Processors and Memory

(CRAY J90™ Series)

HMM-118-A

Cray Research Private

Cray Research, Inc.

Record of Revision

REVISION	DESCRIPTION
	March 1995. Original printing.
A	July 1995. Revised to include 8 X 8 midplane information for CRAY J932 system.

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
890 Industrial Blvd.
Chippewa Falls, WI 54729

Processors and Memory

Figures

Tables

Introduction

This document describes the processor and memory modules for the CRAY J916 and CRAY J932 computer systems (hereafter referred to as CRAY J90 series systems). A basic CRAY J90 series system includes one mainframe cabinet and one input/output (I/O) cabinet.

The mainframe cabinet may contain either two, four, or eight memory modules and from one to eight processor modules. A memory module may be either half-populated or fully populated and can use either 256K X 16 or 1M X 16 dynamic random-access memory (DRAM) chips. Each processor module contains one I/O channel (on the CI ASIC), up to four Y1 channels, and up to 4 central processing units (CPUs), for a maximum of 32 CPUs per system.

Refer to the *Hardware Overview* document for more information on system configurations, including memory options and sizes.

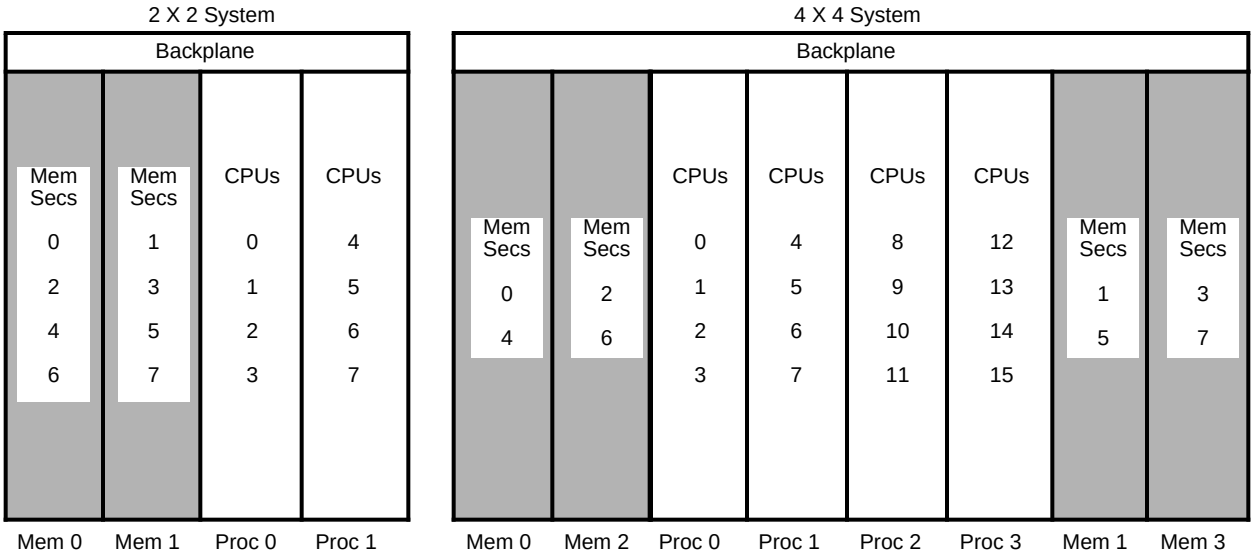
The CRAY J90 series processor and memory modules use application-specific integrated circuit (ASIC) technology. Each module is composed of an array of ASICs, which are based on very large-scale integration (VLSI) complementary metal oxide semiconductor (CMOS) technology.

Backplane Configurations

The processor and memory modules connect to the CRAY J90 series system by means of a backplane that contains physical slots for the processor and memory modules. Three types of backplanes are available for CRAY J90 series systems: a 2 X 2 or 4 X 4 backplane for a CRAY J916 system, or an 8 X 8 midplane for a CRAY J932 system. These numbers refer to the maximum number of processor modules and memory modules that a system can include.

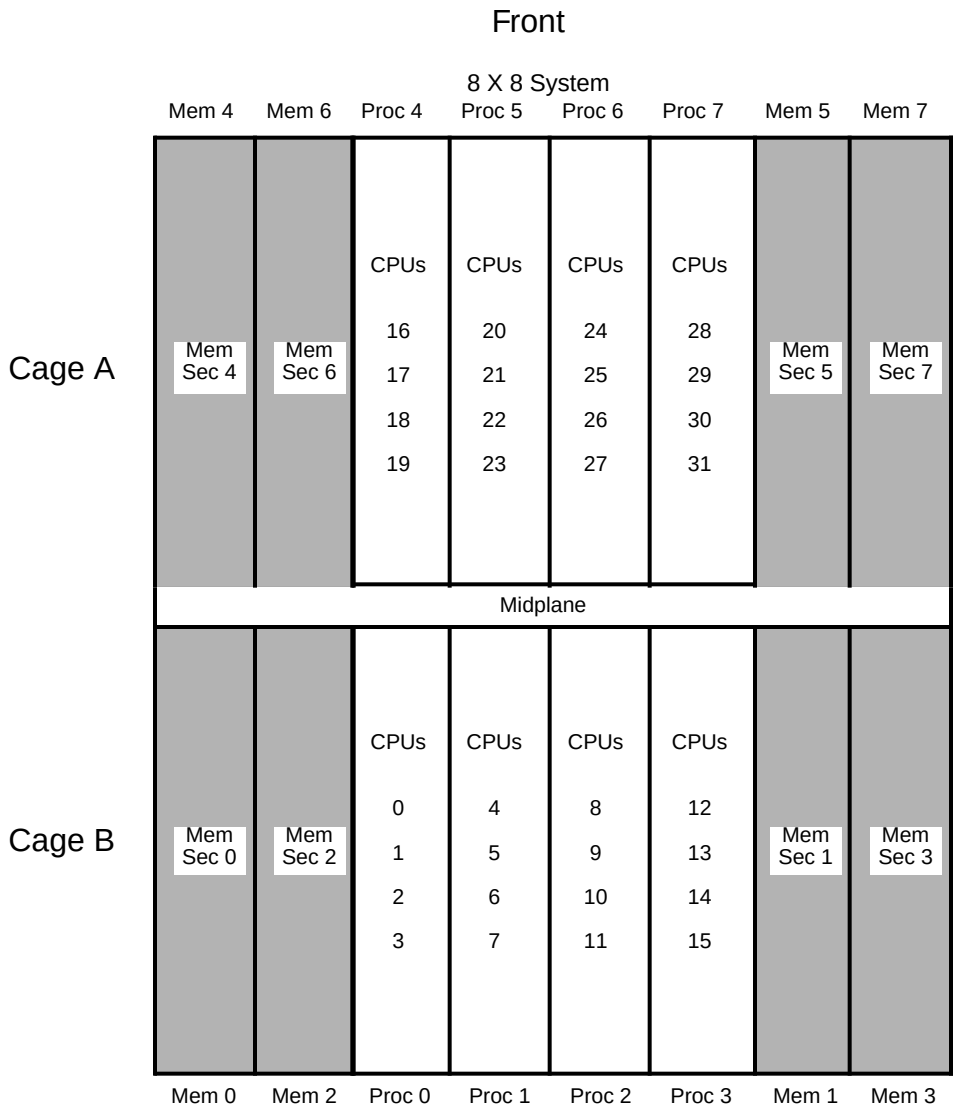
The 2 X 2 backplane configuration (refer to Figure 1) includes two memory modules and one or two processor modules (4 to 8 CPUs). The 4 X 4 backplane configuration (refer to Figure 1) includes four memory modules and from one to four processor modules (4 to 16 CPUs). The 8 X 8 midplane configuration (refer to Figure 2) includes eight memory modules and from four to eight processor modules (16 to 32 CPUs).

The memory module slots in all systems are always fully populated; however, some processor slots could be vacant.



NOTES: Proc 1 slot may be vacant in 2 x 2 configurations.
Proc 1, Proc 2, and Proc 3 slots may be vacant in 4 x 4 configurations.
All memory module slots will always be filled.
The clock module is plugged across from the mainframe modules on the backplane.

Figure 1. 2 X 2 and 4 X 4 Module Slot Locations (Top View)



NOTES: Proc 4, Proc 5, Proc 6, and Proc 7 slots may be vacant in an 8 x 8 configura
All memory module slots will always be filled.

Figure 2. 8 X 8 Module Slot Locations (Top View)

Processor Module

Each processor module contains 4 CPUs with one vector unit and one scalar unit per CPU. Each CPU also has a computation section that consists of operating registers, functional units, and a control section. Refer to Figure 3 for a CPU block diagram. Refer to Figure 4 for a processor module block diagram.

A processor module is approximately 16 in. X 20 in. and contains the logic for 4 CPUs. Each CPU has one high-speed port to each section of memory that supports 800-Mbyte/s write and read operations. The CI ASIC provides the I/O interface. A processor module has 26 layers: 12 signal layers with buried vias, 12 power/ground layers, and 2 surface layers.

A processor module is composed of three printed circuit boards: a CPU board, a channel adapter (CA) board, and a logic power module (LPM). An integral aluminum framework that surrounds each processor module provides mechanical support, protects the components, and directs the airflow. Each CPU printed circuit board (PCB) is approximately 12 in. X 16 in. X 0.16 in. and contains ASICs in a 4 X 6 array.

The following subsections describe the components of a processor, the functions of the ASICs on the processor module, and processor communications.

