A*i*) words from T registers starting at register *jk* to memory starting at address (A0) + (DBA).

Special Cases

There are three special cases, which are determined by the value stored in the A*i* register:

- If (A*i*) register = 0, no words are transferred.
- If (A*i*) register is greater than 100_8 and less than 200_8, a wrap-around condition occurs, in which some B or T registers are either read out twice or are overwritten with new memory data.
- If (A*i*) register is greater than 177_8, bits 2^7 through 2^{23} are truncated, and the block length is equal to the value stored in bits 2^0 through 2^6.

Only bits 2^0 through 2^{27} of the A0 register are used in memory address calculations. Refer to "Absolute Memory Address Calculating" in Section 2 for additional information.

Hold Issue Conditions

Instructions 034 through 037 hold issue for any of the following conditions:

- Either the A0 or the A*i* register is reserved.
- A scalar reference is in CP 1 through CP 5.
- The status register is busy.

Instruction 034 holds issue for any of the following conditions:

- Instruction 035 is in progress.
- Port A is busy.
- Port C is busy and bidirectional memory (BDM) mode is not enabled.

Instruction 035 holds issue for any of the following conditions:

- Instruction 034 is in progress.
- Port C is busy.
- Port A or port B is busy and bidirectional memory (BDM) mode is not enabled.

Instruction 036 holds issue for any of the following conditions:

- Instruction 075 was issued in the preceding CP.
- Instruction 037 is in progress.
- Port B is busy.
- Port C is busy and bidirectional memory (BDM) mode is not enabled.

Instruction 037 holds issue for any of the following conditions:

- Instruction 075 was issued in the preceding CP.
- Instruction 036 is in progress.
- Port C is busy.
- Port A or port B is busy and bidirectional memory (BDM) mode is not enabled.

Execution Time

Instructions 034 through 037 each issue in 1 CP.

The following conditions apply to instruction 034 and 036:

- If (A*i*) ≠ 0, the B or T registers are reserved for (A*i*)/2 +26 CPs.
- If (A*i*) = 0, the B or T registers are reserved for 9 CPs.
- If (A*i*) ≠ 0, port A or B is busy for (A*i*)/2 + 6 CPs.
- If (A*i*) = 0, port A or B is busy for 5 CPs.

The following conditions apply to instruction 035 and 037:

- If (Ai) ≠ 0, the B or T registers are reserved for (A*i*)/2 +8 CPs.
- If (A*i*) = 0, the B or T registers are reserved for 7 CPs.
- If (A*i*) ≠ 0, port C is busy for (A*i*)/2 + 6 CPs.
- If (A*i*) = 0, port C is busy for 5 CPs.

Description

Instructions 034 through 037 perform block transfers between central memory and the B or T registers. Instruction 034*ijk* transfers words from central memory directly into the B registers. Instruction 035*ijk* stores words from B registers directly into central memory. Instruction 036*ijk* transfers words from central memory directly into T registers. Instruction 037*ijk* stores words from T registers directly into central memory.

Instructions 034 through 037 process the B and T registers in circular fashion. The first register involved in the transfer is specified by the *jk* fields. The 7 low-order bits of the A*i* register specify the number of words transmitted. Successive word transfers involve successive B or T registers until B77 or T77 is reached. Register B00 is processed after B77 and register T00 is processed after T77 if the count in the contents of the A*i* register is not exhausted.

The first memory location referenced by the transfer instruction is specified by the contents of register A0. The contents of register A0 are not altered by execution of the instruction. The memory address referenced is incremented by 1 after each word is transferred.

For transfers of B register data to central memory, each 32-bit value is right justified in the memory location, and the 32 high-order bits are cleared. In a transfer from memory to the B registers, only the 32 low-order bits are transmitted; the high-order bits are ignored.

If the contents of the A*i* register equal 0, no words are transferred. If *i* = 0, the contents of register A0 are used for both the block length and the starting memory address. The CAL assembler issues a warning message when *i* = 0.

NOTE: Instruction 034 uses port A, instruction 036 uses port B, and instructions 035 and 037 use port C for block transfers.

Instructions 040 and 041			
Mode	Machine Instruction	CAL Syntax	Description
	040*i*00 *nm*	S*i* *exp*	Transmit *nm* to S*i*, bits $2^0 - 2^{31}$ (bits $2^{32} - 2^{63}$ are set to 0).
	040*i*20 *nm*	S*i* S*i*:*exp*	Transmit *nm* to S*i*, bits $2^0 - 2^{31}$ (bits $2^{32} - 2^{63}$ unchanged).
	040*i*40 *nm*	S*i* *exp*:S*i*	Transmit *nm* to S*i*, bits $2^{32} - 2^{63}$ (bits $2^0 - 2^{31}$ unchanged).
	041*i*00 *nm*	S*i* *exp*	Transmit one's complement of *nm* to S*i*.

Special Cases

There are no special cases.

Hold Issue Conditions

Instructions 040 and 041 hold issue for any of the following conditions:

- The S*i* register is reserved.
- An S register access conflict exists; therefore, one of the following is true:
  - Instruction 071 is in CP 1.
  - One of the instructions from 052 through 055 or instruction 060 or 061 is in CP 2.
  - Instruction 056 or 057 is in CP 3.
- The second and third parcels of the instruction are in another buffer (a 3-CP delay occurs).
- The third parcel of the instruction is in another buffer (a 4-CP delay occurs).
- The second and/or third parcel of the instruction is not in a buffer (a 26-CP delay occurs).

Execution Time

The issue times for instructions 040 and 041 are as follows:

- If both parcels are in the same buffer, the issue time is 2 CPs.
- If parcel 0 is in a different buffer than parcels 1 and 2, the issue time is 5 CPs.
- If parcels 0 and 1 are in a different buffer than parcel 2, the issue time is 6 CPs.

The S*i* register is ready in 1 CP.

Description

These 3-parcel instructions transmit a value to the S*i* register. The assembler generates either an 040*i*00 *nm* or an 041*i*00 *nm* instruction, depending on the value and attributes of the *exp* (*nm*) field.

If the expression has a positive value or either a relocatable or external relative attribute, the assembler generates instruction 040*i*00 *nm*.

If the expression has a negative value and an absolute relative attribute, the assembler generates instruction 041*i*00 *nm*.

Instruction 040*i*20 *nm* transmits the *exp* contained in the combined *nm* fields of the instruction to bit positions 2^0 through 2^{31} of the S*i* register. Bits 2^{32} through 2^{63} of the S*i* register are unchanged by this instruction.

Instruction 040*i*40 *nm* transmits the *exp* contained in the combined *nm* fields of the instruction to bit positions 2^{32} through 2^{63} of the S*i* register. Bits 2^0 through 2^{31} of the S*i* register are unchanged by this instruction.

Instructions 042 and 043			
Mode	Machine Instruction	CAL Syntax	Description
	042*ijk*	S*i* < *exp*	Form ones mask in S*i* *exp* bits from the right; the *jk* field contains the value $100_8 - exp$.
	042*ijk*	S*i* #> *exp* §	Form zeroes mask in S*i* *exp* bits from the left; the *jk* field contains the value *exp*.
	042*i*77	S*i* 1 §	Transmit 1 to the S*i* register.
	042*i*00	S*i* –1 §	Transmit –1 to the S*i* register.
	043*ijk*	S*i* > *exp*	Form ones mask in S*i* *exp* bits from the left; the *jk* field contains the value *exp*.
	043*ijk*	S*i* #< *exp* §	Form zeroes mask in S*i* *exp* bits from the right; the *jk* field contains the value $100_8 - exp$.
	043*i*00	S*i* 0 §	Clear the S*i* register.

Special Cases

There are no special cases.

Hold Issue Conditions

Instructions 042 and 043 hold issue for any of the following conditions:

- The Si register is reserved.
- An S register access conflict exists; therefore, one of the following is true:
  - Instruction 071 is in CP 1.
  - One of the instructions from 052 through 055 or instruction 060 or 061 is in CP 2.
  - Instruction 056 or 057 is in CP 3.

Execution Time

Instructions 042 and 043 issue in 1 CP.

The S*i* register is ready in 1 CP.

Description

Instructions 042 and 043 execute in the scalar logical functional unit.

Instruction 042 generates a mask of $100_8 - jk$ 1's from right to left in the S*i* register. For example, if $jk = 0$, the S*i* register contains all 1 bits (integer value = –1), and if $jk = 77_8$, the S*i* register contains 0's in all but the low-order bit (integer value = 1).

Instruction 043 generates a mask of *jk* 1's from left to right in the S*i* register. For example, if $jk = 0$, the S*i* register contains all 0 bits (integer value = 0) and if $jk = 77_8$, the S*i* register contains 1's in all bits except the low-order bit (integer value = –2).

Instructions 044 through 051			
Mode	Machine Instruction	CAL Syntax	Description
	044*ijk*	Si Sj&Sk	Transmit the logical product of (S*j*) and (S*k*) to S*i*.
	044*ij*0	Si Sj&SB §	Transmit the sign bit (bit 2^{63}) of (S*j*) to S*i*.
	044*ij*0	Si SB&Sj §	Transmit the sign bit (bit 2^{63}) of (S*j*) to S*i* ($j \neq 0$).
	045*ijk*	Si #Sk&Sj	Transmit the logical product of (S*j*) and the complement of (S*k*) to S*i*.
	045*ij*0	Si #SB&Sj §	Transmit (S*j*) with sign bit cleared to S*i*.
	046*ijk*	Si Sj\Sk	Transmit the exclusive OR of (S*j*) and (S*k*) to S*i*.
	046*ij*0	Si Sj\SB §	Toggle the sign bit of (S*j*), and transmit the result to S*i*.
	046*ij*0	Si SB\Sj §	Toggle the sign bit of (S*j*), and transmit the result to S*i* ($j \neq 0$).
	047*ijk*	Si #Sj\Sk	Transmit the logical equivalence of (S*j*) and (S*k*) to S*i*.
	047*i*0*k*	Si #Sk §	Transmit the one's complement of (S*k*) to S*i*.
	047*ij*0	Si #Sj\SB §	Transmit the logical equivalence of (S*j*) and the sign bit to S*i*.
	047*ij*0	Si #SB\Sj §	Transmit the logical equivalence of (S*j*) and the sign bit to S*i* ($j \neq 0$).
	047*i*00	Si #SB §	Transmit the one's complement of the sign bit to S*i*.
	050*ijk*	Si Sj!Si&Sk	Transmit the logical product of (S*i*) and (S*k*) complement ORed with the logical product of (S*j*) and (S*k*) to S*i*; merge (S*i*) and (S*j*) into S*i* using (S*k*) as the mask.
	050ij0	Si Sj!Si&SB §	Transmit the scalar merge of (S*i*) and the sign bit of (S*j*) to S*i*.
	051*ijk*	Si Sj!Sk	Transmit the logical sum of (S*j*) and (S*k*) to S*i*.
	051*i*0*k*	Si Sk §	Transmit (S*k*) to S*i*.
	051*ij*0	Si Sj!SB §	Transmit the logical sum of (S*j*) and the sign bit to S*i*.
	051*ij*0	Si SB!Sj §	Transmit the logical sum of (S*j*) and the sign bit to S*i* ($j \neq 0$).
	051*i*00	Si SB §	Transmit the sign bit to S*i*.

Special Cases

The following special cases apply to instructions 044 through 051:

- If $j = 0$, then (Sj) = 0.
- If $k = 0$, then (Sk) has all bits cleared to 0 except bit 2^{63}.

Hold Issue Conditions

Instructions 044 through 051 hold issue for any of the following conditions:

- The Si register is reserved.
- The Sj or Sk register is reserved (except S0).

Execution Time

Instructions 044 through 051 issue in 1 CP.

The Si register is ready in 1 CP.

Description

Instructions 044 through 051 execute in the scalar logical functional unit.

Instruction 044 forms the logical product (AND) of the contents of the Sj and Sk registers and enters the results into the Si register. Bits of the Si register are set to 1 when corresponding bits of the values stored in the Sj and Sk registers are 1, as shown in the following example:

```
If    (Sj) = 1 1 0 0
And   (Sk) = 1 0 1 0
Then  (Si) = 1 0 0 0
```

The contents of the Sj register are transmitted to the Si register if the j and k designators have the same nonzero value. The Si register is cleared if the j designator is 0. The sign bit of the contents of the Sj register is transmitted to the Si register if the j designator is nonzero and the k designator is 0. The two special CAL forms of instruction 044ij0 perform the same function. In the second CAL form, however, j must not equal 0, which causes an assembly error.

Instruction 045 forms the logical product (AND) of the Sj register contents and the complement of the Sk register contents and enters the results into the Si register. Bits of the Si register are set to 1 when

corresponding bits of the values stored in the S*j* register and the complement of the S*k* register are 1, as shown in the following example, where (S*k*)' = complement of the S*k* register contents:

```
If    (Sk) = 1 0 1 0,
And   (Sj) = 1 1 0 0
      (Sk)' = 0 1 0 1
Then  (Si) = 0 1 0 0
```

S*i* is cleared if the *j* and *k* designators have the same value or if the *j* designator is 0. The contents of the S*j* register with the sign bit cleared are transmitted to the S*i* register if the *j* designator is nonzero and if the *k* designator is 0. The special CAL form of instruction 045*ij*0 also performs this function.

Instruction 046 forms the exclusive OR of the contents of the S*j* and S*k* registers and enters the results into the S*i* register. Bits of the S*i* register are set to 1 when corresponding bits of the S*j* register and the S*k* register are different, as shown in the following example:

```
If    (Sj) = 1 1 0 0
And   (Sk) = 1 0 1 0
Then  (Si) = 0 1 1 0
```

S*i* is cleared if the *j* and *k* designators have the same nonzero value. The S*k* register contents are transmitted to the S*i* register if the *j* designator is 0 and the *k* designator is nonzero. The sign bit of the S*j* register contents is complemented, and the result is transmitted to the S*i* register if the *j* designator is nonzero and the *k* designator is 0. The two special CAL forms of instruction 046*ij*0 also perform the same function. In the second CAL form, however, *j* must not equal 0, which causes an assembly error.

NOTE: Although the \ symbol is used in mathematical logic to denote the logical difference, it is used in CAL to denote the exclusive OR (XOR) operation.

Instruction 047 forms the logical equivalence of the contents of the S*j* and S*k* registers and enters the results into the S*i* register. Bits of the S*i* register are set to 1 when corresponding bits of the S*j* register and the S*k* register are the same, as shown in the following example:

```
If    (Sj) = 1 1 0 0
And   (Sk) = 1 0 1 0
Then  (Si) = 1 0 0 1
```

S*i* is set to all 1's if the *j* and *k* designators have the same nonzero value. The one's complement of the S*k* register contents is transmitted to the S*i* register if the *j* designator is 0 and the *k* designator is nonzero (the same as instruction 047*i*0*k*).

The special CAL form of instruction 047*i*0*k* performs this same function. All bits except the sign bit of the S*j* register contents are complemented and the result is transmitted to the S*i* register if the *j* designator is nonzero and the *k* designator is 0. The result is the same as the one's complement of the result generated by instruction 046*ij*0. The two special CAL forms of instruction 047*ij*0 also perform the same function. In the second CAL form, however, *j* must not equal 0, which causes an assembly error.

