2 PROCESSING UNIT

The CRAY EL series computer system central processing unit, specifically referred to as a processor, contains all the computation and control registers and functional units included in a supercomputer system. This miniaturization is made possible by very large-scale integration (VLSI) complimentary metal oxide semiconductor (CMOS) application-specific integrated circuits (ASICs). Twenty-three of these ASICs are mounted on the single 16 x 22 x .093 in. PC board that forms the primary processor and 16 are mounted on the 8.2 x 20.5 in. secondary processor PC board.

The CMOS ASICs are either 1-micron, 2-layer devices with 100,000 undefined gates or 0.5-micron, 2-layer devices with 200,000 undefined gates, each measuring 2.08 x 2.08 inches. The power consumed by these chips averages less than 5 watts per ASIC, operating at + 5 volts.

The PC boards used to form the processor module are made of 18 separate layers:

- 1 top pad
- 1 bottom pad
- 4 ground (Vss) layers
- 4 power (Vdd) layers
- 8 signal layers

This module uses an average of 60 watts of power per processor. Therefore, the maximum of eight processor modules in a system consumes 480 watts (8 processors x 60 watts = 480 watts). The processor module, like the entire CRAY EL series system, is an air-cooled device.

Support for multiple processors in the mainframe includes:

- Shared memory
- Shared registers
- Shared I/O channels
- Deadlock detection
- Shared deadstart paths

However, the CRAY EL series system does not currently support:

- Performance monitors

- High-speed (HISP) channels
- Very high-speed (VHISP) channels

Some instructions are discussed in this section, with several specific examples. Refer to the Appendix for a list of the instructions that are valid for the CRAY EL series system.

ASIC Descriptions

The following nine types of ASIC chips reside on the processor modules:

- One arbiter ASIC (AR) controls memory access and arbitrates all memory conflicts. It also provides interrupting processor synchronization. This ASIC is found only on a primary processor.

- One address and scalar ASIC (AS) contains all of the address registers and the address functional units (FUs), as well as the scalar (S) registers and scalar FUs.

- Two channel control ASICs (CCs) control or support the functions of two Y1 channels. Each of the Y1 channels is capable of 40-Mbyte/s transfers and each connects to a VME subsystem. The CC chip also contains the console bus interface, which provides console support. This function is implemented only on CC0. On the Revision 04 version of the CC ASIC, HIPPI channel support is also provided. These ASICs are found only on a primary processor.

- Eight data switch ASICs (DSs) steer data between memory, the selected channel, and the required FUs. The DSs also contain the vector registers and some selected exchange registers.

- Four execution unit ASICs (EUs) contain all of the vector and floating-point FUs, with the exception of the floating-point reciprocal FU. This format provides full pipelining to the EU chips and allows each to work on a different problem independently

  The EU pipeline varies from a standard pipeline. When the EU is used as a series pipeline, the results of the computational operation must be returned to the DS ASIC before it can be used as an operand in a continuing computation.

  When the EUs are used in parallel pipelines, it is possible to use all four simultaneously. In this instance, the results of the individual computations can be used in a chaining operation as continuation operands. The restrictions that pertain to the EU chips are:

  - Only one FU in each EU chip can be operating at any one time

  - Only the EU3 ASIC is capable of executing vector mask (VM) instructions (146,147, and 175 instructions)

- One memory control ASIC (MC) provides address generation, which can support up to 512 Mwords of memory. The MC chip also provides both operand and program range error detection.

- Four memory data ASICs (MDs) perform single-error correction, double-error detection (SECDED) functions, including check-bit generation on the read and write memory data. Separate data paths, internally from the primary processor DSs and externally from the secondary processor DSs, are muxed together for each of the four memory sections. A single data path is then connected to each memory section on the memory modules. The MDs are also used to support all of the memory maintenance instructions. These ASICs are found only on a primary processor.

- One processor control ASIC (PC) contains the processor instruction buffers. The CRAY EL series system processor uses eight instruction buffers, each of which is 32 words wide. Instruction issue control and I/O interrupt handling control also reside on the PC chip. Part of the issue control function is a resource scoreboard, which also resides on the PC. Shared register access control is also performed on the PC ASIC (the CRAY EL series system supports seven shared register clusters).

- One reciprocal and control ASIC (RC) contains the floating-point reciprocal FU and the console control registers. The RC on the primary processor also implements the scan and clock control circuits for the combined processor module.

These units interconnect in the way shown on the CRAY EL series system block diagram, Figure 2-2. The signal paths are shown in more detail in Figure 2-2 and Figure 2-3, block diagrams of the processor bus. These diagrams also show the internal contents of each of the ASICs located on the processor module. Figure 2-4 and Figure 2-5 show the locations of the various ASICs on the processor modules.

Processor Module Description/Differences

The CRAY Y-MP EL system supports one to four processors, and the CRAY EL98 system supports one to eight processors. Each processor module can support up to four VME subsystems. The processor module is completely self contained and plugs into the mainframe backplane using 11 connectors that provide 1,230 signal pins. The secondary processor module shares scan, clock control, and memory access with the primary processor module. The secondary processor module connects to the primary processor module using 9 connectors that provide 558 signal pins.

An addition found only on the CRAY EL98 system is the removable memory arbitration bus (MAB) PC board. The MAB is mounted on the top of the mainframe card cage and provides a shared arbitration path between the processor modules. The MAB shares five control lines, seven address lines and four port x vector write data valid lines, which are routed to the MC ASICs on the secondary processor modules. A second set of connectors provides a path for the same signals to the AR ASICs on the primary processors. Signal pin restrictions on the normal backplane made it necessary to add this MAB to the mainframe.

When a processor is indicated as the faulty unit in an incident, the only on-site repair performed is the replacement of the processor module.

A potential problem exists because of the physical placement of the MAB. The MAB must be removed prior to removing any of the processor modules from the card cage or the MAB, the processor module, or both will be damaged. This requires extending the mainframe card cage every time a processor module is to be removed from the mainframe.

The processor module connects to the mainframe memory via four 72-bit bidirectional ports. Each of these ports connects to a separate memory section. Each of the processor modules in a CRAY EL series system contains a copy of all memory and shared register reservations in order to reduce the possibility of conflicts. Each processor has to check its local request registers, not the request registers of all of the processors.

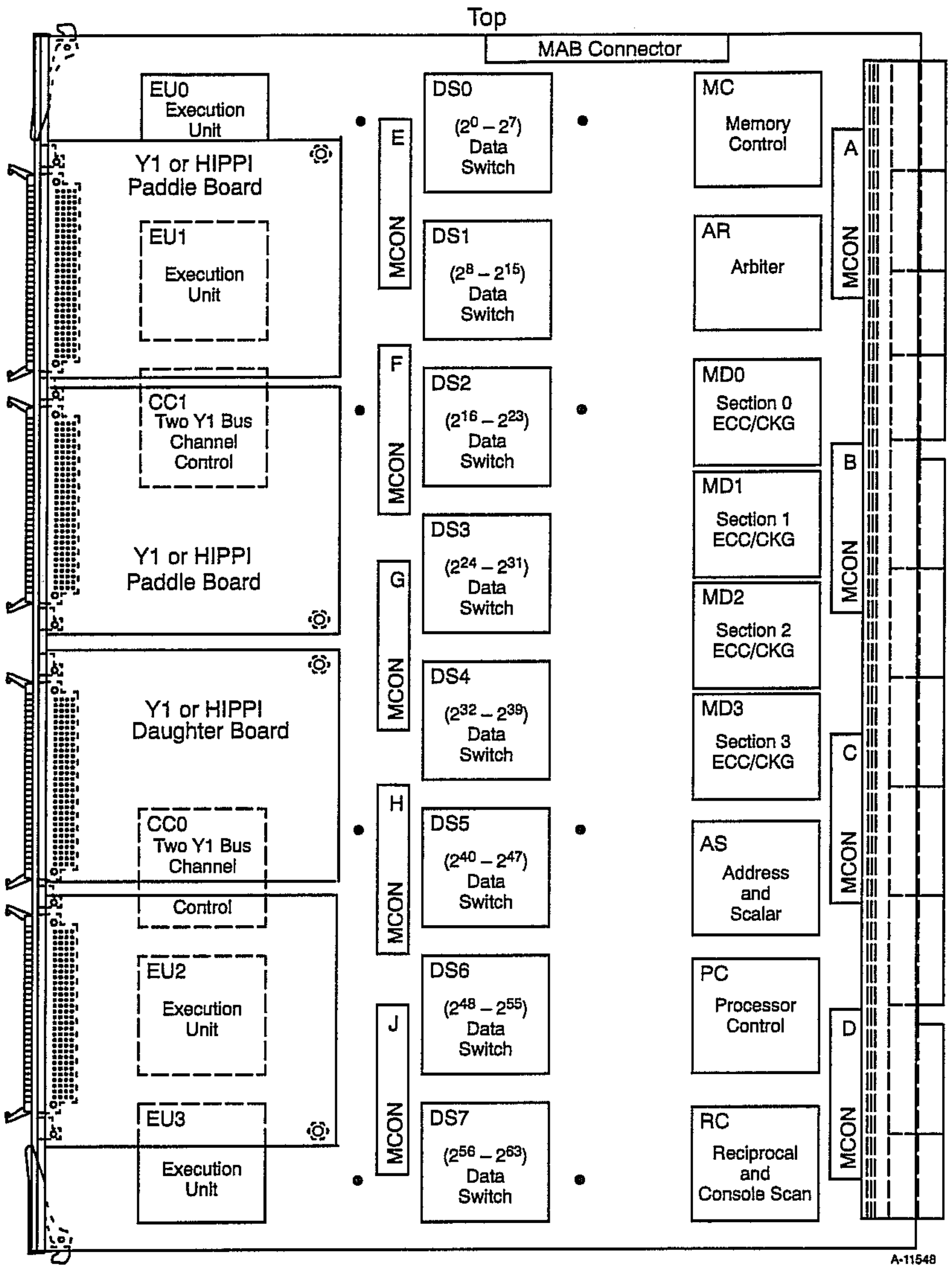

Top
MAB Connector
EU0 Execution Unit
Y1 or HIPPI Paddle Board
EU1 Execution Unit
CC1 Two Y1 Bus Channel Control
Y1 or HIPPI Paddle Board
Y1 or HIPPI Daughter Board
CC0 Two Y1 Bus Channel Control
EU2 Execution Unit
EU3 Execution Unit
E MCON
F MCON
G MCON
H MCON
J MCON
DS0 ($2^0 - 2^7$) Data Switch
DS1 ($2^8 - 2^{15}$) Data Switch
DS2 ($2^{16} - 2^{23}$) Data Switch
DS3 ($2^{24} - 2^{31}$) Data Switch
DS4 ($2^{32} - 2^{39}$) Data Switch
DS5 ($2^{40} - 2^{47}$) Data Switch
DS6 ($2^{48} - 2^{55}$) Data Switch
DS7 ($2^{56} - 2^{63}$) Data Switch
MC Memory Control
AR Arbiter
MD0 Section 0 ECC/CKG
MD1 Section 1 ECC/CKG
MD2 Section 2 ECC/CKG
MD3 Section 3 ECC/CKG
AS Address and Scalar
PC Processor Control
RC Reciprocal and Console Scan
A MCON
B MCON
C MCON
D MCON
A-11548