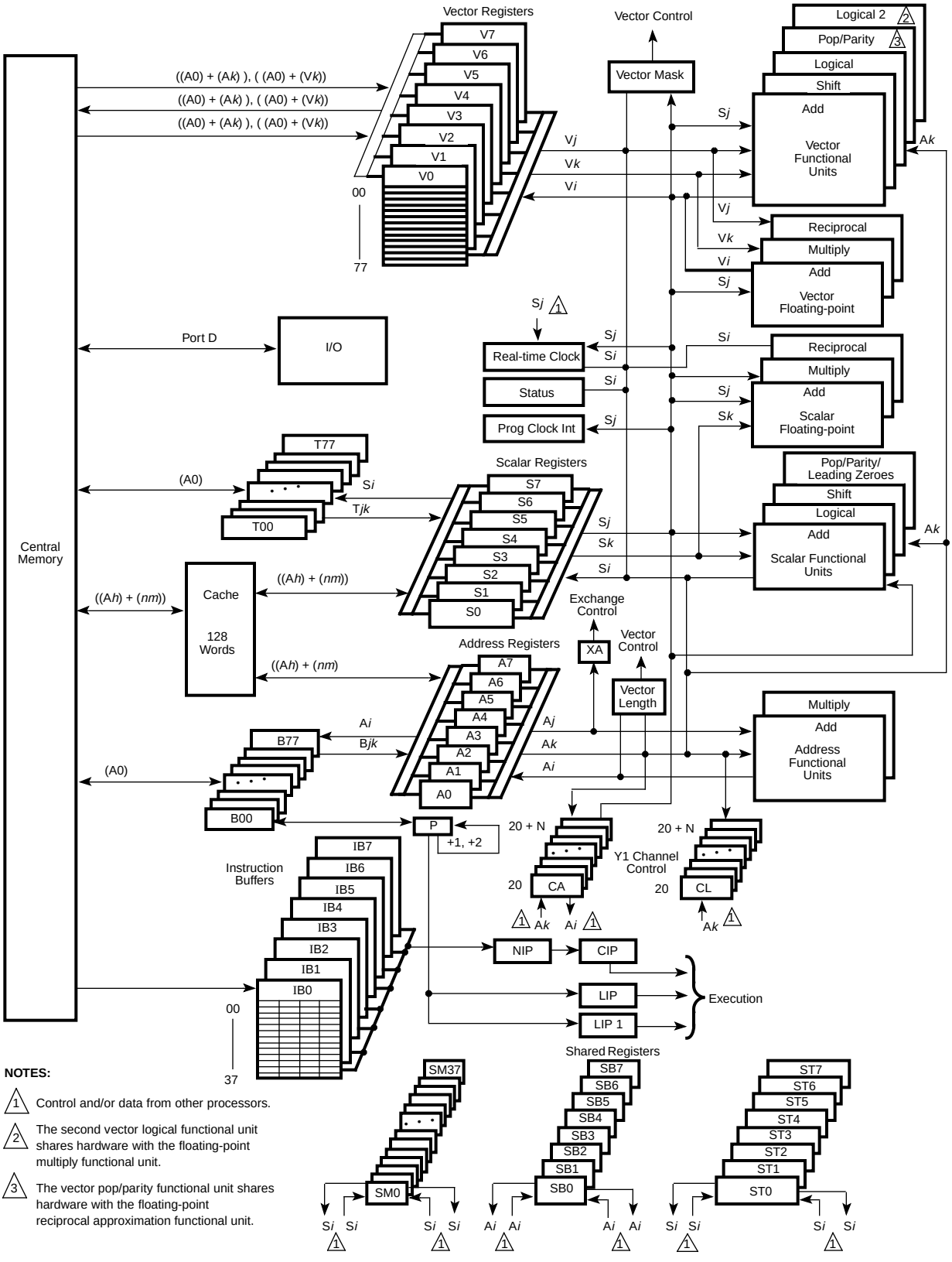


Figure 3. CPU Block Diagram

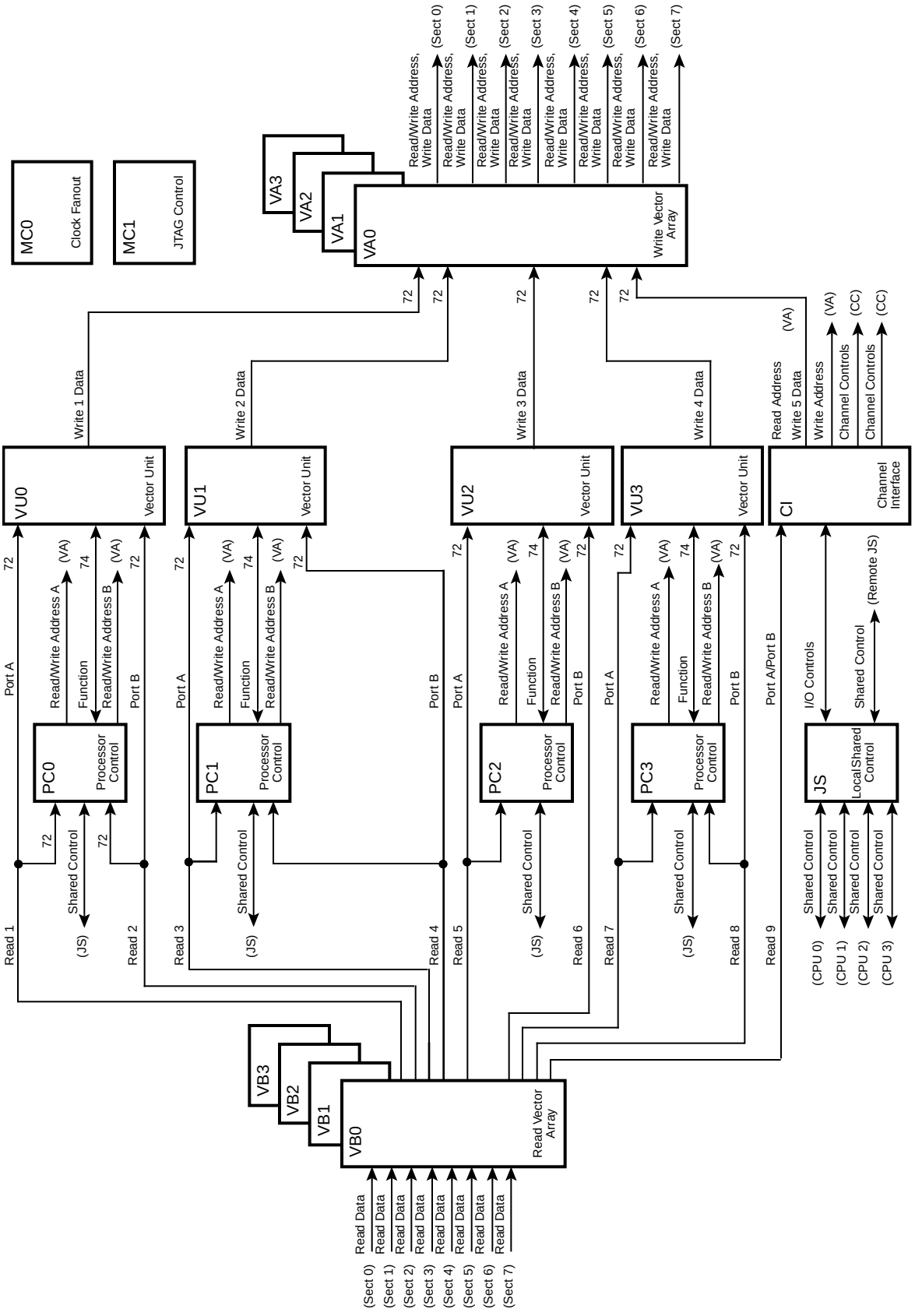


Figure 4. Processor Module Block Diagram (Represents 4 CPUs)

ASICs

Each CPU board contains the following eight types of ASICs, for a total of 20 ASICs on each CPU board. Refer to Figure 5 for the layout of the ASICs on the CPU and channel adapter boards of the processor module.

- Four vector unit ASICs (VU), each of which contains one full set (64 elements) of vector registers and associated vector and floating-point arithmetic units.

- Four vector array, memory write interface ASICs (VA), which control memory access and arbitrate memory conflicts. The VA ASICs send scalar, vector, and I/O requests to memory.

- Four vector array, read data ASICs (VB), which distribute memory read data to the requesting unit on the processor module.

- One shared resources ASIC (JS), which includes the logic for interprocessor communication, shared registers, and interrupt control. The JS ASIC sends I/O memory errors to one of four local processors. Each JS ASIC contains a copy of the global real-time clock (RTC) and the status of each CPU. When the RTC is written, all global copies are updated at the same time. Each individual JS is then responsible for updating the copies of the RTC that are local to each PC ASIC.

  NOTE: Revision 4 processor modules contain a JS ASIC that supports more than 16 CPUs.

- Four processor control ASICs (PC), which contain the A and S registers, local RTC, instruction buffers, B/T registers, performance monitor, issue control, and scalar functional units, including scalar floating-point units. Each PC ASIC also contains a 128-word set-associative cache.

- One channel interface ASIC (CI), which supports the functions of the Y1 channel and HIPPI channel. The CI ASIC passes I/O memory error information to the JS ASIC. The CI ASIC also contains two buses for simultaneous read and write operations with central memory.

  NOTE: Revision 3 processor modules support four Y1 channels or one HIPPI channel pair or two Y1 channels and two HIPPI channel pairs.

- One maintenance and clock fanout ASIC (MC0′), which controls clock fanout to all modules in the system. Refer to the "System Clock" subsection at the end of this document for more information about clock functions.

- One maintenance and clock JTAG control ASIC (MC1′), which provides Joint Test Action Group (JTAG), boundary scan, stop clock, maintenance channel, and reset maintenance functions.

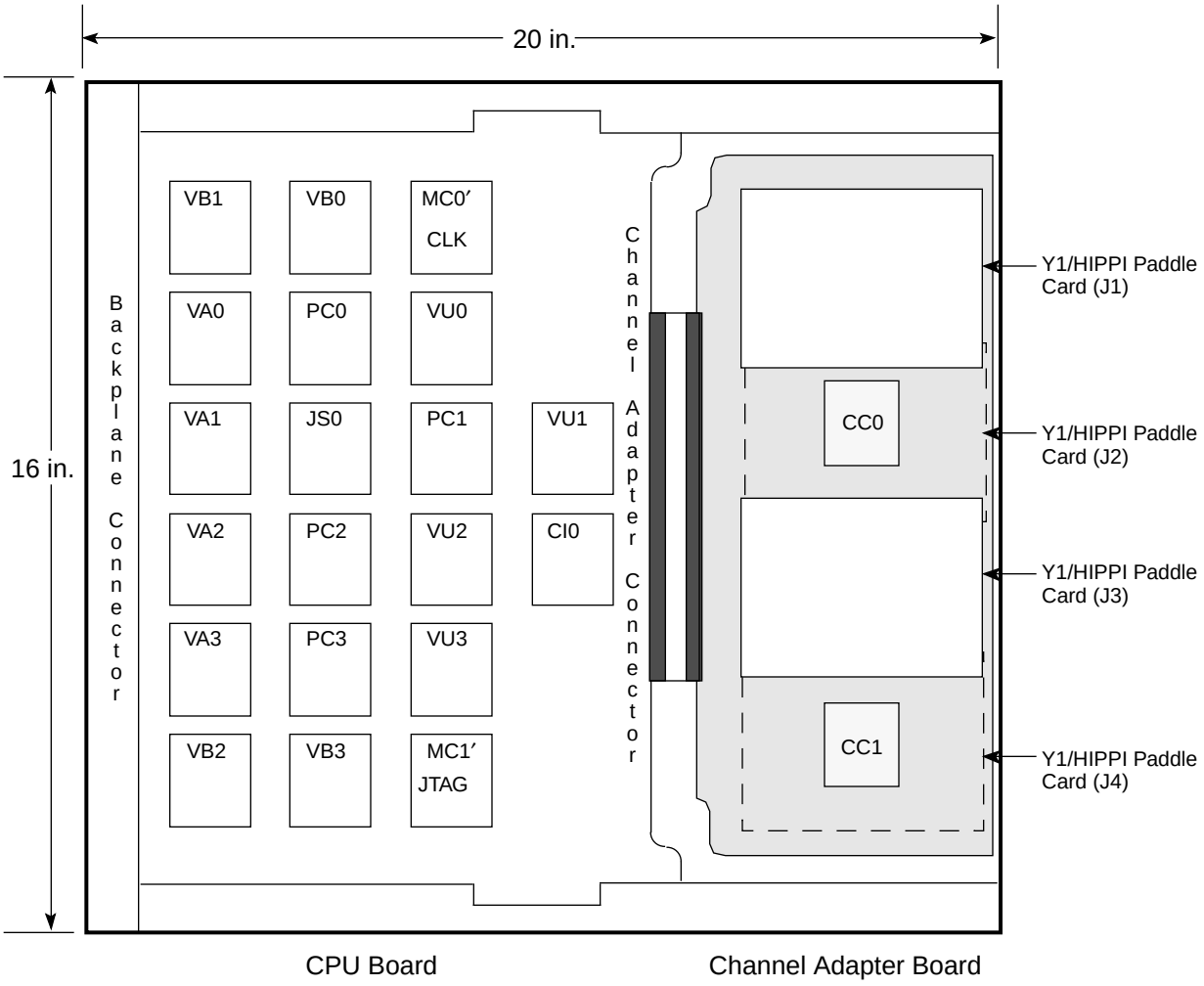


Figure 5. CPU and CA Board ASIC Layout

Interprocessor Communications

The JS ASIC contains the shared resources logic for the CRAY J90 series systems. Each processor module has one JS ASIC. The most significant features of the JS ASIC are:

- Each JS ASIC on each processor module contains a complete copy of all shared resources.
- Read operations of shared resources are done on the local JS ASIC.
- Write operations of shared resources are passed on to all JS ASICs.
- Shared registers are split into separate units to enable multiple concurrent access.

Each JS ASIC has a pair of buses to each of the four PC ASICs on the processor module as well as a pair of buses to the CI ASIC on that module. Each JS ASIC also has a dedicated bus to each of the other JS ASICs on other processor modules.

Shared Registers

When the PC ASCI generates a shared register read command, the local JS ASIC on the processor module receives the request and directs the command to its global logic. The global logic includes the shared registers, deadlock logic, set interprocessor interrupt (SIPI) control, I/O interrupt control, CPU status, and global real-time clock (RTC). The shared register read operation is held until all outstanding reads and writes from the same PC ASIC have completed; the data is then returned to the originating CPU. Because each JS ASIC has an identical copy of the shared resources, there is no need to pass the read request to the other JS ASICs.

When a shared register write command is received by the JS ASIC, the command waits for all outstanding read requests to complete. Then it must wait its turn to be sent on the dedicated bus that interconnects all JS ASICs. It must wait for access because other CPUs may also request access to the same resource. Once the request is granted, the write command is sent to all the JS ASICs in the system. No notification of completion is returned to the initiating CPU.

Real-time Clock

Each JS ASIC receives a copy of the global RTC. When the RTC is written, all global copies are updated at the same time. Each individual JS ASIC updates the copies of the RTC for each local PC ASIC. When the JS ASIC receives the global RTC load command, all other JS activity is halted; after the halt, the JS ASIC transfers the new RTC value to all the PC ASICs at the same time.

Processor Status

The status of each CPU is contained in the global logic of each JS ASIC. The status includes the monitor mode flag bit, selected for external interrupts (SEI) flag bit, and the cluster number. Whenever any of these statuses change, the CPU must send the new value to the JS ASIC.

Test and Set Control

The test and set control logic handles test and set semaphore instructions for the processors. When a processor issues a test and set instruction, it sends a test and set command to the JS, which then passes it on the global logic using the dedicated bus that interconnects all JS ASICs. The global test and set logic is updated on the following conditions:

- Whether or not a processor is doing a test and set instruction
- The cluster number of each processor
- The semaphore register number of each processor

When the test and set logic receives the test and set command, it checks to determine whether the semaphore register is set. If the register is not set, the logic sets it and returns a completion status to the originating processor. If it is already set, the logic enters a wait-on-semaphore state and notifies the originating processor.

