
VAX 6000-400

VAX 6000–400 System Technical User's Guide

Order Number: EK–640EB–TM–002

This manual serves as a reference on how to write software to this machine and covers the information needed to do field-level repair or programming customized to the CPU. It includes information on interrupts, error handling, and detailed theory of operation.

Digital Equipment Corporation

First Printing, July 1989
Revised, July 1990

Printed in U.S.A.

The READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEBNA	PDP	VAXcluster
DEC	ULTRIX	VAXELN
DEC LANcontroller	UNIBUS	VMS
DECnet	VAX	XMI
DECUS	VAXBI	[redacted]™

This document was prepared using VAX DOCUMENT, Version 1.1

Contents

TABLES

Preface

Intended Audience

This manual is written for Digital customer service engineers installing and repairing in the field and for OEMs who are writing specialized applications, such as their own operating systems.

Document Structure

This manual has seven chapters.

- **Chapter 1** gives you a basic introduction to the VAX 6000–400 system and its parts.
- **Chapter 2** tells you about the XMI bus and protocol.
- **Chapter 3** explains the KA64A CPU module.
- **Chapter 4** explains the FV64A vector processor module.
- **Chapter 5** explains the MS62A memory module.
- **Chapter 6** tells you about the DWMBA and its DWMBA/A module and DWMBA/B module.
- **Chapter 7** explains the components of the power system and the cooling system.
- The **Index** provides additional reference support.

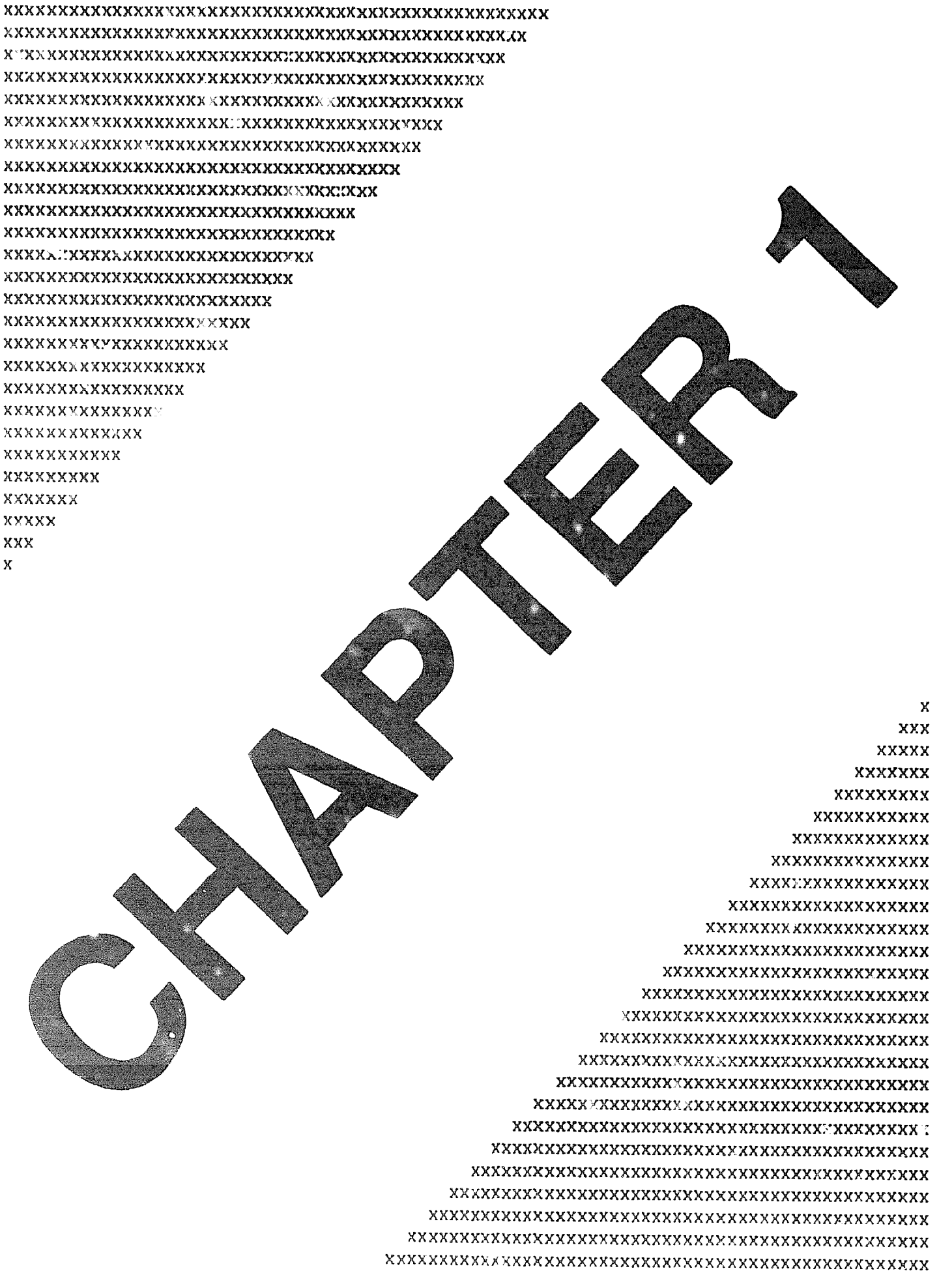
Associated Documents

Documents in the VAX 6000–400 documentation set include:

Title	Order Number
VAX 6000–400 Installation Guide	EK-640EA-IN
VAX 6000–400 Owner's Manual	EK-640EAA-OM
VAX 6000 Series Vector Processor Owner's Manual	EK-60VAA-OM
VAX 6000 Series Vector Processor Programmer's Guide	EK-60VAA-PG
VAX 6000–400 Mini-Reference	EK-640EA-HR
VAX 6000–400 Options and Maintenance	EK-640EB-MG
VAX 6000 Series Upgrade Manual	EK-600EB-UP

Other documents that you may find useful include:

Title	Order Number
VAX Architecture Reference Manual	EY-3459E-DP
VAX Vector Processing Handbook	EC-H0739-46
VAX Systems Hardware Handbook—VAXBI Systems	EB-31692-46
VAXBI Options Handbook	EB-32255-46
TK70 Streaming Tape Drive Owner's Manual	EK-OTK70-OM


CHAPTER 1

1 The VAX 6000–400 System Overview

This chapter describes the system packages and system components and notes the location of components in the cabinet.

This chapter includes the following sections:

- VAX 6000–400 Introduction
- VAX 6000–400 Configurations
- VAX 6000–400 System Architecture
- Typical System
- VAX 6000–400 System (Front View)
- VAX 6000–400 System (Rear View)
- Supported VAXBI Adapters and Options
- XMI Backplane and Card Cage
- VAXBI Backplane and Card Cage
- VAXBI Expander Cabinet
- TK Tape Drive
- Standard I/O Connections
- I/O Bulkhead Connections
- Power System
- Cooling System

1.1 VAX 6000–400 Introduction

The VAX 6000–400, a general purpose computer system designed for growth, can be configured for many different applications. Like other VAX systems, the VAX 6000–400 can support many users in a time-sharing environment.

The VAX 6000–400 does the following:

- Supports a full set of VAX applications
- Functions as a stand-alone system, a member of a VAXcluster, as a boot node of a local area VAXcluster, or as a VAX file server for workstations
- Allows for expansion of processors, memory, and I/O
- Supports vector processing
- Implements symmetric multiprocessing where all processors have equal access to memory
- Uses the VAXBI bus (VAX Bus Interconnect) as the I/O interconnect
- Uses a high-bandwidth internal system bus designed for multiprocessing
- Performs automatic self-test on power-up, reset, reboot, or system initialization

1.2 VAX 6000–400 Configurations

The VAX 6000–400 system family has configuration packages that differ in the number of processors and amount of memory.

Refer to Digital's *Systems and Options Catalog* for the available configurations.

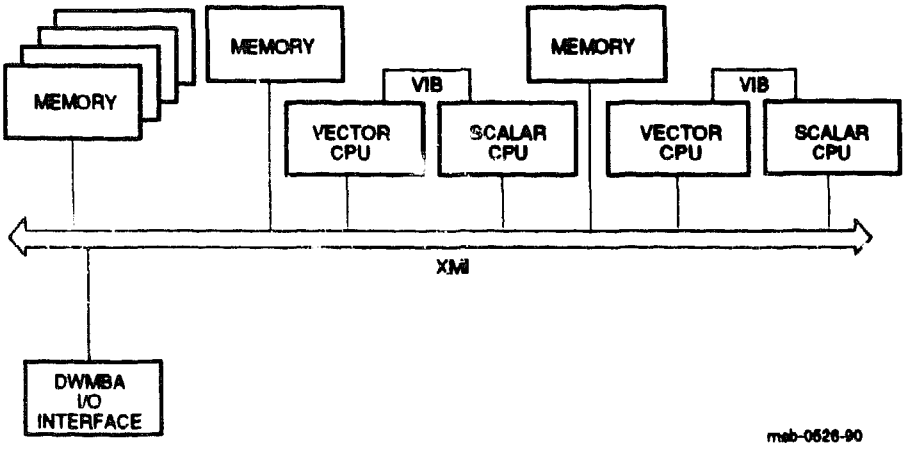
Each configuration has a 60-inch system cabinet that includes one 14-slot high-bandwidth internal system bus backplane (XMI) and two 6-slot VAXBI backplanes.

1.3 VAX 6000–400 System Architecture

The VAX 6000–400 uses a high-speed system bus, called the XMI bus, to interconnect its KA64A CPU module(s) and its MS62A memory module(s). The VAX 6000–400 supports multiprocessing with multiple KA64A CPU modules. Vector processing is also supported.

Figure 1–1 shows a system with two scalar/vector pairs.

Figure 1–1 VAX 6000–400 System Architecture



The XMI is the VAX 6000–400 system bus; the VAXBI bus supports the I/O subsystem. The XMI is a 64-bit system bus with a 64 nanosecond bus cycle, and a maximum throughput of 100 megabytes per second. The DWMBA interconnects the I/O adapters.

The VAXBI and XMI share similar but incompatible connector and module technology. Both the VAXBI and XMI buses have the concept of a node. On the VAXBI bus, a node may be more than one VAXBI module that operates as a single functional unit. On the XMI bus, a node is a single module that occupies one of the 14 logical and physical slots on the XMI bus.

The XMI has three types of nodes: processor nodes (KA64A CPU module), memory nodes (MS62A memory module), and I/O adapters (DWMBA).

A processor node, called a KA64A CPU module, is a single-board processor that contains a central processor unit (CPU) that executes instructions and manipulates data contained in memory; a floating-point processor; 128 Kbytes of onboard cache; and 2 Kbytes of on-chip cache. The VAX 6000–400 system supports up to six processors. Symmetric multiprocessing must also be supported by the operating system.

The KA64A CPU module supports an attached vector processor, the FV64A module.

The KA64A CPU module communicates with main memory over the XMI bus. A KA64A CPU module becomes the boot processor during power-up and that boot processor handles all system communication. The other CPUs become secondary processors and receive system information from the boot processor.

A memory node is an MS62A memory module. Memory is a global resource equally accessible to all processors on the XMI. Each MS62A memory module uses MOS 1-megabit dynamic RAMs, ECC logic, and control logic. The memories are automatically interleaved. An optional battery backup unit protects memory in case of power failure. The VAX 6000–400 system supports up to eight memories.

The DWMBA supports bidirectional communication between the XMI and the VAXBI. That is, from CPUs on the XMI to I/O options on the VAXBI and from I/O options on the VAXBI to memory modules on the XMI. Up to six DWMBA adapters are supported on a VAX 6000–400.

The DWMBA is a 2-board adapter. The DWMBA/A module is installed on the XMI bus, and it communicates with the DWMBA/B module on the VAXBI.

1.4 Typical System

A typical VAX 6000–400 system has a main cabinet, a console terminal and printer, cabinets, an accessories kit, and a set of documentation. It may have additional tape or disk drives and may be a member of a VAXcluster.

Figure 1–2 Typical VAX 6000–400 System

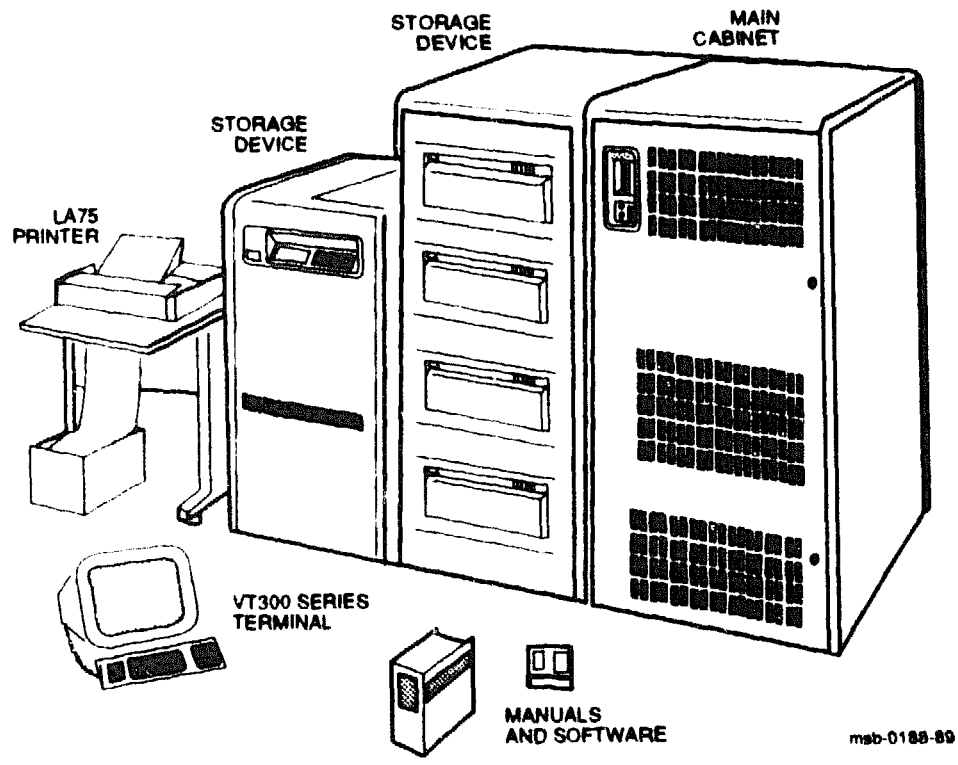


Table 1–1 Typical VAX 6000–400 System

Component	Function
Main cabinet	Houses system components
TK tape drive	Software distribution; stores and transfers data
Console terminal	Manages system and its resources
Console printer	Provides hardcopy of console transactions
System documentation	See *Preface* for full list of documentation related to the VAX 6000–400
Storage cabinets	Provides storage capacity

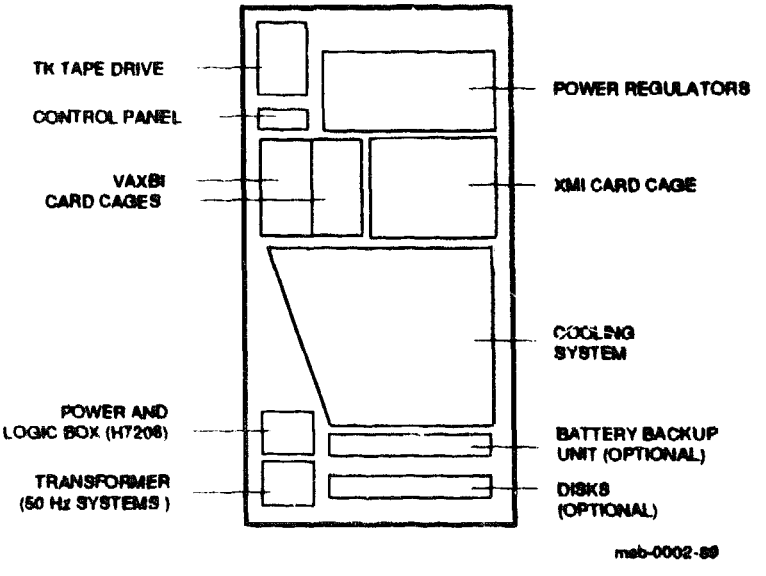
The VAX 6000–400 components include:

- The main cabinet which houses a TK tape drive, the XMI card cage, two VAXBI card cages, the control panel switches, status indicators, and restart controls.
- The TK tape drive, in the main cabinet, is used for installing operating systems, software, and diagnostics and may be used for data interchange.
- The storage cabinets are used for local storage.
- The console terminal is used for booting and for system management operations.
- VAX 6000–400 documentation includes:
  - *VAX 6000–400 Installation Guide*
  - *VAX 6000–400 Owner's Manual*
  - *VAX 6000–400 Mini-Reference*
  - *VAX 6000–400 Options and Maintenance*
  - *VAX 6000–400 System Technical User's Guide*
  - *VAX 6000 Series Upgrade Manual*
  - *VAX 6000 Series Vector Processor Owner's Manual*
  - *VAX 6000 Series Vector Processor Programmer's Guide*

1.5 VAX 6000–400 System (Front View)

The TK tape drive and control panel are on the front of the system cabinet. With the front door open, there is access to the system control panel, power regulators, VAXBI and XMI card cages, the cooling system, and battery backup unit and disks (if present).

Figure 1–3 VAX 6000–400 System (Front View)



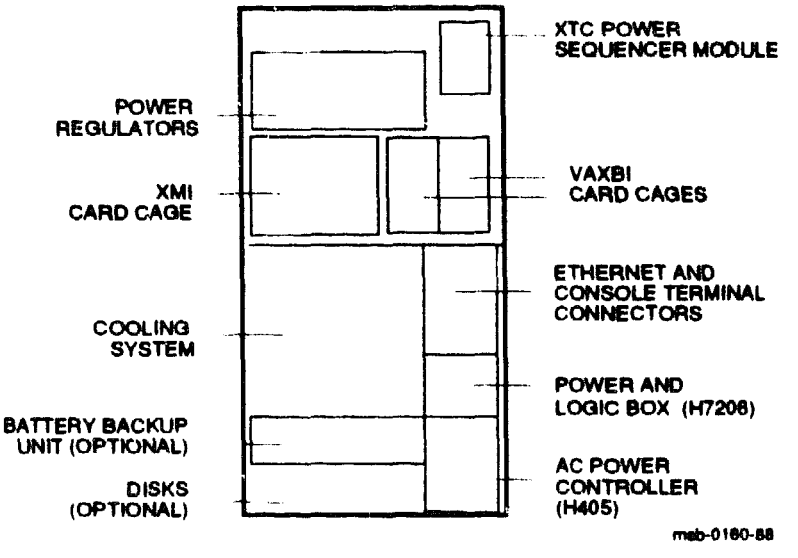
These components are visible from the inside front of the cabinet (see Figure 1–3 for their location):

- TK tape drive
- Control panel
- Power regulators
- Battery backup and disks (if installed)
- XMI card cage
- Two VAXBI card cages
- Cooling system

1.6 VAX 6000–400 System (Rear View)

With the rear door open, there is access to the following: XTC power sequencer, power regulators; the I/O bulkhead space behind the card cages; Ethernet and console terminal connectors; cooling system; power and logic box; battery backup unit and disks (if present); and the AC power controller.

Figure 1–4 VAX 6000–400 System (Rear View)



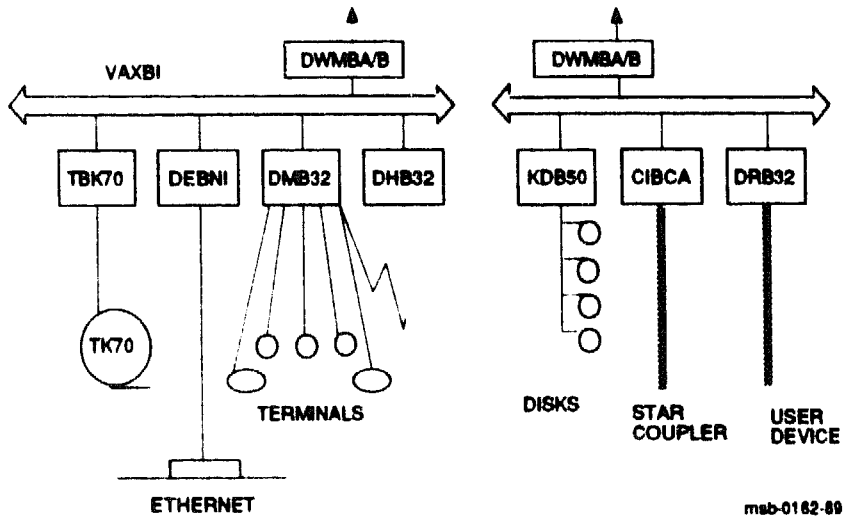
These components are visible from the rear of the cabinet (see Figure 1–4):

- TK tape drive and XTC power sequencer connections
- Five replaceable power regulators
- I/O bulkhead space (The I/O bulkhead panel covers the XMI and VAXBI areas.)
- Ethernet and console terminal connectors
- Cooling system, with open grid over a blower
- Power and logic unit (H7206)
- Battery backup unit and disks (optional)
- AC power controller (H405)

1.7 Supported VAXBI Adapters and Options

The VAX 6000–400 system supports the use of many different VAXBI adapters including CIBCA, DEBNA, DEBNI, DHB32, DMB32, DRB32, DSB32, DWMBA, KDB50, RBV20/RBV64, TBK50, TBK70, and TU81E.

Figure 1–5 VAXBI Adapters

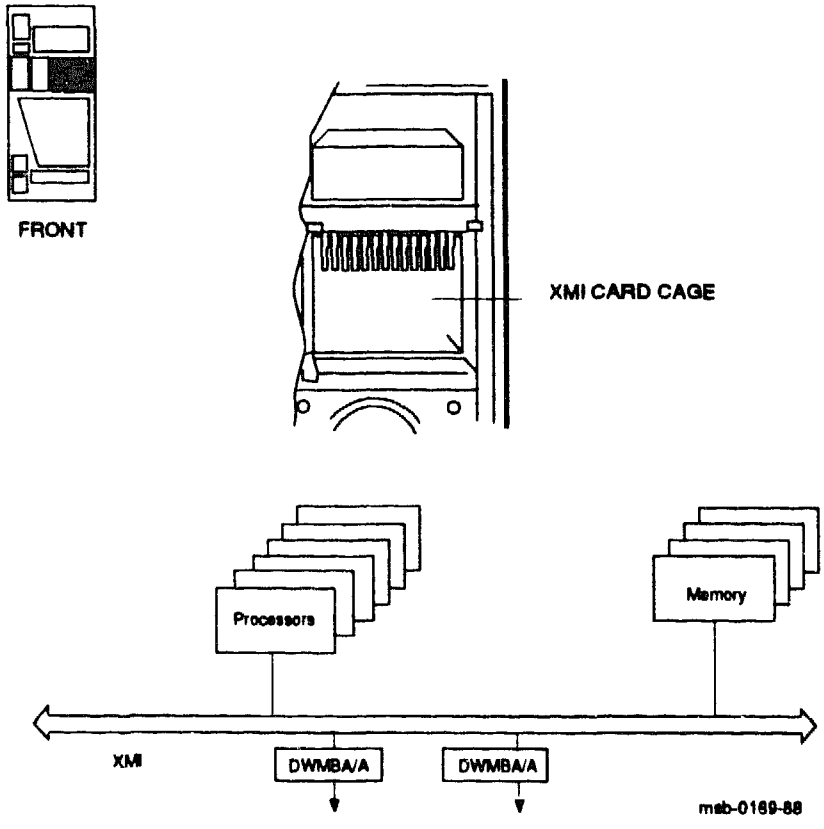


See Digital's *Systems and Options Catalog* for a complete list of VAXBI adapters available for the VAX 6000–400 and the *VAXBI Options Handbook* for detailed information on each VAXBI adapter.

1.8 XMI Backplane and Card Cage

The XMI high-speed system bus interconnects KA64A CPU module(s) and MS62A memory module(s). The XMI card cage has 14 slots and a maximum bandwidth of 100 megabytes per second.

Figure 1–6 VAX 6000–400's XMI



The XMI is a limited-length, pended, synchronous bus with centralized arbitration. The XMI bus can process several transactions simultaneously, making efficient use of the bus bandwidth. The bus includes the XMI backplane, the electrical environment of the bus, the protocol that nodes use on the bus, and the logic to implement this protocol.

The XMI backplane and 14-slot (nodes 1 through E) card cage are located in the upper third of the cabinet on the right side, as viewed from the front of the cabinet. A clear latched door protects the components housed in the XMI card cage and helps to direct the airflow over the modules. Indicator lights on the XMI modules can be viewed through this clear front door.

Each slot of the XMI card cage is hardwired to a 4-bit node ID code that corresponds to the physical slot number in the card cage. The node ID number of the module is its slot position. The nodes are numbered 1 through E (hex) from right to left, as you view the card cage from the front of the cabinet.

For information on installing modules in the XMI card cage, see the *VAX 6000–400 Options and Maintenance* manual. For in-depth technical information, see the appropriate chapter of this manual.

Table 1–2 XMI Slots

Slot	Node	Permissible Modules[1]
1	1	CPU, I/O
2	2	CPU, Mem, I/O
3	3	CPU, Mem, I/O
4	4	CPU, Mem, I/O
5	5	CPU, Mem
6	6	CPU, Mem
7	7	CPU, Mem
8	8	CPU, Mem
9	9	CPU, Mem
10	A	CPU, Mem
11	B	CPU, Mem, I/O
12	C	CPU, Mem, I/O
13	D	CPU, Mem, I/O
14	E	CPU, I/O

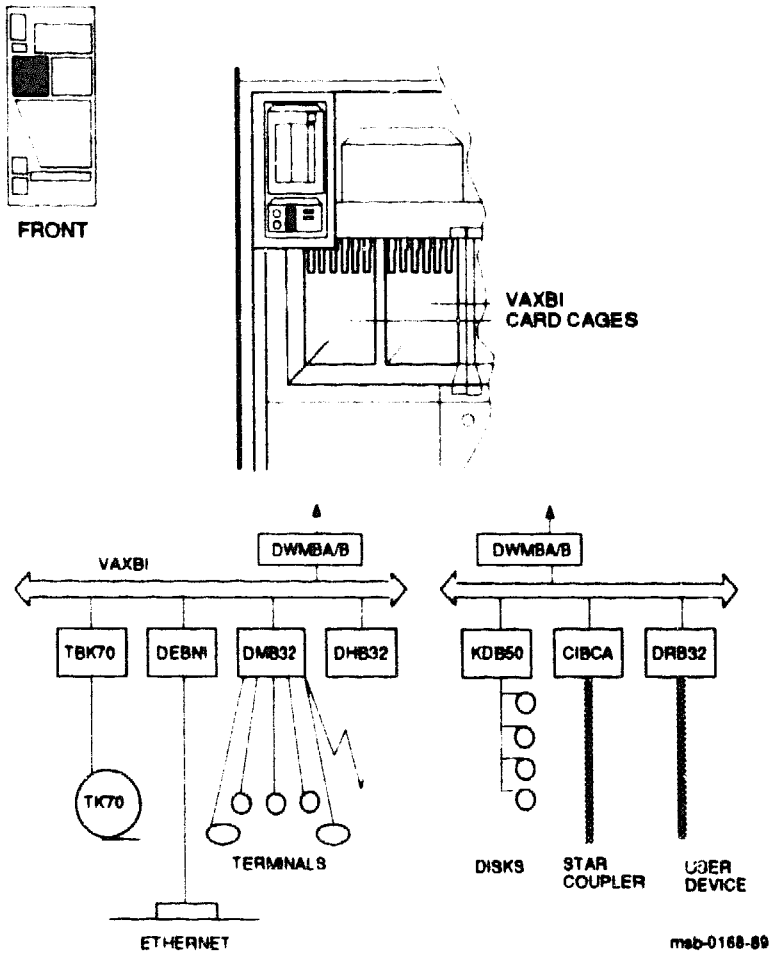
[1]Key to permissible modules:

CPU = KA64A CPU module
Mem = MS62A memory module
I/O = DWMBA

1.9 VAXBI Backplane and Card Cage

The VAXBI is the I/O interface. The VAXBI card cages house modules that connect the system to the Ethernet, VAXclusters, multiple terminals, and other peripherals. This configuration has a 10-Mbyte maximum bandwidth.

Figure 1–7 VAX 6000–400's VAXBI

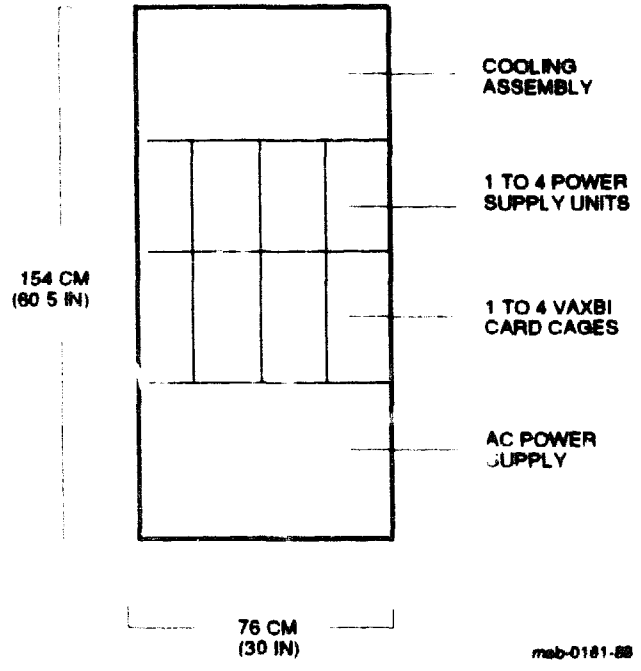


Two 6-slot VAXBI card cages are located in the upper third of the cabinet on the left side, as viewed from the front of the cabinet. A clear latched door protects the components housed in the VAXBI card cages and helps to direct the airflow over the modules. Indicator lights on the VAXBI modules can be viewed through this clear front door. A VAXBI expander cabinet can also be added to the system (see Section 1.10).

1.10 VAXBI Expander Cabinet

The VAXBI expander cabinet can be ordered to increase the system's I/O slots. With a VAXBI expander cabinet, one to four VAXBI card cages can be added.

Figure 1–8 VAXBI Expander Cabinet



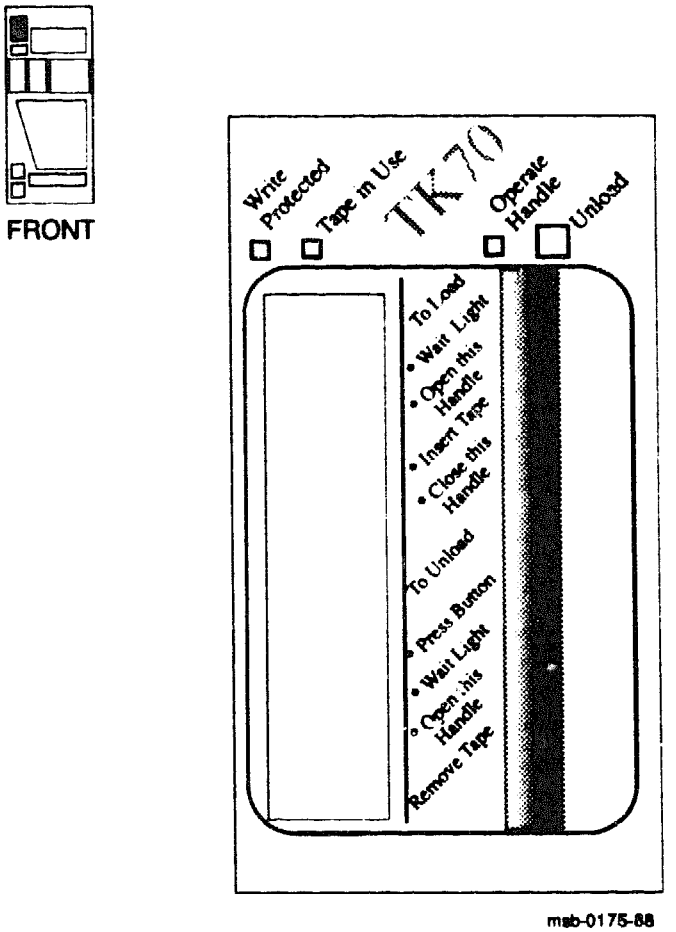
A VAXBI expander cabinet (see Figure 1–8) allows you to attach additional VAXBI channels to provide up to 24 additional slots of VAXBI. Each card cage contains six slots. By cabling from one VAXBI card cage to the next, a bus can be created with more than six nodes. The maximum number of nodes for each VAXBI bus is 16, including the one node required for its DWMBA/B module. The system accepts multiple VAXBI expander cabinets.

Four power supply units provide power to the VAXBI backplanes. Two blowers cool the cabinet, and an AC power controller completes the power system.

1.11 TK Tape Drive

The TK tape drive is mounted at the front of the system cabinet in the upper left corner.

Figure 1–9 TK Tape Drive



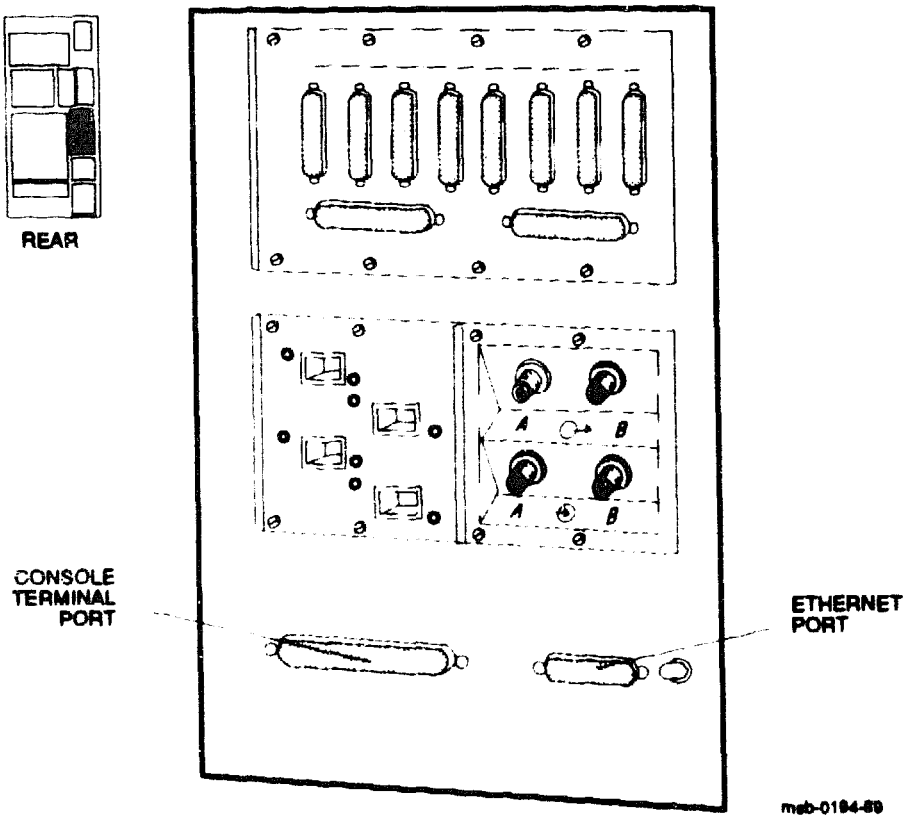
The TK tape drive is used for:

- Installing or updating software
- Loading diagnostics
- Interchanging user data
- Saving, restoring, and updating the contents of the EEPROM
- Loading stand-alone backup

1.12 Standard I/O Connections

The Ethernet and console terminal signal connections are located in the rear of the cabinet, above the AC power controller.

Figure 1–10 Standard I/O Connections



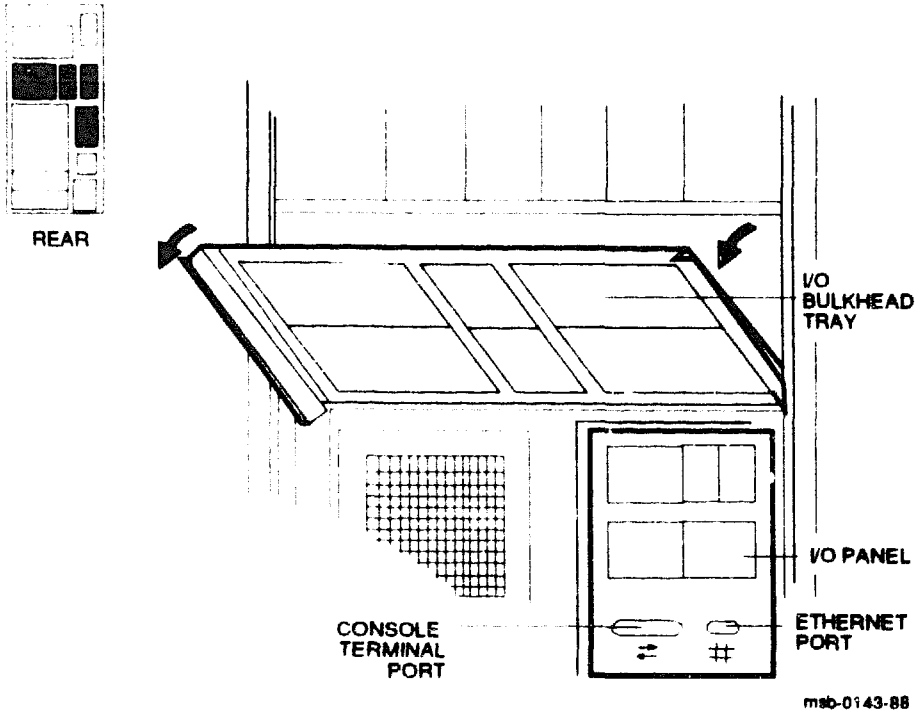
The Ethernet and console terminal ports are at the bottom of the I/O panel. The Ethernet port is a 15-pin receptacle located on the bottom right, and the console terminal port is the 25-pin receptacle on the left.

This I/O panel also has two octal openings for additional I/O connections. Typical I/O panels installed in this connector panel may include the tape drive connect area, the KDB50 disk controller port, or the CI cable connect.

1.13 I/O Bulkhead Connections

The I/O bulkhead tray is located in the rear of the cabinet, above the cooling system and standard I/O connection panel, and below the power regulators. The tray covers the XMI and VAXBI backplanes. It is hinged at the bottom and folds out and down for access to the card cages.

Figure 1–11 I/O Bulkhead



The tray is designed to accommodate a variety of I/O connections.

1.14 Power System

The power system consists of an H405E/F AC power controller, the H7206 power and logic unit, five power regulators for the XMI and VAXBI backplanes, and an optional battery backup unit.

Figure 1–12 Power System (Rear View)

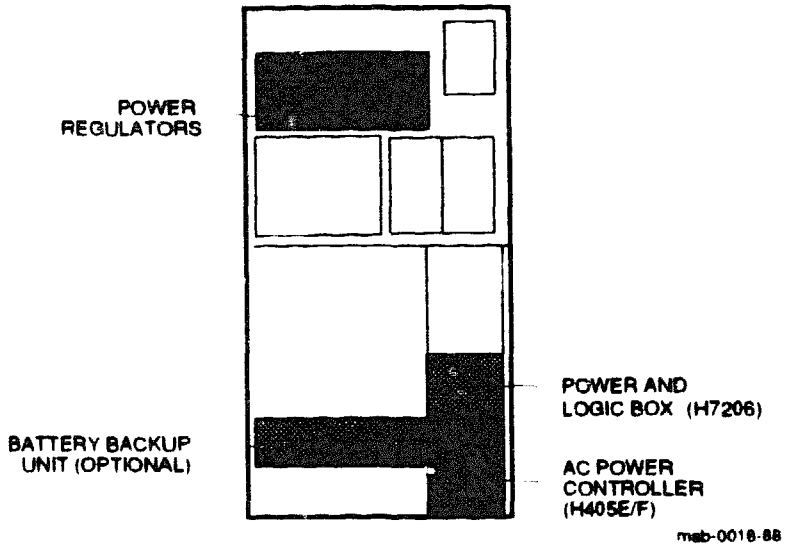


Table 1–3 Input Voltage

Model No.	Hz	Nominal	Phase
H405-E	60	208V	3
H405-F*	50	380V	3
H405-F	50	416V	3

*Change tap for 380V (nominal) operation.

Table 1–4 DC Power Distribution

Voltage	Current (Amps) Min. – Max.	Description
XMI		
+5V	1 – 120	Main logic supply
+5VBB	1 – 120	Memory supply
+12V	0 – 4	Communications devices and TK tape drive
–12V	0 – 2.5	Communications devices
–5.2V	0 – 20	ECL supply
–2V	0 – 7	ECL terminator voltage
VAXBI		
+5V	1 – 120	Main logic supply
+12V	0 – 4	Communications devices
–12V	0 – 2.5	Communication devices
–5.2V	0 – 20	ECL voltage
–2V	0 – 7	ECL teminator voltage
H7206-A power and logic module (PAL)		
+24V	0 – 4	Blowers and airflow sensor
Ethernet transceivers		
+13.5V	0 – 1.5	

Most of the power system is visible from the rear of the cabinet. The AC power controller is in the lower right corner. The power and logic box is above the AC power controller. Across the top of the cabinet are the power regulators for the XMI and VAXBI card cages.

Power is supplied by two H7215 power regulators and three H7214 power regulators. One H7215 and one H7214 supply the power to the VAXBI; one H7215 and two H7214s supply the power to the XMI. See Table 1–4.

The optional H7231 battery backup unit, if present, is located in the front left lower third of the cabinet, near the airflow plenum. This unit supplies 200 watts of +5VBB to system memory for 10 minutes following power interruption.

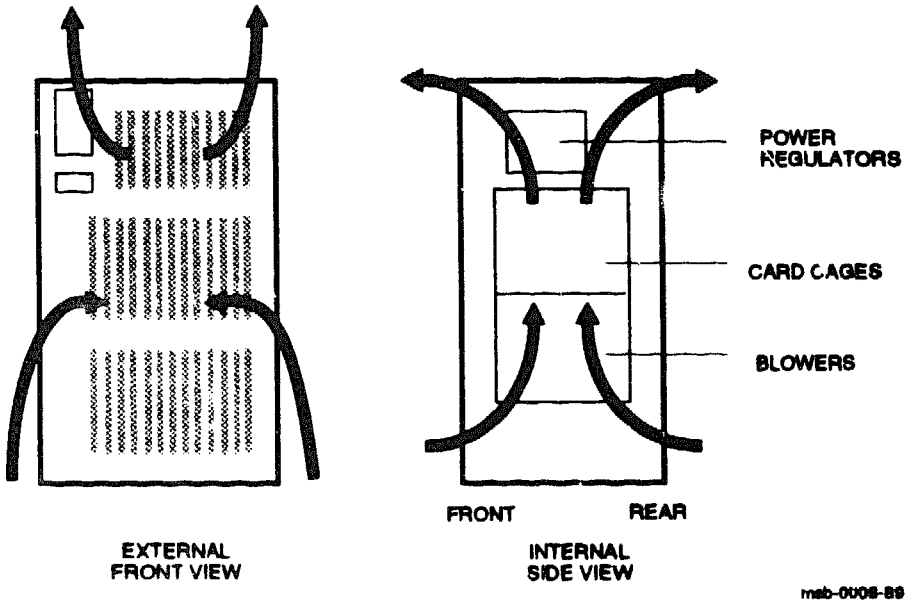
The H7231 battery backup unit power connection is on the H405 AC power controller and is fuse-protected at 10 A (60 Hz) or 6 A (50 Hz).

Three neon lights on the H405-E AC power controller indicate when AC power is present to the unit. The control panel on the front of the system indicates the status of the battery backup unit.

1.15 Cooling System

The cooling system consists of two blowers, an airflow sensor, a temperature sensor, and an airflow path through the card cages and up to the power regulators.

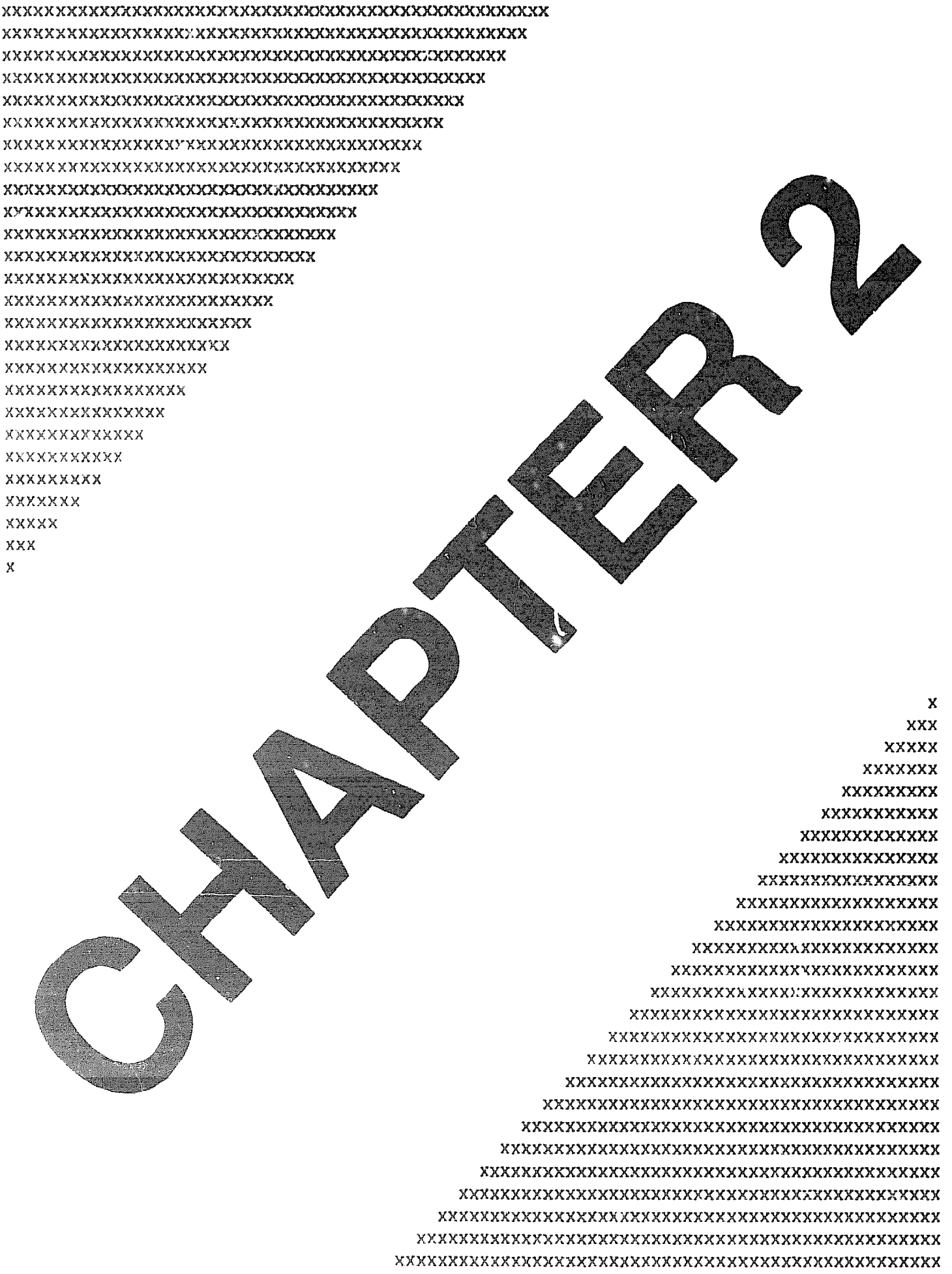
Figure 1–13 Airflow Pattern



The cooling system is designed to keep system components at an optimal operating temperature. It is important to keep the front and rear doors free of obstructions, leaving a clear space of 3 feet (1 meter) from the cabinet, so that air intake is kept at maximum capacity.

Air is taken in by the blowers, located in the lower half of the cabinet. The blowers push air up through the VAXBI and XMI card cages. The airflow continues through the top of the card cages, through the power regulators, and out the top of the front and rear doors.

The system has safety detectors for the cooling system: an airflow sensor and a temperature sensor installed above the power regulators in the top of the cabinet. Extreme conditions activate these detectors. The temperature sensor shuts off the power at the AC power controller if the unit experiences extreme temperatures. If the system has airflow seriously blocked for an extended period of time, then the airflow sensor will shut off power.


CHAPTER 2

2 The XMI

This chapter describes the XMI, which includes a backplane and bus interconnect, protocol, and logic.

This chapter includes the following sections:

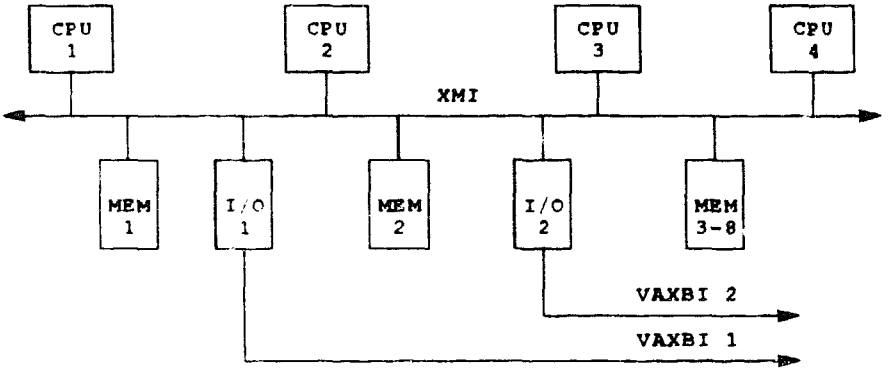
- XMI Overview
- XMI Addressing
- Arbitration Cycles
- XMI Cycles
- XMI Transactions
- XMI Initialization
- XMI Registers
- XMI Errors

2.1 XMI Overview

The XMI is the primary interconnect for the VAX 6000–400 system. The XMI supports multiple processors, multiple memory modules, and multiple I/O adapters. Figure 2–1 shows a four-processor system.

2.1.1 XMI System Block Diagram Description

Figure 2–1 XMI System Block Diagram



The XMI consists of the electrical environment of the XMI bus, the protocol observed by a node on the bus, the backplane, and the logic used to implement the protocol.

The XMI bus is limited length, pended, and synchronous with centralized arbitration. Several transactions can be in progress at a given time, allowing highly efficient use of the bus bandwidth. Arbitration and data transfers can occur simultaneously. The bus supports:

- Quadword-, octaword-, and hexword-length reads to memory
- Quadword- and octaword-length memory writes
- Longword-length read and write operations to I/O space

The longword operations implement byte and word modes required by certain I/O devices. The XMI has a 64 ns bus cycle. The XMI has a bandwidth of 100 Mbytes per second; however, the usable bandwidth depends on transaction length (see Table 2–1).

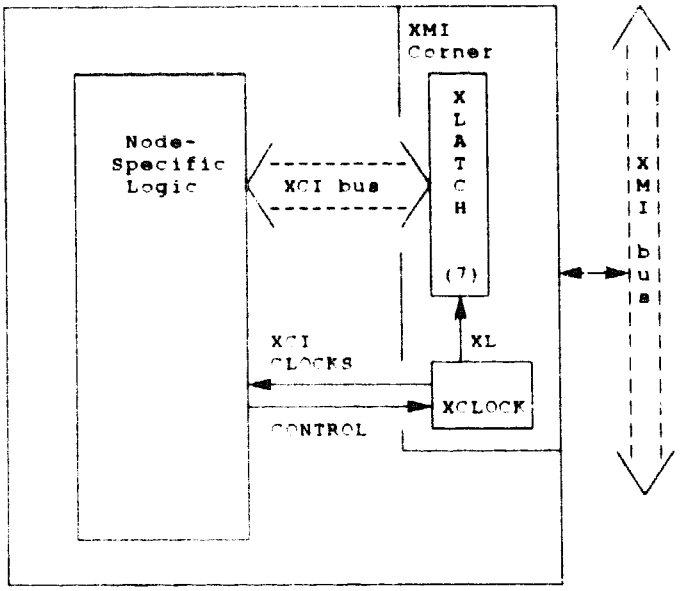
Table 2–1 Usable XMI Bandwidth

Operation	Bandwidth (Mbytes/second)
Longword (4 bytes) Read	31.25
Quadword (8 bytes) Read	62.50
Octaword (16 bytes) Read	83.30
Hexword (32 bytes) Read	100.00
Longword Write	31.25
Quadword Write	62.50
Octaword Write	83.30

2.1.2 XMI Corner

The XMI uses similar, but incompatible, connector and module technology as the VAXBI bus and, like the VAXBI, XMI modules have an area with predefined etch with custom components, which serves as the interface between the module and the XMI bus. This predefined etch and components is called the XMI Corner.

Figure 2–2 XMI Node Block Diagram Showing the XMI Corner



The XMI Corner has a predefined etch and parts placement. The custom components in the XMI Corner are called XLATCH and XCLOCK. Both components are implemented in CMOS and interface node-specific logic to the XMI Corner components over the XMI Corner interface (XCI) bus. The XMI Corner, in turn, interfaces directly to the XMI bus. (See Figure 2–2.)

Each node has a set of three clock signals, which are distributed radially to each node from a central source on the backplane. These clocks are received by the XCLOCK chip, which then provides a set of clock waveforms (XCI clocks) to the node-specific logic and the required control lines (XL lines) for the seven XLATCH chips. The XLATCH chips provide the interface to all the XMI lines except those directly interfaced to the XCLOCK chip.

2.1.3 XMI Data Transactions

The XMI supports various data transactions, as shown in Table 2–2.

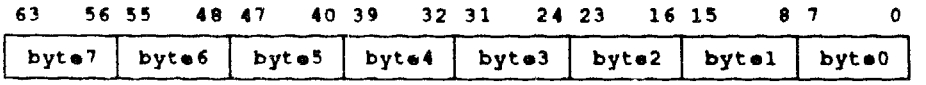
Table 2–2 Data Transactions Supported by the XMI

Transaction	Length	I/O Space	Memory Space
Read	Longword	X	
	Quadword		X
	Octaword		X
	Hexword		X
Interlock Read	Longword	X	
	Quadword		X
	Octaword		X
	Hexword		X
Write Masked	Longword	X	
	Quadword		X
	Octaword		X
Unlock Write	Longword	X	
	Quadword		X
	Octaword		X

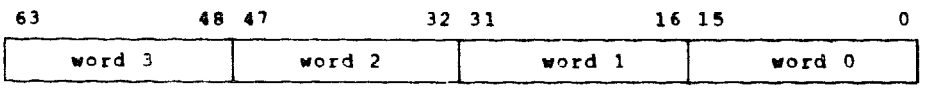
2.1.4 XMI Terms

The following terms are used to describe XMI transactions:

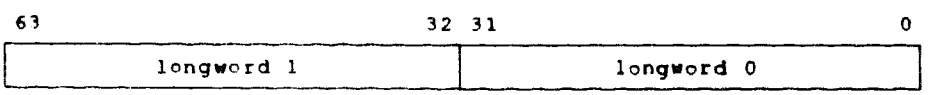
Term	Definition
Node	A hardware device that connects to the XMI backplane.
Transfer	The smallest quantum of work that occurs on the XMI. An example of a transfer is the command cycle of a read. Another example is the command cycle for a write, followed by data cycles.
Cycle	The complete execution of one XMI clock time period.
Transaction	The logical task being performed (such as a read). A transaction is composed of one or more transfers. As an example of a transaction, the read consists of a command transfer followed, some time later, by a return data transfer.
Commander	The node that initiated the transaction in progress. For example, the commander initiates a read transaction while the responder (data source) initiates the read data transfer. The responder is not the commander for the read data transfer because the transfer was requested by the commander node.
Responder	The node that responds to the commander in a transaction.
Transmitter	The node that is sourcing the information on the bus. For example, during a read transaction the commander is the transmitter during the command transfer but is the receiver during the return data transfer.
Receiver	The node that is the target during a transfer.
Naturally aligned	Describes a data quantity whose address could be specified as an offset, from the beginning of memory, of an integral number of data elements of the same size. The lower bits of a naturally aligned data item are zero. All XMI writes transfer a naturally aligned block of data.
Wraparound read	An octaword or hexword read where read data is returned with the specifically addressed quadword first, independent of alignment. The remaining data in the naturally aligned block of data containing the addressed quadword is returned in subsequent transfers. See Section 2.1.5.
Byte	A single 8-bit entity.



Word	A single 16-bit entity.

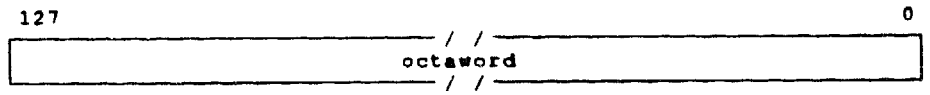


Longword	A single 32-bit entity.

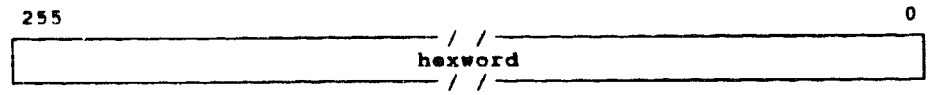


Term	Definition
Quadword	A single 64-bit entity.

Octaword	A single 128-bit entity.



Hexword	A single 256-bit entity.



2.1.5 Wraparound Reads

Read data is returned in a specific pattern referred to as "wraparound read" for octaword or hexword read operations. In a wraparound read, the specifically addressed quadword is returned first, independent of alignment. The remaining data in the naturally aligned block of data containing the addressed quadword is returned in subsequent transfers.

XMI protocol requires that all octaword and hexword reads, both normal and interlocked, be treated as wraparound reads.

2.1.6 Octaword Wraparound Read

The following is an example of an octaword wraparound read at VAX byte address 00000018 (hex):

00000018 First quadword
00000010 Second quadword

2.1.7 XMI Interrupt Transactions

The XMI supports three types of interrupt transactions, listed in Table 2–3.

Table 2–3 XMI Interrupt Transactions

Type	Mnemonic
Interrupt Request	INTR
Identify (Interrupt Acknowledge)	IDENT
Implied Vector Interrupt	IVINTR

The INTR and IDENT transactions implement device interrupts. An I/O node issues an INTR transaction to a processor to interrupt the processor at a specified interrupt priority level (IPL). The processor responds to the INTR by issuing an IDENT transaction to the interrupting I/O node, soliciting an interrupt vector.

An INTR transaction can be broadcast to multiple processor nodes. The first processor to respond with IDENT receives the interrupt vector. All other processors, upon seeing the IDENT directed to the interrupting device, cease their interrupt-pending condition. If IDENTs are issued simultaneously by two or more processors, the first to gain the bus will service the interrupt while the other(s) force a microcode passive release.

The IVINTR transaction implements single-cycle interrupt transactions where the interrupt priority and the interrupt vector value are implied by bits in the interrupt type field. The IVINTR transaction implements VAX interprocessor interrupts (IPL = 16 (hex), vector = 80 (hex)) and write error interrupts (IPL = 1D (hex), vector = 60 (hex)). Since the value of the interrupt vector is indicated by the value of the IPL field, IVINTR transactions do not require a corresponding interrupt acknowledge cycle.

See Section 2.5.5 and Section 2.5.6 for more information on interrupt transactions.

2.1.8 Arbitration

The XMI protocol includes arbitration because, at any time, any or all of the nodes may desire the use of the XMI. Arbitration determines which node gains the XMI when more than one node requests the XMI simultaneously.

Table 2–4 XMI Arbitration Lines

Name	Use
XMI CMD REQ L	Initiates XMI transactions
XMI RES REQ L	Returns data
XMI GRANT L	Indicates which node has been granted the XMI bus for the next cycle

The VAX 6000–400 supports an XMI bus of 14 nodes. Arbitration cycles occur in parallel with data transfer cycles, since the XMI has a set of lines dedicated to arbitration. These lines are listed in Table 2–4.

When a node desires ownership of the bus, it asserts one of its two request lines (XMI CMD REQ L or XMI RES REQ L) that are connected to the central arbiter. The XMI CMD REQ L line is used by nodes to initiate XMI transactions (that is, act as a commander) while the XMI RES REQ L line is used by nodes to return data to a commander (that is, act as a responder). The XMI arbiter maintains two independent round-robin queues, one for each request type. The responder requests are given higher priority than commander requests.

See Section 2.3 for more information on arbitration.

2.1.9 Bus Integrity

The XMI bus contains a number of features to enhance the integrity and reliability of the bus.

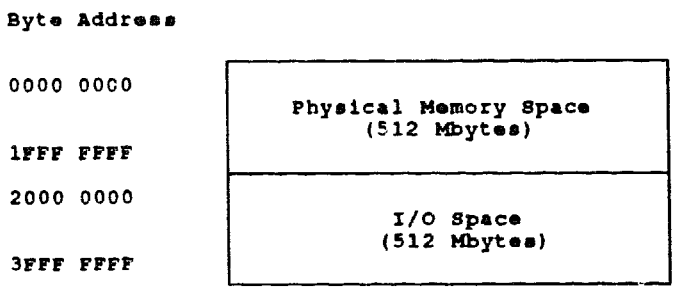
The features of the XMI that enhance bus integrity and reliability are:

- All bus information transfer lines are parity protected.
- Bus confirmation signals are ECC protected.
- XMI protocol permits detection and recovery of almost all single-bit errors on the information transfer lines and bus confirmation signal lines.
- XMI protocol defines timeout conditions that are used to detect failures.

2.2 XMI Addressing

The XMI supports one gigabyte (2^{30} bytes) of address space, divided into the physical memory space and I/O space, shown in Figure 2–3.

Figure 2–3 XMI Memory and I/O Address Space



2.2.1 XMI Memory Space

A/D<29> selects between the memory and I/O space. A/D<29> = 0 selects physical memory space, while A/D<29> = 1 selects I/O space.

The upper two bits of an XMI address are used to define transfer size.

2.2.2 XMI I/O Space

XMI I/O space is divided into private space, nodespace, and eight I/O adapter address space regions.

Figure 2–4 XMI I/O Space Address Allocation

Byte Address		Size
2000 0000	XMI Private Space	24 Mbytes
2180 0000	XMI Nodespace	16 x 512 Kbytes
2200 0000	I/O Adapter 1 Address Space	32 Mbytes
2400 0000	I/O Adapter 2 Address Space	32 Mbytes
2600 0000	I/O Adapter 3 Address Space	32 Mbytes
2800 0000	I/O Adapter 4 Address Space	32 Mbytes
2A00 0000	Reserved	192 Mbytes
3600 0000	I/O Adapter B Address Space	32 Mbytes
3800 0000	I/O Adapter C Address Space	32 Mbytes
3A00 0000	I/O Adapter D Address Space	32 Mbytes
3C00 0000	I/O Adapter E Address Space	32 Mbytes
3E00 0000	Reserved	32 Mbytes
3FFF FFFF		

2.2.2.1 XMI Private Space

References to XMI private space are serviced by resources local to a node, such as local device CSRs and boot ROM. The references are not broadcast on the XMI. XMI private space is a 24-Mbyte address region located from 2000 0000 to 217F FFFF.

2.2.2.2 XMI Nodespace

The VAX 6000–400 XMI nodespace is a collection of 16 512-Kbyte regions located from 2180 0000 to 21FF FFFF. Nodes 0 and F are not implemented. Each XMI node is allocated one of the 512-Kbyte regions for its control and status registers. The starting address of the 512-Kbyte region associated with a given node is computed as follows:

2180 0000 + Node ID * 80000

Table 2–5 XMI Nodespace Addresses

Slot	Node	Nodespace	I/O Adapter Space
1	1	2188 0000 – 218F FFFF	2200 0000 – 23FF FFFF
2	2	2190 0000 – 2197 FFFF	2400 0000 – 25FF FFFF
3	3	2198 0000 – 219F FFFF	2600 0000 – 27FF FFFF
4	4	21A0 0000 – 21A7 FFFF	2800 0000 – 29FF FFFF
5	5	21A8 0000 – 21AF FFFF	N/A[1]
6	6	21B0 0000 – 21B7 FFFF	N/A[1]
7	7	21B8 0000 – 21BF FFFF	N/A[1]
8	8	21C0 0000 – 21C7 FFFF	N/A[1]
9	9	21C8 0000 – 21CF FFFF	N/A[1]
10	A	21D0 0000 – 21D7 FFFF	N/A[1]
11	B	21D8 0000 – 21DF FFFF	3600 0000 – 37FF FFFF
12	C	21E0 0000 – 21E7 FFFF	3800 0000 – 39FF FFFF
13	D	21E8 0000 – 21EF FFFF	3A00 0000 – 3BFF FFFF
14	E	21F0 0000 – 21F7 FFFF	3C00 0000 – 3DFF FFFF

[1]Slots in the center of the XMI card cage have no I/O connectors because of the daughter card's presence.

2.2.2.3 I/O Adapter Address Space

I/O adapter address space consists of eight 32-Mbyte address regions used to access VAXBI I/O adapters. Longword-length references directed to a VAXBI's I/O adapter address space will be reissued on that VAXBI bus. XMI transactions are translated into a corresponding VAXBI transaction. The VAXBI address of the transaction is computed from XMI addresses as 2000 0000 + offset, where offset is the difference between the XMI address and the start of the appropriate DWMBA/A module's address space. XMI devices can only access VAXBI I/O space, as VAXBI memory space is not accessible to nodes on the XMI.

To calculate the address of the first register in nodespace (the DTYPE register):

- The base address of I/O space is 2000 0000 (hex).
- Bits<28:25> correspond to the XMI node number, which is the same as the slot number except that node numbers are in hexadecimal while slot numbers are in decimal. The VAX 6000–400 XMI nodes typically assigned to VAXBI channels are 1, 2, 3, 4, B, C, D, and E. These are used for:

  E - the VAXBI in the middle of the system cabinet
  D - the VAXBI in the left of the system cabinet
  C - the first VAXBI in an expander cabinet; usually leftmost
  B - the second VAXBI in an expander cabinet; usually center-left
  1 - the third VAXBI in an expander cabinet; usually center-right
  2 - the fourth VAXBI in an expander cabinet; usually rightmost
  3 - the fifth VAXBI in an expander cabinet; not usually used
  4 - the sixth VAXBI in an expander cabinet; not usually used

- Bits<16:13> correspond to the VAXBI node number. For the VAXBIs inside the system cabinet, the node number is usually the same as the slot number: 1, 2, 3, 4, 5, or 6.

2.2.2.4 How to Find a Register in VAXBI Address Space

The first part of a VAXBI adapter's physical XMI address depends on which XMI slot the DWMBA/A module occupies. The second part of the address depends on the adapter's VAXBI node number, which is shown in the SHOW CONFIGURATION display.

Note: VAXBI slot and node numbers are not identical. The placement of the VAXBI node ID plug on the backplane determines the node ID, so seeing that a particular option is in a certain slot does not guarantee that the slot and node number are identical. Use the VAXBI node identification from the SHOW CONFIGURATION command.

The XMI slot number can be determined in two ways:

- By looking at the XMI card cage (numbering of slots is shown in Figure 2–5)
- By entering at the console a SHOW CONFIGURATION command

A typical response is shown below.

```
>>> SHOW CONFIGURATION

      Type            Rev
  1+  KA64A    (8082) 0006
  2+  KA64A    (8082) 0006
  9+  MS62A    (4001) 0002
  A+  MS62A    (4001) 0002
  D+  DWMBA/A  (2001) 0002
  E+  DWMBA/A  (2001) 0002

  XBI D
  1+  DWMBA/B  (2107) 0007
  5+  DMB32    (0109) 210B
  6+  DEBNI    (0118) 0100

  XBI E
  1+  DWMBA/B  (2107) 0007
  4+  KDB50    (010E) 0F1C
  6+  TBK70    (410B) 0307
```

Assume that you want to examine the Device Register (DTYPE) for the DEBNI, which is node 6 in the first VAXBI channel shown above (XBI D). Figure 2–5 also shows which VAXBI cage is linked to which XMI slot.

To get the address for the DEBNI Device Register (DTYPE), do the following:

1 From Table 2–5 find XMI node D and take the 2-digit prefix for the start of that node's window space (3A).

2 From Table 2–6 find VAXBI node 6 and in column 2 you can see that the starting address for VAXBI node 6 is xx00 C000.

3 Combine this second number with the 2-digit prefix. You now have the adapter's base address (3A00 C000) in VAXBI address space, indicated by lowercase bb.

4 From Table 2–7, VAXBI Registers, you can see that the VAXBI Device Register (DTYPE) is at bb + 00, which is 3A00 C000.

The Device Register for the DEBNI would be examined by:

```
>>> E/L/P 3A00C000
```

Figure 2–5 VAX 6000–400 Slot Numbers

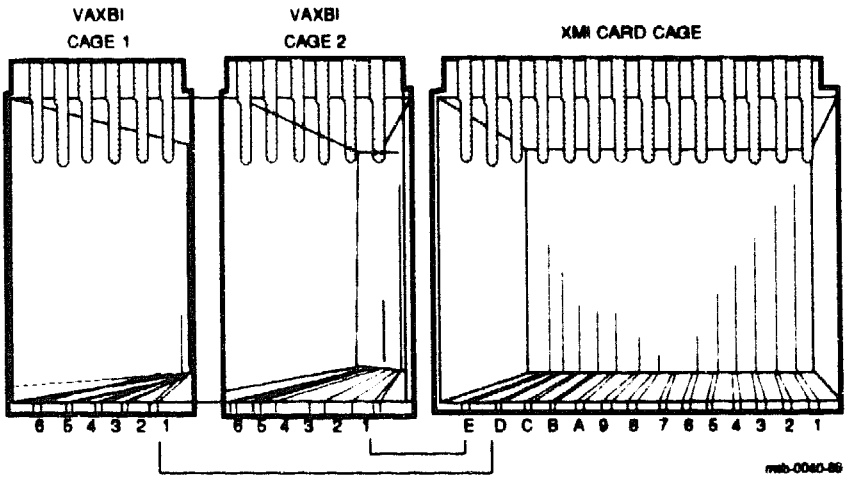


Table 2–6 VAXBI Nodespace and Window Space Address Assignments

Node Number	Nodespace Addresses Starting	Nodespace Addresses Ending	Window Space Addresses Starting	Window Space Addresses Ending
0	xx00 0000	xx00 1FFF	xx40 0000	xx43 FFFF
1	xx00 2000	xx00 3FFF	xx44 0000	xx47 FFFF
2	xx00 4000	xx00 5FFF	xx48 0000	xx4B FFFF
3	xx00 6000	xx00 7FFF	xx4C 0000	xx4F FFFF
4	xx00 8000	xx00 9FFF	xx50 0000	xx53 FFFF
5	xx00 A000	xx00 BFFF	xx54 0000	xx57 FFFF
6	xx00 C000	xx00 DFFF	xx58 0000	xx5B FFFF
7	xx00 E000	xx00 FFFF	xx5C 0000	xx5F FFFF
8	xx01 0000	xx01 1FFF	xx60 0000	xx63 FFFF
9	xx01 2000	xx01 3FFF	xx64 0000	xx67 FFFF
A	xx01 4000	xx01 5FFF	xx68 0000	xx6B FFFF
B	xx01 6000	xx01 7FFF	xx6C 0000	xx6F FFFF
C	xx01 8000	xx01 9FFF	xx70 0000	xx73 FFFF
D	xx01 A000	xx01 BFFF	xx74 0000	xx77 FFFF
E	xx01 C000	xx01 DFFF	xx78 0000	xx7B FFFF
F	xx01 E000	xx01 FFFF	xx7C 0000	xx7F FFFF

Table 2–7 VAXBI Registers

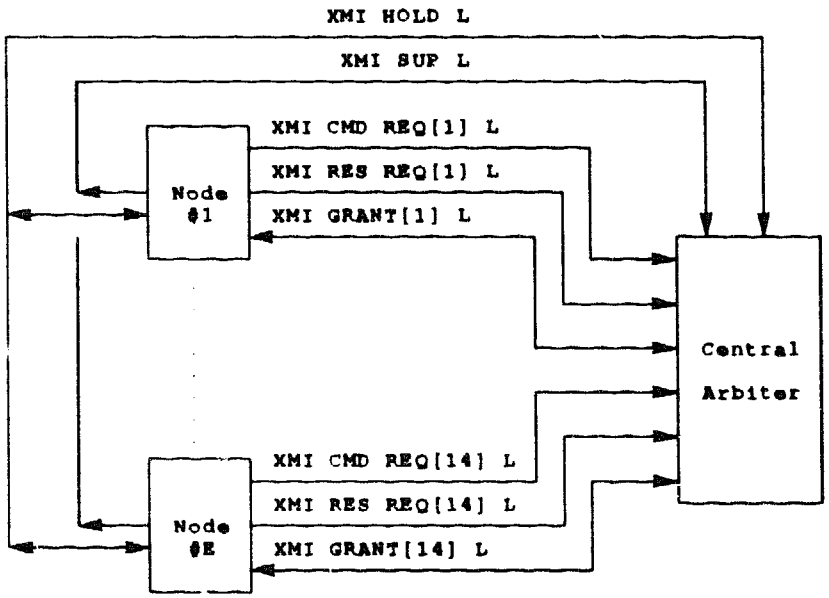
Name	Mnemonic	Address[1]
Device Register	DTYPE	bb+00
VAXBI Control and Status Register	VAXBICSR	bb+04
Bus Error Register	BER	bb+08
Error Interrupt Control Register	EINTRSCR	bb+0C
Interrupt Destination Register	INTRDES	bb+10
IPINTR Mask Register	IPINTRMSK	bb+14
Force-Bit IPINTR/STOP Destination Register	FIPSDES	bb+18
IPINTR Source Register	IPINTRSRC	bb+1C
Starting Address Register	SADR	bb+20
Ending Address Register	EADR	bb+24
BCI Control and Status Register	BCICSR	bb+28
Write Status Register	WSTAT	bb+2C
Force-Bit IPINTR/STOP Command Register	FIPSCMD	bb+30
User Interface Interrupt Control Register	UINTRCSR	bb+40
General Purpose Register 0	GPR0	bb+F0
General Purpose Register 1	GPR1	bb+F4
General Purpose Register 2	GPR2	bb+F8
General Purpose Register 3	GPR3	bb+FC
Slave-Only Status Register	SOSR	bb+100
Receive Console Data Register	RXCD	bb+200

[1]The abbreviation "bb" refers to the base address of a VAXBI node (the address of the first location of the nodespace).

2.3 Arbitration Cycles

The XMI protocol includes arbitration because, at any time, any or all of the nodes may desire the use of the XMI. Arbitration determines which node gains the XMI when more than one node requests the XMI simultaneously. Arbitration cycles occur in parallel with data transfer cycles, since the XMI has a set of arbitration-dedicated lines.

Figure 2–6 XMI Arbitration Block Diagram



The XMI protocol architecturally supports up to 16 XMI nodes. However, the VAX 6000–400 implementation supports 14 nodes. Each node on the XMI bus has a hexadecimal identification number (1 through E) called the node ID, which is provided by the node's hardwired XMI NODE ID<3:0> H lines. The physical slot number equals the node ID. Slot 1 is the rightmost slot in the XMI card cage when viewed from the front of the cabinet.

Any or all nodes may desire the use of the XMI at any given time. Arbitration cycles occur in parallel with data transfer cycles by using a set of lines dedicated to arbitration. The XMI CMD REQ L line, the XMI RES REQ L line, and the XMI GRANT L line go between the central arbiter and each node. The XMI CMD REQ L line is used by nodes to initiate XMI transactions (to act as a commander), while the XMI RES REQ L line is used to return data to a commander (to act as a responder). The XMI arbiter maintains two independent round-robin queues, one for each of the request types. The responder requests have a higher priority than commander requests.

During any given cycle, all nodes have the opportunity to request the bus. The arbiter receives all the requests, decides which node will be granted the bus, and uses that node's XMI GRANT L line to tell the node that it has been selected. In the next cycle, the selected node begins its transfer.

The XMI has two additional arbitration control signals, XMI HOLD L and XMI SUP L. XMI SUP L suppresses all commander requests but allows responder requests to continue to be serviced. Assertion of XMI HOLD L guarantees that the current XMI transmitter will be granted ownership of the bus in the next cycle, independent of the value of any other outstanding requests. The XMI HOLD L signal is used for multicycle transfers, allowing the current transmitter to keep ownership of the bus for consecutive cycles.

A node can temporarily block the start of additional XMI transactions by asserting the XMI SUP L signal should it have difficulties in keeping up with bus traffic, such as a memory queue becoming full or a CPU backing up on cache invalidate operations due to XMI writes.

The XMI arbitration scheme consists of three priority classes:

- Hold, which has the highest priority and guarantees that the current transmitter will be granted the bus in the next cycle.
- Responder requests, the next highest priority.
- Commander requests, the lowest priority.

Within the responder and commander classes, priority is distributed in a round-robin manner.

2.4 XMI Cycles

The purpose of an XMI cycle is determined by four signal lines on the XMI backplane, XMI F<3:0> L.

2.4.1 Function Codes

The XMI uses four lines to encode the function being performed on the bus. Table 2–8 lists the function codes.

Table 2–8 XMI Function Codes

XMI F<3:0> L Logic Levels					
3	2	1	0	Function	Mnemonic
0	0	0	0	NULL cycle	NULL
0	0	0	1	Command cycle	CMD
0	0	1	0	Write Data cycle	WDAT
0	0	1	1	Reserved (decoded as NULL)	
0	1	0	0	Lock Response	LOC
0	1	0	1	Read Error Response	RER
0	1	1	0	Reserved (decoded as NULL)	
0	1	1	1	Reserved (decoded as NULL)	
1	0	0	0	Good Read Data 0	GRD0
1	0	0	1	Good Read Data 1	GRD1
1	0	1	0	Good Read Data 2	GRD2
1	0	1	1	Good Read Data 3	GRD3
1	1	0	0	Corrected Read Data 0	CRD0
1	1	0	1	Corrected Read Data 1	CRD1
1	1	1	0	Corrected Read Data 2	CRD2
1	1	1	1	Corrected Read Data 3	CRD3

2.4.2 Command Cycles

During XMI command cycles, commander nodes initiate XMI transactions. The commander drives its commander ID on XMI ID<5:0> L and drives command information on D<63:0> L, as shown in Figure 2–7 and Figure 2–8.

Figure 2–7 Data Transaction Command Cycle Format

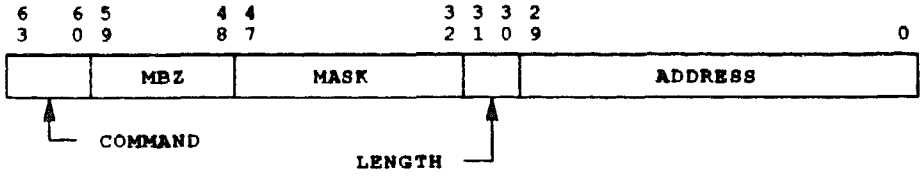
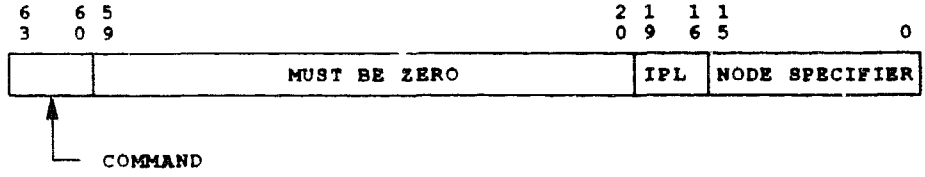


Figure 2–8 Interrupt Transaction Command Cycle Format



The command cycle has the command fields discussed in the following subsections:

- Command field
- Mask field
- Length field
- Address field
- Node Specifier field

2.4.2.1 Command Field

The Command field is XMI D<63:60> L. The Command field specifies the transaction being initiated in the command cycle. (See Table 2–9.)

Table 2–9 XMI Command Codes

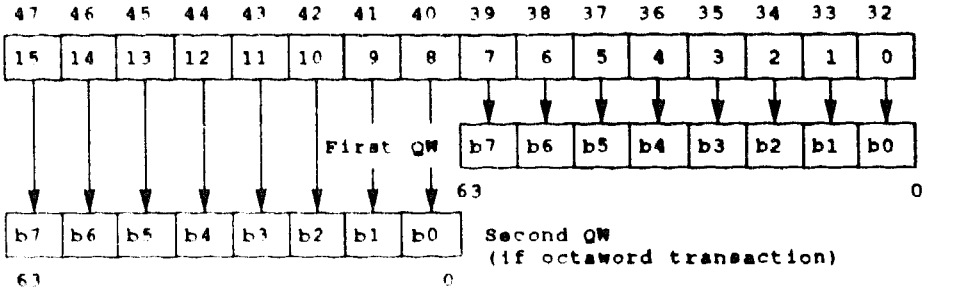
XMI D<63:60> L Logic Levels					
63	62	61	60	Command	Mnemonic
0	0	0	0	Reserved	
0	0	0	1	Read	READ
0	0	1	0	Interlock Read	IREAD
0	0	1	1	Reserved	
0	1	0	0	Reserved	
0	1	0	1	Reserved	
0	1	1	0	Unlock Write Mask	UWMASK
0	1	1	1	Write Mask	WMASK
1	0	0	0	Interrupt	INTR
1	0	0	1	Identify	IDENT
1	0	1	0	Reserved	
1	0	1	1	Reserved	
1	1	0	0	Reserved	
1	1	0	1	Reserved	
1	1	1	0	Reserved	
1	1	1	1	Implied Vector Interrupt	IVINTR

2.4.2.2 Mask Field

The Mask field is XMI D<47:32> L. The Mask field supplies byte-level mask information for the XMI Write Mask and Unlock Write Mask transactions. During nonwrite transactions this field is a "don't care," but proper parity is still generated. (See Figure 2–9.)

The maximum length of a write transaction is an octaword, which requires 16 mask bits in the upper longword of the command. The mask bits define which bytes of the following write data cycles are to be written to the specified locations. For longword- and quadword-length writes, the unused mask bits (D<47:36> L and D<47:40> L, respectively) are unspecified and are ignored by responders, other than to check parity.

Figure 2–9 Mask Field Bit Assignments



2.4.2.3 Length Field

The Length field is XMI D<31:30> L. The Length field is used to define the number of words in the XMI data transfer. Table 2–10 shows the Length field coding. Longword-length transactions are only used in I/O space. Quadword-, octaword-, and hexword-length transactions are only used in memory space. Hexword lengths are only used for Read or Interlock Read transactions.

Table 2–10 XMI Transaction Length Codes

XMI D<31:30> L Logic Levels		
31	30	Size
0	0	Hexword
0	1	Longword
1	0	Quadword
1	1	Octaword

2.4.2.4 Address Field

The Address field, XMI D<29:0> L, defines the address of an XMI read or write transaction. The number of significant bits in the address depends on the transaction type and length.