Figure 2-4. Primary Processor Module

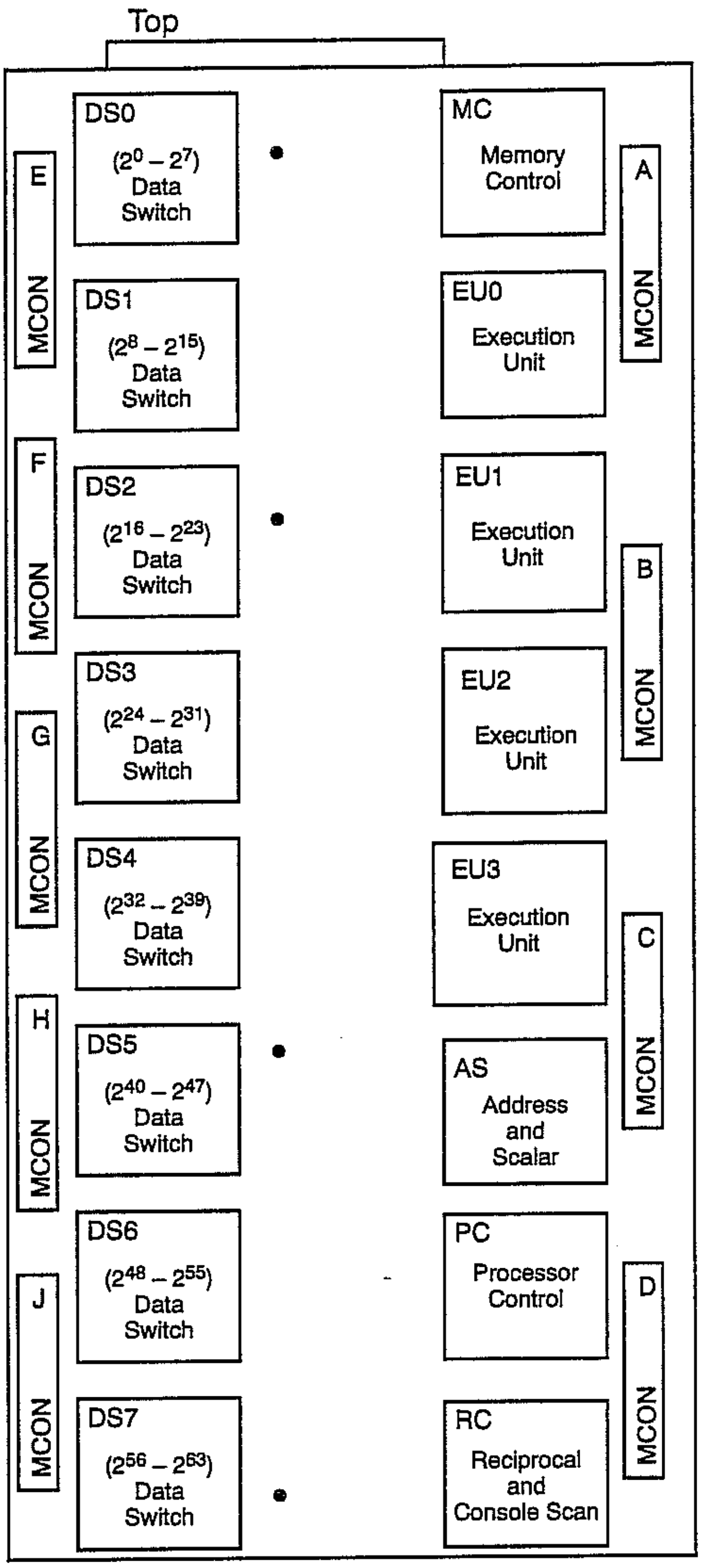

Top
DS0 ($2^0 - 2^7$) Data Switch
DS1 ($2^8 - 2^{15}$) Data Switch
DS2 ($2^{16} - 2^{23}$) Data Switch
DS3 ($2^{24} - 2^{31}$) Data Switch
DS4 ($2^{32} - 2^{39}$) Data Switch
DS5 ($2^{40} - 2^{47}$) Data Switch
DS6 ($2^{48} - 2^{55}$) Data Switch
DS7 ($2^{56} - 2^{63}$) Data Switch
MC Memory Control
EU0 Execution Unit
EU1 Execution Unit
EU2 Execution Unit
EU3 Execution Unit
AS Address and Scalar
PC Processor Control
RC Reciprocal and Console Scan
E MCON
F MCON
G MCON
H MCON
J MCON
A MCON
B MCON
C MCON
D MCON
A-11549

Figure 2-5. Secondary Processor Module

Processor Control

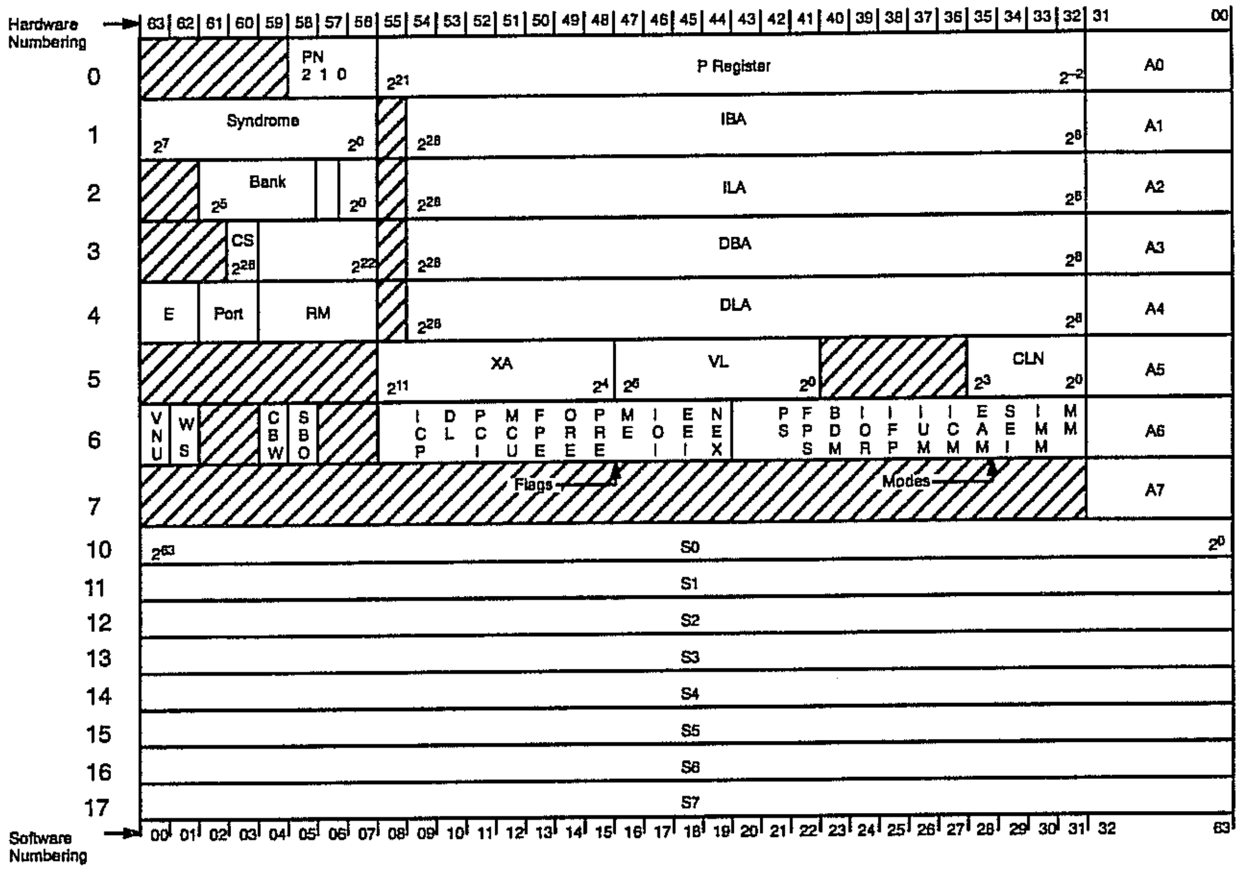
To run a program within the CRAY EL series system, you must load the program into the selected processor and then execute the program functions. This process includes using an exchange sequence, a fetch sequence, and an issue sequence. The exchange sequence brings the program parameters into the appropriate registers within the selected processor. A fetch sequence immediately follows the exchange sequence which transfers a block of instructions from memory into the processor's instruction buffers. After the instructions are loaded into the instruction buffers, the issue sequence begins executing the instructions one at a time.

The instruction to be executed is indicated by a program address (P) register. This P register points to the location in the instruction buffer containing the desired instruction, which causes that instruction to be decoded and then executed. When the selected instruction is executed, the P register increments, causing a new instruction to begin the decode process. This function continues until a desired instruction cannot be found in the instruction buffers. When the instruction cannot be found, another fetch sequence is initiated, causing another block of instructions to be loaded into the instruction buffers. This process continues until the program terminates; then, another exchange sequence occurs, which allows a new program to use the processor.

Figure 2-6 represents the exchange package used in the CRAY EL series system and is useful for troubleshooting. The exchange package may contain pertinent information about the state of the processor at the point of failure. To extract this information, compare the condition of the flag bits, mode bits, and the error word bits with the respective charts shown below the exchange package in Figure 2-6. In some cases, useful information can also be found in the A registers or the S registers, which are also shown in Figure 2-6. The usefulness of the A or S register information depends on the particular diagnostic used.

The Exchange Mechanism

Each processor uses an exchange mechanism for introducing a program into execution or for switching from one program to another. This exchange mechanism depends on a block of program parameters, called an exchange package, that contains the required addresses and limits imposed on the program. Another component of the exchange mechanism is the exchange sequence, which is a basic processor operation used to load the exchange package.



Word 6		Flag Bits	Cause Exchange
55		Reserved	
54	ICP	Interrupt from internal processor	MM'
53	DL	Deadlock Interrupt	MM' and IMM
52	PCI	Programmable Clock Interrupt (staged)	MM'
51	MCU	MCU Interrupt	MM'
50	FPE	Floating-point Error Interrupt	MM and IMM
49	ORE	Operand Range Error Interrupt	MM' and IMM
48	PRE	Program Range Error Interrupt	MM' and IMM
47	ME	Memory Error Interrupt	Always
46	IOI	I/O Interrupt (staged)	MM'
45	EEI	Error Exit Interrupt	Always
44	NEX	Normal Exit Interrupt	Always

Word 6		Mode Bits
63	VNU	Vectors Not Used
62	WS	Waiting on Semaphores
59	CBW	Concurrent Block Write (CRAY EL Series System only)
58	SBO	Scalar and Block Overlap (CRAY EL Series System only)
43		Reserved
42	PS	Program State
41	FPS	Floating-point Status
40	BDM	Bidirectional Memory
39	IOR	Interrupt Operand Range Error
38	IFP	Interrupt Floating-point Error
37	IUM	Interrupt Uncorrectable Memory
36	ICM	Interrupt Correctable Memory
35	EAM	Extended Address Mode (32-bit)
34	SEI	Selected External Interrupt
33	IMM	Interrupt Monitor Mode
32	MM	Monitor Mode

A-9829

Figure 2-6. Exchange Package

The Exchange Package

The exchange package is a block of memory that contains the basic parameters for a particular program. This block of memory is 16 words long and is used to provide continuity when a program stops and restarts from one section of the program to the next, or when a program terminates and a new program is introduced.