Whenever a clear semaphore (SM) or load SM instruction is executed, the status logic in the JS ASIC for each CPU determines whether the semaphore it is waiting on is cleared. If the semaphore register is cleared, the CPU makes a request to set it. One of the requesting CPUs is granted permission. The SM is set and the CPU is notified that the instruction has completed. Simultaneously, the processor that originated the test and set instruction is holding issue. It holds issue until it receives a response from the JS ASIC. If the JS ASIC returns a completion command to the CPU, then the test and set instruction issues and execution continues. If the JS ASIC returns a deadlock command, the program

(P) register is backed up and the processor exchanges with the deadlock flag set. A deadlock chain passes the waiting on semaphore (WS) bit and cluster number (CLN) for each CPU to processor status logic in each JS ASIC.

Interprocessor Interrupt

The interprocessor interrupt logic is part of the JS ASIC global logic. It routes interprocessor interrupts to the correct CPU. When a processor issues a SIPI instruction, it sends the command to the local JS ASIC, which passes it on to other JS ASICs. The interconnecting bus interface logic routes the SIPI to the correct processor.

I/O Interrupt

When an I/O interrupt occurs, the CI ASIC sends the interrupt command to the JS ASIC along with the interrupting channel number. The global logic receives the command and channel number and sends them on the dedicated bus that interconnects all JS ASICs. After the processor to be interrupted has been determined, the local JS ASIC sends the interrupt to the PC ASIC.

I/O Memory Errors

On the CRAY J90 series systems, the I/O channels do not share memory ports with specific processors. Each I/O channel is associated with the 4 CPUs that share the processor module. Through the scan configuration, one of the processors is selected to handle I/O memory errors. When a memory error occurs on I/O, the CI ASIC passes the error information to the local JS ASIC. The JS ASIC then sends the error to the CPU configured to handle memory I/O errors.

Scalar Cache

A scalar cache memory enables portions of the main memory address space to be mapped into a small high-speed memory. Only scalar (A and S) references are cached. The cache is split into a number of lines or groups of data words, which represent contiguous blocks of main memory locations. Each cache line has an associated tag that identifies which main memory address the line represents.

The cache may be direct-mapped (in which case a given memory address may be mapped to only one position in the cache) or associatively mapped (in which the memory address may be mapped to any location in the cache). A combination of these mapping techniques is most practical and is called *set-associative*. The CRAY J90 series system uses this set-associative scheme.

The cache is a 128-word, 2-way, set-associative cache, which contains 128 1-word lines in 64 sets of 2 lines each. The line replacement algorithm is least recently used (LRU) on a per-set basis. The only cache instruction is a 0016*j*1 instruction that invalidates cache in processor A*j*. The entire cache is invalidated on an exchange or a cache flush operation.

The following list summarizes the general operation of cache:

- Scalar reads that hit a valid cache word do make memory requests, but these are redundant and are later aborted.
- Only scalar misses (read or write) allocate cache lines.
- Memory returns for scalar reads update the cache and pass the return data on to the CPU.
- Scalar writes store through the cache.
- Vector writes invalidate matching cache lines.
- An exchange or flush invalidates the entire cache.

When a scalar write to cache operation occurs that results in a hit (or allocate), the referenced word is updated. When a scalar read to cache operation occurs that results in a hit, the referenced word is read and sent immediately to the CPU. A scalar read or write operation that missed the cache and cannot allocate a new cache line makes a normal memory request, which does not affect the cache.

Processor Control

The basic timing control for processor and memory modules is provided by the master clock/scan PCB. The processor cabinet contains the master clock/scan PCB. To run a program within the system, you must load the program into the selected CPU and then issue 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 CPU. The fetch sequence, which immediately follows the exchange sequence, transfers a block of instructions from memory into the CPU's instruction buffers. After the instructions are loaded into the instruction buffers, the issue sequence invokes the instructions one at a time.

The program address (P) register indicates the instruction to be issued. The P register points to the location in the instruction buffer that contains the desired instruction; this causes that instruction to be decoded and then issued. When the

selected instruction is issued, the P register increments and causes a new instruction to begin the decode process. This function continues until the P register cannot locate a desired instruction in the instruction buffers. When the P register cannot locate an instruction, another fetch sequence is initiated, which loads another block of instructions into the instruction buffers.

Exchange Mechanism

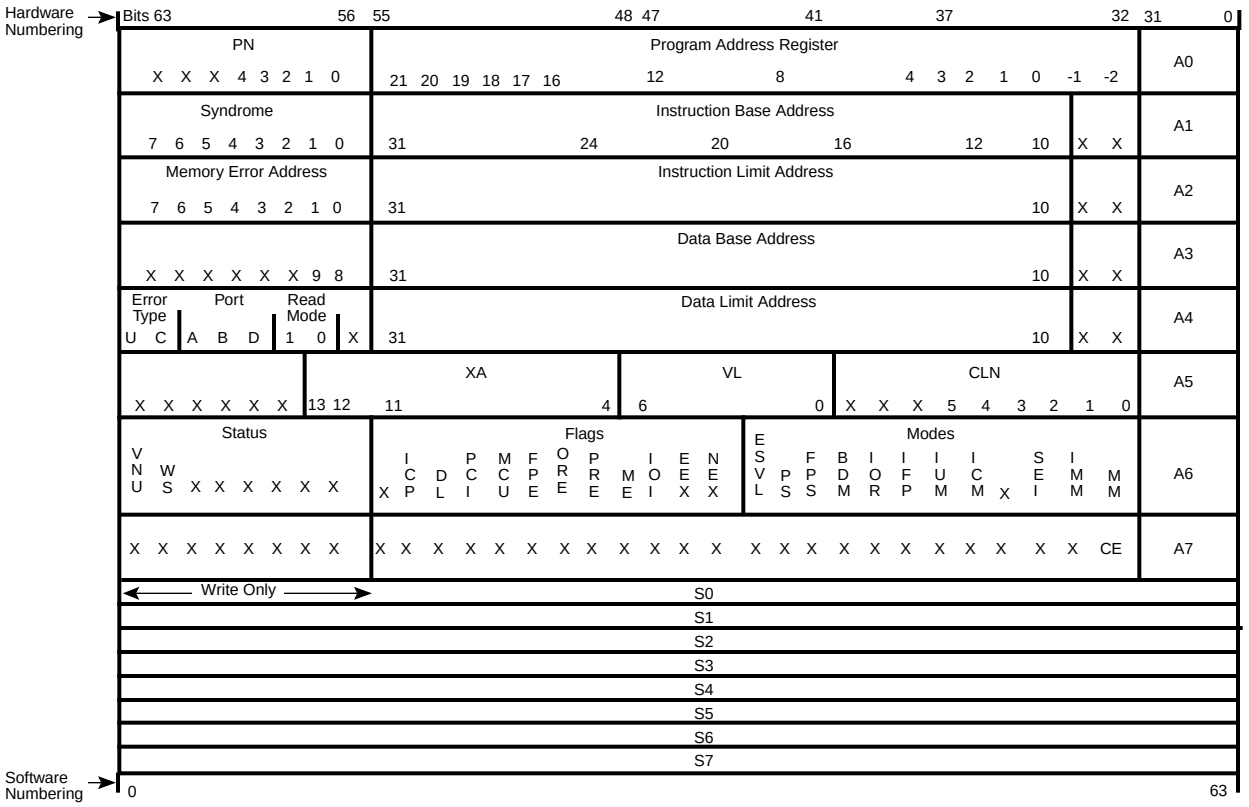
Each CPU uses an exchange mechanism to load programs or to switch from one program to another. This exchange mechanism uses a block of program parameters called an *exchange package*, which contains the required addresses and limits imposed on the program. Another component of the exchange mechanism is the *exchange sequence*, which is a basic CPU operation that loads a new exchange package and saves a copy of the current one.

Exchange Package

The contents of the exchange package (refer to Figure 6) include eight address (A) registers, eight scalar (S) registers, and the contents of specific parameter registers. Refer to Table 1 for descriptions of the acronyms and bits that represent these parameters. Table 2 describes the exchange package port and read mode value translations. 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 provides 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 CRAY J90 series system uses a new mode bit called CE (cache enable).

Exchange Sequence

The exchange sequence provides a process that deactivates the currently executing program and places its current operating parameters 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 them into the CPU exchange registers.



Read Mode

Bit 1	Bit 0	Port A	Port B	Port D Channel
0	0	EX	Fetch A	n+1
0	1	B	T	n+3
1	0	Vector	Vector	n+5
1	1	A/S	Fetch B	n+7

n = Processor module number X10 + 20

Figure 6. Exchange Package

Table 1. Exchange Packet Bit Assignments

		Software	Hardware	
Field	Word	Bits		Description
PN	0	3 – 7	60– 56	Processor number
P	0	8 – 31	55 – 32	Program address register
S	1	0 – 7	63 – 56	Syndrome bits
IBA	1	8 – 29	55 – 34	Instruction base address register
MEA	2	0 – 7	63 – 56	Memory error address
ILA	2	8 – 29	55 – 34	Instruction limit address register
MEA	3	6 – 7	57 – 56	Memory error address (continued)
DBA	3	8 – 29	55 – 34	Data base address register
E	4	0 – 1	63 – 62	Read error type
PORT	4	2 – 4	61 – 59	Port
RM	4	5 – 6	58 – 57	Read mode
DLA	4	8 – 29	55 – 34	Data limit address register
XA	5	6 – 15	57 – 48	Exchange address register
VL	5	16 – 22	47 – 41	Vector length register
CLN	5	26 – 31	37 – 32	Cluster number
VNU	6	0	63	Vector not used
WS	6	1	62	Waiting for semaphore
FLAGS	6	9 – 19	54 – 44	Flag register
MODES	6	20 – 31	43 – 32	Mode register
CE	7	31	32	Cache enable
A	0 – 7	32 – 63	32 – 0	Eight A registers
S	8 – 15	0 – 63	63 – 0	Eight S registers

Table 2. Exchange Package Read Mode and Port Translations

Port Value	Mode Value	Type of Transfer when Error Occurred	Explanation
4 = A	0	EX	Error occurred while reading the exchange package
4 = A	1	B	Error occurred during a read to the B registers
4 = A	2	V	Error occurred during a vector read from memory
4 = A	3	A, S	Error occurred during a memory read to the A or S registers
2 = B	0	Fetch A	Error occurred during an instruction fetch operation on port A
2 = B	1	T	Error occurred during a block transfer to the T registers
2 = B	2	V	Error occurred during a vector read from memory
2 = B	3	Fetch B	Error occurred during an instruction fetch operation on port B
1 = D	0	Y1 or HIPPI	SECDED error occurred during a memory read for channel (n) output
1 = D	1	Y1 or HIPPI	SECDED error occurred during a memory read for channel (n + 3) output
1 = D	2	Y1 or HIPPI	SECDED error occurred during a memory read for channel (n + 5) output
1 = D	3	Y1 or HIPPI	SECDED error occurred during a memory read for channel (n + 7) output

n = Processor module number

Channel Adapter PCB

An additional 7 in. X 16 in. printed circuit board (PCB) called a channel adapter (CA) attaches to the end of the CPU board. The CA provides the interface between the external channel and the CPU. The CA provides a 64-bit wide path between the CI ASIC on the CPU board and the two channel control (CC) ASICs on the CA board. Each CA has paddle cards (sometimes called *daughter* cards) for I/O functions. The paddle card options include the Y1 channel and memory High Performance Parallel Interface (HIPPI) source and destination channels. Refer to Figure 5 on page 10 for the paddle card locations.

CC ASICs

The CC ASICs control all channel activity. Each CA board contains two CC ASICs: CC0 and CC1. Each CC ASIC controls two paddle cards and provides support for either two Y1 channels or a HIPPI channel pair, except for CC0 on CPU 0 (channels 20/21), which must function as Y1 channels for proper CPU to IOS control (refer to Table 3).

The CA board provides support for both VMEbus and HIPPI channel protocols. Each CA board can support any of the following configurations:

- Four Y1 channels
- Two HIPPI channel pairs
- Two Y1 channels and one HIPPI channel pair

The input/output buffer board (IOBB) in the IOS buffers the data and provides the IOS interface to the CC ASICs on the CA board of the CPU. One Y1 channel services one IOBB within the IOS. A physical limitation of the configuration exists because HIPPI channels and Y1 channels cannot be combined on the same CC ASIC. CC0 controls connectors J1 and J2 on the channel adapter board, and CC1 control connectors J3 and J4 (refer to Figure 5).

Table 3. Processor Modules and Associated Y1 Channel Numbers

Processor Module Number	CC0				CC1			
	Y1 Channels		Y1 Channels		Y1 Channels		Y1 Channels	
	Input	Output	Input	Output	Input	Output	Input	Output
0	20	21	22	23	24	25	26	27
1	30	31	32	33	34	35	36	37
2	40	41	42	43	44	45	46	47
3	50	51	52	53	54	55	56	57
4	60	61	62	63	64	65	66	67
5	70	71	72	73	74	75	76	77
6	100	101	102	103	104	105	106	107
7	110	111	112	113	114	115	116	117

NOTE: All channel numbers listed are octal numbers.