Quadword and octaword write transactions are assumed to be naturally aligned, allowing the lower bits of the address to be "don't care." Reads require that the lower bits be significant because memory does wraparound reads. All wrapped reads need to identify the quadword to be transferred first.

For longword-length transactions, XMI D<1:0> L are only significant for a VAXBI word-mode or byte-mode transaction in I/O space. XMI D<1> L is required for word mode and bits<1:0> are required for byte mode.

The relationship between the high and low words, the state of bit<1>, and the data bits is:

XMI D<1> = 1 ⇒ high word ⇒ D<31:16>
XMI D<1> = 0 ⇒ low word ⇒ D<15:0>

The data returned on the opposite word of the one specified will have correct parity, but its data is unspecified.

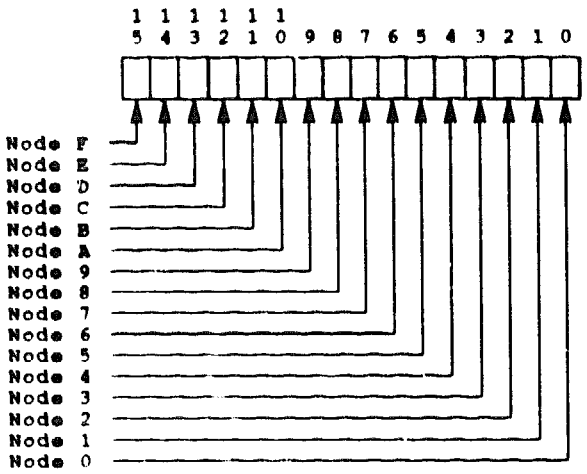
For a longword-oriented device, bit<1> is ignored as an address bit and a full longword of data is returned for a read operation.

2.4.2.5 Node Specifier Field

The Node Specifier field is XMI D<15:0> L. During command cycle interrupt transactions (INTR, IDENT, IVINTR), the Node Specifer field is used to specify the source or destination of an interrupt. (See Figure 2–8.) The relationship between bits in the Node Specifer field and the source/destination of an interrupt transaction is shown in Figure 2–10.

The VAX 6000–400 uses nodes 1 through E.

Figure 2–10 Node Specifier Field



2.4.3 Write Data Cycles

A function code of 0010 identifies an XMI write data cycle. Write data cycles immediately follow the XMI command cycle during an XMI write transfer. During this cycle, the commander drives its ID on XMI ID<5:0> L and drives write data on D<63:0> L. The full 64 bits of data are used during quadword-length or larger writes. For longword-length writes, only the lower longword D<31:0> L is used and the value of the upper longword is unspecified. In either case, the full 64 bits are used when checking XMI P<2:0> L.

2.4.4 Good Read Data (GRD) and Corrected Read Data (CRD) Response Cycles

Function codes 1000 through 1111 are used to identify return data in response to a Read, Interlock Read, or IDENT transaction. The Good Read Data response (GRDn, codes 1000 – 1011) indicates that the quadword of data is error-free. The Corrected Read Data response, CRDn, codes 1100 – 1111) indicate that the corresponding quadword of data stored in memory contained a single-bit error which was successfully corrected using ECC prior to shipment on the XMI. Both types of read data responses contain a sequence ID located in XMI F<1:0> L, which is used to identify when a read data cycle has been lost due to an XMI parity error.

During a read data response cycle, the responder drives the commander's ID on XMI ID<5:0> L and read data on D<63:0> L. All 64 bits of data are used during quadword- and octaword-length reads. For longword-length reads, only the lower longword (D<31:0> L) is used. In this case, the value of the upper longword is unspecified. In either case, the full 64 bits are used when checking XMI P<2:0> L.

2.4.5 Locked Response Cycle (LOC)

The Locked Response indicates that the location specified in an Interlock Read transaction was already locked. During this cycle the responder drives 0100 on XMI F<3:0> L and the commander's ID on XMI ID<5:0> L. The value of the data bits, D<63:0> L, is unspecified but must be consistent with P<2:0> L. A Locked Response signals the termination of an Interlock Read transaction. When issued, it is always the first and only read response to the transaction.

2.4.6 Read Error Response Cycle (RER)

The Read Error Response indicates that a Read, Interlock Read, or IDENT transaction completed unsuccessfully due to an error condition at the responder node. The Read Error Response is used for an uncorrectable memory error or a reference to a nonexistent location on the VAXBI. During this cycle the responder drives 0101 on XMI F<3:0> L and the commander's ID on XMI ID<5:0> L. The value of the data bits, D<63:0> L, is unspecified but must be consistent with XMI P<2:0> L. A Read Error Response signals the termination of the transaction, and no further read responses are provided.

2.4.7 The Null Cycle

A null cycle is an unused XMI cycle as no node has requested the bus. The null cycle is ignored by all XMI responders.

2.5 XMI Transactions

XMI transactions are listed in Table 2–11.

Table 2–11 XMI Transactions

Name	Mnemonic
Read	READ
Interlock Read	IREAD
Write Mask	WMASK
Unlock Write Mask	UWMASK
Interrupt	INTR
Identify	IDENT
Implied Vector Interrupt	IVINTR

2.5.1 Read Transaction

The Read (READ) transactions are used to transfer a longword, quadword, octaword, or hexword of data from the responder to the commander. A Read transaction is initiated by a commander driving the XMI address and function lines to represent a longword read, quadword read, octaword read, or hexword read. The Read command cycle is decoded by all responder nodes. The node that recognizes its own address latches that address and command. This node is the responder.

When the responder has the requested data, it initiates a return data transfer. Multiple transfers may be necessary to transfer all the quadwords in a given octaword or hexword transaction. The commander monitors the bus traffic waiting for its return data, and then latches the information. The commander issues its own ID in the ID field during the command cycle. The responder returns this same ID with the return read data so that the commander can recognize the return read data it had requested.

Longword-length transactions can only be used in I/O space while quadword-, octaword-, and hexword-length transactions can only be used in memory space.

2.5.2 Interlock Read Transaction

An Interlock Read (IREAD) transaction, combined with a corresponding Unlock Write transaction, permits mutually exclusive access to memory space locations.

The IREAD transaction works in memory space. This transaction gains access to a shared object in memory. The exact effect of the Interlock Read depends on the state of the memory's lock bit. Quadword-, octaword-, and hexword-length transactions are used in memory space.

Memory has 16 locks on hexword boundaries. If the memory is already locked, memory responds to IREAD with a Locked Response, and no data is returned. This tells the commander that the shared memory structure is not available at this time. The commander responds to the locked response by repeating the IREAD.

If the memory is not locked, memory locks itself to further IREADs upon receipt of an IREAD and provides the data contained in the addressed locations(s) to the commander. Unlocking the memory requires an UWMASK transaction.

The use of Interlock Read transactions in I/O space is implementation dependent. Most I/O locations treat an Interlock Read like a regular READ. Only longword-length transactions can be used in I/O space.

2.5.3 Write Mask Transaction

Write Mask (WMASK) transactions transfer data from the commander to the responder.

WMASK transactions transfer quadwords or octawords from the commander to the memory-space responder, such as the MS62A memory module. The commander gains the XMI and sends a command cycle specifying the type of transaction (a longword Write Mask, quadword Write Mask, or an octaword Write Mask), a byte Write Mask, and the desired address. The commander immediately follows this with one or two cycles of write data. All nodes on the XMI decode the address, and the node that recognizes the address becomes the responder.

The responder accepts the command, address, and data and performs the requested write. The mask field that accompanies each command and address has 16 bits. Each bit corresponds to a byte of data in the associated one or two quadwords. If the bit is zero, then that byte is not written; if the bit is one, then that byte is written.

All cache-resident nodes on the XMI are required to monitor write traffic and perform cache invalidates if the XMI write "hits" a block stored in cache.

The XMI has the concept of a "cache invalidate" transaction that does not result in an update of main memory. A commander can perform an invalidate operation by issuing either a quadword Write Mask or octaword Write Mask command with the mask field equal to all zeros. The size of the region to be invalidated is specified in the Length field. Since an invalidate operation is a degenerate case of a Write Mask transaction, it obeys all the Write Mask transaction requirements, including supplying the appropriate write data cycles consistent with the transaction length. As the write data will be discarded by the responder, the value of XMI D<63:0> L during these cycles is unspecified but is consistent with XMI P<1:0> L.

2.5.4 Unlock Write Mask Transaction

The Unlock Write Mask (UWMASK) transaction, combined with a corresponding Interlock Read transaction, is used to relinquish the locked memory location after an Interlock Read.

After a node successfully gains the lock in memory and finishes the required access to the shared structure, it then relinquishes the lock by performing an UWMASK to the memory with appropriate data. The memory, which has been monitoring the bus traffic, reacts to the Unlock Write by unlocking memory and writing the data in the request.

If an UWMASK transaction is directed to a location that is not currently locked, the responder performs the write operation. An implementation might log this as an error in its CSRs.

UWMASK transactions to I/O space are implementation dependent and can only be longword length.

2.5.5 Interrupt and Identify Transactions

Any I/O device can send an interrupt to one or more processor nodes. A processor eventually issues an Identify and then performs the necessary service routine.

Any I/O adapter on the XMI can send out an Interrupt (INTR) transaction to one or more CPU nodes, as designated by a destination mask. One of the processors eventually issues an Identify (IDENT) transaction at a selected level <7:4> and chooses one interrupting node to send it to. That processor then clears the I/O interrupt but other I/O interrupts (if any) wait to maintain the CPU interrupt request. An interrupt vector is eventually sent to the CPU that issued the IDENT. This CPU then performs the interrupt service routine.

Interrupting nodes do not need to reissue their interrupts after one node/level is serviced. Each CPU monitors the XMI for IDENTs issued by another node. An IDENT issued by one CPU to an interrupting device causes the other processor nodes to clear their corresponding interrupt-pending flag. An interrupting node is not allowed to have more than one interrupt outstanding at a given level.

If more than one processor issues an IDENT for the same interrupt, the first processor node to win the XMI processes the interrupt and the other CPUs clear their corresponding interrupt-pending flags and abort the IDENT by taking a microcode passive release, which is not seen by the operating system

2.5.6 Implied Vector Interrupt Transactions

The Implied Vector Interrupt (IVINTR) is a single-cycle transfer used to implement VAX interprocessor interrupts and write error interrupts where the interrupt priority and interrupt vector are implied by the type of interrupt.

Interprocessor interrupts are issued at IPL 16 (hex) with a vector of 80H. Write error interrupts are issued at IPL 1DH with a vector of 60 (hex). Since the value of the interrupt vector is indicated by the value of the Type field, IVINTR transactions do not require a corresponding IDENT (identify or interrupt acknowledge cycle).

The IVINTR transaction contains a 4-bit Type field used to specify the type of interrupt. Only two bits are used: <16> specifies an interprocessor interrupt, while <17> specifies a write error interrupt. The IVINTR transaction also contains a 16-bit Node Specifier field (one bit per node) indicating which nodes are to be interrupted. Interprocessor interrupt transactions can be directed to more than one node. Write error interrupt transactions are directed to only one node.

2.5.7 Transaction Examples

Examples are found in the following subsections:

- Single Data Cycle Reads
- Multiple Data Cycle Reads
- Longword and Quadword Writes
- Multiple Data Cycle Writes

2.5.7.1 Single Data Cycle Reads

The four types of single data cycle reads are:

- Longword Read
- Longword Interlock Read (IREAD)
- Quadword Read
- Quadword Interlock Read

Figure 2–11 Read Transaction

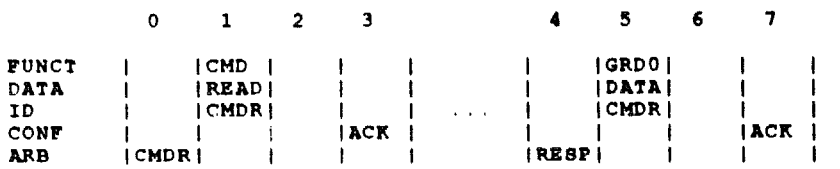
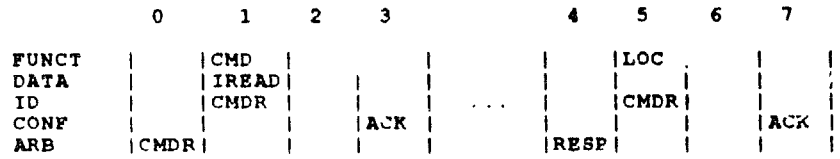


Figure 2–12 Interlock Read Transaction to a Locked Location



The Read transactions consist of a command transfer followed by a return data transfer, as shown in Figure 2–11. The two transfers are the command (FUNCT = CMD) and the read data response (FUNCT = GRD0). The commander arbitrates for the bus in cycle 0 and wins. In cycle 1, it drives the function, command, address of the read, and its own ID (for later use to identify the returning data). In cycle 3, the responder confirms receipt of the information.

Some variable time later, in this example at cycle 4, the return data transfer begins with the responder arbitration for the bus. Having won it, the responder drives the function, the data, and the commander's ID in cycle 5. The status of the returning data is specified in the read response function code, either Good Read Data, Corrected Read Data, or Read Error Response. The commander monitors the bus, checking for an ID match during read data cycles to indicate that the read data is meant for that commander.

If the particular transaction requested had been an Interlock Read, and if the memory was already interlocked, the responder would have provided a Locked Response (LOC) in place of the returned data. (See Figure 2–12.)

2.5.7.2 Multiple Data Cycle Reads

The four types of multiple data cycle reads are:

- Octaword Read
- Octaword Interlock Read
- Hexword Read
- Hexword Interlock Read

Figure 2–13 Multiple Data Cycle Reads Command Cycle

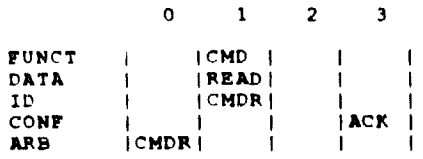


Figure 2–14 Read Data Cycles

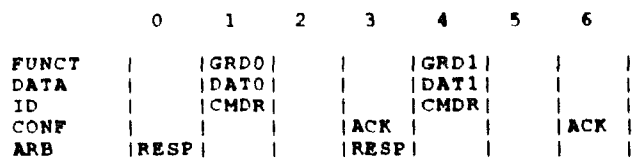
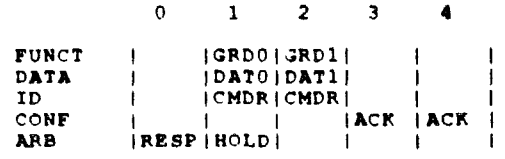


Figure 2–15 Read Data Cycles with HOLD



The four multiple data cycle Read transactions move either 16 bytes (octaword) or 32 bytes (hexword) of data from the responder to the commander. Figure 2–13 is the command transfer of the transaction. The Interlock Read checks the state of the lock bit in the memory and qualifies the request, based on its state. This illustration applies to both octaword and hexword reads.

Figure 2–14 is a diagram of the return data transfer applicable to octaword reads, moving four longwords of data. The function field of the bus in cycle 1 indicates "good read data 0" with the ID field identifying the intended receiver (the transaction commander). Cycle 4 is a Good Read Data 1 cycle. Each cycle provides a new quadword of read data while the ID remains unchanged.

Read data may be returned in consecutive cycles through the use of HOLD, as shown in Figure 2–15. The transmitter asserts HOLD in the first cycle to ensure that it maintains the use of the bus until it completes the transfer. HOLD is the highest priority arbitration line and guarantees use for a maximum of four consecutive cycles. The confirmation is returned to the commander two cycles after the command cycle.

Bus usage during a hexword read with a single correctable read error is shown in Figure 2–16.

Figure 2–17 illustrates the events during a return data of hexword length containing an uncorrectable read error. When memory encounters an uncorrectable read error, it returns a Read Error Response and suppresses further read responses for that transaction.

Figure 2–16 Hexword Read with Single Correctable Read Error

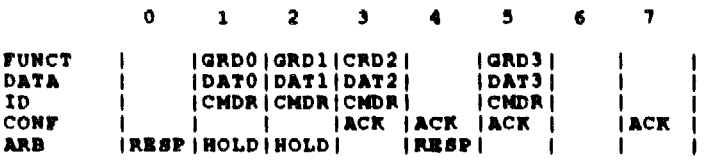
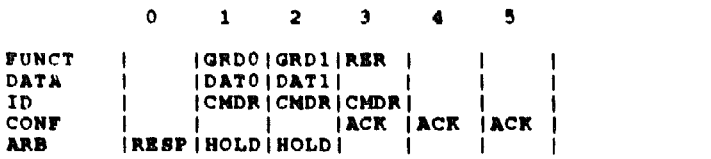


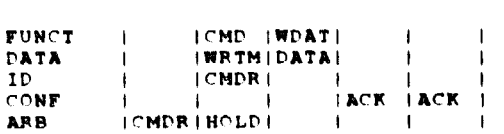
Figure 2–17 Hexword Data Return with Uncorrectable Read Error



2.5.7.3 Longword and Quadword Writes

Longword and quadword writes can be either Write Mask or Unlock Write Mask transactions

Figure 2–18 Longword and Quadword Writes

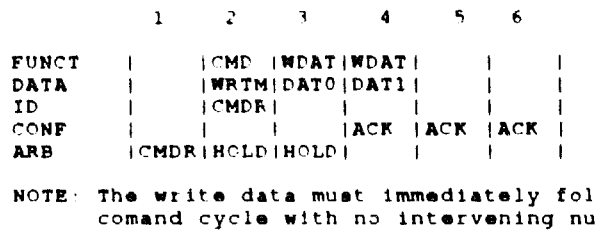


Longword and quadword writes move the number of bytes specified by the Mask field. The commander arbitrates for the XMI bus and, upon winning it, drives the appropriate write command, the intended address, the data mask, its own ID, and asserts HOLD to signal that it will need the next cycle as a write data cycle. It then provides the write data but no ID field, having identified itself in the command cycle. Cycles 3 and 4 show the confirmation from the responder.

2.5.7.4 Multiple Data Cycle Writes

The multiple data cycle writes are the octaword Write Mask and the octaword Unlock Write Mask transactions.

Figure 2–19 Multiple Data Cycle Writes

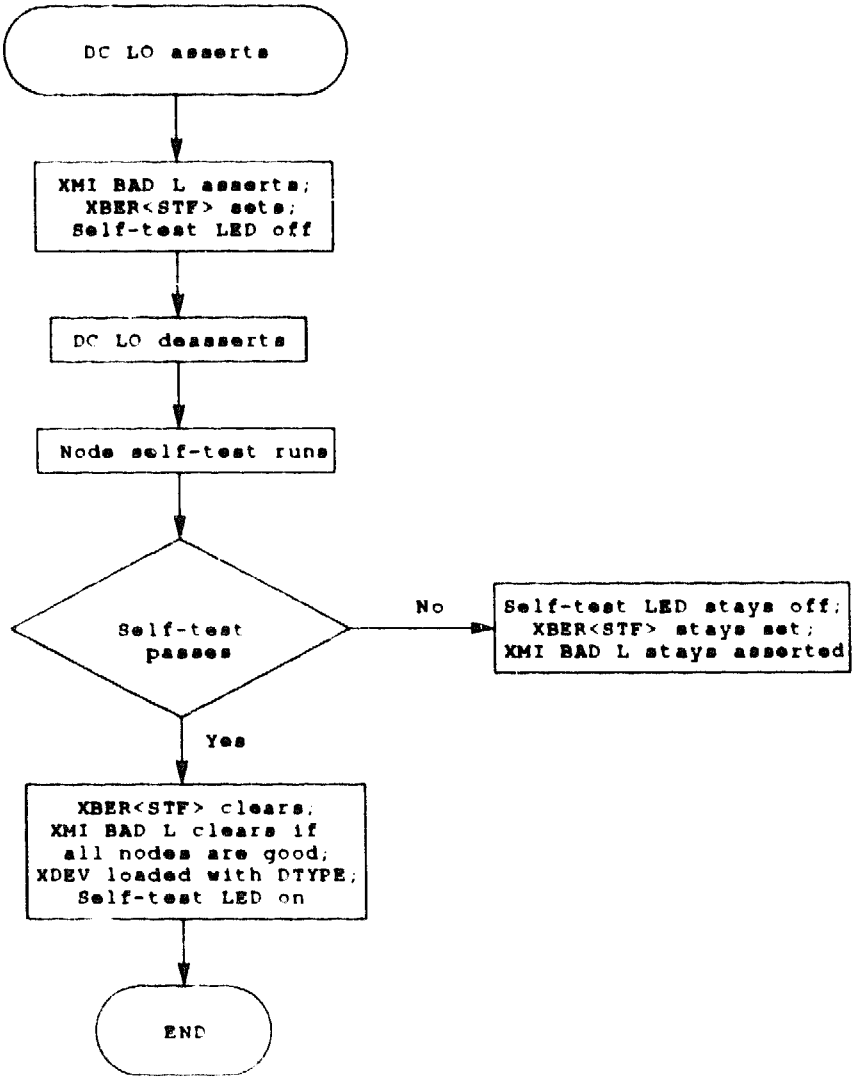


Multiple data cycle writes identify the first cycle of the transfer with the desired write length. HOLD is asserted while successive cyles provide new data.

2.6 XMI Initialization

Regardless of the method used to cause a node to initialize, the initialization process consists of the same steps.

Figure 2–20 XMI Initialization Flowchart



2.6.1 Causes of an Initialization

Three causes of XMI initialization are:

- Power-down/power-up
- System reset
- Node reset

2.6.2 Power-Up

On power-up, the XMI AC LO L, XMI DC LO L, and XMI RESET L lines are sequenced to provide initialization of all nodes in the system. The XMI initialization flowchart is shown in Figure 2–20.

During normal power-up, a node cannot access XMI-accessible memory space locations until the deassertion of XMI AC LO L. However, memory nodes clear memory locations following the deassertion of XMI DC LO L if a cold start is indicated. During a system reset sequence, it is possible for the resetting node to access memory prior to the deassertion of XMI AC LO L, but no other node can access memory prior to the deassertion of XMI AC LO L.

During brownout power conditions, XMI AC LO may assert and later deassert without an assertion of XMI DC LO L. The XMI AC LO L signal remains asserted for a period of time after the deassertion of XMI DC LO L, allowing a node's internal initialization signals to be removed before a power restart interrupt is raised.

XMI DC LO L warns of the impending loss of DC power and is used for initialization on power-up. DC power and the XMI clock become valid before the deassertion of XMI DC LO L. The XMI DC LO L signal is asserted after the assertion of XMI AC LO L, allowing the power-fail routine to save processor state in memory and to halt. The result of any XMI transaction in progress when XMI DC LO L asserts is indeterminate.

XMI DC LO L asserts before the loss of DC power so that nodes such as disk controllers can stop certain activities before the removal of power.

In a power outage, first AC power is lost, then (if not restored quickly), DC power falls below acceptable levels, asserting first XMI AC LO L and then XMI DC LO L.

During a power outage, systems with the battery backup option have their memory nodes supplied with battery power, allowing memory contents to be retained. After power is restored, the memory is not reinitialized. However, if the power outage is lengthy and exhausts the battery, the data in memory is no longer reliable. One function of the XTC power sequencer is to monitor the battery backup power voltage and assert the XMI RESET L line if the battery was exhausted.

2.6.3 System Reset

A power-down/power-up sequence can be emulated through the use of the XMI RESET L line, which causes the sequencing of XMI AC LO L and XMI DC LO L in the same way as a true power-down/power-up sequence. This allows all nodes in the system to be returned (or "reset") to their power-up state without cycling the power supplies. The XTC power sequencer is used to carry out the reset sequence.

A system reset is caused by:

- Software that asserts XMI RESET L by writing to IPR55, IORESET, with an MTPR instruction. For example, the console INITIALIZE command generates a system reset, if no argument is given, by using this mechanism.
- Pushing the control panel Restart button. This causes the assertion of the XMI RESET L signal.

The XTC power sequencer monitors the XMI RESET L line and drives the XMI AC LO L, XMI DC LO L, and XMI RESET L lines. Upon detection of an asserted XMI RESET L line, the XTC power sequencer begins the reset sequence. If XMI RESET L is asserted while XMI AC LO L and XMI DC LO L are deasserted, the XTC power sequencer asserts XMI AC LO L first, then XMI DC LO L, and finally deasserts XMI DC LO L. In response, all XMI nodes perform self-test and initialization. When the RESET line is deasserted, the XTC power sequencer deasserts XMI AC LO L, completing the emulation of the power-down/power-up sequence. If the RESET line remains asserted until after XMI DC LO L is deasserted, then all memory nodes reset, including those with battery backup.

2.6.4 Node Reset

A single node in a system can be reset without resetting the entire system by writing a one to the Node Reset bit (NRST) in the XMI Bus Error Register of that particular node. The node is inaccessible for the duration of its initialization and XMI BAD L is asserted. Accessing the node during self-test may cause a self-test failure. Software drivers that share a node must agree in advance that a node needs to be reset and lock the selection of that node.

The console INITIALIZE command generates a node reset if a node ID argument is provided.

2.7 XMI REGISTERS

This section describes the registers required for all XMI nodes.

Each XMI node is required to have a set of two or three registers in a specified location within the node's nodespace, as shown in Table 2–12. Table 2–13 defines the abbreviations used to describe the type of bits in the register descriptions.

Descriptions of module-specific XMI registers are given in the chapters on the XMI options.

Table 2–12 XMI Registers

Register	Mnemonic	Address	Node Requirements
Device Register	XDEV [1]	BB [2] + 0000 0000	All nodes
Bus Error Register	XBER	BB + 0000 0004	All nodes
Failing Address Register	XFADR	BB + 0000 0008	Commanders only

[1] X in the mnemonic indicates that this is an XMI register.

[2] BB = base address of a node, which is the address of the first location in nodespace.

Table 2–13 Abbreviations for Bit Type

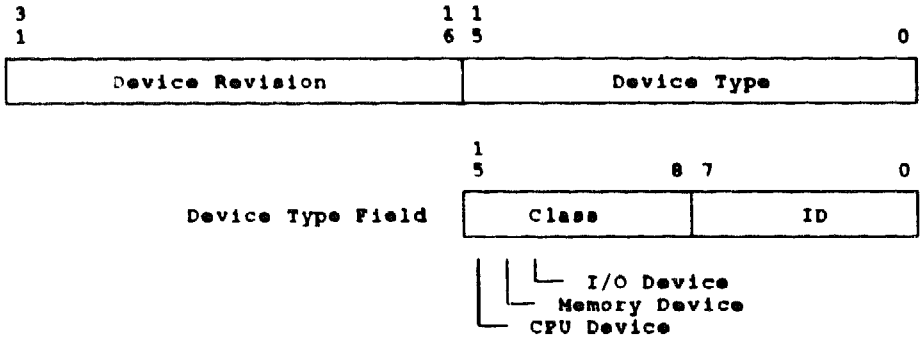
Abbreviation	Definition
0	Initialized to logic level zero
1	Initialized to logic level one
X	Initialized to either logic state
RO	Read only
R/W	Read/write
R/W1C	Read/cleared by writing a 1

Device Register (XDEV)

The Device Register contains information to identify the node. Both fields are loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS

Nodespace base address + 0000 0000



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

Identifies the functional revision level of the device. The use of the Device Revision field is implementation dependent.

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

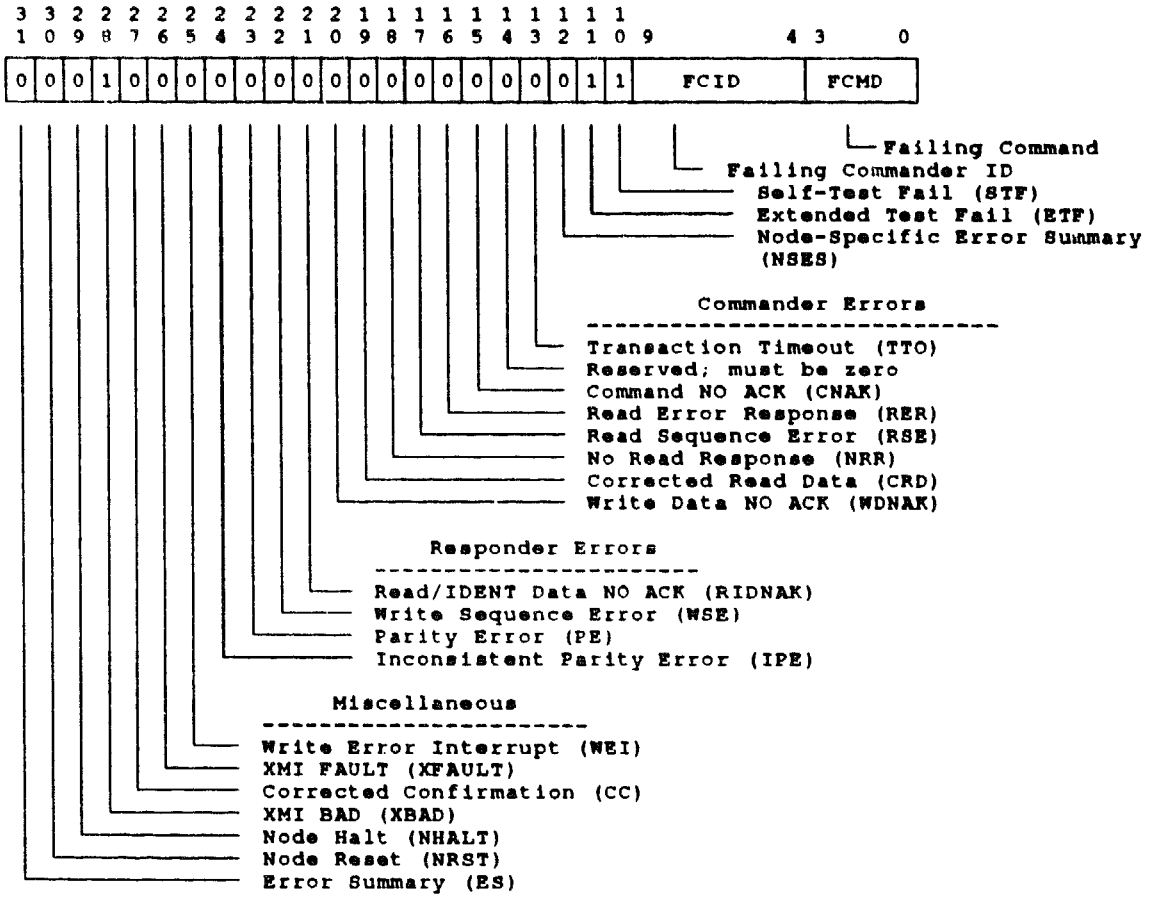
Type: R/W, 0

Identifies the type of node. The Device Type field is broken into two subfields: Class and ID. The Class field indicates the major category of the node. The currently defined classes are CPU, memory, and I/O. The ID field uniquely identifies a particular device within a specified class.

Bus Error Register (XBER)

The Bus Error Register contains error status on a failed XMI transaction. This status includes the failed command, commander ID, and an error bit that indicates the type of error that occurred. This status remains locked up until software resets the error bit(s).

ADDRESS *Nodespace base address + 0000 0004*



bit<31>

Name: Error Summary
Mnemonic: ES
Type: RO, 0

ES represents the logical OR of the error bits in this register. Therefore, ES asserts when any error bit asserts.

bit<30>

Name: Node Reset
Mnemonic: NRST
Type: R/W, 0

Writing a one to NRST initiates a complete power-up reset similar to the assertion and deassertion of XMI DC LO L (see note below); the node performs self-test and asserts XMI BAD L until it is successfully completed. Like power-up reset, nodes are precluded from accessing the node from the time it is node reset until it completes self-test (or the maximum self-test time is exceeded).

NOTE: During the time that a node is responding to node reset, the node does not access other nodes on the XMI bus. In response to a real power-up sequence (caused by XMI DC LO L), the NRST bit will be reset. Following a node reset sequence, it will remain set allowing the processor to recognize that it should not attempt to go through the normal boot process.

bit<29>

Name: Node Halt
Mnemonic: NHALT
Type: R/W, 0

Writing a one to NHALT forces the node to go into a "quiet" state while retaining as much state as possible. The CPU halts and goes into console mode waiting for console commands.

bit<28>

Name: XMI BAD
Mnemonic: XBAD
Type: R/W, 1

On reads, XBAD indicates the state of the XMI BAD signal. A one indicates that BAD is asserted. Writes to this location supply the state to be driven on the wired-OR XMI BAD L line by this node; writing a one asserts XMI BAD L, while writing a zero releases it. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<27>

Name: Corrected Confirmation
Mnemonic: CC
Type: R/W1C, 0

CC sets when the node detects a single-bit CNF error. Single-bit CNF errors are automatically corrected by the XCLOCK chip.

bit<26>

Name: XMI FAULT
Mnemonic: XFAULT
Type: R/W1C, 0

When set, XFAULT indicates that the XMI FAULT signal has been asserted for at least one cycle. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<25>

Name: Write Error Interrupt
Mnemonic: WEI
Type: R/W1C, 0

When set, WEI indicates that the node has received a write error interrupt transaction. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<24>

Name: Inconsistent Parity Error
Mnemonic: IPE
Type: R/W1C, 0

When set, IPE indicates that the node has detected a parity error on an XMI cycle and the confirmation for the errored cycle was ACK. This indicates that at least one node (the responder) detected good parity during the cycle time that this node detected a parity error. Only XMI processor nodes are required to implement this bit. If not implemented, nodes return zero.

bit<23>

Name: Parity Error
Mnemonic: PE
Type: R/W1C, 0

When set, PE indicates that the node has detected a parity error on an XMI cycle.

bit<22>

Name: Write Sequence Error
Mnemonic: WSE
Type: R/W1C, 0

When set, WSE indicates that the node aborted a write transaction due to missing data cycles. Only XMI responder nodes are required to implement this bit. If not implemented, nodes return zero.

bit<21>

Name: Read/IDENT Data NO ACK
Mnemonic: RIDNAK
Type: R/W1C, 0

When set, RIDNAK indicates that a Read or IDENT data cycle (GRDn, CRDn, LOC, RER) transmitted by the node has received a NO ACK confirmation.

bit<20>

Name: Write Data NO ACK
Mnemonic: WDNAK
Type: R/W1C, 0

When set, WDNAK indicates that a Write data cycle (GRDn, CRDn, LOC, RER) transmitted by the node has received a NO ACK confirmation.

bit<19>

Name: Corrected Read Data
Mnemonic: CRD
Type: R/W1C, 0

When set, CRD indicates that the node has received a CRDn read response. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

bit<18>

Name: No Read Response
Mnemonic: NRR
Type: R/W1C, 0

When set, NRR indicates that a transaction initiated by the node failed due to a read response timeout. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

bit<17>

Name: Read Sequence Error

Mnemonic: RSE

Type: R/W1C, 0

When set, RSE indicates that a transaction initiated by the node failed due to a read sequence error. Only XMI commander nodes are required to implement this bit. This bit will be set only if the reattempt fails on commanders implementing error recovery. If this bit is not implemented, nodes return zero.

bit<16>

Name: Read Error Response

Mnemonic: RER

Type: R/W1C, 0

When set, RER indicates that a node has received a Read Error Response. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

bit<15>

Name: Command NO ACK

Mnemonic: CNAK

Type: R/W1C, 0

When set, CNAK indicates that a command cycle transmitted by the node has received a NO ACK confirmation caused by either a reference to a nonexistent memory location or a command cycle parity error. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero. For commanders implementing error recovery, this bit is set only if the reattempts fail.

bit<14>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bit<13>

Name: Transaction Timeout
Mnemonic: TTO
Type: R/W1C, 0

When set, TTO indicates that a transaction initiated by the node failed due to a transaction timeout. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero. For commanders implementing error recovery, this bit is set only if the reattempts fail.

bit<12>

Name: Node-Specific Error Summary
Mnemonic: NSES
Type: RO, 0

When set, NSES indicates that a node-specific error condition has been detected. The exact nature of the error is contained in node-specific registers.

bit<11>

Name: Extended Test Fail
Mnemonic: ETF
Type: R/W1C, 1 (processors), 0 (all others)

When set, ETF indicates that the node has not yet passed its extended test. This bit clears when the node passes its extended test. Only processor nodes implement extended test; all other nodes power up with ETF cleared.

bit<10>

Name: Self-Test Fail
Mnemonic: STF
Type: R/W1C, 1

When set, STF indicates that the node has not yet passed its self-test. This bit is cleared by the user interface when the node passes its self-test.

bits<9:4>

Name: Failing Commander ID
Mnemonic: FCID
Type: RO

This field logs the commander ID of a failing transaction. Only XMI commander nodes are required to implement this field. If not implemented, nodes return zero.

bits<3:0>

Name: Failing Command

Mnemonic: FCMD

Type: RO

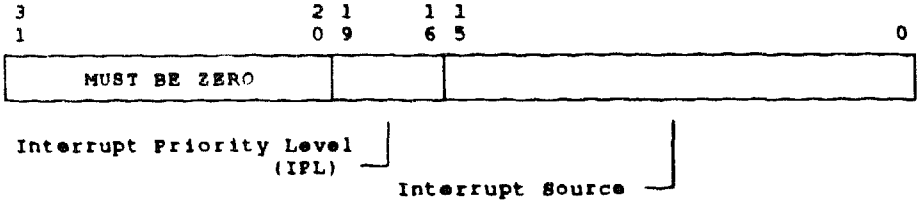
This field logs the command code of a failing transaction. Only XMI commander nodes are required to implement this bit. If not implemented, nodes return zero.

Failing Address Register (XFADR)

The Failing Address Register logs address and length information associated with a failing transaction. Only XMI commander nodes are required to implement this register. There are three interpretations of XFADR, depending on XBER<FCMD>.

ADDRESS *Nodespace base address + 0000 0008*

XFADR, when XBER<3:0> (FCMD) is 9 (hex), an IDENT transaction:



bits<31:20>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<19:16>

Name: Interrupt Priority Level

Mnemonic: IPL

Type: RO

IPL is a bit mask that specifies the interrupt priority level for the IDENT command as shown below:

Bit	IPL (hex)
19	17
18	16
17	15
16	14

bits<15:0>

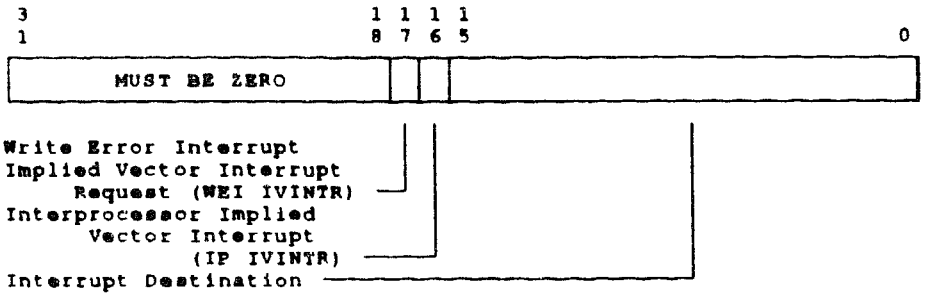
Name: Interrupt Source

Mnemonic: None

Type: RO

The Interrupt Source field is a bit mask that specifies the IDENT command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1.

XFADR, when XBER<3:0> (FCMD) is F (hex), an IVINTR transaction:



bits<31:18>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<17>

Name: Write Error Interrupt Implied Vector Interrupt Request

Mnemonic: WEI IVINTR

Type: RO

WEI IVINTR sets if the implied vector interrupt request was for a write error interrupt.

bit<16>

Name: Interprocessor Implied Vector Interrupt

Mnemonic: IP IVINTR

Type: RO

IP IVINTR sets if the implied vector interrupt request was for an interprocessor interrupt.

bits<15:0>

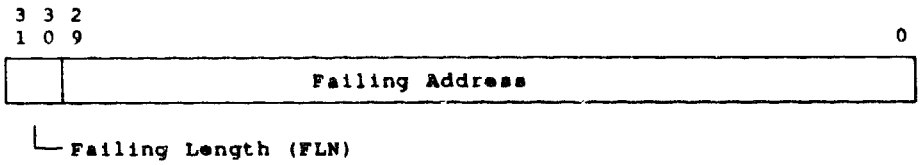
Name: Interrupt Destination

Mnemonic: None

Type: RO

The Interrupt Destination field is a bit mask that specifies the IVINTR command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1.

XFADR, when XBER<3:0> (FCMD) is neither an IDENT transaction nor an IVINTR transaction:



bits<31:30>

Name: Failing Length

Mnemonic: FLN

Type: RO

FLN logs the value of XMI D<31:30> during the command cycle of a failing transaction.

bits<29:0>

Name: Failing Address

Mnemonic: None

Type: RO

The Failing Address field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

2.8 XMI Errors

The XMI bus detects all single-bit transmission-related errors on XMI D<63:0> L, XMI F<3:0> L, XMI ID<5:0> L, XMI P<2:0> L, and XMI CNF lines. The XMI protocol permits XMI commanders to recover from all transient memory space read/write transaction errors as well as from most I/O space read/write transaction errors.

2.8.1 Error Conditions

2.8.1.1 Parity Error

To detect single-bit errors, all nodes monitor parity of the bus. Any XMI receiver detecting bad parity ignores the cycle and returns a NO ACK confirmation.

2.8.1.2 Inconsistent Parity Error

Under certain error conditions, some nodes might detect bad parity while others compute proper parity. If the intended target of the transaction computes good parity, then the cycle may be ACKed (and assumed good by the commander), even if other nodes ignore the cycle due to bad parity. For XMI memory-space write transactions, this class of error may result in cache coherency problems due to cached processors failing to perform cache invalidates. For XMI IVINTR transactions, some destinations of the IVINTR transaction may not receive the interrupt. All other XMI transactions are insensitive to this class of error.

2.8.1.3 Transaction Timeout

The XMI protocol specifies that a timeout of 16 milliseconds be used by commanders to detect transaction failure. Responders ensure that transactions do not exceed these timeout values.

- Response Timeout—During an XMI Read, Interlock Read, or IDENT transaction, if a commander does not receive all read responses within a certain number of cycles after the transaction is issued, the transaction is considered to have failed. This does not imply that a responder has "died" since XMI receivers ignore cycles with bad parity and response timeouts can occur as a result of ignored cycles.
- Retry Timeout—An XMI commander needs to reissue an XMI transaction if it receives a NO ACK or a Locked Response. If the commander has not successfully completed the transaction within the timeout period, the transaction has failed.

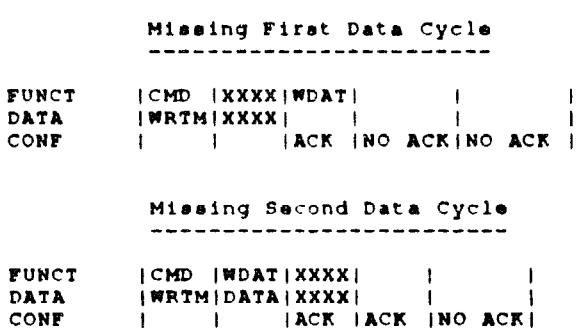
2.8.1.4 Sequence Error

Many transactions require that XMI cycles occur in a certain sequence. When the cycles occur out of sequence, the transaction is in error.

Read, Interlock Read, and IDENT transactions use sequence IDs embedded in the read data responses (GRDn, CRDn, RER—the sequence ID for RER is implicitly 0). The required order for read responses is 0, 0, 0...1, and 0...3 for longword (including IDENT), quadword, octaword, and hexword length transactions, respectively. For example, if the commander detects data returned out of sequence (such as GRD0, GRD2, GRD3), then it NO ACKs the out-of-order read response (GRD2) and the subsequent read response (GRD3) for that transaction.

Correct sequencing of write transactions is determined by the location of the write data cycles relative to the write command cycle rather than using sequence IDs. The write command cycle and associated write data cycles must occur in contiguous timeslots. If a responder detects missing data cycles in a write transaction, the incorrect cycle (and subsequent write data cycles) are NO ACKed. Figure 2–21 shows examples of failing octaword write transactions.

Figure 2–21 Failed Octaword Write Transaction



2.8.2 Error Handling

XMI commanders and responders react to error conditons as follows:

- Receivers that detect bad parity ignore the cycle.
- Responders ignore any write transactions containing a sequence or parity error; that is, none of the data at the referenced location is modified because the entire write transaction is ignored.
- Responders receiving a NO ACK confirmation to a read response do not transmit further read responses associated with that transaction within 10 XMI cycles of the NO ACK.
- Memory nodes do not set a lock bit unless all read responses associated with an Interlock Read transaction receive an ACK confirmation.
- Memory nodes do not clear a lock bit unless all write data cycles associated with the Unlock Write Mask transaction are properly received.
- Cached processors detecting an inconsistent parity error either flush their caches or perform a machine check.

2.8.3 Error Recovery

Error recovery involves one or more reattempts of the failed transaction before reporting a hard error. A failed XMI transaction is retried under the following circumstances:

- All transactions receiving a NO ACK confirmation for the command cycle are retried automatically by the hardware. The NO ACK can result from either a reference to nonexistent memory locations (NXM) or from bus parity errors. Transactions failing the retry are assumed to be to an NXM.
- Failing XMI Write transactions are retried.
- XMI IDENT transactions receiving a response timeout are retried. Since this may result in a lost interrupt vector, the consequences are implemented by software.
- Failing XMI I/O space Write Mask or Unlock Write Mask transactions are retried.
- Failing DWMBA I/O space Read or Interlock Read transactions receiving a response timeout are NOT retried since some I/O devices might have read side effects.

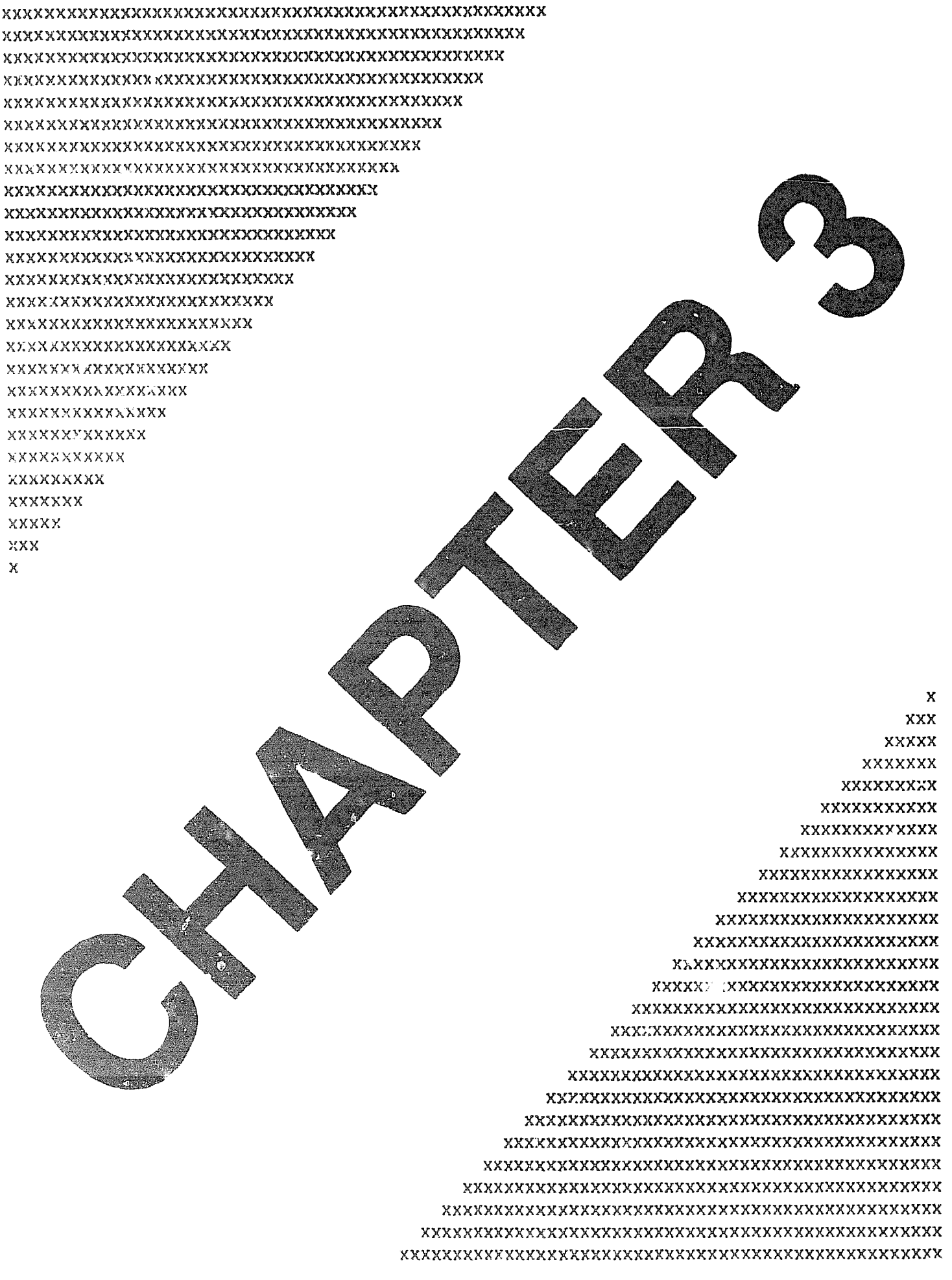
2.8.4 Error Reporting

The XMI bus protocol supports two mechanisms that signal error conditions to processors if normal transaction-level error reporting cannot be used.

Normal transaction-level error reporting mechanisms include NO ACK, Read Error Response (RER), and timeout. The mechanisms that signal error conditions to processors if normal transaction-level error reporting cannot be used are:

- Write error interrupt—This transaction is directed to one or more CPU nodes, resulting in each targeted CPU taking an IPL 1D (hex) error interrupt. The CPU then identifies the source of the write error interrupt.
- XMI FAULT—When XMI FAULT is asserted, all XMI CPUs take an IPL 1D (hex) error interrupt .

An example of a write error interrupt is if the DWMBA is unable to complete either an XMI-to-VAXBI windowed write operation or a VAXBI-to-XMI windowed write operation. Then the DWMBA issues a write error IVINTR transaction to the nodes designated in the DWMBA AIVINTR destination register.


CHAPTER 3

3 KA64A CPU Module

This chapter describes the KA64A CPU module, the processor for the VAX 6000–400 system.

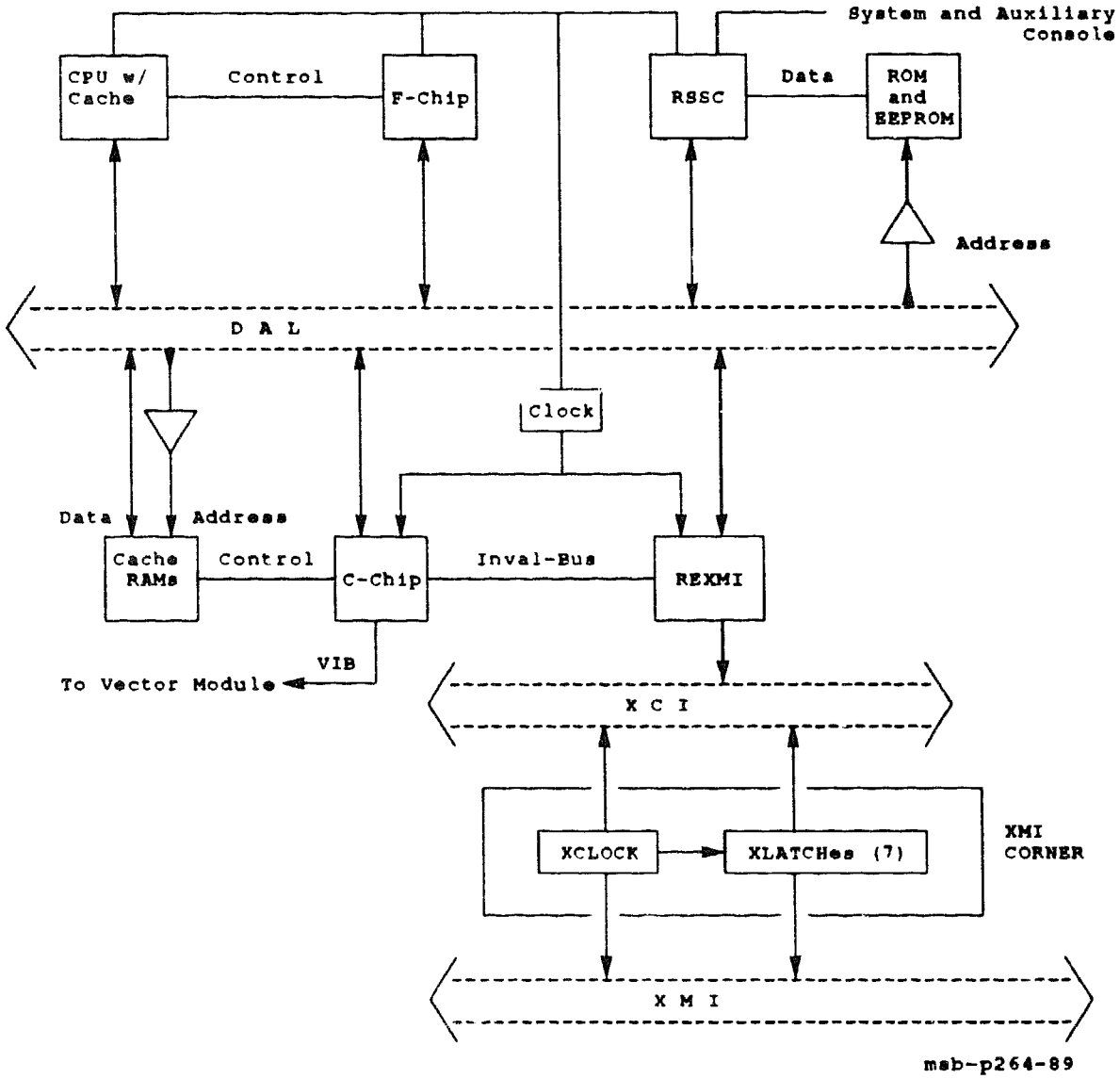
This chapter includes the following sections:

- Overview
- Block Diagram Description
- CPU Section
- Cache Memory
- System Support Chip
- XMI Interface
- Scalar/Vector Interaction
- KA64A CPU Module Registers
- Initialization, Self-Test, and Booting
- Interprocessor Communication Through the Console Program
- Error Handling

3.1 Overview

The KA64A CPU module is a single-board VAX CPU using five chips for the processor section and three chips for the REXMI interface to the XMI bus.

Figure 3–1 KA64A CPU Module Block Diagram



The processor set consists of five chips:

- CPU-chip (DC520)
- F-chip (floating-point accelerator chip) (DC523)
- C-chip (backup cache controller chip) (DC592)
- Clk-chip (clock distribution chip) (DC521)
- RSSC (system support chip) (DC549)

The REXMI interface set consists of three chips:

- Two XDP (data path chips)
- An XCA (control chip)

The two chipsets implement a VAX CPU module with the following features:

- A VAX CPU that supports the 242-instruction VAX base instruction group and associated data types, full VAX memory management, and a 4-Gbyte virtual address space. For more information, see the *VAX Architecture Reference Manual*.
- Support for 512 Mbytes of physical memory.
- A floating-point accelerator that improves the execution of the F, D, and G_format floating-point instructions and the longword variants of integer multiply.
- A two-level instruction and data cache subsystem. A 2-Kbyte primary cache is located on the CPU-chip while a 128-Kbyte backup cache is implemented with external data RAMs and is controlled by the C-chip. Both caches contain I- and D-stream data and both caches implement tag and data parity protection.
- Control for an optional vector module.
- A VAX-compatible macrocoded console program.
- A set of processor clock registers that support the following:
  - — A VAX-standard time-of-year (TOY) clock with battery backup
  - — An interval timer with 10 millisecond interrupts
  - — Two programmable timers, similar in function to the VAX-standard interval timer
- A bootstrap and diagnostic facility that provides:
  - — Full microcode and macrocode power-up self-testing
  - — Node initialization
  - — Booting from various VAXBI devices

- An XMI interface that includes:
  - A write buffer with four octaword entries that decreases the bus and memory bandwidth use by packing writes into larger, more efficient blocks prior to sending them to main memory.
  - Octaword cache fill logic that loads the second-level cache with four longwords of data on each cache miss.
  - XMI write monitoring logic that uses the Inval-Bus to the C-chip to identify write addresses that must be invalidated in both caches.
  - Full support of the XMI bus protocol's error recovery and logging requirements.

3.2 Block Diagram Description

The KA64A CPU module consists of four major blocks:

1. **CPU and floating-point accelerator**
2. **C-chip and backup cache**
3. **System support chip**
4. **XMI interface**

Figure 3–1 shows the KA64A CPU module block diagram.

3.2.1 CPU and Floating-Point Accelerator

The CPU-chip and F-chip cooperate as the CPU section of the KA64A CPU module. They implement the base instruction group of the VAX architecture.

The CPU-chip is a pipelined CPU that provides the hardware and microcode needed to execute instructions, handle exceptions, and otherwise implement the VAX architecture. The CPU-chip contains a 64-entry, fully associative translation buffer where both process and system-space mappings are kept. The CPU-chip includes a 2-Kbyte, direct-mapped instruction and data cache with a quadword block and fill size. Access to this cache takes one cycle. The data/address lines (DAL) interface is a high-performance, fully-handshaked, synchronous bus that includes a write buffering protocol to isolate the instruction pipeline from the DAL.

The F-chip is a pipelined execution unit that enhances the computation phase of floating-point and certain integer instructions. The F-chip receives operands from the CPU-chip, computes the results, and passes the result and status back to the CPU-chip to complete the instruction.

3.2.2 C-Chip and Backup Cache

The C-chip implements the tag store and necessary control for a 128-Kbyte backup cache using 24 16-K x 4 data RAMs on the module. The backup cache is direct-mapped, with a quadword access size, an octaword fill size, and a four-octaword allocate (block) size. The cache is read-allocate, no write-allocate, and write through. The C-chip maintains a tag store that has 2048 entries (tags), one for each of the four-octaword cache blocks. Access to data in the backup cache takes two cycles longer than access to the primary cache.

The C-chip includes copies of the primary cache tags and a special address interface (the Inval-Bus) through which the REXMI can check XMI write addresses against the data in both caches. If the address corresponds to cached data, the REXMI requests an invalidate cycle on the DAL to purge the cached data.

The C-chip also provides the operand and control interface between the KA64A CPU module and an optional vector module. The C-chip issues vector instructions over the vector interface bus (VIB) to the vector module, which then executes the instruction, including all memory references needed to load or store vector registers.

3.2.3 System Support Chip

The RSSC system support chip provides functions to support the CPU in a system environment. Included in the RSSC is support for external ROM/EEPROM, one Kbyte of battery-backed-up RAM, console terminal UARTs, bus reset logic, an interval timer, programmable timers, a time-of-year clock, bus timeout logic, and halt arbitration logic.

3.2.4 XMI Interface

The XMI interface is between the DAL bus and the XMI bus and consists of the XMI Corner (one XCLOCK and seven XLATCHes) and the three-chip REXMI set.

3.3 CPU Section

The CPU section of the KA64A CPU module consists of the CPU-chip and the F-chip floating-point accelerator that cooperate to execute the VAX base instruction group and provide full VAX memory management. For more information, see the *VAX Architecture Reference Manual*.

The CPU description that follows includes the following topics:

- Data Types
- Instruction Set
- Memory Management
- Exceptions and Interrupts
- System Control Block
- Process Structure
- Floating-Point Accelerator

3.3.1 Data Types

The KA64A CPU module supports the following subset of VAX data types:

- Byte
- Word
- Longword
- Quadword
- Character string
- Variable-length bit field
- Absolute queues
- Self-relative queues
- F_floating
- G_floating
- D_floating

The remaining VAX data types are supported by macrocode emulation.

3.3.2 Instruction Set

The KA64A CPU module supports the following instruction classes:

- Integer arithmetic and logical
- Address
- Variable-length bit field
- Control
- Procedure call
- Miscellaneous
- Queue
- Character string
- Operating system support
- F_floating
- G_floating
- D_floating

The KA64A CPU module has special microcode to aid the macrocode emulation of the following instruction groups:

- MATCHC, MOVTC, MOVTUC
- Decimal string
- CRC
- EDITPC

The following instruction groups are not implemented but are emulated by macrocode:

- Octaword
- H_floating
- POLYF, POLYD, and POLYG
- EMODF, EMODD, and EMODG
- ACBF, ACBD, and ACBG
- Compatibility-mode instructions

The KA64A CPU module decodes the VAX vector instruction set and passes operands and control information to the optional vector module, if one is installed. If a vector module is not installed, execution of a vector instruction results in a reserved instruction fault.

3.3.3 Memory Management

The KA64A CPU module implements full VAX memory management. System space addresses are mapped through single-level page tables, and process space addresses are mapped through two-level page tables. Refer to the *VAX Architecture Reference Manual* for descriptions of the virtual-to-physical address translation process and the format for VAX page table entries (PTEs).

3.3.3.1 Translation Buffer

The CPU-chip includes a 64-entry, fully associative, translation buffer to reduce the overhead associated with translating virtual addresses to physical addresses. The translation buffer caches VAX PTEs. Each entry stores a PTE for translating virtual addresses in either VAX process space or VAX system space. Each entry is divided into two parts: a 24-bit tag register and a 27-bit PTE register.

The tag register is used to store the virtual page number (VPN) of the virtual page that the corresponding PTE register maps. The tag register also stores a valid bit (TB.V) that indicates a valid VPN in the tag. The PTE register stores the 21-bit page frame number (PFN) field, the PTE.V bit, the PTE.M bit, and the 4-bit protection (PROT) field from the corresponding VAX PTE.

During virtual to physical address translation, the contents of the 64 tag registers are compared with the VPN field (bits <31:9> of the virtual address of the reference). If there is a match with one of the tag registers and the TB.V bit indicates that the entry is valid, then a translation buffer "hit" has occurred, and the contents of the corresponding PTE register are used for the translation.

If there is no match, the translation buffer does not contain the necessary VAX PTE information to translate the address of the reference. The PTE that maps the page is fetched from memory and the translation buffer is updated by replacing the entry at the location indicated by the replacement pointer. The replacement algorithm is Not Last Used (NLU), because hardware does not replace the last valid transaction accessed. The replacement pointer is called the NLU pointer.

3.3.3.2 Memory Management Control Registers

Four internal processor registers (IPRs) control memory management:

- IPR56, Memory Management Enable Register (MAPEN)
- IPR57, Translation Buffer Invalidate All Register (TBIA)
- IPR58, Translation Buffer Invalidate Single Register (TBIS)
- IPR63, Translation Buffer Check Register (TBCHK)

Three pairs of IPRs specify the base and length of P0, P1, and S0 space:

- IPR8, P0 Base Register (P0BR)
- IPR9, P0 Length Register (P0LR)
- IPR10, P1 Base Register (P1BR)
- IPR11, P1 Length Register (P1LR)
- IPR12, System Base Register (SBR)
- IPR13, System Length Register (SLR)

Two IPRs are used by diagnostic software to test the translation buffer:

- IPR47, Translation Buffer Tag Register (TBTAG)
- IPR59, Translation Buffer Data Register (TBDATA)

Memory management is enabled/disabled using MAPEN, IPR56. Writing zero to MAPEN with a Move To Processor Register (MTPR) instruction disables memory management; a one enables. MAPEN is read with a Move From Processor Register (MFPR) instruction to determine if memory management is enabled.

NOTE: The contents of the translation buffer are UNPREDICTABLE whenever memory management is disabled. The CPU-chip flushes the entire translation buffer contents before memory management is enabled.

Translation buffer entries that map a particular virtual address are invalidated by writing the virtual address to TBIS (IPR58) using the MTPR instruction.

CAUTION: All affected process pages MUST be invalidated in the translation buffer whenever software changes a valid PTE for the system or the current process region, or whenever software changes a system PTE that maps any part of the current process page table.

The entire translation buffer is invalidated by writing a zero to TBIA (IPR57) using the MTPR instruction.

The base and length of the P0, P1, and S0 page tables are changed by writing the appropriate address or length to P0BR, P0LR, P1BR, P1LR, SBR, or SLR. The entire translation buffer is flushed whenever a change is made to any of these six registers.

NOTE: A full invalidation of the translation buffer, whether performed as the result of an explicit write to TBIA or as an implied clear due to writes to MAPEN or any base/length register, resets the NLU pointer to the first location in the translation buffer.

When a process context is loaded with the Load Process Context (LDPCTX) instruction, all translation buffer entries that map process-space pages are automatically invalidated. System-space mappings are preserved.

Changes to memory management parameters due to an MTPR instruction to P0BR, P0LR, P1BR, P1LR, SBR, SLR, MAPEN, TBIA, or TBIS are sent to the vector module if one is installed. Changes due to an LDPCTX instruction are not sent to the vector module.

To determine if the translation buffer contains a valid translation for a particular virtual page, write a virtual address within that page to TBCHK using an MTPR instruction. If the translation buffer contains a valid translation for the page, the condition code V bit (PSL<1>) would be set.

Diagnostic software uses TBTAG and TBDATA to test the operation of the translation buffer. A write to TBTAG writes bits <31:9> of the source data into the VPN field of the current tag location and clears the TB.V bit. A subsequent write to TBDATA interprets the source data as a page table entry (PTE) and writes PTE.V, PTE.M, PTE.PROT, and PTE.PFN into the current PTE location, sets the TB.V bit, and increments the NLU pointer.

TBTAG and TBDATA are for diagnostic purposes only and are not to be written during normal operation. The following restrictions apply to writes to these registers:

- The NLU pointer must be in a known state. A write of zero to TBIA initializes it to the first location in the array.
- Memory management must be enabled before using TBTAG and TBDATA, since writing to MAPEN implicitly writes a zero to TBIA and resets the NLU pointer.
- During diagnostic use of TBTAG and TBDATA, data- and instruction-stream references must not change the NLU pointer.

The TBIS, TBIA, TBCHK, TBTAG, and TBDATA IPRs are write only. A MFPR instruction to them causes a reserved operand fault.

3.3.4 Exceptions and Interrupts

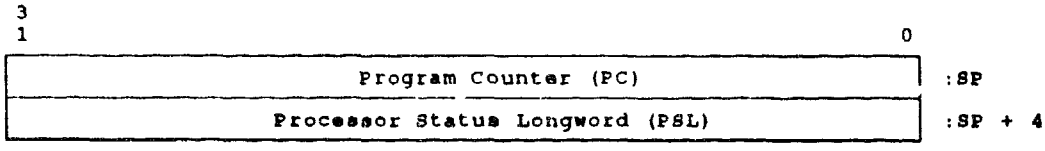
Sometimes events require execution of software routines outside the explicit flow of instructions.

An **exception** is an event caused by the currently executing process that invokes a software routine in the context of the currently executing process. Exception handlers are often system routines, not process routines.

An **interrupt** is an event caused by some activity outside the current process that invokes a software routine outside the context of the current process.

The CPU chip reports exceptions and interrupts by constructing a frame on the stack and then dispatching to the service routine through an event-specific vector in the system control block (SCB). The minimum stack frame for any interrupt and exception is a program counter/processor status longword (PC/PSL) pair, as shown in Figure 3–2.

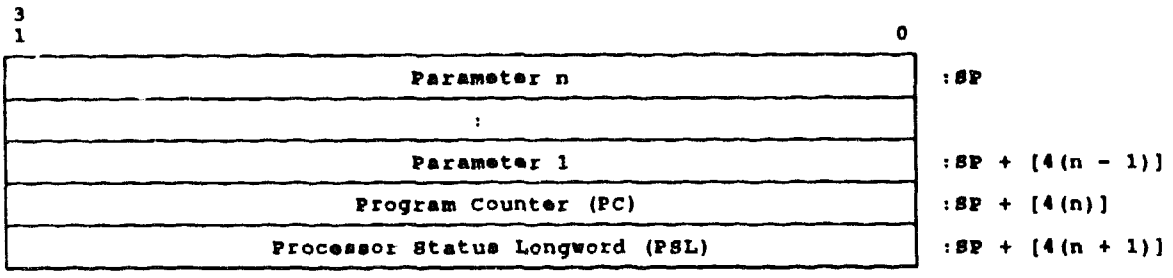
Figure 3–2 Minimum Stack Frame



This minimum stack frame is used for all interrupts. Certain exceptions expand the stack frame by pushing additional parameters on the stack above the PC/PSL pair, as shown in Figure 3–3.

The parameters that are pushed on the stack above the PC/PSL pair, if any, depend on the exception being reported.

Figure 3–3 Large Stack Frame



3.3.4.1 Interrupts

A subset of the 31 VAX interrupt priority levels (IPLs) is implemented by the CPU-chip. When an interrupt request is generated, the CPU-chip hardware compares the request with the current IPL of the CPU. If the new request is of higher priority, an internal request is generated. At the completion of the current instruction, or at selected points during the execution of interruptable instructions, a microcode interrupt handler is invoked to process the request. The microcode handler, with hardware assistance, determines the highest priority interrupt, updates the IPL, pushes a PC/PSL pair on the stack, and dispatches to a macrocode interrupt handler through the appropriate location in the SCB.

Multiprocessor invalidate serialization is guaranteed only for device or special interrupts requested by the REXMI at IPL 14 through 17 (hex).

The interrupt system is controlled by three IPRs:

- IPR18, Interrupt Priority Level (IPL) Register
- IPR20, Software Interrupt Request Register (SIRR)
- IPR21, Software Interrupt Summary Register (SISR)

The IPL register is used for loading the interrupt priority level field in PSL<20:16>. The SIRR is used for creating software interrupt requests. The SISR records pending software interrupt requests at levels 1 through 15.

Table 3–1 lists the IPLs implemented by the KA64A CPU module.

Table 3–1 KA64A CPU Module Interrupts

Interrupt Level (hex)	Interrupt Condition	SCB Vector (hex)
1F – Forced console entry or Machine check	CTRL/P typed at the console, Node Halt bit (XBER<29>) set, node reset, or system reset	None; the console is entered using the console halt procedure and is nonmaskable.
1F – Machine check	Machine check	04
1E – Power Fail	XMI AC LO L assertion	0C
1D – "Hard" error notification	XMI FAULT assertion (XBER<26> set)	60
	XMI write error interrupt (XBER<25> set)	60
	XMI inconsistent parity error (XBER<24> set)	60
	DAL write parity error (RCSR<28> set)	60
	All forms of IDENT errors where retry fails or RCSR<9>=1	60
	All forms of Write error where retry fails or RCSR<9>=1	60
	Vector module hard errors VINTSR<2>, VINTSR<7>	60
1C – 1B	Unused	
1A – "Soft" error notification	P-cache tag parity error on read, write, or invalidate (PCSTS<8> set)	54
	P-cache data parity error on I-stream or nonrequested D-stream read (PCSTS<10> set)	54
	Data parity error on I-stream or nonrequested D-stream read (PCSTS<9> set)	54
	Memory error on I-stream or nonrequested D-stream read (PCSTS<11> set)	54
	Cache fill error (RCSR<27>) set if RCSR<15> not set	54
	Parity error on an XMI cycle (XBER<23> set)	54
	Corrected XMI CNF error (XBER<27> set)	54
	Correctable read data errors (XBER<19>) set if RCSR<14> not set	54
	C-chip backup tag store parity errors (BCSTS<1>, BCSTS<2>, and/or BCSTS<3> set)	54
	C-chip primary tag store parity errors (BCSTS<20> and/or <19> set)	54
	C-chip DAL protocol errors (BCSTS<4> set)	54
	Vector module soft errors (VINTSR<1> set)	54

Table 3–1 (Cont.) KA64A CPU Module Interrupts

Interrupt Level (hex)	Interrupt Condition	SCB Vector (hex)
19 – 18	Unused	
17 – Device interrupt	XMI Level 7 interrupt (INTR)	Supplied by the device
16 – Device or special interrupt	XMI interprocessor interrupt (IVINTR)[1]	80
	XMI level 6 interrupt (INTR)	Supplied by the device
	Interval timer interrupt	C0
15 – Device or special interrupt	Console terminal receive interrupt[1]	F8
	Console terminal transmit interrupt	FC
	Programmable timer interrupt (timer 0 takes priority over timer 1)	Programmable by writing to the TIVR*n* register
	XMI level 5 interrupt (INTR)	Supplied by the device
14 – Device interrupt	XMI level 4 interrupt (INTR)	Supplied by the device
13 – 10	Unused	
0F – 01	Software interrupt request	80 to 9C indexed by the level

[1]At this IPL, the priority of interrupts is shown in descending order.

3.3.4.2 Exceptions

Table 3–2 lists the KA64A CPU module-specific instances of the seven classes of exceptions in the VAX.

Table 3–2 KA64A CPU Module Exceptions

Exception Class	Instances
Arithmetic traps/faults	Integer overflow trap
	Integer divide-by-zero trap
	Subscript range trap
	Floating overflow fault
	Floating divide by zero fault
	Floating underflow fault
Memory management exceptions	Access control violation fault
	Translation not valid fault
Operand reference exceptions	Reserved addressing mode fault
	Reserved operand fault or abort
Instruction execution exceptions	Reserved/privileged instruction fault
	Emulated instruction fault
	Extended function (XFC) fault
	Change-mode trap
	Breakpoint fault
Tracing exceptions	Trace fault
System failure exceptions	Kernel stack not valid abort
	Interrupt stack not valid abort
Machine-check exceptions (aborts)	P-cache read error abort
	DAL read error abort
	DAL write error abort
	F-chip status error abort
	Translation buffer status error abort
	Internally detected inconsistency abort

Exceptions fall into one of three types:

- Traps
- Faults
- Aborts

A **trap** occurs at the end of an instruction. This means that the PC saved on the stack points to the next instruction had the trap not occurred.

A **fault** occurs during an instruction that leaves the registers and memory in a consistent state so that eliminating the fault condition and restarting the instruction gives correct results. After the instruction faults, the PC saved on the stack points to the instruction that was executing when the fault occurred.

An **abort** occurs during an instruction that leaves the value of the registers and memory UNPREDICTABLE, so that the instruction cannot be restarted, completed, simulated, or undone. In most cases the CPU-chip's microcode attempts to convert an abort into a fault by restoring the state that was present at the start of the instruction that caused the abort.

Refer to Section 3.3.7 for an important caution on operating system exception handlers.

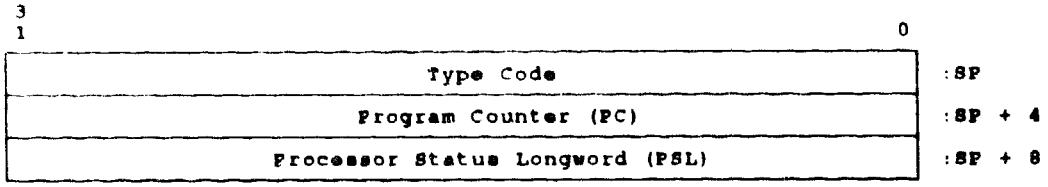
3.3.4.3 Unique Exceptions

The following exceptions are unique to the CPU-chip. The other exceptions are described in the *VAX Architecture Reference Manual*.

Arithmetic Exceptions

Arithmetic exceptions are detected during the execution of integer or floating-point arithmetic instructions. The exception is reported as either a trap or a fault, depending on the specific event. Figure 3–4 shows the arithmetic exception stack frame.

Figure 3–4 Arithmetic Exception Stack Frame



The exceptions are reported in the manner shown in Table 3–3.

Table 3–3 Arithmetic Exceptions Type Codes

Code (hex)	Type	Exception
1	Trap	Integer overflow
2	Trap	Integer divide-by-zero
7	Trap	Subscript range
8	Fault	Floating overflow
9	Fault	Floating divide-by-zero
A	Fault	Floating underflow

Memory Management Exceptions

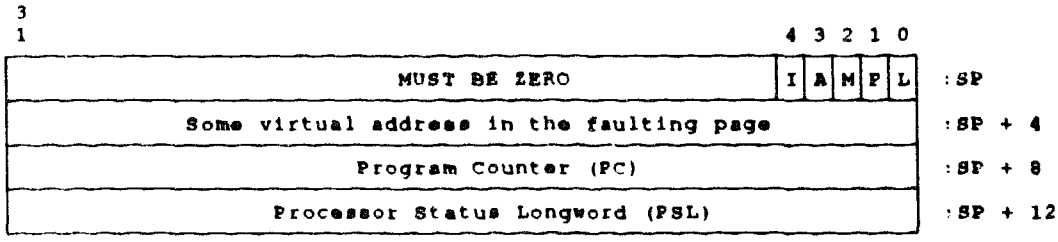
Memory management exceptions are detected during a memory reference and are always reported as faults. The memory management exceptions are listed in Table 3–4.

Table 3–4 Memory Management Exceptions

SCB Vector (hex)	Exception
20	Access control violation
20	Vector alignment fault (vector module only)
20	I/O space vector reference (vector module only)
24	Translation not valid

All memory management exceptions push the same frame on the stack, as shown in Figure 3–5.

Figure 3–5 Memory Management Exception Stack Frame



The M, P, and L bits (bits<2:0>) of the parameter pointed to by the stack pointer are described in the *VAX Architecture Reference Manual* under Memory Management Faults and Parameters. The A bit (bit<3>) is used to distinguish an access control violation from a vector alignment fault, which are both reported through SCB vector 20 (hex). When A is zero, the exception is an access control violation; when A is one, the exception is a vector alignment fault. For all other memory management faults, or if a vector module is not installed, A is zero.

The I bit (bit<4>) indicates that the exception reported through SCB vector 20 (hex) was caused by a vector module reference to an I/O space address.

Emulated Instruction Exceptions

The CPU-chip implements the VAX base instruction group and provides microcode that supports the macrocode emulation of certain other instructions. Two types of emulation exceptions depend on the state of PSL<27> (First Part Done, FPD). If FPD is zero at the beginning of the instruction, the instruction has no microcode assistance and the exception is reported through SCB vector C8 (hex) as a trap with the stack frame shown in Figure 3–6 and the stack frame's parameters listed in Table 3–5.

If PSL<FPD> is a one at the beginning of the instruction, the instruction has microcode assistance and the exception is reported through SCB vector CC (hex) as a fault with the stack frame shown in Figure 3–7. In this case, PC is the opcode of the emulated instruction.

Figure 3–6 Emulated Instruction Trap

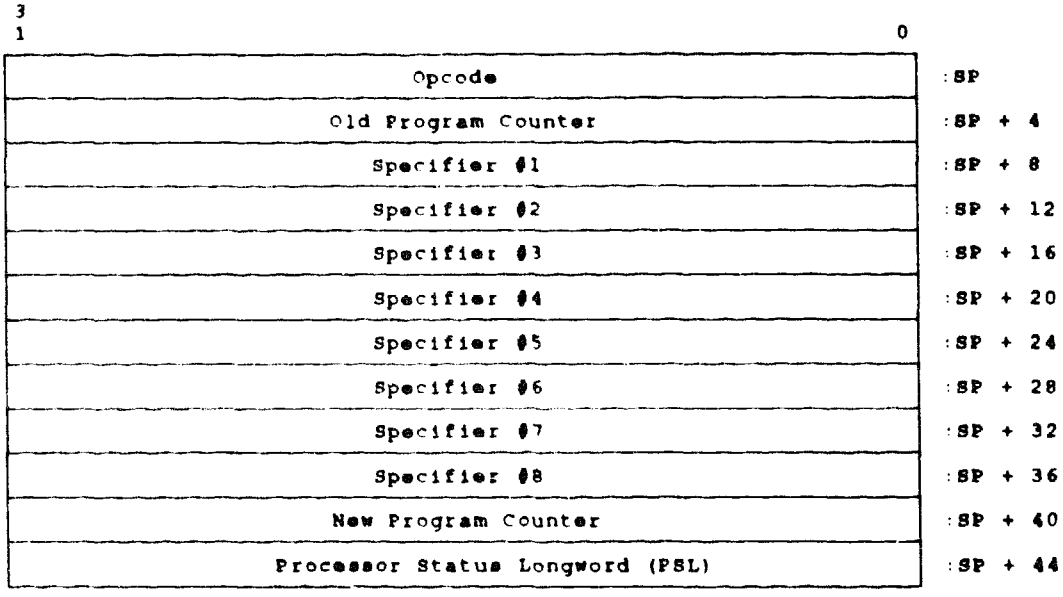
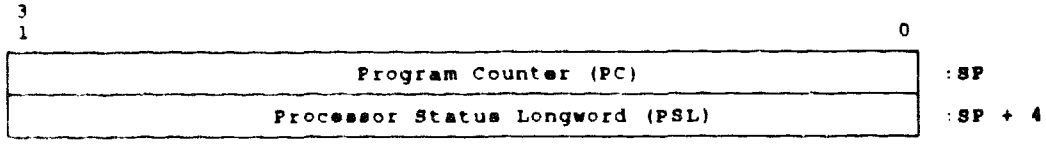


Table 3–5 Emulated Instruction Trap Stack Frame Parameters

Parameter	Description
Opcode	Zero-extended opcode of the emulated instruction.
Old PC	Program counter of the opcode of the emulated instruction.
Specifiers	Address of the specified operand for specifiers of either access type write (.wx) or address (.ax). Operand value for specifiers of access type read (.rx). For read-type operands whose size is smaller than a longword, the remaining bits are UNPREDICTABLE. For those instructions that do not have eight specifiers, the remaining specifier longwords contain UNPREDICTABLE values.
New PC	Program counter of the instruction following the emulated instruction.
PSL	PSL saved at the time of the trap.

Figure 3–7 Emulated Instruction Fault



Vector Module Disable Fault

A vector module disable fault is initiated through SCB vector 68 (hex) if the CPU-chip issues a vector instruction to the optional FV64A vector processor and the vector module is disabled. There are no parameters for this exception other than the PC/PSL pair. The reason for the exception can be found in the appropriate vector module registers (see Section 4.9.4.3).

Machine Check Exceptions

A machine check exception is reported through SCB vector 04 (hex) when the CPU-chip detects an error condition. The frame pushed on the stack for a machine check indicates the type of error and provides internal state information that may help identify the cause of the error. The machine check stack frame is shown in Figure 3–8 and its parameters are described in Table 3–6. Table 3–7 lists and describes the machine check codes.

Software must acknowledge machine checks by writing a zero to IPR38, MCESR, as a second machine check causes an ERR_MCHK_MCHK console halt.