Instruction 050 merges the contents of the S*i* and S*j* registers according to the bit locations of the ones mask stored in S*k*; the result is entered into the S*i* register. The bits of the resultant S*i* register equal the bits of the initial S*i* register when the corresponding bits of the S*k* mask are 0, and the bits of the resultant S*i* register equal the bits of the S*j* register when the corresponding bits of the S*k* mask are 1. The operation, defined by the Boolean equation (S*i*) = (S*j*)(S*k*) + (S*i*) (S*k*)', is shown in the following example:

If (S*k*) = 1 1 1 1 0 0 0 0,

(S*k*)' = 0 0 0 0 1 1 1 1
And (S*i*) = 1 1 0 0 1 1 0 0
And (S*j*) = 1 0 1 0 1 0 1 0
Then (Si) = 1 0 1 0 1 1 0 0

If the *j* designator is 0 and the *k* designator is nonzero, bits of the S*i* register are cleared when the corresponding bits of the S*k* register are 1. If the *j* designator is nonzero and the *k* designator is 0, the sign bit of the S*j* register contents replaces the sign bit of the S*i* register. The special CAL form of instruction 050*ij*0 performs this same function. The sign bit of the S*i* register is cleared if the *j* and *k* designators are both 0.

Instruction 051 forms the logical sum (inclusive OR) of the contents of the S*j* and S*k* registers and enters the result into the S*i* register. Bits of the S*i* register are set when one of the corresponding bits of either the S*j* or S*k* register or both is set, as in the following example:

If (S*j*) = 1 1 0 0
And (S*k*) = 1 0 1 0
Then (S*i*) = 1 1 1 0

The contents of the S*j* register are transmitted to the S*i* register if the *j* and *k* designators have the same nonzero value. The S*k* register contents are transmitted to the S*i* register if the *j* designator is 0 and the *k* designator is nonzero. The S*j* register contents, with the sign bit set to 1, are transmitted to the S*i* register if the *j* designator is nonzero and the *k* designator is 0. The two special CAL forms of instruction 051*ij*0 also perform the same function. In the second CAL form, however, *j* must not equal 0, which causes an assembly error.

A ones mask consisting of only the sign bit is entered into the S*i* register if the *j* and *k* designators are both 0. The special CAL form of instruction 051*i*00 performs this same function.

NOTE: For instructions 044 through 051, the abbreviation SB designates the sign bit, not a shared address register.

Instructions 052 through 055			
Mode	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* = $100_8 - jk$.
	054*ijk*	S*i* S*i* < *exp*	Shift (S*i*) left *exp* places to S*i*; *exp* = *jk*.
	055*ijk*	S*i* S*i* > *exp*	Shift (S*i*) right *exp* places to S*i*; *exp* = $100_8 - jk$.

Special Cases

There are no special cases.

Hold Issue Conditions

Instructions 052 through 055 hold issue if the S*i* register is reserved.

Instructions 052 and 053 also hold issue if the S0 register is reserved.

Execution Time

Instructions 052 through 055 issue in 1 CP.

For instructions 052 and 053, the S0 register is ready in 3 CPs.

For instructions 054 and 055, the S*i* register is ready in 3 CPs.

Description

Instructions 052 through 055 execute in the scalar shift functional unit. The instructions cause values in an S register to be shifted by an amount specified by *exp* (*jk* field). All shifts are end-off with zero fill, meaning that data that is shifted out of a register, either to the right or the left, is lost, and that the trailing edge of the data is replaced in the register with zeroes.

Instruction 052 shifts the S*i* register contents left *jk* places and enters the result into the S0 register; the shift range is 0 through 63, left. If the shift count is 64, instruction 053000 is generated and the S0 register is cleared.

Instruction 053 shifts the S*i* register contents right $100_8 - jk$ places and enters the result into the S0 register; the shift range is 1 through 100_8, right. If the shift count is 0, instruction 052000 is generated and the contents of the S0 register are not altered.

Instruction 054 shifts the S*i* register contents left *jk* places and enters the result into the S*i* register; the shift range is 0 through 77_8, left. If the shift count is 100_8, instruction 055*i*00 is generated and the S*i* register is cleared.

Instruction 055 shifts the S*i* register contents right $100_8 - jk$ places and enters the result into the S*i* register; the shift range is 1 through 100_8, right. If the shift count is 0, instruction 054*i*00 is generated and the contents of the S*i* register are not altered.

Instructions 056 and 057			
Mode	Machine Instruction	CAL Syntax	Description
	056*ijk*	Si Si,Sj < Ak	Shift (S*i*) and (S*j*) left (A*k*) places to S*i*.
	056*ij*0	Si Si,Sj < 1 §	Shift (S*i*) and (S*j*) left one place to S*i*.
	056*i*0*k*	Si Si < Ak §	Shift (S*i*) left (A*k*) places to S*i*.
	057*ijk*	Si Sj,Si > Ak	Shift (S*j*) and (S*i*) right (A*k*) places to S*i*.
	057*ij*0	Si Sj,Si > 1 §	Shift (S*j*) and (S*i*) right one place to S*i*.
	057*i*0*k*	Si Si > Ak §	Shift (S*i*) right (A*k*) places to S*i*.

Special Cases

Special cases for instructions 056 and 057 are as follows:

- If $j = 0$, then (Sj) = 0.
- If $k = 0$, then (Ak) = 1.
- Perform a circular shift if $i = j \neq 0$ and (Ak) is between 0 and 64 inclusive.
- Hold issue for 1 CP after instruction 056 or 057 is issued.

Hold Issue Conditions

Instructions 056 and 057 hold issue if the Si register is reserved or if either the Sj or the Ak register is reserved (except S0 and/or A0).

Execution Time

Instructions 056 and 057 issue in 1 CP.

The Si register is ready in 4 CPs.

Description

Instructions 056 and 057 execute in the scalar shift functional unit. The instructions shift 128-bit values formed by logically joining two S registers. Shift counts are obtained from the A*k* register; all shift counts are considered positive, and all 32 bits of the A*k* register contents are used for the shift count.

In CAL, replacing the A*k* register reference with 1 is the same as setting the *k* designator to 0 (instruction 056*ij*0 or 057*ij*0) because a reference to register A0 provides a shift count of 1. Omitting the S*j* register reference in CAL is the same as setting the *j* designator to 0 (instruction 056*i*0*k* or 057*i*0*k*). In this case, the S*i* register contents are concatenated to a word of 0's.

The shifts are circular if the shift count does not exceed 64 and if the *i* and *j* designators are equal and nonzero. For shifts greater than 64, the shift is end-off with zero fill, meaning that data that is shifted out of the combined registers, either to the right or the left, is lost, and that the trailing edge of the data is replaced in the registers with zeroes.

The S*j* register contents are unchanged if $i \neq j$. If $i = j$ and the shift count is greater than 64, the result produced in the S*i* register is the same as would be obtained by issuing instruction 054 or 055 with a shift count 64 less than that used for instruction 056 or 057.

Instruction 056 forms a 128-bit value by concatenating the S*i* and S*j* register contents, by shifting the resulting value to the left by an amount specified by the 7 low-order bits of the A*k* register contents, and by entering the 64 high-order bits of the result into the S*i* register. The S*i* register is cleared if the shift count exceeds 127. Instruction 056 produces the same result as instruction 054 if the shift count does not exceed 63 and the *j* designator is 0. This same function is performed by the special CAL form of the instruction, 056*i*0*k*.

Instruction 057 forms a 128-bit value by concatenating the S*j* and S*i* register contents, by shifting the resulting value to the right by an amount specified by the 7 low-order bits of the A*k* register contents, and by entering the 64 low-order bits of the result into the S*i* register. The S*i* register is cleared if the shift count exceeds 127. Instruction 057 produces the same result as instruction 055 if the shift count does not exceed 63 and the *j* designator is 0. This same function is performed by the special CAL form of the instruction, 057*i*0*k*.

Instructions 060 and 061			
Mode	Machine Instruction	CAL Syntax	Description
	060*ijk*	S*i* S*j* + S*k*	Transmit the integer sum of (S*j*) and (S*k*) to S*i*.
	060*i*0*k*	S*i* S*k* §	Transmit (S*k*) to S*i*.
	060*ij*0	S*i* S*j* + S0 §	Transmit the integer sum of (S*j*) and 2^{63} to S*i*.
	061*ijk*	S*i* S*j* – S*k*	Transmit the integer difference of (S*j*) and (S*k*) to S*i*.
	061*i*0*k*	S*i* –S*k* §	Transmit the negative of (S*k*) to S*i*.
	061*ij*0	S*i* S*j* – S0 §	Transmit the integer difference of (S*j*) and 2^{63} to S*i*.

Special Cases

Special cases for instructions 060 and 061 are as follows:

- If $j = 0$ and $k = 0$, then (S*i*) = 2^{63}.

Special cases for instruction 060 are as follows:

- If $j = 0$ and $k \neq 0$, then (S*i*) = (S*k*).
- If $j \neq 0$ and $k = 0$, then (S*i*) = (S*j*) with bit 2^{63} complemented.

Special cases for instruction 061 are as follows:

- If $j = 0$ and $k \neq 0$, then (S*i*) = –(S*k*).
- If $j \neq 0$ and $k = 0$, then (S*i*) = (S*j*) with bit 2^{63} complemented.

Hold Issue Conditions

Instructions 060 and 061 hold issue if the S*i* register is reserved or if either the S*j* or S*k* register is reserved (except S0).

Execution Time

Instructions 060 and 061 issue in 1 CP.

The S*i* register is ready in 3 CPs.

Description

Instructions 060 and 061 execute in the scalar add functional unit.

Instruction 060*ijk* forms the integer sum of the S*j* and S*k* register contents and enters the result into the S*i* register; no overflow is detected. The high-order bit of the S*i* register is set, and all other bits of the S*i* register are cleared if the *j* and *k* designators are both 0. The S*k* register contents are transmitted to the S*i* register if the *j* designator is 0 and the *k* designator is nonzero. The S*j* register contents with the sign bit complemented are transmitted to the S*i* register if the *k* designator is 0 and the *j* designator is nonzero.

Instruction 061*ijk* forms the integer difference of the contents of the S*j* and S*k* registers and enters the result into the S*i* register; no overflow is detected. The high-order bit of the S*i* register is set, and all other bits of the S*i* register are cleared when the *j* and *k* designators are both 0. The negative (two's complement) of the S*k* register contents is transmitted to the S*i* register if the *j* designator is 0 and the *k* designator is nonzero. The S*j* register contents with the sign bit complemented are transmitted to the S*i* register if the *k* designator is 0 and the *j* designator is nonzero.

Instruction 061*i*0*k* is a special CAL form of instruction 061 that transmits the negative (two's complement) of the S*k* register contents to the S*i* register. If the *k* designator is also 0, the high-order bit of the S*i* register is set, and all other bits of the S*i* register are cleared.

Instructions 062 and 063			
Mode	Machine Instruction	CAL Syntax	Description
	062*ijk*	S*i* S*j* + FS*k*	Transmit the floating-point sum of (S*j*) and (S*k*) to S*i*.
	062*i*0*k*	S*i* + FS*k* §	Transmit the normalized (S*k*) to S*i*.
	063*ijk*	S*i* S*j* – FS*k*	Transmit the floating-point difference of (S*j*) and (S*k*) to S*i*.
	063*i*0*k*	S*i* – FS*k* §	Transmit the normalized negative of (S*k*) to S*i*.

Special Cases

Special cases for instruction 062 are as follows:

- If the exponent in (S*k*) is valid and if $j = 0$ and $k \neq 0$, then (S*i*) = (S*k*) normalized.
- If the exponent in (S*j*) is valid and if $j \neq 0$ and $k = 0$, then (S*i*) = (S*j*) normalized.

Special cases for instruction 063 are as follows:

- If the exponent in (S*k*) is valid and if $j = 0$ and $k \neq 0$, then (S*i*) = –(S*k*) normalized. The sign of (S*i*) is the opposite of the sign of (S*k*) if (S*k*) $\neq 0$.
- If the exponent in (S*j*) is valid and if $j \neq 0$ and $k = 0$, then (S*i*) = (S*j*) normalized.

Hold Issue Conditions

Instructions 062 and 063 hold issue for any of the following conditions:

- The S*i* register is reserved.
- The S*j* or S*k* register is reserved (except S0).
- The status register is busy.
- Instructions 170 through 173 are in progress. In this case, the functional unit stays busy for (VL)/2 + 4 CPs.

Execution Time

Instructions 062 and 063 issue in 1 CP.

The S*i* register is ready in 6 CPs.

Description

Instructions 062 and 063 execute in the floating-point add functional unit. Operands sent to the functional unit should be in floating-point format; the result is normalized even if the operands are unnormalized. If the *k* designator is 0, the operand sent to the functional unit is 2^{63}. In floating-point format, this operand is a negative number with a coefficient of 0 and an exponent of 0, and it is regarded by the functional unit as –0. However, this anomaly is resolved within the circuitry of the floating-point add functional unit, and any 0 result is entered into the S*i* register as 0.

Instruction 062*ijk* forms the floating-point sum of the S*j* and S*k* register contents and enters the normalized result into the S*i* register. The S*k* register contents are normalized and transmitted to the S*i* register if the *j* designator is 0, if the *k* designator is nonzero, and if the exponent in (S*k*) is valid. The S*j* register contents are normalized and transmitted to the S*i* register if the *j* designator is nonzero, if the *k* designator is 0, and if the exponent of the S*j* register contents is valid.

Instruction 062*i*0*k* is a special CAL form of instruction 062 that transmits the normalized S*k* register contents to the S*i* register.

Instruction 063*ijk* forms the floating-point difference of the S*j* and S*k* register contents and enters the normalized result into the S*i* register. The S*k* register contents are normalized, and the negative (two's complement) of this quantity is transmitted to the S*i* register if the *j* designator is 0, if the *k* designator is nonzero, and if the exponent of the S*k* register contents is valid. The S*j* register contents are normalized and transmitted to the S*i* register if the *j* designator is nonzero, if the *k* designator is 0, and if the exponent of the S*j* register contents is valid.

Instruction 063*i*0*k* is a special CAL form of instruction 063 that transmits the negative (two's complement) of the floating-point quantity in the S*k* register to the S*i* register as a normalized floating-point number.

Instructions 064 through 067			
Mode	Machine Instruction	CAL Syntax	Description
	064*ijk*	S*i* S*j* * FS*k*	Transmit the floating-point product of (S*j*) and (S*k*) to S*i*.
	065*ijk*	S*i* S*j* * HS*k*	Transmit the half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*.
	066*ijk*	S*i* S*j* * RS*k*	Transmit the rounded floating-point product of (S*j*) and (S*k*) to S*i*.
	067*ijk*	S*i* S*j* * IS*k*	Transmit the reciprocal iteration 2 – (S*j*) * (S*k*) to S*i*.

Special Cases

If $j = 0$, then (Sj) = 0.