The contents of the exchange package include the eight address (A) registers, eight scalar (S) registers, and the contents of specific parameter registers. A diagram of the exchange package is shown in Figure 2-6. Refer to Table 2-1 for a description of the acronyms and the actual bits used to represent these parameters. The following subsections provide detailed descriptions of the fields.

Table 2-1. Exchange Package Field Descriptions

Field	Word	Bits	Description
PN	0	$2^{56} - 2^{58}$	Processor number
P	0	$2^{32} - 2^{55}$	Program address register
S	1	$2^{56} - 2^{63}$	Syndrome bits
IBA	1	$2^{32} - 2^{54}$	Instruction base address
BANK	2	$2^{56} - 2^{61}$	Read address bank
ILA	2	$2^{32} - 2^{54}$	Instruction limit address
CS	3	$2^{56} - 2^{62}$	Read address chip select (bit mask)
DBA	3	$2^{32} - 2^{54}$	Data base address
E	4	$2^{62} - 2^{63}$	Read error type
PORT	4	$2^{60} - 2^{61}$	Port used
RM	4	$2^{56} - 2^{59}$	Read mode
DLA	4	$2^{32} - 2^{54}$	Data limit address
XA	5	$2^{48} - 2^{55}$	Exchange address register
VL	5	$2^{41} - 2^{47}$	Vector length register
CLN	5	$2^{32} - 2^{35}$	Cluster number
VNU	6	2^{63}	Vector not used
WS	6	2^{62}	Waiting for semaphore
CBW	6	2^{59}	Concurrent block write
SBO	6	2^{58}	Scalar block overlap

Table 2-1. Exchange Package Field Descriptions (continued)

Field	Word	Bits	Description
FLAGS	6	$2^{44} - 2^{55}$	Flag register
MODES	6	$2^{32} - 2^{43}$	Mode register
A	0 – 7	$2^{0} - 2^{31}$	Address registers
S	10 – 17	$2^{0} - 2^{63}$	Scalar registers

Processor Number Field

The processor number (PN) field indicates which processor performed the exchange sequence. The value in the PN field is inserted into the exchange package from the backplane. Logical slot 0 contains primary processor 0 and secondary processor 4 in the mainframe, logical slot 1 contains primary processor 1 and secondary processor 5, etc. Refer to Figure 3-3 for mainframe chassis configuration.

Program Address Register Field

The program address (P) register contents are stored in the P field of the exchange package. The instruction located at this memory address is the first instruction issued when the program enters the execution phase.

Syndrome Field

The 8-bit syndrome field specifies the syndrome code generated from the SECDED circuits if an error occurs on a memory read operation.

Read Address Bank Field

If an error is detected during a memory read operation, the bank number of the error is retained in the 6-bit read address bank field.

Read Address Chip Select Field

The read address chip select (CS) field identifies the chip in which a memory read error occurs. The CS bit is bit 2^{28} of the memory read address.

Read Error Type Field

The 2-bit read error (E) type field is used to determine the type of memory or I/O error detected. Bit 2^{63} is set if the error is uncorrectable, and bit 2^{62} is set if the error is correctable.

Port Field

The 2-bit port field defines the port where a memory read error or an I/O error occurs. These bits are used with the read mode bits to identify which operation was in progress when the error took place. Refer to Table 2-2 for a translation of these fields.

Table 2-2. Port and Read Mode Field Translation

P Port								RM Read Mode				Port Usage
2^{61}	2^{60}	2^{61}	2^{60}	2^{61}	2^{60}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}	
A 0	0	B 0	1	C 1	0	D 1	1	0	0	0	1	Exchange
0	0	0	1	1	0	–	–	0	0	1	0	A or S
–	–	–	–	–	–	D x	x*	0	1	0	1	I/O single
–	–	–	–	–	–	D x	x*	0	1	1	1	I/O block
0	0	0	1	1	0	–	–	1	0	0	0	B or T
–	–	–	–	–	–	D x	x*	1	0	1	1	Fetch
0	0	0	1	1	0	–	–	1	1	0	0	Vector stride
0	0	0	1	1	0	–	–	1	1	1	0	Vector gather/scatter

NOTE: The x designates don't care.

Read Mode Field

The read mode (RM) field bits are used with the port field bits to determine the read operation that was in progress when a read error occurred. Refer to Table 2-2 for a translation of these bits.

The fields described previously (syndrome, bank, CS, E, port, and RM) are called memory error data fields. These fields are valid only in the exchange package if one of two conditions are met. The first condition is that the interrupt on correctable memory (ICM) bit must be set in the mode register and a correctable memory error must be detected. This condition causes an exchange to occur. The second condition that validates the memory error data fields is the interrupt on uncorrectable memory (IUM). This bit must be set in the mode register and an uncorrectable memory error must be detected, which causes an exchange to take place.

In these two instances, the exchange package contains valid data that can be useful in the repair of the CRAY EL series system, both at the field-replaceable unit (FRU) level and at the component level. When possible, the contents of the exchange package memory error data fields should accompany a memory module when it is returned to Central Repair for evaluation and component replacement. This information can be in the form of a screen snap when a printer is available, or handwritten when no printer is available.

Data Base Address Register Field

The data base address (DBA) register field is used to hold the base, or first address, of the user's data area. The base address is used to determine the location in memory of a program's data. Each time an instruction from the program makes a memory reference, the memory address generated by the instruction is added to the DBA to create the absolute memory address.

In the CRAY EL series system, the DBA is bit 2^{54} through bit 2^{32}, with bit 2^{32} through bit 2^{39} equal to 0's. This condition causes the contents of this register to always be a multiple of 400_8.

Data Limit Address Register Field

The data limit address (DLA) register field holds the highest address of the user's data area and is used to determine the highest absolute memory address a program can use for data. Each time an instruction makes a memory reference, the absolute memory address that is generated is compared to the address held in the DLA. If the absolute memory address is equal to or greater than the DLA value, an out-of-range condition occurs. If the absolute memory address is less than the DLA, the program is allowed to proceed.

If the out-of-range condition occurs during a memory read reference, the reference is allowed to issue and complete, but a zero value is transferred from memory. A memory write reference that exceeds the DLA is allowed to issue, but no memory write operation occurs.

Instruction Base Address Register Field

The instruction base address (IBA) register field holds the user's base address. This base address is used to determine where a program's instruction area is located in memory. When an instruction fetch sequence occurs, an absolute memory address is created by adding the relative address generated by the fetch control logic (the upper 22 bits of the P register) to the contents of the IBA register.

Instruction Limit Address Register Field

The instruction limit address (ILA) register field holds the limit address that has been defined as the user's instruction area. The ILA is used to determine the maximum absolute memory address that can be accessed during an instruction fetch sequence.

If the absolute memory address is not within the range of the addresses contained in the IBA and the ILA, the processor generates a program range error interrupt.

Exchange Address Register Field

The exchange address (XA) register field specifies the address of the first word of a 16-word exchange package. The XA register contains the upper 8 bits of a 12-bit area that is used to specify the absolute memory address. The lower 4 bits of this area are forced to 0's, causing the conditional requirement for the exchange package to always start on a 16-word boundary. Likewise, the 12-bit limit restricts the exchange package to the lower 10000_8 words of memory.

Vector Length Register Field

The vector length (VL) register field is 7 bits wide and is used to define the length of all vector operations. The length of a vector operation indicates the number of vector elements that are allowed to be used by a vector instruction. The value in the VL register can be changed during program execution by using the 00200*k* instruction. When the VL register content is equal to 0, the actual number of vector elements used is 100_8 because the VL register is a circular countdown register.

Cluster Number Register Field

The cluster number (CLN) register field determines which cluster of shared registers the processor uses when executing the program. The number of shared registers available is equal to the number of processors plus one. These clusters are composed of shared B (SB) registers, shared

T (ST) registers, and semaphore (SM) registers. The exchange package contains a number between 1 and 7 in the CLN register field, which determines which cluster of shared registers that processor can access. To prevent a processor from accessing any shared registers, a 0 is inserted into the CLN field.

Vector Not Used Field

The vector not used (VNU) field reflects the condition of use of the vector instructions 077*ijk* and 140*ijk* through 170*ijk*. If none of these instructions are set in the previous execution interval (that time when the exchange package was active), then the VNU bit is set. Any time one of the vector instructions is used during the execution interval, VNU is equal to 0.

Waiting for Semaphore Field

When the waiting for semaphore bit is set in the exchange package, it indicates that an exchange occurred while a test and set instruction (0034*jk*) was holding issue in the next instruction parcel (NIP) register area.

Concurrent Block Writes Bit

When the concurrent block writes (CBW) bit is set in the exchange package, the CRAY EL series system can have more than one write port active at one time. The CBW bit is specific to the CRAY EL series system, which is the first Cray Research mainframe that allows more than one write port to memory. When executed, the 002604 instruction enables the CBW; the 002504 instruction disables CBW.

Scalar Block Overlap Bit

The scalar block overlap (SBO) bit is another exchange package bit that enables a CRAY EL series system-specific function. The function of the SBO is to allow scalar memory references to intermix with memory block references. The 002606 instruction enables the SBO, and the 002506 instruction disables the SBO.

The remainder of the fields in the exchange package are definitions of the interrupt flags that are set in the exchange package when an exchange sequence is initiated by one of the interrupts and the mode bits that are user selectable to define program execution. The contents of the A registers and the S registers are also shown in the exchange package.

Exchange

The purpose of the exchange mechanism in the CRAY EL series system is to provide a means of switching program execution between different programs.

The exchange sequence provides a process by which the currently executing program is gracefully deactivated, with its current operating parameters placed in memory for later retrieval. The next step in the process retrieves the operating parameters of the new program from a specified memory location and places these parameters into the processor operating registers. The processor operating registers that are used to contain operating parameters are referred to as the exchange registers, which are spread across several ASICs on the processor module.

A limited number of situations can cause an exchange to occur:

- Deadstart
- Normal program exit
- Error program exit
- Interrupt

The deadstart sequence starts a program after a power-off/power-on operation or if the operating system is initialized in the mainframe. When these situations occur, the contents of all control latches, words in memory, and all registers are considered invalid.

The mainframe is deadstarted through the IOS 0 maintenance channel, which connects processor 0 to the IOBB in IOS 0. When a multiple-processor system is initialized, the deadstart sequence can occur in only one processor because only one processor can exchange to memory location 0 at one time. The remaining processors in the system must wait for an interrupt from the internal processor (ICP) before they can exchange to memory location 0.

The following steps occur when a deadstart exchange is generated:

1. Using the scan path, the deadstart script sets the following registers and flags:

    - XA = 0
    - MM = 0
    - P = 000000
    - NIP = 001000
    - IBxV = 0
    - NIPVLD = 0 (next instruction parcel valid)
    - HLDISS = 1 (hold issue)
    - All data channels ready to write to memory

2. The deadstart script then causes a write to memory location 0 of 006000 000000 000000 000000 .

3. For each processor, the console sends a processor RUN ON (010000) command; processors will do IFETCH to location 0 and loop on the 006000 instruction.