Figure 3–8 Machine Check Stack Frame

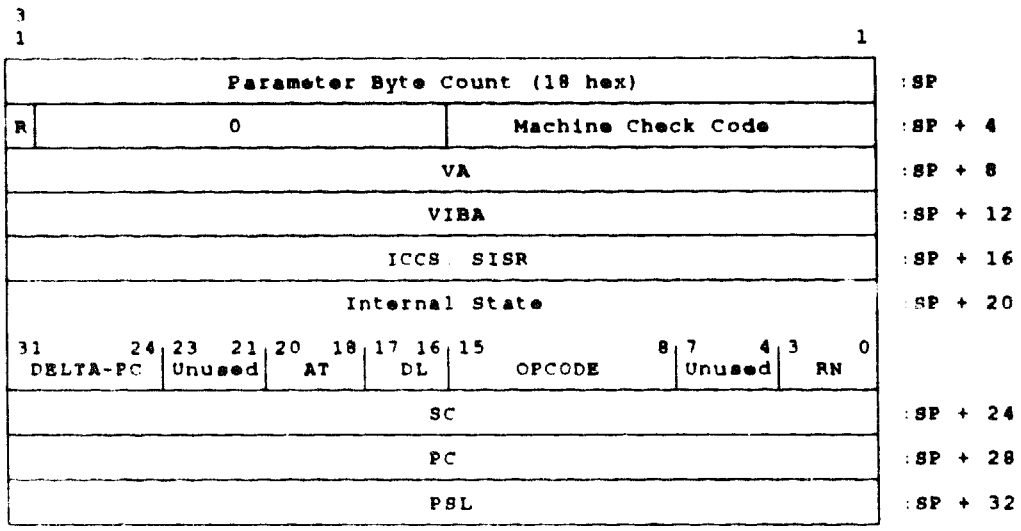


Table 3–6 Machine Check Stack Frame Parameters

Parameter	Description
Parameter Byte Count	The size of the stack frame in bytes, not including PSL, PC, and the byte count longword. It is always 18 (hex) bytes. Stack frame PC and PSL values are always referenced using this count as an offset from the stack pointer.
R (VAX Restart bit)	A flag from the hardware and microcode to the operating system to be used in the software equation to determine if the current macroinstruction is restartable after error cleanup. Other terms in the equation are PSL<27> (First Part Done, FPD), PCSTS<6> (Trap2), and RCSR<20> (Unlock Write Pending, UWP).
Machine check code (bits<15:0>)	Table 3–7 lists and describes the machine check codes.
VA	The virtual address being processed by the CPU at the time of the fault. VA is not necessarily relevant; the error handler checks the specific error address corresponding to the device or mechanism that signaled the error.
VIBA	The CPU prefetch virtual instruction buffer address at the time of the fault.
ICCS..SISR	The interrupt state information where bit<22> is ICCS<6> and bits<15:1> are SISR<15:1>.
Internal State	The internal state at the time of the fault. The internal state has the following layout:

Delta-PC, bits<31:24> — Difference between the values of the current incremented PC at the time that the machine check was detected and the PC of the instruction opcode. The exact interpretation of Delta-PC requires a detailed knowledge of the internal pipeline operation of the CPU-chip and is not used by software to make recovery decisions.

Unused, bits<23:21>

AT, bits<20:18> — The current setting of the E-box (the CPU-chip's execution unit or main data path) access-type latch, relating to the last (or upcoming) memory reference.

Value (binary)	Interpretation
000	Read
001	Write
010	Modify
011	Unassigned, CPU-chip error
100	Unassigned, CPU-chip error
101	Address
110	Variable bit
111	Branch

Table 3–6 (Cont.) Machine Check Stack Frame Parameters

Parameter	Description	
	DL, bits<17:16>	The current setting of the E-box data length latch, relating to the last (or forthcoming) memory reference.
		Value (binary) — Interpretation: 00 — Byte; 01 — Word; 10 — Long, F_Floating; 11 — Quad, D_Floating, G_Floating
	Opcode, bits<15:8>	The opcode (second opcode, if two-byte) of the instruction being processed at the time of the fault.
	Unused, bits<7:4>	
	RN, bits<3:0>	The value of the E-box RN register at the time of the fault, which may indicate the last GPR referenced by the E-box during specifier or instruction flows.
SC	Internal microcode-accessible register.	
PC, PSL	The program counter and processor status longword at the time of the fault.	

Table 3–7 Machine Check Codes

Code (hex)	Mnemonic	Description	Restart Condition
01	MCHK_FP_PROTOCOL_ERROR	Protocol error during F-chip operand/result transfer.	(R=1).(FPD=0).(UWP=0)
02	MCHK_FP_ILLEGAL_OPCODE	Illegal opcode detected by F-chip.	(R=1).(FPD=0).(UWP=0)
03	MCHK_FP_OPERAND_PARITY	Operand parity error detected by F-chip.	(R=1).(FPD=0).(UWP=0)
04	MCHK_FP_UNKNOWN_STATUS	Unknown status returned by F-chip.	(R=1).(FPD=0).(UWP=0)
05	MCHK_FP_RESULTS_PARITY	Returned F-chip result parity error.	(R=1).(FPD=0).(UWP=0)
08	MCHK_TBM_ACV_TNV	Translation buffer miss status generated in ACV/TNV microflow.	((R=1)+(FPD=1)).(UWP=0)
09	MCHK_TBM_ACV_TNV	Translation buffer hit status generated in ACV/TNV microflow.	((R=1)+(FPD=1)).(UWP=0)
0A	MCHK_INT_TD_VALUE	Undefined INT.ID value during interrupt service	((R=1)+(FPD=1)).(UWP=0)
0B	MCHK_MOVC_STATUS	Undefined state bit combination in MOVCX	(FPD=1).(UWP=0)
0C	MCHK_UNKNOWN_IBOX_TRAP	Undefined trap code produced by the I-box (the CPU-chip's instruction fetch and decode unit)	(R=1)+(FPD=0).(UWP=0)
0D	MCHK_UNKNOWN_CS_ADDR	Undefined control store address reached	((R=1)+(FPD=1)).(UWP=0)
10	MCHK_BUSERR_READ_PCACHE	P-cache tag or data parity error during read	((R=1)+(FPD=1)).(WUP=0).(TR2=0)
11	MCHK_BUSERR_READ_DAL	DAL bus or data parity error during read	((R=1)+(FPD=1)).(WUP=0).(TR2=0)
12	MCHK_BUSERR_WRITE_DAL	DAL bus error on write or clear write buffer	None
13	MCHK_UNKNOWN_BUSERR_TRAP	Undefined bus error microtrap	None
14	MCHK_VECTOR_STATUS	Vector module error	None

Where:

R is the VAX restart bit in the machine check stack frame
FDP is PSL<27>, First Part Done
UWP is RCSR<20>, Unlock Write Pending
TR2 is PCSTS<6>, Trap2
. is the logical AND operation
+ is the logical OR operation

3.3.4.4 Console Halt

A console halt is a microcode-initiated hardware restart sequence. Control passes to the console program during the restart sequence. A console halt happens when the CPU-chip's microcode detects:

- An inconsistency in internal state
- An incorrectly terminated DAL transaction
- A kernel-mode HALT instruction
- An asserted XBER<NHALT>
- A system reset
- An asserted XBER<NRST>

The hardware restart sequence is as follows:

1 The CPU-chip's microcode saves the current CPU state.
   - The stack pointer is saved in the appropriate stack pointer IPR
     - — IPR0, Kernel Stack Pointer
     - — IPR1, Executive Stack Pointer
     - — IPR2, Supervisor Stack Pointer
     - — IPR3, User Stack Pointer
     - — IPR4, Interrupt Stack Pointer

2 The current PC is saved in IPR42, SAVPC.

3 The PSL, halt code, MAPEN<0>, and a validity bit are saved in IPR43, SAVPSL.
   - SAVPSL<31:16> and <7:0> are loaded from PSL<31:16> and <7:0>.
   - SAVPSL<15> is set to MAPEN<0>.
   - SAVPSL<14> is set to zero if the PSL is valid and is set to one if the PSL is not valid. If the halt is due to a system reset, SAVPSL<14> is undefined.
   - SAVPSL<13:8> is set to the console halt code. The console halt codes are listed in Table 3–8.

4 The CPU-chip's microcode then initializes the following CPU state to the values shown:

State	Initialized Value
SP	IPR4, Interrupt Stack Pointer
PSL	041F 0000 (hex), IPL=31
MAPEN	0
ACCS<31>	0
ICCS	0 after reset only (halt code = 3)
SISR	0 after reset only (halt code = 3)
ASTLVL	4 after reset only (halt code = 3)
ACCS<1:0>	0 after reset only (halt code = 3)
All else	Undefined

5 Control passes to the console program code at 2004 0000 (hex).

Table 3–8 Halt Codes

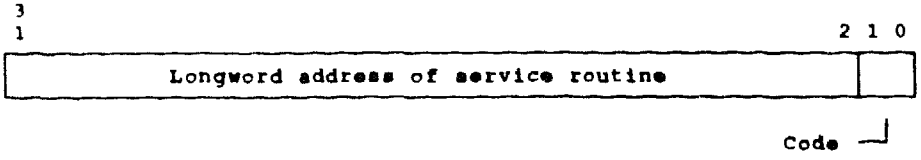
Code (hex)	Mnemonic	Description
02	ERR_HLTPIN	CTRL/P, break, or external halt
03	ERR_PWRUP	Initial power-up
04	ERR_INTSTK	Interrupt stack not valid during exception processing
05	ERR_DOUBLE	Machine check during exception processing
06	ERR_HLTINS	HALT instruction executed in kernel mode
07	ERR_ILLVEC	SCB vector bits<1:0> = 11
08	ERR_WCSVEC	SCB vector bits<1:0> = 10
0A	ERR_CHMFI	CHM*x* instruction executed while on interrupt stack
10	ERR_MCHK_ACV_TNV	ACV/TNV during machine check processing
11	ERR_KCHK_ACV_TNV	ACV/TNV during kernel-stack-not-valid
12	ERR_MCHK_MCHK	Machine check during machine check processing
13	ERR_KSNV_MCHK	Machine check during kernel-stack-not-valid processing
19	ERR_IE_PSL26_24_101	PSL<26:24> = 101 during interrupt or exception
1A	ERR_IE_PSL26_24_110	PSL<26:24> = 110 during interrupt or exception
1B	ERR_IE_PSL26_24_111	PSL<26:24> = 111 during interrupt or exception
1D	ERR_REI_PSL26_24_101	PSL<26:24> = 101 during REI
1E	ERR_REI_PSL26_24_110	PSL<26:24> = 110 during REI
1F	ERR_REI_PSL26_24_111	PSL<26:24> = 111 during REI
3F	ERR_SELFTEST_FAILED	Microcoded self-test failed in the CPU-chip. This can only happen during power-up.

3.3.5 System Control Block

The system control block (SCB) is a number of physically contiguous pages. The first page contains vectors for traps, faults, and software interrupts. Subsequent pages contain vectors for device interrupts. IPR17, the System Control Block Base Register, points to the SCB.

An SCB vector is an aligned longword in the SCB. The CPU-chip's microcode dispatches interrupts and exceptions through the SCB vector, shown in Figure 3–9.

Figure 3–9 System Control Block Vectors



Bits <31:2> of each vector supply the address of the operating system service routine for the interrupt or exception. The routine is longword aligned, as the microcode forces the lower two bits of the address to 00 (hex).

Bits <1:0> of each vector are a code:

00 – The event is to be serviced on the kernel stack unless the CPU is already on the interrupt stack. If the CPU is already on the interrupt stack, the event is to be serviced on the interrupt stack.

01 – The event is to be serviced on the interrupt stack. If the event is an exception, the IPL is raised to 1F (hex).

10 – ERR_WCSVEC, console halt code of 08, which causes a console entry.

11 – ERR_ILLVEC, console halt code of 07, which causes a console entry.

Table 3–9 shows the SCB layout.

Table 3–9 System Control Block Layout

Vector (hex)	Name	Type	Number of Parameters	Notes
00	Passive release	Interrupt	None	IPL is raised to requested IPL
04	Machine check	Abort	6	Parameters reflect machine state
08	Kernel stack not valid	Abort	None	Must be serviced on the interrupt stack
0C	Power fail	Interrupt	None	IPL is raised to 1E (hex)
10	Reserved/privileged instruction	Fault	None	
14	Customer reserved instruction	Fault	None	XFC instruction
18	Reserved operand	Fault/abort	None	Not always recoverable
1C	Reserved addressing mode	Fault	None	
20	Access control violation/vector alignment fault	Fault	2	Parameters are virtual address, status code
24	Translation not valid	Fault	2	Parameters are virtual address, status, code
28	Trace pending (TP)	Fault	0	
2C	Breakpoint instruction	Fault	0	
30	Unused	–	–	
34	Arithmetic	Trap/fault	1	Parameter is type code
38 – 3C	Unused	–	–	
40	CHMK	Trap	1	Parameter is sign-extended operand word
44	CHME	Trap	1	Parameter is sign-extended operand word
48	CHMS	Trap	1	Parameter is sign-extended operand word
4C	CHMU	Trap	1	Parameter is sign-extended operand word
50	Unused	–	–	
54	Soft error notification	Interrupt	None	IPL is 1A (hex)
58 – 5C	Unused	–	–	
60	Hard error notification	Interrupt	None	IPL is 1D (hex)
64	Unused	–	–	
68	Vector module disabled	Fault	None	Vector instructions
6C – 7C	Unused	–	–	
80	Interprocessor interrupt	Interrupt	None	IPL is 16 (hex)
84	Software level 1	Interrupt	None	
88	Software level 2	Interrupt	None	Usually used for AST delivery
8C	Software level 2	Interrupt	None	Usually used for process scheduling
90 – BC	Software levels 4 through 15	Interrupt	None	

Table 3–9 (Cont.) System Control Block Layout

Vector (hex)	Name	Type	Number of Parameters	Notes
C0	Interval timer	Interrupt	None	IPL is 16 (hex)
C4	Unused	–	–	
C8	Emulation start	Fault	10	Same mode exception, FPD=0; parameters are opcode, PC, specifiers
CC	Emulation continue	Fault	None	Same mode exception, FPD=1; no parameters
D0 – F4	Unused	–	–	
F8	Console receiver	Interrupt	None	IPL is 15 (hex)
FC	Console transmitter	Interrupt	None	IPL is 15 (hex)
100 – FFFC	Device vectors	Interrupt	None	Device interrupt vectors

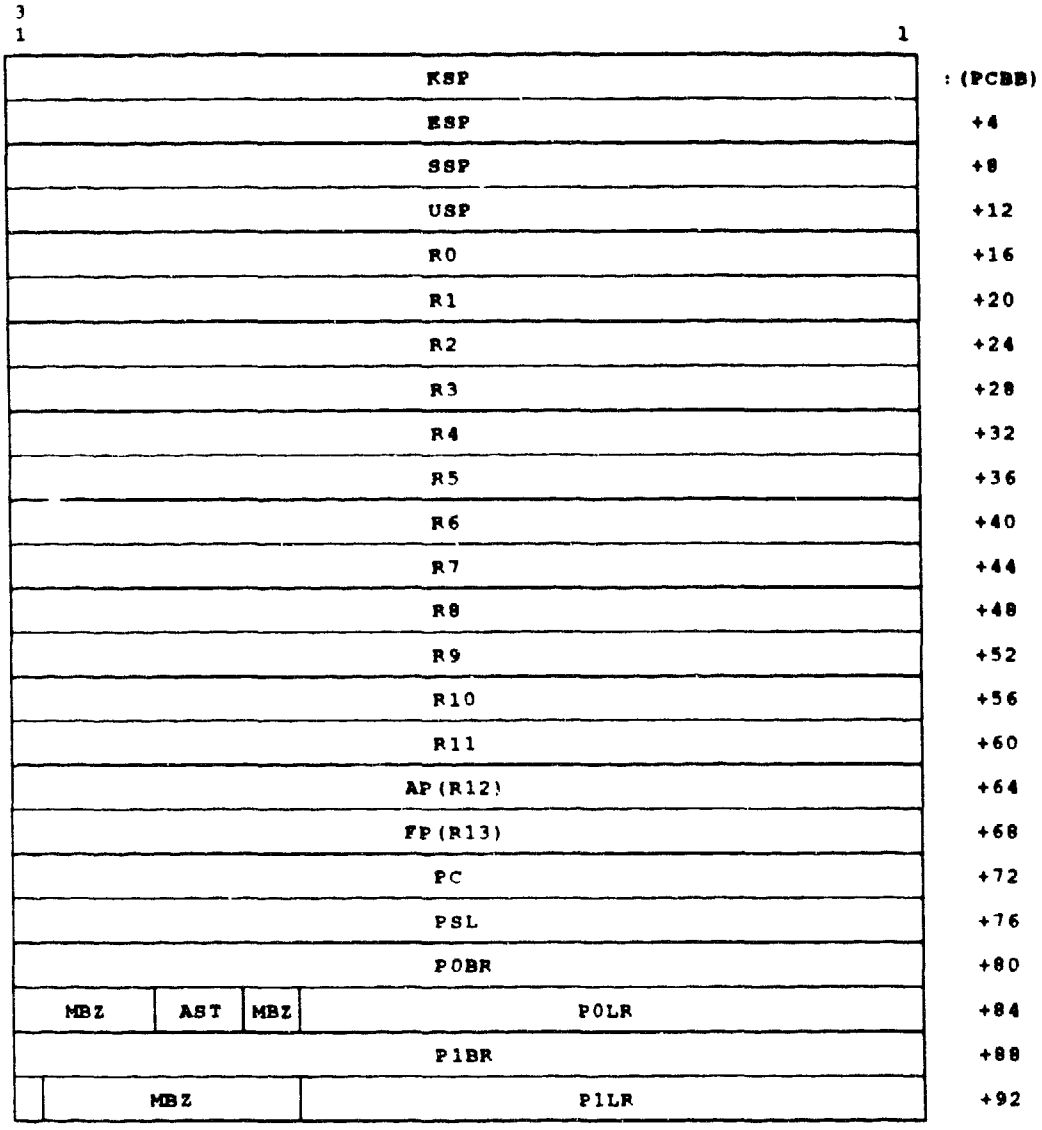
3.3.6 Process Structure

A process is a single thread of execution. The context of the current process is contained in the process control block (PCB).

The physical address of the current PCB is changed by writing to the Process Control Block Register (PCBB, IPR16). The LDPCTX instruction loads a process context from the PCB as described in the *VAX Architecture Reference Manual*, except for the PME (bit<31>) of PCB + 92, shown in Figure 3–10. PME is ignored. LDPCTX flushes only the process-space entries from the translation buffer; system-space entries are preserved.

Other process structure functions are implemented as described in the *VAX Architecture Reference Manual*.

Figure 3–10 Process Control Block



3.3.7 Floating-Point Accelerator

The KA64A CPU module includes the F-chip floating-point accelerator that enhances the performance of floating-point and certain integer calculations.

The following VAX instructions are processed by the F-chip:

- F_floating add, subtract, multiply, divide, convert, move, compare, negate, and test instructions. ACBF, EMODF, and POLYF are not processed by the F-chip.
- D_floating add, subtract, multiply, divide, convert, move, compare, negate, and test instructions. ACBD, EMODD, and POLYD are not processed by the F-chip.
- G_floating add, subtract, multiply, divide, convert, move, compare, negate, and test instructions. ACBG, EMODG, and POLYG are not processed by the F-chip.
- Longword-length integer multiply instructions.

The F-chip is enabled/disabled by using the F-Chip Present bit (Accelerator Control and Status Register, ACCS<1>). When the F-chip is disabled, the execution of a floating-point instruction results in a reserved instruction exception. The execution of a longword-length integer multiply instruction is emulated by the CPU-chip's microcode.

The F-chip supports the following data types:

- F_floating
- D_floating
- G_floating

The F-chip supports the following conversions:

- Byte conversion to/from floating formats
- Word conversion to/from floating formats
- Longword conversion to/from floating formats and multiply

The CPU-chip parses the opcode and instruction specifiers and sends opcode and operands to the F-chip. The CPU-chip explicitly transfers operands from the GPRs, the instruction stream, and the primary cache to the F-chip. Floating-point short literals are transferred in unexpanded form as the F-chip expands them to the correct format. Operands from the backup cache or from memory are returned to both the CPU-chip and the F-chip simultaneously.

When the F-chip receives the last operand for an instruction, it begins the computation of the result. In parallel, the CPU-chip completes any instruction setup, such as parsing a destination specifier. The CPU-chip next requests the result from the F-chip and stalls until the result is returned. The CPU-chip then stores the result where the destination specifier indicates, either in a GPR or memory, and sets the PSL condition codes.

The F-chip cannot fetch operands from I/O space. I/O space operands cause a reserved operand fault.

The F-chip tests for exception conditions and reports any to the CPU-chip when the result is requested. Detected exceptions include reserved operands, floating divide by zero, floating overflow, floating underflow, and data parity errors.

The CPU-chip's microcode disables the F-chip as part of the power-up initialization process. The execution of any floating-point instruction results in a reserved instruction exception until the F-chip is enabled. The console program enables the F-chip by setting ACCS<1> and then tests the operation of the F-chip. If the F-chip fails these tests, the console program clears ACCS<1>, leaving the F-chip disabled.

CAUTION: The F-chip has a problem that affects operating system exception handlers but not interrupt handlers or user software. The exception only happens under the following conditions:

1. **The F-chip receives a DIVF3, DIVD3, or DIVG3 opcode and the first two operands. It then starts calculating.**
2. **The CPU-chip, while parsing the third operand of the DIV instruction, incurs an exception.**
3. **One of the first few instructions of the exception handler is an F-chip instruction, such as MULL.**
4. **The SCB vector, the first few instructions of the exception handler, and the appropriate page table entries are all in the primary cache or the secondary cache.**
5. **The CPU-chip fetches, decodes, and sends an opcode to the F-chip on exactly the 10th, 20th, or 21st cycle for F, D, and G.**

The results of the MULL are corrupted when the problem occurs, but are otherwise reliable.

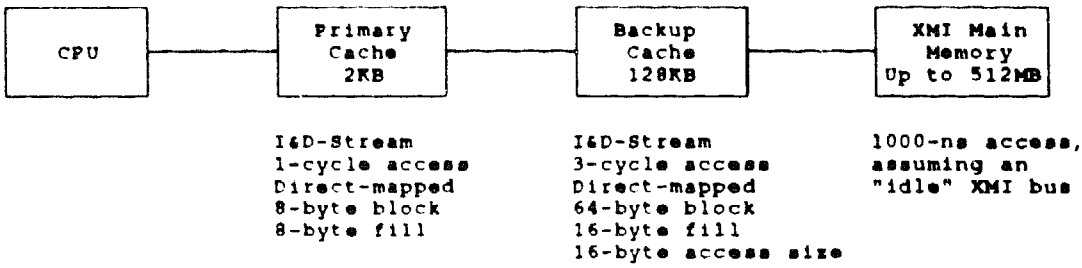
To avoid this problem, the operating system exception handler must not have any F-chip instructions within the first five instructions.

3.4 Cache Memory

The KA64A CPU module has a two-level cache to maximize CPU performance. The first-level cache is a 2-Kbyte primary cache (P-cache) contained in the CPU-chip. The second-level cache is a 128-Kbyte backup cache (B-cache) consisting of the C-chip, for cache control logic, and 24 16-K x 4 static RAMs, for data storage.

The C-chip contains the tag store and the control logic for the B-cache. Figure 3–11 shows the memory hierarchy.

Figure 3–11 CPU Module Cache Memory



The P-cache is direct-mapped, with a quadword fill and allocate (block) size. It is read-allocate, no-write-allocate, and write-through. The P-cache tag store contains one tag and one valid bit for each P-cache block, totaling 256 tags mapping 256 quadwords. Each tag entry includes an 18-bit tag, one valid bit, and one parity bit. Each data block contains eight data bytes, each byte having one parity bit.

The 128-Kbyte B-cache is direct mapped, with an octaword fill and a 4-octaword allocate (block) size. The B-cache is read allocate, no-write-allocate, and write-through. The C-chip implements the B-cache tag store and provides control for the cache RAMs, which contain data and data-parity. The backup cache tag store contains 2048 tag entries and each tag entry contains a 12-bit tag, four valid bits, and a parity bit. The data RAMs are organized into 16-K locations of eight data bytes, with a parity bit for each byte.

3.4.1 Cache Coherency

Data cached by a CPU must remain coherent with data in main memory. Therefore, any main memory writes done by a processor or an I/O device must invalidate data cached by all processors in the system. The REXMI interface invalidates cached data corresponding to any memory write that it sees on the XMI bus.

The C-chip provides the mechanism for the REXMI interface to determine if data is being cached. The C-chip maintains a duplicate copy of the P-cache tag store that it can access in parallel with the B-cache tag store. There are two ports that check write addesses against the addresses in both the P-cache and the B-cache.

The DAL is the first port. The Inval-Bus and associated control signals provide the second port. When the REXMI interface sees a main memory write on the XMI bus, it presents the write address on the Inval-Bus and makes an Inval-Bus request to the C-chip. The C-chip uses otherwise idle cycles to look up the address in both tag stores to determine if any data is being cached for the address.

If the C-chip finds data cached for the supplied address, the REXMI requests the DAL and issues an invalidate transaction to force each cache to invalidate data for the address. If the C-chip finds no cached data at the write address, the REXMI ignores the memory write. This filtering mechanism for invalidates avoids using the DAL except when a write address is found to be cached, improving overall system performance.

3.4.2 Cache Performance with Multiprocessing

The assignment of processes to particular processors has a significant impact on the overall system throughput. When a process is scheduled to run on a processor other than the one it last ran on, the cache hit rate decreases because the data that was in the original processor's cache is not necessarily in the new processor's cache. This slows the execution of the process as memory access to get the data is much slower than cache access. Memory traffic also increases.

Invalidate traffic seen by the original processor increases as memory writes done on the new processor invalidate any cached data on the original processor.

Software should minimize the migration of processes between processors.

3.4.3 Primary Cache

The primary cache (P-cache) resides in the CPU-chip. The P-cache is arranged in 64 rows with four data quadwords and four tag entries per row, as shown in Figure 3–12. Each physical address is logically subdivided as shown in Figure 3–13. The organization of each tag entry is shown in Figure 3–14. The data array organization of each quadword is shown in Figure 3–15.

Figure 3–12 Primary Cache Organization

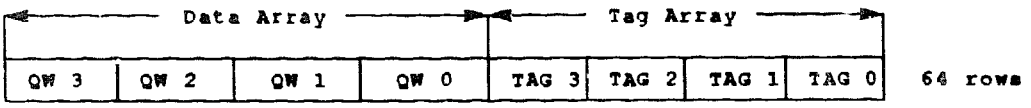
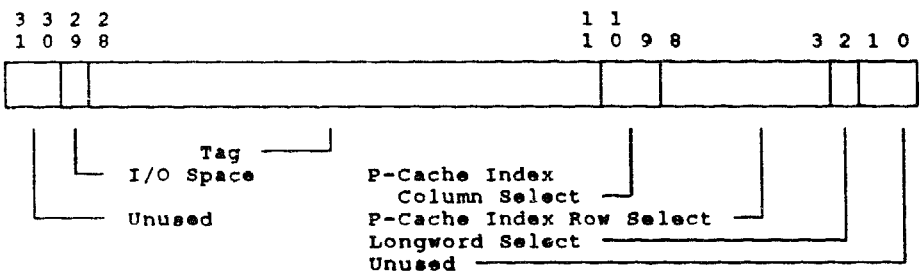
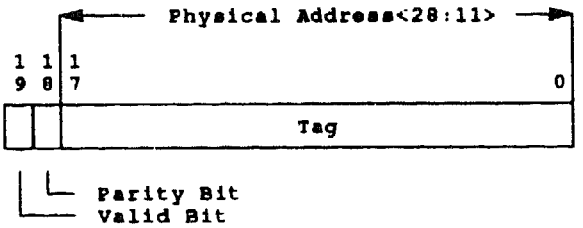


Figure 3–13 Primary Cache Physical Address



Bits <31:30>	Unused.	
Bits <29:2>	The physical addresses supplied to the P-cache.	
	Bit <29>	Specifies I/O space. I/O space addresses are not cached.
	Bits <28:11>	Tag
	Bits <10:9>	Selects one of four columns of tags.
	Bits <8:3>	Selects one of the rows of the P-cache memory.
	Bit <2>	Selects a longword out of the quadwords of the P-cache.
Bits <1:0>	Unused.	

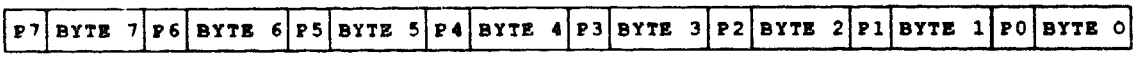
Figure 3–14 Tag Entry Organization



The tag consists of:

Bit <19>	The valid bit, used to indicate that the corresponding entry in the P-cache is valid. The valid bit is not included in the tag parity calculation.
Bit <18>	The tag parity, which is parity computed by the P-cache over the 18 tag bits.
Bits <17:0>	The tag, bits <28:11> of the physical address.

Figure 3–15 Quadword Data Array Organization of the Primary Cache



Each P-cache entry consists of one quadword. Parity information is maintained separately for each byte. The P-cache is either flushed or switched off by the resulting microtrap routine if a parity error is detected on the data coming from the P-cache.

Four registers are used to control the P-cache, store status, test, and recover from errors:

- Primary Cache Tag Array Register, PCTAG (IPR124)
- Primary Cache Index Register, PCIDX (IPR125)
- Primary Cache Error Address Register, PCERR (IPR126)
- Primary Cache Status Register, PCSTS (IPR127)

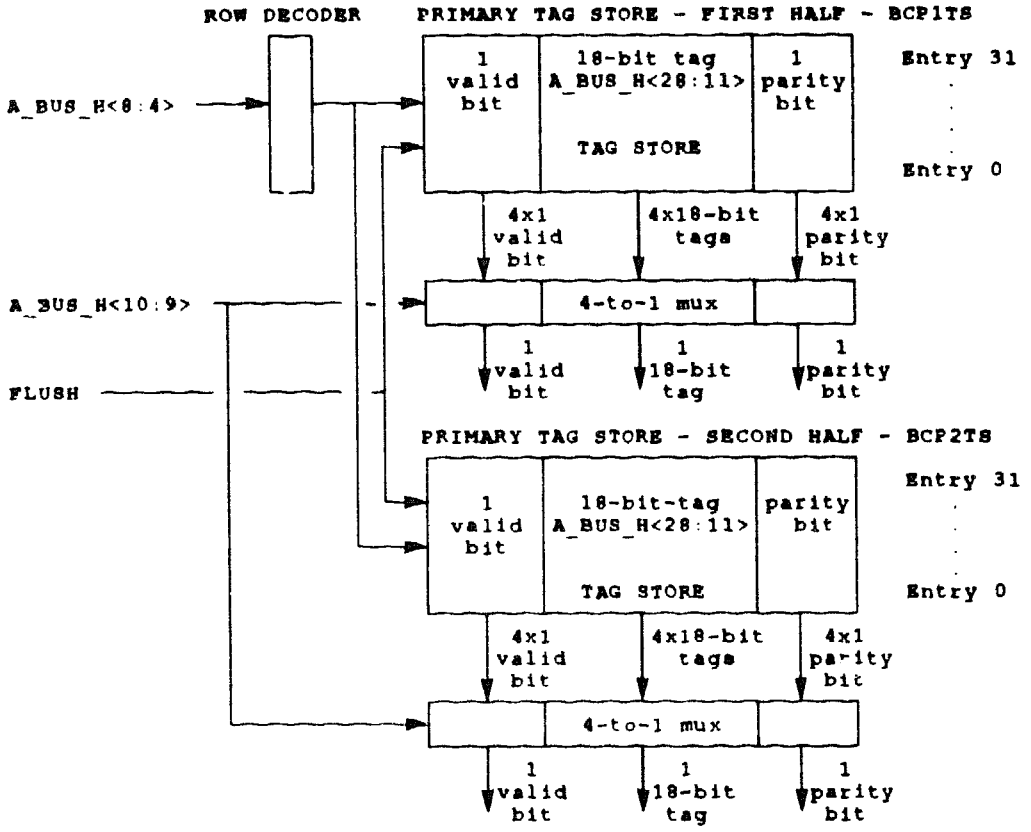
An IPR read can be accomplished in two ways, by software using a Move From Processor Register (MFPR) instruction, or by the console operator using the EXAMINE/I console command.

An IPR write can also be accomplished in two ways, by software using a Move To Processor Register (MTPR) instruction, or by the console operator using the DEPOSIT/I console command.

3.4.3.1 Duplicate Primary Cache Tag Store Block Diagram Description

The C-chip maintains a duplicate copy of the P-cache tag store to efficiently process cache invalidates. Figure 3–16 shows the duplicate P-cache tag store block diagram.

Figure 3–16 Duplicate Primary Cache Tag Store Block Diagram

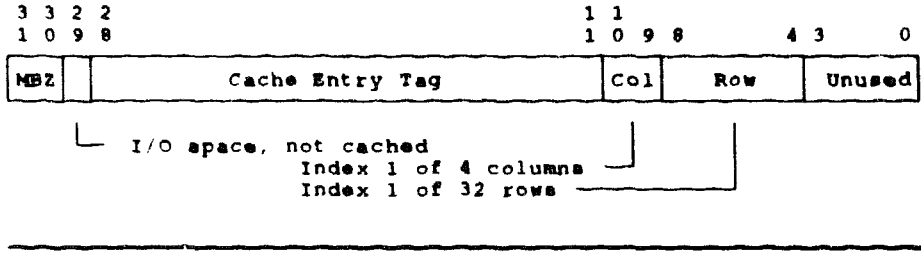


The duplicate P-cache tag store contains one tag and one valid bit for each quadword block in the P cache. With 256 quadword blocks in the P-cache, the P-cache tag store contains 256 entries of 20 bits each. Each entry consists of an 18-bit tag (bits<28:11> of the physical address), one valid bit, and one parity bit. The C-chip copy of the P-cache tag store is organized in two banks, each of which has 32 rows and four columns.

Figure 3–17 shows the VAX physical address while it is used in P-cache tag store addressing during IPR operations. The P-cache tag store row is indexed using bits<8:4> of the address. The P-cache tag store column is indexed using bits<10:9> of the address. On a P-cache tag store access, both halves of the P-cache tag store are accessed. A hit is calculated separately in each half.

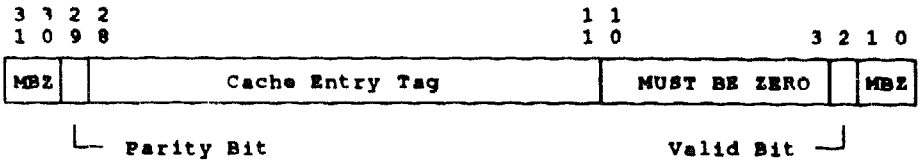
The P-cache tag store tag, valid bit, and parity are written explicitly using an IPR write of the tag store (either an MTPR instruction or the console command DEPOSIT/I), or read explicitly using an IPR read (either an MFPR of the tag store or the console command EXAMINE/I).

Figure 3–17 Primary Cache Tag Store Addressing Using Physical Address



The C-chip responds to an IPR read of the P-cache tag store register by driving the D_BUS as shown in Figure 3–18. The P-cache tag store row and column index fields of BCIDX, bits<10:4>, are used as the index to the tag array. BCIDX must have been previously written, using an IPR write, to ensure predictable results from the IPR read of the P-cache tag store.

Figure 3–18 D_BUS Format to Access the Primary Cache Tag Store



On an IPR write of the P-cache tag store, the C-chip writes the contents of the D_BUS onto the P-cache tag store. The P-cache tag store row and column index fields of BCIDX are used as the index to the tag array. BCIDX must have been previously written using an IPR write to ensure predictable results for the IPR write of the P-cache tag store.

The valid bit is the valid bit of the P-cache tag store entry. The P-cache entry tag is the tag portion of the P-cache tag store entry. The parity bit corresponds to odd parity as calculated on the tag.

3.4.3.2 Maintaining Primary Cache Consistency

To maintain P-cache consistency, it is important to use the proper sequence of events for reenabling a tag store that has been disabled. Any state change to the CPU's P-cache must be reflected in the C-chip copy of the primary cache tag store. During normal operation, changes are updated automatically by the C-chip when a cacheable read occurs.

While the backup cache tag store is disabled, its contents may change due to DAL operations. The backup cache tag and data stores are updated for all cache fills that occur while the backup tag store is disabled. The backup cache tag and data stores do not get updated for writes and invalidates that occur while the backup tag store is disabled.

To prevent stale data getting into the backup cache while it is disabled, follow the following procedure:

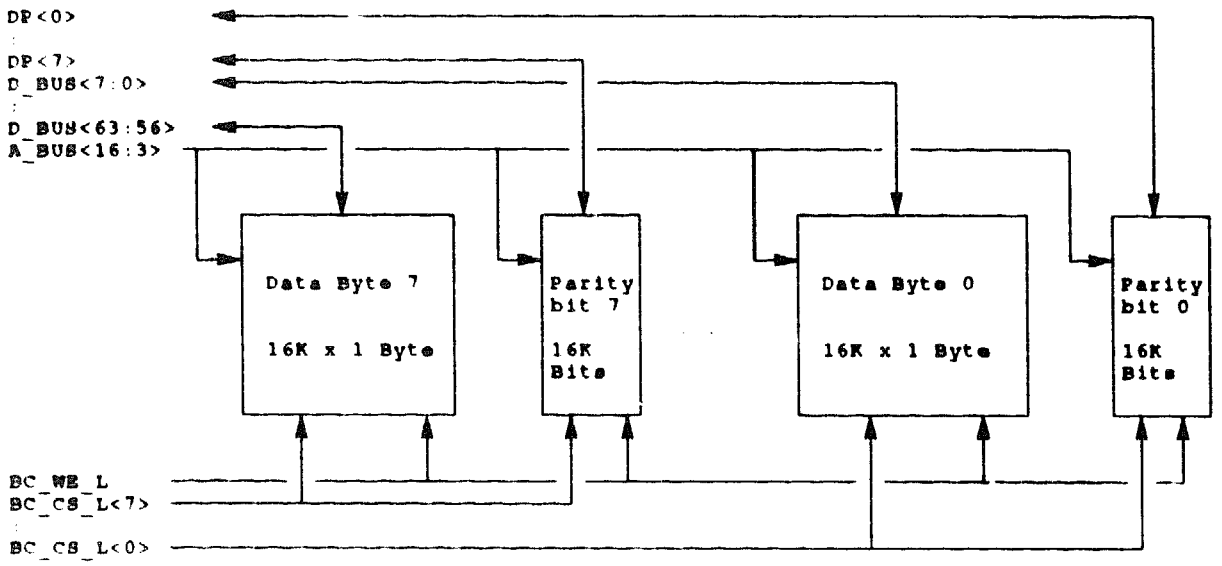
1. Ensure that the IPL is set to 31 to prevent interrupts from occurring between instructions.
2. Flush the backup cache tag store using an IPR write.
3. Immediately enable the backup cache without intervening instructions.
4. Again flush the backup cache tag store without intervening instructions.

Software that modifies the instruction stream, such as debuggers, must not allow any writes to occur during the instruction sequence flush-enable-flush of the backup cache. Never insert breakpoints anywhere in the sequence.

3.4.4 Backup Cache

The backup c che (B-cache) is a 128-Kbyte cache, direct mapped, quadword ss size, with an octaword fill (subblock) size and a four-octaword allocate (block) size. The backup cache is also read-allocate, no-write-allocate, and write-through. The backup cache is shown in Figure 3–19.

Figure 3–19 Backup Cache Organization



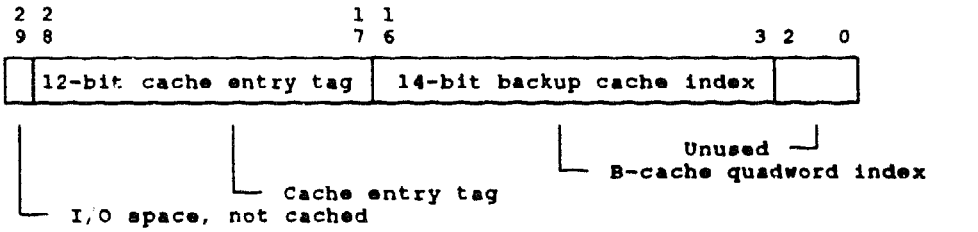
The data and control signals for the B-cache RAMs are:

- D_BUS<63:0>. Eight bytes of data.
- DP<7:0>. Eight bits of data parity. One bit for each data byte.
- A_BUS<16:3>. Address for the cache RAMs.
- BC_WE_L. Write enable. The C-chip drives write enable, which is normally deasserted. It asserts for memory read misses, cache fills, and writes.
- BC_CS_L<7:0> Chip select. One line for each byte of data and associated parity bit. The C-chip drives these signals. During a cache read, BC_CS_L<7:0> asserts and BC_WE_L deasserts, so all cache RAMs drive the D_BUS. During a read miss and cache fill, BC_CS_L<7:0> and BC_WE_L assert, allowing every byte to be writen with returned data. During a memory write transaction, the CPU-chip drives BM_L<7:0> as a byte mask for the data. The C-chip asserts the corresponding BC_CS_L<7:0> lines so that only the intended bytes are written.

3.4.4.1 Backup Cache RAM Addressing

The VAX physical address, as used by the B-cache RAMs, is shown in Figure 3–20. The B-cache index is the portion of the address used to address the RAMs, while the remaining high-order bits, except the I/O bit, are used as the tag for the entry.

Figure 3–20 Backup Cache RAM Addressing

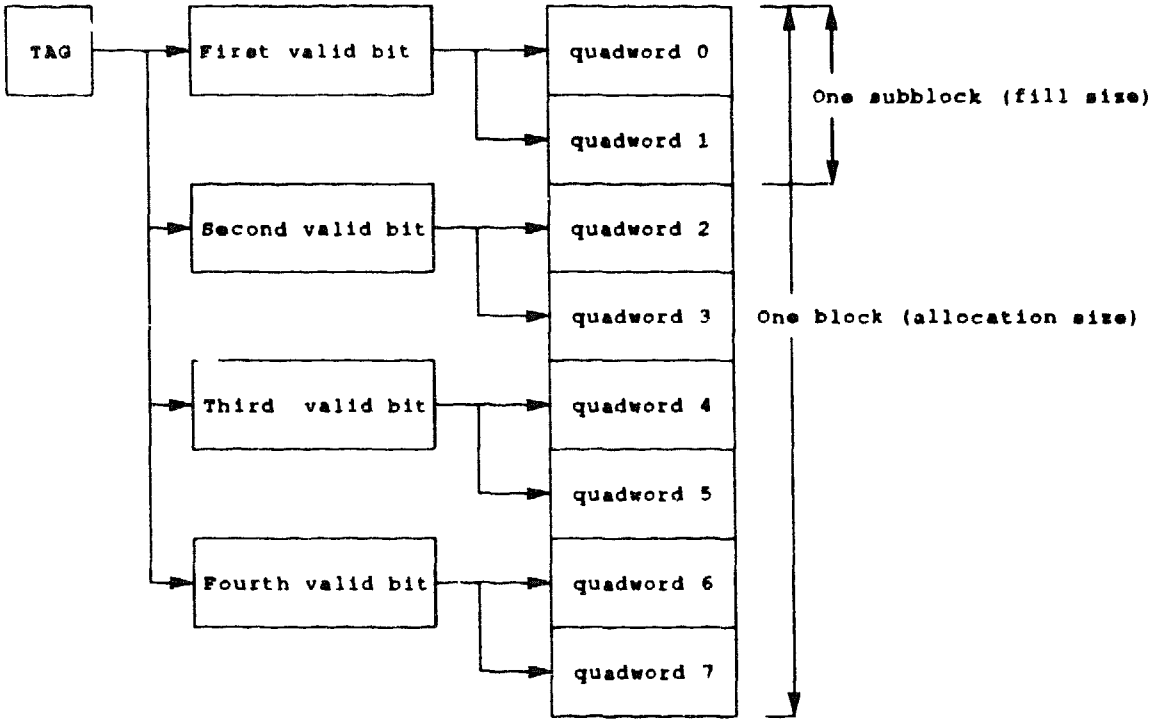


3.4.4.2 Backup Cache Tag Store Organization

The tag store is organized with one tag and four valid bits corresponding to each four-octaword block of the cache. Each valid bit corresponds to one octaword subblock. When a cache tag miss occurs on a read, a block is allocated, a subblock is filled, and the corresponding valid bit is set. When a cache tag compare is successful but the valid bit is not set, a subblock is filled from memory, and the corresponding valid bit is set.

Figure 3–21 shows the backup cache tag store organization.

Figure 3–21 Backup Cache Tag Store Organization



3.4.4.3 Backup Cache Internal Processor Registers

Several C-chip registers are accessed using IPR reads and writes.

An IPR read can be accomplished in two ways, by software using a Move From Processor Register (MFPR) instruction, or by the console operator using the EXAMINE/I console command.

An IPR write can also be accomplished in two ways, by software using a Move To Processor Register (MTPR) instruction, or by the console operator using the DEPOSIT/I console command.

During the IPR access, A_BUS_H<10:3> tells which IPR is being addressed. The C-chip internal processor registers are:

- Backup Cache Backup Tag Store, BCBTS (IPR113)
- Backup Cache Primary Cache Tag Store, First Half, BCP1TS (IPR114)
- Backup Cache Primary Cache Tag Store, Second Half, BCP2TS (IPR115)
- Backup Cache Refresh Register, BCRFR (IPR116)
- Backup Cache Index Register, BCIDX (IPR117)
- Backup Cache Status Register, BCSTS (IPR118)
- Backup Cache Control Register, BCCTL (IPR119)
- Backup Cache Error Address Register, BCERR (IPR120)
- Backup Cache Flush Backup Cache Tag Store Register, BCFBTS (IPR121)
- Backup Cache Flush Primary Cache Tag Store Register, BCFPTS (IPR122)

3.4.4.4 Backup Cache Tag Store Block Diagram Description

Figure 3–22 is the backup cache tag store block diagram. Figure 3–23 shows the backup cache tag store addressing using a physical address.

Figure 3–22 Backup Cache Tag Store Block Diagram

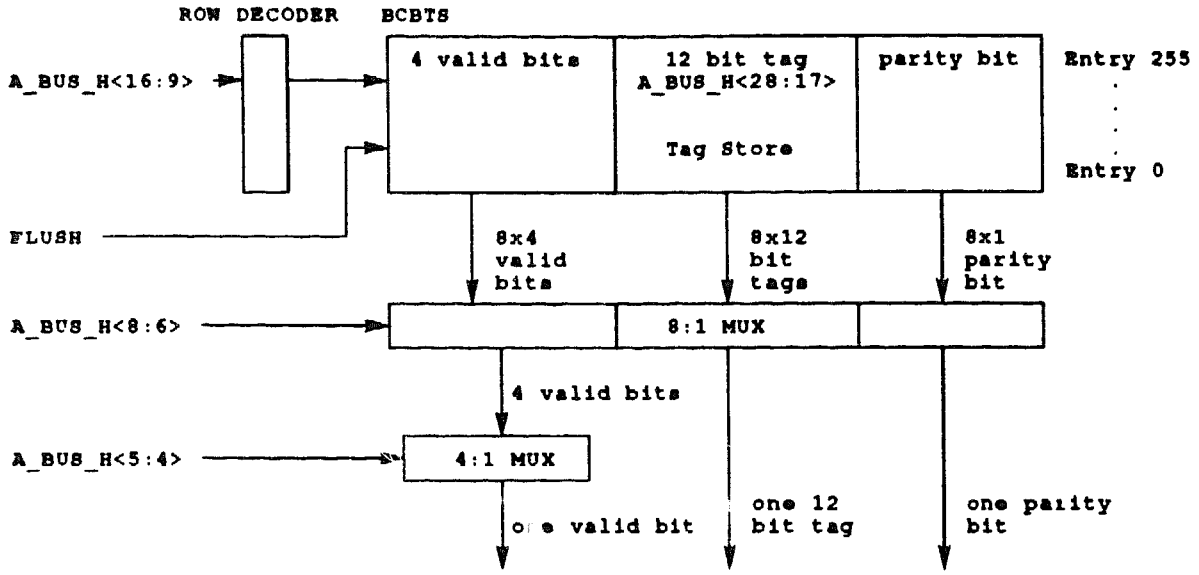
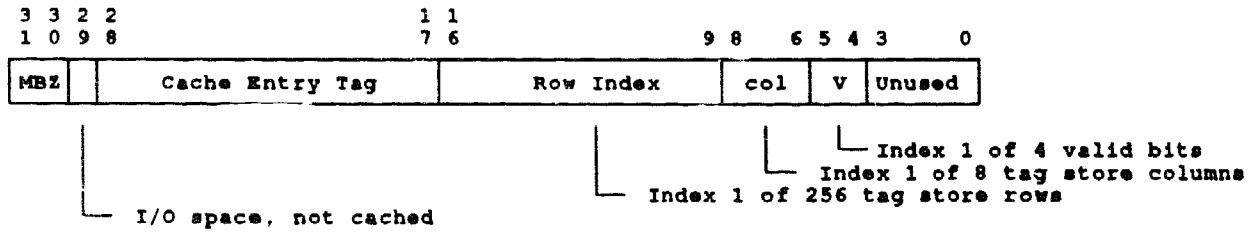


Figure 3–23 Backup Cache Tag Store Addressing Using Physical Address

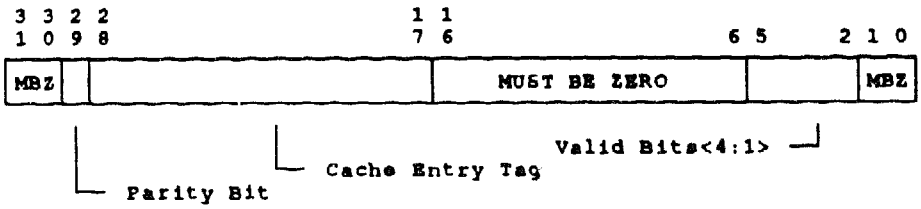


A tag store entry (tag, valid bits, and parity bit) can be written explicitly by using an IPR write of the tag store or read using an IPR read of the tag store. The location of the entry that is accessed by the IPR read is contained in the index register, BCIDX.

The C-chip responds to an IPR read of the backup cache tag store register by driving the D_BUS as shown in Figure 3–24. The backup cache tag store row and column index fields of BCIDX, bits<16:6>, are used as the index to the tag array. BCIDX must have been previously written, using an IPR write, to ensure predictable results from the IPR read of the tag store.

On an IPR write of the tag store register, the C-chip writes the contents of the D_BUS into the tag store. The backup cache tag store row and column index fields of BCIDX are used as the index to the tag array. BCIDX must have been previously written, using an IPR write, to ensure predictable results for the IPR write of the tag store.

Figure 3–24 D_BUS Format to Access BCBTS



The four valid bits of the tag store entry (BCBTS<5:2>), the valid bit selected by physical address bits<5:4>, and the subblock number in the tag store correspond as follows:

BCBTS	A_BUS<5:4>	Subblock Number
<5>	11	4
<4>	10	3
<3>	01	2
<2>	00	1

The cache entry tag contains the tag portion of the tag store entry, which is the high order physical address bits.

The parity bit is for odd parity, as calculated on the tag.

3.4.4.5 Using the Backup Cache Registers

The 10 C-chip registers control the backup cache tag store and the C-chip copy of the primary cache tag store. Operating system software reads these IPR registers with the Move To Processor Register (MTPR) and writes them with Move From Processor Register (MFPR) instructions, which must be executed in kernel mode. A console operator can do an IPR read using an EXAMINE/I command and can do an IPR write using a DEPOSIT/I command.

Control of the Backup Cache

Normal operational control (enable and disable) of the backup cache and the C-chip copy of the primary cache tag store is done with writes to BCCTL. The backup cache and the C-chip copy of the primary cache tag store are flushed during normal operation by writing a zero to BCFBTS and BCFPTS, respectively.

Backup Cache Initialization

The console program is responsible for the power-up initialization of the backup cache tag store and the C-chip copy of the primary cache tag store by writing each entry with an invalid tag with good parity. Each entry is written with a write to BCIDX, followed by a write to BCBTS, BCP1TS, or BCP2TS. As part of the backup cache initialization, cache refresh is enabled and the cache RAM speed is specified by writing to BCCTL. Both caches are left disabled.

Diagnostics

The backup cache data, the backup cache tag store, and the C-chip copy of the primary cache tag store are tested by reading and writing cache tags via BCIDX, BCBTS, BCP1TS, and BCP2TS. Cache refresh is tested by reading and writing BCRFR. Error detection is tested by constructing an error and then reading the state from BCSTS and BCERR.

Backup Cache Refresh Register (BCRFR), IPR116

The refresh register contains separate addresses to refresh the C-chip's copy of the P-cache tag store and the primary cache tag store. When BCCTL<3> (Enable Refresh) is set and a refresh is done, each refresh address field is incremented separately. The primary cache tag store is completely refreshed after 32 refresh microcycles, and the backup cache tag store is completely refreshed after 256 refresh microcycles.

When the Enable Refresh bit is not set in BCCTL the refresh addresses are only changed explicitly through an IPR write. The tag store rows are only refreshed when they are accessed explicitly through reads, writes, IPR reads, or IPR writes. BCRFR is used, instead of BCIDX, to access the backup cache copy of the primary cache tag store and the primary cache tag store during IPR operations on those registers when the Enable Refresh bit is not set.

The refresh register may be written using an IPR write, or read using IPR read. If Enable Refresh is set when the IPR operation is done, the result of the operation is unpredictable.

Backup Cache Status Register (BCSTS), IPR118

BCSTS is read using an IPR read. All bits are only writable by hardware except Status Lock (bit<0>), which is cleared using an IPR write.

If BCSTS<25> (I-BUS CYCLE) is set, BCSTS<24:21> (DAL CMD field) is unpredictable since a DAL command is not necessarily being processed during an Inval-Bus cycle. The three Hit fields, BCSTS<20>, <19>, and <18> (P2TS HIT, P1TS HIT, and BTS HIT) and BCSTS<4> (BUS ERR) show the results from the access of the backup and primary cache tag stores.

When BCSTS<24:21> (DAL CMD) is a memory read or write, the results of the backup cache tag store access are given by BCSTS<1> (BTS PERR), <18> (BTS HIT), and <17> (BTS COMPARE). The results of the primary cache tag store access are given by BCSTS<3> and <2> (P2TS HIT and P1TS HIT). The primary cache tag store error bits have no meaning and are zero because parity is not calculated on the contents of the primary cache tag store copy during these transactions.

BCSTS<23:21> (DAL CMD) is a DMA cache fill or memory write when the DAL command is DMG (DMA Grant). The results of the read of both tag stores are contained in the status bits. It is not possible to tell if DMG was asserted using the contents of BCSTS.

Table 3–10 shows the C-chip transactions and the bits loaded into BCSTS. *Pn*TS PERR refer to BCSTS<3> and <2>.

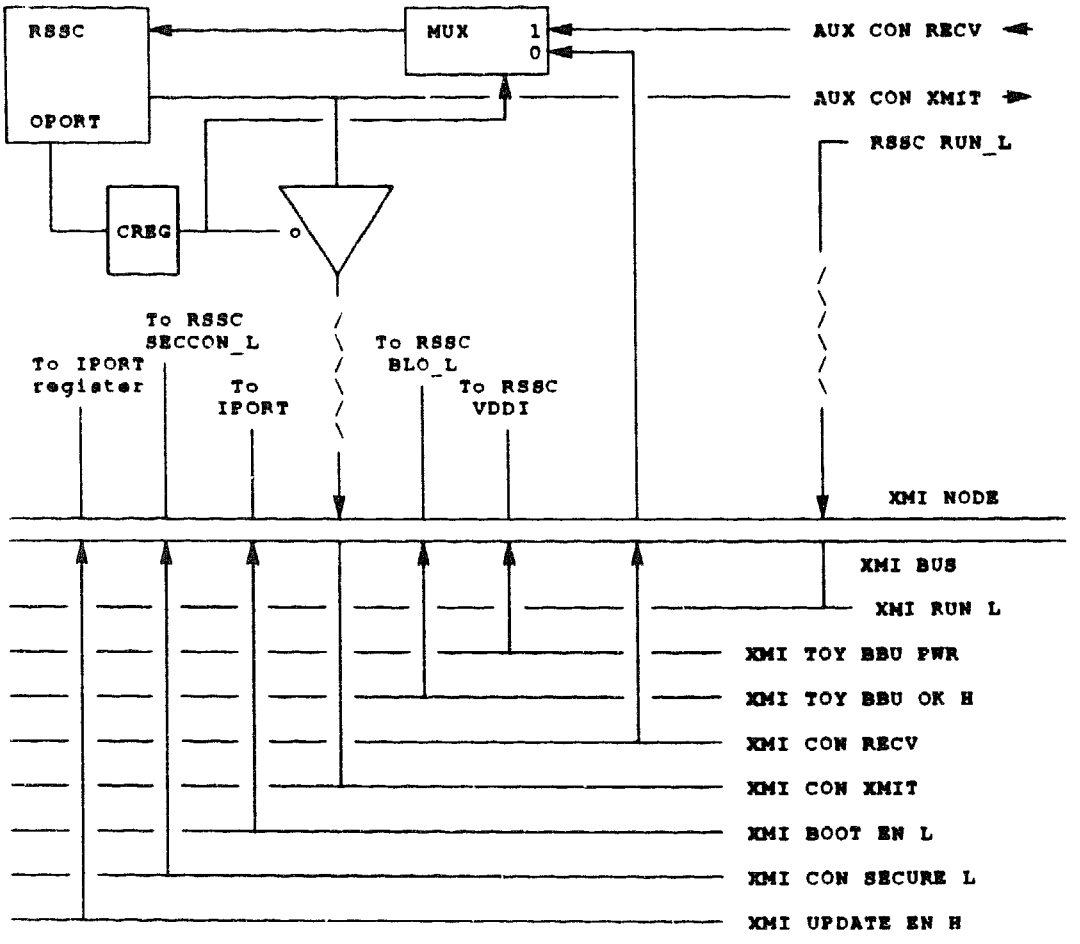
Table 3–10 Bits Loaded into BCSTS During C-Chip Transactions

	Loaded		
Cycle Type	BTS ERR	*Pn*TS PERR	All Other Bits
Read	Yes	No	Yes
Write	Yes	No	Yes
DMA Fill	Yes	Yes	Yes
DMA Write	Yes	Yes	Yes
Inval-Bus	Yes	Yes	Yes
IPR Read/write	No	No	No

3.5 System Support Chip

The system support chip (RSSC) provides functions to support the VAX 6000–400 system environment, including the VAX console. The VAX console is the combined hardware-software interface that controls the system at power-up and when in halt mode.

Figure 3–25 Control Panel Connections Including Console Lines



3.5.1 Console Serial Line

The console serial line provides a full-duplex, EIA RS-423 serial line interface that is also RS-232C compatible. The console serial line is shown in Figure 3–25. The only format supported is 8-bit data with no parity and one stop bit. Four internal processor registers (IPRs) control the operation of the console serial line; they are:

- Console Receiver Control/Status, RXCS (IPR32)
- Console Receiver Data Buffer, RXDB (IPR33)
- Console Transmitter Control/Status, TXCS (IPR34)
- Console Transmitter Data Buffer, TXDB (IPR35)

3.5.1.1 Console Serial Line Connections

The XMI bus provides bused console signals to each node in the system. This means that each processor node has equal access to the system console lines. During power-up initialization, all CPU modules examine each other's XBER<STF>, RCSR<BPD>, and node ID to determine which node will be the boot processor and drive the console transmit line, which is a tristate TTL line.

Only the boot processor communicates with the system console.

CREG0<TERM SEL> selects which console lines are attached to the RSSC's console transmit and receive registers.

3.5.1.2 CTRL/P Detection

The console serial line unit of the RSSC recognizes CTRL/P as a BREAK condition. When the RSSC detects a valid BREAK condition, RXDB<RCV BRK> is set. If halts are enabled (XMI HALT EN L asserted, a function of a control panel key switch) when RXDB<RCV BRK> sets, the CPU halts and transfers program control to ROM location 2004 0000 (hex). RXDB<RCV BRK> is cleared by reading RXDB.

3.5.1.3 Baud Rate Selection

The receive and transmit baud rates are always identical and are controlled by SSCCNR<TERM BAUD SEL>.

The console program attempts to set the console terminal baud rate to the value contained in an EEPROM location during initialization. If that location does not contain a value, then the terminal is interrogated to ascertain the terminal baud rate. If this also fails, the baud rate is set to 1200.

3.5.1.4 Console Serial Line Interrupt Levels and Vectors

The console serial line receiver and transmitter both generate interrupts at IPL 15 (hex). The receiver interrupts with a vector of F8 (hex), and the transmitter interrupts with a vector of FC (hex).

3.5.2 Time-of-Year Clock and Timers

The KA64A CPU module includes a time-of-year clock, a subset of the VAX interval clock, and two additional programmable timers. The implementation of the time-of-year clock (TODR) and the subset of the interval clock (ICCS) are as described in the *VAX Architecture Reference Manual*. The programmable timers contain additional functions beyond the interval clock.

The two programmable timers are modeled after the standard VAX interval clock, with a control bit added to stop the timer on overflow. The programmable timers are accessed through I/O space registers instead of IPRs. If enabled for interrupts, the timers interrupt at IPL 15 (hex) on overflow. The SCB vector is programmable. Each timer is composed of four registers: a timer control register, a timer interval register, a timer next interval register, and a timer interrupt vector register.

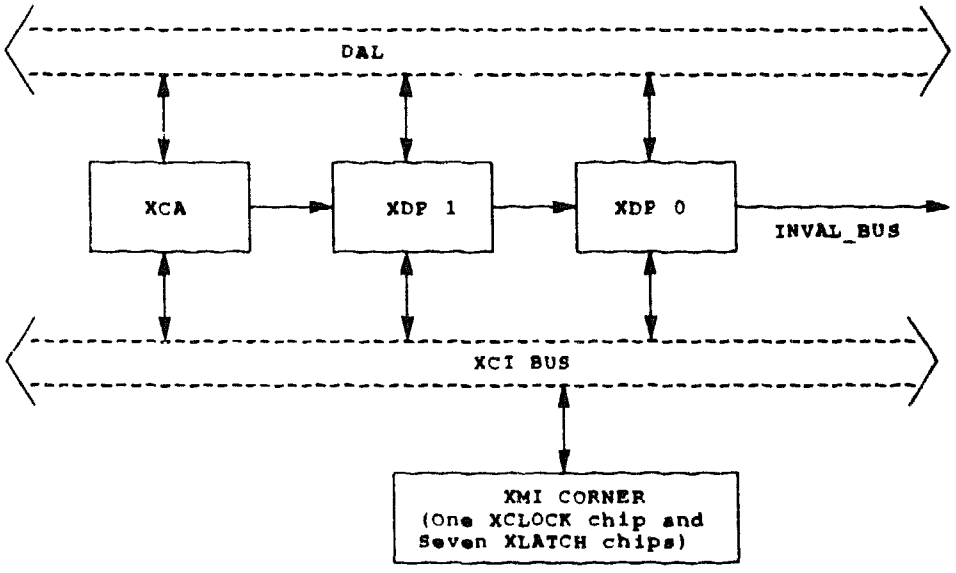
The time-of-year clock and timers include these registers (addresses are in hexadecimal):

- Time-of-Year Clock Register, TODR (IPR27)
- Interval Clock Control and Status Register, ICCS (IPR24)
- Timer Control Registers 0 and 1, TCR0 and TCR1 (2014 0160 and 2014 0170, respectively)
- Timer Interval Registers 0 and 1, TIR0 and TIR1 (2014 0164 and 2014 0174, respectively)
- Timer Next Interval Registers 0 and 1, TNIR0 and TNIR1 (2014 0168 and 2014 0178, respectively)
- Timer Interrupt Vector Registers 0 and 1, TIVR0 and TIVR1 (2014 016C and 2014 017C, respectively)

3.6 XMI Interface

The KA64A CPU module uses the REXMI chipset to interface between the XCI bus and the CPU module's DAL bus.

Figure 3–26 REXMI Block Diagram



The REXMI block diagram is shown in Figure 3–26. Two XDP data path chips and an XCA control/address chip make up the REXMI. Each XDP chip is responsible for 32 bits of the REXMI's 64-bit data path. The XCA chip is responsible for the address data path, the DAL (data/address lines) bus control logic, XMI control logic, and control of the two XDPs.

The KA64A CPU module and all other XMI nodes gain access to the XMI bus through the XMI Corner, a predefined area of an XMI module. The node logic, in turn, interfaces to the XMI Corner through the XCI (XMI Corner interface) bus. The XMI bus, protocol, address space, and XMI Corner are described in Chapter 2.

The main tasks of the REXMI are to:

- Translate CPU-chip memory and I/O space references to the appropriate XMI transactions
- Implement a write buffer to reduce traffic to main memory and improve processor performance
- Support control of cache fills and cache invalidates
- Support XMI-required interrupt logic
- Implement all XMI-required registers

3.6.1 KA64A CPU Module XMI Private I/O Address Space Map

Figure 3–27 KA64A CPU Module Private I/O Address Space Map

Byte Address		
2000 0000	Control Register (CREG) write-enable	
2000 0004 2000 3FFF	RESERVED	256 Kbytes
2004 0000 2007 FFFF	Self-Test/Console/Boot Code (halt protected) two (2) 128KB X 8 PROMs	256 Kbytes
2008 0000 2008 7FFF	Self-Test/Console/Boot Code (halt protected) one (1) 32KB X 8 EEPROM	32 Kbytes
2008 8000 200B FFFF	RESERVED	224 Kbytes
200C 0000 200F FFFF	Self-Test/Console/Boot Code (not halt protected, PROM)	256 Kbytes
2010 0000 2010 7FFF	Self-Test/Console/Boot Code (not halt protected, EEPROM)	32 Kbytes
2010 8000 2013 FFFF	RESERVED	224 Kbytes
2014 0000 2014 03FF	RSSC CSRs	1 Kbyte
2014 0400 2014 07FF	RSSC Battery-Backed-Up RAM	1 Kbyte
2014 0800 2100 FFFF	RESERVED	14.8 Mbytes
2101 0000 2101 FFFF	Interprocessor IVINTR Generation "Virtual" Registers	64 Kbytes
2102 0000 2102 FFFF	Write Error IVINTR Generation "Virtual" Registers	64 Kbytes
2103 0000 217F FFFF	RESERVED	7.75 Mbytes

Figure 3–27 shows the CPU module's implementation of its XMI private space. Addresses 2000 0000 to 217F FFFF (hex) of XMI I/O space are not accessible from the XMI bus.

Certain sections of the XMI private space contain self-test, console, and boot code that is halt protected. That is, the code continues to execute after a CTRL/P halt is received. Other sections of self-test, console, and boot code in XMI private space are not halt protected, and execution halts upon receipt of a CTRL/P halt.

3.6.2 XMI Transactions

The REXMI generates these XMI transactions:

- Octaword-length memory Reads
- Quadword-length memory Interlock Reads
- Quadword memory Write Masks
- Octaword memory Write Masks
- Quadword memory Unlock Write Masks
- Longword I/O Reads
- Longword I/O Interlock Reads
- Longword I/O Write Masks
- Longword I/O Unlock Write Masks
- Write error IVINTRs
- Interprocessor IVINTRs
- IDENTs (in response to CPU-chip interrupt acknowledge)

The REXMI responds to these XMI transactions:

- Longword-length nodespace Reads
- Longword-length nodespace Write Masks
- Interrupts (INTRs and IVINTRs)

All memory writes are monitored for cache invalidates.

Table 3–11 describes the REXMI generation and response for various CPU-chip-to-XMI operations while Table 3–12 describes the REXMI generation and response for various XMI-to-CPU-chip operations.

Table 3–11 REXMI Transaction Generation/Response for CPU-Chip-to-XMI Operations

CPU-Chip Operation	Resulting XMI Operation
Memory Space References	
Read (misses both caches)	Octaword Read
Interlock Read (forced to miss both caches)	Quadword Interlock Read
Write Mask	No XMI transaction generated. Data is loaded into write buffer and written to memory later with either a quadword Write Mask or an octaword Write Mask.
Unlock Write Mask (forced to miss write buffer)	Quadword Unlock Write Mask
I/O Space References	
Read (forced to miss both caches)	Longword Read
Interlock Read (forced to miss both caches)	Longword Interlock Read
Write Mask (forced to miss write buffer)	Longword Write
Unlock Write Mask (forced to miss write buffer)	Longword Unlock Write Mask
Miscellaneous References	
Interrupt acknowledge	XMI IDENT (assuming that an XMI interrupt is pending and no RSSC or IP IVINTR interrupts are pending)
I/O space write to IVINTR generation space	XMI IVINTR
Clear write buffer	Empty write buffer to memory and process all queued invalidates

Table 3–12 REXMI Transaction Generation/Response for XMI-to-CPU-Chip Operations

XMI Transaction	Resulting CPU-Chip Operation
XMI memory space writes (all types) from other nodes	Load invalidate queue, perform Inval-Bus lookup. If hit, then do invalidate on DAL at the next opportunity.
XMI memory space writes (all types) from same node	If RCSR<ESI>=1, load invalidate queue. Perform Inval-Bus lookup. If hit, then do invalidate on DAL at the next opportunity.
XMI writes to XMI nodespace	Write the appropriate CSR.
XMI read to XMI nodespace	Respond with appropriate CSR data.
XMI INTR	Set the appropriate interrupt-pending bit and post interrupt request to CPU-chip.
XMI interprocessor IVINTR	Set the IP IVINTR pending bit and post IPL 16 (hex) interrupt request.
XMI write error IVINTR	Set XBER<WEI> and post "hard error" interrupt.
XMI parity error detected	Set XBER<PE> and post a "soft error" interrupt. If "inconsistent," also set XBER<IPE>.
XMI IDENT	Clear the appropriate interrupt-pending bit.
XMI FAULT bit asserted	Set XBER<XFAULT> and post "hard error" interrupt.

3.6.3 Invalidates

The REXMI monitors all write traffic from other nodes to memory space to maintain cache coherency between the KA64A CPU module caches and main memory. When the REXMI detects an XMI write to a currently cached memory location, the REXMI must perform an invalidate on the DAL. The C-chip works with the REXMI to determine if data is currently cached. The C-chip maintains a duplicate copy of primary cache tag store that it accesses in parallel with the backup cache tag store whenever invalidate addresses are placed on the Inval-Bus. If the C-chip detects an Inval-Bus address match in either tag store, it notifies the REXMI of the hit. This implements a filtering mechanism for invalidates that avoids using the DAL except when a write address is found cached.

When the REXMI detects a memory write by another node on the XMI, it places the write address into an invalidate queue. This address is driven onto the Inval-Bus, and the REXMI requests the C-chip to do a cache lookup. If this address misses in both caches, the REXMI deletes the entry from the invalidate queue and goes on to the next entry. If the address hits in either cache, the REXMI performs a DMA write transaction on the DAL to this address, invalidating the address in both caches.

The REXMI's invalidate queue is 16 entries deep. The REXMI uses the XMI suppress line (XMI SUP L) to suppress XMI transactions, keeping the invalidate queue from overflowing. If 14 or more entries in the invalidate queue are valid, the REXMI asserts the suppress line, permitting up to two more XMI writes to occur. The suppression of XMI commands allows the REXMI to catch up on invalidate processing and to open up queue entries for future invalidate addresses. When the number of valid entries drops below 14, the REXMI deasserts the suppress line.

The REXMI processes all invalidates that arrived in XMI cycles before the read command is ACKed on the XMI in response to a read-type DAL command (memory read, I/O space read, read interrupt vector). The REXMI accomplishes this by logically "marking" the valid entries in the invalidate queue when the entries must all be processed before this data is returned. If the XMI command is retried, the valid entries in the invalidate queue are re-marked every time the command/address cycle is driven onto the XMI.

If the REXMI finds an entry that corresponds to data being cached during the processing of the marked entries in the invalidate queue, the REXMI requests a retry of the command on the DAL and requests ownership of the DAL to process the invalidate. Once the REXMI has processed all marked entries in the invalidate queue and data has been returned from the XMI, the REXMI releases DAL ownership. The CPU-chip then retries the read-type command and the REXMI responds with the data that it buffered from the XMI.

If an invalidate address that is in the same cache block as an outstanding cacheable memory read arrives between the cycle that the command/address for the read is driven on the XMI and the time that the final data is driven back on the DAL, invalidate lookups may produce incorrect answers. For cacheable memory reads, the REXMI suspends invalidate queue processing after all marked entries have been completed. Invalidate queue processing resumes after all read data has been returned on the DAL, which is after the DMA cache fill transaction (except for read locks, since no fill is done). Invalidate queue processing is not suspended for I/O space reads or for read interrupt vector transactions.

The REXMI processes all queued invalidates before acknowledging the DAL transaction during a clear write buffer command. If any of these invalidates hit in either cache, the REXMI retries the clear write buffer command and processes all invalidates that hit on the DAL before letting the CPU-chip retry the clear write buffer command.

3.6.4 Write Buffer

Noninterlocked memory writes are buffered by the REXMI through a four-octaword write buffer. The write buffer combines multiple CPU write requests into XMI octaword writes, conserving XMI bandwidth. The REXMI loads write data into a write buffer entry until it receives a write to a new octaword address or until a purge write buffer condition occurs.

In the absence of a write to a new octaword address or a purge write buffer condition, data remains buffered in the currently open entry and is not written to memory. When the REXMI receives a write to a new octaword address, the current write buffer entry is marked full, and a new write buffer is opened with the new octaword address. If a purge write buffer condition occurs, the REXMI marks the current entry full if it contains valid write data. The REXMI then waits until all full entries are sent over the XMI.

The REXMI does not wait for a memory write to reach the XMI before acknowledging the DAL write. Instead, the REXMI acknowledges the write immediately after it determines that the write buffer can accept the current write data, freeing the DAL for other transactions.

An XMI write cycle begins when at least one write buffer entry is marked full. An XMI quadword write is generated if the data fits into one aligned quadword. Otherwise, the REXMI sends an XMI octaword write to main memory. The write buffer data is transmitted to main memory in the same order as the entries were generated.

A full write buffer condition exists when three entries become full and the remaining entry contains valid write data. During a full write buffer condition, the REXMI requests a retry of the transaction in response to all noninterlocked memory write commands until one of the write buffer entries is emptied.

A write buffer address content addressable memory (CAM) is associated with the write buffer. The write buffer address CAM compares read request addresses with pending write request addresses. If a CAM match occurs on a read, the entire write buffer is flushed (the data in the write buffer is transmitted over the XMI bus to memory and then the write buffer is cleared) before the read request starts on the XMI.

The REXMI never terminates a write DAL transaction in error. If an error occurs during the write, the REXMI acknowledges the DAL transaction and reports the error via an IPL 1D (hex) hard error interrupt request.

The REXMI automatically flushes the write buffer:

- Before an XMI I/O space read or write is performed. While XMI private space writes are in I/O space, they do not cause the write buffer to flush, except for writes to IVINTR generation space for both interprocessor and write error IVINTRs.
- Before an Interlock Read or Unlock Write Mask reference is performed.
- Before the REXMI issues an XMI read to an octaword location that includes data contained in the write buffer. The write buffer contents are flushed to main memory and then the XMI read is issued. Reads that "miss" the write buffer do not force a write buffer flush (reads are bypassed around nonconflicting memory writes).
- In response to a clear write buffer command from the CPU-chip. The CPU-chip microcode generates explicit clear write buffer commands in the following cases:
  - At the exit from all interrupt and exception processing, after any exception-specific stack writes are performed, including all interrupts (software, device, and urgent) and all exceptions (faults, traps, aborts, and machine checks). This type of exit is the generic microcode exit from any condition that starts a new code flow based on a vector read from the SCB.
  - At the end of CHMx, after the arguments have been pushed on the stack.
  - At the end of REI.
  - At the end of SVPCTX, after all PCB writes have been done.
  - At the end of LDPCTX, after state is restored and the PC/PSL pair has been pushed on the stack.
  - At entry to console mode as a result of a microcode-detected error condition, kernel-mode, or front-panel halt request.
  - After a kernel-mode MTPR write to the MAPEN bit in IPR56.
  - At the start of machine check processing.
  - At the start of microcode I-stream error processing.
  - At the end of the execution of any Move From Vector Processor (MFVP) instruction or an MFPR from a 9x (hex) vector module register.
  - After the execution of an MSYNC vector instruction (see Section 4.4.4.10).

When the console program does a DEPOSIT command, the program explicitly flushes the deposit data to its destination by performing a read of XDEV. The REXMI does not automatically flush the write buffer in this case.

3.6.5 Interrupts

The XMI supports device interrupts (INTRs) and implied vector interrupts (IVINTRs). Device interrupts are generated by the XMI I/O devices. IVINTRs are either interprocessor interrupts or write error interrupts. The REXMI responds to all XMI interrupts and generates IVINTRs.

3.6.5.1 Device Interrupts (INTRs)

The four priority levels of XMI device interrupts are IPL 14 (hex) through IPL 17 (hex). The REXMI keeps a set of 32 interrupt-pending bits (eight I/O-capable nodes with four IPL levels). If an interrupt command with an interrupt destination field matching the node ID of the CPU module is received, the REXMI sets the appropriate interrupt-pending bit, based on the IPL of the interrupt and the node ID of the commander. All eight bits at each interrupt level are ORed together and sent to the CPU on device interrupt lines IRQ L<3:0>.

3.6.5.2 Implied Vector Interrupts (IVINTRs)

The REXMI generates and responds to interprocessor (IP) and write error IVINTRs.

The REXMI sets an interprocessor interrupt-pending bit when an IP IVINTR command with an interrupt destination field matching the CPU module's node ID is received. This bit is ORed with the eight IPL 16 (hex) interrupt-pending bits and sent to the CPU-chip.

Write error interrupts cause the REXMI to set XBER<WEI> (write error interrupt) and request a hard error interrupt at IPL 1D (hex).

A byte-length I/O space write, such as MOVB or CLRB, to the IVINTR-generation "virtual" registers causes the REXMI to generate an IVINTR command on the XMI bus. The addresses, in hexadecimal, for these IVINTR-generation "virtual" registers are:

- 2101 0000 to 2101 FFFF for IP IVINTRs
- 2102 0000 to 2102 FFFF for WEIs

The low 16 bits of the address are used as the XMI destination mask during the IVINTR command cycle for both types of IVINTRs. The REXMI treats IVINTR transactions like normal I/O space writes on the DAL. The write buffer is flushed to memory before the IVINTR command is sent. The REXMI does not acknowledge the write until after the ACK for the IVINTR command is received. If an error occurs on the IVINTR command, the DAL transaction is acknowledged, and the error is reported by requesting a hard error interrupt at IPL 1D (hex).

3.6.5.3 Read Interrupt Vector and IDENT

The CPU-chip generates a read interrupt vector transaction in response to the assertion of one of the device interrupt lines, IRQ L<3:0>. The interrupt vector can come from an XMI I/O device, the RSSC, or the REXMI (if an IP IVINTR is pending). The IPL of RSSC interrupts is programmable through control bits in the RSSC and the REXMI, and is set to IPL 15 (hex) by the console program. IP IVINTRs are serviced at IPL 16 (hex). If multiple interrupts are pending at the same level, the REXMI gives the read interrupt vector command priority in the order:

- RSSC interrupts (only if the read interrupt vector is at the RSSC-programmed IPL, which should be IPL 15 (hex))
- IP IVINTRs (IPL 16 (hex) only)
- XMI interrupts

The REXMI determines the source of the interrupt vector after receiving a read interrupt vector command. If an RSSC interrupt is pending at the RSSC-programmed IPL (normally, IPL 15 (hex)), the REXMI allows the RSSC to service the interrupt request. If the interrupt is at IPL 16 (hex) and an IP IVINTR is pending, the REXMI returns a vector of 80 (hex) to the CPU-chip. If the interrupt source is the XMI bus, the REXMI sends an XMI IDENT command to request the interrupt vector.

While the REXMI waits for the vector offset from the XMI during an XMI IDENT, any invalidate that hits in either cache forces the REXMI to request a retry of the read interrupt vector transaction, to allow serialization of events on the XMI during read operations (memory reads, I/O space reads, and read interrupt vectors). All invalidates received prior to the ACK of the IDENT command are processed. Once the vector is returned and all invalidates are processed, the REXMI allows the CPU-chip to retry the read interrupt vector transactions. The REXMI then returns the vector it buffered from the XMI.

If the REXMI receives a read interrupt vector transaction at the RSSC-programmed IPL and there is no RSSC interrupt pending, it transmits an IDENT on the XMI. If an invalidate hit occurs, the REXMI retries the read interrupt vector transaction so it can first process the invalidate. If an RSSC interrupt request is pending when the read interrupt vector transaction is retried, the REXMI cannot allow the RSSC to respond to the transaction, as this would cause the IDENT to be lost. Instead, the IDENT data is returned to satisfy the transaction, and service of the RSSC interrupt is delayed until the original interrupt is completed.

If an XMI error occurs during an IDENT transaction, the REXMI terminates the DAL transaction in error and reports the error by requesting a hard error interrupt at IPL 1D (hex).

3.6.6 XMI Registers

The REXMI registers are located in the KA64A CPU module's nodespace and are directly accessible by all XMI nodes. A read or write access to nonimplemented nodespace causes the REXMI to NO ACK the XMI transaction. Nonimplemented nodespace includes node 0 and node F (hex) and empty slots on the XMI backplane.

Addresses, in hexadecimal, of the REXMI registers are as follows:

- XMI Device Register, XDEV (nodespace + 0000 0000)
- XMI Bus Error Register, XBER (nodespace + 0000 0004)
- XMI Failing Address Register, XFADR (nodespace + 0000 0008)
- XMI General Purpose Register, XGPR (nodespace + 0000 000C)
- REXMI Control and Status Register, RCSR (nodespace + 0000 0010)

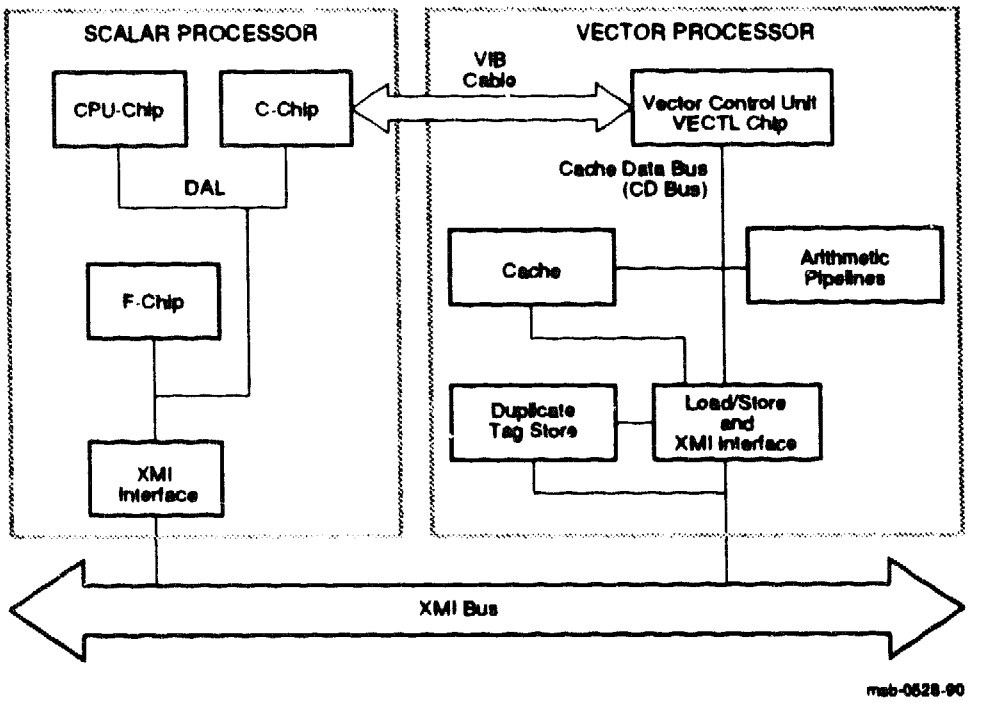
The REXMI's XMI registers have the following characteristics:

- The mask bits are ignored on writes to the REXMI's CSRs. A full longword write is always performed.
- Interlocks are not supported. Interlock Read and Unlock Write Mask commands are treated like Read and Write Mask commands, respectively.
- The XMI responder queue is only one deep so the REXMI NO ACKs subsequent CSR references until the read data for the queued CSR read has been returned.