If $k = 0$, then (Sk) = 2^{63}.

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

- Both operand sign bits are 0.
- The combined total number of 0 bits to the right of the least significant 1 bit in the two operands is greater than or equal to 48. For example, if the *j* operand has 36 trailing 0's and the *k* operand has 12 trailing 0's, then a correct integer multiplication is performed.

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

Hold Issue Conditions

Instructions 064 through 067 hold issue for any of the following conditions:

- The S*i* register is reserved.
- Either the S*j* or the S*k* register is reserved (except S0).
- The status register is busy.

- Instructions 160 through 167 are in progress. In this case, the functional unit is busy for (VL)/2 + 4 CPs.
- Instructions 140 through 145 are in progress. In this case, the second vector logical functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instructions 064 through 067 issue in 1 CP.

The S*i* register is ready in 6 CPs.

Description

Instructions 064 through 067 execute in the floating-point multiply functional unit. Operands sent to the functional unit should be in floating-point format; the result may not be normalized if the operands are not normalized.

Instruction 064*ijk* forms the floating-point product of the S*j* and S*k* register contents and enters the result into the S*i* register.

Instruction 065*ijk* forms the half-precision rounded floating-point product of the S*j* and S*k* register contents and enters the result into the S*i* register. The 19 low-order bits of the result are cleared. This instruction can be used in the division algorithm when only 30 bits of accuracy are required.

Instruction 066*ijk* forms the rounded floating-point product of the S*j* and S*k* register contents and enters the result into the S*i* register. This instruction is used in the reciprocal approximation sequence.

Instruction 067*ijk* forms the quantity of 2 minus the floating-point product of the S*j* and S*k* register contents and enters the result into the S*i* register.

Instruction 070			
Mode	Machine Instruction	CAL Syntax	Description
	070*ij*0	S*i* /HS*j*	Transmit the floating-point reciprocal approximation of (S*j*) to S*i*.

Special Cases

If $j = 0$, then (Sj) = 0.

If the Sj register contents are 0, a range error occurs and the result is invalid.

The Si register contents are invalid if the Sj register contents are not normalized. A normalized value is indicated when bit 2^{47} of the Sj register contents equals 1. This bit is not tested to determine its value.

Hold Issue Conditions

Instruction 070 holds issue for any of the following conditions:

- The Si register is reserved.
- The Sj register is reserved (except S0).
- The status register is busy.
- Instruction 174 is in progress, making the functional unit busy for (VL)/2 + 4 CPs.

Execution Time

Instruction 070 issues in 1 CP.

The Si register is ready in 10 CPs.

Description

Instruction 070 executes in the reciprocal approximation functional unit. Instruction 070 forms an approximation to the reciprocal of the normalized floating-point quantity in the S*j* register and enters the result into the S*i* register. The result is invalid if the contents of the S*j* register are not normalized or are equal to 0.

The reciprocal approximation instruction produces a result of 30 significant bits; the 18 low-order bits are 0's. The number of significant bits can be increased to 48 by using the reciprocal iteration instruction and a multiplication operation.

Instruction 071			
Mode	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 an unnormalized floating-point number.
	071*i*30	S*i* 0.6	Transmit 0.75 x 2^{48} to S*i* as a normalized floating-point constant.
	071*i*40	S*i* 0.4	Transmit 0.4 to S*i* as a normalized floating-point constant.
	071*i*50	S*i* 1.0	Transmit 1.0 to S*i* as a normalized floating-point constant.
	071*i*60	S*i* 2.0	Transmit 2.0 to S*i* as a normalized floating-point constant.
	071*i*70	S*i* 4.0	Transmit 4.0 to S*i* as a normalized floating-point constant.

Special Cases

Special cases for instruction 071 are as follows:

- If $k = 0$, then (A*k*) = 1.
- If $j = 0$, then (S*i*) = (A*k*).
- If $j = 1$, then (S*i*) = (A*k*) with the sign extended.
- If $j = 2$, then (S*i*) = (A*k*) unnormalized.
- If $j = 3$, then (S*i*) = 0.6 x 2^{60} (octal).
- If $j = 4$, then (S*i*) = 0.4 x 2^{0} (octal).
- If $j = 5$, then (S*i*) = 0.4 x 2^{1} (octal).
- If $j = 6$, then (S*i*) = 0.4 x 2^{2} (octal).
- If $j = 7$, then (S*i*) = 0.4 x 2^{3} (octal).

Hold Issue Conditions

Instruction 071 holds issue for any of the following conditions:

- The S*i* register is reserved.

- An S register access conflict exists; therefore, one of the following is true:
  - One of the instructions from 053 through 055 or instruction 060 or 061 is in CP 1.
  - Instruction 056 or 057 is in CP 2.

Instructions 071*i*0*k* through 071*i*2*k* hold issue if the A*k* register is reserved (except A0).

Execution Time

Instruction 071 issues in 1 CP.

The S*i* register is ready in 2 CPs.

Description

Instruction 071 transmits either a variation of the A*k* register contents or one of five floating-point constants to the S*i* register, depending on the value of the *j* designator.

Instruction 071*i*0*k* transmits the 32-bit value in the A*k* register to the low-order bits of the S*i* register; the high-order bits of the S*i* register are filled with zeroes. The value is treated as an unsigned integer. A value of 1 is entered into the S*i* register when the *k* designator is 0.

Instruction 071*i*1*k* transmits the 32-bit value in the A*k* register to the low-order bits of the S*i* register. The sign bit of the A*k* register is extended through the high-order bits of the S*i* register. The value is treated as a signed integer. A value of 1 is entered into the S*i* register when the *k* designator is 0.

Instruction 071*i*2*k* transmits the 32-bit value in the A*k* register to S*i* as an unnormalized floating-point quantity. The exponent in bits 2^{62} through 2^{48} of S*i* is set to 40060_8, and the A*k* register contents are entered as the coefficient of S*i* in bits 2^{47} through 2^0. If the sign bit of the A*k* register contents is set, the two's complement of the A*k* register contents is entered into the S*i* register as the coefficient. The sign of the coefficient (bit 2^{63}) is the same as the sign of the A*k* register contents. To normalize the quantity in S*i*, use instruction 062*i*0*k* to send the quantity as the *k* operand through the floating-point add functional unit.

A sequence of instructions is used to convert an integer whose absolute value is less than 32 bits to floating-point format.

The following CAL code is an example of this instruction sequence:

```
CAL code:  A1  S1
           S1  +FA1
           S1  +FS1     (11 CPs required)
```

Instructions 071*i*30 through 071*i*70 are initially recognized by the assembler as the symbolic instruction S*i exp*. The assembler then checks the expression for any of the constant values (explained in following paragraphs). If it finds one of the instructions in the exact syntax shown, it generates the corresponding Cray Research machine instruction. If none of the indicated constant values are found, instruction 040*i*00 *nm* or 041*i*00 *nm* is generated. These constant values generate more efficient instructions when commonly used values are entered into S*i*.

Instruction 071*i*30 transmits the floating-point constant of 0.75×2^{48} decimal or 0.6×2^{60} octal (0400606000000000000000_8) to S*i*. This constant is used to convert integers with an absolute value of less than 47 bits to floating-point numbers. A sequence of instructions is used for the conversion. The following CAL code is an example of this instruction sequence; it shows the conversion of an integer stored in S1 to a floating-point number.

```
CAL code:  S2  0.6
           S1  S2–S1
           S1  S2–FS1   (11 CPs required)
```

Instruction 071*i*40 transmits the floating-point constant 0.4 (0400040000000000000000_8) to the S*i* register.

Instruction 071*i*50 transmits the floating-point constant 1.0 (0400014000000000000000_8) to the S*i* register.

Instruction 071*i*60 transmits the floating-point constant 2.0 (0400024000000000000000_8) to the S*i* register.

Instruction 071*i*70 transmits the floating-point constant 4.0 (0400034000000000000000_8) to the S*i* register.

Instruction 072			
Mode	Machine Instruction	CAL Syntax	Description
	072*i*00	S*i* RT	Transmit (RTC) to S*i*.
	072*i*02	S*i* SM	Transmit (SM) to S*i*.
	072*ij*3	S*i* ST*j*	Transmit (ST*j*) to S*i*.
	072*ij*6	S*i* ST,A*j*	Transmit (ST) designated by (A*j*) to S*i*.

Special Cases

For instruction 072*i*00, the RTC register is valid 8 CPs after instruction 0014*j*0 issues.

For instructions 072*i*02 through 072*ij*6, if CLN = 0, then (S*i*) = 0.

Hold Issue Conditions

Instruction 072 holds issue if the S*i* register is reserved.

Instruction 072*i*00 holds issue if an S register access conflict exists; therefore, one of the following is true:

- Instruction 076 is in CP 4.
- An instruction from 026*ij*4 through 026*ij*7 or an instruction from 072*i*02 through 072*ij*6 is in CP 7.
- Instruction 033 is in CP 18.
- Instruction 073*i*02 or instruction 0014*j*3 is in CP 7 of any CPU in the same cluster.
- LOAD EX CLN is in CP 7 of any CPU in the same cluster.

Instructions 072*i*02 through 072*ij*6 hold issue for 3 CPs and continue to hold if a shared paths access conflict occurs with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Instruction 072*ij*6 holds issue if the A*j* register is reserved.

Execution Time

Instruction 072 issues in 1 CP.

For instruction 072*i*00, the S*i* register is ready in 1 CP.

For instructions 072*i*02 through 072*ij*6, the S*i* register is ready in 8 CPs.

Description

Instruction 072*i*00 transmits the 64-bit value of the real-time clock (RTC) to the S*i* register. The RTC increments by 1 each CP and can be set only by the monitor with instruction 0014*j*0.

Instruction 072*i*02 transmits the values of all the semaphore registers into the S*i* register. The thirty-two 1-bit SM registers are left-justified in the S*i* register with SM00 occupying the sign bit.

Instruction 072*ij*3 transmits the ST*j* register contents to the S*i* register.

Instruction 072*ij*6 transmits the ST register contents designated by the A*j* register contents to S*i*.

Instruction 073			
Mode	Machine Instruction	CAL Syntax	Description
	073*i*00	S*i* VM	Transmit (VM) to S*i*.
	073*i*02	SM S*i*	Transmit (S*i*) to SM.
A	073*i*10	S*i* VM1	Transmit (VM1) to S*i*.
C	073*i*21	S*i* SR2	Read PM counters 00 – 17 and increment pointer.
C	073*i*25	SR2 S*i*	Issue PM maintenance advance.
C	073*i*31	S*i* SR3	Read PM counters 20 – 37 and increment pointer.
C	073*i*75	SR7 S*i*	Transmit (S*i*) to Maintenance Mode register.
	073*ij*1	S*i* SR*j*	Transmit (SR*j*) to S*i*.
	073*ij*3	ST*j* S*i*	Transmit (S*i*) to ST*j*.
	073*i*05	SR0 S*i*	Transmit (S*i*) to SR0.
	073*ij*6	ST,A*j* S*i*	Transmit (S*i*) to ST designated by (A*j*).

Special Cases

Instructions 073*i*02, 073*ij*3, and 073*ij*6 perform no operation if CLN = 0.

Instructions 073*i*21 and 073*i*31 should not be issued while the performance monitor is busy. Bit 2^{47} of status register 0 is set when the performance monitor is busy.

Hold Issue Conditions

Instruction 073 holds issue if the S*i* register is reserved.

For instructions 073*i*00 and 073*i*10, the following apply:

- The instruction holds issue if an S register access conflict exists; therefore, one of the following is true:
  - Instruction 071 is in CP 1.
  - An instruction from 052 through 055 or instruction 060 or 061 is in CP 2.

- Instruction 056 or 057 is in CP 3.

- If instruction 14x or 175 is in progress in the full vector logical functional unit, the VM is busy for (VL)/2 + 5 CPs.

- If instruction 003 is in progress, the VM is busy for 3 CPs.

Instructions 073*i*21, 073*i*31, and 073*ij*1 hold issue for 1 CP and continue to hold if a floating-point instruction was issued during the previous 13 CPs.

Instructions 073*i*02, 073*ij*3, and 073*ij*6 hold issue for 3 CPs and continue to hold if a shared paths access conflict occurs with another CPU. Refer to "Shared Paths Access Priority" in Section 2 for more information.

Instructions 073*ij*1 and 073*i*05 hold issue if the status register is busy.

Instruction 073*ij*6 holds issue if the A*j* register is reserved.

Execution Time

Instruction 073 issues in 1 CP.

For instructions 073*i*00, 073*i*10, 073*i*21, 073*i*31, and 073*ij*1, the S*i* register is ready in 1 CP.

For instruction 073*i*05, the status register remains busy for 5 CPs plus 2 CPs after the following conditions are satisfied:

- No scalar memory references are in CP 1 through CP 5.
- Ports A, B, and C are not busy.
- No floating-point instructions were issued in the previous CP.

Description

Instruction 073*i*00, when executed in Y-MP mode, transmits the 64-bit contents of the VM register to the S*i* register. The VM register is usually read after being set by instruction 175.

Instruction 073*i*00, when executed in C90 mode, transmits the 64-bit contents of VM register 0 (lower) to the S*i* register. These 64 bits represent elements 0 through 63.

Instruction 073*i*02 loads the semaphore registers from the 32 high-order bits of the S*i* register. SM00 receives bit 2^{63} of the contents of the S*i* register.

Instruction 073*i*10 transmits the 64-bit contents of VM register 1 (upper) into the S*i* register. These 64 bits represent elements 64 through 127.

Instruction 073*i*21 reads consecutive 16-bit segments of performance monitor (PM) counters 0 through 17_8 into bits 2^{32} through 2^{47} of the S*i* register. Instruction 073*i*31 reads consecutive 16-bit segments of PM counters 20_8 through 37_8 into bits 2^{32} through 2^{47} of the S*i* register. Neither of these instructions should be issued if the PM is busy (if bit 2^{47} of status register 0 is set).

Each PM counter is 48 bits wide and is divided into three 16-bit segments. A performance counter pointer selects the 16-bit segment to be read into the S register. This pointer is cleared on entry to or exit from monitor mode or by instruction 001500. Each successive execution of instruction 073*i*21 or 073*i*31 advances the pointer, enabling the next instruction of the same type to read the next 16-bit segment of the appropriate PM counter. A 3-CP delay must occur between successive PM reads for the data to be valid. The read sequences for the PM counters are shown below.

Instruction 073*i*21 reads PM counters 0 through 17_8:

- First read returns counter 00 (bits $2^0 - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$).
- Second read returns counter 00 (bits $2^{16} - 2^{31}$) to S*i* (bits $2^{32} - 2^{47}$).
- Third read returns counter 00 (bits $2^{32} - 2^{47}$) to S*i* (bits $2^{32} - 2^{47}$).
- Fourth read returns counter 01 (bits $2^0 - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$) (48 reads return the pointer to counter 00).