Channel Communications

All data transfers between a CPU and a peripheral device pass through the IOS, except for HIPPI channel data transfers. The HIPPI channel transfers data between the CPU and external devices. The processor module connects to the IOS through a set of I/O cables called Y1 channels (refer to Figure 7). Each Y1 channel uses a 32-bit wide, bidirectional data bus with byte parity to transfer command or data words between the CPU and the IOS.

Processor and Memory Communication

Each processor module has a path to each of the 8 memory sections. To resolve conflicts, each of the 4 CPUs on a processor module contains a copy of all memory and shared register reservations.

In each CPU, the operating registers, instruction buffers, and exchange package have access to central memory through memory ports. Each CPU has two ports, port A and port B, to enable up to two simultaneous memory references from each CPU (two memory read operations or one read operation and one write operation). Another port, port D, is used for I/O and instruction fetch operations.

HIPPI

Each processor module can be configured with a HIPPI channel pair. The HIPPI channel provides high-speed communication between the CRAY J90 series system and other HIPPI-compatible external devices. The HIPPI is a 100-Mbyte/s, point-to-point, simplex interface that conforms to industry standards and is contained on the processor modules.

The HIPPI channel transfers data by using a signaling protocol that enables transfers at distances up to 82 ft (25 m). HIPPI protocol uses two separate channels that form a HIPPI channel pair. These two channels are the HIPPI destination (input) and the HIPPI source (output) channels.

Table 4 shows the possible input/output channel configurations for the Y1 channel, the HIPPI input channel (HI-I), and the HIPPI output channel (HI-O). The processor-to-Y1 and processor-to-HIPPI data-level translations are handled by the individual Y1 or HIPPI paddle cards.

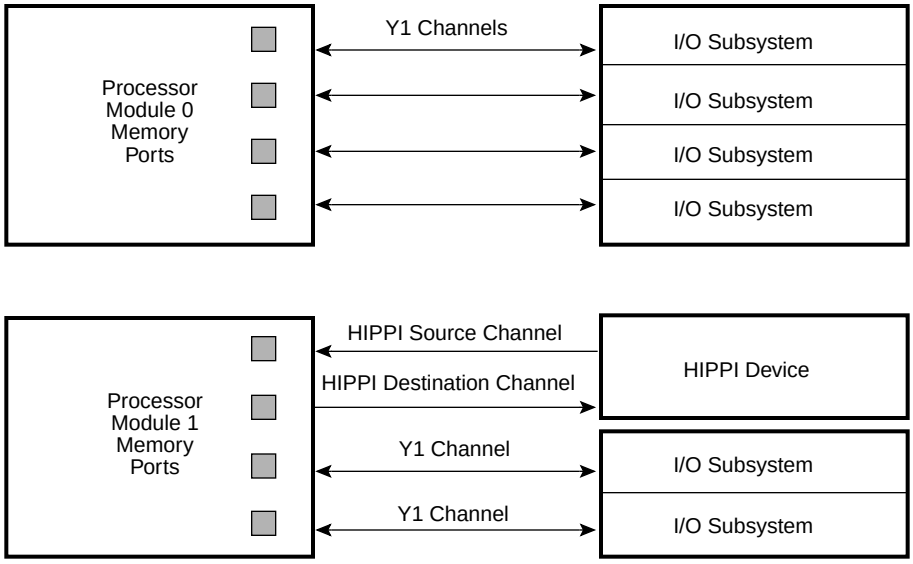


Figure 7. Maximum I/O Configuration per Processor Module

Table 4. HIPPI or Y1 Channel Configurations

Proc Mod 0	Proc Mod 1	Proc Mod 2	Proc Mod 3	Proc Mod 4	Proc Mod 5	Proc Mod 6	Proc Mod 7
Paddle Card Slot J1							
*20/21, Y1	*30/31, Y1 *30, HI-I	40/41, Y1 40, HI-I	50/51, Y1 50, HI-I	60/61, Y1 60, HI-I	70/71, Y1 70, HI-I	100/101, Y1 100, HI-I	110/111, Y1 110, HI-I
Paddle Card Slot J2							
*22/23, Y1	*32/33, Y1 *33, HI-O	42/43, Y1 43, HI-O	52/53, Y1 53, HI-O	62/63, Y1 63, HI-O	72/73, Y1 73, HI-O	102/103, Y1 103, HI-O	112/113, Y1 113, HI-O
Paddle Card Slot J3							
24/25, Y1 24, HI-I	34/35, Y1 34, HI-I	44/45, Y1 44, HI-I	54/55, Y1 54, HI-I	64/65, Y1 64, HI-I	74/75, Y1 74, HI-I	104/105, Y1 104, HI-I	114/115, Y1 114, HI-I
Paddle Card Slot J4							
26/27, Y1 27, HI-O	36/37, Y1 37, HI-O	46/47, Y1 47, HI-O	56/57, Y1 57, HI-O	66/67, Y1 67, HI-O	76/77, Y1 77, HI-O	106/107, Y1 107, HI-O	116/117, Y1 117, HI-O

* Either paddle card 0 or paddle card 1 can be configured as the deadstart channel. By default, paddle card 0, Y1 channel 20/21, is configured as the deadstart channel.

Power PCB

Each processor module has its own logic power module (LPM) located on the back side of the processor module. The LPM receives 48-Vdc power and converts it to the closely regulated 3.3-Vdc power needed by the processor ASICs. The onboard power board is also used on memory modules. On a memory module, a converter supplies 5-Vdc power for the DRAM chips and 3.3-Vdc power for the ASICs. Both power sources are provided in n + 1 units for improved reliability.

The LPM also provides several local control functions such as voltage margining, local inhibit, and air-stream overtemperature sensing. Refer to the *Power, Cooling, and Control* document for more information on power.

Memory

Central memory consists of two, four, or eight memory modules that provide from 32 to 1024 Mwords of memory. Central memory has a peak memory bandwidth of 12.8 Gbytes/s for a 2 X 2 backplane, 25.6 Gbytes/s for a 4 X 4 backplane, and 51.2 Gbytes/s for an 8 X 8 midplane. Each memory word consists of 72 bits: 64 data bits and 8 check bits for error detection.

The memory components are distributed across all memory modules; the number and type of components depend on the system configuration. Refer to Figure 12 for a memory module block diagram. DRAM chips provide storage for data and correction bits. The DRAM chips have a 70-ns access time.

Refer to Figure 8 and Figure 9 for illustrations of half-populated and fully populated memory modules in a 4 X 4 system. In a 2 X 2 or 4 X 4 backplane configuration, central memory is divided into 8 sections. Each memory section contains 4 subsections, and each memory section in a 2 X 2 system contains 2 subsections. Each subsection contains 16 pseudobanks when fully populated and 8 banks when half populated.

In a 2 X 2 system, a fully populated memory contains 256_{10} banks, and a half-populated memory contains 128_{10} banks. In a 4 X 4 system, a fully populated memory contains a total of 512_{10} banks; a half-populated memory contains 256_{10} banks. Each central memory bank can be accessed once every 14 CPs.

Refer to Figure 10 and Figure 11 for illustrations of half-populated and fully populated memory modules in an 8 X 8 system. In an 8 X 8 midplane configuration, central memory is also divided into 8 sections. Each memory section contains 8 subsections. Each subsection contains 16 pseudobanks when fully populated and 8 banks when half populated. A fully populated memory contains $1{,}024_{10}$ banks, and a half-populated memory contains 512_{10} banks.