3.7 Scalar/Vector Interaction

The KA64A CPU module supports an optional attached vector module. For details on the FV64A vector processor, see Chapter 4.

Figure 3–28 Scalar/Vector Pair Block Diagram



The scalar module includes support for an optional vector module that executes VAX vector instructions. The interface between the CPU-chip and the vector module is provided by the C-chip (see Figure 3–28). Command and control information are transferred between the C-chip and the vector module over the vector interface bus, the VIB. The VIB is a cable between the scalar and vector modules.

Control of the vector module is distributed between the vector interface in the C-chip and the vector module itself. The C-chip contains the Vector Interface Error Status Register (VINTSR), IPR123, which contains information about the scalar CPU side of the VIB (see Section 3.8 for the VINTSR description). Information about the vector module is also provided by the scalar module's Accelerator Control and Status Register (ACCS), IPR40. ACCS<0>, when set, indicates that a vector module is attached to the scalar module.

The registers that reside on the vector module contain information about the vector module side of the interface. For descriptions of these registers, see Section 4.8.

3.7.1 Vector Instruction Execution

To an executing program, the scalar/vector pair of modules appears as one processor. The CPU-chip decodes vector instructions and parses their specifiers as it does for other instructions. Then one, two, or three longwords of operand and control information are transferred to the vector module using IPR write commands.

Operand information, if any, is transferred to the vector module in reverse specifier order. That is, the last specifier's operand is transferred first, followed by the next-to-last, and so forth. The final transfer passes the opcode and control information to the vector module, which validates any previous transfers and enables the vector module to begin executing the instruction.

The vector module executes most vector instructions in parallel with the scalar CPU execution of subsequent instructions. Some vector instructions (the load, store, gather, scatter, MFVP, and MFPR instructions) require the synchronization of the two modules. To synchronize, the scalar module reads the Vector Memory Activity Check (VMAC) Register at the end of the instruction. Until the vector module responds the CPU-chip stalls, effectively forcing the synchronous execution of (at least part of) the vector instruction.

The programmer must take special steps to ensure that this barrier synchronization operates correctly.

CAUTION: To ensure that the barrier synchronization operates correctly, the programmer must read the Vector Processor Status Register (VPSR), IPR144, and spin on reading this register until the VPSR Busy bit is clear. Then reading the Vector Memory Activity Check Register (VMAC), IPR146, *twice* guarantees synchronization and that all errors have been reported.

3.7.2 Exceptions and Errors

There are three types of exceptions reported by the vector module: memory management exceptions, hardware errors reported through machine checks, and hardware errors reported through interrupts.

Memory management exceptions are reported to the CPU-chip in response to the register read done at the end of the CPU-chip instruction execution. The CPU-chip passes these exceptions to the operating system through the normal ACV, TNV, and machine check SCB vectors.

Hardware errors that are reported by a machine check are reported using the system control block (SCB) machine check vector 04 (hex). Other hardware errors are reported by hard error interrupts to the CPU using SCB vector 60 (hex). Soft errors are reported using SCB machine check vector 54 (hex). See Section 3.3.4 for more on exceptions and interrupts.

Exceptions other than memory management exceptions cause the vector module to disable itself. Hard errors also disable the vector processor. The disabled condition is detected when the CPU-chip attempts to issue another instruction to the vector module and at the end of all MFPR instructions. Both cases result in a vector module disabled fault (SCB vector 68 hex) to the operating system.

3.8 KA64A CPU Module Registers

The KA64A CPU module registers consist of internal processor registers, KA64A CPU module registers in XMI private space, and XMI required registers.

3.8.1 Internal Processor Registers

The processor state is stored in internal processor registers rather than in memory. See Table 3–13 and Table 3–14. The processor state is composed of 16 general purpose registers (GPRs), the processor status longword (PSL), and internal processor registers (IPRs).

Nonprivileged software can access the GPRs and the lower half of the processor status longword (PSL<15:0>). The IPRs and PSL<31:16> can only be accessed by privileged software. The IPRs are explicitly accessible by the Move To Processor Register (MTPR) and Move From Processor Register (MFPR) instructions, which require kernel mode privileges. The console operator can read an IPR with the EXAMINE/I command and write an IPR with the DEPOSIT/I command.

Table 3–13 KA64A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
0 (0)	Kernel Stack Pointer	KSP	R/W	PROC	1	CPU-chip
1 (1)	Executive Stack Pointer	ESP	R/W	PROC	1	CPU-chip
2 (2)	Supervisor Stack Pointer	SSP	R/W	PROC	1	CPU-chip
3 (3)	User Stack Pointer	USP	R/W	PROC	1	CPU-chip
4 (4)	Interrupt Stack Pointer	ISP	R/W	CPU	1	CPU-chip

[1]See Table 3–14.

[2]Key to Scopes:

PROC = Per-process register, loaded by LDPCTX (load process context instruction).
CPU = CPU-wide register, not affected by LDPCTX.

[3]Key to Classes:

1 = Implemented by the KA64A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA64A CPU module.
3 = Implemented by the FV64A vector processor module.
4 = Not implemented. Read as zero; NOP on write. These registers should not to be referenced during normal operation as no other instructions can be executed by the CPU until a timeout period that might be longer than device or CPU timeouts has expired.
5 = Access not allowed; accesses result in a reserved operand fault.
6 = Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n Init = The register is initialized on a KA64A CPU module reset (power-up, system reset, and node reset).

Table 3–13 (Cont.) KA64A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
5 – 7 (5 – 7)	Reserved				4	
8 (8)	P0 Base	P0BR	R/W	PROC	1	CPU-chip
9 (9)	P0 Length	P0LR	R/W	PROC	1	CPU-chip
10 (A)	P1 Base	P1BR	R/W	PROC	1	CPU-chip
11 (B)	P1 Length	P1LR	R/W	PROC	1	CPU-chip
12 (C)	System Base	SBR	R/W	CPU	1	CPU-chip
13 (D)	System Length	SLR	R/W	CPU	1	CPU-chip
14 – 15 (E – F)	Reserved				4	
16 (10)	Process Control Block Base	PCBB	R/W	PROC	1	CPU-chip
17 (11)	System Control Block Base	SCBB	R/W	CPU	1	CPU-chip
18 (12)	Interrupt Priority Level	IPL	R/W	CPU	1 Init	CPU-chip
19 (13)	AST Level	ASTLVL	R/W	PROC	1 Init	CPU-chip
20 (14)	Software Interrupt Request	SIRR	WO	CPU	1	CPU-chip
21 (15)	Software Interrupt Summary	SISR	R/W	CPU	1 Init	CPU-chip
22 – ` (16 – 17)	Reserved				4	
24 (18)	Interval Clock Control and Status[4]	ICCS	R/W	CPU	2 Init	CPU-chip
25 – 26 (19 — 1A)	Reserved				4	
27 (1B)	Time-of-Year Clock[5]	TODR	R/W	CPU	1	RSSC
28 (1C)	Console Storage Receiver Status	CSRS	R/W	CPU	6 Init	RSSC
29 (1D)	Console Storage Receiver Data	CSRD	RO	CPU	6 Init	RSSC
30 (1E)	Console Storage Transmitter Status	CSTS	R/W	CPU	6 Init	RSSC

[1]See Table 3–14.

[2]Key to Scopes:

PROC = Per-process register, loaded by LDPCTX (load process context instruction).
CPU = CPU-wide register, not affected by LDPCTX.

[3]Key to Classes:

1 = Implemented by the KA64A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA64A CPU module.
3 = Implemented by the FV64A vector processor module.
4 = Not implemented. Read as zero; NOP on write. These registers should not to be referenced during normal operation as no other instructions can be executed by the CPU until a timeout period that might be longer than device or CPU timeouts has expired.
5 = Access not allowed; accesses result in a reserved operand fault.
6 = Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n Init = The register is initialized on a KA64A CPU module reset (power-up, system reset, and node reset).

[4]Interval timer requests are posted at IPL 16 with a vector of C0 (hex). The interval timer is the lowest priority device at the IPL.

[5]TODR is maintained during power failure by the XMI TOY BBU PWR line on the XMI backplane.

Table 3–13 (Cont.) KA64A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
31 (1F)	Console Storage Transmitter Data	CSTD	WO	CPU	6 Init	RSSC
32 (20)	Console Receiver Control/Status	RXCS	R/W	CPU	2 Init	RSSC
33 (21)	Console Receiver Data Buffer	RXDB	RO	CPU	2 Init	RSSC
34 (22)	Console Transmitter Control/Status	TXCS	R/W	CPU	2 Init	RSSC
35 (23)	Console Transmitter Data Buffer	TXDB	WO	CPU	2 Init	RSSC
36 – 37 (24 – 25)	Reserved				4	
38 (26)	Machine Check Error Summary	MCESR	WO	CPU	2	CPU-chip
39 (27)	Reserved				4	
40 (28)	Accelerator Control and Status	ACCS	R/W	CPU	2 Init	CPU-chip
41 (29)	Reserved				4	
42 (2A)	Console Saved Program Counter	SAVPC	RO	CPU	2	CPU-chip
43 (2B)	Console Saved Processor Status Longword	SAVPSL	RO	CPU	2	CPU-chip
44 – 46 (2C – 2E)	Reserved				4	
47 (2F)	Translation Buffer Tag	TBTAG	WO	CPU	2	CPU-chip
48 – 54 (30 – 36)	Reserved				4	
55 (37)	I/O Reset	IORESET	WO	CPU	2	RSSC
56 (38)	Memory Management Enable	MAPEN	R/W	CPU	1 Init	CPU-chip
57 (39)	Translation Buffer Invalidate All	TBIA	WO	CPU	1	CPU-chip
58 (3A)	Translation Buffer Invalidate Single	TBIS	WO	CPU	1	CPU-chip
59 (3B)	Translation Buffer Data	TBDATA	WO	CPU	2	CPU-chip
60 – 61 (3C – 3D)	Reserved				4	
62 (3E)	System Identification	SID	RO	CPU	1	CPU-chip
63 (3F)	Translation Buffer Check	TBCHK	WO	CPU	1	CPU-chip

[1]See Table 3–14.

[2]Key to Scopes:

PROC = Per-process register, loaded by LDPCTX (load process context instruction).
CPU = CPU-wide register, not affected by LDPCTX.

[3]Key to Classes:

1 = Implemented by the KA64A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA64A CPU module.
3 = Implemented by the FV64A vector processor module.
4 = Not implemented. Read as zero; NOP on write. These registers should not to be referenced during normal operation as no other instructions can be executed by the CPU until a timeout period that might be longer than device or CPU timeouts has expired.
5 = Access not allowed; accesses result in a reserved operand fault.
6 = Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n Init = The register is initialized on a KA64A CPU module reset (power-up, system reset, and node reset).

Table 3–13 (Cont.) KA64A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
64 – 111 (40 – 6F)	Reserved				4	
112 (70)	Backup Cache Reserved	BC112	R/W	CPU	6	C-chip
113 (71)	Backup Cache Backup Tag Store	BCBTS	R/W	CPU	2	C-chip
114 (72)	Backup Cache Primary Cache Tag Store, first half	BCP1TS	R/W	CPU	2	C-chip
115 (73)	Backup Cache Primary Cache Tag Store, second half	BCP2TS	R/W	CPU	2	C-chip
116 (74)	Backup Cache Refresh	BCRFR	R/W	CPU	2	C-chip
117 (75)	Backup Cache Index	BCIDX	R/W	CPU	2	C-chip
118 (76)	Backup Cache Status	BCSTS	R/W	CPU	2 Init	C-chip
119 (77)	Backup Cache Control	BCCTL	R/W	CPU	2 Init	C-chip
120 (78)	Backup Cache Error Address	BCERR	RO	CPU	2	C-chip
121 (79)	Backup Cache Flush Backup Cache Tag Store	BCFBTS	WO	CPU	2	C-chip
122 (7A)	Backup Cache Flush Primary Cache Tag Store	BCFPTS	WO	CPU	2	C-chip
123 (7B)	Vector Interface Error Status	VINTSR	R/W	CPU	2	C-chip
124 (7C)	Primary Cache Tag Array	PCTAG	R/W	CPU	2	CPU-chip
125 (7D)	Primary Cache Index	PCIDX	R/W	CPU	2	CPU-chip
126 (7E)	Primary Cache Error Address	PCERR	R/W	CPU	2	CPU-chip
127 (7F)	Primary Cache Status	PCSTS	R/W	CPU	2 Init	CPU-chip
128 – 143 (80 – 8F)	Reserved				4	
144 (90)	Vector Processor Status	VPSR	R/W	CPU	3	VECTL
145 (91)	Vector Arithmetic Exception	VAER	R/W	CPU	3	VECTL
146 (92)	Vector Memory Activity Check	VMAC	RO	CPU	3	VECTL
147 (93)	Vector Translation Buffer Invalidate All	VTBIA	WO	CPU	3	L/S

[1]See Table 3–14.

[2]Key to Scopes:

PROC = Per-process register, loaded by LDPCTX (load process context instruction).
CPU = CPU-wide register, not affected by LDPCTX.

[3]Key to Classes:

1 = Implemented by the KA64A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA64A CPU module.
3 = Implemented by the FV64A vector processor module.
4 = Not implemented. Read as zero; NOP on write. These registers should not to be referenced during normal operation as no other instructions can be executed by the CPU until a timeout period that might be longer than device or CPU timeouts has expired.
5 = Access not allowed; accesses result in a reserved operand fault.
6 = Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n Init = The register is initialized on a KA64A CPU module reset (power-up, system reset, and node reset).

Table 3–13 (Cont.) KA64A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
148 – 156 (94 – 9C)	Reserved				6	
157 (9D)	Vector Indirect Address	VIADR	R/W	CPU	3	VECTL
158 (9E)	Vector Indirect Data Low	VIDLO	R/W	CPU	3	VECTL
159 (9F)	Vector Indirect Data High	VIDHI	R/W	CPU	3	VECTL
160 – 255 (A0 – FF)	Reserved				4	
256 (100) and up	Reserved				5	

[1]See Table 3–14.

[2]Key to Scopes:

PROC = Per-process register, loaded by LDPCTX (load process context instruction).
CPU = CPU-wide register, not affected by LDPCTX.

[3]Key to Classes:

1 = Implemented by the KA64A CPU module as specified in the *VAX Architecture Reference Manual*.
2 = Implemented uniquely by the KA64A CPU module.
3 = Implemented by the FV64A vector processor module.
4 = Not implemented. Read as zero; NOP on write. These registers should not to be referenced during normal operation as no other instructions can be executed by the CPU until a timeout period that might be longer than device or CPU timeouts has expired.
5 = Access not allowed; accesses result in a reserved operand fault.
6 = Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n Init = The register is initialized on a KA64A CPU module reset (power-up, system reset, and node reset).

NOTE: The vector internal processor registers IPR144 through IPR159 are described in Chapter 4.

Table 3–14 Types of Registers and Bits

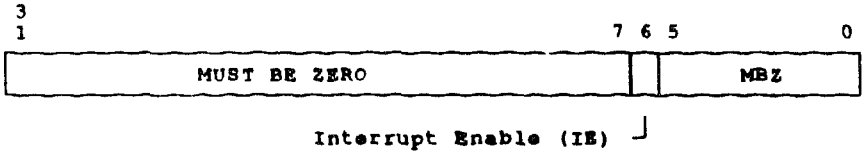
Type	Description
0	Initialized to logic level zero
1	Initialized to logic level one
X	Initialized to either logic level
RO	Read only
R/W	Read/write
R/Cleared on W	Read/cleared on write
R/W1C	Read/cleared by writing a one
WO	Write only

Interval Clock Control and Status Register (ICCS)

The ICCS contains control information for the interval clock. The interval clock is used for accounting, for time-dependent events, and for maintaining the software date and time. Interval timer requests are posted at IPL 16 (hex) through SCB vector C0 (hex). The interval timer is the lowest priority device at IPL 16.

ADDRESS

IPR24 (CPU-chip)



bits<31:7>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<6>

Name: Interrupt Enable

Mnemonic: IE

Type: R/W, 0

IE enables and disables interval timer interrupts. When IE is set, an interval timer interrupt is requested every 10 milliseconds. When IE is clear, interval timer interrupts are disabled.

bits<5:0>

Name: Reserved

Mnemonic: None

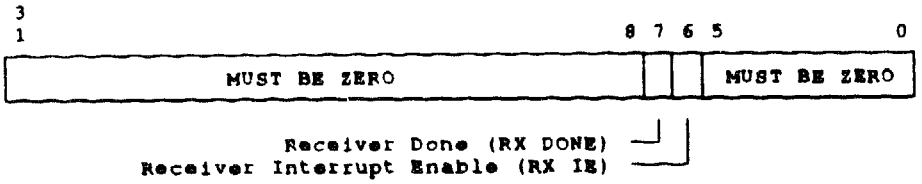
Type: –

Reserved; must be zero.

Console Receiver Control and Status Register (RXCS)

The RXCS controls and reports the status of incoming data on the console serial line.

ADDRESS *IPR32 (RSSC)*



bits<31:8>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<7>

Name: Receiver Done
Mnemonic: RX DONE
Type: RO, 0

RX DONE is set when an entire character has been received and is ready to be read from RXDB<7:0> (Received Data). RX DONE is automatically cleared when RXDB<7:0> is read.

bit<6>

Name: Receiver Interrupt Enable
Mnemonic: RX IE
Type: R/W, 0

RX IE enables receiver interrupts. If RX IE is set and a character is received, as indicated by the setting of RX DONE, a receiver interrupt is requested. If RX DONE is set and software then sets RX IE, a receiver interrupt is requested. The interrupt request is cleared when it is serviced, when RXBD is read, or when RX IE is cleared by software.

bits<5:0>

Name: Reserved

Mnemonic: None

Type: –

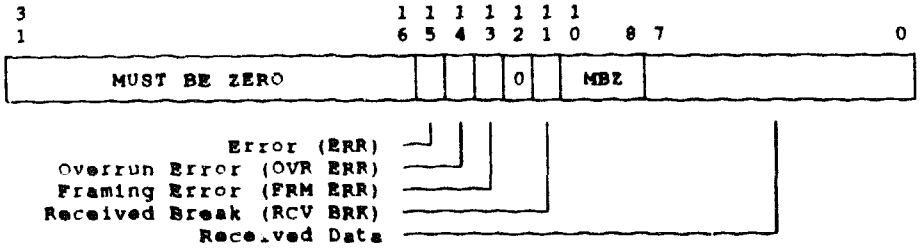
Reserved; must be zero.

Console Receiver Data Buffer Register (RXDB)

RXDB buffers incoming serial-line data and captures error information. Error conditions remain until the next character is received, at which point the error bits are updated.

ADDRESS

IPR33 (RSSC)



bits<31:16>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<15>

Name: Error

Mnemonic: ERR

Type: RO, 0

ERR is set if either bit<14> or <13> is set. ERR is clear if both bits are clear.

bit<14>

Name: Overrun Error

Mnemonic: OVR ERR

Type: RO, 0

OVR ERR is set if a previously received character was not read before being overwritten by the present character and remains set until the register is read.

bit<13>

Name: Framing Error
Mnemonic: FRM ERR
Type: RO, 0

FRM ERR is set if the present character did not have a valid stop bit and remains set until the register is read.

bit<12>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<11>

Name: Received Break
Mnemonic: RCV BRK
Type: RO, 0

RCV BRK is set following the receipt of a CTRL/P character and remains set until the register is read.

bits<10:8>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<7:0>

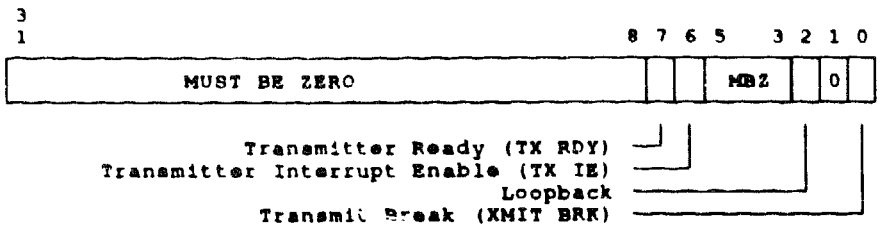
Name: Received Data
Mnemonic: None
Type: RO

Received Data contains the last character received from the console.

Console Transmitter Control and Status Register (TXCS)

TXCS controls and reports the status of outgoing data on the console serial line.

ADDRESS *IPR34 (RSSC)*



bits<31:8>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero

bit<7>

Name: Transmitter Ready

Mnemonic: TX RDY

Type: RO, 1

TX RDY sets when TXDB<7:0> (Transmit Data) can receive a character. It clears when TXDB<7:0> is loaded with a character, and it remains clear until the character is transferred to the serialization buffer.

bit<6>

Name: Transmitter Interrupt Enable

Mnemonic: TX IE

Type: R/W, 0

TX IE enables transmitter interrupts. If TX IE is set and the transmitter interrupt becomes ready (that is, TX RDX sets), then a transmitter interrupt is requested. If TX RDY is set and software sets TX IE, then a transmitter interrupt is requested. The interrupt request is cleared when it is serviced or if TX IE is cleared by software.

bits<5:3>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<2>

Name: Loopback
Mnemonic: None
Type: R/W, 0

When Loopback is set, loopback mode is enabled so that the external serial output is set to MARK and the serial output is connected to the serial input (a loopback).

bit<1>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<0>

Name: Transmit Break
Mnemonic: XMIT BRK
Type: R/W, 0

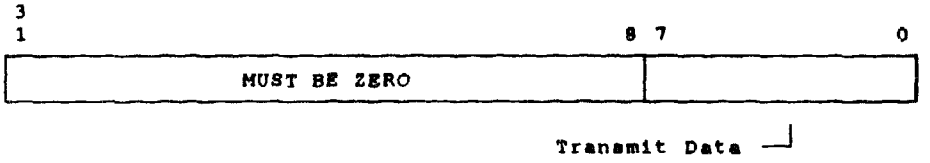
When XMIT BRK is set, the serial output is forced to the SPACE condition. While XMIT BRK is set, the transmitter operates normally, but the output line remains low so that software can transmit dummy characters to time the break.

Console Transmitter Data Buffer Register (TXDB)

TXDB buffers outgoing data on the console serial line.

ADDRESS

IPR35 (RSSC)



bits<31:8>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<7:0>

Name: Transmit Data

Mnemonic: None

Type: WO

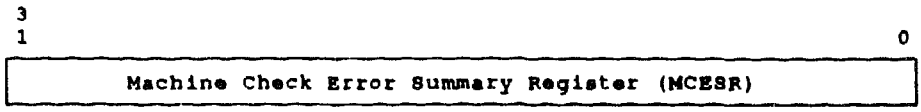
Transmit Data contains the character to be transmitted on the console serial line.

Machine Check Error Summary Register (MCESR)

MCESR allows software to acknowledge receipt of a machine check.

ADDRESS

IPR38 (CPU-chip)



bits<31:0>

Name: Machine Check Error Summary Register

Mnemonic: MCESR

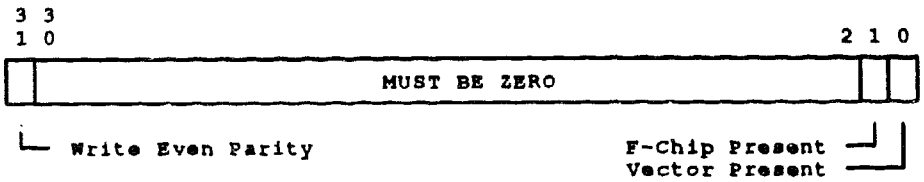
Type: WO

MCESR allows software to acknowledge receipt of a machine check. When the microcode invokes the software machine check handler, it sets a "machine check in progress" flag. If a machine check or memory management exception occurs when this flag is set, the microcode initiates a console double error halt. Machine check handler software needs to clear the "machine check in progress" flag as soon as possible by writing a zero to MCESR to reenable normal machine check and memory management exception reporting.

Accelerator Control and Status Register (ACCS)

The ACCS provides control for the F-chip, the optional vector module, and the generation of bad data parity on write operations.

ADDRESS *IPR40 (CPU-chip)*



bit<31>

Name: Write Even Parity

Mnemonic: None

Type: WO, 0

Write Even Parity enables the generation of bad data parity for write operations. If this bit is set to a one, all subsequent cache or memory writes and F-chip operand transfers are done with bad data parity on all bits. This bit is used for diagnostics only and is never set during normal operations.

Write Even Parity is automatically cleared if ACCS is read with an MFPR (Move From Processor Register) instruction. This bit is also cleared at the start of any interrupt, exception, or console halt, preventing an exception stack frame from being written with bad data parity.

CAUTION: The PTE.M bit MUST BE SET in any page table entry (PTE) mapping pages to be written while Write Even Parity is enabled. Otherwise, a PTE may be written with bad parity during a PTE.M bit update.

bits<30:2>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<1>

Name: F-Chip Present

Mnemonic: None

Type: R/W, 0

The F-Chip Present bit enables use of the F-chip. If this bit is a one, floating-point and longword-length integer multiply instructions are passed to the F-chip for execution. If this bit is a zero, the execution of a floating-point instruction results in a reserved instruction fault. F-Chip Present is set during normal operation. If an F-chip error is detected, the F-chip may be disabled if the operating system emulation software is loaded.

bit<0>

Name: Vector Present

Mnemonic: None

Type: R/W

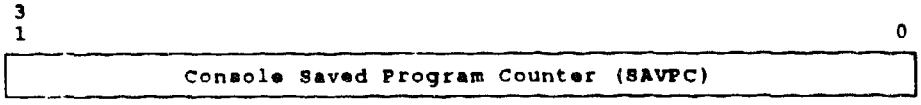
Vector Present enables use of the optional vector processor. If this bit is a one, vector instructions are decoded and passed to the vector processor for execution. If this bit is a zero, the execution of a vector instruction results in a reserved instruction fault. At power-up, the console code determines if a vector processor is in the system and, if the vector processor passes self-test and extended test, it sets the state of this bit. The bit is cleared during reset.

Console Saved Program Counter Register (SAVPC)

During a hardware restart sequence, the current program counter (PC) is saved in SAVPC.

ADDRESS

IPR42 (CPU-chip)



bits<31:0>

Name: Console Saved Program Counter

Mnemonic: SAVPC

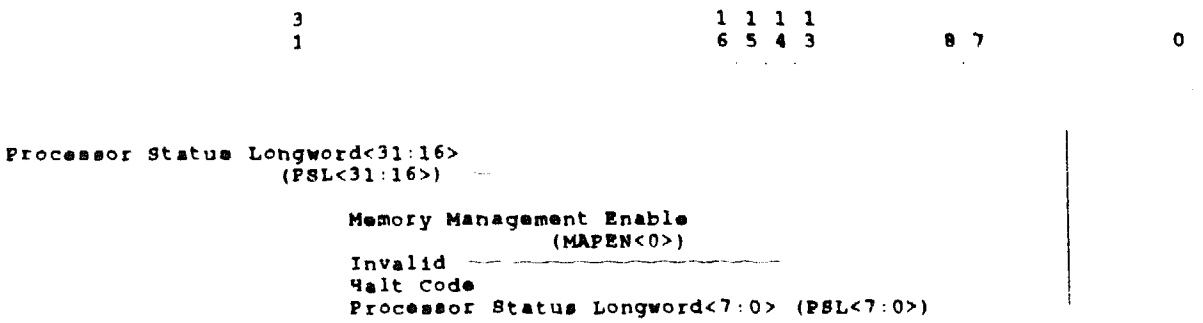
Type: RO

If the CPU-chip microcode detects an inconsistent internal state, an incorrectly terminated DAL transaction, a kernel-mode HALT, a system reset, a node halt, a node reset, or the detection of CTRL/P, the microcode initiates a console halt (a hardware restart sequence which passes control to the console code). During the hardware restart sequence, the current program counter (PC) is saved in SAVPC.

Console Saved Processor Status Longword (SAVPSL)

If the CPU-chip's microcode detects an inconsistent internal state, an incorrectly terminated DAL transaction, a kernel-mode HALT, or a system reset, the microcode initiates a console halt (a hardware restart sequence which passes control to the console code). During the hardware restart sequence, the processor status longword (PSL), halt code, MAPEN<0>, and an invalid bit are saved in SAVPSL.

ADDRESS *IPR43 (CPU-chip)*



bits<31:16>

Name: Processor Status Longword<31:16>

Mnemonic: PSL<31:16>

Type: RO

At the time of a console halt, PSL<31:16> is loaded into SAVPSL<31:16>.

bit<15>

Name: Memory Management Enable<0>

Mnemonic: MAPEN<0>

Type: RO

At the time of a console halt, MAPEN<0> is loaded into SAVPSL<15>.

bit<14>

Name: Invalid

Mnemonic: None

Type: RO

At the time of a console halt, a zero is loaded into the Invalid bit if the PSL is valid and a 1 is loaded if it is not valid. The Invalid bit is undefined after a halt due to a system reset.

bits<13:8>

Name: Halt Code

Mnemonic: None

Type: RO

At the time of a console halt, console halt code is loaded into SAVPSL<13:8>.

bits<7:0>

Name: Processor Status Longword<7:0>

Mnemonic: PSL<7:0>

Type: RO

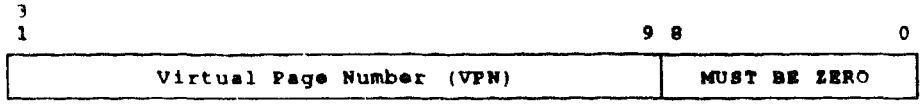
At the time of a console halt, PSL<7:0> is loaded into SAVPSL<7:0>.

Translation Buffer Tag Register (TBTAG)

TBTAG is a diagnostic register used to test the operation of the translation buffer. A write to TBTAG writes bits <31:9> of the source data into the Virtual Page Number (VPN) field of the current tag location and clears TB.V (valid bit). A subsequent write to TBDATA interprets the source data as a page table entry (PTE) and writes TPE.V (valid bit), PTE.M (modify bit), PTE.PROT (protection field), and PTE.PFN (page frame number) into the current PTE location, sets the TB.V bit, and increments the NLU pointer. This re ster is not written to during normal operation.

ADDRESS

IPR47 (CPU-chip)



bits<31:9>

Name: Virtual Page Number
Mnemonic: VPN
Type: WO

bits<8:0>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

I/O Reset Register (IORESET)

IORESET forces a system reset.

ADDRESS

IPR55 (RSSC)

bits<31:0>

Name: IORESET

Mnemonic: NONE

Type: R/W

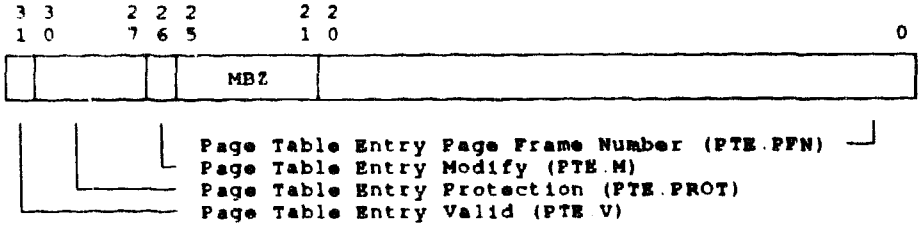
A KA64A CPU module can force a system reset by writing to IORESET with an MTPR (Move To Processor Register) instruction.

Translation Buffer Data Register (TBDATA)

TBDATA is a diagnostic register used to test the operation of the translation buffer (TB). A write to TBDATA, after a write to TBTAG, interprets the source data as a page table entry (PTE) and writes PTE.V (valid bit), PTE.M (modify bit), PTE.PROT (protection field), and PTE.PFN (page frame number) into the current PTE location, sets the TB.V (valid bit), and increments the not last used (NLU) pointer. This register is not written to during normal operation.

ADDRESS

IPR59 (CPU-chip)



bit<31>

Name: Page Table Entry Valid

Mnemonic: PTE.V

Type: WO

This field is a repeat of the entry valid bit of the VAX PTE. When PTE.V is a one, then PTE.M and PTE.PFN are valid.

bits<30:27>

Name: Page Table Entry Protection

Mnemonic: PTE.PROT

Type: WO

This field is a repeat of the protection field of the VAX PTE. It indicates what access modes a process can use to reference the page.

bit<26>

Name: Page Table Entry Modify

Mnemonic: PTE.M

Type: WO

This bit is a repeat of the modify bit of the VAX PTE. When M is one, the page may have been modified. M is not valid if V is zero.

bits<25:21>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<20:0>

Name: Page Table Entry Page Frame Number
Mnemonic: PTE.PFN
Type: WO

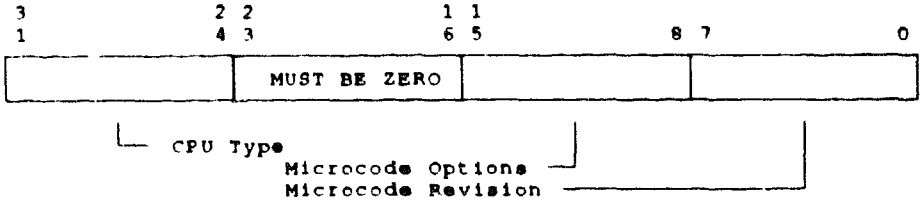
This field is a repeat of the page frame number field of the VAX PTE. It contains the upper 21 bits of the physical address of the base of the page. PFN is not used if V is zero.

System Identification Register (SID)

SID specifies the processor type and its microcode revision level. It can only be accessed locally. Other devices on the XMI determine the nature of a node by reading its XMI Device Register (XDEV).

ADDRESS

IPR62 (CPU-Chip)



bits<31:24>

Name: CPU Type

Mnemonic: None

Type: RO

This field is always 11 (decimal), indicating the KA64A CPU module's CPU-chip.

bits<23:16>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bits<15:8>

Name: Microcode Options

Mnemonic: None

Type: RO

Microcode Options specifies the microcode options included in this CPU-chip. At present, there are no defined options and the field contains zeros

bits<7:0>

Name: Microcode Revision

Mnemonic: None

Type: RO

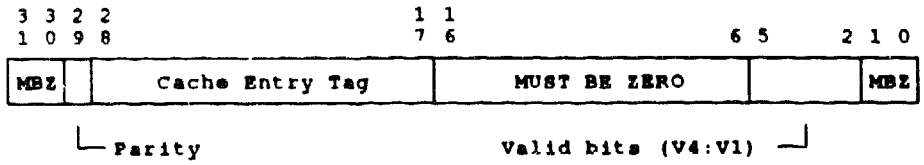
This field specifies the microcode revision of the CPU-chip.

Backup Cache Backup Cache Tag Store Register (BCBTS)

BCBTS is a processor register that is used to access the tag store, which contains 2048 entries of 17 bits each. Each entry consists of a parity bit, one 12-bit tag, and four valid bits for each 4-octaword block of the backup cache. The entries are arranged in rows that are eight entries wide. Only one entry can be accessed at a time by an MTPR/MFPR instruction. The location of BCBTS is pointed to by BCIDX.

ADDRESS

IPR113 (C-chip)



bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<29>

Name: Parity

Mnemonic: None

Type: R/W

The Parity bit contains odd parity for the 12-bit tag. The user supplies correct parity when writing to BCBTS.

bits<28:17>

Name: Cache Entry Tag

Mnemonic: None

Type: R/W

The Cache Entry Tag field contains bits<28:17> of the backup cache address.

bits<5:2>

Name: Valid

Mnemonic. None

Type: R/W

Each of the four Valid bits is associated with one octaword subblock of the 4-octaword block. Bits <5:4> of the backup cache address are used to select the valid bit corresponding to the addressed octaword, as shown:

BCBTS Bits	A_bus<5:4> value	Subblock Number
2	00	1
3	01	2
4	10	3
5	11	4

bit<1:0>

Name: Reserved

Mnemonic: None

Type: –

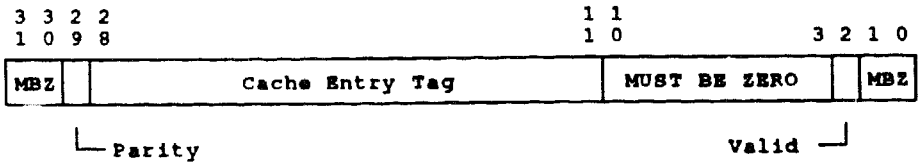
Reserved; must be zero.

Backup Cache Primary Cache 1 Tag Store Register (BCP1TS)

BCP1TS is the first half of the Backup Cache Primary Cache Tag Store. There are 256 20-bit entries in the primary cache, evenly divided between BCP1TS and BCP2TS. Each entry consists of one valid bit, an 18-bit tag, and one parity bit. The entries are arranged into rows that are four entries wide. Only one entry can be accessed at a time by an MTPR/MFPR.

ADDRESS

IPR114 (C-chip)



bits<31:30>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<29>

Name: Parity
Mnemonic: None
Type: R/W

The Parity bit contains odd parity.

bits<28:11>

Name: Cache Entry Tag
Mnemonic: None
Type: R/W

The Cache Entry Tag field contains the tag portion of the tag store entry.

bits<10:3>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<2>

Name: Valid
Mnemonic: None
Type: R/W

The Valid bit of the tag store entry.

bits<1:0>

Name: Reserved
Mnemonic: None
Type: –

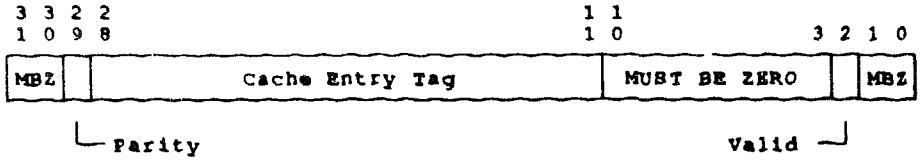
Reserved; must be zero.

Backup Cache Primary Cache 2 Tag Store Register (BCP2TS)

BCP2TS is the second half of the Backup Cache Primary Cache Tag Store. There are 256 20-bit entries in the primary cache, evenly divided between BCP1TS and BCP2TS. Each entry consists of one valid bit, an 18-bit tag, and one parity bit. The entries are arranged into rows that are four entries wide. Only one entry can be accessed at a time by an MTPR/MFPR.

ADDRESS

IPR115 (C-chip)



bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<29>

Name: Parity

Mnemonic: None

Type: R/W

The Parity bit contains odd parity.

bits<28:11>

Name: Cache Entry Tag

Mnemonic: None

Type: R/W

The Cache Entry Tag field contains the tag portion of the tag store entry.

bits<10:3>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<2>

Name: Valid
Mnemonic: None
Type: R/W

The Valid bit of the tag store entry.

bits<1:0>

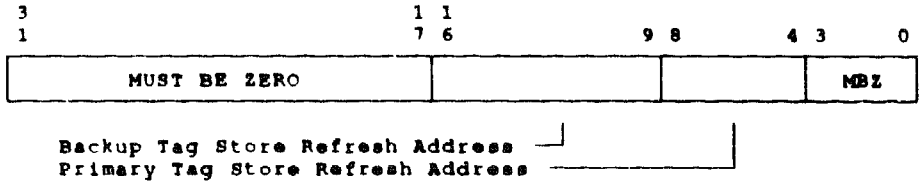
Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

Backup Cache Refresh Register (BCRFR)

The BCRFR contains the backup and primary tag store refresh addresses.

ADDRESS *IPR116 (C-chip)*



bits<31:17>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero

bits<16:9>

Name: Backup Tag Store Refresh Address

Mnemonic: BTS REFRESH ADDRESS

Type: R/W

The backup tag store refresh address corresponds to the backup tag store row index. It is incremented each time a refresh of a tag store row is done, if Enable Refresh in the BCSTS is set.

bits<8:4>

Name: Primary Tag Store Refresh Address

Mnemonic: PTS REFRESH ADDRESS

Type: R/W

The primary tag store refresh address corresponds to the primary tag store row index. It is incremented each time a refresh of a tag store row is done, if Enable Refresh in the BCSTS is set.

bits<3:0>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero

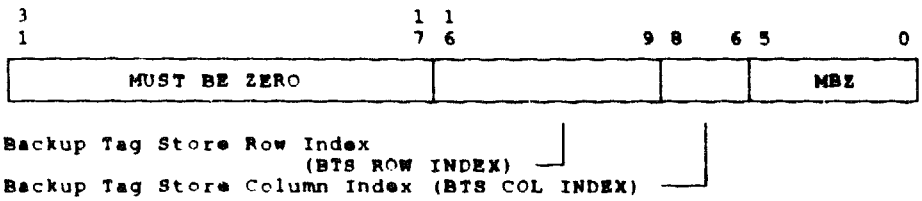
Backup Cache Index Register (BCIDX)

BCIDX accesses BCBTS, BCP1TS, and BCP2TS, the backup and primary tag stores, through IPR reads and writes. When a tag store is accessed, the bits that correspond to the respective tag store indexes are used. The entire register is read using an IPR read of BCIDX. Bits<16:4> are all written during an IPR write of BCIDX. The primary tag store uses bits <10:4>. The backup tag store uses bits <16:6>.

ADDRESS

IPR117 (C-chip)

BCIDX, w[illegible] used for backup tag store indexing:



bits<31:17>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<16:9>

Name: Backup Tag Store Row Index

Mnemonic: BTS ROW INDEX

Type: R/W

BTS ROW INDEX stores the backup tag store row index.

bits<8:6>

Name: Backup Tag Store Column Index
Mnemonic: BTS COL INDEX
Type: R/W

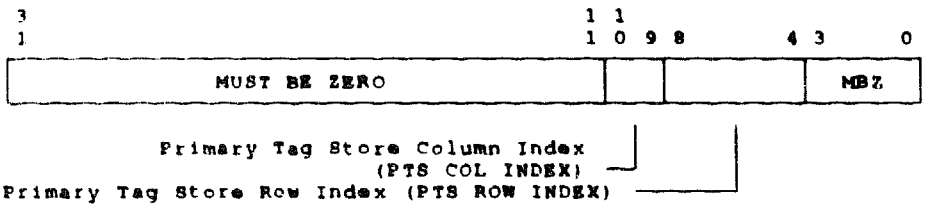
BTS COL INDEX stores the backup tag store column index.

bits<5:0>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

BCIDX, when used for primary tag store indexing:



bits<31:11>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<10:9>

Name: Primary Tag Store Column Index
Mnemonic: PTS COL INDEX
Type: –

PTS COL INDEX stores the primary tag store column index.

bits<8:4>

Name: Primary Tag Store Row Index
Mnemonic: PTS ROW INDEX
Type: R/W

PTS ROW INDEX stores the primary tag store row index.

bits<3:0>

Name: Reserved
Mnemonic: None
Type: –

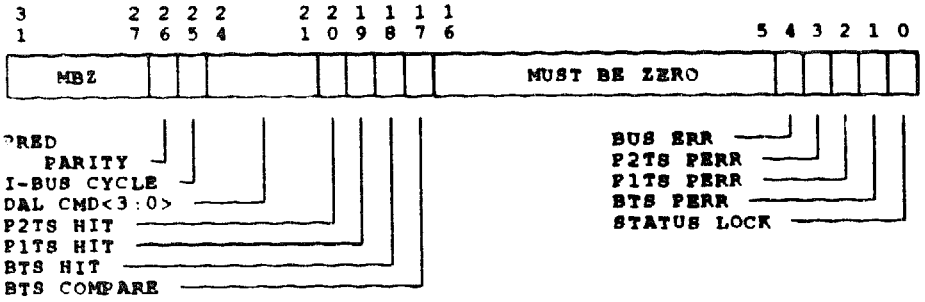
Reserved; must be zero.

Backup Cache Status Register (BCSTS)

BCSTS is loaded during every memory read or memory write DAL transaction; when the backup tag store is accessed and parity is calculated; and during DMA transactions that are recognized by the C-chip (DMA cache fills and memory writes). The register is also loaded during every microcycle that is used to service an Inval-Bus request. This register, the error address register (BCERR), and both tag stores are disabled when BCSTS<0> is set.

ADDRESS

IPR118 (C-chip)



bits<31:27>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<26>

Name: Predicted Parity

Mnemonic: PRED PARITY

Type: RO

PRED PARITY is the output of the predicted parity generator. It is loaded whenever the status latch is loaded.

bit<25>

Name: Inval-Bus Request Cycle
Mnemonic: I-BUS CYCLE
Type: RO

I-BUS CYCLE is set when BCSTS is loaded during a microcycle that was servicing an Inval-Bus request.

bits<23:21>

Name: DAL Command
Mnemonic: DAL CMD<3:0>
Type: RO

DAL CMD<3:0> is stored with the last non-IPR DAL command. It is unpredictable if I-BUS CYCLE is set.

bit<20>

Name: Primary Tag Store Hit #2
Mnemonic: PTS2 HIT
Type: RO

PTS2 HIT is the result of the hit calculation from the last access of the second half of the primary tag store.

bit<19>

Name: Primary Tag Store Hit #1
Mnemonic: PTS1 HIT
Type: RO

PTS1 HIT is the result of the hit calculation from the last access of the first half of the primary tag store.

bit<18>

Name: Backup Tag Store Hit
Mnemonic: BTS HIT
Type: RO

BTS HIT stores the result of the last backup tag store hit calculation for the last backup tag store access.

bit<17>

Name: Backup Tag Store Tag Comparison
Mnemonic: BTS COMPARE
Type: RO

BTS COMPARE is the result of the tag comparison from the last backup tag store access.

bits<16:5>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<4>

Name: Bus Error
Mnemonic: BUS ERR
Type: RO, 0

BUS ERR sets when an error occurs on the DAL that may cause corrupted cache RAM data during read miss, write, or fill transactions. It does not happen during a read to I/O space transaction. When BUS ERR sets, Status Lock also sets, disabling the backup tag store and the primary tag store copy. BUS ERR clears when Status Lock clears through an IPR write or during reset.

bit<3>

Name: Primary Tag Store Parity Error #2
Mnemonic: P2TS PERR
Type: RO, 0

P2TS PERR sets when a parity error occurs in the second half of the primary tag store. The bit is not loaded into BCSTS unless ENABLE PTS (BCCTL<2>) is set. When P2TS PERR sets, Status Lock also sets. P2TS PERR clears when the Status Lock bit clears through an IPR write or during reset.

bit<2>

Name: Primary Tag Store Parity Error #1
Mnemonic: P1TS PERR
Type: RO, 0

P1TS PERR sets when a parity error occurs in the first half of the primary tag store. The bit is not loaded into BCSTS unless ENABLE PTS (BCCTL<2>) is set. When P1TS PERR sets, Status Lock also sets. P1TS PERR clears when the Status Lock bit clears through an IPR write or during reset

bit<1>

Name: Backup Tag Store Parity Error

Mnemonic: BTS PERR

Type: RO, 0

BTS PERR sets when a parity error occurs in the backup tag store. The bit is not loaded into BCSTS unless ENABLE BTS (BCCTL<1>) is set. When BTS PERR sets, Status Lock also sets. BTS PERR clears when the Status Lock bit clears through an IPR write or during reset.

bit<0>

Name: Status Lock

Mnemonic: None

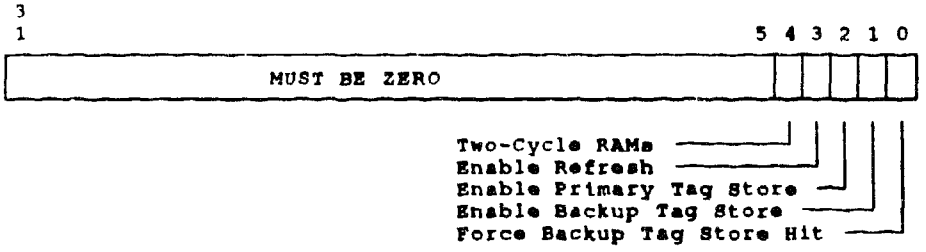
Type: R/W1C, 0

Status Lock is set by hardware when a parity error occurs in either the backup or primary tag stores or when a bus error occurs. It locks the register and the error address register against further modification and disables both tag stores. Any Inval-Bus request results when Status Lock is set. When Status Lock clears, by writing a one to IPR118 or on reset, BUS ERR, PTS PERR, P1TS PERR, and P2TS PERR (bits <4:1>) all clear.

Backup Cache Control Register (BCCTL)

Logic external to the C-chip uses BCCTL to control the C-chip. BCCTL is read using an IPR read and is written using an IPR write. Since all bits are written at once, they must contain valid data when the IPR write is issued. Three bits are also hardware-writable when the RESET L line is asserted.

ADDRESS *IPR119 (C-chip)*



bits<31:5>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<4>

Name: Two-Cycle RAMs

Mnemonic: None

Type: R/W

Two-Cycle RAMs indicates the speed of the cache RAMs enabling the C-chip to know how many microcycles are necessary to access the RAMs. Console code sets this bit to zero.

bit<3>

Name: Enable Refresh

Mnemonic: None

Type: R/W

When Enable Refresh is one, automatic refresh of the tag stores proceeds normally. That is, each time a refresh is done, the refresh counter, which indicates row address, is incremented. When Enable Refresh is zero, automatic refresh is disabled and the refresh register incrementer is disabled. This allows explicit control of the refresh register through IPR writes to that register.

CAUTION: The tag store data may be corrupted if each row of each tag store is not refreshed at least once every millisecond.

When Enable Refresh is zero, the refresh register is used instead of the index register during IPR accesses of the primary and backup tag stores, allowing the path from the refresh register to the tag stores to be tested.

bit<2>

Name: Enable Primary Tag Store

Mnemonic: ENABLE PTS

Type: R/W, 0

When ENABLE PTS is zero, the copy of the primary tag store is disabled, the PTS does not contribute to the calculation of the INV HIT L signal on an Inval-Bus request and primary cache tag store parity errors are not loaded into BCSTS and do not cause the assertion of the SERR IRQ L signal. ENABLE PTS is cleared by hardware only when the RESET L signal is asserted. ENABLE PTS must be set when the primary cache is enabled to ensure proper invalidate filtering.

While the PTS is disabled, its contents MAY CHANGE as a result of DAL operations and must be flushed through an IPR write to BCFPTS before it is reenabled to ensure correct operation. The primary cache must also be flushed.

bit<1>

Name: Enable Backup Tag Store

Mnemonic: ENABLE BTS

Type: R/W, 0

When ENABLE BTS is clear, the backup cache is disabled. When the backup cache is disabled:

- All reads produce a cache miss and no writes to the cache are done.
- The backup tag store does not contribute to the calculation of the INV HIT L signal on an Inval-Bus request (only the access of the primary cache produces INV HIT L).
- Backup cache tag store parity errors are not loaded into the BCSTS and do not cause the assertion of the SERR IRQ L signal.

ENABLE BTS is cleared by hardware only when the RESET L signal is asserted and does not clear during normal operation.

If FORCE BHIT is set and ENABLE BTS is clear, the response of the backup tag store is unpredictable.

While the backup tag store is disabled, its contents MAY CHANGE as a result of DAL operations. To ensure correct operation, it must be flushed through an IPR write of IPR121 (BCFBTS) before it is reenabled.

bit<0>

Name: Force Backup Tag Store Cache Hit

Mnemonic: FORCE BHIT

Type: R/W, 0

When FORCE BHIT is set, all non-I/O space backup tag store accesses produce a cache hit, including READ LOCK accesses. No backup cache tag store parity errors are reported. All Inval-Bus requests result in the assertion of the INV HIT L signal regardless of the contents of the backup tag store. If ENABLE BTS is clear and FORCE BTS is set, the backup tag store response is unpredictable.

If a primary tag store parity error occurs, causing Status Lock to be set, the backup cache does not disable as normal and the FORCE BHIT condition overrides the Status Lock condition. If BUS ERR sets, causing Status Lock to set, the backup cache does not disable as the FORCE BHIT condition overrides the Status Lock condition.

When the C-chip is in FORCE BHIT mode, the cache RAM data is written for each non-I/O space memory write and the cache RAM data is read on every non-I/O space memory read. The state of the backup tag store is unpredictable and it must be flushed and initialized before it is returned to the normal mode.

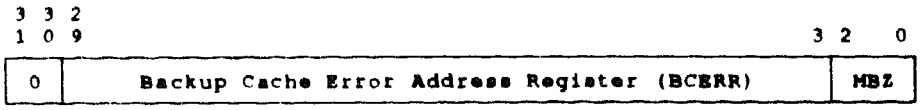
FORCE BHIT clears during reset and is only set to one by diagnostics.

Backup Cache Error Address Register (BCERR)

BCERR is a read-only register that is loaded with the address of the current transaction by hardware every time BCSTS is loaded.

ADDRESS

IPR120 (C-chip)



bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<29:3>

Name: Backup Cache Error Address Register

Mnemonic: BCERR

Type: RO

BCERR is loaded with the address of the current transaction by hardware every time BCSTS is loaded. The first error sets BCSTS<0>, Status Lock, locking BCERR against further writes until Status Lock clears.

When BCERR is loaded during an Inval-Bus transaction, bits <29> and <3> are both zeros since the Inval-Bus drives only bits <28:4>.

The address contained in BCERR when a bus error occurs is unpredictable. The DAL error may occur several cycles after the address corresponding to the transaction and BCERR may have been overwritten by an Inval-Bus transaction.

BCERR is read using an IPR read; IPR writes are ignored.

bits<2:0>

Name: Reserved

Mnemonic: None

Type: –

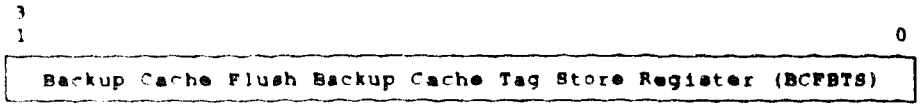
Reserved; must be zero.

Backup Cache Flush Backup Cache Tag Store Register (BCFBTS)

An IPR write of BCFBTS clears all the valid bits in the backup tag store.

ADDRESS

IPR121 (C-chip)



bits<31:0>

Name: Backup Cache Flush Backup Tag Store

Mnemonic: BCFBTS

Type: WO

An IPR write of BCFBTS clears all the valid bits in the backup tag store. The C-chip ignores the contents of the D-bus during the transaction.

Backup Cache Flush Primary Cache Tag Store Register (BCFPTS)

An IPR write of BCFPTS clears all the valid bits in the primary tag store.

ADDRESS

IPR122 (C-chip)

bits<31:0>

Name: Backup Cache Flush Primary Tag Store

Mnemonic: BCFPTS

Type: WO

An IPR write of BCFPTS clears all the valid bits in the primary tag store. The C-chip ignores the contents of the D-bus during the transaction.

Vector Interface Error Status Register (VINTSR)

VINTSR contains error status and control information for the scalar/vector interface and for the vector module.

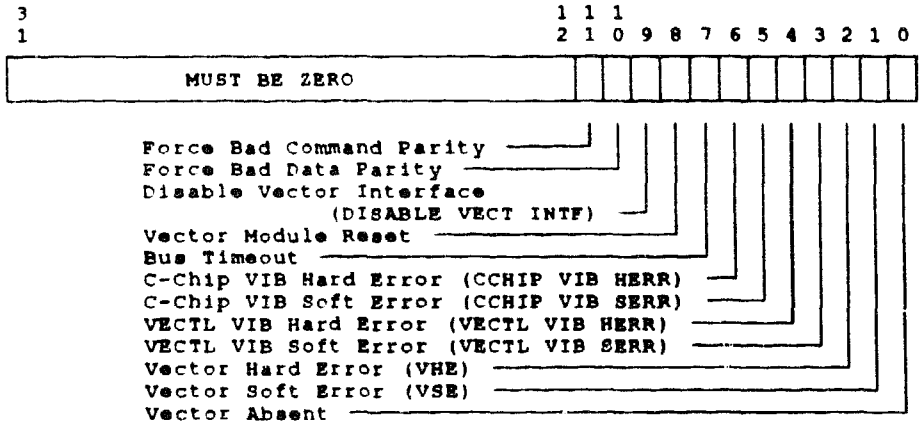
Software reads this register to determine if VECTL VIB HERR, CCHIP VIB HERR, Bus Timeout, or Vector Absent are set before accessing the vector module over the VIB. If any of these bits is set, the VIB is not functioning properly, and it is impossible to communicate with the vector module without first resetting it. If resetting is not sufficient, ACCS<Vector Present> needs to be cleared.

No VIB transactions are initiated if any of the following is set:

- Hard error bits:
  VHE, VECTL VIB HERR, CCHIP VIB HERR, or Bus Timeout
- Vector Module Reset
- Disable Vector Interface
- Vector Absent

ADDRESS

IPR123 (C-chip)



mab-p175-90

bits<31:12>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero

bit<11>

Name: Force Bad Command Parity

Mnemonic: None

Type: R/W, 0

Force Bad Command Parity, when set, causes the C-chip to generate bad command parity on all subsequent VIB operations. Clearing Force Bad Command Parity returns the C-chip to normal mode, and good parity is generated.

bit<10>

Name: Force Bad Data Parity

Mnemonic: None

Type: R/W, 0

Force Bad Data Parity, when set, causes the C-chip to generate bad data parity on all subsequent VIB operations. Clearing Force Bad Data parity returns the C-chip to normal mode, and good parity is generated.

bit<9>

Name: Disable Vector Interface

Mnemonic: DISABLE VECT INTF

Type: R/W, 1

DISABLE VECT INTF, when set, disables the vector interface functions of the C-chip. The C-chip does not respond to any IPR read or write operations directed at the vector interface (addresses 08–0B (hex)). Reads and writes of VINSTR still function normally.

It is possible that errors resulting from both outstanding VIB operations and from the assertion of either the VECT SERR L or VECT HERR L signals are reported (through the HERR IRQ L and SERR IRQ L signals) while the interface is disabled. Also, previously reported error status bits are not explicitly cleared when the interface is disabled or enabled.

The C-chip powers up with DISABLE VECT INTF set, disabling the interface. The console always clears this bit whether there is a vector module or not. Software determines the presence of the vector module before enabling the vector interface to avoid possible errors. If the vector interface is enabled at any time other than power-up, software must first reset the vector module by setting Vector Module Reset, VINTSR<8>, before enabling the vector interface.

bit<8>

Name: Vector Module Reset

Mnemonic: None

Type: R/W, 0

Vector Module Reset is set by software to reset the vector module and return it to its power-on state.

If Vector Module Reset is set and DISABLE VECT INTF is not set, any IPR reads or writes directed at the vector interface (addresses 08-0B (hex)) are terminated with the ERR L signal on the scalar DAL asserted. If the IPR operation is a read, the C-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the CPU-chip. Reads and writes of VINTSR still function normally. The C-chip deasserts the VECTOR UNIT ENABLED L signal as long as this bit is set, inhibiting further vector instructions.

While Vector Module Reset is set, the C-chip ignores any vector module errors reported through the VECT SERR L or VECT HERR L signals. As no VIB transactions are initiated while Vector Module Reset is set, these are the only types of errors possible.

Vector Module Reset is not to be set for less than 100 scalar module cycles in order to allow the vector module sufficient time to return to its power-on state. Because the VIB clocks stop when this bit is set, the VIB outputs of the C-chip (with the exception of the VECT MODULE RESET L signal) are indeterminate until two full cycles of the VIB clocks have completed. As a result, software must wait at least 100 scalar module cycles after clearing Vector Module Reset before accessing the vector module. At the end of two full VIB cycles, the C-chip deasserts the VECT AS L signal, asserts the VECT INTERFACE RESET L signal, and drives indeterminate data with good parity on the VECT DATA H and VECT DP H signals.

If Vector Module Reset is set during a pending VIB transaction, the C-chip VIB interface returns to the home state. It is only possible for this bit to be set during a pending VIB write transaction, as a read ties up the scalar DAL and prevents VINTSR from being written.

bit<7>

Name: Bus Timeout

Mnemonic: None

Type: R/W1C, 0

Bus Timeout sets when the C-chip detects that an IPR read or write to one of the vector interface IPR addresses has been terminated with the ERR L signal asserted on the DAL by a third party, indicating that a bus timeout has occurred. The C-chip asserts the HERR IRQ L signal for one cycle on the scalar DAL when Bus Timeout sets.

If Bus Timeout is set and DISABLE VECT INTF is not set, any IPR reads or writes directed at the vector interface (addresses 08–0B (hex)) are terminated with the ERR L signal on the scalar DAL asserted. If the IPR operation is a read, the C-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the CPU-chip. Reads and writes of VINTSR still function normally. The C-chip also deasserts the VECTOR UNIT ENABLED L signal and asserts the VECT INTERFACE RESET L signal as long as Bus Timeout is set, inhibiting the execution of further vector instructions.

If an instruction transfer operation caused the timeout error, the C-chip interface logic resets to the idle state and asserts the VECT INTERFACE RESET L signal, causing the VECTL chip to reset its VIB interface state.

If a read operation caused the timeout error, the C-chip drives the VIB machine check code (D BUS H<8:7> = 00 (binary)) when the read is terminated with the ERR L signal on the scalar DAL asserted, resulting in a machine check in the scalar CPU. The C-chip deasserts the VECT AS L signal to terminate the read operation on the VIB. The VECTL chip then stops driving the VECT DATA H and VECT DP H signals in the next VIB cycle, and the C-chip begins to drive these lines again in this same cycle. The C-chip interface logic resets to the idle state and asserts the VECT INTERFACE RESET L signal, causing the VECTL chip to reset its VIB interface state.

Software must reset the vector module by setting the Vector Module Reset bit, VINTSR <8>, before attempting to access the vector module after a timeout has occurred.

bit<6>

Name: C-Chip VIB Hard Error

Mnemonic: CCHIP VIB HERR

Type: R/W1C, 0

CCHIP VIB HERR sets when the C-chip detects an unrecoverable VIB error. The only unrecoverable VIB error is the second occurrence of a read data parity error during a VIB read operation. The read that caused the error is terminated with the ERR L signal on the scalar DAL asserted, and the VIB machine check code (D BUS H<8:7> = 00 (binary)) driven back as the read data. This results in a machine check in the scalar CPU. When CCHIP VIB HERR sets, the C-chip asserts the HERR IRQ L signal for one cycle on the scalar DAL.

If CCHIP VIB HERR is set and DISABLE VECT INTF is not set, any IPR reads or writes directed at the vector interface (addresses 08–0B (hex)) are terminated with the ERR L signal on the scalar DAL asserted. If the IPR operation is a read, the C-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the CPU-chip. Reads and writes of VINTSR still function normally. The C-chip also deasserts the VECTOR UNIT ENABLED L signal and asserts the VECT INTERFACE RESET L signal as long as CCHIP VIB HERR is set, inhibiting the execution of further vector instructions.

bit<5>

Name: C-Chip VIB Soft Error

Mnemonic: CCHIP VIB SERR

Type: R/W1C, 0

CCHIP VIB SERR sets when the C-chip detects a recoverable VIB error. The only recoverable VIB error is the first occurrence of a read data parity error during a VIB read operation. When CCHIP VIB SERR sets, the C-chip asserts the SERR IRQ L signal for one cycle on the scalar DAL.

bit<4>

Name: VECTL VIB Hard Error

Mnemonic: VECTL VIB HERR

Type: R/W1C, 0

VECTL VIB HERR sets when the VECTL chip detects an unrecoverable VIB error, which are unrecoverable command and write data parity errors on the VIB. The C-chip detects these by recognizing that a VIB operation has terminated in the assertion of the VECT ERR L signal. When VECTL VIB HERR sets, the C-chip asserts the HERR IRQ L signal for one cycle on the scalar DAL.

If a read operation caused the command parity error, the read is terminated with the ERR L signal on the scalar DAL asserted, and the VIB machine check code (D BUS H<8:7> = 10 (binary)) driven back as the read data, resulting in a machine check in the scalar CPU.

If VECTL VIB HERR is set and DISABLE VECT INTF is not set, any IPR reads or writes directed at the vector interface (addresses 08–0B (hex)) are terminated with the ERR L signal on the scalar DAL asserted. If the IPR operation is a read, the C-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the CPU-chip. Reads and writes of VINTSR still function normally. The C-chip also deasserts the VECTOR UNIT ENABLED L signal and asserts the VECT INTERFACE RESET L signal as long as VECTL VIB HERR is set, inhibiting the execution of further vector instructions.

bit<3>

Name: VECTL VIB Soft Error

Mnemonic: VECTL VIB SERR

Type: R/W1C, 0

VECTL VIB SERR sets when the VECTL chip detects a recoverable VIB error, which are recoverable command and write data parity errors on the VIB. The C-chip detects these cases by recognizing that a VIB operation has terminated with the VECT RTY L signal asserted.

When VECTL VIB SERR sets, the C-chip asserts the SERR IRQ L signal for one cycle on the scalar DAL. The VECT INTERFACE RESET L signal is asserted for one VIB cycle before another transaction is started on the VIB.

bit<2>

Name: Vector Hard Error

Mnemonic: VHE

Type: R/W1C, 0

VHE sets when the vector processor asserts the VECT HERR L signal as a result of detecting an unrecoverable error internal to the vector module. When VHE sets, the C-chip asserts the HERR IRQ L signal for one cycle on the scalar DAL.

If the VECT HERR L signal is asserted during a pending VIB read transaction, the read is unaffected and completes as if the VECT HERR L signal had not been asserted. Therefore, the C-chip waits until the read completes on the DAL before asserting the VECT INTERFACE RESET L signal. If the VECT HERR L signal is asserted while the C-chip has write data in its instruction buffer, the C-chip flushes the instruction buffer and returns to the home state. The VECT INTERFACE RESET L signal is asserted immediately in this case.

If the pending read operation caused the vector module internal error, the read is terminated on the VIB with the VECT RDY L and VECT EXCEP L signals asserted, in addition to reporting the error through the VECT HERR L signal. The data returned indicates a vector module machine check. The read is then terminated with the ERR L signal on the scalar DAL asserted, and the vector module machine check code (D BUS H<8:7> = 10 (binary)) driven back as the read data. This results in a machine check in the scalar CPU.

If VHE is set and DISABLE VECT INTF is not set, any IPR reads or writes directed at the vector interface (addresses 08–0B (hex)) are terminated with the ERR L signal on the scalar DAL asserted. If the IPR operation is a read, the C-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the CPU. Reads and writes of VINTSR still function normally. The C-chip also deasserts the VECTOR UNIT ENABLED L signal and asserts the VECT INTERFACE RESET L signal as long as VHE is set, inhibiting the execution of further vector instructions.

bit<1>

Name: Vector Soft Error

Mnemonic: VSE

Type: R/W1C, 0

VSE sets when the vector processor asserts the VECT SERR L signal as a result of detecting a recoverable error internal to the vector unit. When VSE sets, the C-chip asserts the SERR IRQ L signal for one cycle on the scalar DAL.

bit<0>

Name: Vector Absent

Mnemonic: None

Type: R/W1C, 0

Vector Absent sets when the C-chip detects the deassertion of the VECT PRESENT L signal on the VIB, which the C-chip samples to determine the presence of a vector module. This signal is pulled high on the scalar module and low on the vector module. If the cable is connected, the VECT PRESENT L signal will be asserted, and if it is not connected, the VECT PRESENT L signal will be deasserted.

When Vector Absent sets, the C-chip asserts the HERR IRQ L signal for one cycle on the scalar DAL. Vector Absent remains set until it is cleared either by software or by resetting the C-chip, even if the VECT PRESENT L signal becomes asserted again.

To determine if the absence of the vector module was transient or permanent, software writes a one to Vector Absent and then checks to see if it cleared. If it did clear, the error was transient, and the vector module is again present. If the bit did not clear, the vector module is still absent and should be marked as such in ACCS<Vector Present>. The HERR IRQ L signal does not assert in this case.

If Vector Absent is set and DISABLE VECT INTF is not set, any IPR reads or writes directed at the vector interface (addresses 08–0B (hex)) are terminated with the ERR L signal on the scalar DAL asserted. If the IPR operation is a read, the C-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the CPU. Reads and writes of the status register still function normally.

While Vector Absent is set, the C-chip deasserts the VECTOR UNIT ENABLED L signal to stop further issue of vector instructions, and the VECT MODULE RESET L signal is asserted to force the vector module to its reset state.

Because the VIB clocks may or may not be running when Vector Absent is set, the VIB outputs of the C-chip (with the exception of the VECT MODULE RESET L signal) are indeterminate until two full cycles of the VIB clocks have been completed with the VECT PRESENT L signal asserted. As a result, software waits at least 100 scalar module cycles after clearing Vector Absent before accessing the vector module. At the end of two full VIB clock cycles with the VECT PRESENT L signal asserted, the C-chip deasserts the VECT AS L signal, asserts the VECT INTERFACE RESET L signal, and drives indeterminate data with good parity on the VECT DATA H and VECT DP H signals.

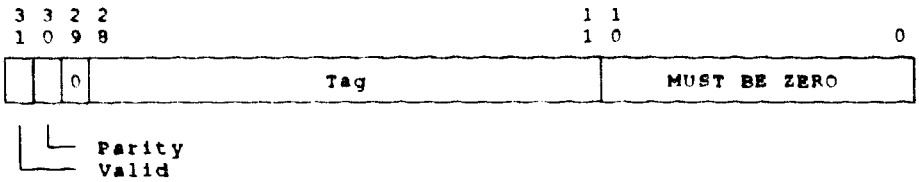
If Vector Absent is set during a pending VIB transaction, the C-chip VIB interface returns to the home state. If the pending transaction is a read, it is terminated with the ERR L signal on the scalar DAL asserted, and the VIB machine check code (D BUS H<8:7> = 00 (binary)) is returned to the CPU.

Primary Cache Tag Array Register (PCTAG)

PCTAG provides the mechanism for reading and writing the tag array of the primary cache. To read the tag array, the desired index is first written into PCIDX with an MTPR (Move To Processor Register) instruction. PCTAG is then read with an MFPR (Move From Processor Register) instruction. To write the tag array, the desired index is written into PCIDX with an MTPR. PCTAG is then written with an MTPR.

ADDRESS

IPR124 (CPU-chip)



bit<31>

Name: Valid
Mnemonic: None
Type: R/W

Valid bit for tag.

bit<30>

Name: Parity
Mnemonic: None
Type: R/W

Parity bit (odd parity) for tag.

bit<29>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero

bits<28:11>

Name: Tag
Mnemonic: None
Type: R/W

The primary cache tag contents.

bits<10:0>

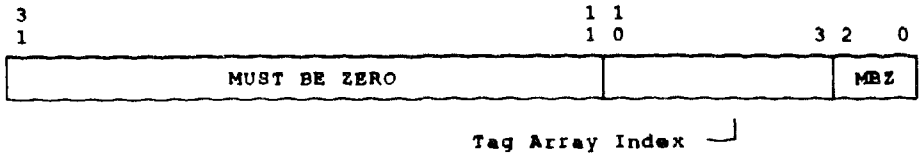
Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero

Primary Cache Index Register (PCIDX)

PCIDX must be written with the desired index of the tag entry before an IPR write to PCTAG is performed.

ADDRESS *IPR125 (CPU-chip)*



bits<31:11>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<10:3>

Name: Tag Array Index
Mnemonic: None
Type: R/W

The Tag Array Index field must be written with the desired index of the tag entry before an IPR read or write to PCTAG is performed.

bits<2:0>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

Primary Cache Error Address Register (PCERR)

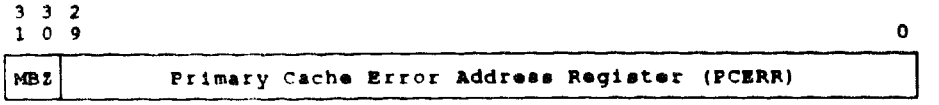
PCERR latches and holds the physical address of an error when PCSTS<7>, Trap1, sets for read commands.

PCERR is also used, during diagnostic testing, to look into the Refresh Counter and Refresh Timer registers.

ADDRESS

IPR126 (CPU-chip)

PCERR, when Trap1 is set:



bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<29:0>

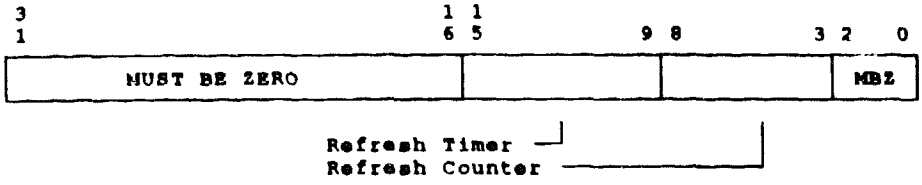
Name: Primary Cache Error Address Register

Mnemonic: PCERR

Type: RO

PCERR latches and holds the physical address of an error when PCSTS<7>, Trap1, sets for read commands. Since write errors are asynchronous to the instruction pipeline, the address latched for write commands is not the address of the error and the write error address is not available.

PCERR, when Trap1 is not set:



bits<31:16>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<15:9>

Name: Refresh Timer

Mnemonic: None

Type: R/W

PCERR is used during diagnostic testing to look into the Refresh Counter and Refresh Timer registers. An IPR write of the PCERR writes data bits<15:9> into the Refresh Timer. If Trap1 is not set, an IPR read of the PCERR returns the current value of the Refresh Timer.

bits<8:3>

Name: Refresh Counter

Mnemonic: None

Type: R/W

PCERR is used during diagnostic testing to look into the Refresh Counter and Refresh Timer registers. An IPR write of the PCERR writes bits <8:3> into the Refresh Counter. If Trap1 is not set, an IPR read of the PCERR returns the current value of the Refresh Counter.

bits<2:0>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

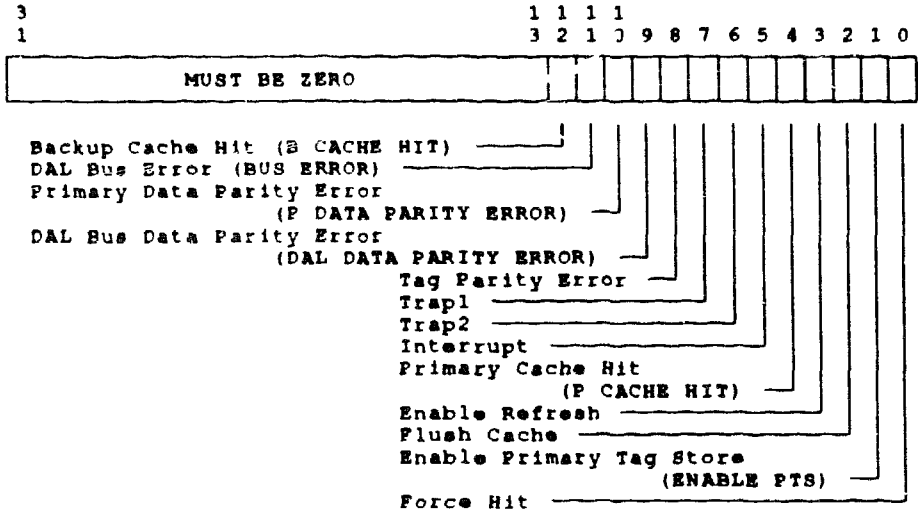
NOTE: When PCSTS<3>, Enable Refresh, is set, the Refresh Counter increments for every NOP and the Refresh Timer increments for every cycle that is not a NOP or an IPR write to PCERR. The internal latency involved with moves to/from processor instructions causes the count values of the Refresh Counter and Timer to change. To allow the count values of the Refresh Counter and Timer to remain unchanged, Enable Refresh must be clear.

Primary Cache Status Register (PCSTS)

PCSTS is used to control the primary cache mode of operation, flushing the cache, and maintaining error information.

ADDRESS

IPR127 (CPU-chip)



bits<31:13>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<12>

Name: Backup Cache Hit

Mnemonic: B CACHE HIT

Type: R0, 0

B CACHE HIT is updated on every primary cache reference unless Interrupt, Trap1, or Trap2 is already set. B CACHE HIT indicates that the condition that caused PCSTS to lock was a reference that hit in the backup cache. For DAL data parity errors, B CACHE HIT is set to indicate that the backup cache is the source of the error or is clear to indicate that the memory subsystem is the source.

bit<11>

Name: DAL Bus Error

Mnemonic: BUS ERROR

Type: RO, 0

BUS ERROR is updated on every primary cache reference unless Interrupt, Trap1, or Trap2 is already set. BUS ERROR sets when a DAL read, write, or clear write buffer command terminates by ERR L and is otherwise clear. If the DAL command was an I-stream read, Interrupt is also set, and the error is reported as an IPL 1A (hex) soft error interrupt. If the DAL command was a D-stream read, write, or clear write buffer, Trap 1 or Trap2 is also set, and a microtrap error is reported with a subsequent machine check.

bit<10>

Name: Primary Data Parity Error

Mnemonic: P DATA PARITY ERROR

Type: RO, 0

P DATA PARITY ERROR is updated on every primary cache reference unless Interrupt, Trap1, or Trap2 is already set. This bit is set when a read hits in the primary cache and the requested data has a parity error. It is clear otherwise. If the DAL command was an I-stream read, Interrupt is also set and the error is reported as an IPL 1A (hex) soft error interrupt. If the DAL command was a D-stream read, write, or clear write buffer, Trap1 or Trap2 is also set, and a microtrap error is reported with a subsequent machine check.

bit<9>

Name: DAL Bus Data Parity Error

Mnemonic: DAL DATA PARITY ERROR

Type: RO, 0

DAL DATA PARITY ERROR is updated on every primary cache reference unless Interrupt, Trap1, or Trap2 is already set. This bit is set when the data returned in response to a non-I/O space DAL read has a parity error. It is clear otherwise. If the error is detected on the non-requested longword of a D-stream read, or on either longword of an I-stream read, Interrupt is also set and the error is reported as an IPL 1A (hex) soft error interrupt. If the error is detected on the requested longword of a D-stream read, Trap1 or Trap2 is also set, and a microtrap error is reported with a subsequent machine check.

bit<8>

Name: Tag Parity Error

Mnemonic: None

Type: RO, 0

Tag Parity Error is updated on every primary cache reference unless Interrupt, Trap1, or Trap2 is already set. This bit is set if a tag parity error is detected during a read, write, or invalidate reference. It is clear otherwise. If Tag Parity Error is set, Interrupt is also set and the error is reported as an IPL 1A (hex) soft error interrupt. If the reference was a D-stream read that hit, Trap1 or Trap2 is also set, and a microtrap error is reported with a subsequent machine check.

bit<7>

Name: Trap1

Mnemonic: None

Type: R/W1C, 0

Trap1 sets when a detected error is to be reported as a microtrap and subsequent machine check. PCSTS<12:8> and PCERR are latched unless Trap1 is already set. If Trap1 sets, the primary cache is automatically disabled without changing ENABLE PTS, PCSTS<1>. Trap1 clears during an IPR write to PCSTS.

NOTE: If Interrupt, PCSTS<5>, is already set when another error sets Trap1, the information the error reported as an interrupt is lost. Because errors reported as interrupts are nonfatal, software assumes that PCSTS<11:8> are all set and performs the error recovery procedures for each of the four possible errors.

bit<6>

Name: Trap2
Mnemonic: None
Type: R/W1C, 0

Trap2 sets when a detected error is to be reported as a microtrap and subsequent machine check and Trap1 is already set. If Trap2 sets, a nested, FATAL error has occurred, and PCSTS<12:8> and PCERR contain information about the first error. If Trap2 sets, the primary cache is automatically disabled without changing ENABLE PTS, PCSTS<1>. Trap2 clears during an IPR write to PCSTS.

bit<5>

Name: Interrupt
Mnemonic: None
Type: R/W1C, 0

Interrupt sets when an error is detected that is to be reported as a soft error interrupt at IPL 1A (hex). PCSTS<12:8> are latched unless Interrupt or Trap1 is already set and remain latched until Interrupt is cleared or another error sets Trap1. If Trap1 sets, the primary cache automatically disables, although PCSTS<1>, ENABLE PTS, is not changed. Interrupt clears during an IPR write to PCSTS.

bit<4>

Name: Primary Cache Hit
Mnemonic: P CACHE HIT
Type: RO, 0

P CACHE HIT is the latched value of the primary cache comparator output. It is updated for all D-stream reads, writes, or invalidate cycles and is used only during diagnostic testing of the primary cache hit logic.

bit<3>

Name: Enable Refresh
Mnemonic: None
Type: R/W, 0

When Enable Refresh is set, automatic refresh of the primary cache is enabled and the refresh counter increments. When Enable Refresh clears, refresh is disabled, the refresh counter does not increment, and the refresh timer logic is disabled. Enable Refresh is set for normal primary cache operations.

bit<2>

Name: Flush Cache

Mnemonic: None

Type: R/W1C

Flush Cache clears all valid bits in the primary cache tag array when it is written with a one. Hardware then clears Flush Cache in the next cycle, so it is always read as a zero.

NOTE: The state of the primary cache is unpredictable if PCSTS<1>, ENABLE PTS, is zero. Therefore, the primary cache must be flushed before it is enabled, with either a separate IPR write before ENABLE PTS is set or by setting both Flush Cache and ENABLE PTS with the same IPR write.

bit<1>

Name: Enable Primary Tag Store

Mnemonic: ENABLE PTS

Type: R/W, 0

ENABLE PTS controls normal operation of the primary cache. When ENABLE PTS is a one, both I-stream and D-stream references are cached and primary cache tag and data parity error are reported. I/O references are never cached. When ENABLE PTS is a zero, all references (read, write, and invalidate) result in a miss.

bit<0>

Name: Force Hit

Mnemonic: None

Type: R/W, 0

Force Hit is used only for diagnostic testing and must be clear during normal operation. When set, the primary cache forces a hit for all memory references. Memory write requests still go to the external memory. Since I/O references are not cached, they are not affected by Force Hit.