Instruction 073*i*31 reads PM counters 20_8 through 37_8:

- First read returns counter 20 (bits $2^0 - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$).
- Second read returns counter 20 (bits $2^{16} - 2^{31}$) to S*i* (bits $2^{32} - 2^{47}$).
- Third read returns counter 20(bits $2^{32} - 2^{47}$) to S*i* (bits $2^{32} - 2^{47}$).
- Fourth read returns counter 21 (bits $2^0 - 2^{15}$) to S*i* (bits $2^{32} - 2^{47}$) (48 reads return the pointer to counter 20).

For more information on the performance monitor, including a description of the hardware events monitored by each counter, refer to "Performance Monitor" in Section 3.

Instruction 073*i*25 is used in conjunction with instruction 073*i*75 to perform maintenance on the PM counters. Instruction 073*i*25 causes a PM maintenance advance if the CPU is in maintenance mode and if the

PM maintenance bit (bit 2^{62}) is set in the maintenance modes register. Refer to "Testing Performance Counters" in Section 3 for more information on PM maintenance.

Instruction 073*i*75 transmits bits 2^{54} through 2^{63} of the S*i* register contents to the maintenance modes register if the CPU is in maintenance mode. If the CPU is not in maintenance mode, all values in the maintenance modes register are forced to 0. Table 7-4 shows the functions of the bits in the maintenance modes register.

Table 7-4. Maintenance Modes Register Bits

Bit Position	Function
54	Enables instruction buffer parity maintenance.
55	Enables B/T/V/PM parity maintenance.
56	Enables shared register parity maintenance (forced to 0 if no SR or I/O access is allowed).
57	Enables memory / vector register parity maintenance.
58	Maintenance parity value.
59	Disables error correction (forced to 0 if no SR or I/O access is allowed).
60	Enables V0 to check byte for 1771*x*0 (,A0,1 V*j*).
61	Moves read check byte to data field (forced to 0 if no SR or I/O access is allowed).
62	Enables PM maintenance.
63	Disables PM.

Instruction 073*ij*1 transmits status register SR*j* contents to S*i*. Instruction 073*ij*5 transmits the S*i* register contents to SR*j*. The 8 status registers in the CRAY Y-MP C90 mainframe are organized as shown in Figure 7-7. For a detailed description of the the status registers, refer to "Status Registers" in Section 3. Bits 2^{48} through 2^{52} of SR0 are the only bits that can be written to while the system is in user mode; this is done with instruction 073*i*05.

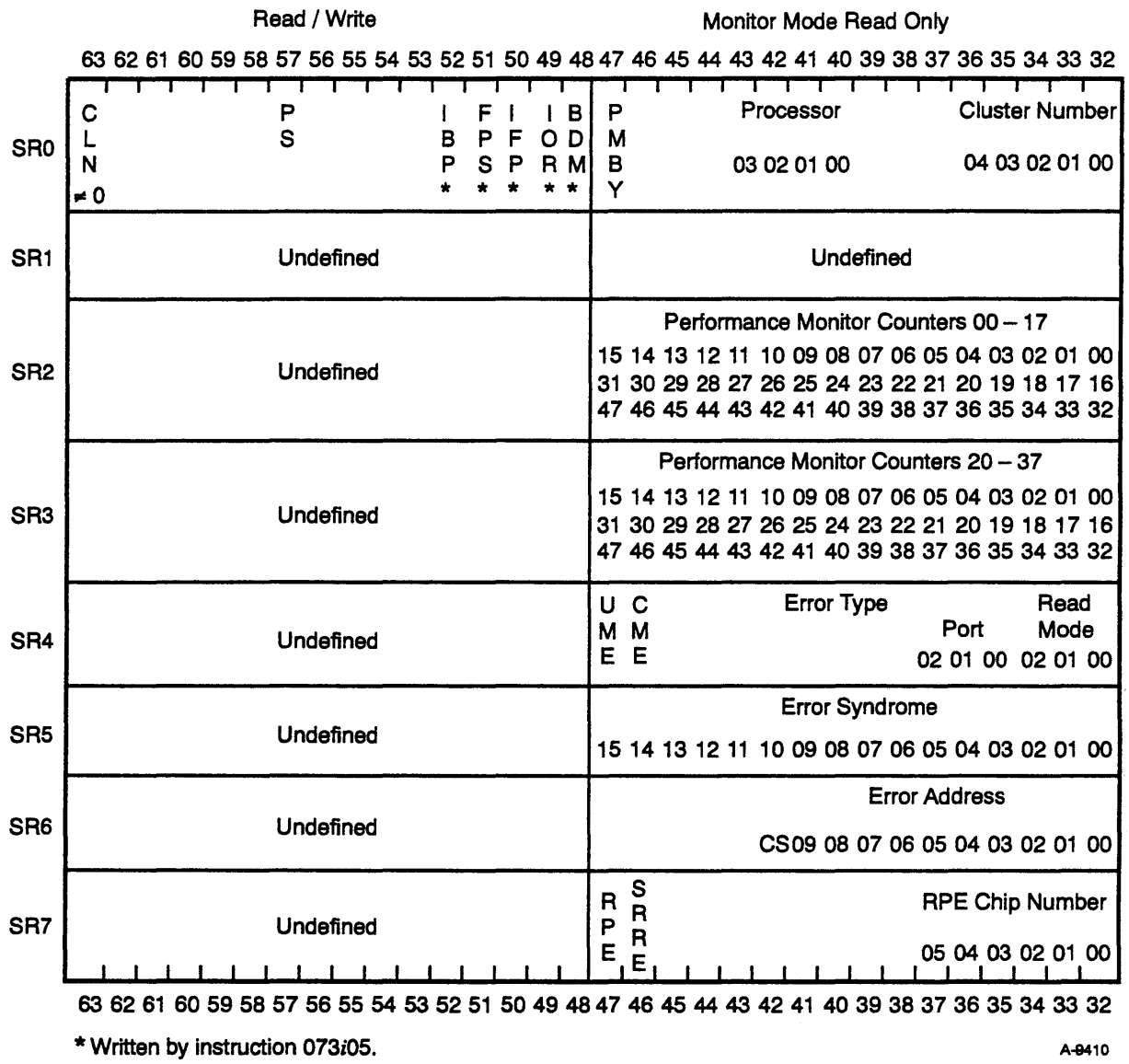


Figure 7-7. Status Registers

Instruction 073*ij*3 transmits the S*i* register contents to the ST*j* register.

Instruction 073*ij*6 transmits the ST register contents designated by the A*j* register contents to S*i*.

Instructions 074 and 075			
Mode	Machine Instruction	CAL Syntax	Description
	074*ijk*	S*i* T*jk*	Transmit (T*jk*) to S*i*.
	075*ijk*	T*jk* S*i*	Transmit (S*i*) to T*jk*.

Special Cases

There are no special cases.

Hold Issue Conditions

Instructions 074 and 075 hold issue if either instruction 036 or instruction 037 is in progress.

Instruction 074 holds issue if instruction 075 was issued in the preceding CP.

Execution Time

Instructions 074 and 075 issue in 1 CP.

For instruction 074, the S*i* register is ready in 1 CP.

Description

Instruction 074 transmits the T register contents specified by *jk* to the S*i* register.

Instruction 075 transmits the S*i* register contents to the T register specified by *jk*.

Instructions 076 and 077			
Mode	Machine Instruction	CAL Syntax	Description
	076*ijk*	S*i* V*j*,A*k*	Transmit (V*j* element (A*k*)) to S*i*.
	077*ijk*	V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*).
	077*i*0*k*	V*i*,A*k* 0 §	Clear element (A*k*) of register V*i*.

Special Cases

If $j = 0$, then (S*i*) = 0.

If $k = 0$, then (A*k*) = 1.

Hold Issue Conditions

Instructions 076 and 077 hold issue if the A*k* register is reserved (except A0).

Instruction 076 holds issue if the S*i* register is reserved or if the V*j* register is reserved as an operand or a result.

Instruction 077 holds issue if the S*j* register is reserved or if the V*i* register is reserved as an operand or a result.

Execution Time

Instructions 076 and 077 issue in 1 CP.

For instruction 076, the S*i* register is ready in 5 CPs.

For instruction 077, the V*i* register is ready in 4 CPs.

Description

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

Instruction 076*ijk* reads the address of a specific vector element from the 7 low-order bits of A*k* (6 bits in Y-MP mode). It then transmits the contents of that element of register V*j* to register S*i*.

Instruction 077*ijk* transmits the contents of register S*j* to the element of register V*i* that is specified by the 7 low-order bits of A*k*. Element 1 (the second element of register V*i*) is selected if the *k* designator is 0.

Instruction 077*i*0*k* enters zeroes into the element of register V*i* that is specified by the 7 low-order bits (6 bits in Y-MP mode) of A*k*. The second element of register V*i* (element 1) is cleared if the *k* designator is 0.

Instructions 10*h* through 13*h*			
Mode	Machine Instruction	CAL Syntax	Description
	10*hi*00 *nm*	A*i* *exp*,A*h*	Read from memory address ((A*h*) + *exp* + (DBA)) to A*i*.
	100*i*00 *nm*	A*i* *exp*,0 §	Read from memory address (*exp* + (DBA)) to A*i*.
	100*i*00 *nm*	A*i* *exp*, §	Read from memory address (*exp* + (DBA)) to A*i*.
	10*hi*00 00	A*i* ,A*h* §	Read from memory address ((A*h*) + (DBA)) to A*i*.
	11*hi*00 *nm*	*exp*,A*h* A*i*	Write (A*i*) to memory address ((A*h*) + *exp* + (DBA)).
	110*i*00 *nm*	*exp*,0 A*i* §	Write (A*i*) to memory address (*exp* + (DBA)).
	110*i*00 *nm*	*exp*, A*i* §	Write (A*i*) to memory address (*exp* + (DBA)).
	11*hi*00 00	,A*h* A*i* §	Write (A*i*) to memory address ((A*h*) + (DBA)).
	12*hi*00 *nm*	S*i* *exp*,A*h*	Read from memory address ((A*h*) + *exp* + (DBA)) to S*i*.
	120*i*00 *nm*	S*i* *exp*,0 §	Read from memory address (*exp* + (DBA)) to S*i*.
	120*i*00 *nm*	S*i* *exp*, §	Read from memory address (*exp* + (DBA)) to S*i*.
	12*hi*00 00	S*i* ,A*h* §	Read from memory address ((A*h*) + (DBA)) to S*i*.
	13*hi*00 *nm*	*exp*,A*h* S*i*	Write (S*i*) to memory address ((A*h*) + *exp* + (DBA)).
	130*i*00 *nm*	*exp*,0 S*i* §	Write (S*i*) to memory address (*exp* + (DBA)).
	130*i*00 *nm*	*exp*, S*i* §	Write (S*i*) to memory address (*exp* + (DBA)).
	13*hi*00 00	,A*h* S*i* §	Write (S*i*) to memory address ((A*h*) + (DBA)).

Special Cases

If $h = 0$, then A$h = 0$.

Hold Issue Conditions

Instructions 10*h* through 13*h* hold issue for any of the following conditions:

- Port A, B, or C is busy.
- Register A*h* is reserved, $h \neq 0$.

- Three instructions between 10*h* and 13*h* are in CP 1, CP 3, and CP 4, and there is a memory conflict with the instruction in CP 4.
- The status register is busy.
- The second and/or third parcel of the instruction is not in a buffer (a 26-CP delay occurs).
- A*i* is reserved (Instructions 10*h* and 11*h* hold issue).
- S*i* is reserved (Instructions 12*h* and 13*h* hold issue).

Execution Time

The issue times for instructions 10*h* through 13*h* are as follows:

- If all parcels are in the same buffer, the issue time is 2 CPs.
- If parcel 0 is in one buffer and parcels 1 and 2 are in another buffer, the issue time is 5 CPs.
- If parcels 0 and 1 are in one buffer and parcel 2 is in another buffer, the issue time is 6 CPs.

For instruction 10*h*, the A*i* register is ready in 24 CPs.

For instruction 12*h*, the S*i* register is ready in 23 CPs.

A subsection is ready for the next scalar read or write operation in 7 CPs if no conflicts occur.

After issuing an instruction between 10*h* and 13*h*, an attempt to issue an instruction between 034 and 037 or instruction 176 or 177 causes ports A, B, or C to be considered busy for 5 CPs (plus additional CPs if there are conflicts).

Description

Instructions 10*h* through 13*h* transmit data between memory and an A register or an S register.

For these instructions, only the value of the expression is used if the *h* designator is 0 or if a 0 or blank field is used in place of A*h*. Only the contents of A*h* are used if the expression is omitted. An expression with a parcel-address attribute causes an assembly error.

Instructions 10*hi*00 *nm* through 10*hi*00 00 read the 32 low-order bits of a memory word directly into an A register. The memory address is determined by adding the address in the A*h* register to the value of the expression in the *nm* field.

Instructions 11*hi*00 *nm* through 11*hi*00 00 write 32 bits from register A*i* directly into memory. The high-order bits of the memory word are cleared. The memory address is determined by adding the address in the A*h* register to the value of the expression in the *nm* field.

Instructions 12*hi*00 *nm* through 12*hi*00 00 read the contents of a memory word directly into an S register. The memory address is determined by adding the address in the A*h* register to the value of the expression in the *nm* field.

Instructions 13*hi*00 *nm* through 13*hi*00 00 write the contents of register S*i* directly into memory. The memory address is determined by adding the address in the A*h* register to the value of the expression in the *nm* field.

Instructions 140 through 147			
Mode	Machine Instruction	CAL Syntax	Description
	140*ijk*	V*i* S*j*&V*k*	Transmit the logical products of (S*j*) and (V*k* elements) to V*i* elements.
	141*ijk*	V*i* V*j*&V*k*	Transmit the logical products of (V*j* elements) and (V*k* elements) to V*i* elements.
	142*ijk*	V*i* S*j*!V*k*	Transmit the logical sums of (S*j*) and (V*k* elements) to V*i* elements.
	142*i*0*k*	V*i* V*k* §	Transmit (V*k* elements) to V*i* elements.
	143*ijk*	V*i* V*j*!V*k*	Transmit the logical sums of (V*j* elements) and (V*k* elements) to V*i* elements.
	144*ijk*	V*i* S*j* \V*k*	Transmit the exclusive ORs of (S*j*) and (V*k* elements) to V*i* elements.
	145*ijk*	V*i* V*j* \V*k*	Transmit the exclusive ORs of (V*j* elements) and (V*k* elements) to V*i* elements.
	145*iii*	V*i* 0 §	Clear the elements of V*i*.
	146*ijk*	V*i* S*j*!V*k*&VM	Transmit (S*j*) if VM bit = 1, or (V*k* element) if VM bit = 0, to V*i* elements.
	146*i*0*k*	V*i* #VM&V*k* §	Transmit the vector merge of (V*k* elements) and 0 to V*i* elements.
	147*ijk*	V*i* V*j*!V *k*&VM	Transmit (V*j* elements) if VM bit = 1, or (V*k* elements) if VM bit = 0, to V*i* elements.

Special Cases

If $j = 0$, then (Sj) = 0.

Hold Issue Conditions

Instructions 140 through 147 hold issue for any of the following conditions:

- The V*k* register is reserved as an operand.
- The V*i* register is reserved as an operand or as a result.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.