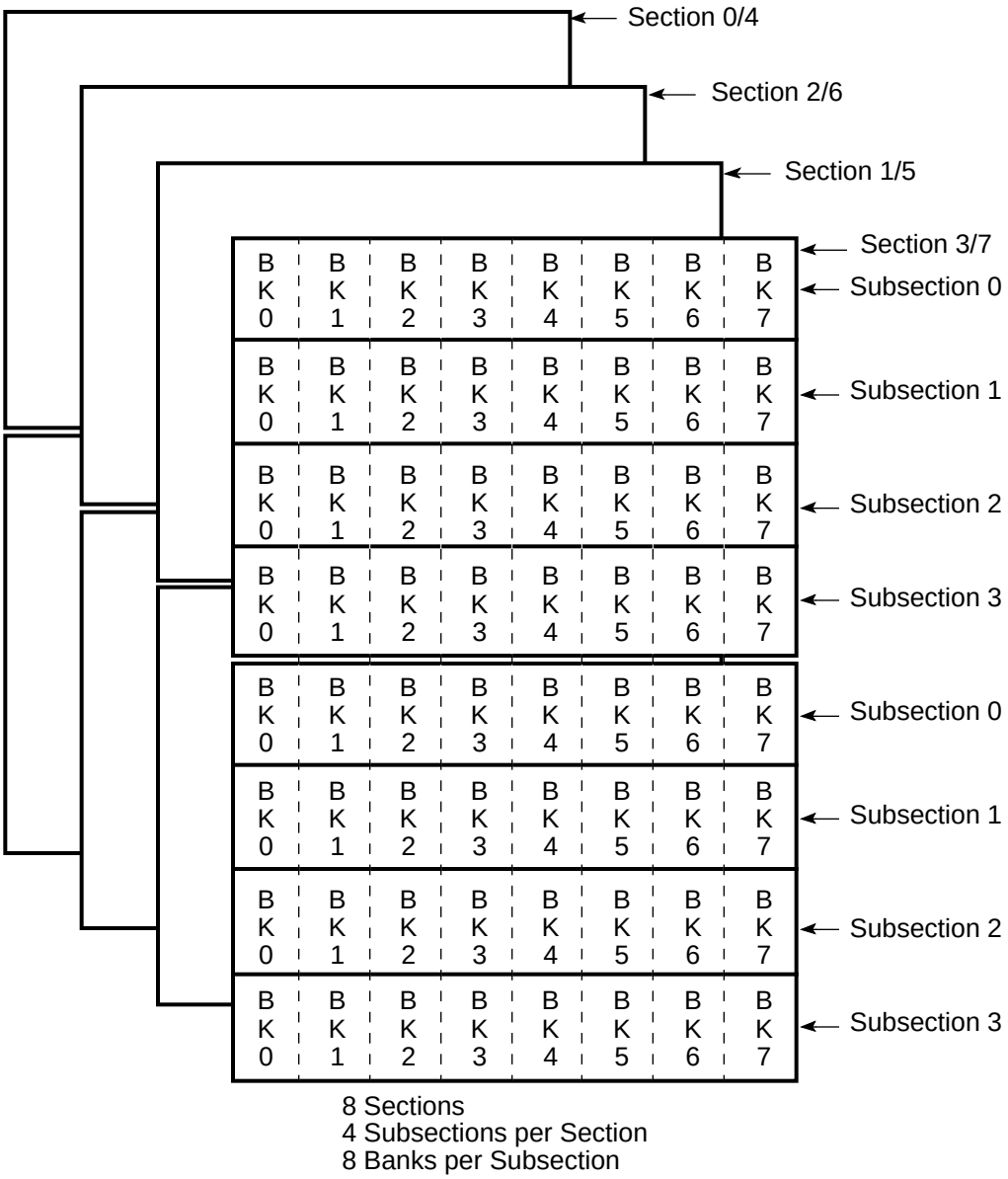


Figure 8. Memory Module Layout - 4 X 4 Half Populated

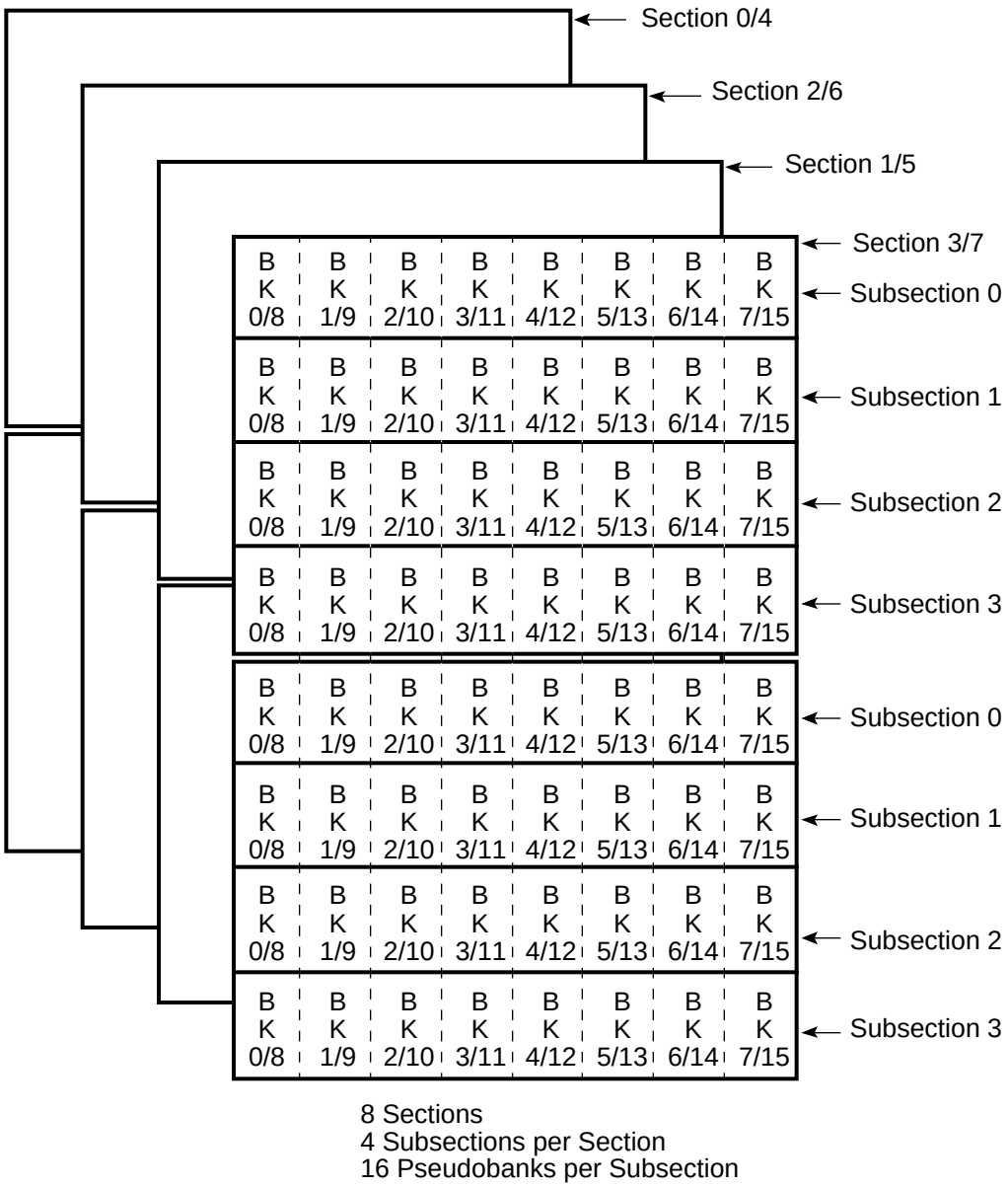


Figure 9. Memory Module Layout - 4 X 4 Fully Populated

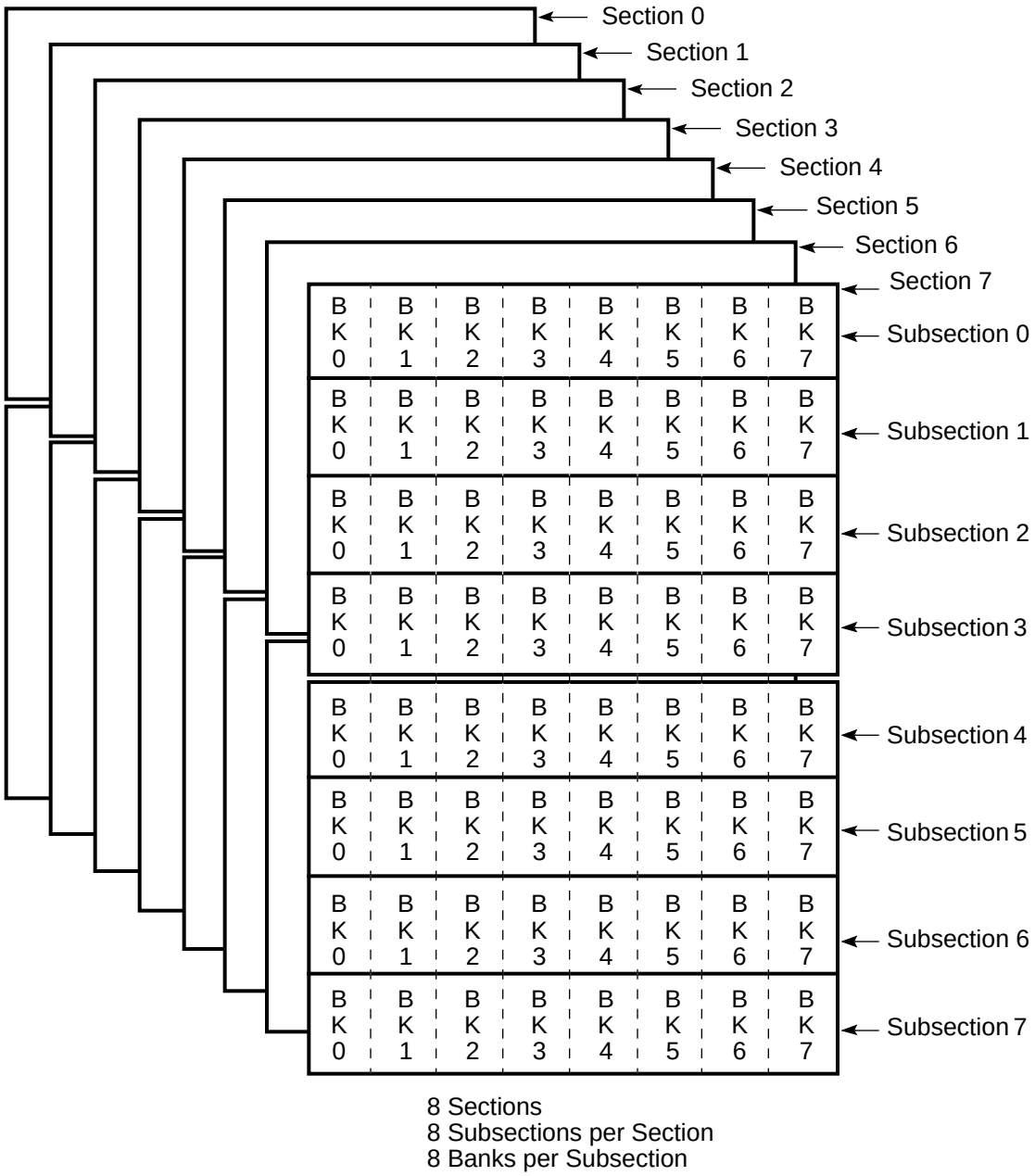


Figure 10. Memory Module Layout - 8 X 8 Half Populated

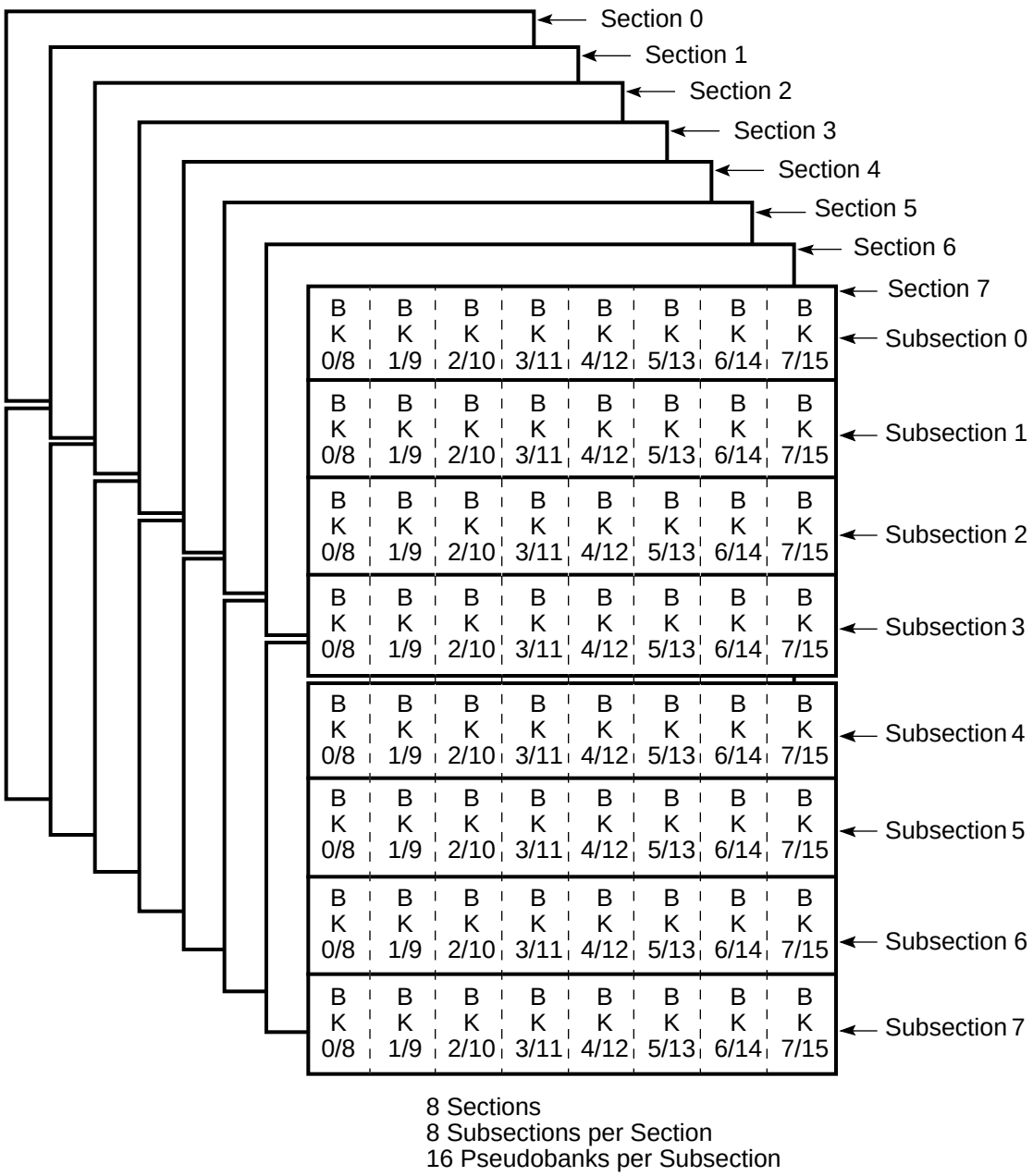


Figure 11. Memory Module Layout - 8 X 8 Fully Populated

Refer to Table 5 for the memory configurations available for 2 X 2, 4 X 4, and 8 X 8 systems.

Table 5. Memory Configurations

Backplane Configuration	Number of Processor Modules	Number of CPUs	Number of Memory Modules	Memory Sizes 256K x 16 DRAMs		Memory Sizes 1M x 16 DRAMs	
				MEM16	MEM32	MEM64	MEM128
2 x 2	1 or 2	4 or 8	2	32 MW	64 MW	128 MW	256 MW
4 x 4	1, 2, 3, or 4	4, 8, 12, or 16	4	64 MW	128 MW	256 MW	512 MW
8 x 8	4, 5, 6, or 7	16, 20, 24, or 32	8	128 MW	256 MW	512 MW	1024 MW

Memory Module Construction

A memory module is approximately 16 in. X 19.5 in. X 0.16 in. and comprises 26 metal layers: 12 signal layers with buried vias, 12 power/ground layers, and 2 surface layers. Each memory module contains 64 dual in-line memory (DIM) modules. Each DIM module, which is a custom daughter card measuring 1 in. X 6 in., contains ten DRAM chips in a fully populated system. A half-populated memory system contains five DRAM chips. A 132-pin header mounts the DIM modules on-edge to the memory module. The DIM modules are soldered to the memory module and are not field replaceable. Refer to Figure 13 for an illustration of the DIM module.

The aluminum framework that surrounds the memory module provides mechanical support during engagement and alignment of the modules in the backplane. This framework also provides support for all the components on the module. Aluminum covers on the front and back of the module direct airflow.

Each memory module has its own power supply in the form of a logic power module (LPM) located on the back side of the printed circuit board. The LPM receives 48-Vdc power and converts it to the closely regulated 5.0-Vdc power needed by the memory DRAMs and 3.3-Vdc power for the ASICs. The power module provides n + 1 capabilities for improved reliability. The power module also provides several local control functions such as voltage margining, local inhibit, and air-stream overtemperature sensing. Refer to the *Power, Cooling, and Control* document for more information on power.