When Force Hit is set, primary cache tag parity error reporting associated with D-stream reads, writes, and invalidates and primary cache data parity errors associated with D-stream reads are disabled. DAL errors (parity errors associated with the data present on the DAL or bus errors) are not affected. Force Hit is not used to satisfy I-stream reads, as primary cache tag or data parity errors detected during I-stream reads cause a loop. Force Hit can be used to initialize the primary cache data array. Force Hit is cleared on chip reset.

3.8.2 XMI Registers

In addition to the internal processor registers, the KA64A CPU module contains registers in XMI private space and XMI required registers in XMI nodespace. These registers are listed in Table 3–15 and Table 3–16.

The KA64A CPU module's XMI registers have the following characteristics:

- The mask bits are ignored on writes to the KA64A CPU module's control and status registers. The CPU always performs a full longword write.
- Interlocks are not supported. Interlock Read and Unlock Write Mask transactions are treated as Read and Write Mask transactions, respectively.

The KA64A CPU module's XMI required registers, but not its private space registers, have the following characteristic:

- The XMI responder queue is only one deep so the KA64A CPU module will NO ACK subsequent CSR references until the read data for the queued CSR read has been returned.

Table 3–15 KA64A CPU Module Registers in XMI Private Space

Register	Mnemonic	Address	Location
CREG Write Enable	CREGWE	2000 0000	RSSC
Console ROM (halt protected)		2004 0000 – 2007 FFFF	Console ROM
System Type	SYS TYPE	2004 0004	Console ROM
Console EEPROM (halt protected)		2008 0000 – 2008 7FFF	Console EEPROM
Console ROM (not halt protected)		200C 0000 – 200F FFFF	Console ROM
Console EEPROM (not halt protected)		2010 0000 – 2010 7FFF	Console EEPROM
RSSC Base Address	SSCBAR	2014 0000	RSSC
RSSC Configuration	SSCCNR	2014 0010	RSSC
RSSC Bus Timeout Control	SSCBTR	2014 0020	RSSC
RSSC Output Port	OPORT	2014 0030	RSSC
RSSC Input Port	IPORT	2014 0040	RSSC
CREG Base Address	CRBADR	2014 0130	RSSC
CREG Address Decode Mask	CRADMR	2014 0134	RSSC
EEPROM Base Address	EEBADR	2014 0140	RSSC
EEPROM Address Decode Mask	EEADMR	2014 0144	RSSC
Timer 0 Control	TCR0	2014 0160	RSSC
Timer 0 Interval	TIR0	2014 0164	RSSC
Timer 0 Next Interval	TNIR0	2014 0168	RSSC
Timer 0 Interrupt Vector	TIVR0	2014 016C	RSSC
Timer 1 Control	TCR1	2014 0170	RSSC
Timer 1 Interval	TIR1	2014 0174	RSSC
Timer 1 Next Interval	TNIR1	2014 0178	RSSC
Timer 1 Interrupt Vector	TIVR1	2014 017C	RSSC
RSSC Interval Counter	SSCICR	2014 01F8	RSSC
RSSC Internal RAM		2014 0400 – 2014 07FF	RSSC
IP IVINTR Generation	IPIVINTRGEN	2101 0000 – 2101 FFFF	XCA
WE IVINTR Generation	WEIVINTRGEN	2102 0000 – 2102 FFFF	XCA

Table 3–16 XMI Registers for the KA64A CPU Module

Register	Mnemonic	Address	Location
XMI Device	XDEV	BB[1] + 0000 0000	XCA
XMI Bus Error	XBER	BB + 0000 0004	XCA
XMI Failing Address	XFADR	BB + 0000 0008	XCA
XMI General Purpose	XGPR	BB + 0000 000C	XCA
REXMI Control and Status	RCSR	BB + 0000 0010	XCA

[1]BB = base address of a node, which is the address of the first location in nodespace.

Control Register 0 (CREG0)

CREG0 controls some KA64A CPU module functions.

Writing to CREG0 is a multistep process:

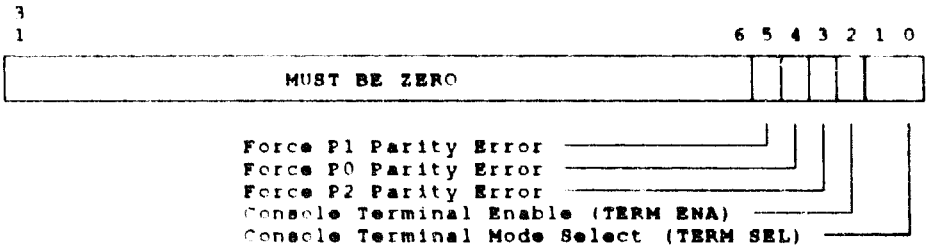
1 Write OPORT<9:2> (CREG DATA) with the data to be written to CREG0.

2 Write OPORT<0> (CREG0 SEL) with a one and OPORT<1> (CREG1 SEL) with a zero.

3 Write anything to CREGWE. The data is ignored and may be of any value.

Steps 1 and 2 provide the data to be written and give CREG0 as the destination. Step 3 transfers OPORT<CREG DATA> to CREG0.

ADDRESS *None (Virtual Register)*



bits<31:6>

Name: Reserved

Mnemonic: None

Type: WO

Reserved; must be zero.

bit<5>

Name: Force P1 Parity Error

Mnemonic: None

Type: WO, 0

Force P1 Parity Error controls bad parity generation of the XMI P1 parity bit, which protects the upper 32 data bits on the XMI bus. If Force P1 Parity Error is a one, the P1 parity bit is inverted for every REXMI-generated transaction. When Force P1 Parity Error is a zero, the P1 parity bit is correctly generated. This bit is set only during diagnostic testing.

bit<4>

Name: Force P0 Parity Error

Mnemonic: None

Type: WO, 0

Force P0 Parity Error controls bad parity generation of the XMI P0 parity bit, which protects the lower 32 data bits on the XMI bus. If Force P0 Parity Error is a one, the P0 parity bit is inverted for every REXMI-generated transaction. When Force P0 Parity Error is a zero, the P0 parity bit is correctly generated. This bit is set only during diagnostic testing.

bit<3>

Name: Force P2 Parity Error

Mnemonic: None

Type: WO, 0

Force P2 Parity Error controls bad parity generation of the XMI P2 parity bit, which protects the function and ID fields on the XMI bus. If Force P2 Parity Error is a one, the P2 parity bit is inverted for every REXMI-generated transaction. When Force P2 Parity Error is a zero, the P2 parity bit is correctly generated. This bit is set only during diagnostic testing.

bit<2>

Name: Console Terminal Enable

Mnemonic: TERM ENA

Type: WO, 0

TERM ENA enables the KA64A CPU module to drive the system console line on the XMI backplane. If TERM ENA is a one, console output is transmitted to both the auxiliary console and the system console lines on the XMI backplane. If TERM ENA is a zero, console output is transmitted only to the auxiliary console line.

bits<1:0>

Name: Console Terminal Mode Select

Mnemonic: TERM SEL

Type: WO, 0

TERM SEL selects the console terminal mode as shown below:

<1:0>	Mode	Description
00	Auxiliary Console	The auxiliary console line is connected to the RSSC console terminal input.
01	System Console	The XMI backplane console line is connected to the RSSC console terminal input.
10	Auxiliary Console Loopback	The auxiliary console output is connected back to the RSSC console terminal input.
11	System Console Loopback	The XMI console output is connected back to the RSSC console terminal input. If TERM ENA is zero, the transmitted character is not transmitted on the XMI backplane console line.

Control Register 1 (CREG1)

CREG1 drives eight module LEDs.

Writing to CREG1 is a multistep process:

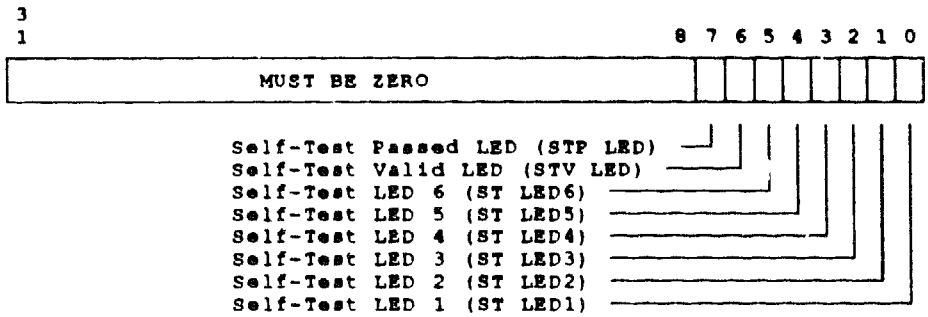
1 Write OPORT<9:2> (CREG DATA) with the data to be written to CREG1.

2 Write OPORT<0> (CREG1 SEL) with a one and OPORT<1> (CREG0 SEL) with a zero.

3 Write anything to CREGWE. The data is ignored and may be of any value.

Steps 1 and 2 provide the data to be written and give CREG1 as the destination. Step 3 transfers OPORT<CREG DATA> to CREG1.

ADDRESS *None (Virtual Register)*



bits<31:8>

Name: Reserved

Mnemonic: None

Type: WO

Reserved; must be zero.

bit<7>

Name: Self-Test Passed LED

Mnemonic: STP LED

Type: WO, 0

STP LED drives the Self-Test Passed module LED and indicates that self-test was successful. STP LED contains a one to light the LED.

bit<6>

Name: Self-Test Valid LED

Mnemonic: STV LED

Type: WO, 0

STV LED validates the state of ST LED6 through ST LED1. It sets very early in the console initialization and indicates that the power-up sequence was sufficient to start the console code and allow the console to write CREG1.

bits<5:0>

Name: Self-Test LED 6 through Self-Test LED 1

Mnemonic: ST LED*n*

Type: WO, 0

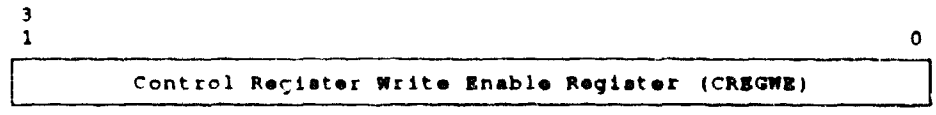
These six bits drive the six module LEDs that indicate the current state of the self-test or extended test. Writing a one to these bits lights the corresponding LED.

Control Register Write Enable Register (CREGWE)

CREGWE is used, together with OPORT, to write to CREG0 and CREG1.

ADDRESS

2000 0000 (RSSC)



bits<31:0>

Name: Control Register Write Enable

Mnemonic: CREGWE

Type: WO, 0

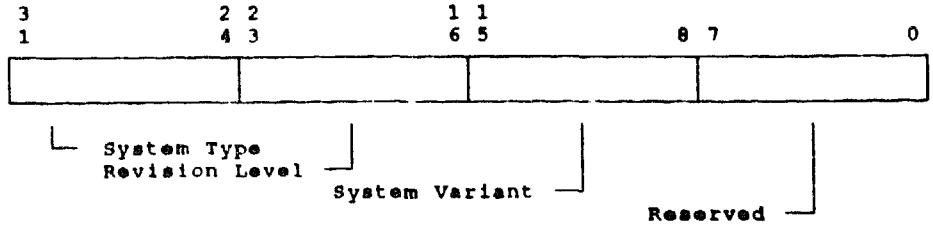
CREGWE enables writes to Control Register 0 and Control Register 1 (CREG0 and CREG1). The data that is in OPORT<9:2> (CREG DATA) is sent to the selected Control Register (OPORT<0>, OPORT<1>) whenever anything is written to CREGWE.

System Type Register (SYS TYPE)

SYS TYPE specifies the system type, its console revision level, and licensing infomation. It can only be accessed locally. Other devices on the XMI determine the nature of a node by reading its XMI Device Register (XDEV).

ADDRESS

2004 0004 (Console ROM)



bits<31:24>

Name: System Type
Mnemonic: None
Type: RO

This field is always 02 (hex), indicating the type of system.

bits<23:16>

Name: Revision Level
Mnemonic: None
Type: RO

Revision level indicates the revision number of the console software. The first revision level is 10 (hex).

bits<15:8>

Name: System Variant
Mnemonic: None
Type: RO

System Variant distinguishes the variants of similar systems. The first KA64A CPU module variant level is 01 (hex).

bits<7:0>

Name: Reserved

Mnemonic: None

Type: RO

This field is reserved for DIGITAL use only.

RSSC Base Address Register (SSCBAR)

SSCBAR controls the base address of a 2-Kbyte block of control and status registers and RAM that is part of the RSSC. Hardware reset initializes SSCBAR to its address, 2014 0000 (hex).

ADDRESS

2014 0000 (RSSC)

bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<29>

Name: Reserved

Mnemonic: None

Type: –, 1

Reserved; must be one.

bits<28:11>

Name: RSSC Base Address

Mnemonic: SSCBA

Type: R/W

SSCBAR controls the base address of the 2-Kbyte block and is set to 2014 0000 (hex) by console code during processor initialization.

bits<10:0>

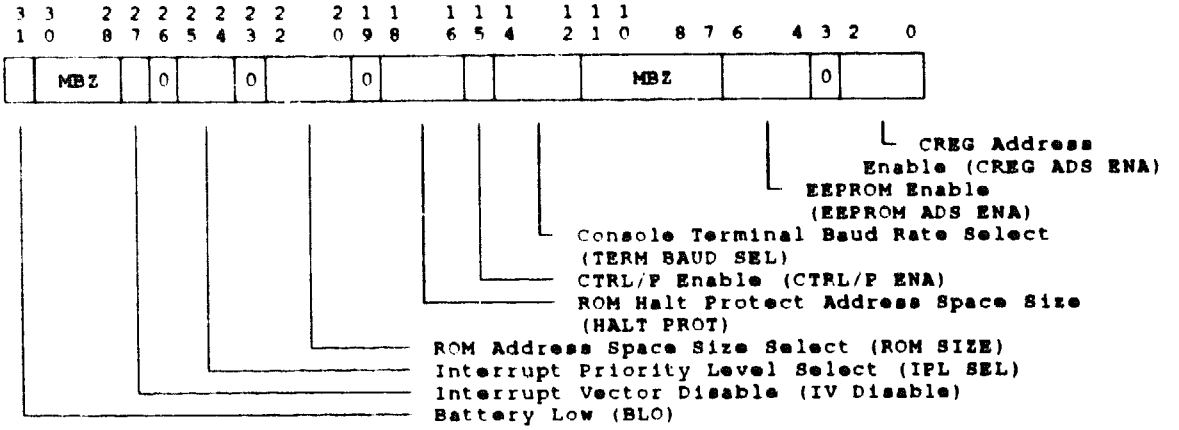
Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved: must be zero.

RSSC Configuration Register (SSCCNR)

SSCCNR provides configuration information for RSSC-controlled features.

ADDRESS *2014 0010 (RSSC)*



bit<31>

Name: Battery Low

Mnemonic: BLO

Type: R/W1C

BLO is set if the TOY clock battery voltage goes below threshold while the module is powered down. If set after power-up, the TODR register is cleared.

bits<30:28>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bit<27>

Name: Interrupt Vector Disable

Mnemonic: IV Disable

Type: R/W, 0

When IV Disable is set, the RSSC is disabled from returning an interrupt vector when the CPU-chip sends a read interrupt vector transaction

bit<26>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<25:24>

Name: Interrupt Priority Level Select
Mnemonic: IPL SEL
Type: R/W, 0

IPL SEL specifies the IPL level of interrupt acknowledge cycles that the console serial line and programmable timers respond to. This field is set to 01 (IPL 15) by console code for normal operation.

bit<23>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<22:20>

Name: ROM Address Space Size Select
Mnemonic: ROM SIZE
Type: R/W, 0

ROM SIZE controls the size of the range of addresses to which the ROM responds, as shown. ROM SIZE is always 111, yielding an address range of 1 Mbyte (2004 0000 to 2013 FFFF).

<22:20>	Size (Kbytes)	Address Range (hex)
000	8	2004 0000 – 2004 1FFF
001	16	2004 0000 – 2004 3FFF
010	32	2004 0000 – 2004 7FFF
011	64	2004 0000 – 2004 FFFF
100	128	2004 0000 – 2005 FFFF
101	256	2004 0000 – 2007 FFFF
110	512	2004 0000 – 200B FFFF
111	1024	2004 0000 – 2013 FFFF

bit<19>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<18:16>

Name: ROM Halt Protect Address Space Size Select

Mnemonic: HALT PROT

Type: R/W, 0

HALT PROT selects the size of the halt-protected ROM/EEPROM address space as shown below. If the last instruction fetch seen by the RSSC was within the address range specified by this field, console halts are disabled, allowing console code to disable nested console entries. To permit the ROM and EEPROM to be double-mapped in the address space, HALT PROT is set to 110 (binary), 512 Kbytes, by console code during processor initialization (before self-test).

<18:16>	Size (Kbytes)	Address Range (hex)
000	8	2004 0000 – 2004 1FFF
001	16	2004 0000 – 2004 3FFF
010	32	2004 0000 – 2004 7FFF
011	64	2004 0000 – 2004 FFFF
100	128	2004 0000 – 2005 FFFF
101	256	2004 0000 – 2007 FFFF
110	512	2004 0000 – 200B FFFF
111		None

bit<15>

Name: CTRL/P Enable

Mnemonic: CTRL/P ENA

Type: R/W, 0

If CPU halts are enabled, (XMI CON SECURE deasserted), CTRL/P ENA is used to select either CTRL/P or BREAK to cause the halt.

When CTRL/P ENA is set, it causes the CPU to be halted, if halts are enabled, when CTRL/P is typed at the console. When CTRL/P ENA is clear, it causes the CPU to be halted, if halts are enabled, when BREAK is typed at the console. CTRL/P ENA is set to one by console code.

bits<14:12>

Name: Console Terminal Baud Rate Select

Mnemonic: TERM BAUD SEL

Type: R/W, 0

TERM BAUD SEL uses the following codes to select the console baud rate for both the system console and the auxiliary console lines:

<14:12>	Baud Rate
000	300
001	600
010	1200
011	2400
100	4800
101	9600
110	19200
111	38400

The console program sets TERM BAUD SEL to 010 (1200 baud) during initialization. The baud rate may be changed by the user.

NOTE: The RSSC baud clock runs about 1.75% fast.

bit<11:7>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<6:4>

Name: EEPROM Address Enable

Mnemonic: EEPROM ADS ENA

Type: R/W, 0

EEPROM ADS ENA configures the programmable address strobe used to enable writes to the EEPROM, as shown below. EEPROM ADS ENA is set to 000 (binary) by console code during processor initialization. When set to 101 (binary), updates to the EEPROM are enabled.

Bit	Operation
6	Enable RDY termination of DAL transaction
5	Enable response to DAL read commands
4	Enable response to DAL write commands

bit<3>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<2:0>

Name: CREG Address Enable

Mnemonic: CREG ADS ENA

Type: R/W, 0

CREG ADS ENA configures the programmable address strobe used to enable writes to CREG0 and CREG1 by controlling the address strobe as follows:

Bit	Operation
2	Enable RDY termination of DAL transaction
1	Enable response to DAL read commands
0	Enable response to DAL write commands

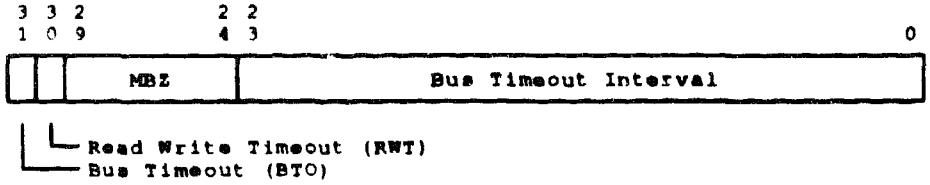
CREG ADS ENA is set to 101 by the console program during initialization

RSSC Bus Timeout Control Register (SSCBTR)

SSCBTR contains the timeout value used to ter minate DAL bus transactions. When a DAL transaction is started, the RSSC starts a timer that increments every microsecond until the transaction completes. If the time reaches the value contained in Bus Timeout Interval, the RSSC asserts the ERR L signal to terminate the transaction, causing a machine check and preventing system hangs.

ADDRESS

2014 0020 (RSSC)



bit<31>

Name: Bus Timeout

Mnemonic: BTO

Type: R/W1C, 0

BTO sets when any CPU transaction times out and terminates with the ERR L signal asserted. If a CPU read, write, or clear write buffer command is timed out, both BTO and RWT set. For IPR reads and writes and for read interrupt vector commands, only BTO sets.

bit<30>

Name: Read Write Timeout

Mnemonic: RWT

Type: R/W1C, 0

RWT sets when a CPU read, write, or clear write buffer command times out and terminates with the ERR L signal asserted. RWT is examined by error recovery software to determine if the error termination was due to a problem with the RSSC or with the REXMI.

bits<29:24>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<23:0>

Name: Bus Timeout Interval

Mnemonic: None

Type: R/W, 0

The Bus Timeout Interval field supplies the timeout period in 1 microsecond units, with a range of 1 (hex) to **FFFFFF** (hex). This corresponds to a timeout range of 1 microsecond to 16.77 seconds. The timeout function is disabled if the field is zero.

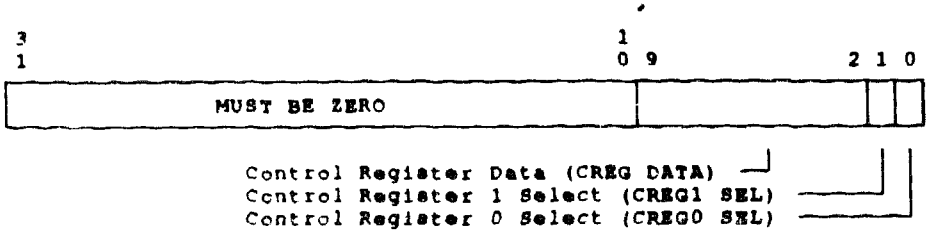
The timeout value is 17700 (hex), corresponding to 96.0 milliseconds.

RSSC Output Port Register (OPORT)

OPORT selects the desired control register and writes it with data.

ADDRESS

2014 0030 (RSSC)



bits<31:10>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<9:2>

Name: Control Register Data

Mnemonic: CREG DATA

Type: R/W, 0

CREG DATA supplies one byte of data to be transferred to the selected control register.

bit<1>

Name: Control Register 1 Select

Mnemonic: CREG1 SEL

Type: R/W, 0

When set, CREG1 SEL selects Control Register 1 as the target to write the data in CREG DATA.

bit<0>

Name: Control Register 0 Select

Mnemonic: CREG0 SEL

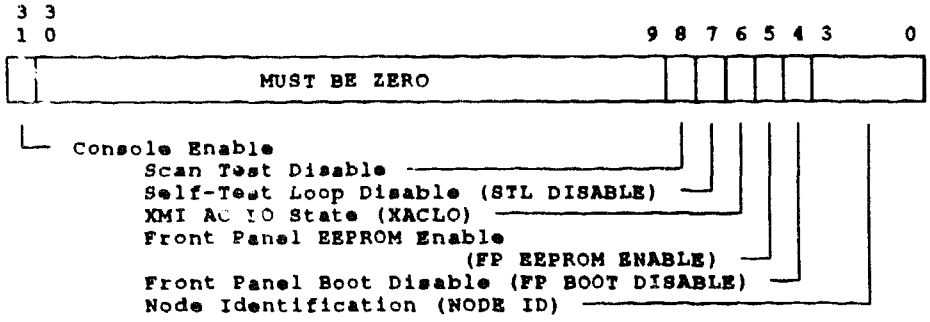
Type: R/W, 0

When set, CREG0 SEL selects Control Register 0 as the target to write the data in CREG DATA.

RSSC Input Port Register (IPORT)

IPORT gives KA64A CPU module state information.

ADDRESS *2014 0040 (RSSC)*



bit<31>

Name: Console Enable

Mnemonic: None

Type: RO

Console Enable indicates the state of the control panel upper key switch, as taken from the XMI CON SECURE L signal. Console Enable is zero when console halts are disabled and is one when console halts are enabled, that is, when the upper key switch is in the Enable position.

bits<30:9>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<8>

Name: Scan Test Disable

Mnemonic: SCAN TEST DISABLE

Type: RO, 1

SCAN TEST DISABLE indicates the state of the IO SCAN TEST L module's I/O pin. A zero indicates that the scan test is enabled, and a one, the default value, indicates that the scan test is disabled.

bit<7>

Name: Self-Test Loop Disable

Mnemonic: STL DISABLE

Type: RO, 1

STL DISABLE indicates the state of the IO SELF TEST LOOP L module's I/O pin. A zero indicates that the console loops on self-test, and a one, the default value, indicates that the console performs self-test only once.

bit<6>

Name: XMI AC LO

Mnemonic: XACLO

Type: RO

XACLO shows the state of the XMI AC LO L signal. A zero indicates that XMI AC LO L is asserted, and a one indicates that the line is deasserted. The console does not attempt to reference memory until XACLO is a one.

bit<5>

Name: Front Panel EEPROM Enable

Mnemonic FP EEPROM ENABLE

Type: RO

FP EEPROM ENABLE shows the state of the control (front) panel lower key switch as a reflection of the XMI UPDATE EN H signal. When FP EEPROM ENABLE is a zero, the EEPROM is disabled; a one indicates that the EEPROM is enabled, that is, the lower key switch is in the Update position.

bit<4>

Name: Front Panel Boot Disable

Mnemonic: FP BOOT DISABLE

Type: RO

FP BOOT DISABLE indicates the state of the control (front) panel lower key switch. The input to the bit is the XMI BOOT EN L signal. A zero indicates that booting is enabled, that is, the lower key switch is in the Auto Start position, and a one indicates that booting is disabled.

bits<3:0>

Name: Node Identification

Mnemonic: NODE ID

Type: RO

NODE ID contains the node identification of the XMI backplane slot.

Control Register Base Address Register (CRBADR)

CRBADR supplies the base address for Control Register 0 and Control Register 1 write enables.

ADDRESS

2014 0130 (RSSC)

bits<31:30>

Name: Reserved

Mnemonic: None

Type: –, 0

Reserved; must be zero.

bits<29:2>

Name: Control Register Base Address

Mnemonic: CRBAD

Type: R/W, 0

CRBAD supplies the base address for Control Register 0 and Control Register 1 write enables. CRBADR is loaded with 2000 0000 (hex) during normal operation.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: –, 0

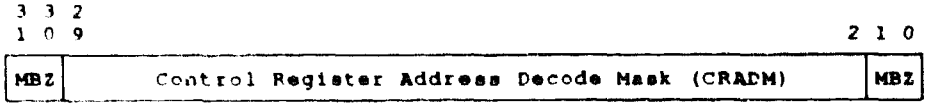
Reserved; must be zero.

Control Register Address Decode Mask Register (CRADMR)

CRADMR supplies the bit mask that selects the address extent of the Control Register address decode.

ADDRESS

2014 0134 (RSSC)



bits<31:30>

Name: Reserved

Mnemonic: None

Type: –, 0

Reserved; must be zero.

bits<29:2>

Name: Control Register Address Decode Mask

Mnemonic: CRADM

Type: R/W, 0

CRADM supplies the bit mask that selects the address extent of the Control Register address decode. Ones in CRADM indicate those bits that are to be ignored during the address compare. CRADM is loaded with zero during normal operation as there is only one address being decoded by this strobe.

bits<1:0>

Name: Reserved

Mnemonic: None

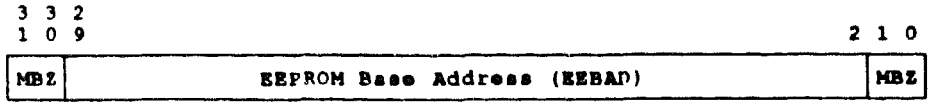
Type: –, 0

Reserved; must be zero

EEPROM Base Address Register (EEBADR)

EEBADR supplies the base address for EEPROM write enables.

ADDRESS *2014 0140 (RSSC)*



bits<31:30>

Name: Reserved

Mnemonic: None

Type: –, 0

Reserved; must be zero.

bits<29:2>

Name: EEPROM Base Address

Mnemonic: EEBAD

Type: R/W, 0

EEBAD supplies the base address for EEPROM write enables. EEBAD is loaded with 2008 0000 (hex) during normal operation.

NOTE: **Although the EEPROM is double-mapped in the address space, it is written only through the first mapping (through addresses 2008 0000 to 2008 7FFF (hex)).**

bits<1:0>

Name: Reserved

Mnemonic: None

Type: –, 0

Reserved; must be zero.

EEPROM Address Decode Mask Register (EEADMR)

EEADMR supplies the bit mask that selects the address extent of the EEPROM address decode.

ADDRESS

2014 0144 (RSSC)

bits<31:30>

Name: Reserved

Mnemonic: None

Type: –, 0

Reserved; must be zero.

bits<29:2>

Name: EEPROM Address Decode Mask Register

Mnemonic: EEADMR

Type: R/W, 0

EEADMR supplies the bit mask that selects the address extent of the EEPROM address decode. Ones in EEADMR indicate those bits that are to be ignored during the address compare. EEADMR is loaded with 0000 7FFF (hex) during normal operation as the address range of the EEPROM is 32 Kbytes.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: –, 0

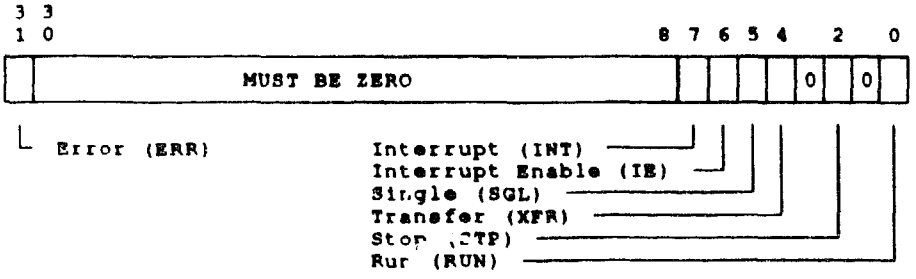
Reserved; must be zero.

Timer Control Register 0 (TCR0)

TCR0 controls timer 0.

ADDRESS

2014 0160 (RSSC)



bit<31>

Name: Error

Mnemonic: ERR

Type: R/W1C, 0

ERR is set whenever the Timer Interval Register overflows and INT is already set, indicating a missed overflow.

bits<30:8>

Name: Reserved

Mnemonic: None

Type: R/W

Reserved; must be zero.

bit<7>

Name: Interrupt

Mnemonic: INT

Type: R/W1C, 0

INT is set whenever the Timer Interval Register overflows. If IE is set when INT is set, an interrupt is posted at IPL 15.

bit<6>

Name: Interrupt Enable
Mnemonic: IE
Type: R/W, 0

When IE is set, the timer interrupts at IPL 15 when INT is set.

bit<5>

Name: Single
Mnemonic: SGL
Type: R/W, 0

Setting SGL causes the Timer Interval Register to increment by one if the RUN bit is cleared. If RUN is set, then writes to SGL are ignored. SGL is always read as zero.

bit<4>

Name: Transfer
Mnemonic: XFR
Type: R/W, 0

Setting XFR causes the Timer Next Interval Register to be copied into the Timer Interval Register.

bit<3>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bit<2>

Name: Stop
Mnemonic: STP
Type: R/W, 0

STP determines whether the timer stops after an overflow. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops.

bit<1>

Name: Reserved

Mnemonic: None

Type: R/W

Reserved; must be zero.

bit<0>

Name: Run

Mnemonic: RUN

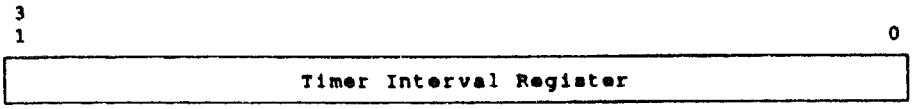
Type: R/W, 0

When RUN is set, the Timer Interval Register is incremented once every microsecond. INT is set when the timer overflows. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops. When RUN is clear, the Timer Interval Register is not incremented automatically.

Timer Interval Register 0 (TIR0)

TIR0 contains the interval count for timer 0.

ADDRESS *2014 0164 (RSSC)*



bits<31:0>

Name: Timer Interval Register 0

Mnemonic: TIR0

Type: RO, 0

When TCR0<0> (RUN) is one, TIR0 is incremented once every microsecond. When the counter overflows, TCR0<7> is set, and an interrupt is posted at IPL 15 if TCR0<6> is set. Then, if TCR0<2> is zero, TCR0<0> is cleared and counting stops.

Timer Next Interval Register 0 (TNIR0)

TNIR0 sets the interval timer 0.

ADDRESS *2014 0168 (RSSC)*

bits<31:0>

Name: Timer Next Interval Register 0

Mnemonic: TNIR0

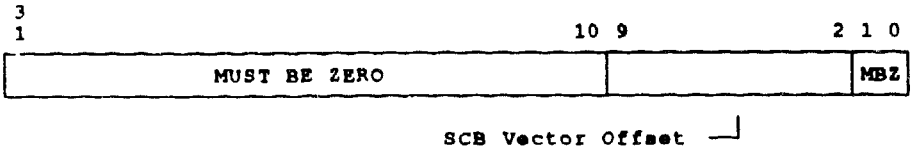
Type: R/W, 0

TNIR0 contains the two's complement of the interval in microseconds of the value that is written into TIR0 after an overflow or in response to the setting of TCR0<4> (XFR). Therefore, to set the interval to 200 microseconds, the value of –200 (FFFF FF38 (hex)) needs to be loaded into TNIR0.

Timer Interrupt Vector Register 0 (TIVR0)

TIVR0 is used by timer 0.

ADDRESS *2014 016C (RSSC)*



bits<31:10>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bits<9:2>

Name: SCB Vector Offset
Mnemonic: None
Type: R/W, 0

When TCR0<6> (IE) and TCR0<7> (INT) transition to a one, an interrupt is posted at IPL 15. When a timer's interrupt is acknowledged, the contents of SCB Vector Offset are passed to service the interrupt request.

NOTE: Both timers interrupt at the same IPL as the console serial line. When multiple interrupts are pending, the console serial line has priority over the timers, and timer 0 has priority over timer 1.

bits<1:0>

Name: Reserved
Mnemonic: None
Type: R/W, 0

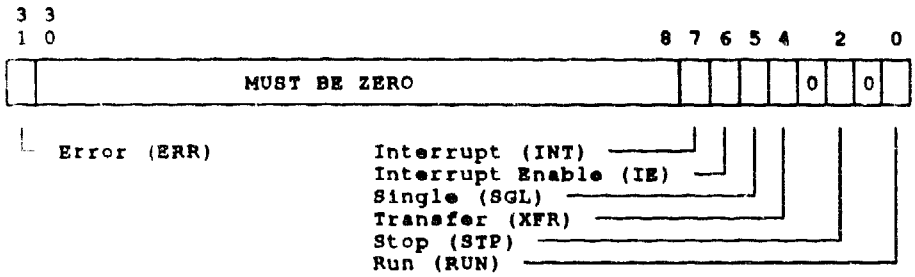
Reserved; must be zero.

Timer Control Register 1 (TCR1)

TCR1 controls timer 1, which is used by the console program.

ADDRESS

2014 0170 (RSSC)



bit<31>

Name: Error

Mnemonic: ERR

Type: R/W1C, 0

ERR is set whenever the Timer Interval Register overflows and INT is already set, indicating a missed overflow.

bit<30:8>

Name: Reserved

Mnemonic: None

Type: R/W

Reserved; must be zero.

bit<7>

Name: Interrupt

Mnemonic: INT

Type: R/W1C, 0

INT is set whenever the Timer Interval Register overflows. If IE is set when INT is set, an interrupt is posted at IPL 15.

bit<6>

Name: Interrupt Enable
Mnemonic: IE
Type: R/W, 0

When IE is set, the timer interrupts at IPL 15 when INT is set.

bit<5>

Name: Single
Mnemonic: SGL
Type: R/W, 0

Setting SGL causes the Timer Interval Register to increment by one if the RUN bit is cleared. If RUN is set, then writes to SGL are ignored. SGL is always read as zero.

bit<4>

Name: Transfer
Mnemonic: XFR
Type: R/W, 0

Setting XFR causes the Timer Next Interval Register to be copied into the Timer Interval Register.

bit<3>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bit<2>

Name: Stop
Mnemonic: STP
Type: R/W, 0

STP determines whether the timer stops after an overflow. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops.

bit<1>

Name: Reserved
Mnemonic: None
Type: R/W

Reserved; must be zero.

bit<0>

Name: Run
Mnemonic: RUN
Type: R/W, 0

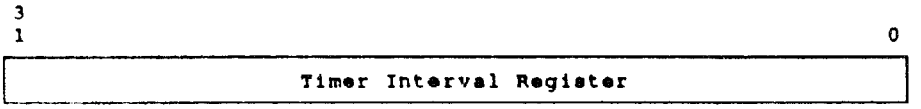
When RUN is set, the Timer Interval Register is incremented once every microsecond. INT is set when the timer overflows. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops. When RUN is clear, the Timer Interval Register is not incremented automatically.

Timer Interval Register 1 (TIR1)

TIR1 contains the interval count for timer 1, which is used by the console program.

ADDRESS

2014 0174 (RSSC)



bits<31:0>

Name: Timer Interval Register 1

Mnemonic: TIR1

Type: RO, 0

When TCR1<0> (RUN) is one, TIR1 is incremented once every microsecond. When the counter overflows, TCR1<7> is set, and an interrupt is posted at IPL 15 if TCR1<6> is set. Then, if TCR1<2> is zero, TCR1<0> is cleared and counting stops.

Timer Next Interval Register 1 (TNIR1)

TNIR1 sets the interval for timer 1. It is used by the console program.

ADDRESS

2014 0178 (RSSC)

bits<31:0>

Name: Timer Next Interval Register 1

Mnemonic: TNIR1

Type: R/W, 0

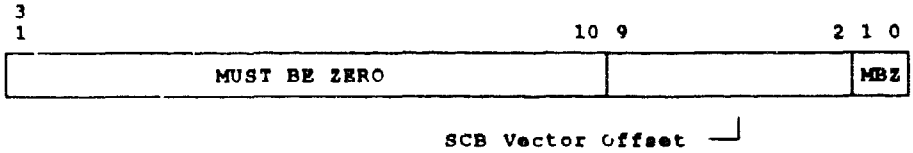
TNIR1 contains the two's complement of the interval in microseconds of the value that is written into TIR1 after an overflow or in response to the setting of TCR1<4> (XFR). Therefore, to set the interval to 200 microseconds, the value of –200 (FFFF FF38 (hex)) needs to be loaded into TNIR1.

Timer Interrupt Vector Register 1 (TIVR1)

TIVR1 is used by timer 1, which is used by the console program.

ADDRESS

2014 017C (RSSC)



bits<31:10>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bits<9:2>

Name: SCB Vector Offset

Mnemonic: None

Type: R/W, 0

When TCR1<6> (IE) and TCR1<7> (INT) transition to a one, an interrupt is posted at IPL 15. When a timer's interrupt is acknowledged, the contents of SCB Vector Offset are passed to service the interrupt request.

NOTE: Both timers interrupt at the same IPL as the console serial line. When multiple interrupts are pending, the console serial line has priority over the timers, and timer 0 has priority over timer 1.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: R/W, 0

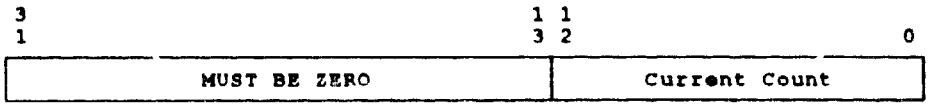
Reserved; must be zero.

Interval Counter Register (SSCICR)

SSCICR controls transitions of the INT TIM L pin on the CPU-chip during diagnostic testing.

ADDRESS

2014 01F8 (RSSC)



bits<31:13>

Name: Reserved

Mnemonic: None

Type: –, 0

Reserved; must be zero.

bits<12:0>

Name: Current Count

Mnemonic: None

Type: R/W

Current Count controls transitions of the INT TIM L signal on the CPU-chip. A high-to-low transition of that signal, when enabled by ICCS<6> (Interrupt Enable) and PSL<20.16> (IPL), causes an interval timer interrupt at IPL 16 (hex).

Current Count increments once every microsecond. When this results in a carry out (from 1FFF (hex)), the INT TIM L signal is toggled and Current Count resets to a value of 0C78 (hex). This causes the INT TIM L signal to toggle every 5 milliseconds as 2000 (hex) through 0C78 (hex) = 1388 (hex) = 5000 (decimal), which causes a high-to-low transition every 10 milliseconds.

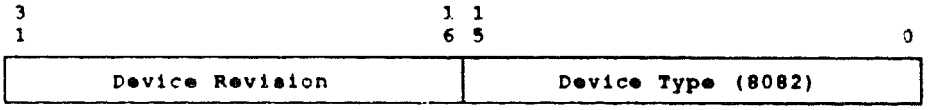
On reset, Current Count clears to zero, so the first interval is 8192 microseconds rather than the expected value of 5000 microseconds. SSCICR is used only for diagnostic purposes and is not written during normal system operation.

Device Register (XDEV)

The Device Register contains information to identify the node. Both fields are loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS

Nodespace base address + 0000 0000 (XCA)



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

Identifies the functional revision level of the module in hexadecimal. The DREV field always reflects the letter revision of the module as follows:

KA64A CPU Module Revision	DREV (decimal)	DREV (hex)
A0	1	0001
A1	1	0001
B0	2	0002
B1	2	0002
.		
.		
.		
Z0	26	001A

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

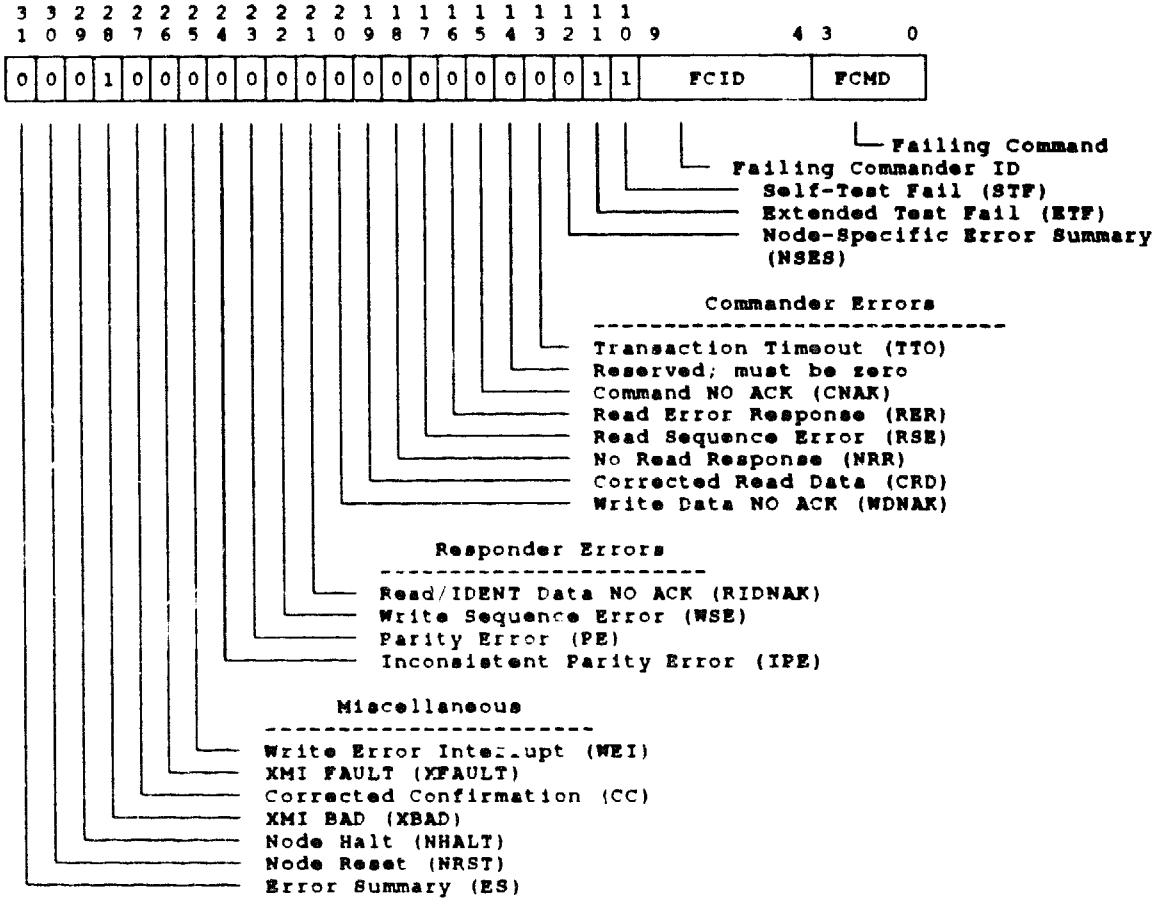
Type: R/W, 0

Identifies the type of node. The Device Type field is broken into two subfields: Class and ID. The Class field indicates the major category of the node. The ID field uniquely identifies a particular device within a specified class. DTYPE contains 8082 (hex) for the KA64A CPU module.

Bus Error Register (XBER)

The Bus Error Register contains error status on a failed XMI transaction. This status includes the failed command, commander ID, and an error bit that indicates the type of error that occurred. This status remains locked up until software resets the error bit(s).

ADDRESS *Nodespace base address + 0000 0004 (XCA)*



CAUTION: There is an anomaly in the KA64A CPU module that causes RCSR<4> (Second Error) to set, under some circumstances, resulting in a spurious hard error interrupt. Hard error interrupt handlers must deal with interrupts that have no error bits set. See Figure 3–34 for valid hard error interrupts.

bit<31>

Name: Error Summary

Mnemonic: ES

Type: RO, 0

The state of ES represents the logical OR of the error bits STF, ETF, NSES, TTO, CNAK, RER, RSE, NRR, CRD, WDNAK, RIDNAK, WSE, PE, IPE, WEI, XFAULT, and CC in this register. Therefore, ES is asserted if any error bit is asserted. ES clears when all error bits are cleared.

bit<30>

Name: Node Reset

Mnemonic: NRST

Type: R/W, 0

Writing a one to NRST initiates, FOR THIS NODE ONLY, a complete power-up reset similar to the assertion and deassertion of XMI DC LO L (see note below); the node performs self-test and asserts XMI BAD L until self-test is successfully completed. Like power-up reset, nodes are precluded from accessing the node from the time it is node reset until it completes self-test (or the maximum self-test time is exceeded).

NOTE: During the time that a node is responding to node reset, the node does not access other nodes on the XMI and it asserts the XMI BAD L signal. In response to a real power-up sequence (caused by XMI DC LO L), the NRST bit resets. Following a node reset sequence, NRST remains set, allowing the processor to recognize that it should not attempt to go through the normal boot process.

bit<29>

Name: Node Halt

Mnemonic: NHALT

Type: R/W, 0

Writing a one to NHALT while halts are enabled, forces the node to go into a "quiet" state while retaining as much state as possible. The KA64A CPU module will force the CPU to halt at the next instruction boundary and go into console mode waiting for console commands. The console code clears NHALT before exit to prevent an immediate reentry.

bit<28>

Name: XMI BAD

Mnemonic: XBAD

Type: R/W, 1

On reads, XBAD indicates the state of the XMI BAD L signal. A one indicates that XMI BAD L is asserted. XMI BAD L is asserted when any one (or more) nodes assert the line and deasserts only when no node asserts it.

Writes to XBAD cause the state to be driven on the wired-OR XMI BAD L line by this node; writing a one asserts XMI BAD L, while writing a zero releases this node's contribution to XMI BAD L.

XBAD asserts on reset, causing XMI BAD L to assert. XMI BAD L remains asserted until all nodes stop asserting it.

bit<27>

Name: Corrected Confirmation

Mnemonic: CC

Type: R/W1C, 0

CC sets when the KA64A CPU module detects a single-bit CNF error. Single-bit CNF errors are automatically corrected by the XCLOCK chip. When CC sets, Error Summary (ES) also sets.

bit<26>

Name: XMI FAULT

Mnemonic: XFAULT

Type: R/W1C, 0

When set, XFAULT indicates that the XMI FAULT signal has been asserted for at least one cycle. When XFAULT sets, Error Summary (ES) also sets.

bit<25>

Name: Write Error Interrupt

Mnemonic: WEI

Type: R/W1C, 0

When set, WEI indicates that the KA64A CPU module received a write error interrupt transaction (IVINTR). When WEI sets, a hard error interrupt is sent to the CPU and Error Summary (ES) sets.

bit<24>

Name: Inconsistent Parity Error

Mnemonic: IPE

Type: R/W1C, 0

When set, IPE indicates that the KA64A CPU module detected a parity error on an XMI cycle and the confirmation for the errored cycle was ACK. This indicates that at least one node (the responder) detected good parity during the cycle time that this KA64A CPU module detected a parity error. If this was a successful write to memory, it could leave the cache incoherent. The KA64A CPU module checks all XMI transactions, not just those it generates. When IPE sets, both Parity Error (PE) and Error Summary (ES) set.

bit<23>

Name: Parity Error

Mnemonic: PE

Type: R/W1C, 0

When set, PE indicates that the REXMI detected a parity error on an XMI cycle. When PE sets, Error Summary (ES) also sets. If appropriate, Inconsistent Parity Error (IPE) sets.

bit<22>

Name: Write Sequence Error

Mnemonic: WSE

Type: R/W1C, 0

During CSR writes to this KA64A CPU module, the REXMI checks the transmitted sequence number against the one that it expects. If the numbers do not match, WSE and Error Summary (ES) set, and the CSR write is ignored.

bit<21>

Name: READ/IDENT Data NO ACK

Mnemonic: RIDNAK

Type: R/W1C, 0

When set, RIDNAK indicates that this KA64A CPU module received a NO ACK confirmation in response to a CSR read. When RIDNAK sets, Error Summary (ES) sets.

bit<20>

Name: Write Data NO ACK
Mnemonic: WDNAK
Type: R/W1C, 0

When set, WDNAK indicates that a write data cycle transmitted by the KA64A CPU module received a NO ACK confirmation. WDNAK sets only if the reattempt fails or is disabled. When WDNAK sets, Error Summary sets. If error retry is enabled, Transaction Timeout (TTO) also sets.

bit<19>

Name: Corrected Read Data
Mnemonic: CRD
Type: R/W1C, 0

When set, CRD indicates that this KA64A CPU module received a CRD read response, meaning that memory saw a parity error when reading data out of memory and corrected it.

bit<18>

Name: No Read Response
Mnemonic: NRR
Type: R/W1C, 0

When set, NRR indicates that a transaction initiated by this KA64A CPU module failed due to a read response timeout. When NNR sets, Error Summary (ES) and Transaction Timeout (TTO) also set.

bit<17>

Name: Read Sequence Error
Mnemonic: RSE
Type: R/W1C, 0

When set, RSE indicates that a transaction initiated by the KA64A CPU module failed due to a read sequence error, since the REXMI checks the transmitted sequence number against the one that it expects. The data returned as the result of a read transaction or interrupt vector returned in an IDENT transaction is out of sequence. When RSE sets, Error Summary (ES) also sets.

bit<16>

Name: Read Error Response

Mnemonic: RER

Type: R/W1C, 0

When set, RER indicates that a node on the XMI received a Read Error Response, meaning that the result of a read transaction or an interrupt vector returned in an IDENT transaction is in error. When RER sets, Error Summary (ES) also sets.

bit<15>

Name: Command NO ACK

Mnemonic: CNAK

Type: R/W1C, 0

When set, CNAK indicates that a command cycle transmitted by the KA64A CPU module received a NO ACK confirmation, usually caused by a reference either to a nonexistent memory location or to an I/O space location. This bit is set only if the error recovery reattempts fail or are disabled. When CNAK sets, Error Summary (ES) also sets.

bit<14>

Name: Reserved

Mnemonic: None

Type: R/W, 0

Reserved; must be zero.

bit<13>

Name: Transaction Timeout

Mnemonic: TTO

Type: R/W1C, 0

When set, TTO indicates that a transaction initiated by this KA64A CPU module failed due to a transaction timeout. The timeout counter is started when the REXMI requests the XMI for a transaction. This bit is set only if retries fail. Write Data NO ACK (WDNAK), No Read Response (NRR), and Command NO ACK (CNAK) indicate the cause of the timeout. If none of the bits are set, the REXMI was never granted the XMI bus for the transaction. When TTO sets, Error Summary (ES) also sets.

bit<12>

Name: Node-Specific Error Summary

Mnemonic: NSES

Type: RO, 0

When set, NSES indicates that a node-specific error condition was detected. NSES is the logical OR of RSCR<27> (Cache Fill Error, CFE), RSCR<28> (Write Data Parity Error, WDPE) and RCSR<4> (Second Error, SE). When NSES sets, Error Summary (ES) also sets. NSES clears when all error bits clear.

bit<11>

Name: Extended Test Fail

Mnemonic: ETF

Type: R/W1C, 1

When set, ETF indicates that the KA64A CPU module has not yet passed its extended test. This bit is cleared by console code when the KA64A CPU module passes its extended test.

bit<10>

Name: Self-Test Fail

Mnemonic: STF

Type: R/W1C, 1

When set, STF indicates that the KA64A CPU module has not yet passed its self-test. This bit is cleared by console code when the KA64A CPU module passes its self-test.

bits<9:4>

Name: Failing Commander ID

Mnemonic: FCID

Type: RO

FCID latches the commander ID of a failing transaction.

bits<3:0>

Name: Failing Command

Mnemonic: FCMD

Type: RO

FCMD latches the command code of a failing transaction, as shown below:

<3:0>	Hex	Command
0000	0	Reserved
0001	1	Read
0010	2	Interlock Read
0011	3	Reserved
0100	4	Reserved
0101	5	Reserved
0110	6	Unlock Write Mask
0111	7	Write Mask
1000	8	Interrupt
1001	9	Identify
1010	A	Reserved
1011	B	Reserved
1100	C	Reserved
1101	D	Reserved
1110	E	Reserved
1111	F	Implied Vector Interrupt

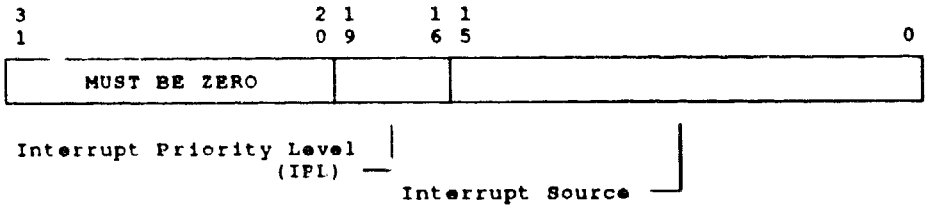
Failing Address Register (XFADR)

The Failing Address Register logs address and length information associated with a failing transaction. The XFADR has an undetermined value on power-up. XFADR, XBER<3:0> (FCMD), and XBER<9:4> (FCID) are latched at the start of every XMI transaction unless one or more of XBER<20> (WDNAK), XBER<18> (NRR), XBER<17> (RSE), XBER<16> (RER), XBER<15> (CNAK), or XBER<13> (TTO) is set at the beginning of the transaction. There are three interpretations of XFADR, depending on XBER<FCMD>.

ADDRESS

Nodespace base address + 0000 0008 (XCA)

XFADR, when XBER<3:0> (FCMD) is 9 (hex), an IDENT transaction:



bits<31:20>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<19:16>

Name: Interrupt Priority Level

Mnemonic: IPL

Type: RO

IPL is a bit mask that specifies the interrupt priority level for the IDENT command as shown below:

Bit	IPL (hex)
19	17
18	16
17	15
16	14

bits<15:0>

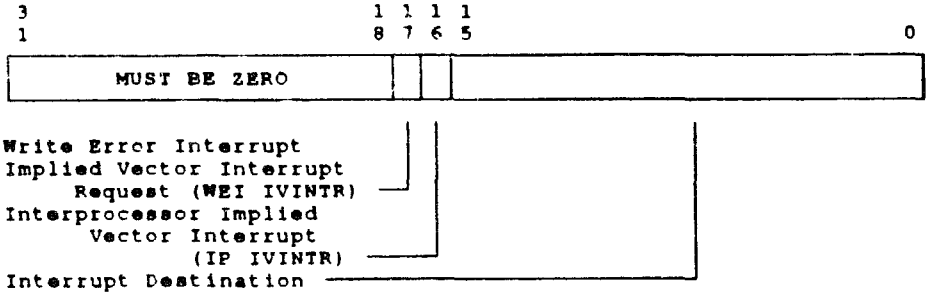
Name: Interrupt Source

Mnemonic: None

Type: RO

The Interrupt Source field is a bit mask that specifies the IDENT command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1.

XFADR, when XBER<3:0> (FCMD) is F (hex), an IVINTR transaction:



bits<31:18>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<17>

Name: Write Error Interrupt Implied Vector Interrupt Request

Mnemonic: WEI IVINTR

Type: RO

WEI IVINTR sets if the implied vector interrupt request was for a write error interrupt.

bit<16>

Name: Interprocessor Implied Vector Interrupt

Mnemonic: IP IVINTR

Type: RO

IP IVINTR sets if the implied vector interrupt request was for an interprocessor interrupt

bits<15:0>

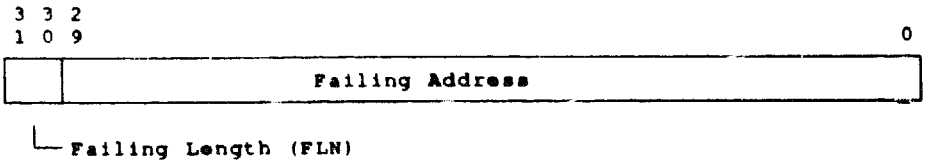
Name: Interrupt Destination

Mnemonic: None

Type: RO

The Interrupt Destination field is a bit mask that specifies the IVINTR command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1.

XFADR, when XBER<3:0> (FCMD) is neither an IDENT transaction nor an IVINTR transaction:



bits<31:30>

Name: Failing Length

Mnemonic: FLN

Type: RO

FLN logs the value of XMI D<31:30> during the command cycle of a failing transaction.

bits<29:0>

Name: Failing Address

Mnemonic: None

Type: RO

The Failing Address field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

XMI General Purpose Register (XGPR)

The XGPR is a general purpose register that is visible to the XMI bus. This register is used during self-test and by the ROM-based diagnostics.

ADDRESS

Nodespace base address + 0000 000C (XCA)

bits<31:0>

Name: XMI General Purpose Register

Mnemonic: XGPR

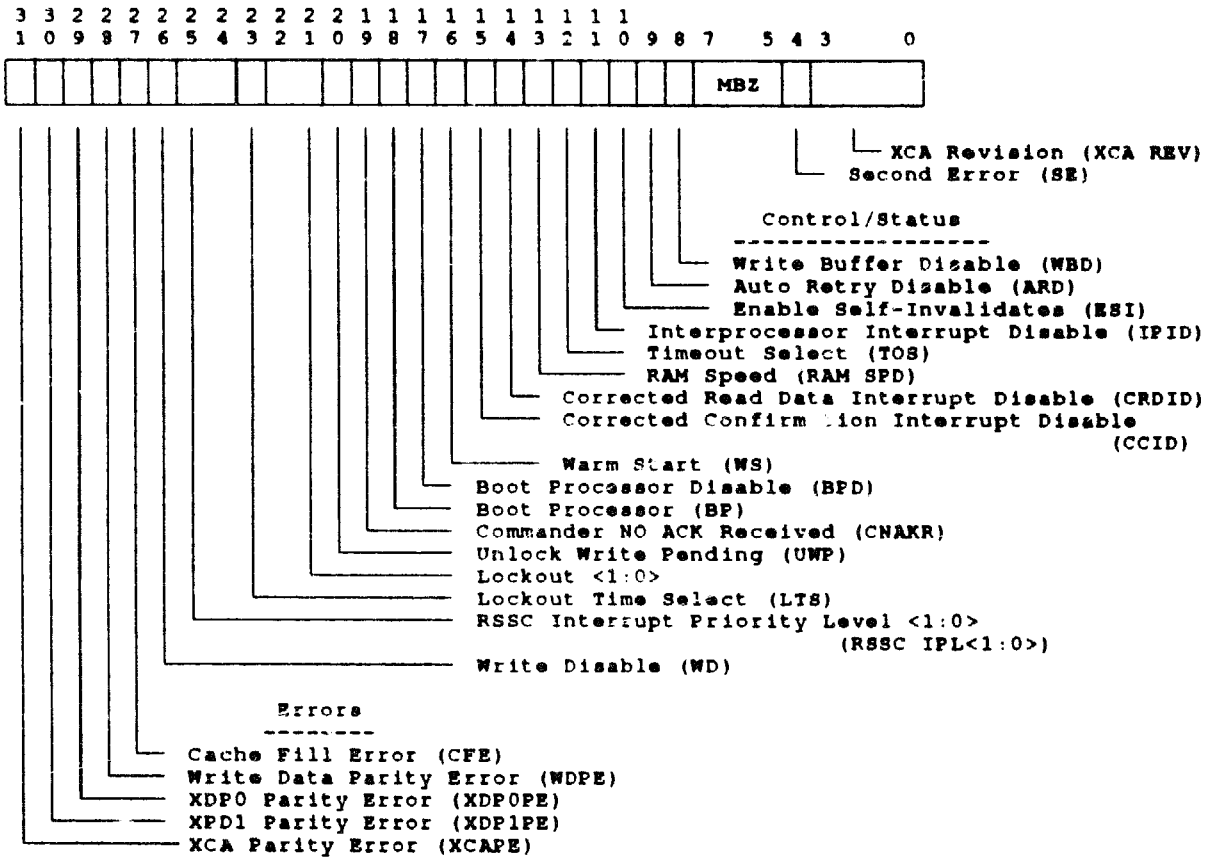
Type: R/W, 0

The general purpose register is used by self-test and during ROM-based diagnostics.

REXMI Control and Status Register (RCSR)

RCSR provides KA64A CPU module control and status to the XMI bus.

ADDRESS *Nodespace base address + 0000 0010 (XCA)*



CAUTION: Writes to RCSR from another node cause UNPREDICTABLE behavior of the system. This node's RCSR MUST be written only from this node's KA64A CPU module.

bit<31>

Name: XCA Parity Error
Mnemonic: XCAPE
Type: R/W1C, 0

XCAPE sets when the XCA detects a parity error on XMI D<31:0>, XMI F<3:0>, or XMI ID<5:1>. XCAPE sets with XBER<23> (Parity Error).

bit<30>

Name: XDP1 Parity Error
Mnemonic: XDP1PE
Type: R/W1C, 0

XDP1PE sets when the XDP1 chip detects a parity error on XMI D<31:0>. XDP1PE sets with XBER<29> (Parity Error).

bit<29>

Name: XDP0 Parity Error
Mnemonic: XDP0PE
Type: R/W1C, 0

XDP0PE sets when the XDP0 chip detects a parity error on XMI D<31:0>. XDP0PE sets with XBER<29> (Parity Error).

bit<28>

Name: Write Data Parity Error
Mnemonic: WDPE
Type: R/W1C, 0

WDPE sets when a parity error is detected during a memory write transaction on the DAL bus. WDPE sets with XBER<12> (Node-Specific Error Summary).

bit<27>

Name: Cache Fill Error
Mnemonic: CFE
Type: R/W1C, 0

CFE sets when an error occurred in the second quadword of a memory read. The cause of the error is distinguished by XBER<18> (No Read Response), XBER<17> (Read Sequence Error), XBER<16> (Read Error Response), and XBER<13> (Transaction Timeout). CFE sets with XBER<12> (Node-Specific Error Summary).

bit<26>

Name: Write Disable

Mnemonic: WD

Type: R/W, 0

WD is used only during self-test and diagnostic testing. It writes data into the cache without generating XMI memory writes. WD is set to zero for normal operation.

When WD is set to one, it causes the REXMI to acknowledge write transactions but does not send the write to memory. For clear write buffer transactions, the REXMI marks any occupied write buffer entry full (none should be), and acknowledges the transactions without waiting for the write buffer or the invalidate queue to be flushed.

bits<25:24>

Name: RSSC Interrupt Priority Level<1:0>

Mnemonic: RSSC IPL<1:0>

Type: R/W, 0

RSSC IPL<1:0> selects the IPL for RSSC-requested interrupts, as shown below. RSSC IPL<1:0> must be set to 01 (hex) for normal operation.

<1:0>	IPL (hex)
00	14
01	15
10	16
11	17

CAUTION: RCSR<25:24> (RSSC IPL<1:0>) MUST match SSCCNR<25:24> (IPL SEL) to ensure correct operation. If these values differ, the operation of the interrupt system is UNDEFINED.

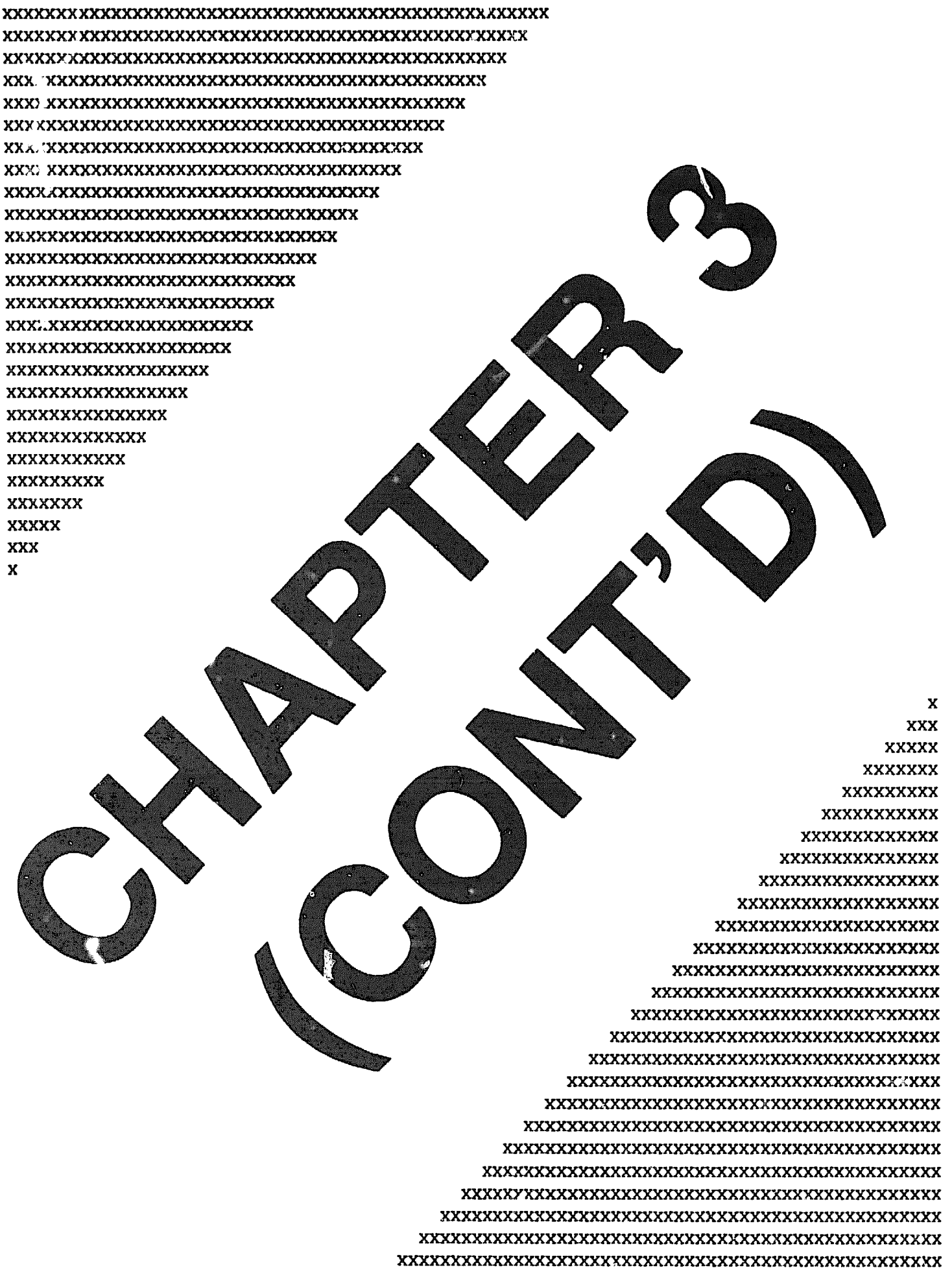
bit<23>

Name: Lockout Time Select

Mnemonic: LTS

Type: R/W, 0

LTS selects one of two values for the delay between the initial request of the XMI and the assertion of the XMI LOCKOUT L signal if a lockout condition is detected. The delay allows normal bus protocols to resolve potential lockout conditions before the REXMI invokes the lockout protocols. When LTS is zero, the delay is 256 XMI cycles; when one, the delay is 4096 cycles. LTS is set to zero for normal operation.


CHAPTER 3
(CONT'D)

bits<22:21>

Name: Lockout<1:0>

Mnemonic: None

Type: R/W

Lockout <1:0> selects the lockout avoidance mode as shown below.

<1:0>	Description
00	Interlock lockout avoidance is disabled. The XMI LOCKOUT L signal is still asserted as defined for Lockout<1:0> = 01.
01	Interlock lockout avoidance is enabled.
10	Reserved
11	Reserved

Set Lockout<1:0> to 01 for normal operation.

bit<20>

Name: Unlock Write Pending

Mnemonic: UWP

Type: R/W1C, 0

UWP is set by the REXMI when an Interlock Read is generated by this KA64A CPU module. The REXMI clears UWP when a subsequent Unlock Write is generated by this KA64A CPU module. The setting and clearing of UWP is not gated by the successful transmission of the XMI transaction. UWP is used during error recovery to determine if a read error is recoverable.

bit<19>

Name: Commander NO ACK Received

Mnemonic: CNAKR

Type: R/W1C, 0

CNAKR sets whenever a command/address NO ACK is received in response to an XMI commander transfer. A NO ACK is not necessarily an error on the XMI as it is used for a retry function. CNAKR is used by diagnostic tests to determine if a transfer was NO ACKed. The REXMI automatically reattempts all XMI commander transfers that are NO ACKed, unless RCSR<9> (Auto Retry Disabled) is a one, until a timeout occurs.

bit<18>

Name: Boot Processor
Mnemonic: BP
Type: R/W, 0

BP is set by console code to indicate that this KA64A CPU module is the boot processor.

bit<17>

Name: Boot Processor Disable
Mnemonic: BPD
Type: R/W, 0

BPD is set by console code on power-up to indicate that this KA64A CPU module node is not eligible to become the boot processor. Following self-test, all KA64A CPU modules examine the BPD and XBER<1> (Self-Test Failed) of each processor node to determine which processor will become the boot processor. A processor node must be initialized before STF clears if its BPD is to take part in the selection of a boot processor.

bit<16>

Name: Warm Start
Mnemonic: WS
Type: RO

WS sets to indicate that battery-backed-up power was maintained during a power failure and that the console code should attempt a "warm start." WS is loaded with the state of the XMI RESET L signal when the XMI DC LO L signal is deasserted. WS is not valid after a node reset.

bit<15>

Name: Corrected Confirmation Interrupt Disable
Mnemonic: CCID
Type: R/W, 0

CCID is a one when corrected confirmation error interrupts are disabled. XBER<27> (Corrected Confirmation) is still set when a corrected confirmation error is detected but the IPL 1A (hex) interrupt request to the CPU-chip is inhibited. If CC interrupts are disabled, software clears XBER<CC> before reenabling interrupts to ensure that only newly generated CC errors cause interrupts.

bit<14>

Name: Corrected Read Data Interrupt Disable

Mnemonic: CRDID

Type: R/W, 0

CRDID is a one when corrected read data interrupts are disabled. XBER<19> (Corrected Read Data) is still set when a corrected read data response is received, but the IPL 1A (hex) interrupt request to the CPU-chip is not allowed. If CRD interrupts are disabled, software clears XBER<CRD> before reenabling interrupts to ensure that only newly generated CRD responses cause interrupts.

bit<13>

Name: RAM Speed

Mnemonic: RAM SPD

Type: R/W, 0

RAM SPD indicates the speed of the backup cache RAMs. RAM SPD is set to zero by the console program for the KA64A CPU module.

CAUTION: RAM SPD MUST match BCCTL<4> (Two-Cycle RAMs) for correct operation. If these values differ, the backup cache operation is UNDEFINED.

bit<12>

Name: Timeout Select

Mnemonic: TOS

Type: R/W, 0

TOS selects the timeout value that is used to detect both responses and reattempt timeout conditions. TOS is used only for diagnostic testing and is a zero during normal operation. When TOS is a zero, the timeout valus is ≈16.77 milliseconds. When TOS is a one, the timeout value is ≈16.38 microseconds.

bit<11>

Name: Interprocessor Interrupt Disable

Mnemonic: IPID

Type: R/W, 0

IPID, when set, does not send interprocessor interrupts to the CPU-chip. The REXMI still responds to XMI interprocessor interrupts but the interrupt from the CPU-chip is masked. IPID is used only for diagnostic testing and is clear during normal operation.

bit<10>

Name: Enable Self-Invalidates
Mnemonic: ESI
Type: R/W, 0

ESI, when set, allows the REXMI to process invalidates for writes generated by this node. Normally, only writes from other nodes cause invalidate processing. ESI and XMI memory permits a single KA64A CPU module node to verify the invalidate operation. ESI is set only for diagnostic testing and is clear during normal operation.

bit<9>

Name: Auto Retry Disable
Mnemonic: ARD
Type: R/W, 0

When set, ARD causes REXMI to disable retries of command cycle NO ACK, write data cycle NO ACK, and interlock read LOC responses. An error is reported to the CPU-chip immediately after the error condition is detected. ARD is used only for diagnostic testing and is clear during normal operation.

bit<8>

Name: Write Buffer Disable
Mnemonic: WBD
Type: R/W, 0

When WBD is a one, the REXMI write buffer is disabled so that all writes are written directly to memory. Since an I/O space write is needed to set this bit, the current contents of the write buffer, if any, are flushed to memory before the write buffer is disabled. WBD is used only for diagnostic testing and is a zero for normal operation.

bits<7:5>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<4>

Name: Second Error

Mnemonic: SE

Type: R/W1C, 0

SE sets when an error occurs while XFADR and XBER<FCMD> are currently locked by a previous error. Errors that lock XFADR and XBER<FCMD> and set SE are WDNAK, NRR, RSE, RER, and CNAK. When SE sets, XBER<NSES> also sets and a hard error interrupt is requested from the CPU.

CAUTION: There is an anomaly in the KA64A CPU module that causes SE to set, under some circumstances, resulting in a spurious hard error interrupt. Hard error interrupt handlers must deal with interrupts that have no error bits set. See Figure 3–34 for valid hard error interrupts.

bits<3:0>

Name: XCA Revision

Mnemonic: XCA REV

Type: RO

XCA REV contains the revision level of the REXMI XCA chip.

3.9 KA64A CPU Module Initialization, Self-Test, and Booting

This section gives the KA64A CPU module initialization overview; describes the results of initialization; and then discusses the bootstrapping or restarting of the operating system.

3.9.1 Initialization Overview

The three ways to reset the KA64A CPU module are:

- Power-Up Sequence—When the VAX 6000–400 is powered up, XMI AC LO L and XMI DC LO L are sequenced so that all XMI nodes are reset.
- System Reset—The XMI emulates a power-up sequence by asserting the XMI RESET L line, causing the power supply to sequence XMI AC LO L and XMI DC LO L as in a "real" power-up. The XMI does not differentiate between a "real" power-up and a system reset. A system reset is caused by:
  - — Software that asserts XMI RESET L by writing to IPR55, IORESET, with an MTPR instruction. For example, the console INITIALIZE command generates a system reset if no argument is given by using this mechanism.
  - — The XTC power sequencer asserts the XMI RESET L line when the control panel Restart button is pushed.
- Node Reset—Any CPU can be "node reset" by setting its XBER<30> (NRST) bit. The console INITIALIZE command generates a node reset if a node ID argument is provided. The difference between the node reset and a system reset is that XMI AC LO L is not sequenced during a node reset.

Typing CTRL-P at the console terminal or certain errors also cause initialization. Refer to Section 3.11 for a discussion of error handling. The *VAX 6000–400 Owner's Manual* discusses the operation of the system console.

In response to a power-up or system reset, the KA64A CPU module(s) perform the following sequence:

1 Reset(s) to a known state. (Refer to the individual registers and their bits for their state on reset, after microcode self-test completes.)

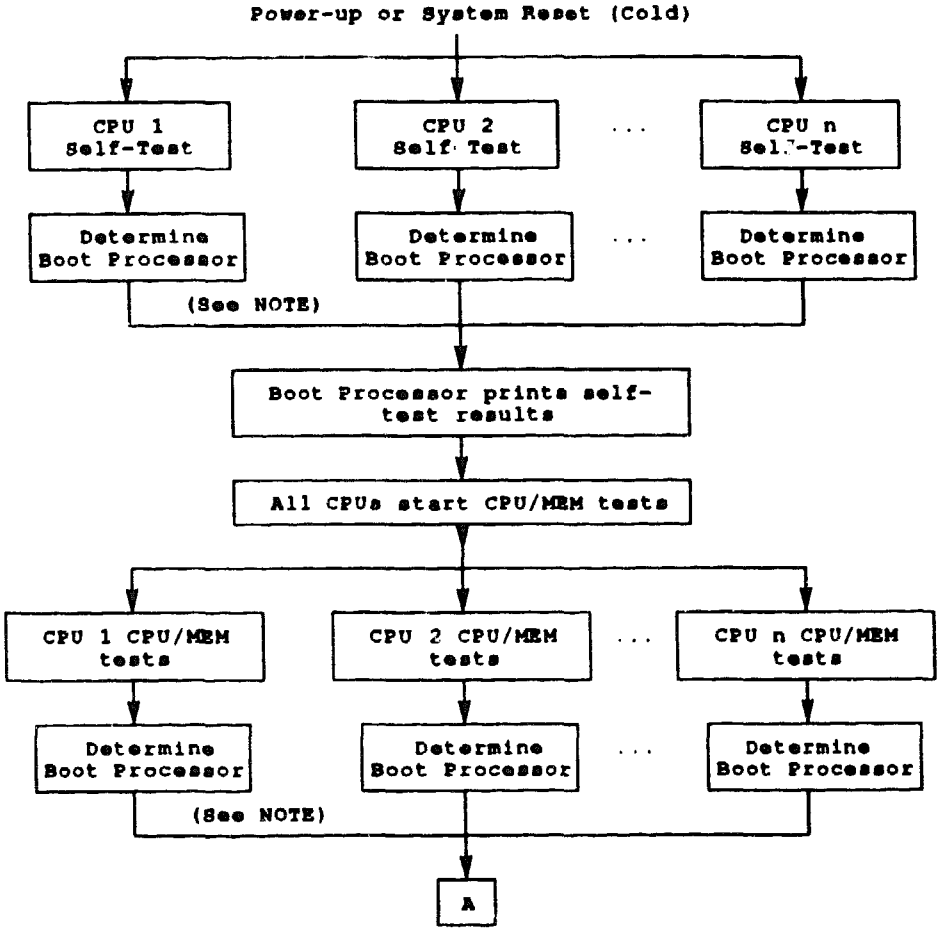
2 All CPUs start executing the console program at 2004 0000 in ROM. The console program initializes the registers and executes a complete ROM-based diagnostic (RBD) self-test and extended tests.

3 The operating system initialization code performs the final system initialization.

3.9.2 Detailed Initialization Description

The following is a flowchart and summary of the initialization process.

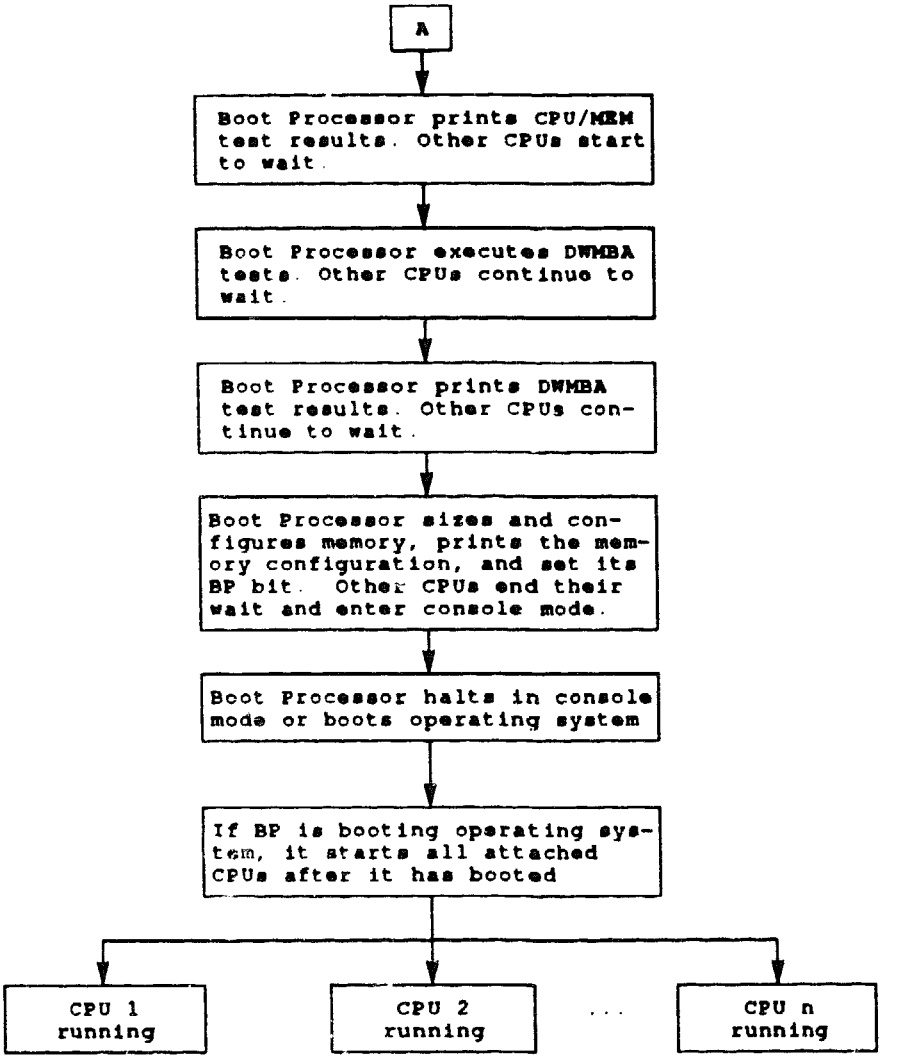
Figure 3–29 Initialization Flowchart



NOTE: The second determination of the Boot Processor occurs even if the original Boot Processor passes all memory tests.