Instructions 140, 142, 144, and 146 hold issue if the S*j* register is reserved.

Instructions 141, 143, 145, and 147 hold issue if the V*j* register is reserved as an operand.

Instructions 146 and 147 hold issue if instruction 14x or instruction 175 is in progress or if instruction 003 was issued in the preceding CP. For these conditions, the full vector logical functional unit is busy for (VL)/2 + 4 CPs.

Instructions 140 and 145 hold issue if the second vector logical functional unit is disabled and if instruction 14x or instruction 175 is in progress. In this condition, the full vector logical functional unit is busy for (VL)/2 + 4 CPs.

Instructions 140 and 145 hold issue for the following conditions if the second vector logical functional unit is enabled:

- An instruction between 140 and 145 or instruction 16x is in progress in the second vector logical/floating-point multiply functional unit. In this condition, the second vector logical functional unit is busy for (VL)/2 + 4 CPs.
- An instruction between 140 and 147 or instruction 175 is in progress in the full vector logical functional unit, or instruction 003 was issued in the preceding CP. For these conditions, the full vector logical functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instructions 140 through 147 issue in 1 CP.

If data is available, register V*j* or V*k* is ready in (VL)/2 + 3 CPs.

If data is available, the functional unit is ready in (VL)/2 + 4 CPs.

If data is available for the full vector logical functional unit or for the second vector logical functional unit, register V*i* is ready in (VL)/2 + 5 CPs.

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

Description

The number of operations performed is determined by the contents of the VL register. All operations start with element 0 of the V*i*, V*j*, or V*k* registers and increment the element number by 1 for each operation performed. All results are delivered to register V*i*.

Instructions 140 through 145 can be executed in either the full vector logical or the second vector logical functional units, provided the second vector logical unit is enabled. If the second vector logical unit is disabled, instructions 140 through 145 can be executed only in the full vector logical unit. Instructions 146 and 147 execute in the full vector logical unit only.

For instructions 140, 142, 144, and 146, a copy of (S*j*) is delivered to the functional unit. The copy is held as one of the operands until the operation is complete. Therefore, (S*j*) can be changed in the next instruction without affecting the vector operation. For instructions 141, 143, 145, and 147, all operands are obtained from V registers.

Instructions 140 and 141 form the logical products (AND) of operand pairs and enter the results into V*i*. Bits of an element of V*i* are set to 1 when the corresponding bits of (S*j*) or (V*j* element) and (V*k* element) are 1, as in the following example:

```
If   (Sj) or (Vj element)  = 1100
And  (Vk element)          = 1010
Then (Vi element)          = 1000
```

Instructions 142 and 143 form the logical sums (inclusive OR) of operand pairs and deliver the results to V*i*. Bits of an element of V*i* are set to 1 when at least one of the corresponding bits of (S*j*) or (V*j* element) and (V*k* element) is 1, as in the following example:

```
If   (Sj) or (Vj element)  = 1100
And  (Vk element)          = 1010
Then (Vi element)          = 1110
```

Instructions 144 and 145 form the exclusive ORs of operand pairs and deliver the results to V*i*. Bits of an element of V*i* are set to 1 when the corresponding bits of (S*j*) or (V*i* element) are different from (V*k* element), as in the following example:

```
If   (Sj) or (Vj element)  = 1100
And  (Vk element)          = 1010
Then (Vi element)          = 0110
```

Instructions 146 and 147 transmit selected bits of each operand to V*i* depending on the contents of the vector mask register. The VM register stores the vector mask in Y-MP mode, and the VM and VM1 registers together store the vector mask in C90 mode. Operand pairs used for the selection depend on the instruction. For instruction 146, the first operand is always (S*j*), and the second operand is (V*k* element). For instruction 147, the first operand is (V*j* element), and the second operand is (V*k* element). If bit *n* of the VM register is 1, the corresponding bit of the first operand is selected; if bit *n* of the VM register is 0, the corresponding bit of the second operand, (V*k* element), is selected. The following two examples illustrate these points.

Example 1:

The following registers are initialized as shown:

(VL) = 4
(VM) = 06000000000000000000
(S2) = –1
(V6, 00) = 1
(V6, 01) = 2
(V6, 02) = 3
(V6, 03) = 4

Instruction 146726 is executed. Following execution, the first four elements of V7 contain the following values:

(V7, 00) = 1
(V7, 01) = –1
(V7, 02) = –1
(V7, 03) = 4

The remaining elements of V7 are not altered.

Example 2:

The following registers are initialized as shown:

(VL) = 4
(VM) = 06000000000000000000000
(V2, 00) = 1 (V3, 00) = –1
(V2, 01) = 2 (V3, 01) = –2
(V2, 02) = 3 (V3, 02) = –3
(V2, 03) = 4 (V3, 03) = –4

Instruction 147123 is executed. Following execution, the first four elements of V1 contain the following values:

(V1, 00) = –1
(V1, 01) = 2
(V1, 02) = 3
(V1, 03) = –4

The remaining elements of V1 are not altered.

Instructions 150 and 151			
Mode	Machine Instruction	CAL Syntax	Description
	150*ijk*	V*i* V*j* < A*k*	Shift (V*j* elements) left (A*k*) places to V*i* elements.
	150*ij*0	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 (A*k*) places to V*i* elements.
	151*ij*0	V*i* V*j* > 1 §	Shift (V*j* elements) right one place to V*i* elements.
	005400 150*ij*0	V*i* V*j* < V0	Shift (V*j* elements) left (V0) places to V*i* elements.
	005400 151*ij*0	V*i* V*j* > V0	Shift (V*j* elements) right (V0) places to V*i* elements.

Special Cases

If $k = 0$, then (Ak) = 1.

Hold Issue Conditions

Instructions 150 and 151 hold issue for any of the following conditions:

- The V*j* register is reserved as an operand.
- The V*i* register is reserved as an operand or as a result.
- The A*k* register is reserved (except A0).
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.

Instructions 005400 150 and 005400 151 hold issue if V0 is reserved.

If instructions 150 through 153 are in progress, the vector shift functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

The following execution times apply if data is available:

- The V*j* register is ready in (VL)/2 + 3 CPs.
- The V*i* register is ready in (VL)/2 + 7 CPs.
- The functional unit is ready in (VL)/2 + 4 CPs.

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

Description

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

All shifts are end-off with zero fill, meaning that data shifted out of a register, either to the right or the left, is lost, and that the trailing edge of the data is replaced in the register with zeroes. Unlike shift instructions 052 through 055, instructions 150 and 151 obtain the shift count from (A*k*), rather than from the *jk* fields of the instruction, and all 32 bits of (A*k*) are used for the shift count. The elements of V*i* are cleared if the shift count exceeds 63. All shift counts obtained from (A*k*) are considered positive.

Instruction 150*ijk* shifts the contents of the elements of register V*j* to the left by the amount specified by (A*k*) and enters the results into the elements of V*i*. The special CAL form of this instruction shifts the contents of the elements of V*j* one place to the left and enters the results into V*i*.

Instruction 151*ijk* shifts the contents of the elements of register V*j* to the right by the amount specified by (A*k*) and enters the results into the elements of V*i*. The special CAL form of this instruction shifts the contents of the elements of V*j* one place to the right and enters the results into V*i*.

Instruction 005400 150*ij*0 shifts the contents of the elements of register V*j* to the left by the amount specified by (elements of V0) and enters the results into the elements of V*i*. The 6 low-order bits of each element of the V0 register contain the shift counts for the corresponding elements of V*j*. If bits 2^6 through 2^{63} of an element of V0 are not equal to 0, a 0 result is entered into the corresponding element of V*i*.

Instruction 005400 151*ij*0 shifts the contents of the elements of register V*j* to the right by the amount specified by the contents of the elements of V0 and enters the results into the elements of V*i*. The 6 low-order bits of each element of the V0 register contain the shift counts for the corresponding elements of V*j*. If bits 2^6 through 2^{63} of an element of V0 are not equal to 0, a 0 result is entered into the corresponding element of V*i*.

Instructions 152 and 153			
Mode	Machine Instruction	CAL Syntax	Description
	152*ijk*	V*i* V*j*,V*j* < A*k*	Double shift (V*j* elements) left (A*k*) places to V*i* elements.
	152*ij*0	V*i* V*j*,V*j* < 1 §	Double shift (V*j* elements) left one place to V*i* elements.
	005400 152*ijk*	V*i* V*j*,A*k*	Transfer VL–(A*k*) elements of V*j* starting at element (A*k*) to V*i* starting at element 0. ((A*k*) < VL)
	153*ijk*	V*i* V*j*,V*j* > A*k*	Double shift (V*j* elements) right (A*k*) places to V*i* elements.
	153*ij*0	V*i* V*j*,V*j* > 1 §	Double shift (V*j* elements) right one place to V*i* elements.

Special Cases

If $k = 0$, then (Ak) = 1.

Hold Issue Conditions

Instructions 152 and 153 hold issue for any of the following conditions:

- The V*j* register is reserved as an operand.
- The V*i* register is reserved as an operand or a result.
- The A*k* register is reserved (except A0).
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or instruction 077 was issued during the previous 3 CPs.

If instructions 150 through 153 are in progress, the vector shift functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instructions 152 and 153 issue in 1 CP.

If data is available, the V*j* register is ready in (VL)/2 + 3 CPs.

If data is available, the functional unit is ready in (VL)/2 + 4 CPs.

For the 152 instruction, if data is available, the V*i* register is ready in (VL)/2 + 8 CPs.

For the 153 instruction, if data is available, the V*i* register is ready in (VL)/2 + 7 CPs.

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

Description

Instructions 152 and 153 execute in the vector shift functional unit. The instructions shift 128-bit values formed by logically joining the contents of two elements of the V*j* register. The direction of the shift determines whether the high-order bits or the low-order bits of the result are sent to V*i*. All shifts are end-off with zero fill, meaning that data shifted out of the combined registers, either to the right or the left, is lost, and that the trailing edge of the data is replaced in the registers with zeroes. Shift counts are obtained from the A*k* register. The number of operations is determined by the VL register contents.

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

If (VL) > 1, the operation begins by concatenating element 1 of V*j* to element 0 of V*j*, and by left shifting the resulting 128-bit quantity by the amount specified by (A*k*). The 64 high-order bits remaining after the shift are transmitted to element 0 of V*i*. Figure 7-8 shows this operation.

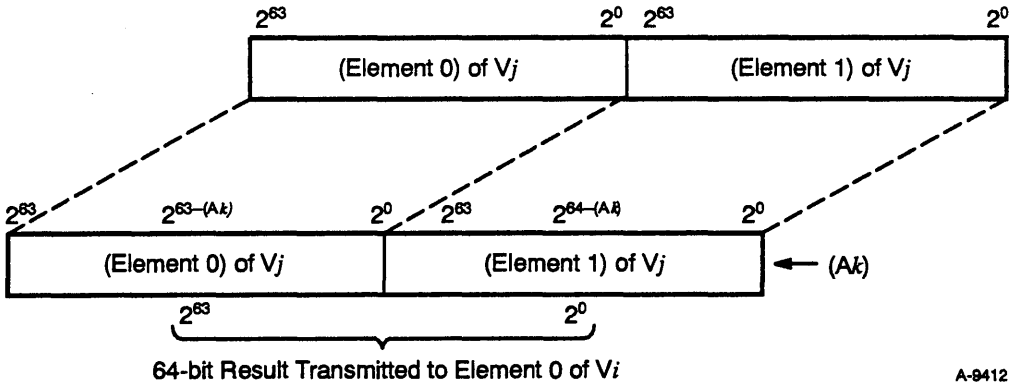


Figure 7-8. Vector Left Double Shift, First Element, (VL)>1

The operation continues by concatenating element 2 (if (VL) > 2) to element 1, and by left shifting the resulting 128-bit quantity by the amount specified by (A*k*). The 64 high-order bits remaining after the shift are transmitted to element 1 of V*i*. Figure 7-9 shows this operation.

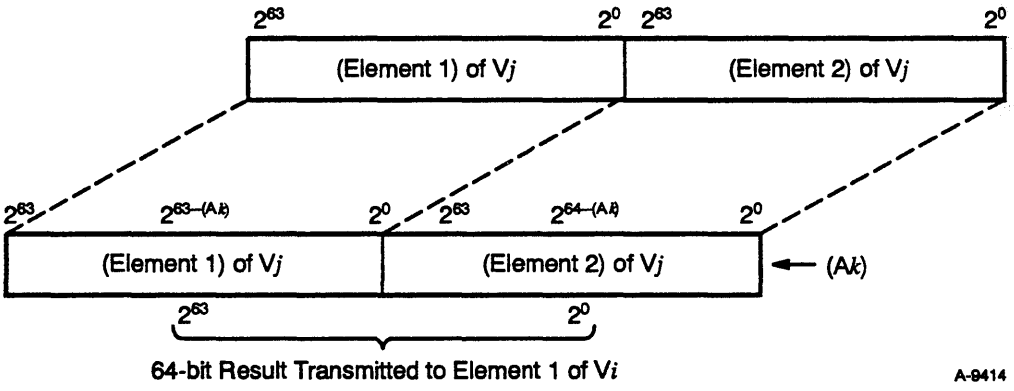


Figure 7-9. Vector Left Double Shift, Second Element, (VL)>2

If (VL) = 2, 64 bits of 0's are concatenated to element 1, and the resulting 128-bit quantity is left shifted by the amount specified by (A*k*). The 64 high-order bits remaining in element 1 of V*j* after the shift are transmitted to element 1 of V*i*. Only these two operations are performed. In general, the last element of V*j*, as determined by (VL), has 64 bits of 0's concatenated to it before the final shift is performed. Figure 7-10 shows this operation.

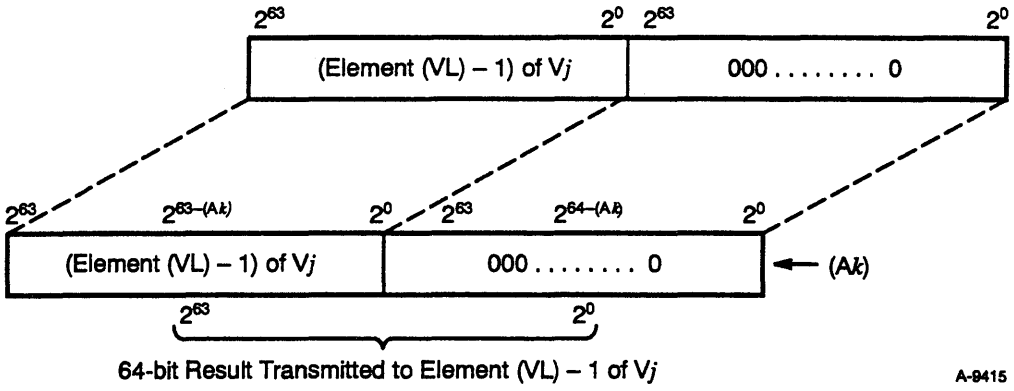


Figure 7-10. Vector Left Double Shift, Last Element

If (A*k*) is greater than or equal to 128, the resulting elements of V*i* contain all 0's. If (A*k*) is greater than 64 but less than 128, each element of the V*i* register contains at least (A*k*) – 64 zeroes.