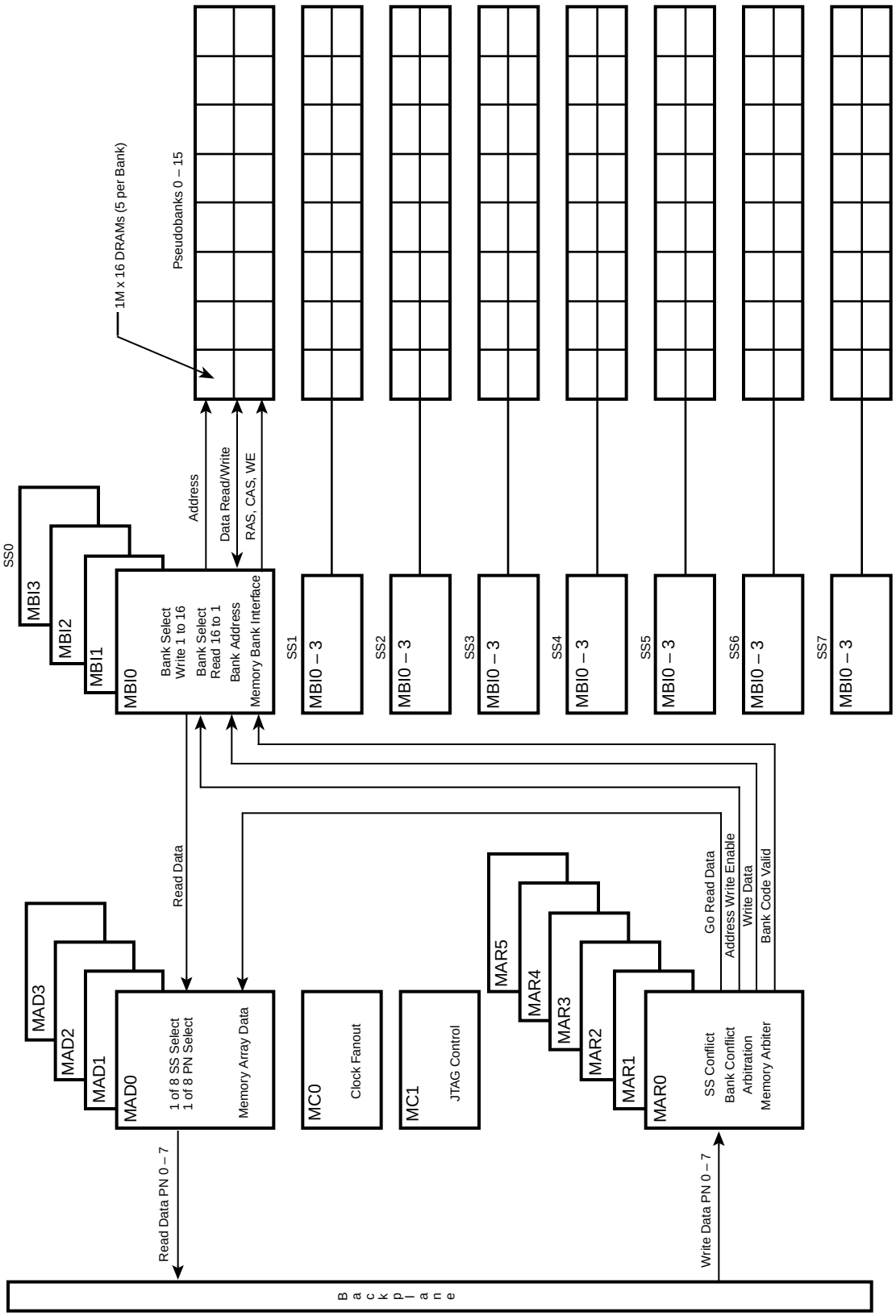


Figure 12. Memory Module Block Diagram

Memory ASIC Descriptions

A memory module contains the following four types of ASICs, for a total of 44 ASICs per memory module: 6 memory arbiter (MAR) ASICs, 4 memory array data (MAD) ASICs, 32 memory bank interface (MBI) ASICs, 1 maintenance and clock fanout (MC0) ASIC, and 1 maintenance and clock for JTAG control (MC1) ASIC. Refer to Figure 13 for a diagram of the memory module ASIC layout.

The MAR ASICs

- Buffer incoming requests
- Queue requests to their target subsections
- Track bank busy and bank access times
- Drive references at appropriate times
- Control the MAD ASICs

NOTE: CRAY J932 systems use a modified MAR ASIC that makes CRAY J932 memory modules incompatible with CRAY J916 memory modules.

The MAD ASICs

- Steer the 72-bit read data
- Control the order in which data is sent to the VB ASICs on the processor modules

The MBI ASICs

- Send write data and addresses from the MAR ASICs to the DRAMs
- Send read data addresses from the MAR ASICs to the DRAMs and steer the read data from the DRAMs to the MAD ASICs

The MC0 ASIC fans out the clock signal to all ASICs on the memory module.

The MC1 ASIC provides the JTAG, boundary scan, stop clock, and reset maintenance functions for the memory module.

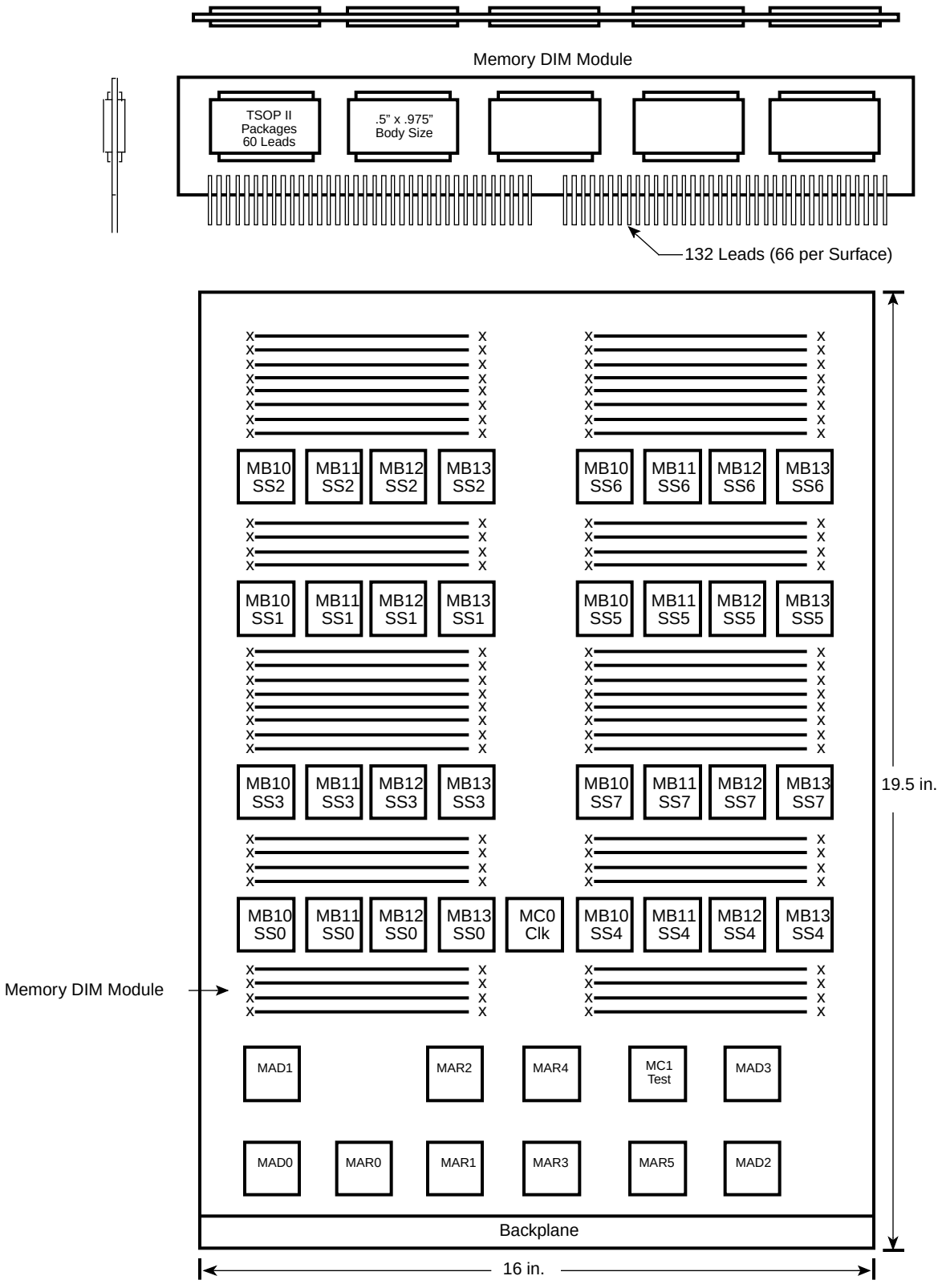
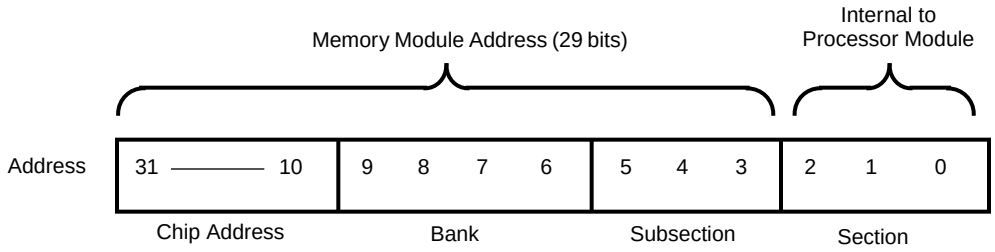


Figure 13. Memory Module ASIC Layout

Memory Addressing

There are 32 absolute address bits, but only 29 address lines are carried onto the backplane. Absolute address bits 0 through 2 select the memory section; bits 3 through 5 select the subsection; bits 6 through 9 select the bank; and bits 10 through 31 select the chip address. Figure 14 shows the four memory address fields. Table 6 lists memory addressing information for the 2 X 2, 4 X 4, and 8 X 8 systems. Refer again to Figure 8 through Figure 11 for the memory sections on each memory module.



NOTE: Subsection bits 4 and 5 are not used in a 2 X 2 system.

Figure 14. Memory Address Bits

Table 6. Memory Addressing

Backplane Configuration	Memory Sections	Memory Module
2 X 2	0, 2, 4, 6 1, 3, 5, 7	0 1
4 X 4	0, 4 1, 5 2, 6 3, 7	0 1 2 3
8 X 8	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7

The memory addressing scheme uses a rotating priority through the 8 sections of memory with 2-section spacing between the slots. Each slot has highest priority to 1 memory section each CP. This is called the slot's *natural* priority. A natural slot priority that is not in use may be *borrowed* by another slot. A

borrowing priority exists for the three *non-natural* slots. I/O operations are unslotted and may use any available slot. I/O priority is configured from lowest to highest priority, depending on system requirements. All read and write requests to memory are handled on a slot basis. Ports A and B of CPU 0 share slot 0; ports A and B of CPU 1 share slot 1; ports A and B of CPU 2 share slot 2; and ports A and B of CPU 3 share slot 3, etc. Refer to the *CRAY J90 Series System Programmer Reference Manual*, Cray Research publication number CSM-0301-000, for more information on memory addressing.

Memory Paths

Each processor module has an independent path into each memory section. Figure 15 shows the central memory architecture. The 4 CPUs and the I/O on a processor module share these eight paths. Each of the eight paths is capable of sending one request to memory per CP and receiving read data from memory at the same rate. Each CPU can have overlapping references in different sections without restrictions. Simultaneous references from a processor module to the same section are not permitted because only one physical path into each memory section exists for requests and write data.

A rotating priority scheme gives the 4 CPUs on a processor module equal access to each section of memory. All memory references for a CPU are sent to each memory section in the proper order. However, no order is guaranteed for memory references between the various CPUs in the system. Each memory section buffers the requests as required by bank busy signals and requires activity from all CPUs in the system. A memory section guarantees order for a request from a single CPU but not for requests between CPUs.

Memory Ports

Each CPU has two ports: port A and port B. Each port has a specific function that is defined jointly by the read mode bits and the port bits in the exchange package. Ports A and B are both read and write ports, but they allow only one write operation to be active at a time. However, both ports may process read operations simultaneously. Also, a read operation may occur on one port while a write operation occurs on the other port, if the bidirectional mode bit (BDM) is set in the exchange package. A third port, port D, is used for I/O and instruction fetch operations.