4. The deadstart script loads the operating system and initial exchange package to location 0 from the IOS using any data channel.

5. The deadstart script deadstarts the first processor (for example, processor 0) by sending MCURUPT to that processor.

6. While processors 1 through 3 remain in an idle state, processor 0 exchanges to location 0.

7. When processor 0 completes the initial exchange and becomes active, it reloads another exchange package at location 0 and sends an ICP interrupt to one of the idle processors, allowing that processor to exchange to location 0 and become active.

8. Steps 6 and 7 are repeated until all processors in the system are active and able to perform.

NOTE: The processors can be deadstarted in any sequence.

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

There are also two exit instructions that can initiate an exchange sequence: the error exit instruction (000000) and the normal exit instruction (004000). These two instructions are provided so a program can request its own termination. The normal exit instruction allows a program to exchange back to the monitor area. The error exit instruction is used if an abnormal condition is detected by the object program. In this case, the program can exchange to an error handling program at a location specified by the XA field of the exchange package.

The final condition that can cause an exchange is an interrupt that is received by the object program. When this occurs, the object program exchanges with another program at the address specified by the XA field of the exchange package.

Fetch

Any exchange sequence is immediately followed by an instruction fetch operation (a fetch), which reads program code from memory and places it in an instruction buffer. The instruction buffers hold the program code until the code is required by the execution sequence. When needed, the program code is moved from the instruction buffers into the processor's issue registers.

The CRAY EL series system uses a P register to initiate a fetch sequence and eight instruction buffers to store the program code. Figure 2-7 shows the relationship between the P register, the instruction buffers, and central memory.

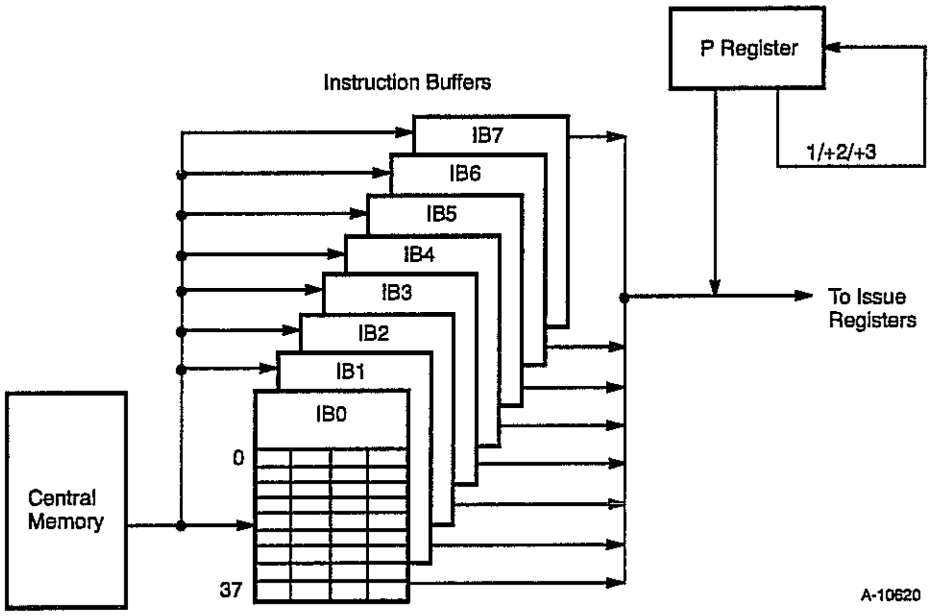


Figure 2-7. Instruction Fetch Hardware

The eight instruction buffers used by the CRAY EL series system can each hold 40_8 (00 through 37_8) words. Each of these words is composed of four 16-bit instruction parcels, providing a total content of 128 parcels. The instruction parcels are held in the instruction buffer until they are selected by the P register to be moved into the issue registers.

The first instruction in an instruction buffer always has a word address that is a multiple of 40_8. This word address allows the entire area of addresses for a single buffer to be defined by the upper 17 bits of the P register.

Each of the instruction buffers has an associated instruction buffer address register (IBAR). The IBAR contains the upper 17 bits of the P register and an IBAR valid bit. When set, the IBAR valid bit is used to indicate that the instruction buffer contains valid data. During an exchange sequence, the IBAR valid bit is cleared, which invalidates the previous program's instructions. When a no coincidence condition occurs, an instruction fetch sequence is initiated. Note that this condition can occur during normal program execution as well as during an exchange sequence. After the fetch sequence begins and the instructions have been loaded into an instruction buffer, the IBAR is loaded with the upper 17 bits of the P register and the IBAR valid bit is set.

The 24-bit P register contains the address of the next parcel of program code that is to enter the NIP register. The upper 22 bits of the P register, shown in Figure 2-8, indicate the word address, and the lower 2 bits are parcel select bits.

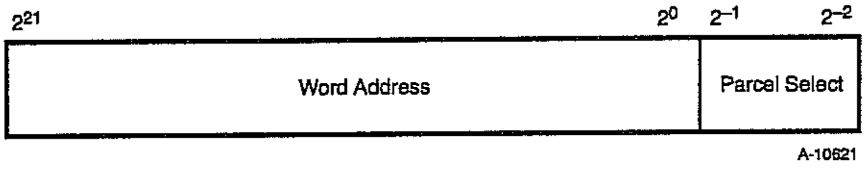


Figure 2-8. P Register

Under normal circumstances, the P register increments sequentially through the program code. This increment occurs whenever an instruction moves into the issue hardware. For a 1-parcel instruction, the P register increments by 1; a 2-parcel instruction causes a +2 increment and a 3-parcel instruction results in a +3 increment. This sequence allows all instructions to issue in 1 clock period. Branch instructions, when encountered, load the P register with the word address of the jump, allowing the program to continue. This branch can occur to another address contained in the instruction buffers, which then issues as if there were no abrupt change. However, it is also possible to issue a branch word address that is not currently held in the instruction buffers. In this instance, a fetch sequence must occur before program execution can continue.

When a program exchanges out, the P register field contains the word address of the instruction immediately following the address of the instruction that last executed.

Instruction Fetch Operation

An instruction fetch operation, or fetch, refers to the series of steps that is performed to read program code from memory into the instruction buffers.

The P register always contains the parcel address of the next instruction that is to be decoded (moved into issue). Each clock period, the P register contents are compared with the eight IBAR contents. If the upper 17 bits of the P register do not match any IBAR, or the IBAR valid bit is not set, a condition called out-of-buffer, or no coincidence, exists. In this condition, there are no valid instructions in the instruction buffers and a fetch sequence is initiated.

If the contents of any one of the IBARs is equal to the upper 17 bits in the P register and the IBAR valid bit is set, a condition called in-buffer or coincidence, exists and no fetch is required.

The fetch sequence uses memory port D to transfer 128 parcels (40_8 words) from memory into the instruction buffer. One word is transferred each clock period. (Refer to Section 3, "Memory," for more information about Memory Port D.)

The instruction buffers are circularly loaded. This means that a fetch sequence moves 128 parcels into the first instruction buffer; then, another fetch is required to fill the next instruction buffer. This process continues sequentially until all eight instruction buffers are filled or until the ILA is reached. If a program requires more than 1024 parcels (eight fetch sequences), the ninth fetch operation overwrites the first instruction buffer used.

The first word delivered to the instruction buffer always contains the instruction that is the first instruction to be executed. As an example, if the P register contains address 124-2 (parcel 2 of word 124) when the fetch operation starts, the first word delivered to the instruction buffer is from memory address 124.

During an instruction fetch operation, instructions are delivered to the instruction buffer at a rate of 1 word per clock period. Because of overhead, it takes 16 clock periods for the first word to arrive from memory, and an additional 2 clock periods for the first instruction to get to the NIP register. Instruction issue can occur concurrently with the fetch operation as long as the required instruction parcel is in the instruction buffer. As long as no memory conflict occurs, an instruction buffer can be loaded in 47 clock periods. However, memory conflicts can lengthen the time required to complete a fetch operation.

Instruction Issue

The CRAY EL series system uses three registers as parts of the instruction issue hardware: NIP*ghijk*, NIP*m*, and NIP*n*. Figure 2-9 shows the registers used, the instruction buffers, and the general flow of the instruction parcels as they move into execution.

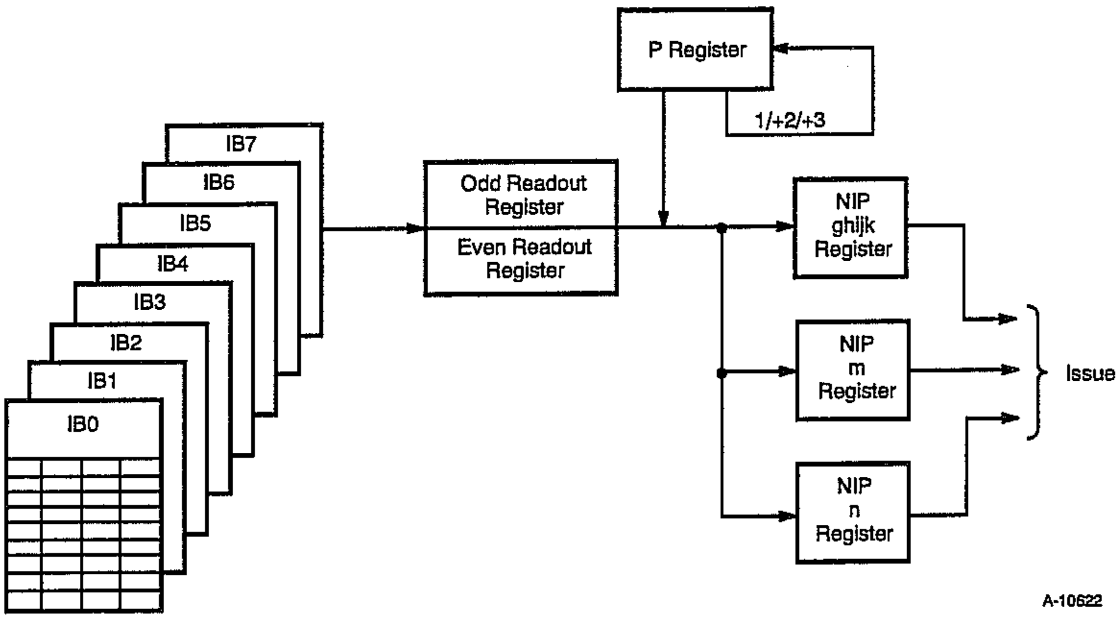


Figure 2-9. Instruction Issue

The instruction buffers hold the program code that was read from memory until it is moved to the issue registers. The instruction buffers have two associated readout registers that are used to channel the flow of instructions between the instruction buffers and the next instruction parcel (NIP) registers. Even-numbered words are loaded into the even readout register, while odd-numbered words move to the odd readout register. Bit 2^0 of the P register determines which readout register to use, while bit 2^{-1} and bit 2^{-2} select the parcel that is to be sent to the NIP registers. The readout registers contain two complete words: one even word and one odd word.