The following example shows a vector left double shift. The registers are initialized as shown:

```
(VL)     = 4
(A1)     = 3
(V4, 00) = 0 00000 0000 0000 0000 0007
(V4, 01) = 0 60000 0000 0000 0000 0005
(V4, 02) = 1 00000 0000 0000 0000 0006
(V4, 03) = 1 60000 0000 0000 0000 0007
```

Instruction 152541 is executed. Following execution, the first four elements of V5 contain the following values:

```
(V5, 00) = 0 00000 0000 0000 0000 0073
(V5, 01) = 0 00000 0000 0000 0000 0054
(V5, 02) = 0 00000 0000 0000 0000 0067
(V5, 03) = 0 00000 0000 0000 0000 0070
```

Instruction 153 performs right shifts. The operation starts with element 0 of V*j*, which is first concatenated to 64 bits of 0's. The resulting 128-bit quantity is then right shifted by the amount specified by (A*k*). The 64 low-order bits remaining in element 0 of V*j* after the shift are transmitted to element 0 of V*i*. If (VL) = 1, this is the only operation performed. Figure 7-11 shows this operation.

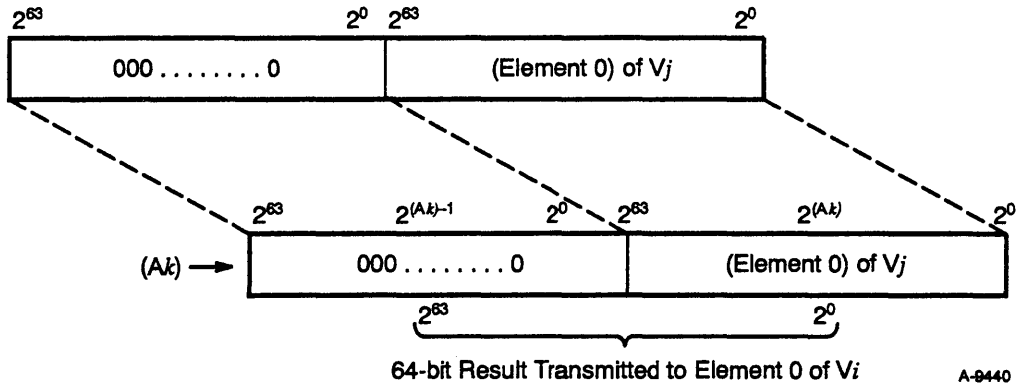


Figure 7-11. Vector Right Double Shift, First Element

If (VL) > 1, the operation continues by concatenating element 1 of V*j* to element 0 of V*j*, and by right shifting the resulting 128-bit quantity by the amount specified by (A*k*). The 64 low-order bits remaining in element 1 after the shift are transmitted to element 1 of V*i*. Figure 7-12 shows this operation.

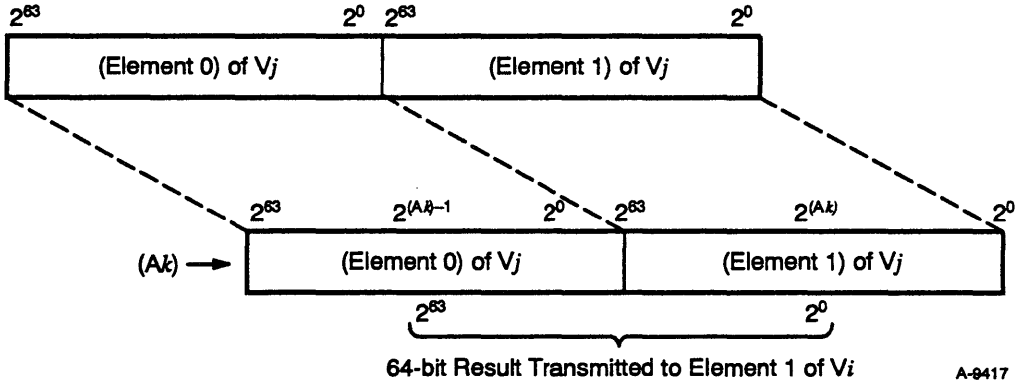


Figure 7-12. Vector Right Double Shift, Second Element, (VL)>1

The last operation performed by instruction 153 concatenates the last element of V*j*, as determined by the contents of VL, to the preceding element before performing the right shift. The 64 low-order bits remaining in the last element after the shift are transmitted to the corresponding element of V*i*. Refer to Figure 7-13.

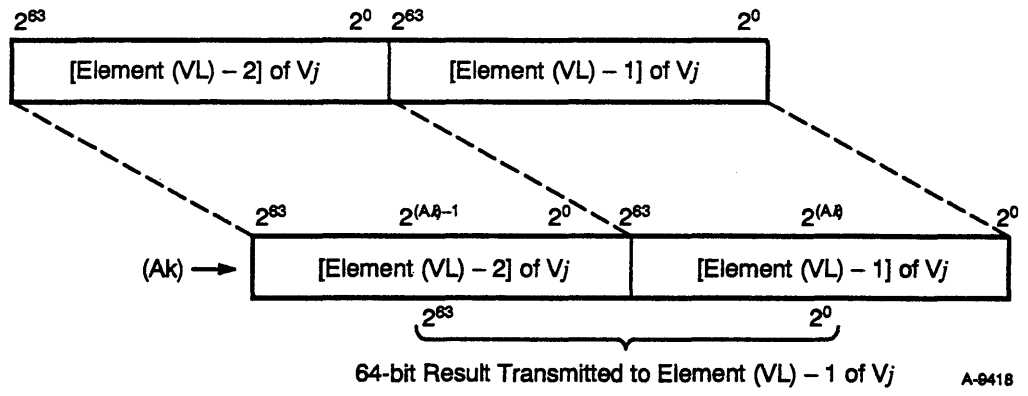


Figure 7-13. Vector Right Double Shift, Last Operation

The following example shows a vector right double shift. The registers are initialized as shown:

(VL) = 4
(A6) = 3
(V2, 00) = 0 00000 0000 0000 0000 0017
(V2, 01) = 0 60000 0000 0000 0000 0006
(V2, 02) = 1 00000 0000 0000 0000 0006
(V2, 03) = 1 60000 0000 0000 0000 0007

Instruction 153026 is executed. After execution, register V0 contains the following values:

(V0, 00) = 0 00000 0000 0000 0000 0001
(V0, 01) = 1 66000 0000 0000 0000 0000
(V0, 02) = 1 50000 0000 0000 0000 0000
(V0, 03) = 1 56000 0000 0000 0000 0000

The remaining elements of register V0 are not altered.

Instruction 005400 152*ijk* performs a vector word shift. This operation transfers the contents of elements (A*k*) through (VL) – 1 of V*j* to elements 0 through [(VL) – 1 – (A*k*)] of V*i*. (VL)–(A*k*) elements are transferred. If (A*k*) ≥ (VL), no elements are transferred. Figure 7-14 shows an example of a vector word shift, in which the registers are initialized as shown, and instruction 005400 152123 is executed.

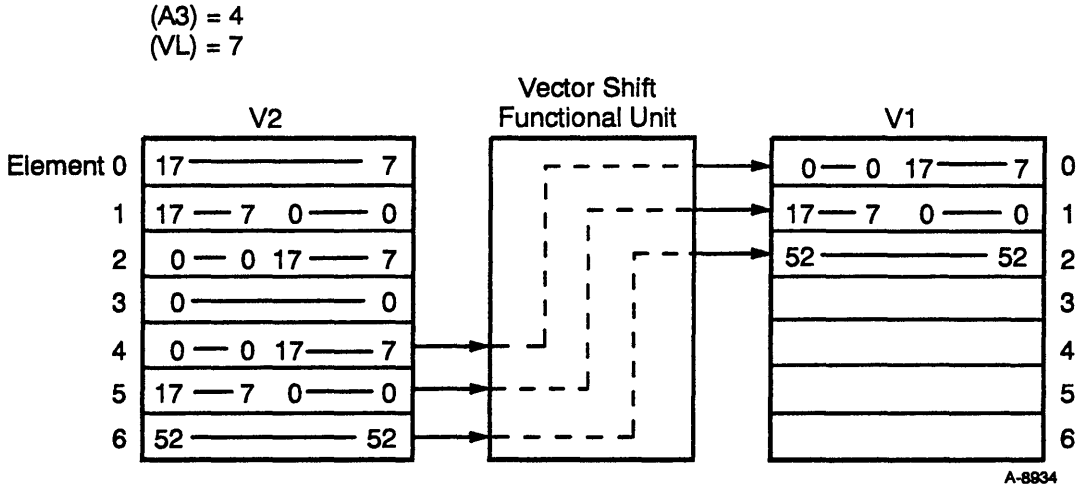


Figure 7-14. Vector Word Shift

Instructions 154 through 157			
Mode	Machine Instruction	CAL Syntax	Description
	154*ijk*	V*i* S*j* + V*k*	Transmit the integer sums of (S*j*) and (V*k* elements) to V*i* elements.
	155*ijk*	V*i* V*j* + V*k*	Transmit the integer sums of (V*j* elements) and (V*k* elements) to V*i* elements.
	156*ijk*	V*i* S*j* – V*k*	Transmit the integer differences of (S*j*) and (V*k* elements) to V*i* elements.
	156*i*0*k*	V*i* –V*k* §	Transmit the two's complement of (V*k* elements) to V*i* elements.
	157*ijk*	V*i* V*j* – V*k*	Transmit the integer differences of (V*j* elements) and (V*k* elements) to V*i* elements.

Special Cases

For instruction 154, if $j = 0$, then (Sj) = 0 and (Vi element) = (Vk element).

For instruction 156, if $j = 0$, then (Sj) = 0 and (Vi element) = –(Vk element).

Hold Issue Conditions

Instructions 154 through 157 hold issue for any of the following conditions:

- The Vk register is reserved as an operand.
- The Vi register is reserved as an operand or a result.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued in the previous 3 CPs.

Instructions 154 and 156 hold issue if the Sj register is reserved (except S0).

Instructions 155 and 157 hold issue if the Vj register is reserved as an operand.

If an instruction between 154 and 157 is in progress, the vector shift functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instructions 154 through 157 issue in 1 CP.

The following execution times apply if data is available:

- The V*j* or V*k* register is ready in (VL)/2 + 3 CPs.
- The V*i* register is ready in (VL)/2 + 7 CPs.
- The functional unit is ready in (VL)/2 + 4 CPs.

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

Description

Instructions 154 through 157 execute in the vector add functional unit. Instructions 154 and 155 perform integer addition; instructions 156 and 157 perform integer subtraction. No overflow is detected. The number of additions or subtractions performed is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to elements of V*i*.

Instructions 154 and 156 transmit a copy of the contents of S*j* to the functional unit, where the copy is retained as one of the operands until the vector operation is completed. The other operand is an element of V*k*. For instructions 155 and 157, both operands are obtained from V registers.

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

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

Instruction 156*ijk* subtracts the contents of each element of V*k* from the contents of the S*j* register and enters the results into the elements of V*i*.

Instruction 156*i*0*k* transmits the negative (two's complement) of each element of V*k* to V*i*.

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

Instructions 160 through 167			
Mode	Machine Instruction	CAL Syntax	Description
	160*ijk*	V*i* S*j**FV*k*	Transmit the floating-point products of (S*j*) and (V*k* elements) to V*i* elements.
	161*ijk*	V*i* V*j**FV*k*	Transmit the floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements.
	162*ijk*	V*i* S*j**HV*k*	Transmit the 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*	Transmit the 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*	Transmit the rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements.
	165*ijk*	V*i* V*j**RV*k*	Transmit the rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements.
	166*ijk*	V*i* S*j**V*k*	Transmit the 32-bit integer products of (S*j*) and (V*k* elements) to V*i* elements.
	167*ijk*	V*i* V*j**V*k*	Transmit the reciprocal iterations 2 – (V*j* elements) * (V*k* elements) to V*i* elements.

Special Cases

If $j = 0$, then (Sj) = 0.

Hold Issue Conditions

Instructions 160 through 167 hold issue for any of the following conditions:

- The V*k* register is reserved as an operand.
- The V*i* register is reserved as an operand or as a result.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.
- The status register is busy.

Instructions 160, 162, 164, and 166 hold issue if the S*j* register is reserved (except S0).

Instructions 161, 163, 165, and 167 hold issue if the V*j* register is reserved as an operand.

If an instruction between 140 and 145 is in progress in the second vector logical functional unit or if instruction 16x is in progress, the floating-point multiply functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instructions 160 through 167 issue in 1 CP.

The following execution times apply if data is available:

- The V*j* and V*k* registers are ready in (VL)/2 + 3 CPs
- The V*i* register is ready in (VL)/2 + 10 CPs
- The functional unit is ready in (VL)/2 + 4 CPs

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

Description

Instructions 160 through 167 execute in the floating-point multiply functional unit. The number of operations performed by an instruction is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each successive operation.

The functional unit assumes that operands are in floating-point format. Instructions 160, 162, 164, and 166 send a copy of the contents of the S*j* register to the functional unit, where the copy is retained as one of the operands until the vector operation is completed. The other operand is an element of V*k*. For instructions 161, 163, 165, and 167, both operands are obtained from V registers.

All results are transmitted to the elements of V*i*. If one of the operands is not normalized, the products may or may not be normalized. If neither operand is normalized, the products are not normalized.

Instruction 160*ijk* forms the floating-point products of the contents of the S*j* register and of each element of V*k*, and it enters the results into the elements of V*i*.

Instruction 161*ijk* forms the floating-point products of the contents of the elements of V*j* and the contents of the corresponding elements of V*k*, and it enters the results into the elements of V*i*.

Instruction 162*ijk* forms the half-precision rounded floating-point products of the contents of the S*j* register and the contents of each element of the V*k* register, and it enters the results into the elements of V*i*. This instruction can be used in a division algorithm when the result only needs to be accurate to 30 bits.

Instruction 163*ijk* forms the half-precision rounded floating-point products of the contents of the elements of V*j* and of the corresponding elements of V*k*, and it enters the results into the elements of V*i*. This instruction can be used in a division algorithm when only 30 bits of accuracy is required.

Instruction 164*ijk* forms the rounded floating-point products of the contents of the S*j* register and of each element of V*k*, and it enters the results into the elements of V*i*.

Instruction 165*ijk* forms the rounded floating-point products of the contents of the elements of V*j* and of the corresponding elements of V*k*, and it enters the results into the elements of V*i*.

Instruction 166*ijk* forms the 32-bit integer products of the contents of the S*j* register and of each element of V*k*, and it enters the results into the elements of V*i*. The S*j* operand must be left shifted by 31_{10} places, and the V*k* operand must be left shifted by 16_{10} places before instruction 166 is executed.

Instruction 167*ijk* forms the quantity of 2 minus the floating-point products of the contents of the elements of V*j* and V*k*, and it enters the results into the elements of V*i*. This instruction is used in the division operation sequence of instructions.