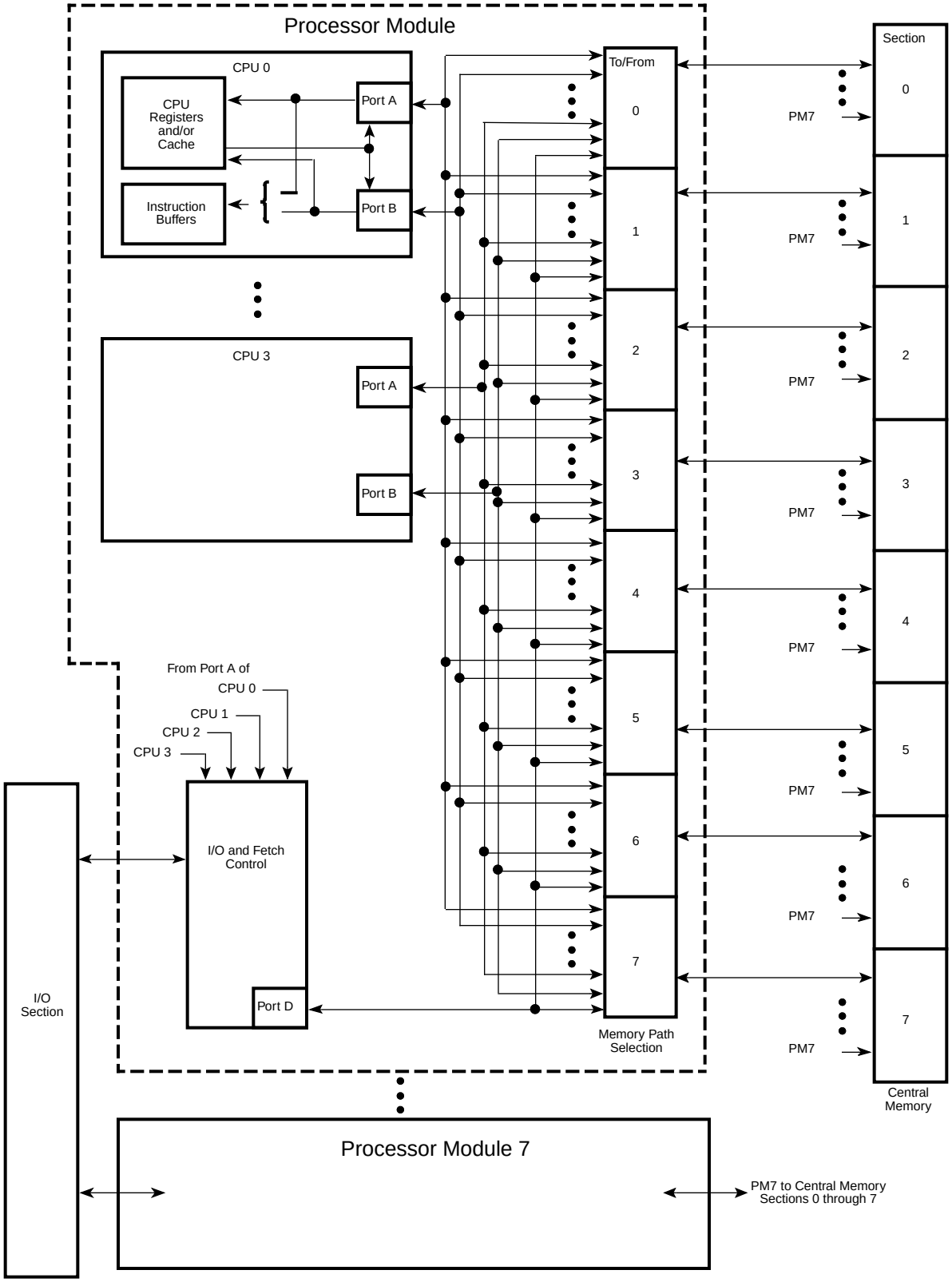


Figure 15. CPU Central Memory Architecture

System Clock

The CRAY J90 series system uses two types of maintenance and clock (MC0) ASICs: the MC0 ASIC located on the master clock board, and the MC0′ ASIC located on all processor and memory modules in the system. The crystal oscillator on the master clock board distributes the 25-MHz (40-ns) system reference clock to the MC0 ASIC, which fans out the clock signal to the MC0′ ASIC on all modules (CPU, memory, and clock). The MC0′ ASIC located on each module fans out the system reference clock to all ASICs on the module. The phase lock loop (PLL) on each ASIC multiplies the 25-MHz system reference clock by 4 (4 X 25 MHz = 100 MHz or 10 ns). The PLL, which is called the *on-chip clock*, is fanned out to all flip-flops within the ASIC.

The crystal oscillator on the master clock board generates the nominal clock period. When clock margins are run, an external clock is brought onto the master clock board through an external port. The onboard crystal oscillator is turned off when this external port is used.

The master clock board also contains the MC1 ASIC, which provides stop clock functions. The master clock board connects to the CC ASIC on the paddle card of the channel adapter board by means of the console bus (conbus).

Boundary Scan

The boundary scan feature is a function of the MC1 (internal maintenance) ASIC. To perform a *boundary scan* means to test the input and output latches on an ASIC and the interconnections between the latches. The boundary scan test also checks the connectivity of the backplane. Boundary scan provides test access to device pins by associating a serial shift register element, or scan cell, with each signal pin. The boundary scan cells link to form a shift register chain around the device boundary. These scan cells are then used to control and observe the device pins.

The boundary scan IEEE standard was developed by the Joint Test Action Group (JTAG). This standard is a collection of design rules applied principally at the integrated circuit level. The primary benefit of this standard is its capability to transform difficult PCB-testing problems into well-structured problems that diagnostic software can analyze.

Refer to the *IOS Based Diagnostics* document, Cray Research publication number HDM-099-0, for more information about the CRAY J90 series boundary scan feature.

Appendix: Instruction Set

The following designators indicate the notational conventions for the CRAY J90 series instructions.

1 Privileged to monitor mode
2 Special CAL syntax
3 Not supported by CAL version 2
4 Generated depending on *exp*
( ) Read as *the contents of* . . .
† Bit 2 of the *i* field = 0

Field	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	Bit 63 = 1

Machine Instruction	CAL Syntax	Description
000000	ERR	Error exit
0010*jk*[1]	CA,A*j* A*k*	Set the CA register for the channel indicated by (A*j*) to (A*k*) and activate the channel.
001000	PASS	This is a no-operation instruction.
0011*jk*[1]	CL,A*j* A*k*	Set the CL register for the channel indicated by (A*j*) to (A*k*) address.
0012*j*0[1]	CI,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*); clear device master clear (output channels only).
0012*j*1[1]	MC,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*); set device master clear (output channels only); clear device ready-held (input channels only).
0013*j*0[1]	XA A*j*	Transmit (A*j*) to the XA register.
0014*j*0[1]	RT S*j*	Load the RTC register with (S*j*).
0014*j*1[1]	SIP A*j*	Send an interprocessor interrupt request to CPU (A*j*).
001401[1]	SIPI	Send an interprocessor interrupt request to CPU 0.
001402[1]	CIPI	Clear the interprocessor interrupt.
0014*j*3[1]	CLN A*j*	Load the CLN register with (A*j*).
0014*j*4[1]	PCI S*j*	Load the II register with (S*j*).
001405[1]	CCI	Clear the programmable clock interrupt request.
001406[1]	ECI	Enable the programmable clock interrupt request.

Machine Instruction	CAL Syntax	Description
001407[1]	DCI	Disable the programmable clock interrupt request.
0015*j*0[1, 3]		Select performance monitor.
001501[1, 3]		Disable port A error correction.
001511[1, 3]		Disable port B error correction.
001521[1, 3]		Disable port D I/O error correction.
001541[1, 3]		Enable replacement of checkbyte with data on ports for writes and the replacement of data with checkbytes on ports for reads.
001551[1, 3]		Replace check bits with V*k* data bits on the path to the VA ASIC during execution of instruction 1771*jk*.
0016*j*1[2]	IVC	Send invalidate cache request to CPU (A*j*).
00200*k*	VL A*k*	Transmit (A*k*) to VL register.
002000[2]	VL 1	Transmit 1 to VL register.
002100	EFI	Enable interrupt on floating-point error.
002200	DFI	Disable interrupt on floating-point error.
002300	ERI	Enable interrupt on operand range error.
002400	DRI	Disable interrupt on operand range error.
002500	DBM	Disable bidirectional memory transfers.
002600	EBM	Enable bidirectional memory transfers.
002700	CMR	Complete memory references.
0030*j*0	VM S*j*	Transmit (S*j*) to VM register.
003000[2]	VM 0	Clear VM register.
0034*jk*	SM*jk* 1, TS	Test and set semaphore *jk*, 0 <*jk* <31_{10}.
0036*jk*	SM*jk* 0	Clear semaphore *jk*, 0 <*jk* <31_{10}.
0037*jk*	SM*jk* 1	Set semaphore *jk*, 0 <*jk* <31_{10}.
004000	EX	Normal exit from the operating system.
0050*jk*	J B*jk*	Jump to (B*jk*).
006*ijkm*	J *exp*	Jump to *exp*.
007*ijkm*	R *exp*	Return jump to *exp* and set register B00 to (P) + 2.
† 010*ijkm*	JAZ *exp*	Jump to *exp* if (A0) = 0 (i_2 = 0).
† 011*ijkm*	JAN *exp*	Jump to *exp* if (A0) ≠ 0 (i_2 = 0).
† 012*ijkm*	JAP *exp*	Jump to *exp* if (A0) positive; (A0) ≥ 0 (i_2 = 0).
† 013*ijkm*	JAM *exp*	Jump to *exp* if (A0) negative (i_2 = 0).

Machine Instruction	CAL Syntax	Description
† 014*ijkm*	JSZ *exp*	Jump to *exp* if (S0) = 0 (i_2 = 0).
† 015*ijkm*	JSN *exp*	Jump to *exp* if (S0) ≠ 0 (i_2 = 0).
† 016*ijkm*	JSP *exp*	Jump to *exp* if (S0) positive; (i_2 = 0).
† 017*ijkm*	JSM *exp*	Jump to *exp* if (S0) negative (i_2 = 0)
020*i*00*mn*[4] or 021*i*00*mn* or 022*ijk*[4]	A*i exp*	Transmit *exp* into A*i* (020 or 022) or transmit one's complement of *exp* into A*i* (021).
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*.
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*7	A*i* SB*j*	Transmit (SB*j*) to A*i*.
027*ij*0	A*i* ZS*j*	Transmit leading zero count of (S*j*) to A*i*.
027*ij*7	SB*j* A*i*	Transmit (A*i*) to SB*j*.
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*[2]	A*i* A*k*	Transmit (A*k*) to A*i*.
030*ij*0[2]	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 (A*j*) and (A*k*) to A*i*.
031*i*00[2]	A*i* -1	Transmit -1 to A*i*.
031*i*0*k*[2]	A*i* -A*k*	Transmit the negative of (A*k*) to A*i*.
031*ij*0[2]	A*i* -A*j*-1	Transmit the integer difference (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*.
033*i*00	A*i* CI	Transmit the channel number of the highest priority interrupt request to A*i* (*j* = 0).
033*ij*0	A*i* CA,A*j*	Transmit the current address of the channel (A*j*) to A*i* (*j* ≠ 0, *k* = 0).
033*ij*1	A*i* CE,A*j*	Transmit the error flag of channel (A*j*) to A*i* (*j* ≠ 0, *k* = 1).
034*ijk*	B*jk*, A*i*, ,A0	Load (A*i*) words from memory starting at address (A0) to B registers starting at register *jk*.
034*ijk*[2]	B*jk*,A*i* 0,A0	Load (A*i*) words from memory starting at address (A0) to B registers starting at register *jk*.

Machine Instruction	CAL Syntax	Description
035*ijk*	,A0 B*jk*,A*i*	Store (A*i*) words from B registers starting at register *jk* to memory starting at address (A0).
035*ijk*^{2}	0,A0 B*jk*,A*i*	Store (A*i*) words from B registers starting at register *jk* to memory starting at address (A0).
036*ijk*	T*jk*,A*i* 0,A0	Load (A*i*) words from memory starting at address (A0) to T registers starting at register *jk*.
036*ijk*^{2}	T*jk*,A*i* 0,A0	Load (A*i*) words from memory starting at address (A0) to T registers starting at register *jk*.
037*ijk*	,A0 T*jk*,A*i*	Store (A*i*) words from T registers starting at register *jk* to memory starting at address (A0).
037*ijk*^{2}	0,A0 T*jk*,A*i*	Store (A*i*) words from T registers starting at register *jk* to memory starting at address (A0).
040*i*00*mn* or 041*i*00*mn*	S*i* *exp*	Transmit *exp* into S*i* (040) or transmit one's complement of *exp* into S*i* (041).
042*ijk*	S*i* <*exp*	Form ones mask in S*i* *exp* bits from right; the *jk* field gets 100_8 - *exp*.
042*ijk*^{2}	S*i* # >*exp*	Form zeroes mask in S*i* *exp* bits from left; the *jk* field gets *exp*.
042*i*77^2	S*i* 1	Enter 1 into S*i* register.
042*i*00^2	S*i* -1	Enter -1 into S*i* register.
043*ijk*	S*i* >*exp*	Form ones mask in S*i* *exp* bits from left; the *jk* field gets *exp*.
043*ijk*^{2}	S*i* #<*exp*	Form zeroes mask in S*i* *exp* bits from right; the *jk* field gets 100_8 *exp*.
043*i*00^2	S*i* 0	Clear the S*i* register.
044*ijk*	S*i* S*j*&S*k*	Transmit the logical product of (S*j*) and (S*k*) to S*i*.
044*ij*0^2	S*i* S*j*&SB	Transmit the sign bit of (S*j*) to S*i*.
044*ij*0^2	S*i* SB&S*j*	Transmit the sign bit of (S*j*) to S*i* (*j* ≠ 0).
045*ijk*	S*i* #S*k*&S*j*	Transmit the logical product of (S*j*) and complement of (S*k*) to S*i*.
045*ij*0^2	S*i* #SB&S*j*	Transmit the (S*j*) with sign bit cleared to S*i*.
046*ijk*	S*i* S*j*\S*k*	Transmit the logical difference of (S*j*) and (S*k*) to S*i*.
046*ij*0^2	S*i* S*j*\SB	Toggle the sign bit of (S*j*), then enter into S*i*.
046*ij*0^2	S*i* SB\S*j*	Toggle the sign bit of (S*j*), then enter into S*i* (*j* ≠ 0).
047*ijk*	S*i* #S*j*\S*k*	Transmit the logical equivalence of (S*k*) and (S*j*) to S*i*.
047*i*0*k*^{2}	S*i* #S*k*	Transmit the one's complement of (S*k*) to S*i*.