The P register value addresses the instruction buffer and moves 3 instruction parcels into the NIP*ghijk*, NIP*m*, and NIP*n* registers. These 3 parcels are selected from the 8 parcels available in the readout registers. At the same time the 3 parcels are gated into the NIP registers, the type of instruction is determined (1-, 2-, or 3-parcel) and the appropriate add operation (1, +2, or +3) is performed on the P register contents. When this addition is complete, the next instruction is selected and moved into the readout registers if all of the current word has been used. The cycle continues until the program terminates.

In the case of a 1-parcel instruction, the P register gates 3 parcels into the NIP registers. Refer to Figure 2-10. The first parcel enters register NIP*ghijk*, the second parcel goes to register NIP*m*, and the third parcel enters register NIP*n*. If there are no conflicts, the instruction parcel located in register NIP*ghijk* issues and the P register increments by 1.

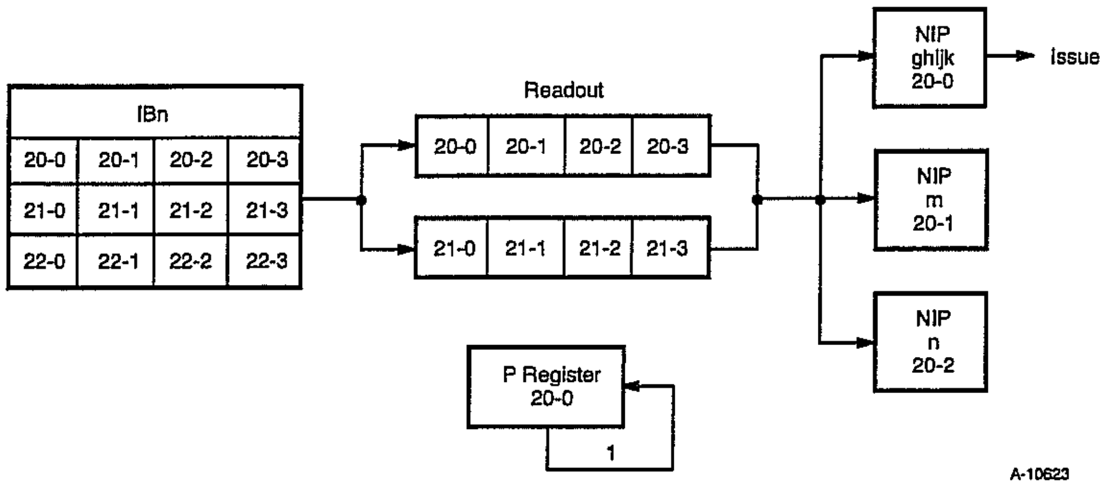


Figure 2-10. Instruction Flow, 1-Parcel Instruction

If the control logic decodes a 2-parcel instruction, the P register loads 3 parcels from the readout registers into the NIP registers. Refer to Figure 2-11. As with a 1-parcel instruction, the first parcel enters register NIP*ghijk*, the second parcel enters register NIP*m*, and the third parcel is placed in register NIP*n*. If there are no conflicts, the parcels placed in register NIP*ghijk* and register NIP*m* are issued, and the P register increments by 2.

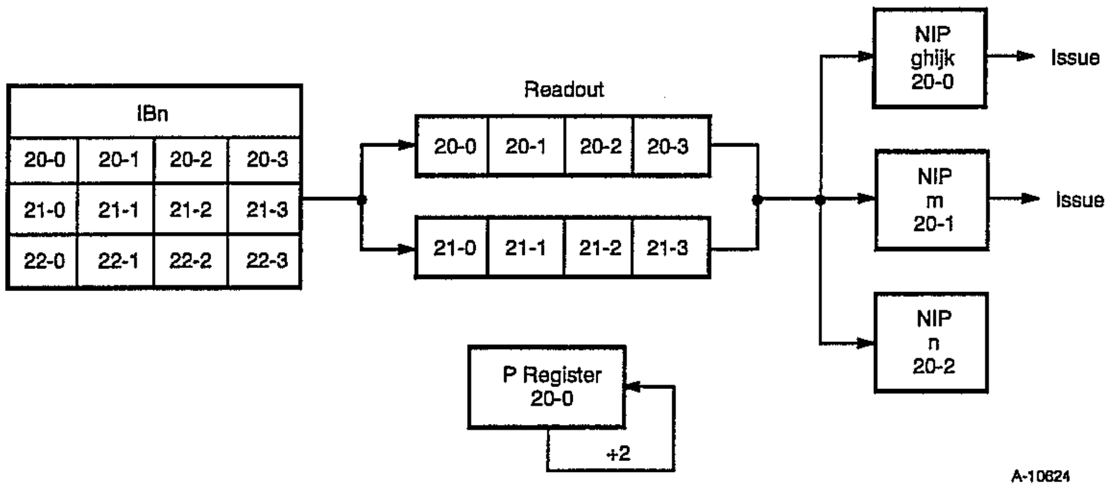


Figure 2-11. Instruction Flow, 2-Parcel Instruction

When a 3-parcel instruction is presented to the control logic, the NIP registers are loaded as previously indicated. However, when the instruction moves into issue, all three NIP registers are used, and the P register increments by 3. Refer to Figure 2-12.

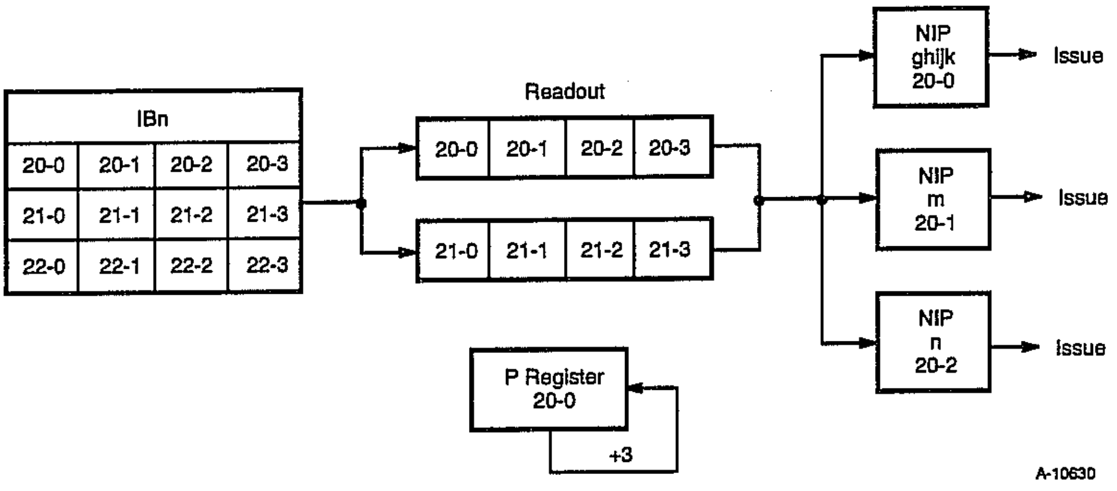


Figure 2-12. Instruction Flow, 3-Parcel Instruction

Any unused instruction parcel remaining in one of the NIP registers is overwritten during the next instruction load sequence, rather than using time to clear the NIP registers before reloading them. Refer to Figure 2-13, Figure 2-14, Figure 2-15, and Figure 2-16 for more information.

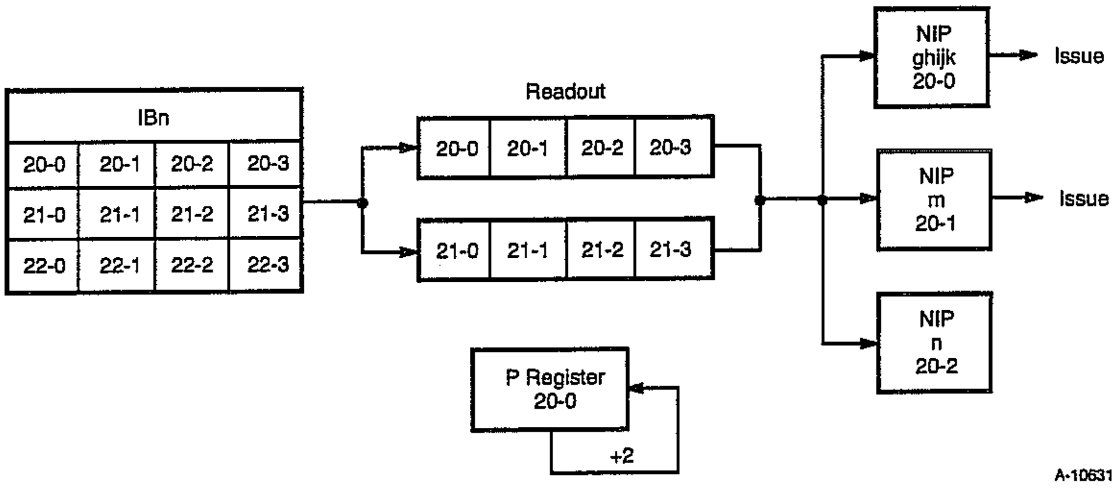


Figure 2-13. Instruction Flow, 2-Parcel Instruction (2 Instructions)

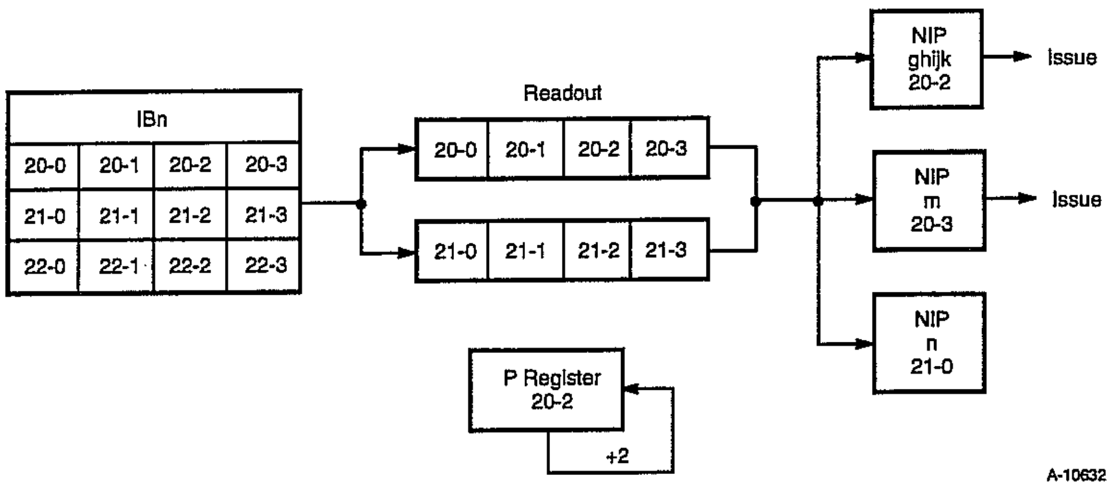


Figure 2-14. Instruction Flow, 2-Parcel Instruction (2 Instructions - Next Instruction Load)