Instructions 170 through 173			
Mode	Machine Instruction	CAL Syntax	Description
	170*ijk*	V*i* S*j* + FV*k*	Transmit the floating-point sums of (S*j*) and (V*k* elements) to V*i* elements.
	170*i*0*k*	V*i* + FV*k* §	Transmit the normalized (V*k* elements) to V*i* elements.
	171*ijk*	V*i* V*j* + FV*k*	Transmit the floating-point sums of (V*j* elements) and (V*k* elements) to V*i* elements.
	172*ijk*	V*i* S*j* – FV*k*	Transmit the floating-point differences of (S*j*) and (V*k* elements) to V*i* elements.
	172*i*0*k*	V*i* – FV*k* §	Transmit the normalized negatives of (V*k* elements) to V*i* elements.
	173*ijk*	V*i* V*j* – FV*k*	Transmit the floating-point differences of (V*j* elements) and (V*k* elements) to V*i* elements.

Special Cases

If $j = 0$, then (Sj) = 0.

Hold Issue Conditions

Instructions 170 through 173 hold issue for any of the following conditions:

- The V*k* register is reserved as an operand.
- The V*i* register is reserved as an operand or as a result.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.
- The status register is busy.

Instructions 170 and 172 hold issue if the S*j* register is reserved (except S0).

Instructions 171 and 173 hold issue if the V*j* register is reserved as an operand.

If instructions 170 through 173 are in progress, the floating-point add functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instructions 170 through 173 issue in 1 CP.

The following execution times apply if data is available:

- The V*j* and V*k* registers are ready in (VL)/2 + 3 CPs.
- The V*i* register is ready in (VL)/2 + 10 CPs.
- The functional unit is ready in (VL)/2 + 4 CPs.

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

Description

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

Instructions 170 and 172 transmit a copy of the contents of the S*j* register to the functional unit, where the copy is retained as one of the operands until the vector operation is completed. The other operand is an element of V*k*. For instructions 171 and 173, both operands are obtained from V registers.

Instruction 170*ijk* forms the floating-point sums of the contents of the S*j* register and the contents of each element of V*k*, and it enters the results into the elements of V*i*.

Instruction 170*i*0*k* is a special CAL form of instruction 170; it normalizes the contents of the elements of V*k* and enters the results into the elements of V*i*.

Instruction 171*ijk* forms the floating-point sums of the contents of the elements of V*j* and the contents of the corresponding elements of V*k*, and it enters the results into the elements of V*i*.

Instruction 172*ijk* forms the floating-point differences of the contents of the S*j* register and the contents of each element of V*k*, and it enters the results into the elements of V*i*.

Instruction 172*i*0*k* is a special CAL form of instruction 172; it forms the normalized negatives (twos complements) of the contents of the elements of V*k* and enters the results into the elements of V*i*.

Instruction 173*ijk* forms the floating-point differences of the contents of the elements of V*j* and the contents of the corresponding elements of V*k*, and it enters the results into the elements of V*i*.

Instruction 174*ij*0			
Mode	Machine Instruction	CAL Syntax	Description
	174*ij*0	V*i* /HV*j*	Transmit the floating-point reciprocal approximation of (V*j* elements) to V*i* elements.

Special Cases

If the contents of a particular element of V*j* are not normalized, the contents of the corresponding element of V*i* are invalid. For a floating-point number to be normalized, bit 2^{47} must be set to 1. This bit is not tested.

Hold Issue Conditions

Instruction 174 holds issue for any of the following conditions:

- The V*j* register is reserved as an operand.
- The V*i* register is reserved as an operand or as a result.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.
- The status register is busy.

If instruction 174 is in progress, the reciprocal approximation functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instruction 174 issues in 1 CP.

The following execution times apply if data is available:

- The V*j* register is ready in (VL)/2 + 3 CPs.
- The V*i* register is ready in (VL)/2 + 14 CPs.
- The functional unit is ready in (VL)/2 + 4 CPs.

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

Description

Instruction 174*ij*0 executes in the reciprocal approximation functional unit. The instruction forms an approximate value of the reciprocal of the normalized floating-point quantity in each element of V*j* and enters the results into the elements of V*i*. The number of elements for which approximations are found is determined by contents of the VL register.

Instruction 174*ij*0 is used in the division sequence to compute the quotients of floating-point quantities. This instruction produces results with 30 significant bits. The next 3 low-order bits are not necessarily accurate, and the remaining 15 low-order bits are 0's. The number of significant bits can be extended to 48 by using the reciprocal iteration instruction (167*ijk*) and a multiplication instruction.

Instructions 174*ij*1 through 174*ij*3			
Mode	Machine Instruction	CAL Syntax	Description
	174*ij*1	V*i* PV*j*	Transmit the population count of (V*j* elements) to V*i* elements.
	174*ij*2	V*i* QV*j*	Transmit the population count parity of (V*j* elements) to V*i* elements.
	174*ij*3	V*i* ZV*j*	Transmit the leading zero count of (V*j* elements) to V*i* elements.

Special Cases

The following special cases apply to instruction 174*ij*3:

- If (V*j* elements) = 0, then (V*i* elements) = 64_8.
- If (V*j* elements) < 0, then (V*i* elements) = 0.

Hold Issue Conditions

Instructions 174*ij*1 through 174*ij*3 hold issue for any of the following conditions:

- The V*j* register is reserved as an operand.
- The V*i* register is reserved as an operand or as a result.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.

If instruction 174*ij*0 is in progress, the vector population/parity functional unit is busy for (VL)/2 + 6 CPs.

If instruction 174*ij*1, 174*ij*2, or 174*ij*3 is in progress, the vector population/parity functional unit is busy for (VL)/2 + 4 CPs.

If instruction 070 is in progress, the vector population/parity functional unit is busy for 5 CPs.

Execution Time

Instructions 174*ij*1 through 174*ij*3 issue in 1 CP.

The following execution times apply if data is available:

- The V*j* register is ready in (VL)/2 + 3 CPs.
- The V*i* register is ready in (VL)/2 + 8 CPs.
- The functional unit is ready in (VL)/2 + 4 CPs.

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

Description

Instructions 174*ij*1 through 174*ij*3 execute in the vector population/parity/leading zero functional unit, which also shares some logic with the reciprocal approximation functional unit. The number of operations performed by one of these instructions is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each operation performed.

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

Instruction 174*ij*2 counts the number of bits set to 1 in each element of V*j*. The least significant bit of each count tells whether an odd or an even number of 1 bits is set in each element. Only this least significant bit of each count is transferred to the least significant bit position of the corresponding element of register V*i*. The remaining 63 bits of the element are set to 0's. The actual population count results are not transferred.

Instruction 174*ij*3 counts the number of leading 0's in each element of V*j* and enters the counts into the 7 low-order bits of the corresponding elements of V*i*. The remaining bits of each element of V*i* are set to 0's. If the contents of a particular element of V*j* equal 0, then a value of 64_8 is transmitted to the corresponding element of V*i*. If the contents of a particular element of V*j* are negative, then the corresponding element of V*i* is set to 0.

Instruction 175			
Mode	Machine Instruction	CAL Syntax	Description
	1750*j*0	VM Vj,Z	Set VM bit if (V*j* element) = 0.
	1750*j*1	VM Vj,N	Set VM bit if (V*j* element) ≠ 0.
	1750*j*2	VM Vj,P	Set VM bit if (V*j* element) ≥ 0.
	1750*j*3	VM Vj,M	Set VM bit if (V*j* element) < 0.
	175*ij*4	V*i*,VM Vj,Z	Set VM bit if (V*j* element) = 0; also, store the compressed indices of the V*j* elements = 0 in the V*i* elements.
	175*ij*5	V*i*,VM Vj,N	Set VM bit if (V*j* element) ≠ 0; also, store the compressed indices of the V*j* elements ≠ 0 in the V*i* elements.
	175*ij*6	V*i*,VM Vj,P	Set VM bit if (V*j* element) ≥ 0; also, store the compressed indices of the V*j* elements ≥ 0 in the V*i* elements.
	175*ij*7	V*i*,VM Vj,M	Set VM bit if (V*j* element) < 0; also, store the compressed indices of the V*j* elements < 0 in the V*i* elements.

Special Cases

The following special cases apply to instruction 175:

- If V*j* element $n = 0$ and $k = 0$ or 4, then VM bit $n = 1$.
- If V*j* element $n \neq 0$ and $k = 1$ or 5, then VM bit $n = 1$.
- If V*j* element $n \geq 0$ and $k = 2$ or 6, then VM bit $n = 1$.
- If V*j* element $n < 0$ and $k = 3$ or 7, then VM bit $n = 1$.
- If V*j* element $n = 0$ and $k = 4$, then the compressed index stored in V*i* = n.
- If V*j* element $n \neq 0$ and $k = 5$, then the compressed index stored in V*i* = n.
- If V*j* element $n \geq 0$ and $k = 6$, then the compressed index stored in V*i* = n.
- If V*j* element $n < 0$ and $k = 7$, then the compressed index stored in V*i* = n.

Hold Issue Conditions

Instruction 175 holds issue for any of the following conditions:

- The V*j* register is reserved as an operand.
- Instruction 003 was issued during the preceding 2 CPs.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.

Instructions 175*ij*4 through 175*ij*7 hold issue if the V*i* register is reserved as an operand or as a result.

If instruction 14x or 175 is in progress, the vector logical functional unit is busy for (VL)/2 + 4 CPs.

Execution Time

Instruction 175 issues in 1 CP.

The following execution times apply if data is available:

- The V*j* register is ready in (VL)/2 + 3 CPs.
- The VM register is ready for use by all instructions except instruction 073 in (VL)/2 + 4 CPs.
- The VM register is ready for use by instruction 073 in (VL)/2 + 5 CPs.
- The V*i* register is ready for use by instructions 175*ij*4 through 175*ij*7 in (VL)/2 + 9 CPs.

If no test conditions are true for instructions 175*ij*4 through 175*ij*7, then (VM) = 0, no write operations to V*i* registers occur, and the elements of V*i* are unchanged by the instruction.

Description

The full vector logical functional unit executes the vector mask and compressed index instruction 175. Instructions 1750*j*0 through 1750*j*3 create a mask in the vector mask registers VM and VM1. The 64 bits of the VM register correspond to elements 0 through 63 of V*j*, and the 64 bits of the VM1 register correspond to elements 64 through 127 of V*j*. The elements of V*j* are tested for the condition specified by the *k* field of the instruction. If the condition is true for an element, the corresponding bit is set to 1 in the VM register. If the condition is not true, the bit is set to 0.

Instructions 175*ij*4 through 175*ij*7 create a vector mask identical to that of instructions 1750*j*0 through 1750*j*3. However, they also create a compressed index list in register V*i* based on the results of testing the contents of the elements of register V*j*.

The number of elements tested is determined by the contents of the VL register; however, the VM and VM1 registers are cleared before the elements of V*j* are tested. Element 0 corresponds to bit 0, element 1 to bit 1, and so on, from left to right in the register.

The type of test made by the instruction depends on the 2 low-order bits of the *k* designator. The high-order bit of the *k* designator is used to select the compressed index option.

For instruction 1750*j*0, if the contents of the element of V*j* equal 0, the VM bit is set to 1. If the contents of the element of V*j* are not 0, the VM bit is set to 0.

For instruction 1750*j*1, if the contents of the element of V*j* are not 0, the VM bit is set to 1. If the contents of the element of V*j* equal 0, the VM bit is set to 0.

For instruction 1750*j*2, if the contents of the element of V*j* are ≥ 0, the VM bit is set to 1. If the contents of the element of V*j* are negative, the VM bit is set to 0.

For instruction 1750*j*3, if the contents of the element of V*j* are negative, the VM bit is set to 1. If the contents of the element of V*j* are ≥ 0, the VM bit is set to 0.

Instructions 175*ij*4 through 175*ij*7 are compressed index instructions. These instructions test for zero, nonzero, positive (≥ 0), and negative elements, respectively. A vector mask is generated, along with a vector containing the indices of those elements of the tested vector that satisfied the tested condition. These stored indices are referred to as compressed indices, because the element pointer for the V*i* register is advanced only when the tested condition is satisfied and an index needs to be stored, resulting generally in a V*i* register of shorter vector length than that of the register tested.

For instruction 175*ij*4, if the contents of the element of V*j* equal 0, the VM bit is set to 1, and the compressed element of V*i* is set to the index of the element of V*j*. If the contents of the element of V*j* are not 0, the VM bit is set to 0, the element pointer for the V*i* register does not advance, and nothing is written to V*i*. Refer to Figure 7-15 for an example of instruction 175*ij*4.

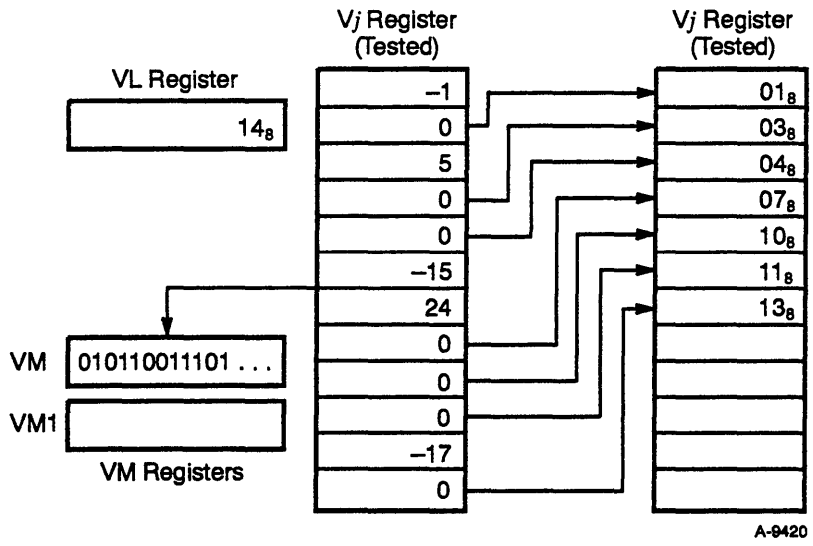


Figure 7-15. Compressed Index Example for Instruction 175*ij*4

For instruction 175*ij*5, if the contents of the element of V*j* are not 0, the VM bit is set to 1, and the compressed element of V*i* is set to the index of the element of V*j*. If the contents of the element of V*j* equal 0, the VM bit is set to 0, the element pointer for the V*i* register does not advance, and nothing is written to V*i*.

For instruction 175*ij*6, if the contents of the element of V*j* are ≥ 0, the VM bit is set to 1, and the compressed element of V*i* is set to the index of the element of V*j*. If the contents of the element of V*j* are negative, the VM bit is set to 0, the element pointer for the V*i* register does not advance, and nothing is written to V*i*.

For instruction 175*ij*7, if the contents of the element of V*j* are negative, the VM bit is set to 1, and the compressed element of V*i* is set to the index of the element of V*j*. If the contents of the element of V*j* are ≥ 0, the VM bit is set to 0, the element pointer for the V*i* register does not advance, and nothing is written to V*i*.

The number of elements tested is determined by the contents of the VL register. The VM register bits corresponding to the untested elements of the V*j* register are cleared.