Figure 3–29 Cont'd. on next page

Figure 3–29 (Cont.) Initialization Flowchart



When a CPU module resets, it first performs a microcode self-test that briefly checks the CPU-chip. Then the console program transfers control to the RBDs, which run a more extensive CPU self-test. After the RBDs complete, control is returned to the console program. Nodes do not access other nodes on the XMI during CPU self-test, limiting CPU self-test to intramodule operations and loopback transactions on the XMI to check their interface to the XMI. If a vector module is present, vector self-test runs at this time.

If the CPU self-test is successful, the Self-Test Passed (STP) LED is lit, XBER<STF> is cleared, and the XMI BAD L signal is cleared.

After CPU self-test completes, a boot processor (BP) is selected from those CPU nodes that passed self-test. This is the first of two BP selections before the operating system starts. The BP prints the results of the self-test.

The BP then controls an additional test that requires all CPU nodes to access memory. This test is called the CPU/MEM test and allows CPU nodes to check additional logic that could not be tested during the CPU self-test. If a vector module is present, it too participates in the CPU/MEM testing.

Each processor extinguishes its STP LED when the CPU/MEM tests are initiated. During the CPU/MEM tests, the CPU nodes continue testing themselves and access preallocated blocks in memory. These tests verify CPU logic that requires memory for testing.

If these tests complete successfully, each CPU node lights its STP LED and clears its XBER<ETF> bit.

The second BP selection now occurs. If the original boot processor completes all of its CPU tests successfully, it remains the BP. Otherwise, another processor node is chosen to be the BP.

The BP then tests all the DWMBAs in the system. If the DWMBA tests are successful, the BP lights the DWMBA/A's yellow LED and the DWMBA/B's yellow LED.

Finally, the BP configures the memory nodes with correct interleave and address parameters, initializes the console communications area (CCA), and allocates at least 256 Kbytes of memory to each of the other CPU nodes.

During a warm restart, no CPU/MEM tests or DWMBA tests are performed. The test sequence goes directly from the CPU self-test to console mode.

On node reset, CPU nodes run self-test and then initialize some state before the console program initializes additional state to enable normal operation. Then the operating system is started. See Table 3–17 for the console-initialized state of the CPU module's registers.

3.9.2.1 Initialization State Summary

When a KA64A CPU module node is reset, the module hardware and microcode initialize certain state to known values (called the hardware-initialized state). The hardware-initialized state is shown for each register field, as explained in Table 3–14. The console then runs its self-test, and in the process loads new states into many of the registers. The KA64A CPU module's state that results from a successful self-test is called the console-initialized state and is the state when the console program transfers control to the operating system. This console-initialization state is shown in Table 3–17.

3.9.2.2 Power-Up Initialization

Entry to the initialization sequence starts at physical address 2004 0000 (hex), the entry point of the console program. Housekeeping chores, such as establishing a scratch area and stack happen first. The SCBB contents are then saved in RSSC RAM, and the SCBB is set to point to the console's SCB, allowing the console program to handle any exceptions or machine checks.

The console program next turns on the Self-Test Valid LED, the first visible indication that the console program has begun executing. The console program then determines the type of reset as more initialization is performed for power-up starts than for warm starts.

If set, the RCSR<WS> (Warm Start) bit indicates that power to the memory arrays was not lost and the memory contents are valid. If the Warm Start bit is zero, memory was lost and the entry is a cold start. If RCSR<NRST> is set, this is a node reset and not a system reset. Memory is assumed to be preserved for node resets. Refer to Section 3.9.2.3 for the warm start initialization and to Section 3.9.2.4 for a node reset.

Significant events of a power-up initialization include the following:

- Self-test is run.
- SSCCNR is configured.
- TIR1 is started.
- SCCBTR is set for its bus timeout interval.
- The interrupt vectors for the programmable timers are set.
- The primary cache, backup cache, and write buffer are initialized:
  - P-cache refresh is enabled, errors are cleared, and the P-cache is disabled.
  - All P-cache tag entries are written with a tag that has good parity and a clear valid bit.
  - B-cache refresh is enabled, errors are cleared, and the backup cache is disabled.
  - All B-cache tag entries are written with a tag that has good parity and a clear valid bit.
  - All B-cache primary cache tag store tag entries are written with a tag that has good parity and a clear valid bit.

— Both caches and the write buffer are disabled.

- EEADMR is set.
- The contents of the EEPROM are verified by checksum.
- SSCCNR is set to allow the BREAK key to be the console break character.
- The initialization of the console scratch area is completed.
- If appropriate, the RCSR<Boot Processor Disable> bit is set.

Table 3–17 CPU Module Registers Console-Initialized State

Register	Bit Settings[1]							
IPL[2]	0000	0000	0000	0000	0000	0000	0001	1111
ASTLVL[3]	0000	0000	0000	0000	0000	0000	0000	0100
SISR[4]	0000	0000	0000	0000	0000	0000	0000	0000
ICCS[5]	0000	0000	0000	0000	0000	0000	0000	0000
RXCS	0000	0000	0000	0000	0000	0000	0000	0000
RXDB	0000	0000	0000	0000	0000	0000	xxxx	xxxx
TXCS	0000	0000	0000	0000	0000	0000	0000	0000
TXDB	0000	0000	0000	0000	0000	0000	xxxx	xxxx
ACCS[6]	0000	0000	0000	0000	0000	0000	0000	00xx
MAPEN[7]	0000	0000	0000	0000	0000	0000	0000	0000
BCSTS[8]	0000	0xxx	xxxx	xxx0	0000	0000	0000	0000
BCCTL[8]	0000	0000	0000	0000	0000	0000	0000	1000
VINTSR[9]	0000	0000	0000	0000	0000	0000	0000	000x
PCSTS[10]	0000	0000	0000	0000	000x	xxxx	000x	1000
CREG0[11]	0000	0000	0000	0000	0000	0000	0000	0xxx
CREG1	0000	0000	0000	0000	0000	0000	1100	0000
SSCBAR	0010	0000	0001	0100	0000	0000	0000	0000
SSCCNR[12]	x000	0001	0111	0110	1xxx	0000	0100	0101
SSCBTR	0000	0000	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
OPORT	0000	0000	0000	0000	0000	00xx	xxxx	xxxx
IPORT[13]	x000	0000	0000	0000	0000	0000	01xx	xxxx
CRBADR	0010	0000	0000	0000	0000	0000	0000	0000
CRADMR	0000	0000	0000	0000	0000	0000	0000	0000
EEBADR	0010	0000	0000	1000	0000	0000	0000	0000
EEADMR	0000	0000	0000	0000	1111	1111	1111	1111
XDEV[14]	xxxx	xxxx	xxxx	xxxx	1000	0000	0000	0010
XBER[15]	0000	0000	0000	0000	0000	00xx	xxxx	xxxx

Table 3–17 (Cont.) CPU Module Registers Console-Initialized State

Register	Bit Settings[1]							
RCSR[16]	0000	0001	0110	0xxx	0000	0000	0000	xxxx

[1]Power-up, system, or node reset
[2]IPL = 31
[3]ASTLVL = 4
[4]No interrupts
[5]No timer interrupts
[6]The lower two bits of ACCS reflect the configuration. F-Chip Present, bit <1>, should always be 1 unless the F-chip failed self-test. Vector Present, bit <0>, should be 1 if a good vector module is present, and 0 otherwise.
[7]Memory management disabled
[8]All error bits in BCSTS are cleared. The tag stores in the C-chip are set to refreshing and disabled. All tags have good parity and the valid bit off.
[9]DISABLE VECT INTF, bit <9>, and all error bits should be cleared in VINTSR. If the vector module passes self-test, ACCS<Vector Present> should be set.
[10]INTERRUPT, TRAP1, and TRAP2 are cleared in PCSTS. The tag store is set to refreshing and disabled. All tags have good parity and the valid bit off.
[11]For the BP, TERM_SEL and TERM_ENA select System Console Mode. For all other CPUs, TERM_SEL and TERM_ENA select Auxiliary Console Mode.
[12]BLO is asserted if the battery power dropped below threshold.
[13]The IPORT bits reflect the state of the module signals, including the node ID of the module.
[14]XDEV<31:16> identify the revision level of the KA64A CPU module.
[15]XBER<FCID> and XBER<FCMD> are latched during self-test and extended test. The final values are unpredictable.
[16]RCSR<BP>, RCSR<BPD>, and RCSR<WS> are set as appropriate.

- The REXMI is initialized:
  - XDEV is loaded with the appropriate module revision and device type values.
  - All error bits in RCSR are cleared and those bits that require initialized state are initialized.
  - BCCTL<Two-Cycle RAMs> and RCSR<RAM SPD> are set to the same value.
  - SSCCNR<IPL SEL> and RCSR<IPL> are set to the same value of 15 (hex).
  - All error bits in XBER are cleared and those bits that require initialized state are initialized.
- If appropriate, XBER<XBAD> is cleared to deassert the XMI BAD L signal. XBAD stays set if self-test fails.
- The console terminal baud rate is selected.
- The boot processor is selected, as described in Section 3.9.2.5. The BP prints the initial self-test results on the console terminal and sets its RCSR<BP>.

- All CPU nodes that pass initial self-test execute the additional self-test, which is the CPU/MEM test.
- If the CPU/MEM test is successful, the CPU again clears XBER<XBAD> and RCSR<BP>.
- The BP is redetermined. This newly determined BP prints the results of the CPU/MEM test on the console terminal.
- The BP runs the DWMBA self-test and prints the results. The status of each VAXBI node is printed.
- Each CPU enables its write buffer unless it is explicitly disabled in the EEPROM.
- RCSR<Lockout> is set for Lockout Enabled (01), and RCSR<LTS> is set to zero.
- The BP configures system memory, as described in Section 3.9.2.6 and prints the memory configuration. The memory interleave parameters are read from EEPROM, and the console communications area (CCA) is built in high memory.
- Initialization of each VAXBI node is performed, as described in Section 3.9.2.7.
- The BP's RCSR<BP> bit is set, indicating to the nonboot processors that they can enter console mode.
- All nonboot processors are queried by the BP for their ROM revision, EEPROM revision, and system serial number. A revision banner is printed with warnings about any mismatches found.
- The autostart sequence, described in Section 3.9.3, is entered.

3.9.2.3 Warm Start Initialization

Some of the cold start and initialization steps are inappropriate with a warm start because the contents of memory are preserved. The sequence is modified by:

- Running a minimal set of tests for the memory interface and DWMBA that do not disturb the memory configuration or contents.
- No memory configuration is performed. Instead, each CPU searches memory for the CCA. If a nonboot processor fails to find the CCA, it displays an error code in its LEDs and hangs. If the BP fails to find the CCA, it reconfigures memory as if a cold start had occurred.

3.9.2.4 Node Reset

A node reset is a modified full system reset since some testing and initialization is inappropriate when the remainder of the system continues to function. The modifications are:

- Self-test results are not displayed.
- Additional tests are not run. Instead, the XBER<ETF> and XBER<XBAD> bits are cleared. Additional test results are not printed.

- DWMBA self-test is not run. No initialization of the DWMBA is performed and, therefore, no test results are printed.
- No memory configuration is performed and no revision banner is printed.
- The XBER<NRST> has its initial value stored in the console scratch memory and is then cleared by self-test. This stored value is used by the console program to determine if the entry resulted from a node reset or a system reset.

3.9.2.5 Boot Processor Determination

Each processor examines all other CPU XBER<STF> (Self-Test Failed), XBER<ETF> (Extended Test Failed), and RCSR<BPD> (Boot Processor Disabled) bits. Any processor with these bits clear is a candidate for BP. The candidate with the lowest XMI node ID is expected to become the BP and set its RCSR<BP>. All nonboot processors wait for the designated BP to set its RCSR<BP> bit. If there is a failure, failure codes are displayed on each processor's LEDs.

If no processor is eligible to become BP (any combination of the RSCR<BPD> bit being set or all CPUs failing self-test), the system appears totally unresponsive. The system operator can then intervene to designate one of the processors as BP by typing ">>n" on the console terminal. Processor *n* then becomes the BP. This method of selecting the BP does not change the state of the BP's STF or BPD bits.

3.9.2.6 Memory Configuration

The console program configures memory by setting the interleave and starting address for each array. The console program completely controls the memory configuration because the console uses a portion of the main memory to hold the console communications area (CCA). The console also builds a physical memory bitmap showing all usable and unusable pages. The results of the CPU/MEM test are used to determine the defective pages.

The memory configuration process verifies that a minimum of 256 Kbytes of usable memory per processor is available plus the space used by the CCA and bitmap. The location of the CCA is determined and marked as unusable in the bitmap.

If the boot processor (BP) is unable to find the minimum required memory, it displays an error code on the LEDs and hangs. The BP also sets its RCSR<Boot Processor Disabled> bit, causing the nonboot processors, if any, to determine a new BP. The new BP repeats the memory configuration attempt.

3.9.2.6.1 Selection of Interleave

The interleave is specified by parameters stored in the EEPROM. These parameters are set with the SET MEMORY command. There are three types of interleave:

- DEFAULT – The console program makes all interleave decisions.
- EXPLICIT – The user supplies configuration data.

- NONE – No arrays will be interleaved.

If the EEPROM specifies DEFAULT interleave, the console program attempts to form interleave sets. The largest interleave factor is obtained for each group of like-sized arrays. If there are more arrays than can be evenly interleaved, the criteria is repeated for the remaining arrays until only single arrays remain. The array with the lowest XMI node ID is assigned to the lowest physical address. All arrays of the same size are configured before arrays of a different size.

Any array containing unrecoverable errors is not included in a default interleave set. Instead, it is configured as uninterleaved. The remaining arrays that would have formed the set are freed up for inclusion in another interleave set, if one can be formed.

If the EEPROM specifies EXPLICIT interleave sets, the console program interleaves and configures the arrays in the order specified. When an array in the set has unrecoverable errors, all arrays in the set are configured without interleaving. If a set specifies a nonexistent array or is otherwise inconsistent, all arrays in the set are configured without interleaving.

If the EEPROM specifies NO interleave, the console configures arrays in order by node ID, with the lowest numbered array at the lowest physical address.

3.9.2.6.2 Memory Testing and the Bitmap

Memory self-test indicates that an array has no unrecoverable (hard) errors, one hard error, or multiple hard errors. Self-test executes on all arrays in parallel and is faster than software testing of memory. An attempt is made to use the results of self-test and avoid performing software testing of memory.

A hard (unrecoverable) error is called an RDS error and is defined as one that is an uncorrectable double-bit error by memory hardware. A correctable (CRD) error is not considered a hard error, and pages containing CRD errors are marked as usable.

If self-test indicates that an interleave set contains no hard errors, the set never undergoes software testing. If an array in an interleave set contains one or more hard errors, that set is uninterleaved and the failing array is software tested.

Software testing is performed, one page at a time, beginning with the lowest addressed page in the array. If required, this testing takes about 7 seconds per megabyte. All locations in the page are written with patterns. The locations are then read. If the value read from any location does not match the pattern, or if a machine check occurs reading any location, that page is marked as unusable, and testing resumes with the next page. The testing patterns are in this order:

- All ones
- All zeros
- Alternating one/zero/one
- Alternating one/zero/one with ECC bits complemented

- The address of each longword
- The complement of the address of each longword

The memory bitmap is initially built in the first block of memory large enough to hold it. When the bitmap has been configured, it is then moved to a page-aligned location below the CCA.

If pages must be marked as bad before enough memory has been found to hold the bitmap, some pages are retested after the bitmap has been built. The bitmap shows, in addition to pages marked with hard errors, pages marked as unusable because they are either the bitmap's own pages or are CCA pages. A page is marked as unavailable when its corresponding bitmap bit is cleared.

3.9.2.7 DWMBA Configuration

The console program performs minimal initialization of the DWMBAs following self-test. The initialization performed for each DWMBA is:

1 The BI Starting Address Register (bb+20) and the BI Ending Address Register (bb+24) are initialized to the starting and ending limits of XMI memory.

2 The BICSR (bb+04) has its BI Broke bit cleared.

3 The DMA-B Transmit Buffer is disabled under these conditions:

- The DWMBA/A module Device Register shows less than 10 (decimal).
- The system contains five or more XMI commanders and the DWMBA/B module Device Register shows a revision of less than 0A (hex).
- The VAXBI contains a DWBUA UNIBUS adapter.

If a DWMBA/A module fails its self-test, as indicated by its XBER<STF> being set, the BP asserts its own XBER<XBAD> bit to drive the XMI BAD L line.

3.9.3 Bootstrapping or Restarting the Operating System

A VAX processor can be in one of these five major states:

- Powered off.
- Bootstrapping (which is attempting to load and start) the operating system. If the memory has lost power before bootstrapping starts, it is a "cold start." If the memory's contents are valid because of the battery backup option, it is a "warm start."
- Halted.
- Restarting a halted operating system.
- Running.

It is the console program that bootstraps a copy of the operating system from a tape or disk device and can attempt to restart an existing memory-resident copy of the halted operating system.

Only the boot processor (BP, also called the primary processor) can execute a BOOT command or perform the automatic bootstrap sequence. Nonboot processors (also called secondary processors) remain in console mode awaiting further commands.

Only the BP attempts a bootstrap following a power-up. If the control panel lower key switch is set to Auto Start, the BP restarts and attempts a bootstrap. Any nonboot processors are in a halt until the operating system starts the nonboot processors by passing START commands through the console communications area (CCA). The nonboot processores do not restart system software unless the control panel lower key switch is set to Auto Start.

3.9.3.1 Operating System Restart

The boot processor console program attempts to start/restart the operating system whenever one of the following events occurs:

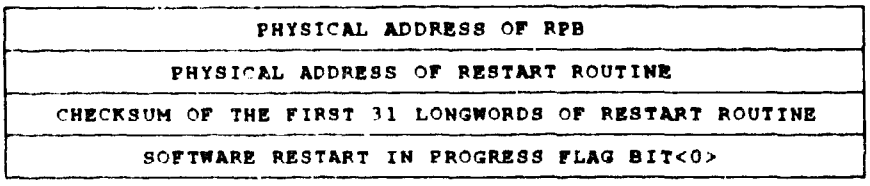
- Power is restored to the processor. If the memory was kept valid by the optional battery backup, it is a warm start; otherwise, it is a cold start.
- A system reset occurs. This is treated as a cold start.
- The running processor halts due to an error halt. This is a restart. A CTRL/P from the console terminal is not considered an error halt.

Restart is suppressed if the control panel key switches are set to Enabled and Halt.

A nonboot processor console attempts a restart only following an error halt only if the control panel lower key switch is in the Auto Start position. For all other halt conditions, the BP is responsible for restarting the nonboot processor.

Restart of the operating system is controlled by a memory data structure called the restart parameter block (RPB), constructed by the operating system. The RPB is a page-aligned structure. The console program's warm start code searches memory for an RPB and, if a valid RPB is found, restarts the operating system at an address stored in the RPB. Figure 3–30 shows the RPD format.

Figure 3–30 Restart Parameter Block Format



The algorithm used to locate the RPB is:

1 Examine the first longword of each page of memory for a location that contains its own physical address. If none is found, the search fails.

2 Test that the second longword of the page contains a valid non-zero physical address. If this test fails, resume Step 1.

3 Obtain the restart address from the second longword. Calculate the signed longword sum of the first 31 longwords of the restart routine, ignoring overflows. If this value does not match the contents of the third longword of the page, resume Step 1.

If all the above tests pass, a valid RPB has been found.

The console program also keeps internal flags to indicate that a restart is in progress. There is one flag for each processor, located at CCA$Q_RESTARTIP, in the CCA. These flags allow the console to avoid repeated attempts to restart a failing system. The operating system clears these flags following the successful restart of a processor.

3.9.3.2 Failing Restart

If the restart of a boot processor fails, a message is displayed on the console terminal and a bootstrap is attempted. A failed restart is a serious condition and causes the other processors to abandon whatever is still running.

If a nonboot processor's restart fails, the console program examines the CCA$_SECSTART field of the CCA. The console program forces a bootstrap in the same manner as for a BP if the bit corresponding to the failing processor is clear. If this bit is set, the console program does not force a bootstrap and the failing processor enters console mode.

The CCA$Q_SECSTART bits are set by the operating system when it attempts to start a nonboot processor. The operating system clears these bits when it is satisfied that the nonboot processor has successfully started executing.

The following scenario, which is peculiar to multiprocessors, is prevented by the CCA$Q_SECSTART bits:

1. A nonboot processor encounters an error halt and then fails to restart.
2. The console program forces a bootstrap.
3. The BP boots and begins running the operating system.
4. The boot processor starts the defective nonboot processor if the nonboot processor passed CPU self-test.
5. The nonboot processor repeats its error halt and fails to restart.
6. The console program again forces a bootstrap and the sequence repeats.

NOTE: A nonboot processor cannot directly perform a reboot because it cannot notify the other nonboot processors that an expected console entry is planned. If the location of the BP changed during the system reset, the fact that a boot was in progress could be lost. To avoid this problem, a nonboot processor forces the boot processor into console mode (via NHALT) and then signals through the CCA that a bootstrap is needed.

3.9.3.3 Restart Parameters

The console program transfers control to the restart address when a valid RPB is found. The console program then passes these restart parameters in the GPRs, as specified by the *VAX Architecture Reference Manual*:

GPR10 – Halt PC

GPR11 – Halt PSL

GPR12, Argument Pointer – Halt code

GPR14, Stack Pointer – Address of the RPB + 512

3.9.3.4 Operating System Bootstrap

The console program causes the BP to attempt to bootstrap the operating system whenever one of the following events occurs:

- The control panel is Enabled and the BOOT command is typed on the console terminal.
- A restart is attempted and fails.
- A power-up occurs when the lower key switch is in the Auto Start position.

Bootstrap attempts to load the primary system bootstrap program, VMB, into memory and begin its execution. VMB is loaded from the device specified by the BOOT command or from a default device recorded in the EEPROM. The VAX 6000–400 uses a set of minimal device handler routines, called boot primitives, to read VMB from the boot device, a technique called "bootblock" booting.

The console program saves the target device information in RSSC RAM as the first phase of bootstrap. The target device information is propagated to all CPUs in the system since the location of the BP can change after a system reset. This causes all processors, memories, and I/O adapters to perform self-test, with the memories being tested as fast as possible. When the console program is reentered, following the reset, it determines that there was an "expected entry," and continues with the bootstrap.

The second phase of bootstrap begins as the console program searches tables in the EEPROM and then the ROM to locate a boot primitive that matches the specified device as the first phase of bootstrap. If a suitable primitive is found, the target device information is saved in GPRs and control transfers to the primitive.

If the boot device is a disk, the primitive loads logical block zero (the "bootblock") into memory and transfers control to it. The bootblock contains code giving the location and size of the VMB image on disk. The bootblock code uses a service routine in the boot primitive to read each block of VMB into memory. If the target device is not a disk, the boot primitive must know how to ask for VMB from the device.

Once VMB is loaded, the console program passes control to it. The boot primitive preserves the boot parameters stored in the GPRs.

A bootstrap can also be triggered by the operating system, via the CCA$V_REBOOT flag in the CCA. This bit is only recognized by the BP.

3.9.3.5 Boot Algorithm

The console maintains a "bootstrap in progress" flag, stored at CCA$V_BOOTIP. This "cold start" flag is used to prevent repeated attempts to automatically bootstrap a failed system.

This algorithm is used to perform the system bootstrap:

1 If this boot attempt is a result of a console BOOT command, skip to Step 3.

2 If the CCA$V_BOOTIP flag is set, the boot fails.

3 Set the CCA$V_BOOTIP flag.

4 Store the boot device and parameters in RSSC RAM on all processors in the system and force a system reset.

5 When the console program is reentered on the BP, resume the bootstrap at this point.

6 Starting at location zero, search for the first page-aligned block of 256 Kbytes of good memory. If such a block cannot be found, the boot fails. This seach is performed by scanning the bitmap built during memory configuration.

7 Search the boot primitive tables in the EEPROM and ROM, in that order, for a primitive that matches the specified boot device. If none is found, the bootstrap fails.

8 Load the GPRs with the boot parameters in the primitive.

9 The console initializes the registers, as described in Table 3–17.

10 Transfer control to the boot primitive.

11 Transfer control to the memory image of VMB at the address loaded in the SP.

If the bootstrap fails, the console program displays a message on the console terminal and then displays the console prompt. If bootstrap succeeds, the operating system clears CCA$V_BOOTIP.

3.9.3.6 Boot Parameters

The console program loads parameters into the GPRs before passing control to VMB. These parameters describe the boot device and any bootstrap options that are to be used. Table 3–18 shows how the registers are used.

Table 3–18 Boot Parameters Loaded Into GPRs

Register	Bits	Description
GPR0	<7:0>	VMB device type code, supplied by the boot primitive
GPR1	<7:4>	XMI node number of the desired DWMBA
	<3:0>	VAXBI node number
GPR2	<15:0>	Remote (HSC) node numbers, if the Boot/Node qualifier was specified
GPR3		Boot device unit number
GPR4		Reserved
GPR5		Software boot control flags
GPR6		Used by the boot primitive to pass information to the bootblock program
GRP7		Physical address of the CCA
GPR8		Reserved
GPR9		Reserved
GPR10		The halt PC
GPR11		The halt PSL
AP		The halt code
FP		Reserved
SP		Address of the 256-Kbyte block of good memory + 512

3.10 Interprocessor Communication through the Console Program

Each CPU of a multiprocessor system must communicate with the other CPUs and the operating system. This section describes the interprocessor communication for a VAX 6000–400 system.

The console program runs on each processor of a multiprocessor VAX 6000–400 system. These copies of the console program must be able to communicate with each other and with the operating system.

When two processors needing to communicate are running, that is, not in console mode, the communications take place using mechanisms provided by the operating system. When one, or both, of the processors is in console mode, communications take place using a shared data structure called the console communications area (CCA).

The boot processor (BP) controls the console terminal and, therefore, most of the communication in the VAX 6000–400. There is no communication between secondary (nonboot) processors.

3.10.1 Required Communications Paths

A processor can be in one of four communication states: a running BP, a BP in console mode, a running nonboot processor, or a nonboot processor in console mode. The following are the communication paths:

1 Running processor to running processor, independent of boot or nonboot.

  The console program is not involved. The processors are supported by the communications mechanisms within the operating system. These paths are used even when the communication is related to the console program. For example, when the system time is modified, the new time must be stored in the time-of-year clock on each processor. The operating system uses its own method to examine or propagate this information.

  A special case of communications on these paths involves the XDELTA system debugger when it is entered on a nonboot processor. The operating system is responsible for passing characters to and from the boot processor and, thus, to the console terminal.

2 Running boot processor console program to/from nonboot processor console program.

The operating system on the BP must send complete console commands to the nonboot console, such as to start or stop the nonboot processor. The nonbooth console program must be able to send responses (human readable messages) to the operating system on the boot processor, such as when the nonboot processor encounters an error halt. The nonboot processor can send these responses at any time

The nonboot processor does not send commands to the boot processor, and the BP does not send responses to the nonboot processor.

3 Console mode BP to/from running nonboot processor.

Whenever the boot processor halts, the nonboot processors eventually wait for resources locked by the BP. The boot processor console supports receiving complete responses from the running nonboot processor.

4 Boot processor console to/from nonboot processor console — two different types of communication.

The boot console sends complete commands to the nonboot processor, allowing the BP console to update the copy of a parameter stored on each processor. An example of this type of communication is to synchronize the console terminal baud rate whenever it is changed on the BP. The nonboot consoles send complete responses to the BP console to report, for example, a processor halt. Since responses arrive complete, there are no interleaving messages on the console terminal.

The nonboot processor does not send commands, and the boot processor does not send responses.

The "Z" command allows the boot processor to communicate with VAXBI devices and, potentially, to nonprocessor XMI nodes. The consoles support character-at-a-time communications to implement the "Z" command, which transfers characters to and from another node so that the other node appears to be directly connected to the console terminal. The boot processor sends single characters of a command to the nonboot processor. The receiving nonboot processor performs all the processing of the input characters, including echoing and line editing. The nonboot processor sends single characters of a response to the BP for immediate display on the console terminal.

3.10.2 Console Communications Area

The console communications area (CCA) is the shared data structure in high physical memory used for communications between console programs. It consists of a one-page header followed by a variable number of pages containing buffers. The header contains status information that must be visible systemwide. The buffers, used for passing messages between processors, are allocated one set for each XMI node that could be in the system.

The CCA is initialized by the boot (primary) processor at system reset. It is allocated beginning on a page boundary from the highest addressed page of system memory that can be located by the boot processor. The header lies in the lowest addressed page of the CCA, followed by buffers.

The CCA is not initialized under any other console entry conditions (node reset or halts). The address of the CCA is obtained from the console state remaining in RSSC RAM.

Diagnostic tests that must test or reconfigure memory could overwrite the CCA. If this should happen, the diagnostic tests must observe the following conventions:

- The diagnostic tests can only be run from the BP.
- The diagnostic tests must force the nonboot processors to stop polling the CCA.
- The diagnostic tests must rebuild the CCA after completing testing.
- The nonboot processors must wait for a signal passed through the XGPR register before locating the new CCA.

The location of the CCA is passed to the operating system at bootstrap time through GPR7. During system initialization, each processor is triggered to search for the CCA. This search starts at the highest addressed memory that can be located by each processor and then works backward. If a processor cannot locate the CCA, it enters an endless loop and cannot participate in the system. The algorithm used by the console program to locate the existing CCA is as follows:

1 Next = highest memory address in system + 1 – 512.

2 If next < 0, then "Failed to find CCA."

3 If (next + CCA$L_BASE) <> next, then goto Step 7.

4 If (next + CCA$W_IDENT) <> "CC", then goto Step 7.

5 Compute sum of bytes at (next) through (next + CCA$B_CHKSUM – 1) ignoring overflow.

6 If sum = (next + CCA$B_CHKSUM), then "Exit with CCA found at next."

7 Next = next – 512.

8 Goto Step 2.

The overall layout of the CCA is shown in Figure 3–31. The contents of the fields are described in Table 3–19.

Figure 3–31 CCA Layout

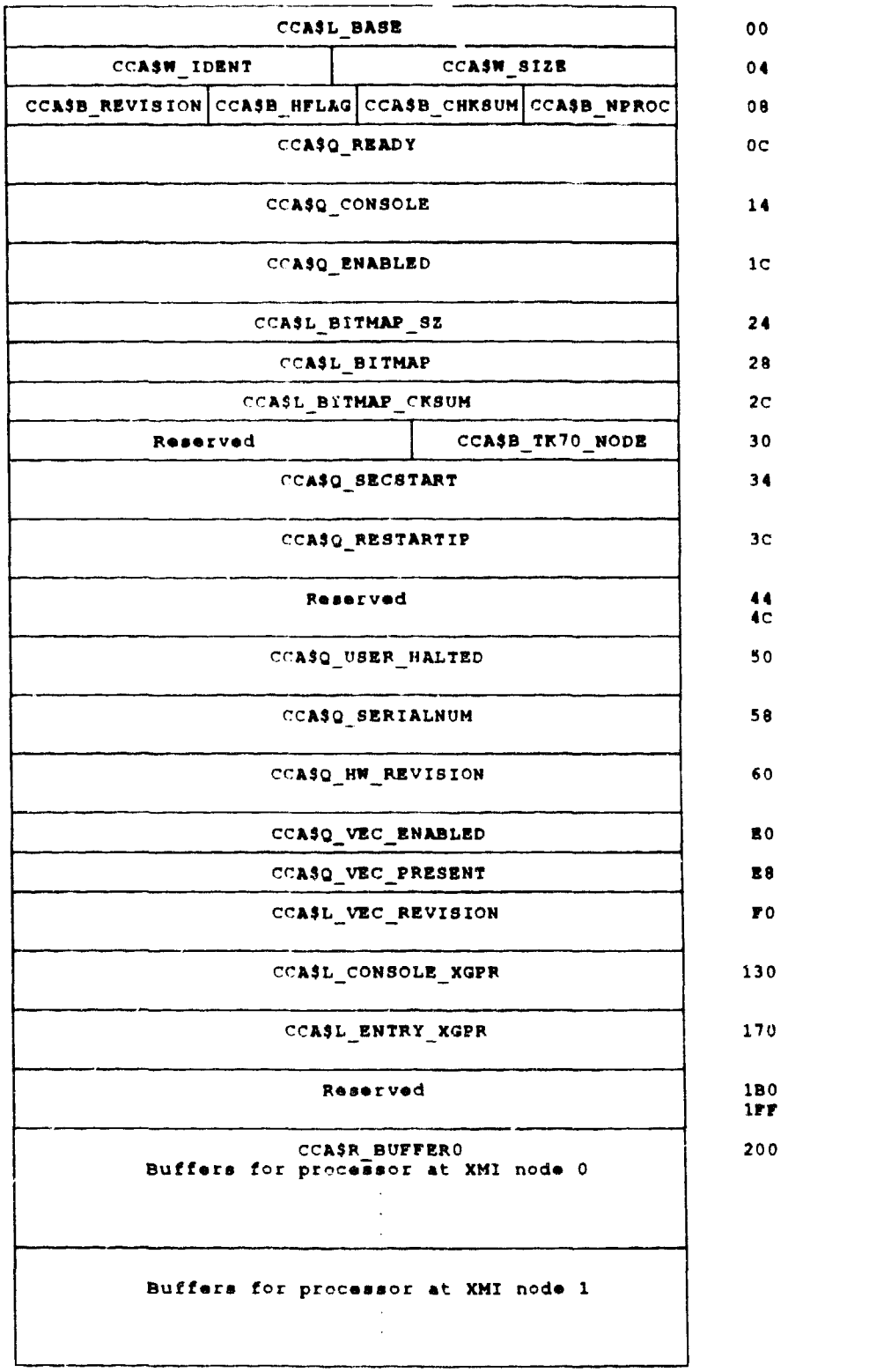
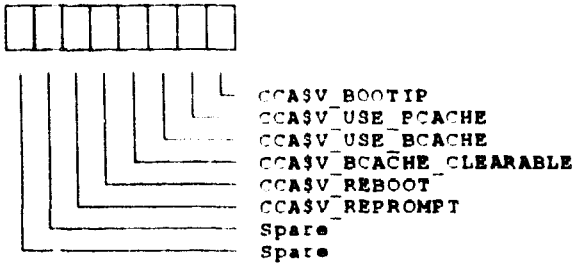


Table 3–19 CCA Fields

Field	Description
CCA$L_BASE	Physical address of the base of the CCA.
CCA$W_SIZE	The size, in bytes, of the CCA.
CCA$W_IDENT	The ASCII characters "CC".
CCA$B_NPROC	The number of processors supported by the CCA. The normal value is 16. The maximum value is 64, limited by the bitmasks in the other fields.
CCA$B_CHKSUM	Checksum of the first CCA$B_CHKSUM-1 bytes of the CCA. Computed by doing signed, byte addition, ignoring any overflow.
CCA$B_HFLAGS	Systemwide status flags:

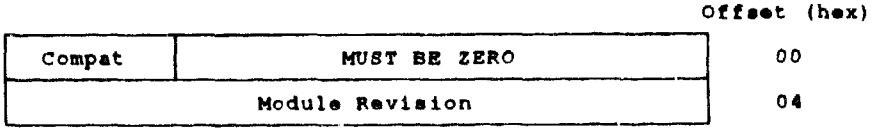


CCA$V_BOOTIP	When set, a bootstrap is being attempted. This prevents repeated attempts to bootstrap after a failure.
CCA$V_USE_ PCACHE	When set, the CPU-chip internal (primary) cache is to be enabled by the operating system. This bit is set based on a value in the EEPROM. The default setting is a one.
CCA$V_USE_ BCACHE	When set, the external (backup) cache is to be enabled by the operating system. This bit is set based on a value in the EEPROM. The default setting is a one.
CCA$V_BCACHE_ CLEARABLE	When set, the backup cache clear operation can be used successfully. This tells the operating system that error recovery requiring cache clears can be performed. This bit is set based on a value in the EEPROM. The default setting is a one.
CCA$V_REBOOT	This bit is tested whenever the console is entered as a result of a CTRL/P or a node halt. If the bit is set, the operating system is requesting a reboot. The system is rebooted from the default boot device. The front panel lower key switch does not inhibit such a reboot. This bit is ignored if the key switch is in the Secure position.
CCA$V_REPROMPT	This bit is used internally by the console program to support the SET CPU command.

CCA$B_REVISION	The revision number for the CCA.
CCA$Q_READY	A bitmask of the processors that have data posted in their transmit buffer for processing by the boot processor. This field allows the operating system to use a Find First Set (FFS) instruction to locate any pending messages. The bits and nodes are numbered, starting with zero.
CCA$Q_CONSOLE	A bitmask indicating the processors known to be in console mode. The appropriate bit is set and cleared by each processor as it enters and leaves console mode.

Table 3–19 (Cont.) CCA Fields

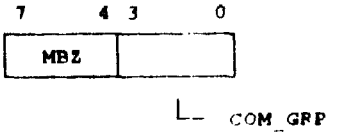
Field	Description
CCA$Q_ENABLED	A bitmask indicating which processors are enabled to leave console mode. A processor sets or clears its bit during console initialization, based on a bit stored in the EEPROM. The EEPROM bit is set with the SET CPU command.
CCA$L_BITMAP_SZ	The size, in bytes, of the physical memory bitmap. The bitmap is always an even number of longwords in length.
CCA$L_BITMAP	The physical address of the physical memory bitmap. The bitmap contains one bit for each page of physical memory present on the system. The bit is clear if the page contains a hard error or if the page is in use by the bitmap or CCA. The bitmap is always page aligned.
CCA$L_BITMAP_ CKSUM	Reserved; not used.
CCA$B_TK70_NODE	This field is used to pass to the operating system the XMI (in bits<7:4>) and VAXBI (in bits<3:0>) node numbers of the adapter that controls the TK tape drive. This field is set initially from a value stored in the EEPROM. If the initially specified node does not contain a TK tape adapter, the console program searches each VAXBI for a suitable adapter. The search starts with the highest XMI and from the highest VAXBI node ID on each VAXBI. The field sets to the location of the adapter, or zero if no adapter is found.
CCA$Q_SECSTART	A bitmask indicating which processors are currently being started by the boot processor. The console program uses this information to avoid repeatedly forcing a bootstrap. This field is set and cleared by the operating system.
CCA$Q_RESTARTIP	A bitmask indicating which processors are currently attempting restarts. Multiple flags are maintained to allow simultaneous error restarts to be performed. The operating system clears these fields if restart or boot succeeds.
CCA$Q_USER_HALTED	A bitmask indicating which processors entered console mode as a result of user intervention (CTRL/P or STOP command). This information allows the operating sytem to make decisions about timeouts in a symmetric multiprocessing configuration.
CCA$Q_SERIALNUM	The system serial number. This field is set to the least significant eight characters of the serial number string stored in the EEPROM.
CCA$Q_HW_REVISION	Consists of a 16-quadword array containing compatibility and module revision information for the processors. Module revisions are an ASCII string. The quadword is zero for nonprocessor nodes. The layout of each quadword is:



The reserved area is filled in from the module-specific area of the EEPROM. The contents of this area (with the exception of the module serial number and module revision) are filled with blanks. The reserved area in the CCA field is also filled with blanks. Anything in the EEPROM will be written into the CCA area. The Module Revision field contains a four-character ASCII representation of the module revision. If the revision is a single alphabetic character, the string begins with a leading blank. The alphabetic part of the revision is used to set the revision field of the CPU Device Register. These fields are set based on values stored in the EEPROM. If the EEPROM is unusable, the chip revisions are set to 30 (hex) and the Module Revision is zero.

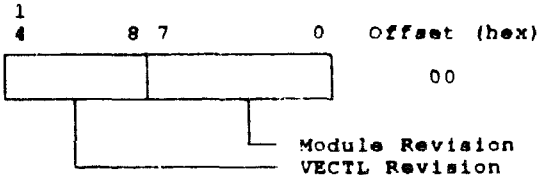
Table 3–19 (Cont.) CCA Fields

Field	Description
	The layout of the Compat field is:



COM_GRP	Compatibility Group. This binary field is used by the operating system to determine if all processors in the system are hardware compatible. Any processors not in the same group as the boot processor are not started.
Module Revision	A four-character ASCII representation of the module revision. If the revision is only a single alphabetic character, the string begins with a leading blank, such as " B02." The alphabetic part of the revision is used to set the revision field of the CPU's XDEV<DREV> field. This field is set based on values stored in the EEPROM. If the EEPROM is unusable, the Module Revision is zero.

CCA$Q_VEC_ENABLED	A bitmask indicating which processors have enabled vector processors. A processor sets or clears its bit during console initialization, based on a bit in the EEPROM. The EEPROM bit is set with the SET CPU/VECTOR_ENABLE command.
CCA$Q_VEC_ PRESENT	A bitmask indicating which processors have working vector processors. A vector processor is working if it passes self-test. During console initialization a scalar processor sets its bit if it has a working vector processor attached.
CCA$L_VEC_REVISION	An array of 16 longwords containing vector module revision information, which is obtained from the Module Revision Register on the vector module. The Module Revision and VECTL Revision are indicated as follows:



CCA$L_CONSOLE_ XGPR	An array of 16 longwords containing the XGPR value used while the console is operating.
CCA$L_VEC_ENTRY	An array of 16 longwords containing the XGPR value used by system software before console entry.

The CCA contains a buffer area for each possible XMI node. Each buffer area contains fields to support both message-oriented and character-at-a-time communications.

The address of the buffer area for XMI node *n* is given by:

$$Buffer_n = \text{Base address of CCA} + 512 + (n * 168)$$

The layout of the buffer area is shown in Figure 3–32, and the buffer fields are described in Table 3–20

Figure 3–32 Layout of XMI Node Buffers

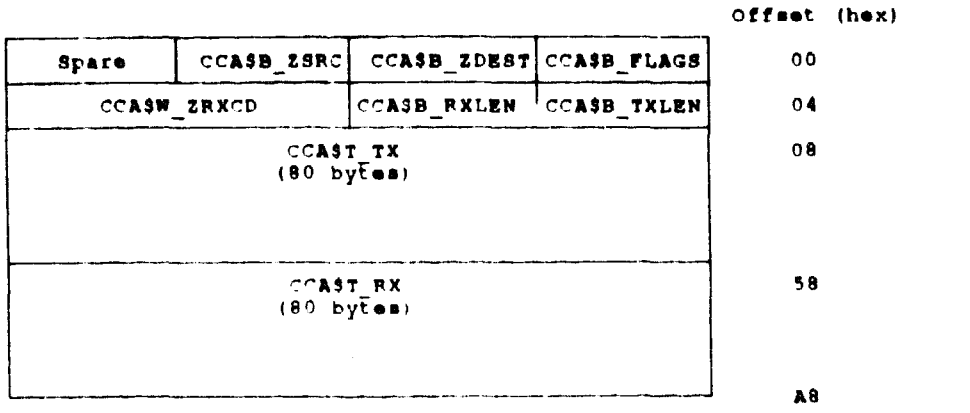
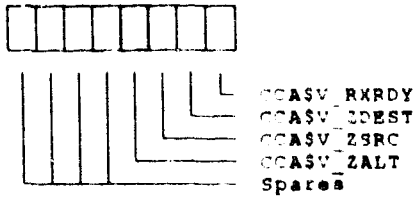


Table 3–20 Buffer Fields

Field	Description
CCA$B_FLAGS	Status flags:

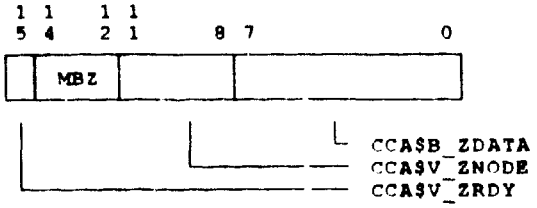


CCA$V_RXRDY	When set, there is a complete message in the CCA$T_RX buffer. The equivalent bit for CCA$T_TX is in CCA$Q_READY of the CCA header.
CCA$V_ZDEST	When set, this node is sending "Z" command data to the node listed in CCA$B_ZDEST.
CCA$V_ZSRC	When set, this node is receiving "Z" command data from the node listed in CCA$B_ZSRC. This bit is always set or cleared by the node originating the "Z" command.
CCA$V_ZALT	When set, the target of the current "Z" command cannot communicate through the CCA. The target is either a non-processor XMI node or a VAXBI node and must be accessed using alternate RXCD protocol, as described in the *VAXBI System Reference Manual*.

CCA$B_ZDEST	When CCA$V_ZDEST is set, this field contains the XMI node number of the node receiving the "Z" command data that this node is sending. If the low four bits of this field identify a node that is a DWMBA, the high order four bits contain the destination VAXBI node number.

Table 3–20 (Cont.) Buffer Fields

Field	Description
CCA$B_ZSRC	If CCA$V_ZSRC is set, this field contains the XMI node number of the node transmitting "Z" command data to this node.
CCA$B_TXLEN	If the bit corresponding to this node is set in CCA$Q_READY, then this field contains the length, in bytes, of the message in CCA$T_TX.
CCA$B_RXLEN	If CCA$V_RXRDY is set in CCA$B_FLAGS, then this field contains the length, in bytes, of the message in CCA$T_RX.
CCA$W_ZRXCD	This field is used for character-at-a-time communication in the same manner as a VAXBI RXCD Register. The layout is:



CCA$B_ZDATA	When CCA$V_ZRDY is set, this field contains one byte of "Z" command data being sent to this node.
CCA$V_ZNODE	When CCA$V_ZRDY is set, this four-bit field contains the XMI node number of the node that transmitted the data in CCA$B_ZDATA.
CCA$V_ZRDY	When this bit is set, there is valid data in the other CCA$W_ZRXCD fields.

CCA$T_TX	This buffer is used by the node to transmit a response to the BP. Only response data is passed through this buffer since a nonboot processor does not send commands to the boot processor.
CCA$T_RX	This buffer is used by the node to receive a command from the boot processor. Only command data is passed through this buffer since a nonboot processor does not receive responses from the BP. Commands must end with a carriage return.

3.10.3 Sending a Message to Another Processor

The following two examples show how the CCA is manipulated when a complete message is sent between two processors.

For the first example, the boot processor, located at XMI node 1, sends a START command to the nonboot processor, located at XMI node 4.

1 Node 1 examines the CCA$V_RXRDY bit in the CCA buffer area for node 4. If the bit is clear, then goto Step 3.

2 Node 1 polls the bit until it clears or until a timeout of 12 seconds is reached. If a timeout occurs, an error is reported.

3 Node 1 moves the text of the START command into the CCA$T_RX buffer for node 4.

4 Node 1 sets the length of the command into the CCA$B_RXLEN field for node 4.

5 Node 1 sets the CCA$V_RXRDY bit for node 4 to indicate that a command is waiting.

6 Whenever node 4 enters its main console loop, it will eventually check for commands to execute. It will examine its local command buffer and then check its CCA$V_RXRDY bit for a command from another node.

7 Node 4 will now process the command contained in its CCA$T_RX buffer.

8 After reading the command, node 4 then clears its CCA$V_RXRDY bit, indicating that the buffer is again available.

For the second example, the nonboot processor, which is located at XMI node 4, halts, enters console mode, and sends a "halted" message to the boot processor, located at XMI node 1.

1 Node 4 examines bit 4 of the CCA$Q_READY field. If the bit is clear, then goto Step 3.

2 Node 4 polls this bit until it clears.

3 Node 4 moves the text of its response into its CCA$T_TX buffer.

4 Node 4 sets the length of the response in its CCA$B_TXLEN field.

5 Node 4 sets bit 4 in CCA$Q_READY to indicates that a response is waiting.

6 Node 4 issues an IVINTR interrupt to node 1. If node 1 is running, this alerts the operating system that a response is waiting. Node 4 polls CCA$Q_READY until bit 4 clears, preventing the nonboot processor from performing any action that might cause the response to be lost before the BP can display it.

7 If node 1 is running, it responds to the IVINTR and eventually checks for console responses, using an FFS instruction to check CCA$Q_READY. If node 1 was in console mode, it would be polling CCA$Q_READY and discover bit 4 set.

8 Node 1 (either the operating system or the console program) processes the response from the CCA$T_TX buffer for node 4. If the console program is running, it displays the response on the console terminal.

9 Node 1 clears bit 4 in CCA$Q_READY, indicating that the buffer is again available.

3.11 Error Handling

This section describes the system-specific error exceptions and interrupts that enable the operating system interface macrocode programmer to determine the cause of the error, console HALT code, or interrupt/exception. This section is organized around the SCB entry points (vectors pointing to service routines) through which all error notifications pass. Recommendations are offered for error recovery strategies.

This section contains the following information:

- An overview of error detection and reporting.
- How to determine what error(s) happened, given the SCB entry point through which the error was dispatched.
- What parameters are pushed on the stack.
- What the failure codes are for halts and machine checks.
- What information exists for each error.
- How to clean up the error once its cause has been determined.
- How to restore the state of the machine, and what level of recovery is possible.

Table 3–21 lists the internally generated system control block (SCB) entry points. The complete list of supported SCB vectors is in Section 3.3.5. Table 3–22 describes the levels of hardware-detected errors by level of severity. Table 3–23 lists the categories of errors, organized by entry point.

Refer to these sections:

- Section 3.3.5 for an explanation of the SCB.
- Section 3.3.4 for an explanation of exceptions and interrupts.

Table 3–21 CPU-Chip Internally Generated SCB Entry Points

Mnemonic	SCB Index (hex)	Description
SCB_MACHCHK[1]	004	Machine check
SCB_KSNV[1]	008	Kernel stack not valid
SCB_PWRFL[1]	00C	Power fail
SCB_RESPRIV	010	Reserved/privileged instruction
SCB_XFC	014	Extended function call (XFC) instruction
SCB_RESOP	018	Reserved operand
SCB_RESADD	01C	Reserved addressing mode
SCB_ACV	020	Access control violation
SCB_TNV	024	Translation not valid
SCB_TP	028	Trace pending
SCB_BPT	02C	Breakpoint trace fault
SCB_ARITH	034	Arithmetic fault
SCB_CHMK	040	Change mode to kernel
SCB_CHME	044	Change mode to executive
SCB_CHMS	048	Change mode to supervisor
SCB_CHMU	04C	Change mode to user
SCB_SMERR[1]	054	Soft error interrupt
SCB_HMERR[1]	060	Hard error interrupt
SCB_VECT_DISABLED	068	Vector module disabled exception
SCB_IPLSOFT	080 – 0BC	Software interrupt levels
SCB_INTTIM	0C0	Interval timer interrupt
SCB_EMULATE	0C8	Emulated instruction trap (PSL<FPD>=0)
SCB_EMULFPD	0CC	Emulated instruction trap (PSL<FPD>=1)

[1]This section describes the entry-point vector in detail

Table 3–22 Hardware-Detected Errors

Error	Description
Console halt	A halt to console mode is caused by one of the errors listed in Table 3–8. For some halt conditions, the console prompts for a command and waits for operator input. For other halt conditions, the console attempts a system restart or a system bootstrap.
Machine check	A hardware error occurred synchronously with the CPU-chip's execution of instructions. Instruction-level recovery and retry may be possible.
Kernel stack not valid	During exception processing, a memory management exception was encountered while trying to push information on the kernel stack.
Power fail	The power supply asserted the power fail signal XCI AC LO L. Software has 4 milliseconds to save processor state.
Soft error interrupt	A hardware error occurred that was not fatal to the process or system. System error software should be able to recover and continue.
Hard error interrupt	A hardware error occurred asynchronously with the CPU-chip's execution of instructions. Instruction-level recovery and retry may be possible.

Table 3–23 Error Summary Based on Notification Entry Points

SCB Index	Entry Point	Error Categories
N/A	Console halt	Interrupt stack not valid, kernel-mode halt, double error, illegal SCB vector, node halt (XBER<NHALT>=1), CTRL/P, node reset
04	Machine check	Floating-point processor related
		Memory management, interrupt, microcode/CPU errors
		Primary cache read error
		Tag parity errors
		Data parity errors
		DAL error on memory read
		DAL data parity errors
		B-cache RAMs
		REXMI-to-CPU-chip parity error
		ERR L signal asserted
		XBER-notified errors
		RSSC DAL timeout
		DAL error on memory write or flush write buffers
		ERR L signal asserted
		RSSC DAL timeout
		Vector module errors
08	Kernel stack not valid	
0C	Power fail	
54	Soft error interrupt	P-cache errors
		REXMI-detected errors
		C-chip tag store parity error
		C-chip detected errors
		Vector module errors
60	Hard error interrupt	XBER-notified errors
		RCSR<WDPE> (DAL write data parity error)
		RCSR<SE> (Second error)
		Vector module errors

3.11.1 General Error Detection and Reporting Characteristics

The following sections describe the general operation of error handling and recovery not associated with specific errors.

3.11.1.1 Primary Cache Error Handling

A P-cache error is reported as either a soft error interrupt at IPL 1A (hex) or as a machine check. Errors reported as interrupts set PCSTS<5>, Interrupt. PCSTS<12:8> describe the error. However, if either the interrupt bit or PCSTS<7>, Trap1, is already set when the error is detected, bits<12:8> are not loaded so that they continue to reflect the first error detected.

Errors reported as a machine check set Trap1, load bits<12:8>, and load PCERR. However, if Trap1 is already set when an error is detected, Trap2 sets and bits<12:8> and PCERR remain latched to the first error. Since errors reported as a machine check are more important than errors reported as interrupts, the state corresponding to the machine check overwrites the state latched from a previous interrupt.

Whenever Trap1, Trap2, or Interrupt sets, the P-cache is automatically disabled.

Table 3–24 describes the P-cache errors.

Table 3–24 Primary Cache Errors

Error	Description
P-cache Tag Parity Error	Reported when detected during a read, write, or invalidate reference and ENABLE PTS=1, Trap1=0, Trap2=0, Interrupt=0. These errors are always reported as interrupts, but if the reference was a D-stream read that hit, the error is also reported as a machine check.
P-cache Data Parity Error	Reported when detected during a read reference and ENABLE PTS=1, Trap1=0, Trap2=0, Interrupt=0, Force Hit=0. These errors are reported as interrupts if the reference was an I-stream read or as a machine check if the reference was a D-stream read that hit.
DAL Data Parity Error	Reported when detected during a non-I/O space read reference that missed in the P-cache. These errors are reported as a machine check when detected on the requested longword of a D-stream read. When the error was detected on the nonrequested longword of a D-stream read or on either longword of an I-stream read, the error is reported as an interrupt.
DAL Bus Error	Reported if a read, write, or clear write buffer command is terminated with the DAL ERR L signal asserted. If detected during a D-stream read, write, or clear write buffer command, the error is reported as a machine check. If detected during an I-stream read, it is reported as an interrupt.
F-chip Result Parity Error	Reported if a data parity error is detected during a result transfer from the F-chip. Always reported as a machine check.

3.11.1.2 Primary Cache Error Recovery

Whenever an error is detected, the P-cache latches the state describing the error in PCSTS and PCERR. The P-cache then automatically disables. Error recovery consists of reading the error information, taking the appropriate action to correct the error, and clearing the lock bits in PCSTS.

The information read in PCSTS and PCERR, with the reporting method of either an interrupt or a machine check, indicates the type of error.

If the error was a tag parity error, the entire tag store is written with invalid tags with good parity. PCERR contains the address of the tag in error only if the tag parity error is reported as a machine check. For all other errors, the P-cache flushes by writing a one to PCSTS<2>, Flush Cache.

The primary tag store in the C-chip is then flushed, the error bits cleared in PCSTS, and the P-cache reenabled. If the error rate is such that the P-cache is to remain disabled, the primary tag store in the C-chip is also disabled.

3.11.1.3 C-Chip Error Handling

Three types of errors are detected by the C-chip:

- Internal C-chip parity errors, including parity errors in the backup cache tag store and in the C-chip's copy of the primary cache tag store
- DAL protocol errors, including invalid sequencing of DAL control signals
- Vector module errors

3.11.1.3.1 Backup Tag Store Parity Errors

The C-chip calculates one bit of odd parity on the tag when it writes the backup tag store, except when the tag, valid bits, and parity are written explicitly using an IPR write. When the tag store is read, parity is recalculated on the tag and this calculated parity bit is compared with the parity bit that was read from the tag store.

For memory read transactions, the C-chip asserts BC HIT L for a hit in the backup cache. If a parity error is discovered, the C-chip prevents the BC HIT L signal from being asserted. Parity generation on the tag that is read from the tag store is not ready by the time the C-chip needs to assert BC HIT L, so a predicted parity scheme is used. The predicted parity generator calculates odd parity on the incoming address bits<28:17>. When the tag store is read, this predicted parity bit is compared to the parity bit that was read from the tag store and, if there is no match, BC HIT L is not asserted. The predicted parity bit is also loaded into the status latch to allow testing of the predicted parity generator.

Odd parity is then calculated on the tag that was actually read from the tag store by a second parity generator, the actual parity generator. This result is compared to the parity bit that was read from the tag store and is also loaded into the status latch. If a parity error is detected, the SERR IRQ L signal is asserted and the Status Lock bit (BCSTS<0>) is set, causing both tag stores to be disabled.

Backup cache tag store parity errors are detected and reported during memory reads, memory writes, cache fills, invalidates, and Inval-Bus requests. An error is not reported if either Status Lock or FORCE BHIT (BCCTL<0>) is set or if ENABLE BTS (BCCTL<1>) is not set.

3.11.1.3.2 Parity Errors of the C-Chip's Copy of the Primary Cache Tag Store

The C-chip calculates one bit of odd parity on the tag when it writes the primary cache tag store copy, except when the tag, valid bit, and parity bit are written explicitly using an IPR write. When the tag store is read, parity is calculated on the tag and this calculated parity bit is compared to the parity bit read from the tag store. If a parity error is discovered, the CPU-chip is notified through assertion of the SERR IRQ L signal and setting the Status Lock bit (BCSTS<0>), disabling both tag stores.

Primary cache tag store copy errors are detected and reported during cache fills, invalidates, and Inval-Bus requests. An error is not reported if ENABLE PTS (BCCTL<2>) is not set or if Status Lock is set.

3.11.1.3.3 DAL Protocol Errors

DAL-related errors cause the backup cache's RAMs to be corrupted. When the C-chip detects a DAL-related error, the BUS ERR bit (BCSTS<4>) and the Status Lock bit (BCSTS<0>) set, disabling both tag stores.

The C-chip detects DAL protocol errors only when ENABLE BTS (BCCTL<1>) is set, because cache RAM data corruption is relevant only when the backup cache is enabled.

The C-chip recognizes DAL errors during the transactions described in Table 3–25.

Table 3–25 Transactions on the DAL with C-Chip Error Detection

Transaction	Description
Read Miss	The memory interface sends data back to the cache RAMs and the CPU-chip during a read miss operation. If the memory interface terminates the cycle with ERR_L asserted, the C-chip asserts SERR IRQ L and disables both tag stores by setting Status Lock (BCSTS<0>). Cache data was not returned properly and the data in the backup cache RAMs was corrupted. If the read miss was to an I/O space address, the C-chip does not detect any DAL errors and does not assert SERR IRQ L.
Cache Fill	When the Two-Cycle RAMs bit (BCCTL<4>) is set during the cache fill, the memory interface delays the deassertion of the AS L signal for an additional cycle to the normal cache fill timing. This allows increased write time of the RAMs. If AS L is deasserted prematurely, the C-chip asserts SERR IRQ L and disables both tag stores by setting the Status Lock bit.
Memory Write	When the Two-Cycle RAMs bit is set during the memory write transaction and RDY L asserts prematurely, the C-chip asserts SERR IRQ L and disables both tag stores by setting the Status Lock bit, whether or not the write address produces a hit in the backup cache tag store. The memory interface, which is responsible for asserting RDY L in this situation, delays the assertion of RDY L to accommodate the increased writing time of the RAMs. If the write address did not produce a hit in the backup cache tag store, the C-chip does not write the cache RAMs and no data is corrupted.
Memory Write	The CPU-chip sends data to the cache RAMs and to the memory interface during the memory write operation. If the memory interface terminates the transactions with ERR L asserted, the C-chip asserts SERR IRQ L and disables both tag stores by setting the Status Lock bit, whether or not the write address produces a hit in the backup tag store. If the memory interface terminates the transaction with ERR L asserted, bad data might have been transferred on the bus and written into the cache RAMs. Data is not corrupted when the write address does not produce a hit in the backup cache tag store because the C-chip does not write the cache RAMs.

3.11.1.3.4 Vector Module Errors

The C-chip's vector interface retries all vector interface bus errors once. Recoverable errors are reported as soft error interrupts. Unrecoverable errors are reported by terminating the DAL transaction with ERR L asserted or by requesting a hard error interrupt.

3.11.1.4 C-Chip Error Recovery

When a C-chip error is detected, BCSTS and BCERR are read to determine the state of the C-chip at the time of the error. BCSTS and BCERR must be read before Status Lock <BCSTS<0>) is cleared.

If the error occurred in the backup cache tag store, the tag store entry that contained the error must be corrected and both tag stores must be flushed. If the error occurred in either half of the primary cache tag store copy, the entry that contained the error must be written with correct parity and both tag stores must be flushed. If the BUS ERR bit (BCSTS<4>) is set, the error was a DAL protocol error and both tag stores must be flushed.

Whenever the C-chip's copy of the primary cache tag store is flushed, the CPU-chip's P-cache must also be flushed to ensure consistency between the two.

To guarantee correct data and error-free results, the caches are first flushed and Status Lock is then cleared to reenable both tag stores.

If BCCTL is to be written, it is done before Status Lock is cleared. For example, a cache that is in error must be disabled before Status Lock is cleared. Otherwise, it would be possible for another error to occur immediately.

The recommended sequence to bring the C-chip back to normal operation is as follows:

1 Read the status register.

2 Read the error address register.

3 If the error was a tag parity error:

   a. Write BCIDX with the address taken from BCERR.

   b. Write that tag store location with tag=0 (arbitrarily chosen), parity=1 (odd parity for tag chosen), valid=0. Repeat for each tag store that produced an error.

4 Read and write BCCTL, if necessary.

5 Flush both tag stores.

6 Flush the CPU-chip's P-cache.

7 Clear the Status Lock bit.

3.11.1.5 REXMI Error Handling

The REXMI implements the error detection and logging required for the XMI protocol. When an error is detected, the command, address, and error status are logged in XBER, RCSR, and XFADR. If retry is enabled, the normal state, the REXMI retries all NO ACKed command and write data cycles until the timeout counter expires.

The method of error reporting is a function of the type of error detected and the transaction that was in progress at the time. Errors are reported either as on the DAL command, or as a hard error interrupt at IPL 1D (hex), or as a soft error interrupt at IPL 1A (hex). Table 3–26 lists the REXMI errors.

Table 3–26 REXMI Errors

Error	Reported By	Indicated By
Read errors in the requested quadword	Machine check	XBER<RSE> XBER<RER> XBER<TTO> XBER<NRR> XBER<CNAK>
Read errors in the nonrequested quadword	Soft error interrupt	XBER<RSE> XBER<RER> XBER<TTO> XBER<NRR> XBER<CNAK> RCSR<CFE>
Write errors	Hard error interrupt	XBER<TTO> XBER<CNAK> XBER<WDNAK>
DAL write data parity errors	Hard error interrupt	RCSR<WDPE>
IDENT errors	Hard error interrupt	XBER<RSE> XBER<RER> XBER<TTO> XBER<NRR> XBER<CNAK>
Assertion of the XMI FAULT L signal	Hard error interrupt	XBER<XFAULT>
Receipt of a write error IVINTR	Hard error interrupt	XBER<WEI>
XMI parity errors	Soft error interrupt	XBER<PE>
XMI parity errors when the cycle is subsequently ACKed by another node	Hard error interrupt	XBER<IPE>
Corrected read data	Soft error interrupt (if not disabled by RSCR<CRDID>)	XBER<CRD>
Corrected confirmation	Soft error interrupt (if not disabled by RSCR<CCID>)	XBER<CC>

Software uses the REXMI error summary bits XBER<ES> (general XMI errors) and XBER<NSES> (REXMI node-specific errors) to determine if the REXMI is the source of any errors.

All XMI command/address transfers are reattempted until acknowledged or a transaction timeout (XBER<TTO>) occurs. All XMI write data transactions are reattempted until acknowledged or a transaction timeout occurs.

All XMI memory writes are disconnected, that is, they are acknowledged by the REXMI and the data is placed in the write buffer to be written later. If a subsequent write buffer unload or purge results in an XMI memory write failure, it is signaled to the CPU-chip by posting a hard error interrupt.

All XMI I/O space reads and writes are "connected." They cause purging of the write buffer prior to their initiation on the XMI and they are not acknowledged until all XMI transactions are successfully completed. If the write buffer purge results in an XMI memory write failure, the I/O transaction is allowed to complete and the error is reported by posting a hard error interrupt. If the write buffer purge is successful, but the subsequent I/O transaction fails to complete, the CPU-chip is released with a read error response or a write DAL Ready command (RDY)/hard error interrupt.

All XMI memory reads are "connected." The CPU-chip waits for all demand-requested data to be returned. If the XMI cannot deliver the data, the failure is signaled by a machine check. Failures to deliver non-demand data, such as cache fill data, result in a soft error. A memory read hit in the write buffer causes the write buffer to be purged. If the purge results in an XMI memory write failure, the read is allowed to complete and the error is reported by posting a hard error interrupt.

Soft errors are signaled by posting a soft error interrupt while hard errors are signaled by posting a hard error interrupt or by using the DAL terminator, the ERR L signal line assertion.

XMI IVINTR transactions are treated like I/O writes but XMI IDENT transactions that end in an error are signaled by the posting of a hard error interrupt.

The XMI interface maintains complete error status on a failed XMI transaction that was initiated by every node. This status includes the failed command, commander ID, address, and an error bit that indicates the type of error that occurred. This status remains latched until software resets the error bit(s).

The XMI supports three parity bits, covering both data and command information. The REXMI generates and checks XMI parity. Both the XCA-chip and the low order XDP-chip check parity across the XCI D<31:0> lines.

The REXMI checks parity on every XMI cycle. If a parity error is detected, a soft error interrupt is posted. However, if the XMI operation is a cycle where the REXMI participates, such as read data, the REXMI either posts a hard error interrupt or asserts the DAL terminator, the ERR L signal, causing a machine check. In general, read-type transactions are terminated with a machine check, and write-type transactions are posted with a hard error interrupt if an error is detected.

REXMI cache fill operations involve two transactions: the first quadword ("requested") and the second quadword ("fill"). Errors are reported differently, depending on whether the error occurred during the first or second quadword. If the error occurs during the first quadword, the error notification is by asserting the ERR L line; if during the second, a soft error interrupt is posted.

The REXMI generates parity on read data being driven by the REXMI onto the DAL bus. The REXMI detects DAL parity errors, one parity bit per data byte across the 64 bits. Table 3–27 shows the REXMI parity coverage.

Table 3–27 REXMI Parity Coverage

Side of REXMI	Number of Parity Bits	Scope	Generate	Detect Error	If Detected, Then
XMI	3 total		Yes	Yes	
	1	Low 32 data bits			
	1	High 32 data bits			
	1	Control bits			
		REXMI passive			Soft error interrupt
		REXMI active Write			Hard error interrupt
		REXMI active Read			ERR L
DAL	8 total	1 per data byte	Yes	Yes	Hard error interrupt

The DAL terminator, the ERR L signal, causes a microcoded-evoked machine check (MCHK_BUSERR_READ_DAL, MCHK_BUSERR_WRITE_DAL). These are the only nonparity DAL-related errors that are detected.

D-stream read, write, and clear write buffer transactions always cause machine checks. IPR and read interrupt vector transactions are handled by microcode.

Severe hardware errors, such as when the DAL is being driven by a source other than the CPU-chip and the CPU-chip is waiting for a response, might cause chip short-circuit damage. There is no mechanism available to avoid this type of bus driver conflict and return to synchronization.

The F-chip reports all errors through protocol signals on the bus between the CPU-chip and the F-chip. Certain severe protocol errors between the CPU-chip and the F-chip are handled through a special mechanism involving discharging resistors on the control lines to time out to an illegal state. This causes a machine check and can also be used for an indication that the F-chip is present.

3.11.1.6 Parity Generation and Detection

Parity generation and check characteristics of the KA64A CPU module are as follows:

- The CPU-chip generates parity on write data and checks parity on memory transaction read data. The CPU-chip does not generate parity on command/address information.
- A CPU-chip I-stream parity error is reported as an interrupt with the appropriate bits set in the P-cache status register. If the error is hard and a D-stream read results in a machine check (MCH_BUSERR_READ_DAL), the microcode tries to recover.
- The CPU-chip's primary cache supports parity on both the tag and data stores.
- The backup cache supports parity on both the tag and data stores. On cache fills and writes, parity is stored and then checked by the CPU-chip during reads.
- The REXMI detects DAL parity errors on memory writes.
- The REXMI generates and checks the three XMI parity bits, which cover both data and command information.
- The F-chip generates parity for F-chip results and checks parity on D_BUS floating operands.
- The RSSC does not support parity so the internal battery-backed-up one Kbyte of RAM is not protected, nor are any of the RSSC internal registers.
- The REXMI CSR registers are not parity protected.
- The C-chip's vector interface retries all vector interface bus errors once. Recoverable errors are reported as soft error interrupts. Unrecoverable errors are reported as hard error interrupts or as machine checks.

3.11.1.7 Microcode-Detected Errors

Errors detected by microcode checks that result in console halts and machine checks are listed in Table 3–28

Table 3–28 Microcode-Detected Errors

Code (hex)	Mnemonic	Description
Console Halts		
02	ERR_HLTPIN	HALT_L asserted (CTRL/P, break, or external halt)
03	ERR_PWRUP	Initial power-up
04	ERR_INTSTK	Interrupt stack not valid during exception processing
05	ERR_DOUBLE	Machine check during exception processing
06	ERR_HLTINS	HALT instruction executed in kernel mode
07	ERR_ILLVEC	SCB vector bits<1:0> = 11
08	ERR_WCSVEC	SCB vector bits<1:0> = 10
0A	ERR_CHMFI	CHM*x* instruction executed while on interrupt stack
10	ERR_MCHK_ACV_TNV	ACV/TNV during machine check processing
11	ERR_KCHK_ACV_TNV	ACV/TNV during kernel-stack-not-valid
12	ERR_MCHK_MCHK	Machine check during machine check processing
13	ERR_KSNV_MCHK	Machine check during kernel-stack-not-valid processing
19	ERR_IE_PSL26_24_101	PSL<26:24> = 101 during interrupt or exception
1A	ERR_IE_PSL26_24_110	PSL<26:24> = 110 during interrupt or exception
1B	ERR_IE_PSL26_24_111	PSL<26:24> = 111 during interrupt or exception
1D	ERR_REI_PSL26_24_101	PSL<26:24> = 101 during REI
1E	ERR_REI_PSL26_24_110	PSL<26:24> = 110 during REI
1F	ERR_REI_PSL26_24_111	PSL<26:24> = 111 during REI
3F	ERR_SELFTEST_FAILED	Microcoded power-up self-test failed in the CPU-chip
Machine Checks		
01	MCHK_FP_PROTOCOL_ERROR	Protocol error during F-chip operand/result transfer.
02	MCHK_FP_ILLEGAL_OPCODE	Illegal opcode detected by F-chip.
03	MCHK_FP_OPERAND_PARITY	Operand parity error detected by F-chip.
04	MCHK_FP_UNKNOWN_STATUS	Unknown status returned by F-chip.
05	MCHK_FP_RESULTS_PARITY	Returned F-chip result parity error.
08	MCHK_TBM_ACV_TNV	Translation buffer miss status generated in ACV/TNV microflow.
09	MCHK_TBH_ACV_TNV	Translation buffer hit status generated in ACV/TNV microflow.
0A	MCHK_INT_ID_VALUE	Undefined INT.ID value during interrupt service.
0B	MCHK_MOVC_STATUS	Undefined state bit combination in MOVCX.
0C	MCHK_UNKNOWN_IBOX_TRAP	Undefined trap code produced by the I-box (the CPU-chip's instruction fetch and decode unit).
0D	MCHK_UNKNOWN_CS_ADDR	Undefined control store address reached.

Table 3–28 (Cont.) Microcode-Detected Errors

Code (hex)	Mnemonic	Description
10	MCHK_BUSERR_READ_PCACHE	P-cache tag or data parity error during read.
11	MCHK_BUSERR_READ_DAL	DAL bus or data parity error during read.
12	MCHK_BUSERR_WRITE_DAL	DAL bus error on write or clear write buffer.
13	MCHK_UNKNOWN_BUSERR_ TRAP	Undefined bus error microtrap.
14	MCHK_VECTOR_STATUS	Vector module error.

Microcode checks also detect the Kernel Stack Not Valid Exception.

3.11.1.8 Self-Test-Detected Errors

There are two levels of self-test errors: power-up CPU-chip microcode self-test and boot-ROM macrocode self-test. A failing microcode self-test results in a console halt with code ERR_SELFTEST_FAILED. A failing macrocode self-test results in:

- The CPU module not deasserting the XMI signal XMI BAD L
- XBER<STF> not clearing
- The module's Self-Test Passed (STP) LED not turned on

3.11.2 Operating System (Macrocode) Error Handling and Recovery

All errors except those leading to a console halt go through SCB vector entry points and are handled by service routines provided by the operating system. A console halt, on the other hand, transfers macrocode execution control directly to the console entry point, 2004 0000.

Error handling and recovery is divided into these steps:

1 State collection

2 Analysis

3 Recovery

4 Retry

3.11.2.1 Error State Collection

All relevant state must be collected before error analysis can begin. The stack frame provides the PC/PSL pair for all exceptions and interrupts. For machine checks, the stack frame also provides details about the error.

Besides the stack frame, machine checks and hard and soft error interrupts usually require analysis of other registers. The state of these registers should be read and saved:

- REXMI Control and Status Register (RCSR)
- XMI Bus Error Register (XBER)
- XMI Failing Address Register (XFADR)
- RSSC Bus Timeout Control Register (SSCBTR)
- Backup Cache Control Register (BCCTL)
- Backup Cache Status Register (BCSTS)
- Backup Cache Error Address Register (BCERR)
- Backup Cache Backup Tag Store (BCBTS)
- Backup Cache Primary Cache 1 Tag Store (BCP1TS)
- Backup Cache Primary Cache 2 Tag Store (BCP2TS)
- Primary Cache Status Register (PCSTS)
- Primary Cache Error Address Register (PCERR)
- Primary Cache Tag Array Register (PCTAG)
- Vector Interface Error Status Register (VINTSR)

Pseudo-code examples assume that each of these registers is saved in a variable whose name is constructed by prepending "s_" to the register name. For example, the RCSR would be saved in the variable s_rcsr.

3.11.2.2 Error Analysis

The error condition is analyzed with the error state obtained during the collection process. See Section 3.11.4, Section 3.11.6, and Section 3.11.7 for guides to analyze machine checks, hard error interrupts, and soft error interrupts, respectively.

Errors detected in or by one of the caches usually result in the cache being automatically disabled. Error analysis and recovery for memory or cache-related errors should be performed with both caches disabled to minimize the possibility of nested errors. The primary cache must be disabled before the primary cache tag store copy in the C-chip to maintain cache coherency.

In some cases an error is reported in two ways. Such a case is P-cache tag parity errors for D-stream read hits, which are reported both as soft error interrupts and as machine checks. The machine check handler error recovery cleans up the error condition so that the error interrupt handler finds no error bits set. Software needs to handle such error interrupts when there is no apparent cause.

3.11.2.3 Error Recovery

Error recovery consists of clearing any latched error state and restoring the system to normal operation.

If full operation cannot be restored, it is necessary to disable the section of hardware that causes errors. Software can maintain error counts that can be compared against error thresholds on every error report. When the count per unit time exceeds the threshold, the hardware section should be disabled. For example, if a cache reports many errors, it should be disabled.

There are special considerations for recovery from cache or memory errors:

- Cache and memory error recovery is always done with both caches disabled:

  PCSTS := ENABLE_REFRESH + $\overline{\text{ENABLE\_PTS}}$ + $\overline{\text{FORCE\_HIT}}$;

  BCCTL := ENABLE_REFRESH + $\overline{\text{ENABLE\_PTS}}$ + $\overline{\text{ENABLE\_BTS}}$ + $\overline{\text{FORCE\_BHIT}}$ + (two – cycle rams state);

  To maintain cache coherency, the P-cache must be disabled before disabling the primary cache tag store copy in the C-chip. The refresh enable bit remains set.

- Error recovery is performed starting with the most distant component and then working toward the CPU. REXMI errors are to be processed first, followed by RSSC errors, C-chip errors, and P-cache errors.

- REXMI errors are cleared by writing the read/write-one-to-clear (R/W1C) bits in RCSR and XBER. The suggested procedure is to write the values saved during error state collection back to registers:

```
IF s_xber<NSES> THEN RCSR := s_rcsr;
XBER := s_xber AND NOT XBAD;
RCSR := s_rcsr;
```

CAUTION: The error bits in RCSR must be cleared before those in XBER to ensure correct operation of the error reporting logic. XBER<XBAD> must be cleared before writing it back to XBER to prevent asserting the XMI BAD L signal by this node if it is being asserted by another node.

- RSSC errors are cleared by writing the R/W1C bits in SSCBTR. The suggested procedure is to write the value saved during error state collection back to the register:

```
SSCBTR := s_sscbtr;
```

- C-chip backup tag store parity errors are recovered by rewriting the tag using the error address register, BCERR:

```
IF s_bcsts<BTS_PERR> THEN
    BEGIN
    BCIDX := s_bcerr;
    BCBTS := %x20000000;
    END;
```

C-chip primary cache tag store parity errors are recovered by rewriting the tag using BCERR:

```
IF s_bcsts<P1TS_PERR> THEN
    BEGIN
    BCIDX := s_bcerr;
    BCP1TS := %x20000000;
    END;
IF s_bcsts<P2TS_PERR> THEN
    BEGIN
    BCIDX := s_bcerr;
    BCP2TS := %x20000000;
    END;
```

C-chip errors are cleared by writing the R/W1C bits in BCSTS. The suggested procedure is to write the value saved during error state collection back to the register:

```
BCSTS := s_bcsts;
```

- P-cache tag parity errors are recovered by rewriting all tags:

```
IF s_pcsts<TAG_PARITY_ERROR> THEN
    FOR i := 0 to 255 DO
        BEGIN
        PCIDX := i * 8;
        PCTAG  = %x40000000;
        END;
```

P-cache errors are cleared as part of the reenabling of the cache process.

- The P-cache and the primary cache tag store copy in the C-chip must be in the same state. If the P-cache is disabled, the C-chip's copy of the primary cache tag store must also be disabled. The converse is also true.

  The required procedure to reenable both caches is:

```
BCFBTS := 0;
BCFPTS := 0;
BCCTL  = ENABLE_REFRESH+ENABLE_PTS+ENABLE_
BTS+(two-cycle rams state);
PCSTS  = s_pcsts OR (ENABLE_REFRESH+ENABLE_
PTS+FLUSH_CACHE) AND NOT FORCE_HIT;
```

  Either cache may be disabled by clearing the appropriate enable bits in BCCTL and PCSTS while performing the sequence shown above. It is important that BCCTL<ENABLE PTS> and PCSTS<ENABLE PTS> be kept in the same state.

System performance degrades if one or both caches are disabled.

3.11.2.4 Error Retry

Error retry is a function of the error type (machine check or error interrupt) and the error state. If a retry is to be attempted, the stack must be trimmed of all parameters except the PC/PSL pair for machine checks (error interrupts have no additional parameters on the stack). A Return from Exception or Interrupt (REI) instruction will then restart the instruction stream and retry the error. Some form of software loop control needs to be provided to limit the possibility of an error loop.

If a retry is not be be attempted, software must determine if the error was fatal to the current process, the processor, or the entire system and then take the appropriate action.

3.11.3 Console Halt and Halt Interrupt

A console halt is not an exception but is a transfer of control by the CPU-chip microcode directly into console program macrocode in the boot ROM at address 2004 0000 (hex). Console halts are initiated at power-up, by certain microcode-detected double-error conditions, and by assertion of a halt interrupt from the RSSC.

A halt interrupt is generated by either of two conditions:

- CTRL/P is typed on the (unsecured) console terminal
- XBER<NHALT>(Node Halt) is asserted

No exception stack frame is associated with a console halt but SAVPC and SAVPSL provide the necessary information, including the halt code, which is loaded into PSAPSL<13:8>.

Table 3–8 lists and describes the console halt codes.

3.11.4 Machine Check Exceptions

A machine check exception indicates a serious system error that is sometimes recoverable by restarting the instruction.

The recoverability is a function of the:

- Machine check code
- VAX Restart bit (R) in the machine check stack frame
- State of PSL<FPD>
- State of the Unlock Write Pending bit (RCSR<UWP>)
- State of the double-error bit (PCSTS<Trap2>)

A machine check results from an internally detected consistency error, such as the microcode reaches an "impossible" state, or from an external detected hardware error, such as a memory parity error.

A machine check is technically an aborted macro instruction. The CPU-chip's microcode attempts to convert the condition to a fault by unwinding the current instruction, with no guarantee that the instruction can be properly restarted. As much information as possible is pushed on the machine check stack frame, and the rest of the error parsing is left to the operating system.

When the software machine check handler routine receives control, it must explicitly acknowledge receipt of the machine check early in the routine to clear the internal machine-check-in-progress flag with the following instruction:

```
MTPR    #0, #PR$_MCESR          ; PR$_MCESR=38
```

The machine check stack frame is shown in Figure 3–8, and its parameters are described in Table 3–6. These parameters are parsed by the error handling macrocode to determine what caused the machine check.

Refer to Section 3.3.7 for an important caution on operating system exception handlers.

Figure 3–33 contains the machine check parse tree, which indicates the causes of each machine check. For those machine checks that have multiple causes, the registers and bits that isolate the cause are listed. The sections following the parse tree provide a description of the machine check, the procedure to recover, and the conditions for restarting the operation.