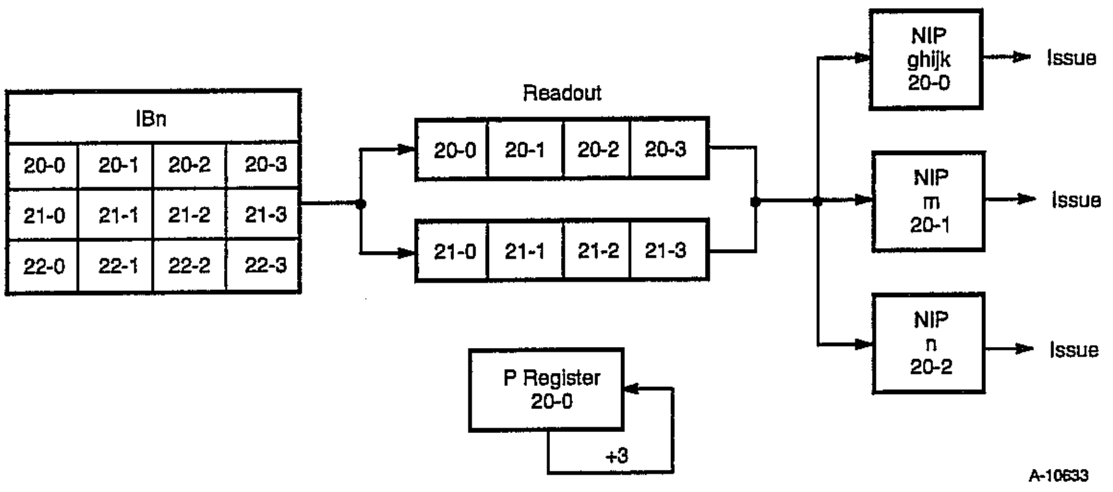


Figure 2-15. Instruction Flow, 3-Parcel Instruction (2 Instructions)

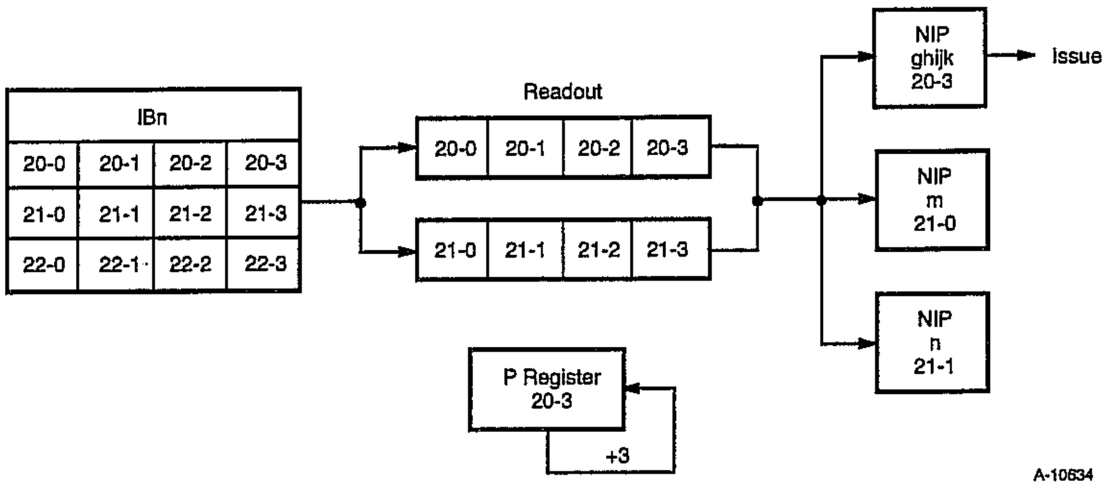


Figure 2-16. Instruction Flow, 3-Parcel Instruction (2 Instructions - Next Instruction Load)

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 brings new code into the instruction buffers and a new value into the P register, and the issue sequence starts again. As long as there are instructions in the instruction buffers and there are no conflicts, the CRAY EL series system can issue a 1-, 2-, or 3-parcel instruction every clock period.

Reservations and Hold Issue Conditions

When an instruction is in the NIP registers, hardware checks determine whether there are any conflicts that will prevent the instruction from completing. These conflicts are referred to as hold issue conditions, because they cause the instruction to be held in the NIP registers until the conflict condition is resolved. Once an instruction issues, reservations are immediately placed on the appropriate registers, paths, ports, or functional units required for the completion of that instruction. The reservations are made on a scoreboard located on the PC ASIC. These reservations are held until a Reset signal is received from the appropriate resource, indicating that the instruction has finished using that resource.

The conditions that cause registers to be reserved are:

- A and S registers used as result registers
- B and T registers used during block transfers
- V registers reserved as either operand or result registers

- Input paths to the A and S registers reserved for the clock period when the data is expected to enter those registers

Port reservations are much the same. Ports A through C are reserved for memory block reads or writes, B and T register transfers, vector stride operations, and vector gather/scatter operations.

A and S functional units and vector functional units are reserved as they are being used.

Another hold issue condition can occur if a scalar instruction references memory and ports A, B, or C are reserved. The scalar block overlap (SBO) bit must also be equal to zero (SBO = 0). This means that the scalar instruction can operate only when all three memory ports are unused.

When using multiple-parcel instructions, it is possible to have a 2-clock-period hold issue condition if the second or third parcel of the instruction is in a different instruction buffer. If the second or third parcel is not in any of the instruction buffers, a fetch operation is performed, which requires 47 clock periods to complete.

Interprocessor Communications

Interprocessor communications in the CRAY EL series system pass data and control between processors. There are three elements of interprocessor communications used by the CRAY EL series system:

- Shared registers
- Semaphore registers
- Interprocessor interrupts

The shared registers pass data between the processors. The semaphore registers synchronize the operation of a program among processors. The interprocessor interrupts allow one processor to initiate an exchange sequence in a different processor. Together, these three functions are especially useful when the CRAY EL series system is used in a multitasking environment.

The shared registers and the semaphore registers are arranged into groups, called clusters, on the processor.

Clusters

The CRAY EL series system contains the shared registers and the semaphore registers, which are assigned to seven clusters on each processor. A processor may access only one cluster at a time. The cluster number (CLN) register in the exchange package determines which cluster the processor can access. The clusters are numbered 1 through 7. A CLN = 0 condition is used to prevent a processor from accessing the shared registers. This function can be used to reserve a processor in single-user mode for a specific program.

There are two ways to change the contents of the CLN register in the CRAY EL series system. One way is during an exchange sequence (refer to the "Exchange" subsection in this section). The second way requires that the processor is in monitor mode and is executing a 0014*j*3 instruction. (Refer to the Appendix for details pertaining to the CRAY EL series system instruction set.)

Shared Registers

Shared registers are used to transfer data among operating registers in different processors. The processor being used by a program can load its shared registers from its own operating registers. A different processor (or several different processors) can then transfer this data into its shared registers, and then transfer the data from the shared registers into its operating registers. This function occurs only if the involved processors are all operating in the same cluster.

The CRAY EL series system contains two types of shared registers: the shared address (SB) registers and the shared scalar (ST) registers. Each cluster contains eight 32-bit SB registers, numbered SB0 through SB7, and eight 64-bit ST registers, numbered ST0 through ST7. Four instructions transfer data between the operating registers and the shared registers in a cluster of a single processor:

- 026*ij*0 transmits (SB*j*) to A*i*
- 027*ij*7 transmits (A*i*) to SB*j*
- 072*ij*3 transmits (ST*j*) to S*i*
- 073*ij*3 transmits (S*i*) to ST*j*

If CLN = 0 when the processor exchange package issues these instructions, the 026 and 072 instructions return a 0 to the designated registers, and the 027 and 073 instructions do not perform any operation.

Semaphore Registers

The semaphore (SM) registers enable a processor to temporarily suspend operation of a program to allow synchronization of that processor with other processors. Each cluster in a processor contains 32_{10} 1-bit SM registers, numbered SM0 through $SM37_8$. During program execution, each processor assigned to a specific operating cluster sets and clears all

SM registers within that cluster. Each of the processors assigned to the cluster can also transmit the contents of the SM registers to or from an S register for examination.

Five instructions are used by the SM registers:

- 0034*jk* tests and sets semaphore *jk*
- 0036*jk* clears semaphore *jk*
- 0037*jk* sets semaphore *jk*
- 072*i*02 transmits (SM) to S*i*
- 073*i*02 transmits (S*i*) to SM

The 0034 instruction tests the state of the semaphore register specified in the *jk* field. If the content of the SM*jk* register is 0, the instruction executes immediately. If the content of SM*jk* is 1, the 0034 instruction holds issue until another processor assigned to the same cluster clears the SM*jk* register. When the 0034 instruction issues in either of these cases, it sets the SM*jk* register to 1.

Instructions 072 and 073 transfer the contents of the semaphore registers to or from the upper 32 bits of the designated S register. The relationship between the registers involved in this transfer is shown in Figure 2-17. The contents of SM0 are transferred to bit position 2^{63} of the S register, while the contents of SM37 are placed in bit position 2^{32}. The lower 32 bits of the selected S register are not used during this type of transfer.

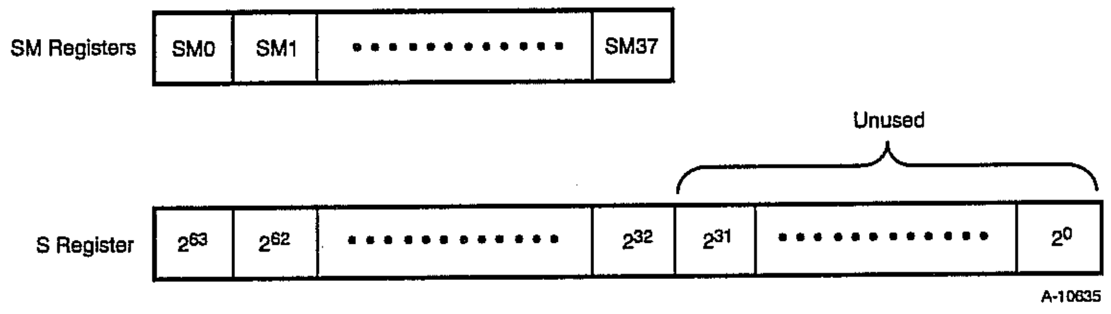


Figure 2-17. Relationship between Semaphore Registers and an S Register

Deadlock

A deadlock condition occurs when all processors assigned to a cluster are holding issue on a 0034*jk* (test and set) instruction. If this condition exists, no program execution can occur in any of the processors assigned to the cluster because each is waiting for one of the others to issue a clear SM (0036*jk*) instruction.

A deadlock interrupt resolves a deadlock condition. Then, in each of the processors where the interrupt occurred, the deadlock (DL) flag sets in the exchange package. This causes each of the deadlocked processors to execute an exchange sequence, clearing the deadlock.

Two special conditions apply to the deadlock condition. First, if a processor has CLN = 0, a deadlock condition cannot occur because the 0034 instruction should perform no operation. If a DL flag is continuously being set and the program is being denied execution because of this, it could be advantageous to test the 0034 instruction in an offline environment to help isolate the error.

The second condition involving a deadlock occurs if only one processor is assigned to a specific cluster. In this case, the processor encounters a deadlock if it issues a 0034 instruction. There is no other processor available to clear that set condition, and a deadlock exchange occurs.