The vector mask instructions, 175*ij*0 through 175*ij*3, and the compressed index instructions, 175*ij*4 through 175*ij*7, are vector counterparts to the scalar conditional branch instructions.

Instructions 176 and 177			
Mode	Machine Instruction	CAL Syntax	Description
	176*i*0*k*	V*i* ,A0,A*k*	Read (VL) words from memory to V*i* starting at address (A0) + (DBA), incrementing by (A*k*).
	176*i*00	V*i* ,A0,1 §	Read (VL) words from memory to V*i* starting at address (A0) + (DBA), incrementing by 1.
	176*i*1*k*	V*i* ,A0,V*k*	Read (VL) words from memory to V*i* using memory addresses ((A0) + (V*k*) + (DBA)).
	1770*jk*	,A0,A*k* V*j*	Write (VL) words from (V*j*) to memory starting at address (A0) + (DBA), incrementing by (A*k*).
	1770*j*0	,A0,1 V*j* §	Write (VL) words from (V*j*) to memory starting at address (A0) + (DBA), incrementing by 1.
	1771*jk*	,A0,V*k* V*j*	Write (VL) words from (V*j*) to memory using memory addresses ((A0) + (V*k*) + (DBA)).

Special Cases

An instruction in CIP holds issue for 1 CP after instruction 176 or instruction 177 issues.

Instruction 176 uses port B, if available. If port B is busy at issue time, instruction 176 uses port A. Instruction 177 uses port C.

For instructions 176*i*0*k* and 1770*jk*, if *k*=0, the memory increment is 1 .

For instructions 176*i*0*k* and 1770*jk*, (A*k*) determines the memory increment. Successive addresses are located in successive subsections. References to the same subsection can be made every 7 CPs or more. Incrementing (A*k*) by 64 puts successive memory references in the same subsection, so a word is transferred at least every 7 CPs. If the address is incremented by 32, every other reference is to the same subsection, and words can transfer no faster than two per 7 CPs. If an addressing increment allows 7 CPs to pass before addressing the same subsection, two words can transfer each CP.

Memory conflicts slow reading or writing of individual vector elements. The elements are read or written in order, so a delay for any element delays all succeeding elements.

For instruction 176, if there is an instruction using its destination register as a source of operands, the execution of that instruction is delayed whenever there is a delay in the result data for instruction 176.

Hold Issue Conditions

Instructions 176 and 177 hold issue for any of the following conditions:

- The A0 register is reserved.
- Instruction 0020xx was issued in the preceding CP.
- Instruction 076 or 077 was issued during the previous 3 CPs.
- A scalar reference occurred in CP 1, CP 2, CP 3, CP 4, or CP 5.
- The status register is busy.

Instruction 176 holds issue for any of the following conditions:

- Ports A and B are busy.
- The V*i* register is reserved as an operand or as a result.
- The program is not in bidirectional memory mode and port C is busy.

Instruction 177 holds issue for any of the following conditions:

- Port C is busy.
- The V*i* register is reserved as an operand.
- The program is not in bidirectional memory mode and ports A and B are busy.

Instructions 176*i*0k and 1770*j*k hold issue if A*k* is reserved, where *k*=1 through 7.

Instructions 176*i*1*k* and 1771*jk* hold issue for any of the following conditions:

- Another instruction 176*i*1*k* or 1771*jk* is in progress.
- The V*k* register is reserved as an operand or the A*k* register is reserved, where *k* equals 1 through 7.

Execution Time

The following execution times apply to instruction 176*i*0*k*:

- The instruction issues in 1 CP.
- The V*i* register is ready in (VL)/2 + 26 CPs, if memory is available.
- Port A or port B is busy for (VL)/2 + 6 CPs.

The following execution times apply to instruction 1770*jk*:

- The instruction issues in 1 CP.
- The V*j* register is ready in (VL)/2 + 3 CPs, if data is available.
- Port C is busy for (VL)/2 + 6 CPs.

The following execution times apply to instruction 176*i*1*k*:

- The instruction issues in 1 CP.
- The V*i* register is ready in (VL)/2 + 30 CPs, if memory is available.
- The V*k* register is ready in (VL)/2 + 3 CPs, if data is available.
- Port A or port B is busy for (VL)/2 + 10 CPs.
- Instruction 176*i*1*k* is busy (VL)/2 + 11 CPs.

The following execution times apply to instruction 1771*jk*:

- The instruction issues in 1 CP.
- The V*i* and V*k* registers are ready in (VL)/2 + 3 CPs, if data is available.
- Port C is busy for (VL)/2 + 10 CPs.
- Instruction 1771*jk* is busy for (VL)/2 + 11 CPs.

Description

Instructions 176 and 177 transfer blocks of data between V registers and memory. Instruction 176 reads data from memory to elements of register V*i*. Instruction 177 writes data from elements of register V*j* to memory. The number of elements transferred is determined by the contents of the VL register.

Instructions 176*i*0*k* and 176*i*00 read words directly from memory and enter them into the elements of the V*i* register. A0 contains the starting 32-bit memory address. This address is incremented by the contents of the A*k* register for each word transmitted. The contents of A*k* can be

positive or negative, allowing both forward and backward streams of references. If the *k* designator is 0, or if 1 replaces A*k* in the operand field of the instruction, the address is incremented by 1.

Instruction 176*i*1*k* gathers words from nonsequential memory locations and enters them into sequential elements of the V*i* register. The V*k* and A0 registers generate the nonsequential memory addresses. The low-order bits of each element of V*k* contain a signed integer, which is added to the contents of A0 to obtain the 32-bit memory address. Figure 7-16 shows an example of the 176*i*1*k* instruction.

In Figure 7-16, the VL register is set to 4, resulting in a transfer of 4 elements. Instruction 176*i*1*k* adds the contents of A0 to the contents of each element of register V*k* to form a memory address. The contents of that address are then read and entered into the V*i* register. Because (A0) = 100 and (element 0 of V*k*) =4, the contents of address 104 are entered into element 0 of V*i*. Similarly, (A0) + (element 1 of V*k*) = 102, and the contents of memory location 102 are entered into element 1 of V*i*. This process continues until the number of elements transferred equals the (VL).

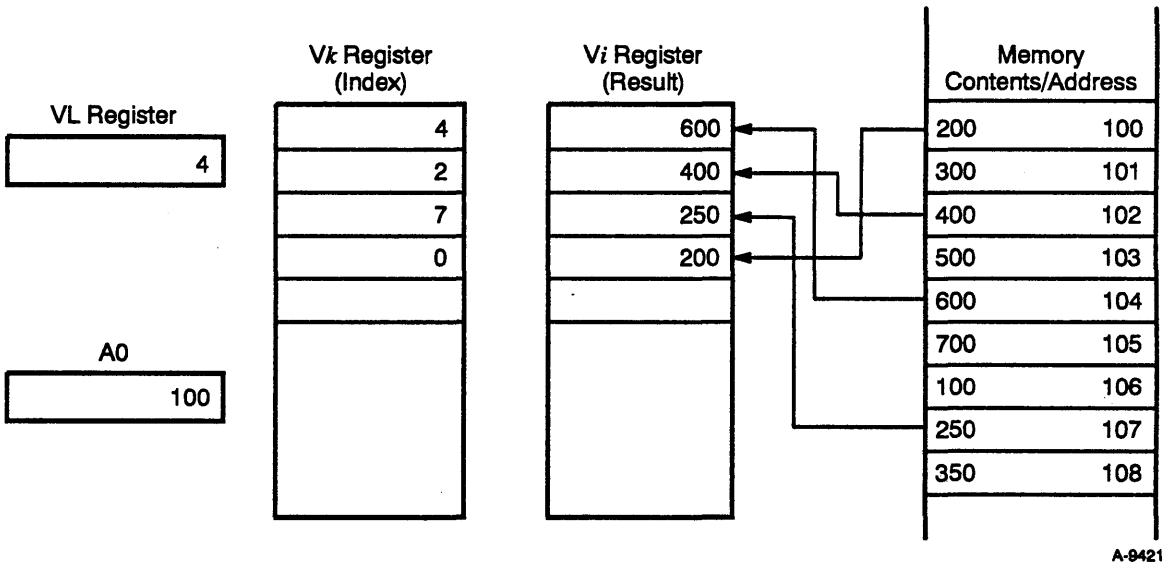


Figure 7-16. Gather Instruction Example

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

Instruction 1771*jk* writes words from the elements of the V*j* register to nonsequential memory locations. The V*k* and A0 registers generate the nonsequential memory addresses. The low-order bits of each element of V*k* contain a signed integer, which is added to the contents of A0 to generate a 32-bit memory address. Figure 7-17 shows an example of the 1771*jk* instruction.

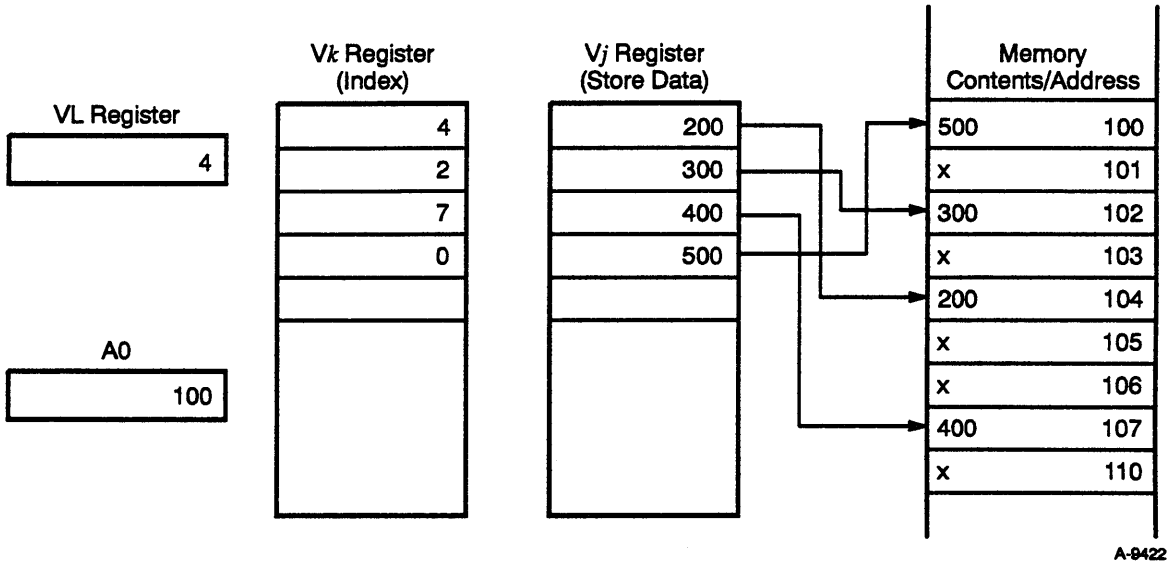


Figure 7-17. Scatter Instruction Example

In Figure 7-17, the VL register is set to 4, resulting in a transfer of 4 elements. Instruction 1771*jk* adds the contents of A0 to the contents of each element of register V*k* to generate a memory address. The contents of an element of V*j* are then entered into the resulting memory address. Because (A0) = 100 and (element 0 of V*k*) = 4, the contents of element 0 of V*j* are entered into memory address 104. Similarly, (A0) + (element 1 of V*k*) = 102, and the contents of element 1 of V*j* are entered into memory location 102. This process continues until the number of elements transferred equals the (VL).

BIBLIOGRAPHY

IOS Model E System Programmer Reference Manual, CRI publication number CSM-1010-000.

This manual describes the architecture and functions of the Cray Research input/output subsystem model E (IOS-E). Detailed information on the I/O processors, I/O buffers, I/O channels, channel adapters, cluster and workstation interfaces, and the I/O instruction set is provided.

IOS Model E System Programmer Reference Manual Change Packet, CRI publication number CSM-1010-001.

This change packet adds information about a function that controls the Programmed Interrupt signals from the IOS to the mainframe.

SSD Solid-state Storage Device System Programmer Reference Manual, CRI publication number CSM-1116-000.

This manual provides detailed information on the SSD product line and its operation with the CRAY Y-MP, CRAY X-MP, and CRAY-1 computer systems.

60 Series Disk Systems Guide, CRI publication number COM-1124-000.

This manual contains information on the DD-60 and DD-61 disk drives, the DE-60 disk enclosure, and the DCA-2 disk channel adapter. Information on hardware, basic theory of operations, and flaw management utilities is also included.

CRAY Y-MP C90 Site Planning Reference Manual, CRI publication number HR-04025.

This manual describes the physical requirements for the CRAY Y-MP C90 computer system. It defines customer and Cray Research, Inc. site planning and preparation responsibilities. This manual also describes the operational requirements, system configurations, mainframe and cooling units specifications and requirements, and computer room floor specifications.

CAL Assembler Version 2 Reference Manual, CRI publication number SR-2003.

This manual describes the CAL assembler.

Symbolic Machine Instructions Reference Manual, CRI publication number SR-0085.

This manual describes the machine instructions used on CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 systems.

CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 CAL Assembler Version 2 Ready Reference, CRI publication number SQ-0083.

This publication is a quick reference booklet that contains both machine instructions and CAL information.

CRAY Y-MP, CRAY X-MP EA, and CRAY X-MP Multitasking Programmer's Manual, CRI publication number SR-0222.

This manual describes multitasking features and concepts, including microtasking. It explains how to partition an executable program among multiple processors on CRAY Y-MP and CRAY X-MP computer systems running either COS or UNICOS.

CF77 Compiling System, Volume 4: Parallel Processing Guide, CRI publication number SG-3074.

This user guide defines and describes the Autotasking feature of the CF77 compiling system release 5.0. Autotasking is the automatic distribution of loop iterations to multiple processors. This user guide is one manual in a set describing the CF77 compiling system.

MWS-E User Guide, CRI publication number CDM-1123-0A0.

This user guide provides procedures and supporting descriptions used in typical day-to-day operations of the MWS-E. Sections describe system startup and shutdown, the mws maintenance login environment, the online diagnostic directory structure, backup and restore procedures, system security, WACS programs and procedures, the error logging system, and the system clear utility.

OWS-E Operator Workstation Operator's Guide, CRI publication number SG-3078.

This user guide provides an overview of the hardware environment, the directory structure, and OpenWindows. It tells you how to boot and dump the IOS-E and the mainframe, how to shut down the OWS-E, how to perform backups of the OWS-E software system, how to use the CPU and network monitors, and how to respond to messages. It is to be used in conjunction with documentation provided by Sun Microsystems, Inc.

MWS-E and OWS-E Hardware Maintenance Manual, CRI publication number CMM-1122-0A0.

This manual describes Sun-based MWS-E and OWS-E workstations that are part of CRAY Y-MP systems shipped with an IOS-E. This manual provides information needed to install, configure, test, maintain, and repair the MWS-E and OWS-E workstations.

INDEX

Boldface numbers refer to illustrations and charts. Individual instructions are listed in numeric order in the table of contents and are not duplicated here.

Reader Comment Form

Title: CRAY Y-MP C90 System Programmer Reference Manual **Number: CSM-0500-000**

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