Machine Instruction	CAL Syntax	Description
047*ij*0^2	S*i* #S*j*\SB	Transmit the logical equivalence of (S*j*) and sign bit to S*i*.
047*ij*0^2	S*i* #SB\S*j*	Transmit the logical equivalence of (S*j*) and sign bit to S*i* (*j* ≠ 0).
047*i*00^2	S*i* #SB	Transmit the one's complement of sign bit into 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*.
050*ij*0^2	S*i* S*j*!S*i*&SB	Transmit the scalar merge of (S*i*) and sign bit of (S*j*) to S*i*.
051*ijk*	S*i* S*j*!S*k*	Transmit the logical sum of (S*j*) and (S*k*) to S*i*.
051*i*0*k*^{2}	S*i* S*k*	Transmit the (S*k*) to S*i*.
051*ij*0^2	S*i* S*j*!SB	Transmit the logical sum of (S*j*) and sign bit to S*i*.
051*ij*0^2	S*i* SB!S*j*	Transmit the logical sum of (S*j*) and sign bit to S*i* (*j* ≠ 0).
051*i*00^2	S*i* SB	Transmit the sign bit into S*i*.
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*.
056*ijk*	S*i* S*i*,S*j*<A*k*	Shift (S*i*) and (S*j*) left by (A*k*) places to S*i*.
056*ij*0^2	S*i* S*i*,S*j*<1	Shift (S*i*) and (S*j*) left one place to S*i*.
056*i*0*k*^{2}	S*i* S*i*<A*k*	Shift (S*i*) left (A*k*) places to S*i*.
057*ijk*	S*i* S*j*,S*i*>A*k*	Shift (S*j*) and (S*i*) right by (A*k*) places to S*i*.
057*ij*0^2	S*i* S*j*,S*i*>1	Shift (S*j*) and (S*i*) right one place to S*i*.
057*i*0*k*^{2}	S*i* S*i*>A*k*	Shift (S*i*) right (A*k*) places to S*i*.
060*ijk*	S*i* S*j*+S*k*	Transmit the integer sum of (S*j*) and (S*k*) 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*^{2}	S*i* -S*k*	Transmit the negative of (S*k*) to S*i*.
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*^{2}	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*^{2}	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*.

Machine Instruction	CAL Syntax	Description
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**!S*k*	Transmit the reciprocal iteration: 2-(S*j*) to S*i*.
070*ij*0	S*j* /HS*j*	Transmit the floating-point reciprocal approximation of (S*j*) to S*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 unnormalized floating-point number.
071*i*30	S*i* 0.6	Transmit 0.75×2^{48} as normalized floating-point constant into S*i*.
071*i*40	S*i* 0.4	Transmit 0.5 as normalized floating-point constant into S*i*.
071*i*50	S*i* 1.0	Transmit 1.0 as normalized floating-point constant into S*i*.
071*i*60	S*i* 2.0	Transmit 2.0 as normalized floating-point constant into S*i*.
071*i*70	S*i* 4.0	Transmit 4.0 as normalized floating-point constant into S*i*.
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*.
073*i*00	S*i* VM	Transmit (VM) to S*i*.
073*i*11[1, 3]		Read the performance counter into S*i*.
073*i*21[1, 3]		Increment upper performance counter.
073*i*31[1, 3]		Clear all maintenance modes.
073*i*61[1, 3]		Increment current performance counter (lower).
073*i*01	S*i* SR0	Transmit (SR0) to S*i*.
073*i*02	SM S*i*	Transmit (S*i*) to SM.
073*ij*3	ST*j* S*i*	Transmit (S*i*) to ST*j*.
074*ijk*	S*i* T*jk*	Transmit (T*jk*) to S*i*.
075*ijk*	T*jk* S*i*	Transmit (S*i*) to T*jk*.
076*ijk*	S*i* V*j*,A*k*	Transmit (V*j* element (A*k*)) to S*i*.
077*ijk*	V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*).
077*i*0*k*[2]	V*i*,A*k* 0	Clear element (A*k*) of register V*i*.
10*hi*00*mn*	A*i* *exp*,A*h*	Load from ((A*h*) + *exp*) to A*i*.

Machine Instruction	CAL Syntax	Description
100*i*00*mn*	A*i exp*,0	Load from (*exp*) to A*i*.
100*i*00*mn*	A*i exp*,	Load from (*exp*) to A*i*.
10*hi*0000	A*i* ,A*h*	Load from (A*h*) to A*i*.
11*hi*00*mn*	*exp*,A*h* A*i*	Store (A*i*) to (A*h*) + *exp*.
110*i*00*mn*	*exp*,0 A*i*	Store (A*i*) to *exp.*
110*i*00*mn*	*exp*, A*i*	Store (A*i*) to *exp*.
11*hi*0000	,A*h* A*i*	Store (A*i*) to (A*h*).
12*hi*00*mn*	S*i exp*,A*h*	Load from ((A*i*) + *exp*) to S*i*.
120*i*00*mn*	S*i exp*,0	Load from (*exp*) to S*i*.
120*i*00*mn*	S*i exp*	Load from (*exp*) to S*i*.
12*hi*0000	S*i* ,A*h*	Load from (A*h*) to S*i*.
13*hi*00*mn*	*exp*,A*h* S*i*	Store (S*i*) to (A*h*) + *exp*.
130*i*00*mn*	*exp*,0 S*i*	Store (S*i*) to *exp*.
130*i*00*mn*	*exp*, S*i*	Store (S*i*) to *exp*.
13*hi*0000	,A*h* S*i*	Store (S*i*) to (A*h*).
140*ijk*	V*i* S*j*&V*k*	Transmit logical products of (S*j*) and (V*k* elements) to V*i* elements.
141*ijk*	V*i* V*j*&V*k*	Transmit logical products of (V*j* elements) and (V*k* elements) to V*i* elements.
142*ijk*	V*i* S*j*!V*k*	Transmit logical sums of (S*j*) and (V*k* elements) to V*i* elements.
142*i*0*k*^{2}	V*i* V*k*	Transmit (V*k* elements) to V*i* elements.
143*ijk*	V*i* V*j*!V*k*	Transmit logical sums of (V*j* elements) and (V*k* elements) to V*i* elements.
144*ijk*	V*i* S*j* V*k*	Transmit logical differences of (S*j*) and (V*k* elements) to V*i* elements.
145*ijk*	V*i* V*j* V*k*	Transmit logical differences of (V*j* elements) and (V*k* elements) to V*i* elements.
145*iii*^{2}	V*i* 0	Clear V*i* elements.
146*ijk*	V*i* S*j*!V*k*&VM	Transmit (S*j*) if VM bit = 1; (V*k*) if VM bit = 0 to V*i*.
146*i*0*k*^{2}	V*i* #VM&V*k*	Transmit vector merge of (V*k*) and 0 to V*i*.
147*ijk*	V*i* V*j*!V*k*&VM	Transmit (V*j*) if VM bit = 1; (V*k*) if VM bit = 0 to V*i*.
150*ijk*	V*i* V*j*<A*k*	Shift (V*j* elements) left by (A*k*) places to V*i* elements.
150*ij*0^2	V*i* V*j*<1	Shift (V*j* elements) left one place to V*i* elements.
151*ijk*	SV*i* V*j*>A*k*	Shift (V*j* elements) right by (A*k*) places to V*i* elements.

Machine Instruction	CAL Syntax	Description
151*ij*0[2]	V*i* V*j*>1	Shift (V*j* elements) right one place to V*i* elements.
152*ijk*	V*i* V*j*,V*j*<A*k*	Double shift of (V*j* elements) left (A*k*) places to V*i* elements.
152*ij*0[2]	V*i* V*j*,V*j*<1	Double shift of (V*j* elements) left one place to V*i* elements.
153*ijk*	V*i* V*j*,V*j*>A*k*	Double shift of (V*j* elements) right (A*k*) places to V*i* elements.
153*ij*0[2]	V*i* V*j*,V*j*>1	Double shift of (V*j* elements) right one place to V*i* elements.
154*ijk*	V*i* S*j*+V*k*	Transmit integer sums of (S*j*) and (V*k* elements) to V*i* elements.
155*ijk*	V*i* V*j* + V*k*	Transmit integer sums of (V*j* elements) and (V*k* elements) to V*i* elements.
156*ijk*	V*i* S*j*-V*k*	Transmit integer differences of (S*j*) and (V*k* elements) to V*i* elements.
156*i*0*k*[2]	V*i* -V*k*	Transmit two's complement of (V*k* elements) to V*i* elements.
157*ijk*	V*i* V*j*-V*k*	Transmit integer differences of (V*j* elements) and (V*k* elements) to V*i* elements.
160*ijk*	V*i* S*j**FV*k*	Transmit floating-point products of (S*j*) and (V*k* elements) to V*i* elements.
161*ijk*	V*i* V*j**FV*k*	Transmit floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements.
162*ijk*	V*i* S*j**HV*k*	Transmit 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 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 rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements.
165*ijk*	V*i* V*j**RV*k*	Transmit 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 32-bit integer product of (S*j*) and (V*k* elements) to V*i* elements.
167*ijk*	V*i* V*j**V*k*	Transmit reciprocal iterations: 2-(V*j* elements)*(V*k* elements) to V*i* elements.
170*ijk*	V*i* S*j*+FV*k*	Transmit floating-point sums of (S*j*) and (V*k* elements) to V*i* elements.
170*i*0*k*[2]	V*i* +FV*k*	Transmit normalized (V*k* elements) to V*i* elements.
171*ijk*	V*i* V*j*+FV*k*	Transmit floating-point sums of (V*j* elements) and (V*k* elements) to V*i* elements.

Machine Instruction	CAL Syntax	Description
172*ijk*	V*i* S*j*-FV*k*	Transmit floating-point differences of (S*j*) and (V*k* elements) to V*i* elements.
172*i*0*k*[2]	V*i* -FVk	Transmit normalized negative of (V*k* elements) to V*i* elements.
173*ijk*	V*i* V*j*-FV*k*	Transmit floating-point differences of (V*j* elements) and (V*k* elements) to V*i* elements.
174*ij*0	V*i* /HV*j*	Transmit floating-point reciprocal approximation of (V*j* elements) to V*i* elements.
174*ij*1	V*i* PV*j*	Transmit population count of (V*j* elements) to V*i* elements.
174*ij*2	V*i* QV*j*	Transmit population count parity of (V*j* elements) 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 (V*j* is negative).
175*ij*4	V*i*, VM V*j*,Z	Set VM bit if (V*j* elements) = 0; also, the compressed indices of the V*j* element = 0 are stored in V*i*.
175*ij*5	V*i*, VM V*j*,N	Set VM bit if (V*j* elements) ≠ 0; also, the compressed indices of the V*j* element ≠ 0 are stored in V*i*.
175*ij*6	V*i*, VM V*j*,P	Set VM bit if (V*j* elements) ≥ 0; also, the compressed indices of the V*j* element ≥ 0 are stored in V*i*.
175*ij*7	V*i*, VM V*j*,M	Set VM bit if (V*j* elements) ≤ 0; also, the compressed indices of the V*j* element ≤ 0 are stored in V*i*.
176*i*0*k*	V*i* ,A0,A*k*	Load from memory starting at (A0) increased by (A*k*) and load into V*i*.
176*i*00	V*i* ,A0,1	Load from consecutive memory addresses starting with (A0) and load into V*i*.
176*i*1*k*	V*i* ,A0,V*k*	Load from memory using memory address ((A0) + (V*k*)) and load into V*i*.
1770*jk*	,A0,A*k* V*j*	Store (V*j*) to memory starting at (A0) increased by (A*k*).
1770*j*0	,A0,1 V*j*	Store (V*j*) to memory in consecutive addresses starting with (A0).
1771*jk*	,A0,V*k* V*j*	Store (V*j*) to memory using memory address ((A0) + (V*k*)).