Interprocessor Interrupts

The final segment of interprocessor communications involves interprocessor interrupts, which occur when a processor interrupts another processor's program. A processor must be in monitor mode to issue the interrupt instructions. The interrupt instructions are:

- 0014*j*1, which sets interprocessor interrupt request of processor (A*j*)
- 001402, which clears interprocessor interrupt request

When the 0014*j*1 instruction is executed, the interrupt from internal processor (ICP) flag is set in the processor specified by (A*j*). If this processor is not in monitor mode, it begins an exchange sequence. The program that enters the processor is in monitor mode and issues a 001402 instruction, clearing the ICP flag. If this instruction does not execute, the processor immediately begins another exchange sequence.

In a special case, the 0014*j*1 instruction is executed with the contents of A*j* equal to the number of the processor executing the instruction (the processor is, in effect, interrupting itself); no operation is performed.

Real-time Clock

The CRAY EL series system has a device called the real-time clock (RTC). The RTC consists of four identical devices, one on each processor, running synchronously to appear as one RTC to the programmer. In this discussion, the combination of circuits forming the actual RTC is treated as a single device, because a mainframe containing only one processor (one RTC device) performs the same way as a mainframe containing eight processors (eight RTC devices).

The RTC is a 64-bit register that increments each clock period. Two instructions affect the RTC:

- 0014*j*0, which transmits (S*i*) to RTC
- 072*i*00, which transmits (RTC) to S*i*

The 0014*j*0 instruction can be executed by a processor that is in monitor mode only. Two or more processors in monitor mode should not attempt to execute this instruction simultaneously because unpredictable results will occur. However, there is no hardware to detect this situation. The 072*i*00 instruction can be executed simultaneously by any number of processors, because only the copy of the RTC on the local processor is read to the S register.

For more detailed information on registers and functional units, refer to the *CRAY Y-MP EL System Programmer Reference Manual,* CSM-0435-000.

MEMORY HIPPI OVERVIEW

This subsection contains basic information about the organization and operation of the memory HIPPI.

HIPPI Fundamentals

HIPPI is a 32-bit parallel unidirectional point-to-point data channel. It is usually installed in input/output pairs for bidirectional operation. Data flows from a HIPPI source to a destination.

From the user's perspective, the basic unit of information on the channel is a packet; however, the channel hardware divides packets into bursts of 256 32-bit words. Ready signals from the destination to the source control the information flow; each Ready signal lets the source send one burst. The Packet signal travels from the source to the destination to mark the boundaries between packets.

The HIPPI source sends a Request signal to a destination. The destination returns a companion signal, called a Connect signal, to the source. When both signals are present, a connection exists, and data flow may begin.

To establish a connection, the source sends the Request signal to the destination. At this time, the source places 32 bits of information (called the I-field) on the data lines. The destination examines the I-field before it responds to the Request signal. The destination responds either by accepting the request or rejecting it. Acceptance consists of sending the Connect signal and transmitting Ready signal pulses. To reject a request, the destination sends a timed Connect signal and clears it again without transmitting any Ready pulses. The source responds by clearing the Request signal.

Connections may be broken either by the source (by clearing the Request signal) or by the destination (by clearing Connect). The other side must respond by clearing its corresponding signal. The source must cease sending data when the connection is broken. When a destination breaks a connection, data in transit can be lost.

Dedicated and Shared HIPPI Channels

The HIPPI driver supports two modes of operation: dedicated and shared. The mode is assigned to each channel when the channel is opened. If a channel is dedicated, only one process can have that channel open at a time. If a channel is shared, several processes can use one HIPPI channel. The driver enforces a protocol on the shared channel, allowing it to determine the destination of each incoming message.

HIPPI Protocol

On dedicated channels, the HIPPI driver treats all HIPPI packets as opaque data; no specific protocol is required or recognized.

Shared channel operation requires that all applications use the ANSI draft HIPPI Framing Protocol (HIPPI-FP). The `sys/hippifp.h` header file contains definitions for the header. The HIPPI driver examines the default path (ULP-ID) field in each input packet. For an application to receive a packet in shared mode, its path value must match the contents of this field. The default value of path is n+128 (where n = minor device number, modulo 16). For example, the ULP-ID for `/dev/hippi/i01` and `/dev/hippi/o01` is 129, the ULP-ID for `/dev/hippi/*02` is 130, and so on.

HIPPI 800 and 1600 Modes

The HIPPI-PH specification describes both 32-bit and 64-bit HIPPI implementations. Nominal data rates are 800 Mbytes/s and 1600 Mbytes/s, respectively. CRAY EL series systems have only the 32-bit HIPPI implementation.

Device Driver

The memory HIPPI implementation includes a device driver. The driver entry points are added to the character device switch table. User programs access the memory HIPPI channels through character special device files that have a major device number of 30. When creating device entries for the memory HIPPI, you must use the major number 30.

By convention, the device driver files are created in the `/dev/hippi0` directory. If a second HIPPI is present, `/dev/hippi1` would contain the special files for this interface. Two files are needed for full-duplex operation. A separate open system call is needed for an input and for an output channel.

You can access the memory HIPPI in three ways:

- Through UNICOS
- Using Transmission Control Protocol/Internet Protocol (TCP/IP)
- Through UltraNet software (not covered here)

The memory HIPPI driver implementation supports either dedicated use by one application (a user program or TCP/IP) or shared usage. If the channel is being shared, all traffic must conform to the HIPPI-FP (Framing Protocol) specification for packet format. When the input channel is run in shared mode, all sharing applications must try to keep multiple reads posted to the driver at all times. This ensures that a buffer is available for incoming data, because system buffering is not used for the HIPPI channel. The user program must post multiple reads when using the `reada` or `listio` system call.

The memory HIPPI driver logs hardware errors into the file `/usr/adm/errfile`. You can use the `-d hippi` option on the `/etc/errpt` command to process these entries.

Channel Configuration

The driver automatically configures HIPPI channels during the UNICOS boot sequence. The memory HIPPI driver tests each potential memory HIPPI channel for the presence of HIPPI hardware by issuing the channel CPU master clear instruction followed by a read of the channel address. If the HIPPI hardware exists, it returns a channel identification that indicates the type of interface attached to the CPU in this position. (Three types of interfaces exist: a Y1 channel to the IOS, a memory HIPPI source, and a memory HIPPI destination.)

The automatic channel configuration eliminates the need to make any manual UNICOS kernel configuration changes for the memory HIPPI. To use the memory HIPPI, you must create the device nodes and update the TCP/IP configuration files; you do not have to reboot UNICOS for these activities.

When the memory HIPPI driver identifies a HIPPI channel during the boot sequence, the console screen displays a message with the following format:

```
HIPPI inout channel 064 initialized
HIPPI output channel 067 initialized
```

These system console messages are also logged in the files `/usr/adm/syslog/kern` and `/usr/adm/syslog/daylog`. The initialization code modifies the channel table. The displayed channel numbers are based on the location of the CPU board on which the memory HIPPI resides. CPU board 0 supports memory HIPPI channels 24 and 27; CPU board 1 supports memory HIPPI channels 44 and 47. CPU board 2 supports memory HIPPI channels 64 and 67; CPU board 0 supports memory HIPPI channels 104 and 107.

A HIPPI CPU module can support two memory HIPPI channels. A 4-CPU CRAY EL series system can support up to four memory HIPPI pairs (input and output). When the software locates a memory HIPPI module, it allocates channel table space to describe the channel and its 16 associated logical paths. The lowest-numbered channel is associated with minor device numbers 0 through 15 (major device number 30). The channel with the next higher number is associated with minor device numbers 16 through 31. The first logical path, path 0 for each channel, is the dedicated path. The remaining paths are shared paths.

I/O Control

Logical path 15 of each channel is reserved for I/O control. This channel can be opened and closed and `ioctl` system calls may be issued, but data transfers will be rejected. Unlike the other shared channels, an open system call on path 15 does not prevent path 0, the dedicated path, from being opened.

The `ioctl` system call is used to change the values for timeouts, set a new I-field, change the disconnect flag, change the I-Field-in-data flag, or to obtain a detailed status from a previous error.

ifconfig Command

The `-ptp` option of the `ifconfig` command (configure network interface parameters) marks the interface as point-to-point. If you specify the `-ptp` option, the output HIPPI channel will be set into `HOLD` mode (not `DISCONNECT`). If you omit the `-ptp` option, the output

HIPPI channel will be placed in DISCONNECT mode. You can set HOLD or DISCONNECT mode in this manner if the interface is associated with the dedicated path (path 0). You cannot change the disconnect flag from the system default if a shared path is chosen for TCP/IP. For memory HIPPI, DISCONNECT mode is the default. You set HOLD or DISCONNECT mode only on the output channel.

You may have to modify programs that take advantage of additional features, such as the ability to read one packet into two user buffers or to write one packet from two buffers. The memory HIPPI allows a maximum of two buffers. The first buffer must be 1024 bytes or smaller. If an incoming packet has an initial short burst (allowed by the HIPPI standard specification) and this burst is smaller than the user's first buffer, the remainder of the first buffer is skipped and the remainder of the packet is written to the user's second buffer.

The memory HIPPI driver does not include support for the configuration of HIPPI disks or HIPPI tapes, but a user program can use the memory HIPPI driver to communicate with these devices.

Testing the Memory HIPPI with UNICOS

Testing the memory HIPPI under UNICOS is possible in either loopback mode or in master/slave mode using the diagnostic VHT.

Loopback Testing

1. Connect one end of a HIPPI cable to the input port of the memory HIPPI board by connecting the cable to the DESTINATION connector (HI RA) at the bulkhead connector.

2. Connect the other end of the HIPPI cable to the output port of the memory HIPPI board by connecting the cable to the SOURCE connector (HI TA) at the bulkhead connector.

3. With the cable connected in loopback (from the SOURCE connector to the DESTINATION connector of the memory HIPPI board), test the raw HIPPI connection using the diagnostic VHT. (*Raw connection* refers to the use of system calls to operate the channel instead of TCP.) An example follows:

```
/etc/vht -i/dev/hippi0/i00 -o/dev/hippi0/o00 -c1000 -D -P -1
```

This example uses the input device `/dev/hippi0/i00` and the output device `/dev/hippi0/o00`. The `-c1000` option limits the pass count to 1000 passes. The `-D` option specifies to disconnect between packets. The `-P -1` option specifies a pseudo-random data test pattern. The following example illustrates a successful VHT execution:

```
/etc/vht -i/dev/hippi0/i00 -o/dev/hippi0/o00 -c1000 -D -P -1
Using I field 00000000
random pattern selection
HIPPI test path 0 completed pass xxx
```

NOTE: *xxx* is the pass count and will increment as VHT executes.

4. Connect a cable from the SOURCE connector of the CRAY EL memory HIPPI board to the DESTINATION connector of another system or a HIPPI switch. Connect a cable from the DESTINATION connector of the CRAY EL memory HIPPI board to the SOURCE connector of another system or a HIPPI switch.

5. If you connect the memory HIPPI to a HIPPI switch, you can perform loopback to the switch by including an I-field in the preceding example. The I-field value depends on the switch configuration. An example of the VHT command line with the I-field included follows:

```
/etc/vht -i/dev/hippi0/i00 -o/dev/hippi0/o00 -c1000 -D -P -1 -I0x01000004
```

Master/Slave Testing Using VHT

If the CRAY EL memory HIPPI is connected to another Cray Research HIPPI, you can perform a master/slave test using the diagnostic VHT when VHT is set up to handle the memory HIPPI driver changes.