Figure 3–33 Machine Check Parse Tree

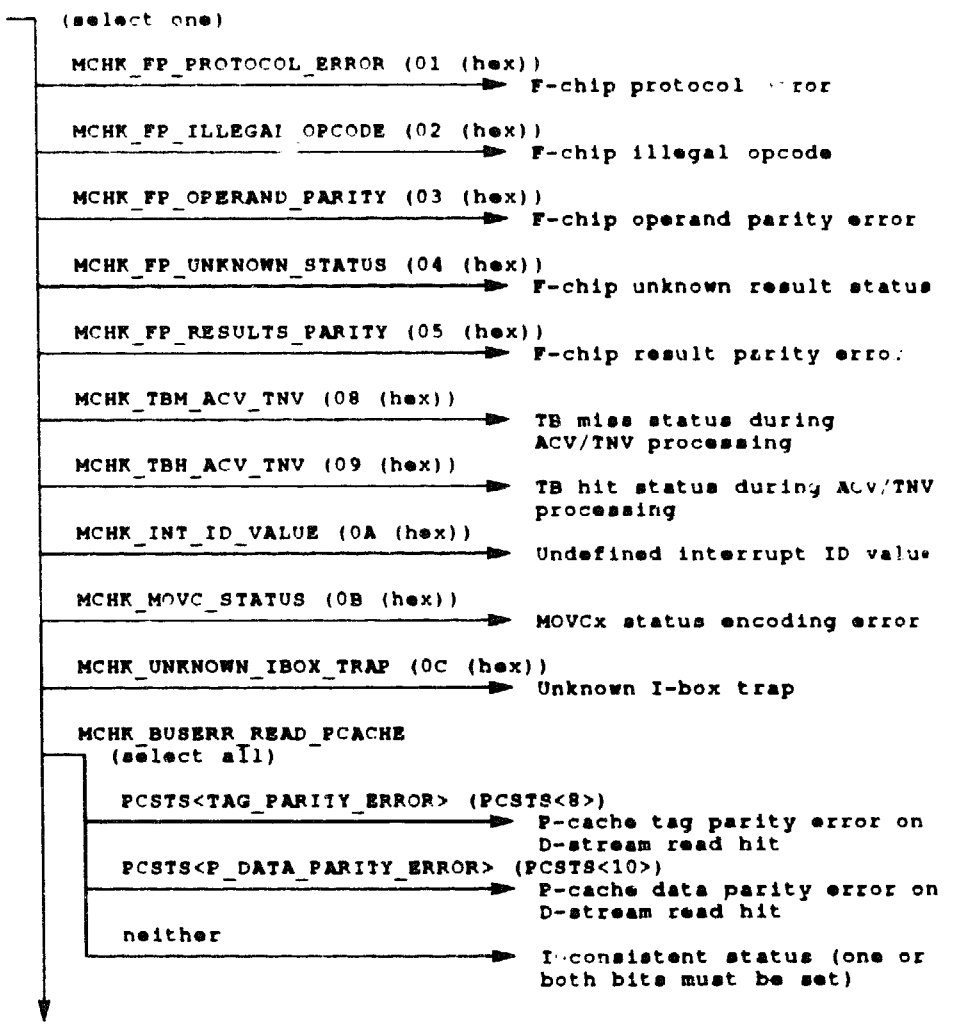


Figure 3–33 Cont'd. on next page

Figure 3–33 (Cont.) Machine Check Parse Tree

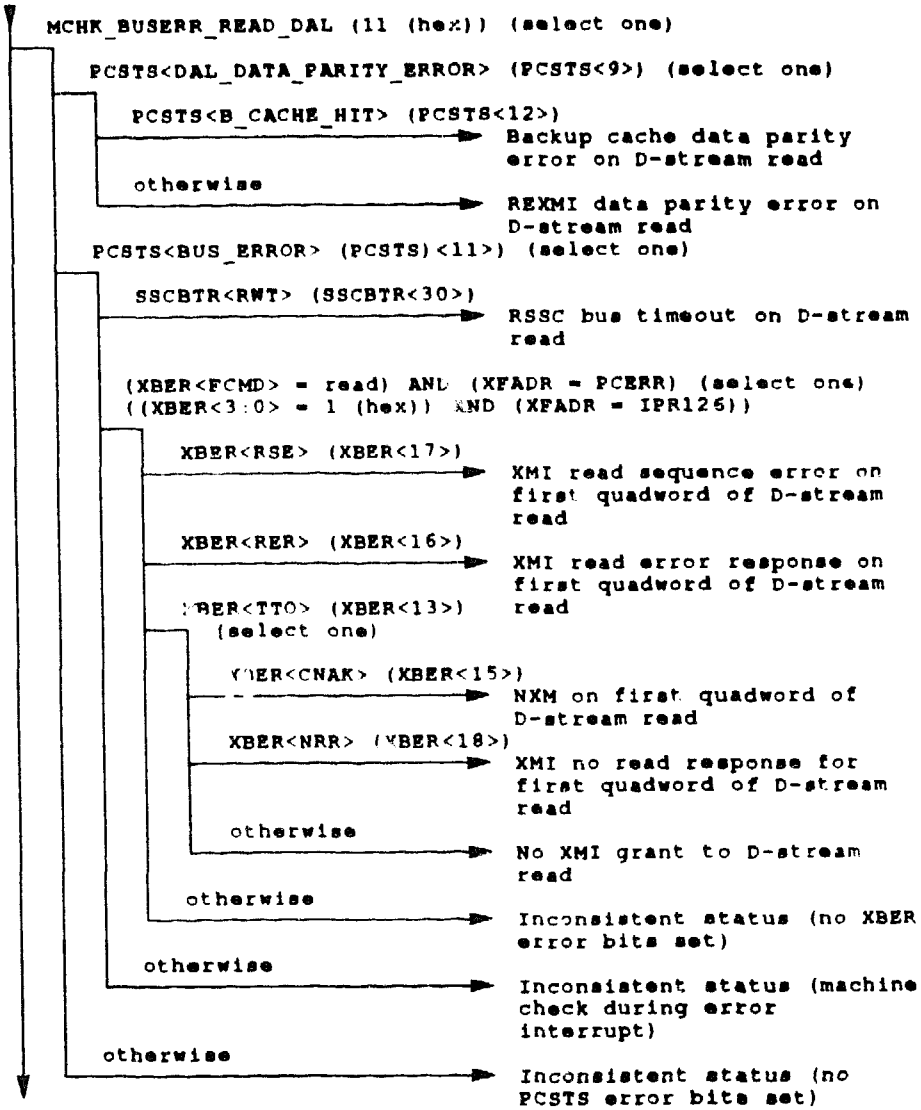
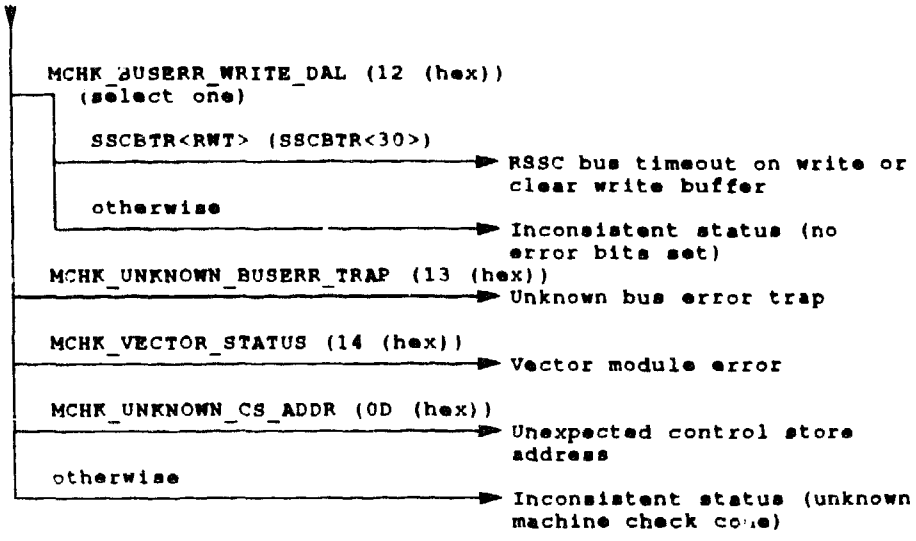


Figure 3–33 Cont'd. on next page

Figure 3–33 (Cont.) Machine Check Parse Tree



NOTES:

(select one) - exactly one case must be true. If zero or more than one is true, the status is inconsistent.

(select all) - more than one case may be true.

otherwise - fall-through case for (select one) if no other options are true.

neither - fall-through case for (select all) if none of the options are true.

The parse tree assumes that retry is enabled (RCSR<ARD> = 0).

3.11.4.1 MCHK_FP_PROTOCOL_ERROR

Description: A protocol error was detected by either the CPU-chip or the F-chip during an operand/result transfer. During a result return, this machine check is caused by one of these cases:

CPSTA <1:0>	CPDAT <2:0>	Detected By
00	xxx	CPU-chip
11	xxx	CPU-chip
10	000	F-chip

The error is probably due to a bit flipped on either the CPSTA or CPDAT signal lines between the CPU-chip and the F-chip during an opcode, operand, or result transfer.

Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the F-chip by writing a zero to ACCS<F-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an F-chip instruction, The restart bit (R) in the stack frame should always be one, PSL<FPD> should always be zero, and RCSR<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (RCSR<UWP>=0)

3.11.4.2 MCHK_FP_ILLEGAL_OPCODE

Description: An illegal opcode was detected by the F-chip and reported during result return. This is probably due to a bit flipped on the CPSTA or CPDAT lines during opcode transfer to the F-chip.

Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the F-chip by writing a zero to ACCS<F-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an F-chip instruction, R should always be one, PSL<FPD> should always be zero, and RCSR<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (RCSR<UWP>=0)

3.11.4.3 MCHK_FP_OPERAND_PARITY

Description: A parity error was detected by the F-chip during an operand transfer and reported during a result return. Backup cache or memory parity errors are also detected by the C-chip, resulting instead in a MCHK_BUSERR_READ_DAL machine check. This machine check indicates that the parity error was detected only by the F-chip, implying that the CPU-chip generated bad parity, or that the CPU-chip and the F-chip saw different operands or parity from the backup cache or memory. The error is a function of the data source, which cannot be determined from the machine check. The possible data sources are:

- GPR
- I-stream
- P-cache
- B-cache
- Memory

The possible causes are a CPU-chip parity checking error, an F-chip parity checking error, and an error on the D-bus (the data lines on the DAL bus) during operand transfer.

NOTE: It is possible to get this machine check if an F-chip operand is read from an I/O space location. CPU-chip parity checking is disabled for I/O space reads but the F-chip checks parity for all operands. Also see Section 3.3.7.

Recovery procedures. No explicit error recovery is required. If the error reoccurs, disable the F-chip by writing a zero to ACCS<F-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an F-chip instruction, R should always be one, PSL<FPD> should always be zero, and RCSR<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (RCSR<UWP>=0)

3.11.4.4 MCHK_FP_UNKNOWN_STATUS

Description: An unassigned status code was returned by the F-chip. This is caused when CPSTA=10 and CPDAT<2:0>=111 appear with the returned result (from F-chip to CPU-chip). This is probably due to a bit flipped on the CPSTA or CPDAT lines during result transfer to the CPU-chip.

Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the F-chip by writing a zero to ACCS<1>.

Restart condition: Because this error is detected during the execution flow of an F-chip instruction, R should always be one, PSL<FPD> should always be zero, and RCSR<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (RCSR<UWP>=0)

3.11.4.5 MCHK_FP_RESULT_PARITY

Description: A result data parity error was detected by the CPU-chip during F-chip result transfer. This is probably due to a bit flipped on the D-bus or parity lines.

Recovery procedures: No explicit error recovery is required. If the error reoccurs, disable the F-chip by writing a zero to ACCS<F-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an F-chip instruction, R should always be one, PSL<FPD> should always be zero, and RCSR<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (RCSR<UWP>=0)

3.11.4.6 MCHK_TBM_ACV_TNV

Description: During ACV/TNV microcode processing, the MMGT.STATUS field specified a TB-miss status, which is not possible during ACV/TNV processing. Probably due to an internal error in the memory management hardware or in the microbranch logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: This error happens during the microcode processing of an ACV/TNV exception on any virtual memory reference. Retry if:

(R=1) AND (PSL<FPD>=1) AND (RCSR<UWP>=0)

3.11.4.7 MCHK_TBH_ACV_TNV

Description: During ACV/TNV microcode processing, the MMGT.STATUS bits specified a TB-hit status, which is not possible during ACV/TNV processing. Probably due to an internal error in the memory management hardware or in the microbranch logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: This error happens during the microcode processing of an ACV/TNV exception on any virtual memory reference. Retry if:

(R=1) AND (PSL<FPD>=1) AND (RCSR<UWP>=0)

3.11.4.8 MCHK_INT_ID_VALUE

Description: During interrupt processing, the microbranch on the contents of the INT.ID register resulted in an unexpected interrupt ID. Probably due to a failure in the interrupt encoding logic or in the microbranch logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: This error can happen during the microcode processing of an ACV/TNV exception on any virtual memory reference. Retry if:

(R=1) AND (PSL<FPD>=1) AND (RCSR<UWP>=0)

3.11.4.9 MCHK_MOVC_STATUS

Description: During the execution of MOVCx, the two state bits that encode the state of the move (forward, backward, fill) were found set to the fourth, illegal, combination. Probably due to a failure in the state bit logic or in the microbranch logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: Because the state bits encode the operation, the instruction cannot be restarted in the middle of the MOVCx. If software can determine that no specifiers have been overwritten (MOVCx destroys R0–R5 and memory due to string writes), the instruction may be restarted from the beginning by clearing PSL<FPD>. This should be done only if the source and destination strings do not overlap and if:

(PSL<FPD>=1) AND (RCSR<UWP>=0)

3.11.4.10 MCHK_UNKNOWN_IBOX_TRAP

Description: The I-box requested a microtrap to report an illegal instruction or a reserved operand fault, but the bits that encode the reason specified an illegal value. Probably due to a failure in the I-box/E-box interface, or in the microsequencer trap logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: Because this microtrap can only occur at an instruction boundary, R should always be one, PSL<FPD> should always be zero, and RCSR<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (RCSR<UWP>=0)

3.11.4.11 MCHK_BUSERR_READ_PCACHE

One of two errors was detected during a D-stream read that hit in the P-cache. The P-cache must be enabled to get either the P-cache tag parity error on D-stream read hit or the P-cache data parity error on D-stream read hit.

PCSTS<Tag Parity Error> and PCSTS<P Data Parity Error> distinguish the cases. If neither bit is set, the status is inconsistent, and the read should not be retried. In both cases, PCERR contains the physical address of the error.

3.11.4.11.1 P-Cache Tag Parity Error on D-Stream Read Hit

Description: A P-cache tag parity error was detected on a D-stream read hit. PCSTS<Trap1>, PCSTS<Interrupt>, and PCSTS<Tag Parity Error> should all be set. This error is also reported as a soft error interrupt.

Recovery procedures: Write all P-cache tags with good parity and cleared valid bits, and perform the full memory error recovery procedures in Section 3.11.2.3. If the error reoccurs, disable both the P-cache and the primary cache tag store copy in the C-chip.

Restart condition: Retry if

((R=1) OR (PSL<FPD>=0)) AND (RCSR<UWP>=0) AND (PCSTS<TRAP2>=0)

3.11.4.11.2 P-Cache Data Parity Error on D-Stream Read Hit

Description: A P-cache data parity error was detected on a D-stream read hit. PCSTS<Trap1> and PCSTS<P DATA PARITY ERROR> should both be set.

Recovery procedures: Perform the full memory error recovery procedures in Section 3.11.2.3. If the error reoccurs, disable both the P-cache and the primary cache tag store copy in the C-chip.

Restart condition: Retry if

((R=1) OR (PSL<FPD>=0)) AND (RCSR<UWP>=0) AND (PCSTS<TRAP2>=0)

3.11.4.12 MCHK_BUSERR_READ_DAL

One of two classes of errors was detected during a D-stream read. PCSTS<DAL DATA PARITY ERROR> and PCSTS<BUS ERROR> distinguish the two classes. If neither or both bits are set, the status is inconsistent and the read should not be retried.

3.11.4.12.1 Data Parity Error on D-Stream Read

A data parity error was detected during a D-stream read. The source of the data parity error is either the backup cache or memory and is distinguished by PCSTS<B CACHE HIT>. In either case, PCERR contains the physical address of the error.

3.11.4.12.1.1 Backup Cache Data Parity Error on D-Stream Read

Description: A data parity error was detected during a D-stream read hit in the backup cache. PCSTS<Trap1>, PCSTS<DAL DATA PARITY ERROR>, and PCSTS<B CACHE HIT> should all be set.

Recovery procedures: Perform the full memory error recovery procedures in Section 3.11.2.3. If the error reoccurs, disable the backup cache.

Restart condition: Retry if

((R=1) OR (PSL<FPD>=1)) AND (RCSR<UWP>=0) AND (PCSTS<TRAP2>=0)

3.11.4.12.1.2 Memory Data Parity Error on D-Stream Read

Description: A data parity error was detected during a D-stream read from memory. An actual memory parity error would be reported as a bus error, described next. This error implies that the parity went bad between the REXMI and the CPU-chip. PCSTS<TRAP1> and PCSTS<DAL DATA PARITY ERROR> should both be set. PCSTS<B CACHE HIT> should be cleared.

Recovery procedures: Perform the full memory error recovery procedures in Section 3.11.2.3.

Restart condition: Retry if

((R=1) OR (PSL<FPD>=1)) AND (RCSR<UWP>=0) AND (PCSTS<TRAP2>=0)

3.11.4.12.2 Bus Error on D-Stream Read

A DAL D-stream read transaction was terminated with the ERR L signal asserted. The error source is determined by the state of SSCBTR<RWT>. The backup cache must be enabled and the reference must be to a non-I/O space address for BCSTS to log the error.

3.11.4.12.2.1 RSSC Bus Timeout on D-Stream Read

Description: The RSSC timed out a D-stream read. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, PCSTS<Trap1>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set. PCERR contains the physical address of the error.

Recovery procedures: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures in Section 3.11.2.3.

Restart condition: This error should never occur unless the reference is to an unimplemented node private space address that the REXMI does not respond to (the RSSC timeout is longer than the REXMI timeout). Retry may or may not be desirable. Retry if:

((R=1) OR (PSL<FPD>=1)) AND (RCSR<UWP>=0) AND (PCSTS<TRAP2>=0)

3.11.4.12.2.2 Memory Error on Requested Quadword of D-Stream Read

Description: The REXMI detected an error on the first quadword of a D-stream read. XBER<FCMD> must indicate a read instruction, and XFADR must match PCERR. Otherwise, the status is inconsistent and the read should not be retried. XFADR and PCERR contain the physical address of the error.

The source of the error is determined by the state of these bits in the XBER:

XBER<RSE> The first quadword of read data was returned with the wrong sequence number. Probably due to a parity error on the returned data. If so, XBER<PE> will also be set. PCSTS<BUS ERROR>, PCSTS<Trap1>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

XBER<RER> The first quadword of read data was returned with an RER response, indicating that the memory got a double-bit error reading the array. Probably not recoverable, but it is possible. PCSTS<BUS ERROR>, BCSTS<Trap1>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

XBER<TTO> The first quadword of read data was not returned before the REXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set in the normal case. If neither bit is set, the REXMI was never granted the XMI for the read command. PCSTS<BUS ERROR>, PCSTS<Trap1>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

XBER<CNAK> The read data command was NO ACKed by the XMI and retry failed (if retry was enabled). Probably indicates a read to nonexistent memory (NXM). XBER<TTO> also should be set unless RCSR<ARD> is set. PCSTS<BUS ERROR>, PCSTS<Trap1>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set. If this is a real NXM, retry will not succeed and should not be attempted if NXMs are expected. If NXMs are not expected and retry is desired, retry under the conditions stated above.

XBER<NRR> The first quadword of read data was not returned before the REXMI timout expired. The read command was ACKed on the XMI, so this is not an NXM. NRR probably set because of a parity error on the returned data (XBER<PE> will also be set). XBER<TTO>, PCSTS<BUS ERROR>, PCSTS<Trap1>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

Recovery procedure: Perform the full memory error recovery procedures described in Section 3.11.2.3.

Restart condition: Retry if:

((R=1) OR (PSL<FPD>=1)) AND (RCSR<UWP>=0) AND (PCSTS<TRAP2>=0)

unless stated otherwise above.

3.11.4.13 MCHK_BUSERR_WRITE_DAL

Description: A DAL Write or clear write buffer transaction was terminated with the ERR L signal asserted. The backup cache must be on and the reference must be to a non-I/O space address for BCSTS to log the error. Since this is never done by the REXMI, it must have been a DAL timeout by the RSSC.

SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, PCSTS<Trap1>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set. If they are not, the error is probably due to a failure in the bus interface unit (BIU) of the CPU-Chip or microsequencer trap logic.

Recovery procedures: Clear SCCBTR<RWT> and SSCBTR<BTO>. Then perform the full memory error recovery procedures described in Section 3.11.2.3.

Restart condition: Retry should not be attempted because the error is not reported until after the instruction that issued the write is completed and the address is not available. The error is assumed to be a systemwide error.

3.11.4.14 MCHK_UNKOWN_BUSERR_TRAP

Description: The BIU requested a microtrap to report a cache or bus error, but the bits that encode the reason specified an illegal value. Probably due to a failure in the BIU or microsequencer trap logic.

Recovery procedures: No explicit error recovery is required.

Restart condition: Retry should not be attempted because this error may be masking a write error.

3.11.4.15 MCHK_VECTOR_STATUS

Description: A vector processor related error was detected and reported to the scalar processor. There are two classes of errors: VIB related errors reported by the C-chip and FV64A vector processor related errors. See also Section 4.9.1.

Recovery procedures: In general, a MCHK_VECTOR_STATUS machine check is not recoverable.

Restart conditions: Retry should not be attempted. Reset and initialize the FV64A vector processor module.

3.11.4.16 MCHK_UNKNOWN_CS_ADDR

Description: An unexpected address was reached in the control store Probably due to a failure in the microsequencer logic or a microcode bug.

Recovery procedures: No explicit error recovery is required.

Restart conditions: Retry if:

((R=1) OR (PSL<FPD>=1)) AND (RCSR<UWP>=0)

3.11.5 Power Fail Interrupt

Power fail interrupts are requested by the XTC power sequencer to report an imminent loss of power to the CPU-module.

Power fail interrupts are requested at IPL 1E (hex) and are dispatched through SCB vector 0C (hex). The stack frame for a power fail interrupt is shown in Figure 3–2.

The VAX 6000–400 supports the standard XMI time of 4 milliseconds to execute the software necessary to save processor state.

3.11.6 Hard Error Interrupt

A hard error interrupt reports an error that was detected asynchronously with instruction execution.

A hard error interrupt results in an interrupt at IPL 1D (hex) being dispatched through SCB vector 60 (hex). Typically, these errors indicate that machine state has been corrupted and that a retry is not possible. The stack frame for a hard error interrupt is shown in Figure 3–2.

Figure 3–34 contains the hard error interrupt parse tree, which indicates the causes of each hard error interrupt. For those hard error interrupts that have multiple causes, the registers and bits that isolate the cause are listed. The sections following the parse tree provide a description of the hard error error, the procedure to recover, and the conditions for restarting the operation.

CAUTION: There is an anomaly in the KA64A CPU module that causes FCSR<4> (Second Error) to set, under some circumstances, resulting in a spurious hard error interrupt. Hard error interrupt handlers must deal with interrupts that have no error bits set. See Figure 3–34 for valid hard error interrupts.

Figure 3–34 Hard Error Interrupt Parse Tree

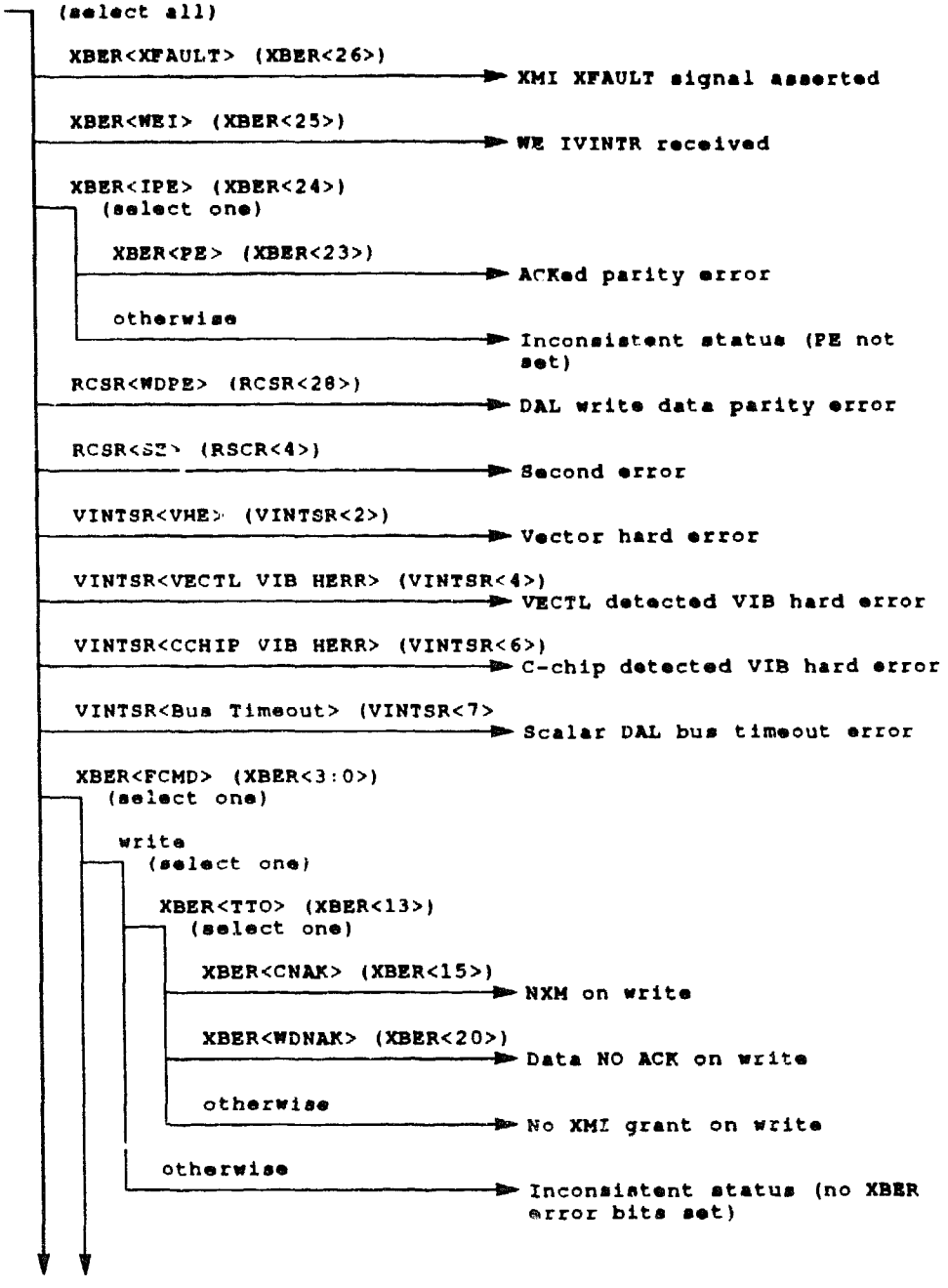
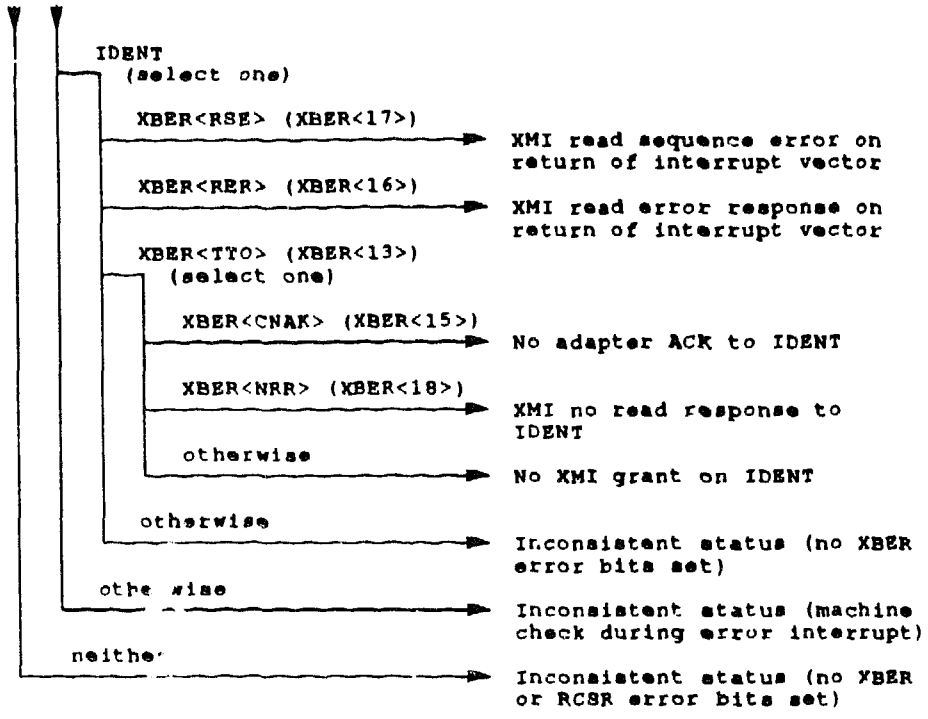


Figure 3–34 Cont'd. on next page

Figure 3–34 (Cont.) Hard Error Interrupt Parse Tree



NOTES:

(select one) - only one case must be true. If none or more than one is true, the status is inconsistent.

(select all) - more than one case may be true.

neither - fall-through case for (select all) if none of the options are true.

otherwise - fall-through case for (select one) if no other options are true.

The parse tree assumes that retry is enabled (RCSR<ARD> = 0).

3.11.6.1 XMI XFAULT Assertion

Description: The XMI FAULT L signal notifies all nodes on the XMI backplane of a severe XMI hardware failure. If it asserts, the REXMI sets XBER<XFAULT> and requests a hard error interrupt.

Recovery procedures: Clear all error bits in the XBER.

Restart conditions: Retry is not possible.

3.11.6.2 XMI Write Error IVINTR

Description: A write error IVINTR XMI command was received by the REXMI. XBER<WEI> indicates this condition. WEIs are generated by other nodes, so the interrupt source (for example, the DWMBA) must be examined to determine if recovery is possible.

Recovery procedures: Clear all error bits in the XBER.

Restart conditions: Retry is not possible.

3.11.6.3 XMI Inconsistent Parity Error

Description: During every XMI cycle, the REXMI checks the XMI parity bits against the computed parity on the received information. If a parity error is detected, XBER<PE> sets and a soft error interrupt is requested. If the cycle is subsequently ACKed, XBER<IPE> sets and the error is reported as a hard error interrupt. Since parity is checked every cycle, XBER<IPE> does not imply that this node transmitted the failing data.

Recovery procedures: Clear all error bits in the XBER.

Restart conditions: Retry is not possible since this is a systemwide error.

3.11.6.4 DAL Data Parity Error on Memory Write

Description: The REXMI detected a parity error on the DAL data for a memory write from the CPU-chip. RCSR<WDPE> should be set. Probably due to a bit flipped on the D-bus or to intentional/unintentional bad parity generated by the CPU-chip. The XMI transaction is suppressed.

Recovery procedures: Clear all error bits in RCSR.

Restart conditions: Retry is not possible since no address is available. The error is assumed to be systemwide.

3.11.6.5 Second Error Detected on XMI

Description: The REXMI detected an error when the XMI Failing Address Register (XFADR) was locked because of a previous error.

Recovery procedures: Clear all bits in XBER and RCSR.

Restart conditions: Retry is not possible since the second error state is unknown.

3.11.6.6 Vector Hard Error

Description: A hard error was detected by the vector processor. See also Section 4.9.2.

Recovery procedures: No recovery.

Restart conditions: Retry should not be attempted. Reset and initialize the FV64A vector processor module.

3.11.6.7 VECTL Detected VIB Hard Error

Description: The FV64A vector processor detected a VIB error. See also Section 4.9.2.

Recovery procedures: No recovery.

Restart conditions: Retry should not be attempted. Reset and initialize the FV64A vector processor module.

3.11.6.8 C-Chip Detected Hard Error

Description: A vector processor related error was detected and reported to the scalar processor. See also Section 4.9.2.

Recovery procedures: No recovery.

Restart conditions: Retry should not be attempted. Reset and initialize the FV64A vector processor module.

3.11.6.9 XMI Errors on Writes

Description: The REXMI detected an XMI error while processing a write command. XBER<FCMD> contains the encoding for a write, which is what distinguished these errors from those caused by other commands. XFADR contains the physical address of the error.

The source of the error is determined by the state of these bits in the XBER:

XBER<TTO>	A write was not completed on the XMI before the REXMI timeout expired. XBER<CNAK> or XBER<WDNAK> should also be set in the normal case. If neither bit is set, the REXMI was never granted the XMI for the write command.
XBER<CNAK>	The write command was NO ACKed on the XMI and retry failed, if it was enabled. Probably an NXM. XBER<TTO> should also be set unless RCSR<ARD> is set.
XBER<WDNAK>	The write data cycle was NO ACKed by the XMI and all retries failed, if retry is enabled. This does not indicate an NXM because NXM would result instead in XBER<CNAK>. XBER<TTO> should also be set unless RCSR<ARD> is set.

Recovery procedures: Clear all error bits in the XBER.

Restart condition: No retry is possible because this error indicates a failed write. Software must determine if the error is limited to the process or if the entire system is affected.

3.11.6.10 XMI Errors on IDENTs

Description: The REXMI detected an XMI error while processing an IDENT command. XBER<FCMD> contains the encoding for an IDENT, which is what distinguishes these errors from those caused by other commands. XFADR contains the node that the IDENT was sent to and the encoded IPL.

The REXMI terminated the DAL command with the ERR_L signal asserted, which the CPU-chip's microcode treats as a microcode passive release. The error is then reported as a hard error.

The source of the error is determined by the state of these bits in the XBER:

XBER<RSE> The interrupt vector was returned with the wrong sequence number.

XBER<RER> The DWMBA returned an RER response.

NOTE: The DWMBA responds with an RER to an XMI IDENT transaction if it receives a passive release from a device in response to the IDENT on the VAXBI bus.

XBER<TTO> The interrupt vector was not returned before the REXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set in the normal case. If neither bit is set, the REXMI was never granted the XMI for the IDENT.

XBER<CNAK> The IDENT command was NO ACKed by the DWMBA and retry failed, if retry is enabled. This may indicate that the target DWMBA for the IDENT is not present. XBER<TTO> should also be set unless RCSR<ARD> is set.

XBER<NRR> The interrupt vector was not returned before the REXMI timeout expired. The read command was ACKed on the XMI, so the DWMBA received the IDENT. XBER<TTO> should also be set.

Recovery procedures: Clear all error bits in the XBER.

Restart conditions: Error retry may automatically occur as the result of device timeouts done by the device driver because an IDENT error implies a missed interrupt. If there is no automatic retry, no explicit retry is possible.

3.11.7 Soft Error Interrupt

Soft error interrupts are requested to report an error that was detected but which did not affect instruction execution.

Soft error interrupts result in an interrupt at IPL 1A (hex) being dispatched through SCB vector 54 (hex). Retry is always possible, after recovery, for soft errors unless stated otherwise for the specific error.

The stack frame for a soft error interrupt is shown in Figure 3–2. Figure 3–35 contains the soft error interrupt parse tree, which indicates the causes of each soft error interrupt. For those soft error interrupts that have multiple causes, the registers and bits that isolate the cause are listed. The sections following the parse tree provide a description of the soft error interrupt, the procedure to recover, and the conditions for restarting the operation.

Figure 3–35 Soft Error Interrupt Parse Tree

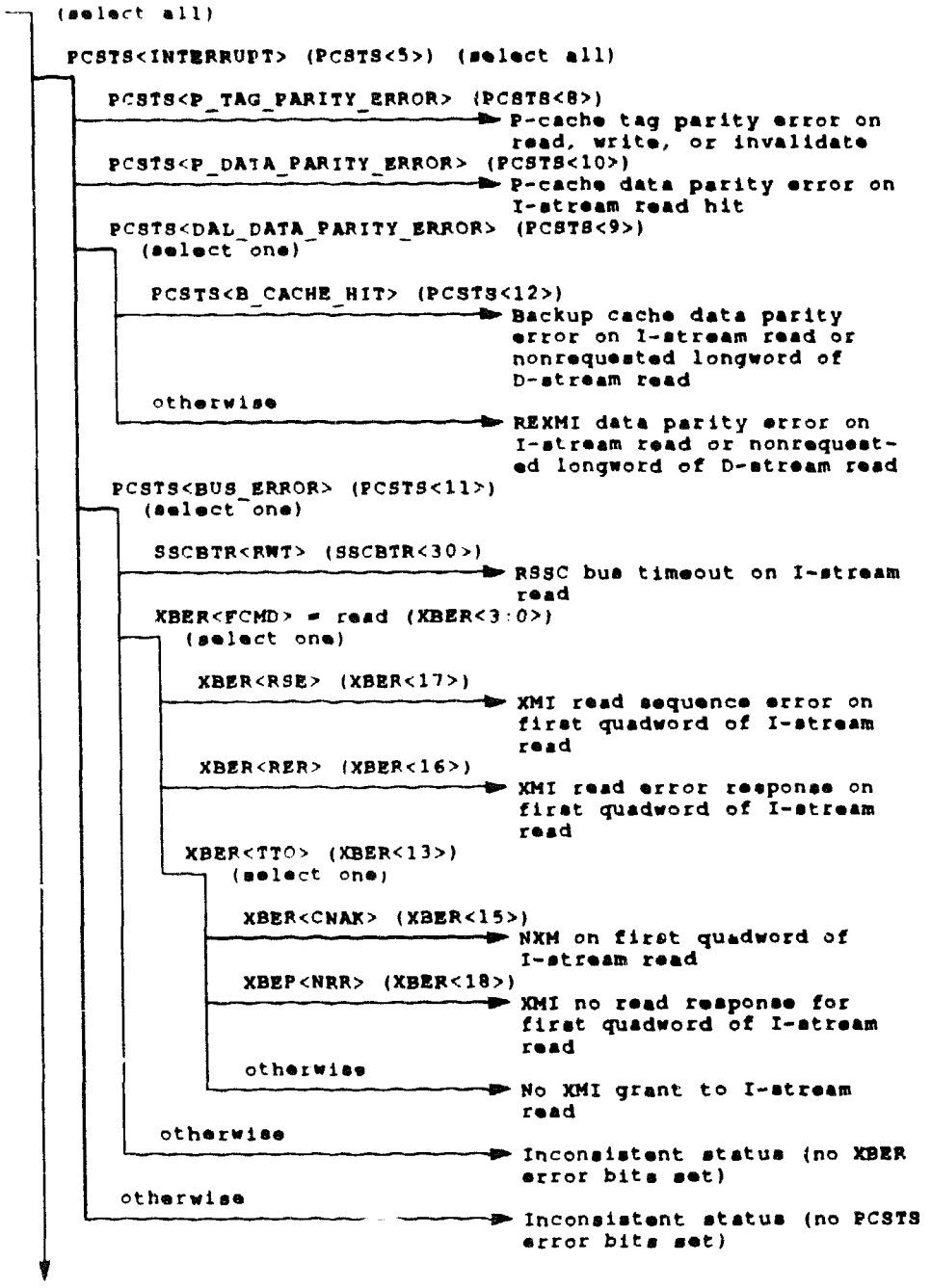
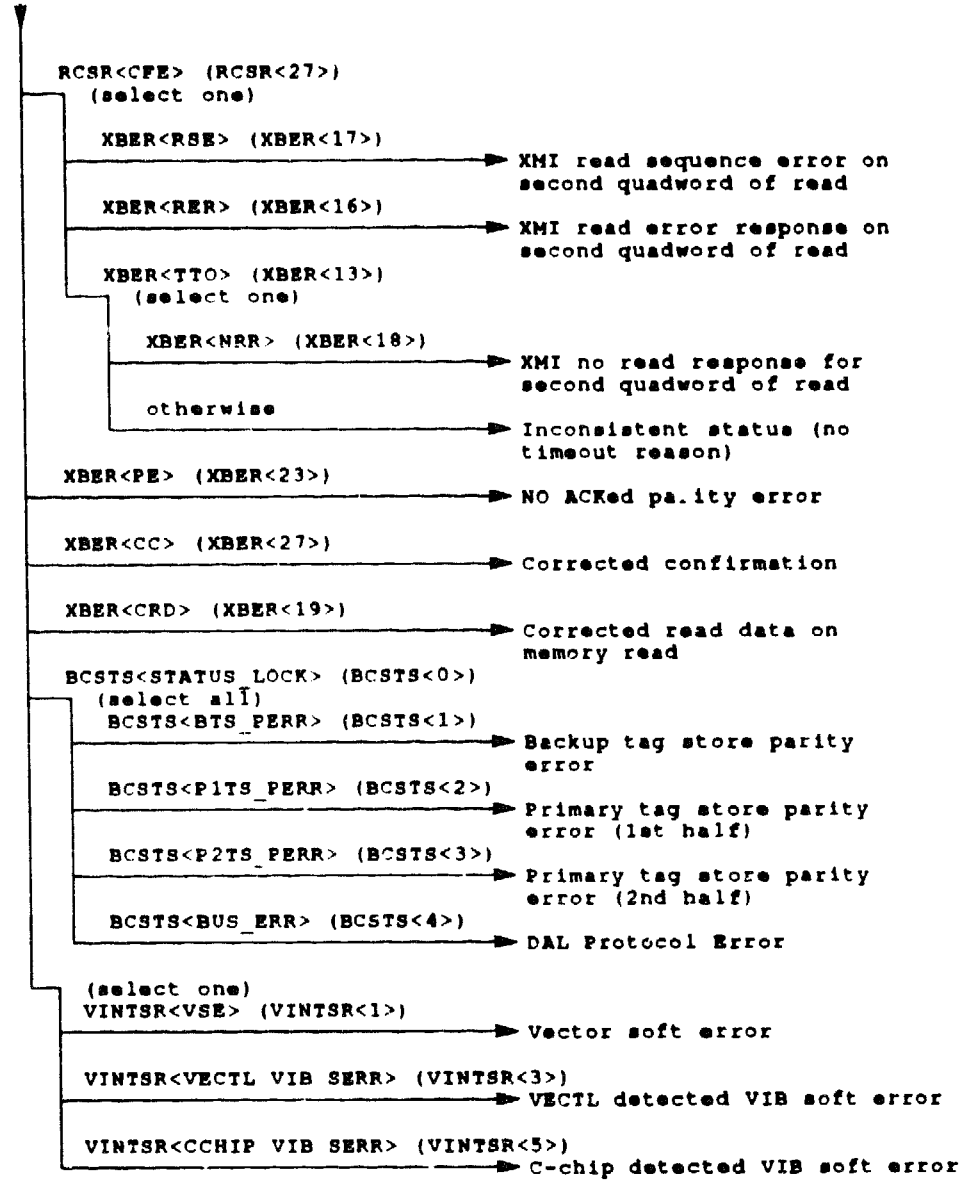


Figure 3–35 Cont'd on next page

Figure 3–35 (Cont.) Soft Error Interrupt Parse Tree



NOTES:

(select one) - exactly one case must be true. If none or more than one is true, the status is inconsistent.

(select all) - more than one case may be true.

otherwise - fall-through case for (select one) if no other options are true.

The parse tree assumes that retry is enabled (RCSR<ARD> = 0).

3.11.7.1 Cache or Memory Errors

A P-cache error, data parity error, or bus error was detected during a memory reference. PCSTS<Interrupt> distinguishes this class from others. At least one of these bits should be set: PCSTS<P TAG PARITY ERROR>, PCSTS<P DATA PARITY ERROR>, PCSTS<DAL DATA PARITY ERROR>, or PCSTS<BUS ERROR>. PCERR does not contain the error address in this class of error

All I-stream errors are automatically retried by the CPU-chip's microcode using D-stream reads. If the error is hard, the D-stream read is reported as a machine check with code MCHK_BUSERR_READ_DAL. If the error is transient, it is reported as a soft error interrupt, indicating that retry was successful

3.11.7.1.1 P-Cache Errors

One of two P-cache errors was detected during an I-stream read, write, or invalidate. The P-cache must be enabled to get either the P-cache tag parity error or the P-cache data parity error on an I-stream read. The state of PCSTS<P TAG PARITY ERROR> and PCSTS<P DATA PARITY ERROR> determine the error

3.11.7.1.1.1 P-Cache Tag Parity Error

Description: A P-cache tag parity error was detected during an I-stream read, a D-stream read miss, a write, or an invalidate. If the error was detected during a D-stream read hit, the error would be reported as a machine check with code MCHK_BUSERR_READ_PCACHE. PCSTS<P TAG PARITY ERROR> and PCSTS<Interrupt> should both be set.

Recovery procedures: Write all P-cache tags with good parity and clear valid bits. Then perform the full memory error recovery procedures described in Section 3.11.2.3. If the error reoccurs, disable both the P-cache and the primary cache tag store copy in the C-chip.

3.11.7.1.1.2 P-Cache Data Parity Error on I-Stream Read Hit

Description: A P-cache data parity error was detected during an I-stream read hit. If the error was detected during a D-stream read hit, the error would be reported as a machine check with code MCHK_BUSERR_READ_PCACHE. PCSTS<P DATA PARITY ERROR> and PCSTS<Interrupt> should both be set

Recovery procedures: Perform the full memory error recovery procedures described in Section 3.11.2.3. If the error reoccurs, disable both the P-cache and the primary cache tag store copy in the C-chip.

3.11.7.1.2 DAL Data Parity Errors

A DAL data parity error was detected during an I-stream read or was in the nonrequested longword of a D-stream read. If the error was detected in the requested longword of a D-stream read, the error would be reported as a machine check with code MCHK_BUSERR_READ_DAL. The source of the data parity error is either the backup cache or memory. It is indicated by PCSTS<B CACHE HIT>.

3.11.7.1.2.1 Backup Cache Data Parity Error

Description: A data parity error was detected during an I-stream read or in the nonrequested longword of a D-stream read that hit in the backup cache. PCSTS<Interrupt>, PCSTS<DAL DATA PARITY ERROR>, and PCSTS<B CACHE HIT> should all be set.

Recovery procedures: Perform the full memory error recovery procedures described in Section 3.11.2.3. If the error reoccurs, disable the backup cache.

3.11.7.1.2.2 Memory Data Parity Error

Description: A data parity error was detected during an I-stream read or in the nonrequested longword of a D-stream read from memory. An actual memory parity error would be detected by the REXMI and reported as a memory read error. A Memory Data Parity Error implies that the parity went bad between the REXMI and the CPU-chip. PCSTS<Interrupt> and PCSTS<DAL DATA PARITY ERROR> should both be set. PCSTS<B CACHE HIT> should be clear.

Recovery procedures: Perform the full memory error recovery procedures described in Section 3.11.2.3.

3.11.7.1.3 Bus Error on I-Stream Read

A DAL I-stream read transaction was terminated with ERR_L. If the error was detected during a D-stream read, the error would be reported as a machine check with code MCHK_BUSERR_READ_DAL. The source of the error is indicated by SSCBTR<RWT>. The backup cache must be enabled, and the reference must be to a non-I/O space address for BCSTS to log the error.

3.11.7.1.3.1 RSSC Bus Timeout on I-Stream Read

Description: The RSSC timed out an I-stream read. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, PCSTS<Interrupt>, BCSTS<BUS ERR> and BCSTS<Status Lock> should all be set.

Recovery procedures: Clear SSCBTR<RWT> and SSCBTR<BTO>. Then perform the full memory error recovery procedures described in Section 3.11.2.3.

Restart conditions: This error should never occur unless the reference is to an unimplemented node private space address to which the REXMI does not respond since the RSSC timeout should be longer than the REXMI timeout.

3.11.7.1.3.2 Memory Error on Requested Quadword of I-Stream Read

Description: The REXMI detected an error on the first quadword of an I-stream read. XBER<FMCD> should be a read. XFADR contains the physical address of the error.

The source of the error is determined by the state of these bits in the XBER:

XBER<RSE> The first quadword of read data was returned with the wrong sequence number. PCSTS<BUS ERROR>, PCSTS<Interrupt>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

XBER<REP> The first quadword of read data was returned with a read error response, indicating that the memory got a double-bit error while reading the array. PCSTS<BUS ERROR>, PCSTS<Interrupt>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

XBER<TTO> The first quadword of read data was not returned before the REXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set in the normal case. If neither bit is set, the REXMI was never granted the XMI for the read command. PCSTS<BUS ERROR>, PCSTS<Interrupt>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

XBER<CNAK> The read data command was NO ACKed by the XMI and retry failed, if retry is enabled. This probably indicates an NXM. XBER<TTO> should also be set unless RCSR<ARD> is set. PCSTS<BUS ERROR>, PCSTS<Interrupt>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

XBER<NRR> The first quadword of read data was not returned before the REXMI timeout expired. The read command was ACKed on the XMI, so this is not an NXM. Probably a parity error occurred on the returned data (if so, XBER<PE> will also be set). XBER<TTO>, PCSTS<BUS ERROR>, PCSTS<Interrupt>, BCSTS<BUS ERR>, and BCSTS<Status Lock> should all be set.

Recovery procedures: Perform the full memory error recovery procedures described in Section 3.11.2.3

3.11.7.2 Cache Fill Errors on the Nonrequested Quadword of a Read

Description: The REXMI detected an error in the nonrequested (fill) quadword of a D-stream or I-stream read. The setting of RCSR<CFE> indicates this error, as opposed to an error detected during the requested quadword of an I-stream read. XFADR contains the physical address of the error, and XBER<FCMD> contains the encoding for a read. The source of the error is determined by bits in the XBER:

XBER<RSE>	The second quadword of read data was returned with the wrong sequence number.
XBER<RER>	The second quadword of read data was returned with a read error response, indicating that the memory got a double-bit error while reading the array. Probably not recoverable.
XBER<NRR>	The second quadword of read data was not returned before the REXMI timeout expired. XBER<TTO> should also be set.

Recovery procedures: Perform the full memory error recovery procedures described in Section 3.11.2.3.

3.11.7.3 Other REXMI-Detected Errors

Description: The REXMI detected an error during an XMI transaction. The state of these bits in the XBER determine the various cases:

XBER<PE>	A parity error was detected during an XMI cycle. If the cycle was subsequently ACKed, XBER<IPE> is set and the error is reported as a hard error interrupt.
XBER<CC>	The XMI Corner corrected a single-bit error in a confirmation code. This interrupt is posted unless RCSR<CCID> is set.
XBER<CRD>	One or more quadwords of read data was returned with a CRD response, indicating the memory corrected a single-bit error. This interrupt is posted unless RCSR<CRDID> is set.

Recovery procedures: Clear all error bits in the XBER.

3.11.8 C-Chip Errors

The C-chip detected a tag parity error during a tag store access or a bus protocol error occurred during a DAL transaction. The C-chip must be enabled (Enable Refresh, Enable BTS, and Enable PTS all equal to one) to detect these errors.

3.11.8.1 C-Chip Backup Tag Store Parity Error

Description: A tag store parity error was detected in the backup tag store during a read, write, fill, invalidate, or Inval-Bus access. BCSTS<BTS PERR> and BCSTS<Status Lock> should both be set. BCERR contains the physical address of the error.

Recovery procedures: Use the BCERR address to rewrite the tag with good parity and a clear valid bit. Then perform the full memory error recovery procedures described in Section 3.11.2.3. If the error reoccurs, disable the backup cache.

3.11.8.2 C-Chip Primary Tag Store Parity Error

Description: A tag store parity error was detected in one of the primary tag stores during a fill, invalidate, or Inval-Bus access. Either BCSTS<P1TS PERR> or BCSTS<P2TS PERR>, or both, should be set, as well as BCSTS<Status Lock>. BCERR contains the physical address of the error.

Recovery procedures: Use the BCERR address to rewrite the tag with good parity and a clear valid bit. Then perform the full memory error recovery procedures described in Section 3.11.2.3. If the error reoccurs, disable both the P-cache and the C-chip copy of the primary cache tag store.

3.11.8.3 C-Chip Bus Protocol Error

Description: If the cache RAM speed in the C-chip (BCCTL<Two-Cycle RAMs>) and the REXMI (RCSR<RAM SPD>) are different, the C-chip may detect a protocol error during memory writes or cache fill operations. BCSTS<BUS ERR> and BCSTS<Status Lock> should both be set.

Recovery procedures: Ensure that the cache RAM speed in the C-chip and the REXMI agree. Then perform the full memory error recovery procedures described in Section 3.11.2.3.

3.11.8.4 C-Chip Vector Module Errors

Description: An error was detected on the vector interface bus (VIB). For FV64A vector processor errors, see Section 4.9.3.

Recovery procedures: Automatic recovery of soft errors.

3.11.9 Kernel Stack Not Valid Exception

A kernel stack not valid exception occurs when a memory management exception is detected while attempting to push information on the kernel stack during microcode processing of another exception. A console halt with an error code of EPR_INTSTK is taken if a memory management exception is encountered while attempting to push information on the interrupt stack.

The kernel stack not valid exception is dispatched through SCB vector 08 (hex) and the stack frame is shown in Figure 3–2.

3.11.10 Errors with No Notification

One category of REXMI-detected errors cause no explicit notification. The error status bits are set and, the next time software inspects the register for other reasons, it may want to check/log these bits.

3.11.10.1 CSR Read Data NO ACK

Description: When another XMI node reads a REXMI CSR, the REXMI responds with the requested data. If the data transfer is not ACKed by the receiving node, the REXMI sets XBER<RIDNAK>. This will usually be reported on the receiving node as a no read response error.

Recovery procedures: Clear all error bits in the XBER.

3.11.10.2 CSR Write Sequence Error

Description: When another XMI node writes a REXMI CSR, the REXMI compares the transmitted sequence number with the one that it expects and NO ACKs the cycle if the two do not match. The CSR write is ignored and XBER<WSE> is set.

Recovery procedures: Clear all error bits in the XBER.

3.11.11 Error Recovery Coding Examples

Example 3–1 is pseudo-code that demonstrates the process of analyzing and recovering from errors reported as machine checks, hard error interrupts, and soft error interrupts. This is not intended to be a complete description but is a means of documenting the important aspects of the process. The code is organized into a number of support procedures followed by the main error handlers.

Example 3–1 Error Recovery Coding

```
PROCEDURE collect_error_state;

! This support procedure collects error state from the
! appropriate XRP registers.

    BEGIN

    s_rcsr := RCSR;                     ! save REXMI RCSR register
    s_xber := XBER;                     ! save REXMI XBER register
    s_xfadr := XFADR;                   ! save REXMI XFADR register
    s_sscbtr := SSCBTR;                 ! save RSSC SSCBTR register
    s_bcctl := BCCTL;                   ! save C-chip BCCTL register
    s_bcsts := BCSTS;                   ! save C-chip BCSTS register
    s_bcerr := BCERR;                   ! save C-chip BCERR register
    BCIDX := s_bcerr;                   ! load index register
    s_bcbts := BCBTS;                   ! save B-cache tags
    s_bcp1ts := BCP1TS;                 ! save P-cache tags
    s_bcp2ts := BCP2TS;
    s_pcsts := PCSTS;                   ! save P-cache PCSTS register
    s_pcerr := PCERR;                   ! save P-cache PCERR register
    PCIDX := s_pcerr;
    s_pctag := pctag;
    s_vintsr := VINTSR;                 ! save C-chip VINTSR register

    END;  ! collect_error_state

PROCEDURE disable_cache;

! This support procedure disables both the P-cache and the backup
! cache during error recovery.  The P-cache must be disabled
! first to ensure cache coherency.

    BEGIN

! Enable P-cache refresh, disable the cache, and make sure
! force hit is off, while leaving the WC bits alone.

    PCSTS := ENABLE_REFRESH+^ENABLE_PTS+^FORCE_HIT;

! Enable backup cache refresh, disable both tag stores,
! and make sure force hit is off.  The correct state for
! TWO_CYCLE_RAMS should also be loaded.

    BCCTL := ENABLE_REFRESH+^ENABLE_PTS+^ENABLE_BTS+
             ^FORCE_BHIT+(two-cycle rams state);

! Initialize flags used to determine if cache should be reenabled.
! If either cache is already disabled, they are not reenabled
! on restart.
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Error Recovery Coding

```
    pcache_disable := s_pcsts<ENABLE_PTS> AND s_bcctl<ENABLE_PTS>;
    bcache_disable := s_bcctl<ENABLE_BTS>;

    END;  ! disable_cache

PROCEDURE enable_cache;

! This support procedure conditionally reenables both caches if
! they were enabled before, and no error thresholds have been
! exceeded.  The primary tag store copy in the C-chip must
! be enabled before the P-cache to ensure cache coherency.

    BEGIN

! Assume that both caches will be enabled.

    bc_enable := ENABLE_REFRESH+ENABLE_PTS+ENABLE_BTS+
                 ^FORCE_BHIT+(two-cycle rams state);

    pc_enable := ENABLE_REFRESH+ENABLE_PTS+^FORCE_HIT+
                 FLUSH_CACHE;

! If backup cache shouldn't be enabled, clear enable bit

    IF bcache_disable THEN
        bc_enable := bc_enable AND NOT ENABLE_BTS;

! If either primary tag store copy or P-cache shouldn't
! be enabled, clear enable bit

    IF pcache_disable THEN
        BEGIN
        bc_enable := bc_enable AND NOT ENABLE_PTS;
        pc_enable := pc_enable AND NOT ENABLE_PTS;
        END;

! Flush all tag stores and reenable the caches as appropriate.

    BCFPTS := 0;                        !It is important to flush the
    BCFBTS := 0;                        !caches in this order with no
    BCCTL := bc_enable;                 !intervening instructions.
    PCSTS := pc_enable;                 !IPL must be set to 31 to
    BCFPTS := 0;                        !prevent interrupts from oc-
    BCFBTS := 0;                        !curring between instructions.

    END;  ! enable_cache

PROCEDURE recover_from_memory_subsystem_errors;

! This support procedure is called to perform error recovery
! for those errors that are related to the cache or memory
! subsystem.  Most error recovery consists of writing the
! original register contents back to the XRP registers.  This
! restores the "M" bits to their original state and clears the
! "WC" error bits.

    BEGIN

! Clear REXMI errors.  RCSR must be cleared before XBER.

    IF s_xber<NSES> THEN
        RCSR := s_rcsr;                 ! clear RCSR errors
    XBER := s_xber AND
        NOT s_xber<XBAD>;               ! clear XBER error
    RCSR := s_rcsr;

! Clear RSSC bus timeout errors.

    SSCBTR := s_sscbtr;                 ! clear RSSC timeout errors
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Error Recovery Coding

```
! Check for C-chip backup tag store parity error.  If error
! threshold is exceeded, disable the cache on exit

    IF s_bcsts<BTS_PERR> THEN           ! if backup tag store parity error,
        BEGIN                           ! use error address to rewrite tag
        BCIDX := s_bcerr;               ! with good parity and invalid tag
        BCBTS := %X20000000;
        IF (backup tag store error threshold exceeded) THEN
            bcache_disable := TRUE;
        END;

! Check for C-chip primary tag store parity errors in either half.  If error
! threshold is exceeded, disable the cache on exit.

    IF s_bcsts<P1TS_PERR> THEN          ! if primary tag store parity error,
        BEGIN                           ! use error address to rewrite tag
        BCIDX := s_bcerr;               ! with good parity and invalid tag
        BCP1TS := %X20000000;
        IF (primary tag store error threshold exceeded) THEN
            pcache_disable := TRUE;
        END;

! recover_from_memory_subsystem_errors, continued.

    IF s_bcsts<P2TS_PERR> THEN          ! if primary tag store parity error,
        BEGIN                           ! use error address to rewrite tag
        BCIDX := s_bcerr;               ! with good parity and invalid tag
        BCP2TS := %X20000000;
        IF (primary tag store error threshold exceeded) THEN
            pcache_disable := TRUE;
        END;

! Clear C-chip errors.

    BCSTS := s_bcsts;                   ! clear c-chip errors

! Check for P-cache tag parity errors, which require rewriting the
! entire tag store.  If error threshold is exceeded, disable the cache
! on exit

    IF s_pcsts<TAG_PARITY_ERROR> THEN   ! if P-cache tag parity error
        BEGIN
        FOR i := 0 to 255 DO            ! rewrite all tags with
            BEGIN                       ! good parity and invalid tag
            PCIDX := i * 8;
            PCTAG := %X40000000;
            END;
        IF (pcache error threshold exceeded) THEN
            pcache_disable := TRUE;
        END;


! Clear P-cache errors.  Note that this leaves the cache enable
! state unchanged.

    pc_enable := s_pcsts AND (TRAP1+TRAP2+INTERRUPT) OR ENABLE_REFRESH
    IF PCSTS<ENABLE_PTS> THEN
        pc_enable := pc_enable OR ENABLE_PTS;
    PCSTS := pc_enable;

    END;
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Error Recovery Coding

```
PROCEDURE machine_check_handler;

! This procedure is invoked via SCB vector 04 (hex) to process
! machine checks.

    BEGIN

! Use software flag to detect nested machine checks, then set
! the flag and ACK the machine check.

    IF mc_in_progress THEN                  ! nested machine check?
        bugcheck;                           ! yes, bugcheck
    mc_in_progress := TRUE;                 ! indicate machine check in progress
    MCESR := 0;                             ! ACK the machine check

! Collect error state from XRP registers and the machine check frame.

    collect_error_state;                    ! read XRP registers
    s_psl := mc_frame [32];                 ! extract PSL from MC stack frame
    s_code := mc_frame [4]<15:0>;           ! extract machine check code
    s_r := mc_frame [4]<31>;                ! extract VAX restart bits

! Disable P-cache and backup cache during error recovery to minimize
! the chance of nested errors.

    disable_cache;


! machine_check_handler, continued.

! Case on machine check code for per-code processing.

    CASE s_code OF                          ! case on machine check fault code

        MCHK_FP_PROTOCOL_ERROR,
        MCHK_FP_ILLEGAL_OPCODE,
        MCHK_FP_OPERAND_PARITY,
        MCHK_FP_UNKNOWN_STATUS,
        MCHK_FP_RESULT_PARITY:
            BEGIN

            ! F-chip related error
            ! If the F-chip error threshold is exceeded, disable
            ! the F-chip by clearing ACCS<F_CHIP_PRESENT>

            IF (fchip error threshold exceeded) THEN
                ACCS := ACCS AND NOT F_CHIP_PRESENT;
            retry := s_r AND (NOT s_psl<FPD>) AND (NOT s_rcsr<UWP>);

            END;

        MCHK_TBM_ACV_TNV,
        MCHK_TBH_ACV_TNV,
        MCHK_INT_ID_VALUE,
        MCHK_UNKNOWN_IBOX_TRAP,
        MCHK_UNKNOWN_CS_ADDR:
            BEGIN

            ! Internal CPU-chip error
            ! Disable retry if internal error threshold is exceeded

            disable_retry := (internal error threshold exceeded);
            retry := (s_r OR s_psl<FPD>) AND (NOT s_rcsr<UWP>) AND
                     (NOT disable_retry);

            END;
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Error Recovery Coding

```
! machine_check_handler, continued

        MCHK_MOVC_STATUS:
            BEGIN

            ! MOVCx error
            ! Disable retry if the internal error threshold is exceeded,
            ! an instruction specifier used R0-R5 or part of the destination
            ! string, or the source and destination strings overlap.
            ! If retry is to be done, clear PSL<FPD> in the machine check
            ! frame so that the instruction is restarted from the beginning.

            disable_retry = (internal error threshold exceeded);
            specifier_destroyed = (specifier in R0-R5 or string);
            string_overlap = (source and destination strings overlap);
            retry = s_psl<FPD> AND (NOT s_rcsr<UWP>) AND
                    (NOT disable_retry) AND (NOT specifier_destroyed) AND
                    (NOT string_overlap);
            IF retry THEN
                mc_frame [32]<FPD> = 0;

            END;

        MCHK_BUSERR_READ_PCACHE,
        MCHK_BUSERR_READ_DAL:
            BEGIN

            ! P-cache or DAL read error
            ! Perform full memory error recovery

            recover_from_memory_subsystem_errors;
        retry = (s_r OR s_psl<FPD>) AND (NOT s_rcsr<UWP>) AND
                    (NOT s_pcsts<TRAP2>);

            END;

        MCHK_BUSERR_WRITE_DAL,
        MCHK_UNKNOWN_BUSERR_TRAP:
            BEGIN

            ! Unexpected errors
            ! Perform full memory error recovery

            recover_from_memory_subsystem_errors;
            retry = FALSE;

            END;

    ! Fault-specific handling complete.

    ! Conditionally reenable the P-cache and backup cache.

    enable_cache;                       ! Enable cache as appropriate

    ! Log all registers and the machine check frame in the error
    ! log for later analysis.  It is important to log everything for
    ! all errors as this will aid in the analysis of cascaded errors.

    (log error state and machine check parameters);

    ! If the error is to be retried, trim the machine check parameters
    ! from the stack and REI.  If the error is not to be retried,
    ! force image exit, generate a bugcheck, or whatever is appropriate.
```

Example 3–1 Cont'd. on next page

Example 3–1 (Cont.) Error Recovery Coding

```
    mc_in_progress := FALSE;                ! machine check complete
    IF retry THEN                           ! retry?
        BEGIN                               ! yes,
        SP := SP + mc_frame [0] + 4;        ! trim parameters
        REI;                                ! and restart instruction
        END
    ELSE                                    ! no,
        (perform appropriate action);

    END;  ! machine_check_handler

PROCEDURE soft_error_interrupt_handler;

! This procedure is invoked via SCB vector 54 (hex) to process
! soft error interrupts at IPL 1A (hex).

    BEGIN

! Collect error state from XRP registers.

    collect_error_state;

! If cache-related error, disable cache during processing.

    IF s_pcsts<INTERRUPT> OR s_bcsts<STATUS_LOCK> THEN
        disable_cache;

! Recover from errors by performing the memory error recovery
! procedures.

    recover_from_memory_subsystem_errors;

! Log errors if necessary.  Cache-related errors should always
! be logged.  Other errors such as CRD may be counted instead.

    (selectively log errors);

! Make sure cache is back on if turned off.

    IF s_pcsts<INTERRUPT> OR s_bcsts<STATUS_LOCK> THEN
        enable_cache;

! Soft error interrupts should be retried.

    REI;

    END;  ! soft_error_interrupt_handler
```

3.11.12 Error Matrix

Table 3–29 summarizes the KA64A CPU module errors and lists the registers that identify the error or contain additional error information. Subsequent tables list the errors in the order of the SCB entry point and show the effects, responses, and states of various system components after an error has occurred. Since the states shown are caused by the hardware, suggested coding actions are shown in the last column. Each type of error has a table.

Table 3–29 Error Summary

	RSSC	P-Cache		B-Cache		REXMI		
Error	SSCBTR	PCSTS	PCERR	BCSTS	BCERR	XBER	RCSR	XFADR
Machine check								
F-chip protocol error	–	–	–	–	–	–	–	–
F-chip illegal opcode	–	–	–	–	–	–	–	–
F-chip operand parity error	–	–	–	–	–	–	–	–
F-chip unknown status	–	–	–	–	–	–	–	–
F-chip result parity error	–	–	–	–	–	–	–	–
TB miss microcode error	–	–	–	–	–	–	–	–
TB hit microcode error	–	–	–	–	–	–	–	–
Undefined INT.ID	–	–	–	–	–	–	–	–
Undefined MOVCx state	–	–	–	–	–	–	–	–
Undefined I-box trap code	–	–	–	–	–	–	–	–
P-cache read error								
P-cache tag parity error	–	yes	yes	–	–	–	–	–
P-cache data parity error	–	yes	yes	–	–	–	–	–
DAL read errors								
B-cache data parity error	–	yes	yes	–	–	–	–	–
DAL data parity error	–	yes	yes	–	–	–	–	–
RSSC DAL read timeout	yes	yes	yes	yes	–	–	–	–
XMI read error	–	yes	yes	yes	–	yes	yes	yes
DAL write errors								
RSSC DAL write timeout	yes	yes	–	yes	–	–	–	–
Undefined bus microtrap	–	–	–	–	–	–	–	–
Undefined control store address	–	–	–	–	–	–	–	–

Table 3–29 (Cont.) Error Summary

	RSSC	P-Cache		B-Cache		REXMI		
Error	SSCBTR	PCSTS	PCERR	BCSTS	BCERR	XBER	RCSR	XFADR
Hard error interrupt (Vector 60)								
XFAULT assertion	–	–	–	–	–	yes	–	–
Write error interrupt	–	–	–	–	–	yes	–	–
Inconsistent parity error	–	–	–	–	–	yes	–	–
DAL data parity on memory writes	–	–	–	–	–	–	yes	–
Second error	–	–	–	–	–	–	yes	–
Undefined bus microtrap	–	–	–	–	–	–	–	–
XMI write errors	–	–	–	–	–	yes	–	yes
XMI IDENT errors	–	–	–	–	–	yes	–	yes
Soft error interrupt (Vector 54)								
P-cache errors								
P-cache tag parity error	–	yes	–	–	–	–	–	–
P-cache data parity error	–	yes	–	–	–	–	–	–
DAL read errors								
B-cache data parity error	–	yes	–	–	–	–	–	–
DAL data parity error	–	yes	–	–	–	–	–	–
RSSC DAL read timeout	yes	yes	–	yes	–	–	–	–
XMI read error	–	yes	–	yes	–	yes	yes	yes
Cache fill errors	–	yes	–	–	–	yes	yes	yes
Other REXMI errors	–	–	–	–	–	yes	–	–
C-chip tag store error	–	–	–	yes	yes	yes	–	–
C-chip protocol error	–	–	–	yes	–	yes	–	–
Errors with no notification								
Write data NO ACK	–	–	–	–	–	yes	–	–
Read/IDENT data NO ACK	–	–	–	–	–	yes	–	–

Table 3–30 P-Cache Tag Parity Error

DAL Command Type	Effect on CPU Execution	Effect on P-Cache	Effect on B-Cache	Effect on Main Memory	State Captured on Error	Actions to Take
D-stream read hit	Machine check	Disable	–	–	PCSTS, PCERR	Rewrite tags, reenable P-cache
D-stream read miss	Soft error interrupt	Disable	–	–	PCSTS	Rewrite tags, reenable P-cache
I-stream read	Soft error interrupt	Disable	–	–	PCSTS	Rewrite tags, reenable P-cache
Write	Soft error interrupt	Disable	–	–	PCSTS	Rewrite tags, reenable P-cache
Invalidate	Soft error interrupt	Disable	–	–	PCSTS	Rewrite tags, reenable P-cache

Table 3–31 P-Cache Data Parity Error

DAL Command Type	Effect on CPU Execution	Effect on P-Cache	Effect on B-Cache	Effect on Main Memory	State Captured on Error	Actions to Take
D-stream read	Machine check	Disable	–	–	PCSTS, PCERR	Flush and reenable P-cache
I-stream read	Soft error interrupt	Disable	–	–	PCSTS	Flush and reenable P-cache

Table 3–32 DAL Data Parity Error

DAL Command Type	Effect on CPU Execution	Effect on P-Cache	Effect on B-Cache	Effect on Main Memory	State Captured on Error	Actions to Take
D-stream read (requested)	Machine check	Disable	–	–	PCSTS, PCERR	Flush and reenable P-cache and B-cache
D-stream read (nonrequested)	Soft error interrupt	Disable	–	–	PCSTS	Flush and reenable P-cache and B-cache
I-stream read	Soft error interrupt	Disable	–	–	PCSTS	Flush and reenable P-cache and B-cache

Table 3–33 RSSC DAL Timeout

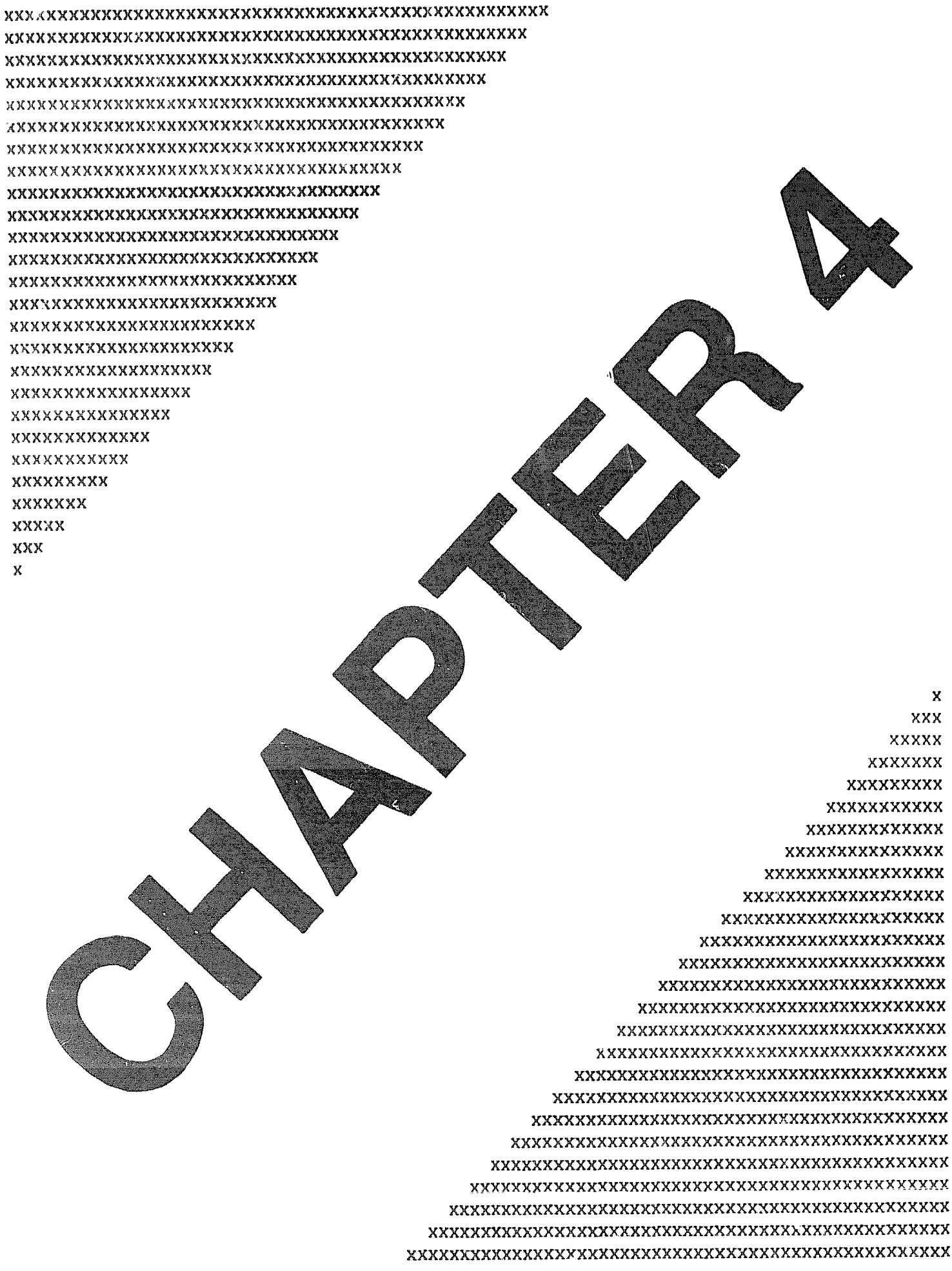
DAL Command Type	Effect on CPU Execution	Effect on P-Cache	Effect on B-Cache	Effect on Main Memory	State Captured on Error	Actions to Take
D-stream read	Machine check	Disable	Disable	–	PCSTS, PCERR, BCSTS, SSCBTR	Flush and reenable P-cache and B-cache, clear RSSC timeout
I-stream read	Soft error interrupt	Disable	Disable	–	PCSTS, BCSTS, SSCBTR	Flush and reenable P-cache and B-cache, clear RSSC timeout
Write	Machine check	Disable	Disable	–	PCSTS, BCSTS, SSCBTR	Flush and reenable P-cache and B-cache, clear RSSC timeout
Clear write buffer	Machine check	Disable	Disable	–	PCSTS, BCSTS, SSCBTR	Flush and reenable P-cache and B-cache, clear RSSC timeout

Table 3–34 Backup Cache Tag Parity Error

DAL Command Type	Effect on CPU Execution	Effect on P-Cache	Effect on B-Cache	Effect on Main Memory	State Captured on Error	Actions to Take
D-stream read	Soft error interrupt	–	Force miss, Disable	–	BCSTS, BCERR	Flush and reenable B-cache
I-stream read	Soft error interrupt	–	Force miss, Disable	–	BCSTS, BCERR	Flush and reenable B-cache
Write	Soft error interrupt	–	Force miss, Disable	–	BCSTS, BCERR	Flush and reenable B-cache
Invalidate	Soft error interrupt	–	Force miss, Disable	–	BCSTS, BCERR	Flush and reenable B-cache
Inval-Bus lookup	Soft error interrupt	–	Force miss, Disable	–	BCSTS, BCERR	Flush and reenable B-cache

Table 3–35 XMI Error

DAL Command Type	Effect on CPU Execution	Effect on P-Cache	Effect on B-Cache	Effect on Main Memory	State Captured on Error	Actions to Take
D-stream read (requested)	Machine check	Disable	Disable	May log error	PCSTS, PCERR, BCSTS, XBER, RCSR, XFARD	Flush and reenable P-cache and B-cache
D-stream read (nonrequested)	Soft error interrupt	–	Fill not done	May log error	XBER, RCSR, XFARD	–
I-stream read (requested)	Soft error interrupt	Disable	Disable	May log error	PCSTS, PCERR, BCSTS, XBER, RCSR, XFARD	Flush and reenable P-cache and B-cache
I-stream read (nonrequested)	Soft error interrupt	–	Fill not done	May log error	XBER, RCSR, XFARD	–
Write	Hard error interrupt	–	–	Write suppressed	XBER, XFADR	–
Read interrupt vector	Hard error interrupt	–	–	–	XBER, XFADR	–


CHAPTER 4

4 FV64A Vector Processor Module

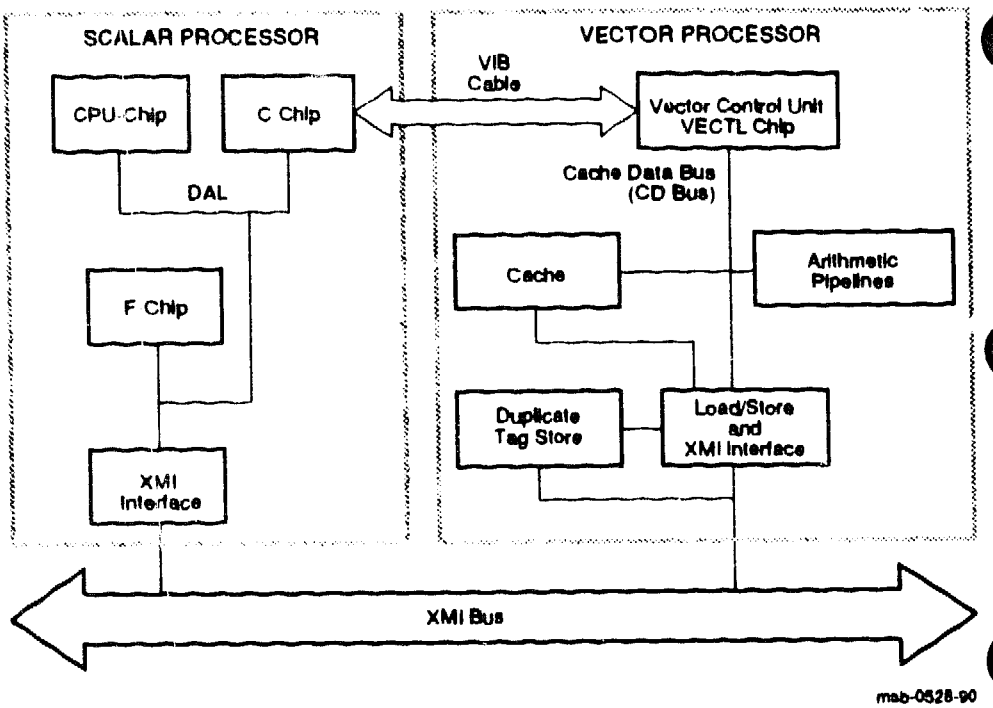
This chapter describes the FV64A vector processor module. The chapter includes the following sections:

- Overview
- Functional Units
- Block Diagram Description
- Vector Control Unit
- Arithmetic Unit
- Load/Store Unit
- Cache Memory
- Vector Processor Registers
- Error Handling

4.1 Overview

The FV64A vector processor is a single-board option that implements the VAX vector instruction set. Figure 4–1 is a block diagram of a scalar/vector pair.

Figure 4–1 Scalar/Vector Pair Block Diagram



The FV64A vector processor requires a scalar KA64A CPU module for operation. Together they implement the base instruction group of the VAX architecture plus the 63 vector instructions.

The scalar and vector modules occupy adjacent slots in the XMI cage. All communication between the scalar and vector modules takes place across the vector interface bus (VIB), a cable that connects the two modules at the rear edges of the modules.

Instructions to the vector module pass over the VIB from the C-chip on the scalar module to the VECTL chip on the vector module. In addition to controlling the operations on the vector module, the VECTL chip also returns status to the scalar processor. All error reporting is done by the scalar processor.

The FV64A vector processor is a standard XMI module. but it is not a full-fledged XMI node. The vector processor functions only as an XMI commander. It performs memory transactions over the XMI bus without intervention from the scalar module. The FV64A vector processor does not issue transactions to I/O space or respond to XMI transactions directed to it.

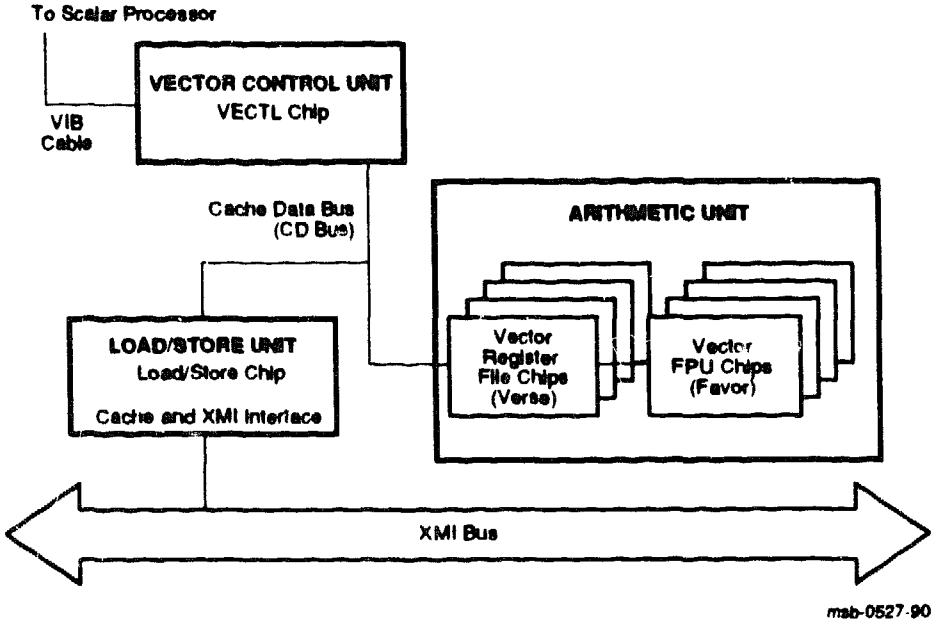
A multiprocessor system can have two scalar/vector pairs. For optimal performance, two memory modules are required with a scalar/vector pair. Four memory modules are required with two scalar/vector pairs.