NOTE: The location and name of the device files on another Cray Research system may be different than those given in the procedure that follows.

To run the master/slave test, you either must be logged on to both Cray Research systems and have access at one central point (that is, using multiple screens or windows) or you must be able to run a remote shell from the Cray Research system that initiates the VHT test to the other Cray Research system. The second case requires that you set up a `.rhosts` file in the `$HOME` directory of the remote host.

Using VHT to start the master/slave VHT test, you must select either the `/etc/hosts` entries to identify the remote Cray Research system host (specified by the `-h` parameter) or use the IP dot notation address.

In the following example, the VHT command is executed on system CrayA. CrayB-ether identifies an ethernet connection to another Cray Research computer, CrayB, that is directly connected on the HIPPI between CrayA and CrayB. If a HIPPI switch exists between CrayA and CrayB, you must add the `-I` option.

1. Enter the following command at the slave system:

```
/etc/vht -hCrayB-ether -i/dev/hippi0/i00 -o/dev/hippii0/o00 -c1000 -D -P -1
```

In this example, the output might look like:

```
Using I-field 00000000
remsh CrayB-ether /etc/vht -hCrayB-ether -i/dev/hippi0/i00
-o /dev/hippi0/o00 -c1000 -D -P -1 -o /dev/hippi0/o00 -u 128
-t 10 -w -1 0 -n 1&
random pattern selection
HIPPI test path 0 completed pass 800
/etc/vht 0 passes 1-1000 no fatal errors
```

2. Run the preceding test using two connections (or windows) and no remote shell. At the system that will be reading data, enter:

```
/etc/vht -w -o /dev/hippi0/i00
```

3. At the system that will be writing data, enter:

```
/etc/vht -w -o /dev/hippi0/o00
```

You must initiate the read before the write so that the write will not time out. You can add other options such as `-c`, `-P`, and so on to the write and read. Each operation must use the same parameters so that `VHT` can determine how many cycles to run and which test patterns to transmit.

UNICOS Testing with TCP/IP

Configure UNICOS for TCP/IP across the HIPPI Connection. You must add, change, or verify the following files to configure the memory HIPPI board for TCP/IP:

- `/etc/hosts`
- `/etc/hycf.hippi`
- `/etc/config/interfaces`

1. Add the alias name(s) for the memory HIPPI input connection to the `/etc/hosts` file. An example follows:

```
129.152.94.1     CrayA-hippi
129.152.94.2     CrayB-hippi
```

NOTE: If a `/etc/hosts.bin` file already exists, replace it with a new `/etc/hosts.bin` file that includes the alias name changes by executing the `/etc/mkbinhost` command.

2. Create a `/etc/hycf.hippi` file. It is used when `/etc/hyroute` is executed to bring up the HIPPI channels. An example of a `/etc/hycf.hippi` file follows:

```
#        Hostname     I-field    In      Out      Mtu
#
direct CrayA-hippi   00000058   0000    0010     65536;
direct CrayB-hippi   00000038   0020    0030     65536;
```

The `hycf` file for the HIPPI defines all of the connections to the HIPPI channel. An entry for the resident CRAY EL HIPPI connection and an entry for all other HIPPI connections must exist. If you connect to multiple HIPPI devices, use a HIPPI switch. The preceding `hycf` file example depicts a HIPPI directly connected between two Cray Research systems.

The host names can be either IP dot notation addresses or aliases of IP addresses. If you use aliases, ensure that they are defined in the `/etc/hosts` file.

The I-field is a 32-bit field that contains information on mode control (bits 31 through 24) and routing control (bits 23 through 0).

The I-field is in hexadecimal format. For a direct HIPPI connection between two Cray Research systems, use only the routing control field. In this case, the routing control field should have a unique number for each HIPPI entry, preferably in the range of 0x00 to 0xFF.

If you are connecting to a HIPPI switch, you must refer to the manufacturer's HIPPI switch documentation for I-field decoding. The memory HIPPI connection does not support 64-bit mode.

The In and Out numbers in the `/etc/hycf.hippi` file reflect an index into the communication table created by the memory HIPPI driver software and point to the minor numbers for the HIPPI device files. This field is in hexadecimal format and is only significant for the resident HIPPI connection entry. For example, the first entry (the resident CRAY EL connection) in the preceding entry has 0000 for the In and 0010 for the Out minor numbers.

The In minor number, 0000, corresponds to the input of the HIPPI device files (in this case, it is `/dev/hippi0/i00`).

```
ls -l /dev/hippi0/o00
crw-rw-rw-  1 root  root   30,   0 May 7 10:49 /dev/hippi0/i00
```

The Out minor number, 0010 (0010 hexadecimal or 16 decimal), corresponds to the output HIPPI device file (in this case, it is `/dev/hippi0/o00`).

```
ls -l /dev/hippi0/o00
crw-rw-rw-  1 root  root   30,  16 May 28 10:34 /dev/hippi0/o00
```

The MTU number reflects the maximum transmission unit that will be allowed for outbound (or write) datagrams. This field is in decimal format. To create the inbound MTU value, use the `/etc/ifconfig` command.

3. Configure TCP/IP for the HIPPI, by first entering the `hyroute` command.

```
/etc/hyroute hi0 -s /etc/hycf.hippi
```

Then enter the `ifconfig` command. Substitute the correct IP address or alias for your interface and the desired netmask for your network.

```
/etc/ifconfig hi0 CrayA-hippi iftype hippi mtu 65536 netmask 0xffffff00
```

If this is a direct HIPPI connection from one Cray Research system to another Cray Research system, add the destination address parameter to the `ifconfig` command. The following example of adds CrayB-hippi as the destination:

```
/etc/ifconfig hi0 CrayA-hippi CrayB-hippi iftype hippi mtu 65536 > netmask 0xffffff00
```

4. Test TCP/IP across the HIPPI channel by issuing the `ping` command to the hosts listed in your `hycf` file (not the resident address).

```
/etc/ping CrayB-hippi
```

Then use the `nettest` client and `nettest` server program to perform a TCP/IP memory-to-memory test. The following screen shows the suggested parameters for the server and client programs. Start the server side first.

```
/etc/nettestd -p tcp &
/etc/nettest -p tcp -s 4 -b 378k -f CrayB-hippi 100 378k
```

The `&` puts the `nettest` daemon in the background; `-p` specifies the TCP protocol; `-s 4` specifies the maximum TCP window shift factor; `-b 378k` specifies that 378 Kbytes of buffer should be allowed; `-f` specifies that full-size read will always be issued on reading the data; `CrayB-hippi` specifies the IP address; `100 378k` specifies the count and size of the write operations.

For example, with CrayA-hippi as the server, start `nettestd` by entering:

```
/etc/nettestd -p tcp &
```

Then on CrayB-hippi host, start the client by entering:

```
/etc/nettest -p tcp -s 4 -b 378k -f CrayB-hippi 100 378k
```

The test results are displayed at the client system.

Stop `nettestd` on the server host by entering:

```
kill -9 [PID]                    #(Enter the process ID number for PID)
```

Error Messages

When a fatal error occurs, the HIPPI driver returns one of the following error codes in `errno`. The error codes and their meanings are as follows:

`EBUSY` - The channel is already open.

`EFAULT` - A bad argument address was specified in an `ioctl` request. This error can be caused by a bad buffer address where the address is not word aligned (any non 8-byte multiple).

- A fatal I/O error occurred or the HIPPI channel closed while asynchronous I/O was active (on a read or write system call).
- An I/O request timed out (on a read or write system call) after transferring some data.

The detailed error status is available in the err field of the hxio structure; this field can be read with the `ioctl` request `HXC_GET`.

`ELATE` - An input request timed out (no data transferred).

`ENXIO` - The HIPPI channel is unavailable (on an open system call), or the channel is not open (on a closed system call). On CX/CEA systems, this code can mean that the IOP failed to allocate some resource.

After a fatal error, the detailed error status is available with the `ioctl` request `HXC_GET`. The driver returns error codes in the err field of the `hxio` structure.

The following detailed error codes are defined on IOS-D systems and CRAY EL series systems with VME HIPPI:

`HXST_BUF` - IOS I/O buffer unavailable on open operation.

`HXST_CHAN` - CPU gave bad channel number to IOS; caused by configuration error.

`HXST_DBG` - Debug mode error; indicates a driver fault (should never occur).

`HXST_EOB` - Unexpected end of packet (input only).

`HXST_FLGS` - Buffer flags do not match; indicates a driver (should never occur).

`HXST_FMEM` - IOS free memory unavailable on open operation.

HXST_HISP - No high-speed channel to this IOP; caused by error or wrong target memory.

HXST_LLEN - Transfer length too long; indicates a driver fault (should never occur).

HXST_LONG - Long block received (input only).

HXST_MOS - MOS buffer unavailable on open operation (debug mode only).

HXST_NDEV - No device present on the channel (hardware signal).

HXST_OK - No error (binary 0).

HXST_OPEN - Channel is not open; indicates a driver fault (should never occur).

HXST_OVER - Data overrun error (input only).

HXST_TM - Bad target memory type; indicates a driver fault (should never occur).

HXST_TMO - Request timed out.

HXST_ZLEN - Buffer length is 0; indicates a driver fault (should never occur).

HXST_CTMO - Connection timed out.

HXST_CNPR - Connection not present.

HXST_CREJ - Connection rejected.

HXST_DISC - Channel disconnected.

HXST_CNPE - No connection pending.

HXST_CAPR - Connection already present.

HXST_CABT - Connection aborted.

HXST_LLRC - LLRC error.

HXST_CPE - Channel parity error.

HXST_CHST - Invalid channel status.

HXST_BOE - Buffer overrun error.

XST_OIM - Odd initial microburst.

HHXST_BPE - Buffer parity error.

HXST_IPE - I-field parity error.

The following detailed error codes apply to CRAY EL series systems with memory HIPPI:

HIST_INV_RL - Request packet length is not valid

HIST_IS_UP - Channel already configured up

HIST_NOT_UP - Channel is not configured up

HIST_NOT_IMP - Function is not implemented

HIST_RTMO - Read request timeout

HIST_DRV_DOWN - Driver terminated by config down

HIST_IS_OPN - Logical path already open

HIST_RAW_OPN - Bad raw channel open request

HIST_NOT_OPN - Logical path is not open

HIST_PTH_CLS - Read abandoned because path closed

HIST_LONG - Long packet excess discarded (nonfatal)

HIST_DTA_ERR - Data integrity error on read

HIST_CHAN_TMO - Channel activation timeout

HIST_DRV_TERM - Driver terminating

HIST_NOT_CNCT - Interconnect-A not present

HIST_HALT_IO - R/W request returned due to halt-io

HIST_INV_PARM - Configure UP parameter that is not valid

HIST_NOT_64 - Cannot write in 64-bit HIPPI mode

HIST_CN_REJ - Connection attempt was rejected

HIST_CN_FAIL - Connection attempt failed (other)

HIST_CN_TMO - Connection request timeout

`HIST_CN_STUCK` - Connect-in will not drop

`HIST_INV_SF` - Device control subfunction that is not valid

`HIST_HANGUP` - Connection went away

`HIST_SECDED` - SECDED error in I/O buffer

`HIST_INTRCNCT` - Lost Interconnect signal

`HIST_EOB` - Short block received