The FV64A vector processor, when in a Model 400 system, requires that the scalar CPU modules meet certain revision requirements. See the *VAX 6000 Series Upgrade Manual* for those requirements.

4.2 Functional Units

Figure 4–2 shows the three functional units of the FV64A vector processor module — the vector control unit, the arithmetic unit, and the load/store unit, which includes the XMI interface.

Figure 4–2 FV64A Vector Processor Functional Units



The FV64A vector processor module has three functional units. All operations of the vector control unit are implemented by the VECTL chip. The arithmetic unit consists of the arithmetic pipelines, which consist of four pairs of register file and FPU chips. The load/store unit controls cache operations, and it performs reads and writes to memory or to the arithmetic pipelines.

The VECTL chip passes instructions to the Load/Store chip over the CD bus. The Load/Store chip then issues XMI memory transactions. Some instructions are issued directly to the arithmetic pipelines to be executed by the register file (Verse) and FPU (Favor) chips. The 16 vector data registers are in the Verse chips. The Favor chips perform the arithmetic operations on the data held in the Verse chips.

The vector processor can perform the overlapped execution of arithmetic and load/store instructions. This capability is possible because the VECTL chip conforms to rules dealing with the issue of instructions. It will not issue instructions that require the results of a previous instruction. The vector processor also supports chaining, a special form of instruction overlap that is possible with multiple function units. The results of an arithmetic instruction can be stored into memory while the arithmetic instruction is still executing.

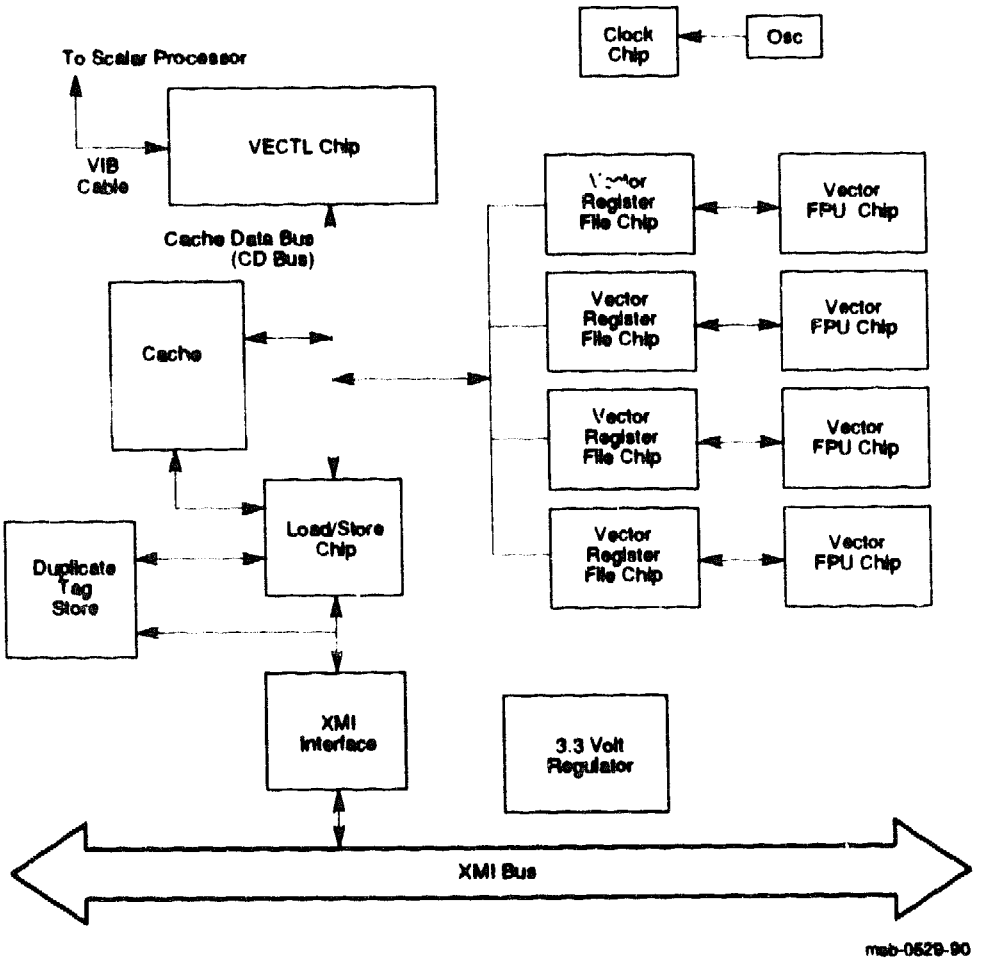
The three functional units are described in more detail in these sections:

- Vector control unit — Section 4.4
- Arithmetic unit — Section 4.5
- Load/store unit — Section 4.6

4.3 Block Diagram Description

Figure 4–3 is a block diagram of the vector module. This section summarizes the major chips on the module.

Figure 4–3 FV64A Vector Processor Block Diagram



The FV64A vector processor chipset consists of five core chips, as follows:

- VECTL, DC582—Vector instruction issue and scalar/vector interface chip
- Verse, DC555—Vector register file chip, 4 chips
- Favor, DC556—Vector arithmetic data path chip, 4 chips
- Load/Store, DC560 — Vector module translation buffer, cache, and XMI interface controller chip
- Clock, DC521—Clock generation chip (same as on scalar module)

The module also has a regulator that converts 5 volts to 3.3 volts.

4.3.1 Vector Control Chip

The vector control (VECTL) chip performs the following functions:

- Interface to the scalar processor. The VECTL chip receives instructions from the scalar module and also returns status.
- Instruction issue. The VECTL chip issues instructions to the other functional units on the vector module and maintains a register scoreboard for the detection of interinstruction dependencies.
- Cache data (CD) bus master control. The VECTL chip controls the CD bus. It relinquishes partial control to the Load/Store chip during execution of load/store instructions.
- IOTA instruction execution takes place in the VECTL chip.
- Implementation of the architecturally defined registers (VLR, VCR, VPSR, VAER, and VMAC). The VECTL chip also has registers that provide access to vector indirect address space (VIADR, VIDHI, and VIDLO).

4.3.2 Vector Register File Chip

The vector register file (Verse) chip is the interface between the Favor (vector FPU) chip and the rest of the vector module. Among its features are:

- It contains one quarter (2 Kbytes) of the storage needed to implement the VAX vector architecture (8 Kbytes). Four Verse and four Favor chips implement the full architecture.
- It provides four ports on the register file: a 64-bit, read/write port to the CD bus for loads and stores, a 32-bit (64-bit internal) read port for operand A, a 32-bit (64-bit internal) read port for operand B, and a 32-bit (64-bit internal) write port for results. A load or store instruction can be writing or reading the registers at one port, and an arithmetic instruction can be reading its operands out of two other ports. Another arithmetic instruction can be writing its results from still another port. All three operations can be done in parallel. When two longword operands are packed into the quadword, two separate Verse chips can each select the appropriate longword.
- It contains registers for holding two instructions, two scalar operands, the vector length embedded in the instruction, and the vector mask. One instruction register is for the current instruction, and the other is for the next instruction to be executed, known as the deferred instruction register. The operand registers are also for the current and deferred operands.
- It performs the vector logical and vector merge instructions and formats integer operations so that they can be executed by the Favor chip.

4.3.3 Vector FPU Chip

The vector FPU (Favor) chip is a pipelined floating-point processor. The Favor chip offers high-performance, pipelined vector floating-point computation for the vector module. Among its features are:

- VAX vector floating-point instructions and data types. The Favor chip implements instruction and data type support for all VAX vector floating-point instructions as well as the integer multiply operation. Floating-point data types F_, D_, and G_floating are supported.
- High-throughput external interface. The Favor chip receives two 32-bit operands from the Verse chip through the multiplexed A/B port every cycle. It drives back a 32-bit result to the Verse chip through the C port in the same cycle.
- Based on the floating-point accelerator chip (the F-chip, DC523) on the KA64A CPU module.
- Five-stage pipelined data path.

4.3.4 Load/Store Chip

The Load/Store (L/S) chip implements the following functions:

- Execution of all load, store, gather, and scatter instructions.
- Virtual address generation logic for memory references.
- Virtual-to-physical address translation logic, using a translation buffer. A 136-entry translation buffer is on the Load/Store chip. The chip also contains the data path and control necessary to implement full VAX memory management (with assistance from the scalar CPU).
- Cache control. The Load/Store chip supports the tag and data store for a 1-Mbyte write-through data cache. It also supports a duplicate tag store for invalidate filtering.
- XMI interface. The Load/Store chip serves as the interface between the vector module and the XMI bus. This includes support for four outstanding cache misses on read requests and a 32-entry write buffer to permit half the data from one store/scatter instruction to be held in the buffer. The performance of the high-speed CD bus can thus be isolated from the performance impact of the slower XMI bus.

4.3.5 Clock Chip

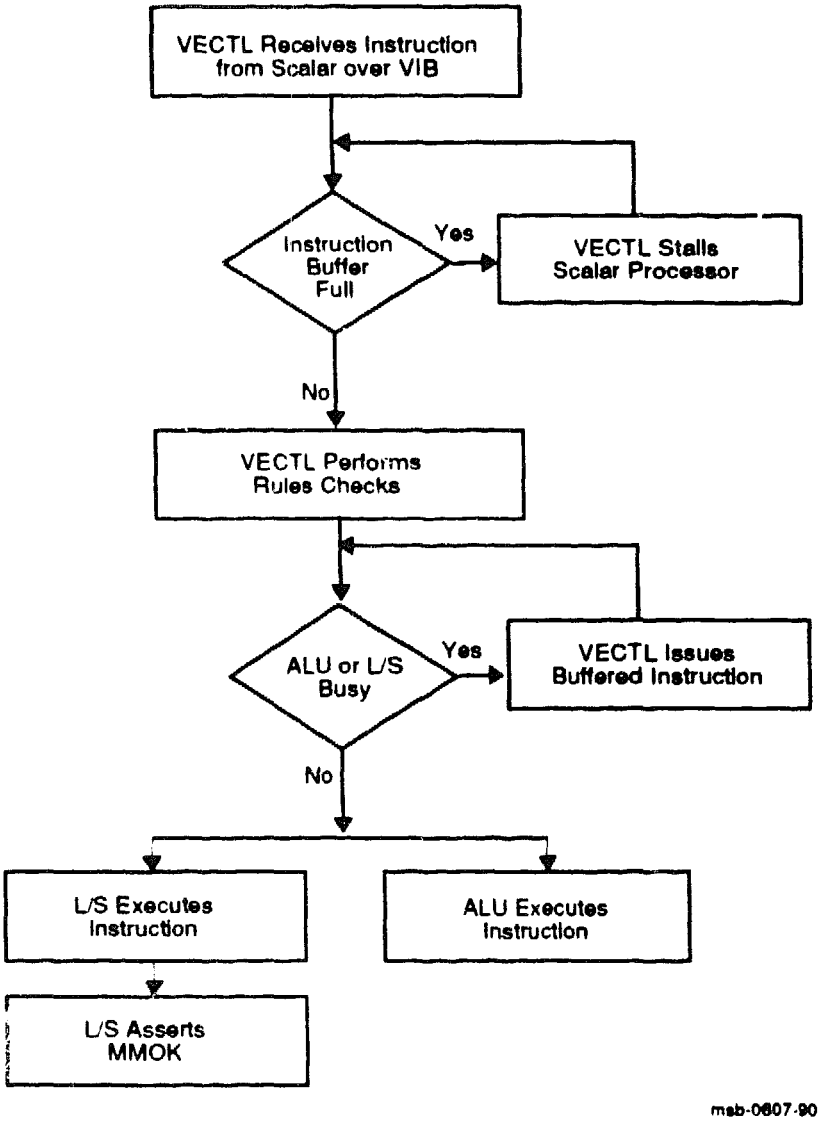
The clock chip (CLK-chip, DC521) is the same chip that is on the KA64A CPU module. It provides the main system clocks to all chips on the vector module except for the Favor chips, which get their clocks from the Verse chips.

4.4 Vector Control Unit

The vector control (VECTL) chip performs these functions:

- **Receives vector instructions from the scalar module and then controls the execution of those operations.**
- **Returns status to the scalar module.**

Figure 4–4 VECTL Chip Instruction Flow



4.4.1 Instruction Flow

Vector instructions are received by the VECTL chip, which buffers the instructions and controls instruction issue to other functional units in the vector module. Figure 4–4 shows the instruction flow from the VECTL chip.

The VECTL chip decomposes the instruction into its opcode and operands, and then performs a check of issue rules (see Section 4.4.2). All source and destination registers must be available before instruction execution begins. The functional unit to be used (arithmetic or load/store) by the instruction during execution must also be available. The instruction is not issued until all required resources are available.

For arithmetic operations, the scalar CPU can then proceed. However, for load/store instructions, the scalar CPU is stalled waiting completion of all address translations. This guarantees that any memory management violations are synchronous, and the scalar CPU can restart the faulting instruction correctly.

The execution of arithmetic instructions and load/store instructions can overlap, because the functional units are independent. To achieve this overlap, the following conditions must be met:

- The arithmetic instruction must be issued before the load/store instruction.
- There must be no register conflict between the arithmetic and load/store instructions.

After issuing a load/store instruction, the VECTL chip must not issue new instructions until the Load/Store chip asserts MMOK, indicating that there will not be any more memory management exceptions.

The interaction between the different functional units of the vector module can greatly affect the performance/execution of vector instructions. For more information on effective use of the vector processor, see the *VAX 6000 Series Vector Processor Programmer's Guide*.

4.4.2 Instruction Issue Rules

The availability of registers is handled by a method called "scoreboarding." The rules governing register availability depend on the type of instruction to be issued.

- For a load instruction, the register to be loaded must not be modified by any currently executing (arithmetic) instruction, and it must not be modified or used as input by any currently deferred (arithmetic) instruction.
- For a store instruction, the register to be stored must not be modified by any currently executing or deferred instruction, but it may be in use as input. The exception is when a chain into store may occur. In this case the store instruction can be issued while the chaining arithmetic instruction is still executing.
- For a scatter or gather instruction, the restrictions for a load or store instruction apply, but also the register containing the offset to be used in the scatter or gather instruction must not be modified by any currently executing or deferred instruction.
- For a load or store under mask instruction, the restrictions for a load or store instruction apply, but also the mask register must not be modified by any currently executing or deferred instruction.
- An arithmetic instruction can be issued as soon as the deferred instruction queue of the arithmetic unit is free. Register checking for these instructions is handled by the arithmetic unit.

In general, there must be no outstanding writes to a needed register from prior instructions, and the destination register of the instruction must not be used by a currently deferred instruction.

4.4.3 Data Types

The FV64A vector processor supports five data types:

- Longword
- Quadword
- F_floating
- G_floating
- D_floating

4.4.4 Instruction Set

The vector processor implements 63 vector instructions, as defined in the *VAX Architecture Reference Manual*.

The vector processor supports the following instruction classes:

- Load/Store
- Gather/Scatter
- Masked Load/Store and Gather/Scatter
- IOTA
- VMERGE and Arithmetic
- MFVP/MTVP
- MFPR/MTPR
- SYNC
- MSYNC
- VSYNC

When an illegal instruction is received by the VECTL chip, the instruction is decoded and placed in the Illegal Instruction Register. The vector processor is disabled, and a hard error interrupt is posted. The Vector Controller Status Register <ILL> bit is set to indicate an illegal instruction.

The following list describes by instruction type the flow of instructions in the machine.

4.4.4.1 Load Instruction

When a load instruction is received by the VECTL chip, it checks the destination vector register against outstanding arithmetic instructions. If there are no register usage conflicts, the instruction is sent to the Load/Store chip.

4.4.4.2 Store Instruction

When a store instruction is received by the VECTL chip, it checks the source vector register against outstanding arithmetic instructions. If there are no conflicts, the instruction is sent to the Load/Store chip. If the source for the store is the destination of the current arithmetic instruction, and the deferred arithmetic instruction does not conflict with the source vector register, and the arithmetic instruction is not a divide, then the VECTL chip waits for a signal from the Verse chips to indicate that the store operation can start. The instruction is then sent to the Load/Store chip.

4.4.4.3 Gather/Scatter Instructions

When a gather or scatter instruction is received by the VECTL chip, it checks the destination/source register against outstanding arithmetic instructions. If there are no conflicts, the instruction is sent to the Load/Store chip. The Load/Store chip will then fetch the offset vector register. When this is complete, the VECTL chip will reissue the instruction, and the gather/scatter operation will take place using the previously stored offset vector register to generate the virtual addresses. Scatter instructions also permit chain-into-store.

4.4.4.4 Masked Load/Store and Gather/Scatter Instructions

The operation for masked memory instructions is identical to the unmasked versions except the following operations are performed first. The VECTL chip checks if any outstanding arithmetic instructions will modify the mask register. If not, the VECTL chip reads the mask from the Verse chips and sends it to the Load/Store chip. The sequence is then performed as above.

4.4.4.5 IOTA Instruction

When an IOTA instruction is received by the VECTL chip, it checks the destination vector register against outstanding arithmetic instructions. The outstanding arithmetic instructions are checked for modification of the mask register as well. If there are no conflicts, the VECTL chip reads the mask register from the Verse chips. The VECTL chip then executes the instruction.

4.4.4.6 VMERGE and Arithmetic Instructions

When the instruction is received and there is room in either the Deferred ALU Instruction Register or the Current ALU Instruction Register, the instruction is loaded. The dispatch control checks for an outstanding load/store or IOTA in progress. If none are found, the instruction is written into the Verse instruction queue.

4.4.4.7 MFVP/MTVP Instructions

When a request is made from/to a control register, the VECTL chip performs the read/write transaction for the appropriate register. If the scalar processor sends an MFVP/MTVP instruction before it recognizes that the vector processor is disabled, the VECTL chip still honors the request. The vector disable fault is taken when the scalar CPU recognizes that the vector processor is disabled, and then an instruction or MFVP/MTVP is encountered in the I-stream.

4.4.4.8 MFPR/MTPR Instructions

When a request is made to read or write an internal processor register, the vector processor honors the request and performs the required operation independent of state. The scalar processor issues MFPR/MTPR instructions even if the vector processor is disabled. The scalar CPU must be in kernel mode to permit the instruction to be issued.

4.4.4.9 SYNC Instruction

The SYNC instruction is a special case of the MFVP instruction. This instruction stalls the scalar processor until the Busy bit of the Vector Processor Status Register (VPSR) is clear, which indicates that the vector processor has completed all instructions in its queues.

4.4.4.10 MSYNC Instruction

The MSYNC instruction is a special case of the MFVP instruction. This instruction stalls the scalar processor until the Load/Store chip has completed all memory transactions. Executing an MSYNC instruction guarantees that all memory activity is complete and that the invalidate queue is empty.

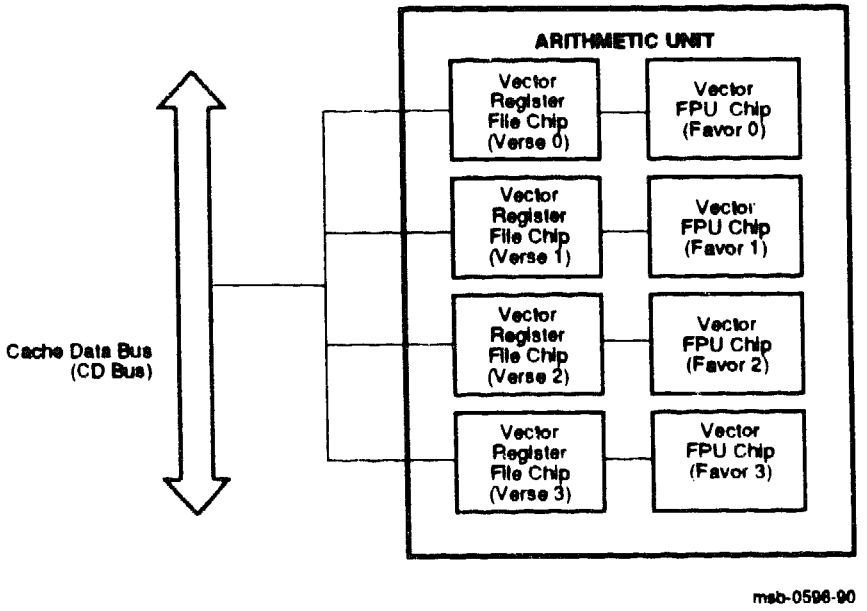
4.4.4.11 VSYNC Instruction

The scalar processor sends this instruction to the VECTL chip. The design of the vector processor does not allow multiple outstanding load/store instructions. This means that the VSYNC instruction is ignored when it is decoded.

4.5 Arithmetic Unit

The arithmetic unit (ALU) consists of four pairs of vector register file chips and FPU chips. Vector instructions are received from the VECTL chip.

Figure 4–5 Vector Arithmetic Unit



All register-to-register vector instructions are handled by the arithmetic unit (ALU). Figure 4–5 shows the arithmetic unit and its four pairs of chips: vector register file (Verse) chips and the vector FPU (Favor) chips. The Favor chips perform all floating-point instructions as well as compare, shift, and integer multiply. (There is no integer divide vector instruction.) The Verse chips do everything else, including merge and logical instructions and all initial instruction handling. The parts of the arithmetic unit appear to perform as a single unit.

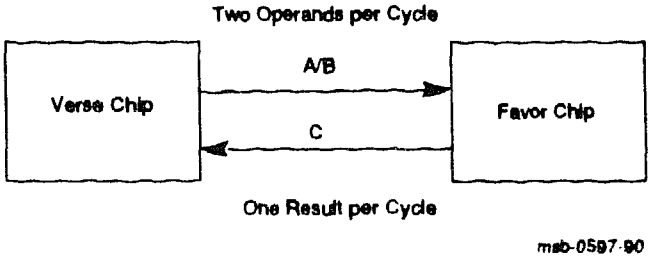
All instructions, except floating-point divide instructions, are fully pipelined. For increased performance all arithmetic instructions are executed by four parallel pipelines. After an initial startup time, the fully pipelined instructions can produce four results each cycle for single-precision instructions and after every other cycle for double-precision instructions.

Each vector register file chip contains every fourth element of the vector registers, thus permitting four-way parallelism. These chips receive instructions from the VECTL chip and data from the cache or load/store, read operands from the registers, and write results back into the registers or into the mask register. If two 32-bit operands come over in a single 64-bit transfer, they can be read or written by two separate register file chips.

The register set has four 64-bit ports (one read/write for memory data, two for read operands, and one for writing results). The register file is implemented with one read/write port and one read-only port, and can be accessed twice per cycle. While one instruction is writing its results, a second can start reading its operands, thus hiding the instruction pipeline delay. Variations in pipeline length between instructions are smoothly handled so that no gaps exist in the flow of write data. The register file can hold two outstanding arithmetic instructions in its internal queue. The arithmetic unit executes two arithmetic instructions in about the time it takes one load or store operation to take place. The data from the register file flows to the vector FPU chip.

Input data to the vector FPU chip comes over a 32-bit bus that is driven twice per cycle, and results are returned on a separate 32-bit bus that is driven once per cycle (see Figure 4–6). The two operands for single-precision instructions can be passed in one cycle, while double-precision operands require two cycles. The FPU chip has a throughput of one cycle per single-precision operation, two cycles per double-precision operations, and 10 or 22 cycles per single- or double-precision divide. Its pipeline delay varies for different operations; for example, the pipeline delay is 5 cycles for all longword-type instructions and 6 cycles for all double-precision instructions except multiply.

Figure 4–6 Verse/Favor Chip Bus Operation



An instruction continues executing until all results are completed. A deferred arithmetic instruction begins execution after the instruction in the pipeline completes or when all the following conditions are met:

- The deferred instruction must not be a "short" instruction; that is, the vectors used by the instruction must be at least eight elements in length.
- The current instruction must not be a "long" instruction; that is, the instruction must not require more than two cycles per element to execute. (The divide instructions are the only "long" instructions.)

In other words, overlap of instruction execution can occur if the results of the deferred instruction will not be completed before the last results from the current instruction. The overlap of instructions will be particularly significant for shorter vectors.

4.6 Load/Store Unit

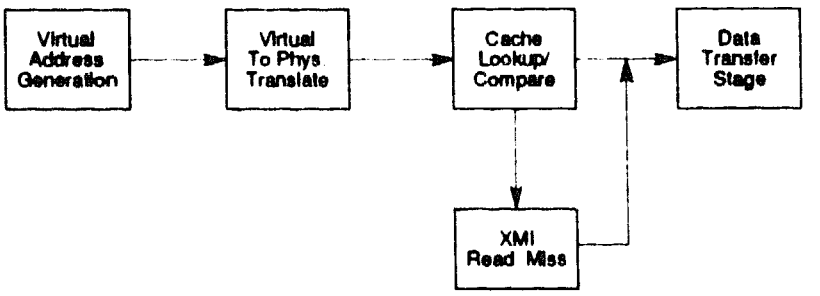
Vector instructions are received by the vector control unit (the VECTL chip) from the scalar processor and then issued to either the load/store unit or the arithmetic unit. The Load/Store chip controls the load/store unit, which consists of a 1-Mbyte cache and the XMI interface logic. There is a duplicate tag store and a 32-element write buffer.

The load/store unit handles all cache and memory accesses for the vector module. It controls the vector cache and includes the XMI interface. The vector module implements a single-level, direct-mapped cache. In addition, the Load/Store chip has a 32-element write buffer that can hold the data and addresses for one vector store or scatter instruction.

Figure 4–7 shows the flow of address and data in the load/store pipeline. Each stage is a single or multiple stage based on the 44.44-ns vector module clock. The XMI stage is a multiple of 64 ns, and the time taken depends on the transaction type and the XMI bus activity. All memory references must flow through the cache stage.

Once a load/store instruction is issued, the load/store unit becomes bus master and controls the internal cache data (CD) bus. When a load/store instruction starts execution, no further instructions can be issued on the CD bus until the instruction completes. The scalar processor is also stalled until the vector module indicates that it is ready to accept more instructions. The Load/Store chip handles the memory reference instructions, the address translation, the cache management, and the memory bus interface.

Figure 4–7 Address/Data Flow In Load/Store Pipeline



If a memory instruction uses register offsets, the offset register is first read into a buffer and then each element of the offset register is added to the base. This saves having to turn around the internal bus for each offset read. If a register offset is not used, addresses are generated by adding the stride to the base. The stride is the number of memory locations (bytes) between the starting address of consecutive vector elements. This virtual address is then translated to a physical address by using an on-chip 136-entry, 68-way fully associative translation buffer. Two entries are checked at once by an address predictor looking for "address translation successful" on the last element (the assertion of the MMOK signal). The early prediction permits the scalar processor to be released and appear to be asynchronous on memory reference instructions. The Load/Store chip handles translation buffer misses on its own but returns the necessary status on invalid or swapped-out pages. When the scalar processor corrects the situation, the instruction is retried from the beginning.

Once a physical address is obtained, the Load/Store chip looks it up in the 32K entry tag store. The address is delayed and passed to the 1-Mbyte cache data store. This delay permits cache lookup to complete before data is written to the cache on store operations. In parallel, the corresponding register file address is presented to the four register file chips. The data and addresses are automatically aligned for load and store operations to permit the correct reading and writing of the register file and cache data RAMs. Up to four cache misses can be outstanding before the read data for the first miss returns, and hits can be taken under misses. Cache parity errors cause the cache to be disabled, and the instruction retried When the instruction completes, a soft error interrupt is sent to the scalar processor.

A duplicate copy of the cache tag store is maintained for filtering cache invalidates from the main memory bus. The cache is write through, with a 32-element write buffer, and memory read instructions that hit in the cache can start while the memory write instructions are emptying the write buffer. The cache fill size is 32 bytes. The entire process is pipelined so that a new 64-bit word can be read or written each cycle.

The load/store unit includes a five-segment pipeline that can accept a new element every cycle. In general, the load/store pipeline handles a single element request at a time. The exception occurs when a load or store instruction is acting on single-precision, unity vector stride data. In this special case, consecutive elements are paired and then each pair is handled as a single request by the load/store pipeline.

The load/store unit does not respond to XMI transactions.

There are several ways to maximize instruction execution overlap. A load/store instruction can execute in parallel with up to two arithmetic instructions, provided the arithmetic instructions are issued first and there are no register conflicts, and chain into store can reduce the perceived execution time of a store instruction. Also, early MMOK allows the scalar/vector communications to be overlapped with load/store instruction execution.

The handling of translation buffer misses has a large effect on the performance of nonunity stride vectors. A stride of two pages (256 longwords or 128 quadwords) or more can result in a TB miss for each data item. A stride of eight pages (1024 longwords or 512 quadwords) or more can result in a TB miss that can cause a cache miss for each data item. Unity stride is most efficient in that it runs sequentially through the data and makes full use of all PTEs fetched.

The write size for a store operation depends upon the stride. The write size for a store with unity stride is an octaword, whereas the write size for a store with nonunity stride is a quadword.

Nonunity stride loads and stores significantly impact the performance level of the XMI memory bus as compared to unity stride operations. A far greater number of memory references are required for nonunity stride than is the case for unity stride. If the ratio of cache-miss load/store to arithmetic instructions is sufficiently high and nonunity stride is used, XMI bus bandwidth can become the limiting performance factor.

For more information on effective use of the vector processor, see the *VAX 6000 Series Vector Processor Programmer's Guide*.

Up to four XMI load operations can be queued at one time. The pipeline continues processing vector element loads until a fourth cache miss occurs. At that point the cache miss queue is full and the pipeline stalls. The pipeline remains stalled until one of the cache misses is serviced. Cache misses on a load instruction degrade the performance of the load/store pipeline.

A cache miss is serviced by a hexword fill. On the XMI a hexword transfer is 80 percent efficient since one address is sent to receive four quadwords of data; an octaword transfer is 67 percent efficient since one address is sent to receive two quadwords of data; a quadword transfer is only 50 percent efficient since one address is sent to receive one quadword of data. For this reason, stores are more efficient with unity stride than with nonunity stride. A larger piece of memory can be referenced by a single address so that fewer memory references are required.

In the case of load instructions, the comparison of unity and nonunity stride is less straightforward. A nonunity stride cache miss load causes a full hexword to be read from memory even though the load requires only a longword or quadword of data. If the additional data is not referenced by subsequent load instructions, then the nonunity stride load is much less efficient than a unity stride load. If subsequent loads do reference the extra data, then nonunity stride load performance improves due to higher cache hit rates for the subsequent loads. For double-precision data there is little degradation due to nonunity stride in this case. For single-precision data, unity stride loads will always show significantly higher performance because of the load/store pipeline optimization for single-precision unity stride loads.

The Load/Store chip has a write buffer that holds half of a vector register; that is, 32 64-bit elements. However, all 64 elements can be packed into the write buffer for single-precision, unity stride store operations. An XMI write transaction is started as soon as the first element is written into the write buffer.

4.6.1 Memory Management

The vector processor implements memory management as described in the *VAX Architecture Reference Manual*. During normal system operation memory management is always enabled. Memory management can be disabled for diagnostic purposes.

4.6.1.1 MMOK Signal

The Load/Store chip asserts MMOK (Memory Management Okay) when all memory references are successful. The assertion of MMOK indicates that the scalar processor can then issue another instruction to the vector processor.

4.6.1.2 Access Mode

Access control is the function of validating whether a particular type of memory access is permitted to a particular page. The access is determined using the following parameters:

- The virtual page number.
- The appropriate length registers.
- The processor mode.
- The protection code in the page table entry (PTE).
- The alignment of the address is correct (must be naturally aligned).
- The intended reference when translated is not an I/O space reference.

The access mode of a running process is the processor mode at the time the vector instruction is issued. The access mode is sent to the VECTL chip when the instruction is issued.

The vector processor is not permitted to access I/O space. If the virtual-to-physical translation results in an address in I/O space, an I/O space reference fault is generated.

If a protection check violation, a length violation, an I/O space reference, or a vector alignment fault occurs, the vector module passes status back to the scalar CPU indicating that an ACV fault occurred.

4.6.1.3 Memory Management Control Registers

The vector processor has memory management control registers that duplicate those in the scalar processor. These are automatically updated whenever changes are made to the registers in the scalar processor.

Three internal processor registers (IPRs) control memory management:

- IPR56, Memory Management Enable Register (MAPEN)
- IPR57, Translation Buffer Invalidate All Register (TBIA)
- IPR58, Translation Buffer Invalidate Single Register (TBIS)

Three pairs of IPRs specify the base and length of P0, P1, and S0 space:

- IPR8, P0 Base Register (P0BR)
- IPR9, P0 Length Register (P0LR)
- IPR10, P1 Base Register (P1BR)
- IPR11, P1 Length Register (P1LR)
- IPR12, System Base Register (SBR)
- IPR13, System Length Register (SLR)

One IPR and three vector indirect registers are used by diagnostic software to test the vector translation buffer:

- IPR147, Vector Translation Buffer Invalidate All (VTBIA)
- RFA 509, Translation Buffer Control Register (LSX_TBCSR)
- RFA 530, Translation Buffer Tag Register (LSX_TBTAG)
- RFA 531, Translation Buffer PTE Register (LSX_PTE)

The vector processor uses another IPR for memory synchronization between the scalar and vector modules:

- IPR146, Vector Memory Activity Check Register (VMAC)

4.6.2 Translation Buffer

The translation buffer (TB) contains 136 page table entries (PTEs). The TB has 68 associative tags with two PTEs per tag. The TB uses a round-robin replacement algorithm. When a TB miss occurs, two PTEs (1 quadword) are fetched from cache. If the fetch from cache results in a cache miss, eight PTEs (one hexword) are loaded into cache from main memory. Two PTEs are installed in the TB.

The translation buffer can be invalidated by executing a TB flush. This is accomplished either by writing the VTBIA register or by writing the scalar TBIA or TBIS registers with the desired virtual address to invalidate a single location.

Modifying any of the scalar IPRs associated with memory management also flushes the translation buffer.

4.7 Cache Memory

The vector module implements a single-level, direct-mapped cache. The cache is write through, and the size is one Mbyte.

The vector processor implements a 1-Mbyte cache, direct-mapped, with a fill of a hexword (block) and a hexword allocate (block size). The cache is read allocate, no-write allocate, and write through. There are 32K tags, and each tag maps one hexword block. Each tag contains one block valid bit, a 9-bit tag, and one parity bit. Each data block contains 32 bytes and 8 parity bits, one for each longword.

Associated with each of the 32K main tags is a duplicate tag in the XMI interface. This tag is allocated in parallel with the main tag and is used for determining invalidates. All XMI write command/address cycles are compared with the duplicate tag data to determine if an invalidate should take place. The resulting invalidate is placed in a queue for subsequent processing in the main tag store.

4.7.1 Cache Organization

The cache is a 1-Mbyte, direct mapped, write-through cache. The cache is arranged in 32K rows with four quadwords and one tag entry per row. Figure 4–8 shows the cache arrangement. Figure 4–9 shows how the physical address is divided.

Figure 4–8 Cache Arrangement

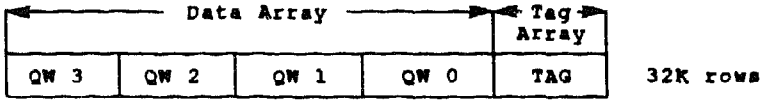
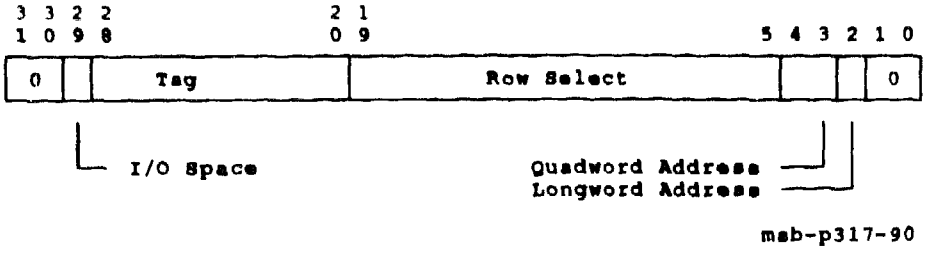


Figure 4–9 Physical Address



The physical address passed to the cache is 27 bits long and is longword aligned. Bit <29> is never passed to the cache, because an I/O space reference generates a memory management exception in the translation buffer. Bits <28:20> are compared to the tag field. Bits <19:5> provide the row select for the cache, bits <4:3> supply the quadword address, and bit <2> supplies the longword address.

Figure 4–10 shows how the main tag memory is arranged. The main tag is written with PA<28:20>, and the valid bit covers a hexword block. The parity bit covers the tag and valid bits. The duplicate tag is identical to the main tag memory.

Figure 4–10 Tag Entry Organization

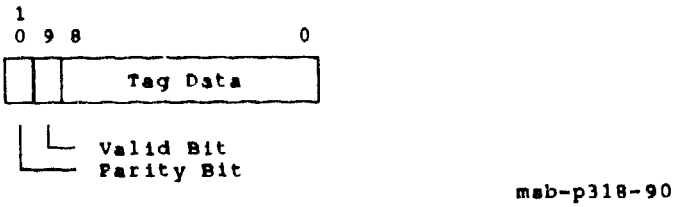
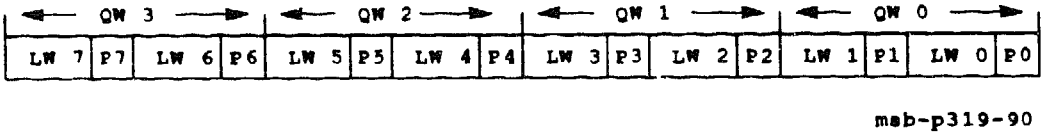


Figure 4–11 shows the organization of the cache data. Each cache block contains four quadwords, with eight longword parity bits.

Figure 4–11 Cache Data Organization



4.7.2 Cache Coherency

All data cached must remain coherent with data in main memory. This means that any write done by a processor or I/O device must displace data cached by all processors.

The XMI interface in the Load/Store chip continuously monitors all XMI write transactions. When a write is detected, the address is compared with the contents of a duplicate tag store to determine if the write should displace data in the main cache. If the write requires that the main cache tag be invalidated, then an invalidate queue entry is generated. The duplicate tag store is a copy of the main tag store. When a main cache tag allocate is performed, the corresponding duplicate tag is also allocated. When an invalidate request is generated, the duplicate tag is immediately invalidated. This mechanism permits full bandwidth operation of the main cache without missing an invalidate request.

There is one invalidate scenario that the basic mechanism does not detect which would lead to the cache being incoherent. The following sequence of events can take place:

1 A load instruction generates a read miss in the cache.

2 The cache tag is allocated, and the read request is passed to the XMI read queue.

3 The XMI read queue controller issues an XMI request and allocates the duplicate tag.

4 A write to this location is performed by another node.

5 The duplicate tag lookup hits, and an invalidate request is generated.

6 The invalidate is immediately dispatched, and the main tag entry cleared.

7 The return read data appears, the cache is filled, and the tag is written with a good tag.

As the scenario shows, the memory returns good read data in the correct order, but it has been updated by the write transaction. Because of the relative order of the transactions, the cache shows that the read data is valid, when it should be invalidated.

To resolve this conflict when the duplicate tag is checked, the read queue checks the write against all outstanding read requests. If a match is found, the read queue sets a "Don't cache" flag in the read request buffer. When read data returns, it is not written in cache.

The scenario now takes place as follows:

1 A load instruction generates a read miss in the cache.

2 The cache tag is allocated, and the read request is passed to the XMI read queue.

3 The XMI read queue controller issues an XMI request and allocates the duplicate tag.

4 A write to this location is performed by another node.

5 The duplicate tag lookup hits, and an invalidate request is generated.

6 *The read queue lookup hits, and a "Don't cache" status flag is set.*

7 The invalidate is immediately dispatched, and the main tag entry cleared.

8 The return read data appears, *the cache is not filled, and the tag is left invalidated.*

The invalidate queue is 16 entries. In its quiescent state the Load/Store chip can process invalidates faster than the XMI can generate them. However, during execution of a load or store instruction, the invalidate queue can fill to a level where normal processing must cease, and the invalidate queue is then emptied before an overflow occurs. The number of entries before this mechanism is enabled is 9.

4.7.3 Memory Synchronization

Due to the scalar write buffer, the vector write buffer, and the pipeline in the Load/Store chip, the possibility exists that all memory transfers have not completed when the VECTL chip flags the scalar processor that an instruction is complete. Using data in memory before guaranteeing coherency can lead to erroneous results. Two mechanisms permit the scalar and vector processors to guarantee tha. all transactions have finished.

- Issuing the MSYNC instruction—used by user programs
- Reading the Vector Memory Activity Check Register—used by operating systems

4.7.3.1 Nonprivileged Memory Synchronization

MSYNC, which can be executed in user mode, causes the scalar processor to wait for the vector processor to finish all memory references and then flush its write buffer. The vector processor also guarantees to field all invalidates immediately until the next instruction is sent. By using the MSYNC instruction, the user can guarantee memory coherency. However, the MSYNC instruction can cause severe performance penalties if misused.

CAUTION: If a vector disable condition occurs, an MSYNC instruction causes a vector disable fault. This would be fatal if the MSYNC instruction is in operating system code that runs at a high IPL.

4.7.3.2 Privileged Memory Synchronization

The Vector Memory Activity Check Register (VMAC) is an IPR that can be read in kernel mode using the MFPR instruction. VMAC allows the processor to perform the MSYNC operation without incurring a disable fault. When the IPR read is issued, the scalar processor is stalled in the same way as above until all memory transactions have taken place.

NOTE: It is not possible to monitor the VPSR<BSY> bit to determine instruction completion. This bit is cleared as soon as the instruction completes, not when the memory transfer completes.

4.8 Vector Processor Registers

The vector module contains internal processor registers (IPRs) and vector indirect registers. Both are accessed by MTPR/MFPR instructions. The latter are read by using the indirect address registers, VIADR, VIDHI, and VIDLO. The vector data registers are also in vector indirect address space. The VLR, VCR, and VMR control registers are read and written by MFVP/MTVP instructions.

The vector procesor has 16 data registers, each containing 64 elements numbered 0 through 63. Each element is 64 bits wide. The *VAX Architecture Reference Manual* specifies the registers used to access the data registers. Other registers used with the data registers are the Vector Length, Vector Count, and Vector Mask Registers (see Figure 4–12 and Figure 4–13).

Figure 4–12 Vector Length and Vector Count Registers

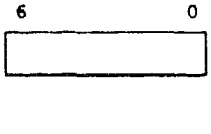
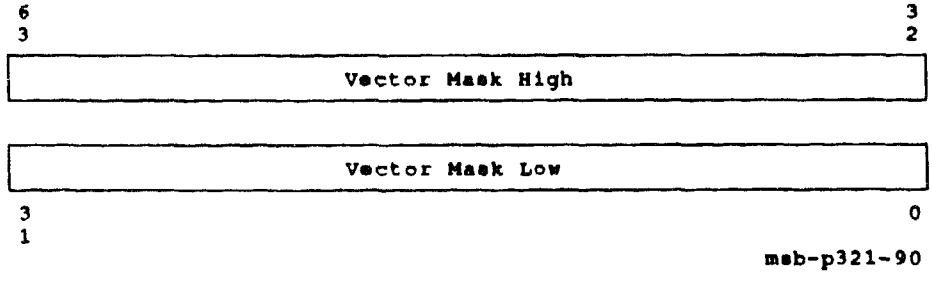


Figure 4–13 Vector Mask Register



- The 7-bit Vector Length Register (VLR) controls how many vector elements are processed. VLR is loaded prior to executing a vector instruction. Once loaded, VLR specifies the length of all subsequent vector instructions until it is loaded with a new value.
- The 7-bit Vector Count Register (VCR) receives the length of the offset vector generated by the IOTA instruction.
- The Vector Mask Register (VMR) has 64 bits, each bit corresponding to an element in a vector register. Bit <0> corresponds to vector element zero. The vector mask is used by the vector compare, merge, IOTA, masked load/store instructions, and masked arithmetic instructions.

VCR and VLR are in the VECTL chip. VMR and the vector data registers are split across the four Verse chips.

In addition to internal processor registers, the vector processor also has registers in vector indirect address space, which are called "vector indirect registers." Table 4–2 lists the internal processor registers (IPRs), and Table 4–3 lists the vector indirect registers.

4.8.1 Access to Registers

The IPRs and the vector indirect registers are accessed through the MTPR and MFPR instructions, which require kernel mode privileges.

The console operator uses the EXAMINE and DEPOSIT commands to read and write the IPRs and the vector indirect registers. The vector data registers can also be accessed from the console. The qualifiers differ:

- /I — to read and write the IPRs
- /M — to read and write the vector indirect registers, except for the 16 vector data registers
- /VE — to read and write the vector data registers

From the console, the Vector Length, Vector Count, and Vector Mask control registers can be specified as VLR, VCR, and VMR after DEPOSIT and EXAMINE commands with no qualifiers.

The following example shows how a vector indirect register can be read and written from the console (ALU_OP is the Arithmetic Instruction Register at the register field address of 440):

```
EXAMINE/M 440
DEPOSIT/M 440 xxxx
```

Software uses MTPR and MFPR instructions to access the vector indirect registers.

How to examine the contents of ALU_OP at address 440:

```
MTPR    #^X440, #VIADR
MFPR    #VIDLO, R0
MFPR    #VIDHI, R1
```

How to deposit data in ALU_OP at address 440:

```
MTPR    #^X440, #VIADR
MTPR    R1, #VIDLO
MTPR    R0, #VIDHI
```

The control registers, Vector Count, Vector Length, and Vector Mask, are accessed with MTVP/MFVP instructions.

4.8.2 Internal Processor Registers

Internal processor registers (IPRs) are accessed through the MTPR and MFPR instructions, which require kernel mode privilege. These instructions to the vector registers can be used even if the vector module is disabled.

Table 4–2 lists the vector module IPRs.

The IPRs in the vector module that are duplicates of registers in the scalar module will be written whenever the scalar IPRs are written, if bit <0>, Vector Present, is set in the scalar module ACCS register.

The Accelerator Control and Status (ACCS) and Vector Interface Error Status (VINTSR) Registers are implemented in the scalar CPU and are described in Chapter 3.

Table 4–1 lists the codes used to categorize the registers and bits in the following register descriptions.

Table 4–1 Types of Registers and Bits

Type	Description
0	Initialized to logic level zero
1	Initialized to logic level one
X	Initialized to either logic level
RO	Read only
R/W	Read/write
R/Cleared on W	Read/cleared on write
R/W1C	Read/cleared by writing a one
WO	Write only
BR	Broadcast read
BW	Broadcast write

Table 4–2 Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Class[2]	Location
8 (8)	Vector Copy—P0 Base	P0BR	WO	4	L/S
9 (9)	Vector Copy—P0 Length	P0LR	WO	4	L/S
10 (A)	Vector Copy—P1 Base	P1BR	WO	4	L/S
11 (B)	Vector Copy—P1 Length	P1LR	WO	4	L/S
12 (C)	Vector Copy—System Base	SBR	WO	4	L/S
13 (D)	Vector Copy—System Length	SLR	WO	4	L/S
40 (28)	Accelerator Control and Status	ACCS	R/W	2 Init	CPU-chip
56 (38)	Vector Copy—Memory Management Enable	LSX_MAPEN	WO	4	L/S
57 (39)	Vector Copy—Translation Buffer Invalidate All	LSX_TBIA	WO	4	L/S
58 (3A)	Vector Copy—Translation Buffer Invalidate Single	LSX_TBIS	WO	4	L/S
123 (7B)	Vector Interface Error Status	VINTSR	R/W	2	C-chip
144 (90)	Vector Processor Status	VPSR	R/W	3	VECTL
145 (91)	Vector Arithmetic Exception	VAER	RO	3	VECTL
146 (92)	Vector Memory Activity Check	VMAC	RO	3	VECTL
147 (93)	Vector Translation Buffer Invalidate All	VTBIA	WO	3	L/S
157 (9D)	Vector Indirect Register Address	VIADR	R/W	3	VECTL
158 (9E)	Vector Indirect Data Low	VIDLO	R/W	3	VECTL
159 (9F)	Vector Indirect Data High	VIDHI	R/W	3	VECTL

[1]See Table 4–1.

[2]Key to Classes:

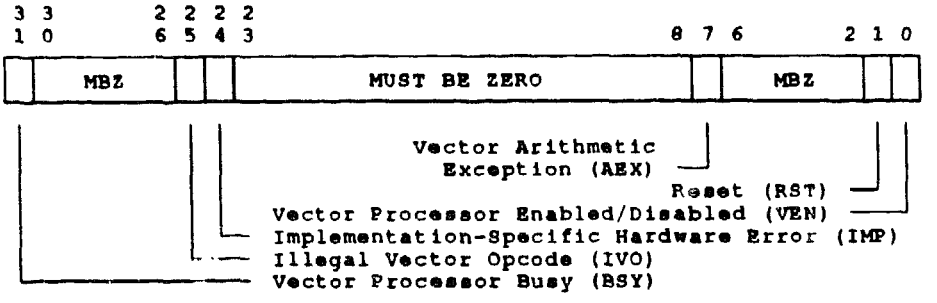
1 = Implemented by the KA64A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA64A CPU module.
3 = Implemented by the FV64A vector processor module.
4 = Implemented in the KA64A CPU module with a copy in the FV64A vector processor module.
n Init = The register is initialized on a KA64A CPU module reset (power-up, system reset, and node reset).

Vector Processor Status Register (VPSR)

The Vector Processor Status Register provides status to the scalar CPU about the state of the vector processor.

ADDRESS

IPR144 (VECTL chip)



msb-p122-90

bit<31>

Name: Vector Processor Busy

Mnemonic: BSY

Type: RO, 0

BSY, when set, indicates that the vector processor is processing vector instructions. When BSY is clear, the vector processor is idle.

bits<30:26>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bit<25>

Name: Illegal Vector Opcode

Mnemonic: IVO

Type: R/W1C, 0

IVO is set when the vector processor is disabled due to receiving one of the following: an illegal vector opcode, an illegal compare function field (3, 7, 8–F), or an illegal convert code (0, B, E) Writing a one to IVO clears IVO; writing a zero to IVO has no effect.

bit<24>

Name: Implementation-Specific Hardware Error
Mnemonic: IMP
Type: R/W1C, 0

IMP, when set, indicates that the vector processor is disabled due to a hardware error. Writing a one to IMP clears the bit and the Vector Controller Status Register (VCTL_CSR); writing a zero has no effect.

bits<23:8>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bit<7>

Name: Vector Arithmetic Exception
Mnemonic: AEX
Type: R/W1C, 0

AEX, when set, indicates that the vector processor encountered an arithmetic disabling condition. The nature of the exception can be determined by examining VAER. Writing a one to AEX clears it and the Vector Arithmetic Exception Register (VAER); writing a zero has no effect.

bits<6:2>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bit<1>

Name: Reset
Mnemonic: RST
Type: WO, 0

Writing a one to RST clears VPSR, VAER, and VCTL_CSR. If RST is set by software while BSY, bit <31>, is set, the operation of the vector processor is undefined. RST is always zero when read.

The following two sequences reset the vector processor:

MTPR #2, VPSR—Reset and leave processor disabled

MTPR #3, VPSR—Reset and leave processor enabled

bit<0>

Name: Vector Processor Enabled/Disabled

Mnemonic: VEN

Type: R/W, 0

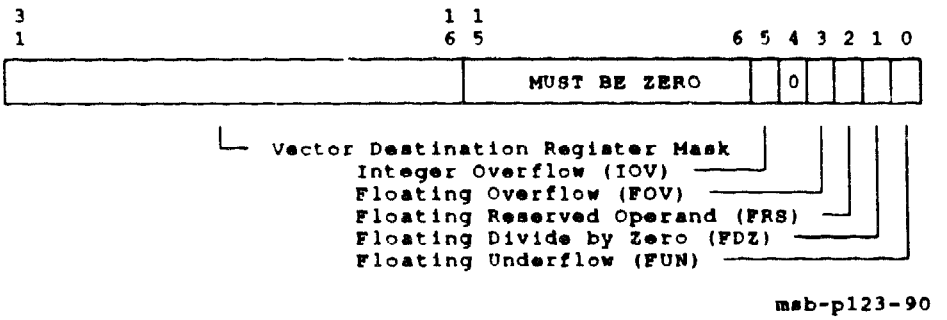
Enable the vector processor by writing a one to Vector Processor Enabled/Disabled. Disable the vector processor by writing a zero to VEN. VEN sets if the processor encounters a disabling fault. When this bit is zero, only MFPR/MTPR instructions can access the internal registers; the scalar CPU blocks MFVP/MTVP instructions and takes a vector disable fault (SCB vector 68 hex). If VEN is reset either by software or by a disabling fault, the vector processor finishes all outstanding instructions, if possible. The disabling condition must be removed before VEN can be set again.

Vector Arithmetic Exception Register (VAER)

Bits are set in VAER as instructions with arithmetic exceptions complete. The bit is set in the Vector Destination Register Mask field for the vector register for the faulting instruction. Writing a one to VPSR<7> (AEX) or VPSR<1> (RST) clears VAER. Writing a one to these bits also clears the Exception Summary Register.

NOTE: The operating system must wait for VPSR<31> (Vector Processor Busy) to be cleared before reading VAER. Spinning on busy ensures that all instructions have completed, and their associated exceptions have been reported. Setting VPSR<RST> before VPSR<BSY> clears produces UNPREDICTABLE results.

ADDRESS *IPR145 (VECTL chip)*



bits<31:16>

Name: Vector Destination Register Mask

Mnemonic: None

Type: RO, 0

The Vector Destination Register Mask field contains the value of the register in the VRC destination field of the currently executing arithmetic instruction, except for VVCMP instructions. The destination in this case is the mask register. If an exception occurs, no bit sets.

bits<15:6>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bit<5>

Name: Integer Overflow
Mnemonic: IOV
Type: RO, 0

IOV sets when an integer arithmetic operation or a conversion from F_, D_, or G_floating to integer overflows the destination precision.

bit<4>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bit<3>

Name: Floating Overflow
Mnemonic: FOV
Type: RO, 0

FOV sets when an F_, D_, or G_floating arithmetic or conversion operation overflows the destination exponent.

bit<2>

Name: Floating Reserved Operand
Mnemonic: FRS
Type: RO, 0

FRS sets when an attempt is made to perform an F_, D_, or G_floating arithmetic, conversion, or comparison operation and one or more of the operand values are reserved.

bit<1>

Name: Floating Divide by Zero
Mnemonic: FDZ
Type: RO, 0

FDZ sets when an attempt is made to perform an F_, D_, or G_floating divide operation with a divisor of zero.

bit<0>

Name: Floating Underflow
Mnemonic: FUN
Type: RO, 0

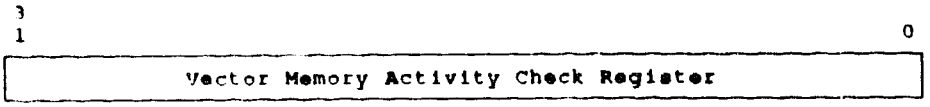
FUN sets when an F_, D_, or G_floating arithmetic or conversion operation underflows the destination exponent.

Vector Memory Activity Check Register (VMAC)

VMAC facilitates memory synchronization between the scalar and vector modules.

ADDRESS

IPR146 (VECTL chip)



mab-p124-90

bits<31:0>

Name: Vector Memory Activity Check

Mnemonic: VMAC

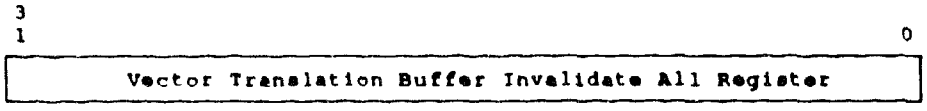
Type: RO, X

VMAC facilitates memory synchronization between the scalar and vector modules. VMAC operation is identical to that of MSYNC, except that the read bypasses the disable mechanism. VMS requires VMAC to perform memory synchronization in kernel mode.

Vector Translation Buffer Invalidate All Register (VTBIA)

Writing VTBIA causes the vector translation buffer to be flushed.

ADDRESS *IPR147 (L/S)*



mab-p125-90

bits<31:0>

Name: Vector Translation Buffer Invalidate All
Mnemonic: VTBIA
Type: WO, X

Writing VTBIA causes the vector translation buffer to be flushed.

Vector Indirect Register Address Register (VIADR)

VIADR provides access to the vector indirect registers and the vector data registers. To access the vector processor indirect address space, you use VIADR and the indirect data registers VIDHI and VIDLO. You load the Register Field Address (RFA) field with the appropriate address, and a read or write to the indirect data registers will then cause the data transfer to take place.

When a read from VIDLO is performed, a quadword is read from the vector indirect register being addressed by VIADR. The quadword is loaded into the VIDHI and VIDLO longword registers. The longword in VIDLO is passed back to the scalar CPU. If a longword is written to VIDHI followed by a longword written to VIDLO, the quadword in VIDHI and VIDLO is loaded into the indirect register corresponding to the address in VIADR.

For example, a vector indirect register such as the Arithmetic Instruction Register (ALU_OP) at the register field address of 440 is accessed with MTPR and MFPR instructions as follows:

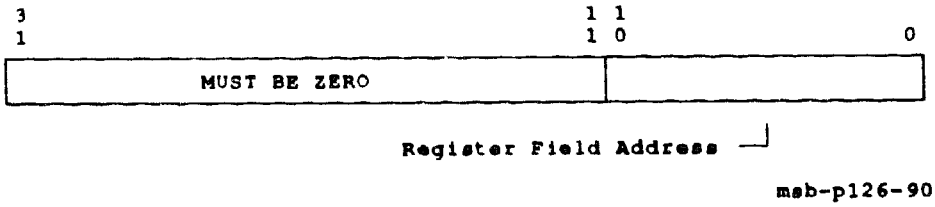
To examine the contents of ALU_OP at address 440:

```
MTPR    #^X440, #VIADR
MFPR    #VIDLO, R0
MFPR    #VIDHI, R1
```

To deposit data in ALU_OP at address 440:

```
MTPR    #^X440, #VIADR
MTPR    R1, #VIDLO
MTPR    R0, #VIDHI
```

ADDRESS *IPR157 (VECTL chip)*



bits<31:11>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<10:0>

Name: Register Field Address

Mnemonic: RFA

Type: R/W, X

If RFA <10> is zero, then RFA <9:0> indicates access to the vector data registers. The address is split into two fields:

- Vector register address (bits <9:6>)
- Vector element address (bits <5:0>)

If RFA <10> is one, then RFA <9:0> indicates access to the diagnostic and control registers in the vector indirect address space.

Codes for address bits <9:6> are as follows:

<9:6>	Location
0001	Verse chip registers
0010	VECTL chip registers
0100	Load/Store and XMI interface registers
1000	Reserved

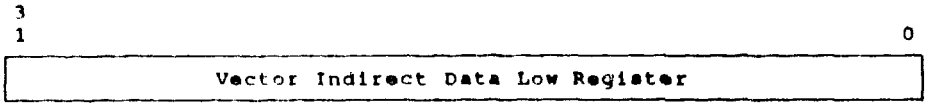
All other addresses are illegal and are ignored by the VECTL chip. A read to an illegal location produces UNPREDICTABLE results; a write is ignored.

Vector Indirect Data Low Register (VIDLO)

VIDLO contains the low-order longword of an internal processor register quadword.

ADDRESS

IPR158 (VECTL chip)



mab-p127-90

bits<31:0>

Name: Vector Indirect Data Low

Mnemonic: VIDLO

Type: R/W, X

A read of VIDLO causes the vector processor to access the vector register element or vector indirect register specified in VIADR. Bits <31:0> of the selected register are loaded into VIDLO and returned to the scalar CPU, where it is stored in dst.wl.

A write of VIDLO causes the vector processor to write the vector register element or vector indirect register specified in VIADR. The low 32 bits are supplied from src.rl in the scalar CPU.

Vector Indirect Data High Register (VIDHI)

VIDHI contains the high-order longword of an internal processor register quadword.

ADDRESS

IPR159 (VECTL chip)

msb-p128-90

bits<31:0>

Name: Vector Indirect Data High

Mnemonic: VIDHI

Type: R/W, 0

VIDHI buffers the high 32 bits of quadword arguments when accessing VIDLO.

4.8.3 Vector Indirect Registers

The vector module has 16 Kbytes of indirect address space. Table 4–3 lists the registers implemented in that space.

Table 4–3 Vector Indirect Registers

Register Field Address (hex)	Register	Mnemonic	Type	Location
000–03F	Vector Register 0	VREG0	R/W	Verse
040–07F	Vector Register 1	VREG1	R/W	Verse
080–0BF	Vector Register 2	VREG2	R/W	Verse
0C0–0FF	Vector Register 3	VREG3	R/W	Verse
100–13F	Vector Register 4	VREG4	R/W	Verse
140–17F	Vector Register 5	VREG5	R/W	Verse
180–1BF	Vector Register 6	VREG6	R/W	Verse
1C0–1FF	Vector Register 7	VREG7	R/W	Verse
200–23F	Vector Register 8	VREG8	R/W	Verse
240–27F	Vector Register 9	VREG9	R/W	Verse
280–2BF	Vector Register 10	VREG10	R/W	Verse
2C0–2FF	Vector Register 11	VREG11	R/W	Verse
300–33F	Vector Register 12	VREG12	R/W	Verse
340–37F	Vector Register 13	VREG13	R/W	Verse
380–3BF	Vector Register 14	VREG14	R/W	Verse
3C0–3FF	Vector Register 15	VREG15	R/W	Verse
440*	Arithmetic Instruction	ALU_OP	R/BW	Verse
448	Scalar Operand Low	ALU_SCOP_LO	R/BW	Verse
44C	Scalar Operand High	ALU_SCOP_HI	R/BW	Verse
450	Vector Mask Low	ALU_MASK_LO	BR/BW	Verse
451	Vector Mask High	ALU_MASK_HI	BR/BW	Verse
454	Exception Summary	ALU_EXC	R/BW	Verse
45C	Diagnostic Control	ALU_DIAG_CTL	R/BW	Verse
480	Current ALU Instruction	VCTL_CALU	R/W	VECTL
481	Deferred ALU Instruction	VCTL_DALU	R/W	VECTL
482	Current ALU Operand Low	VCTL_COP_LO	R/W	VECTL
483	Current ALU Operand High	VCTL_COP_HI	R/W	VECTL
484	Deferred ALU Operand Low	VCTL_DOP_LO	R/W	VECTL

*Addresses from 400–45F in this column specify the address of Verse chip 0; addresses for Verse chips 1, 2, and 3 are found by adding 1, 2, and 3 to the address given. A read must specify each Verse chip by its own address; a write to the address given in the table (for Verse chip 0) is broadcast to all Verse chips.

Table 4–3 (Cont.) Vector Indirect Registers

Register Field Address (hex)	Register	Mnemonic	Type	Location
485	Deferred ALU Operand High	VCTL_DOP_HI	R/W	VECTL
486	Load/Store Instruction	VCTL_LDST	R/W	VECTL
487	Load/Store Stride	VCTL_STRIDE	R/W	VECTL
488	Illegal Instruction	VCTL_ILL	R/W	VECTL
489	Vector Controller Status	VCTL_CSR	R/W	VECTL
48A	Module Revision	MOD_REV	RO	VECTL
500	Vector Copy—P0 Base	LSX_P0BR	WO	L/S
501	Vector Copy—P0 Length	LSX_P0LR	WO	L/S
502	Vector Copy —P1 Base	LSX_P1BR	WO	L/S
503	Vector Copy —P1 Length	LSX_P1LR	WO	L/S
504	Vector Copy —System Base	LSX_SBR	WO	L/S
505	Vector Copy—System Length	LSX_SLR	R/W	L/S
508	Load/Store Exception	LSX_EXC	RO	L/S
509	Translation Buffer Control	LSX_TBCSR	WO	L/S
50A	Vector Copy—Memory Management Enable	LSX_MAPEN	WO	L/S
50B	Vector Copy—Translation Buffer Invalidate All	LSX_TBIA	WO	L/S
50C	Vector Copy—Translation Buffer Invalidate Single	LSX_TBIS	WO	L/S
510	Vector Mask Low	LSX_MASKLO	WO	L/S
511	Vector Mask High	LSX_MASKHI	WO	L/S
512	Load/Store Stride	LSX_STRIDE	WO	L/S
513	Load/Store Instruction	LSX_INST	WO	L/S
520	Cache Control	LSX_CCSR	R/W	L/S
530	Translation Buffer Tag	LSX_TBTAG	R/W	L/S
531	Translation Buffer PTE	LSX_PTE	R/W	L/S

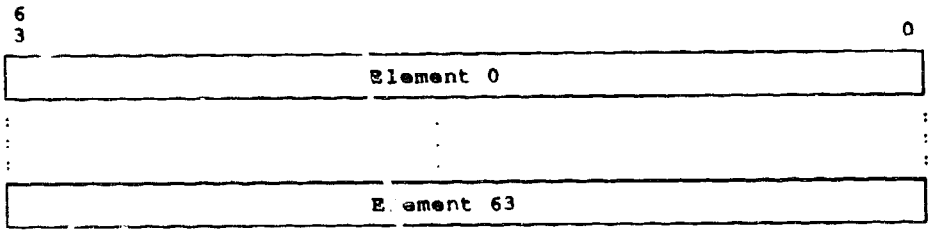
Vector Register *n* (VREG*n*)

There are 16 vector data registers, each of which contains 64 elements, numbered 0 through 63. Each element is 64 bits wide.

The 16 vector data registers are accessed through the indirect address registers. For diagnostics and self-test, the VREGs are only accessible from kernel mode using Move to Processor Register/Move From Processor Register (MTPR/MFPR) instructions using the indirect address registers.

ADDRESS

VREG0 *000–03F (Verse chips)*
VREG1 *040–07F*
VREG2 *080–0BF*
VREG3 *0C0–0FF*
VREG4 *100–13F*
VREG5 *140–17F*
VREG6 *180–1BF*
VREG7 *1C0–17F*
VREG8 *200–23F*
VREG9 *240–27F*
VREG10 *280–2BF*
VREG11 *2C0–2FF*
VREG12 *300–33F*
VREG13 *340–37F*
VREG14 *380–3BF*
VREG15 *3C0–3FF*



msb-p138-90

bits<63:0>

Name: Vector Register *n*

Mnemonic: VREG*n*

Type: R/W

Vector logical instructions read source element bits <31:0> and write the result into destination element bits <31:0> while destination element bits <63:32> receive bits <63:32> of the corresponding element of the Vb source operand.

Destination element bits <63:32> are UNPREDICTABLE for vector instructions that read longword data from memory into a VREG. If the same VREG is used as both source and destination in a Gather Memory Data into Vector Register (VGATH) instruction, the result of the VGATH is UNPREDICTABLE. Destination element bits <63:32> are UNPREDICTABLE for the Generate Compressed Iota Vector (IOTA) instruction.

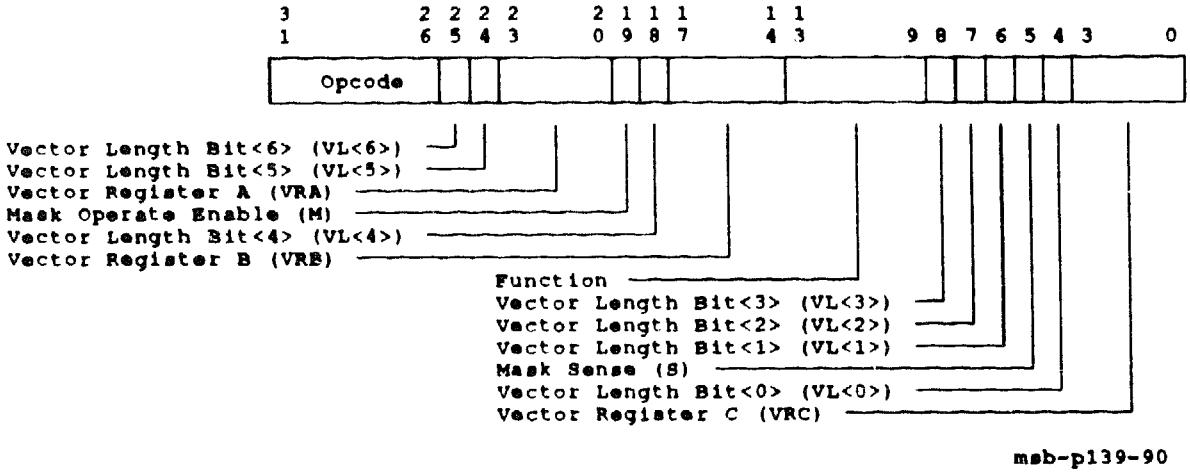
Arithmetic Instruction Register (ALU_OP)

ALU_OP contains the vector operate instruction and is a First In First Out (FIFO) register. Data is put into the FIFO by writes and is removed when an instruction completes, letting the vector controller load in a new instruction while the previous instruction is still operating. If the FIFO is empty, a new instruction starts executing as soon as something is written to it. If an instruction is already in ALU_OP, a new instruction starts reading its operand as soon as the old instruction is finished.

VAX opcodes have been translated to an internal format.

ALU_OP is accessed using the indirect address registers.

ADDRESS *440 (Verse chip)*



msb-p139-90

bits<31:26>

Name: Opcode

Mnemonic: None

Type: R/W, X

Opcode contains the vector operate instruction opcode that is to be performed.

bit<25>

Name: Vector Length Bit<6>

Mnemonic: VL<6>

Type: R/W, X

Vector Length Bit<6> contains the embedded vector length bit<6>.

bit<24>

Name: Vector Length Bit<5>
Mnemonic: VL<5>
Type: R/W, X

Vector Length Bit<5> contains the embedded vector length bit<5>.

bits<23:20>

Name: Vector Register A
Mnemonic: VRA
Type: R/W, X

VRA selects the source operand for a vector operate instruction. VRA is only valid when the vector operation requires two vector register sources. VRA is ignored for instructions that require a scalar source operand.

bit<19>

Name: Mask Operate Enable
Mnemonic: M
Type: R/W, X

M, when set, enables operations controlled by the mask register. Elements must have the corresponding Vector Mask Register bit match Mask Sense (ALU_OP<5>) to be operated on.

bit<18>

Name: Vector Length Bit<4>
Mnemonic: VL<4>
Type: R/W, X

Vector Length Bit<4> contains the embedded vector length bit<4>.

bits<17:14>

Name: Vector Register B
Mnemonic: VRB
Type: R/W, X

VRB selects one of the source registers for vector operate instructions.

bits<13:9>

Name: Function
Mnemonic: None
Type: R/W, X

Function contains the function code for the vector operate instruction.

bit<8>

Name: Vector Length Bit<3>
Mnemonic: VL<3>
Type: R/W, X

Vector Length Bit<3> contains the embedded vector length bit<3>.

bit<7>

Name: Vector Length Bit<2>
Mnemonic: VL<2>
Type: R/W, X

Vector Length Bit<2> contains the embedded vector length bit<2>.

bit<6>

Name: Vector Length Bit<1>
Mnemonic: VL<1>
Type: R/W, X

Vector Length Bit<1> contains the embedded vector length bit<1>.

bit<5>

Name: Mask Sense
Mnemonic: S
Type: R/W, X

Elements must have the corresponding Vector Mask Register bit match Mask Sense to be operated on, when mask operations are enabled by Mask Operate Enable (ALU_OP<19>).

bit<4>

Name: Vector Length Bit<0>
Mnemonic: VL<0>
Type: R/W, X

Vector Length Bit<0> contains the embedded vector length bit<0>.

bits<3:0>

Name: Vector Register C

Mnemonic: VRC

Type: R/W, X

VRC selects the destination register for the results of the vector operate instruction.

Scalar Operand Low Register (ALU_SCOP_LO)

ALU_SCOP_LO is used to store the scalar operand.

ALU_SCOP_LO is accessed using the indirect address registers.

ADDRESS

448 (Verse chip)

mab-p140-90

bits<31:0>

Name: Scalar Operand Low

Mnemonic: ALU_SCOP_LO

Type: R/W, X

The scalar operand is stored when Scalar Operand Low is written. ALU_SCOP_LO is a First In First Out (FIFO) register. Data is put into the FIFO by writes and is removed when an instruction completes, allowing the vector controller to load a new instruction and scalar operand while the previous instruction is still executing. If the FIFO is empty, a new instruction starts executing as soon as something is written to it. If an instruction is already in ALU_SCOP_LO, a new instruction starts reading its operands as soon as the old instruction is finished.

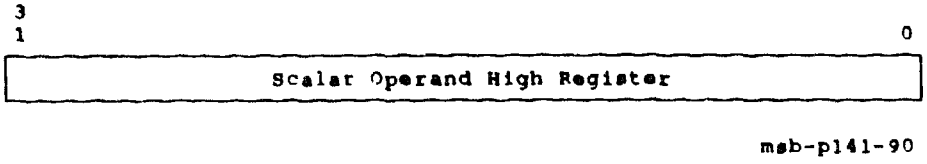
Scalar Operand High Register (ALU_SCOP_HI)

ALU_SCOP_HI is used to store the scalar operand.

ALU_SCOP_HI is accessed using the indirect address registers.

ADDRESS

44C (Verse chip)



bits<31:0>

Name: Scalar Operand High

Mnemonic: ALU_SCOP_HI

Type: R/W, X

The scalar operand is stored when Scalar Operand High is written. ALU_SCOP_HI is a First In First Out (FIFO) register. Data is put into the FIFO by writes and is removed when an instruction completes, allowing the vector controller to load a new instruction and scalar operand while the previous instruction is still executing. If the FIFO is empty, a new instruction starts executing as soon as something is written to it. If an instruction is already in ALU_SCOP_HI, a new instruction starts reading its operands as soon as the old instruction is finished.

Vector Mask Low Register (ALU_MASK_LO)

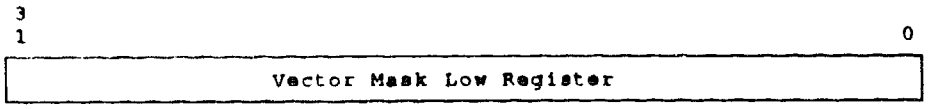
ALU_MASK_LO, when read, causes the low 32 bits of the vector mask to be read. When written, the appropriate mask bits are loaded.

ALU_MASK_LO is read by the Move From Vector Processor (MFVP) vector instruction. It is modified either by a vector compare instruction or by writing the register with a Move To Vector Processor (MTVP) vector instruction.

ALU_MASK_LO can also be accessed using the indirect address registers.

ADDRESS

450 (Verse chip)



mab-p142-90

bits<31:0>

Name: Vector Mask Low

Mnemonic: ALU_MASK_LO

Type: Special Read/Broadcast Write, X

Vector Mask Low, when read, causes the low 32 bits of the vector mask to be read. When written, the appropriate mask bits are loaded.

Vector Mask High Register (ALU_MASK_HI)

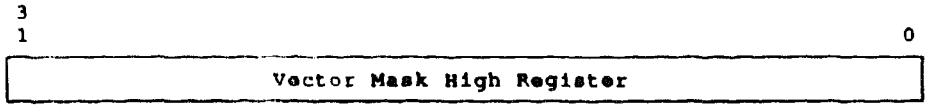
ALU_MASK_HI, when read, causes the high 32 bits of the vector mask to be read. When written, the appropriate mask bits are loaded.

ALU_MASK_HI is read by the Move From Vector Processor (MFVP) vector instruction. It is modified either by a vector compare instruction or by writing the register with a Move To Vector Processor (MTVP) vector instruction.

ALU_MASK_HI can also be accessed using the indirect address registers.

ADDRESS

451 (Verse chip)



msb-p143-90

bits<31:0>

Name: Vector Mask High

Mnemonic: ALU_MASK_HI

Type: Special Read/Broadcast Write, X

Vector Mask High, when read, causes the high 32 bits of the vector mask to be read. When written, the appropriate mask bits are loaded.

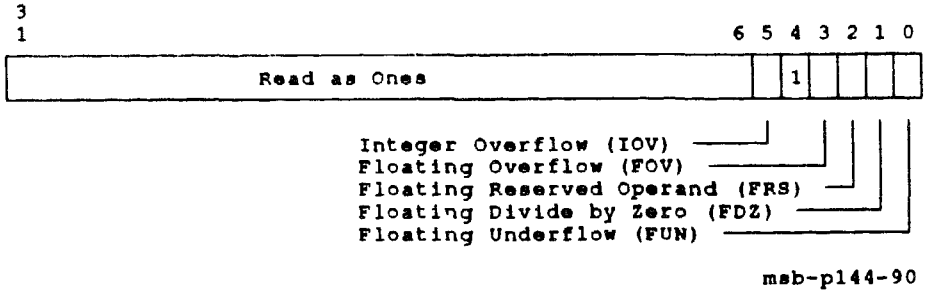
Exception Summary Register (ALU_EXC)

Bits are set in the Exception Summary Register as vector instructions with arithmetic exceptions complete. Any write to this register clears the entire register. Also, writing a one to VPSR<1> (Reset) or VPSR<7> (Vector Arithmetic Exception) clears this register.

ALU_EXC is accessed using the indirect address registers.

ADDRESS

454 (Verse chip)



bits<31:6>

Name: Reserved

Mnemonic: None

Type: RO, 1

Unused; must read as ones.

bit<5>

Name: Integer Overflow

Mnemonic: IOV

Type: R/W, 0

IOV sets when an integer arithmetic operation or a conversion f om F_, D_, or G_floating to integer overflows the destination precision.

bit<4>

Name: Reserved

Mnemonic: None

Type: RO, 1

Unused; must read as one.

bit<3>

Name: Floating Overflow
Mnemonic: FOV
Type: R/W, 0

FOV sets when an F_, D_, or G_floating arithmetic operation or conversion overflows the destination exponent.

bit<2>

Name: Floating Reserved Operand
Mnemonic: FRS
Type: R/W, 0

FRS sets when an attempt is made to perform F_, D_, or G_floating arithmetic, conversion, or comparison operations and one or more of the operand values are reserved.

bit<1>

Name: Floating Divide by Zero
Mnemonic: FDZ
Type: R/W, 0

FDZ sets when an attempt is made to perform an F_, D_, or G_floating divide operation with a divisor of zero.

bit<0>

Name: Floating Underflow
Mnemonic: FUN
Type: R/W, 0

FUN sets when an F_, D_, or G_floating arithmetic operation or conversion underflows the destination exponent.

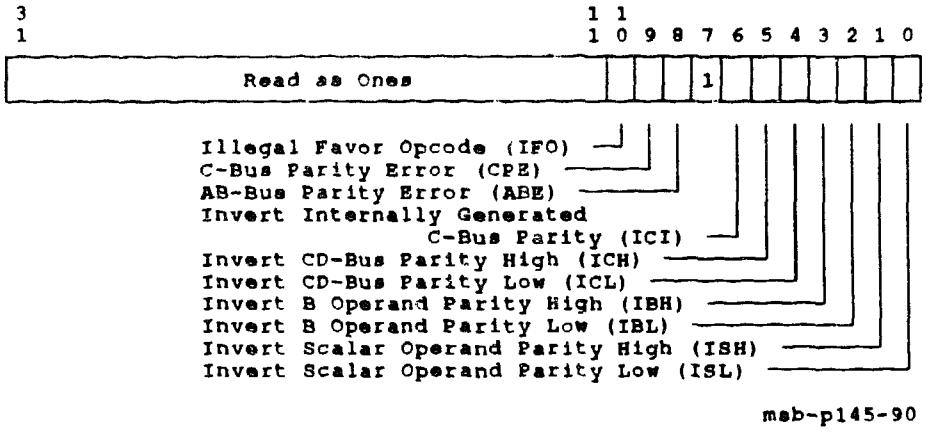
Diagnostic Control Register (ALU_DIAG_CTL)

ALU_DIAG_CTL is used for diagnostic testing and reporting errors.

ALU_DIAG_CTL is accessed using the indirect address registers.

ADDRESS

45C (Verse chip)



bits<31:11>

Name: Reserved

Mnemonic: None

Type: RO, 1

Unused; read as ones.

bit<10>

Name: Illegal Favor Opcode

Mnemonic: IFO

Type: R/W1C, 0

IFO sets if the transmitted Favor opcode is illegal.

bit<9>

Name: C-Bus Parity Error

Mnemonic: CPE

Type: R/W1C, 0

CPE sets if a parity error is detected during a C bus transfer.

bit<8>

Name: AB-Bus Parity Error
Mnemonic: ABE
Type: R/W1C, 0

ABE sets if a parity error is detected during an AB bus transfer.

bit<7>

Name: Reserved
Mnemonic: None
Type: RO, 1

Unused; read as one.

bit<6>

Name: Invert Internally Generated C-Bus Parity
Mnemonic: ICI
Type: R/W, 0

ICI, when set, inverts the internally generated parity on the C bus before writing it, causing bad register file parity.

bit<5>

Name: Invert CD-Bus Parity High
Mnemonic: ICH
Type: R/W, 0

ICH, when set, inverts the high longword parity bit of the CD bus, causing bad parity.

bit<4>

Name: Invert CD-Bus Parity Low
Mnemonic: ICL
Type: R/W, 0

ICL, when set, inverts the low longword parity bit of the CD bus, causing bad parity.

bit<3>

Name: Invert B Operand Parity High
Mnemonic: IBH
Type: R/W, 0

IBH, when set, inverts the high longword parity bit of the B operand from the register file before sending it on the ABPP, causing bad parity.

bit<2>

Name: Invert B Operand Parity Low
Mnemonic: IBL
Type: R/W, 0

IBL, when set, inverts the low longword parity bit of the B operand from the register file before sending it on the ABPP, causing bad parity.

bit<1>

Name: Invert Scalar Operand Parity High
Mnemonic: ISH
Type: R/W, 0

ISH, when set, inverts the high longword parity bit from the scalar register before sending it on the ABPP, causing bad parity.

bit<0>

Name: Invert Scalar Operand Parity Low
Mnemonic: ISL
Type: R/W, 0

ISL, when set, inverts the low longword parity bit from the scalar register before sending it on the ABPP, causing bad parity.

Current ALU Instruction Register (VCTL_CALU)

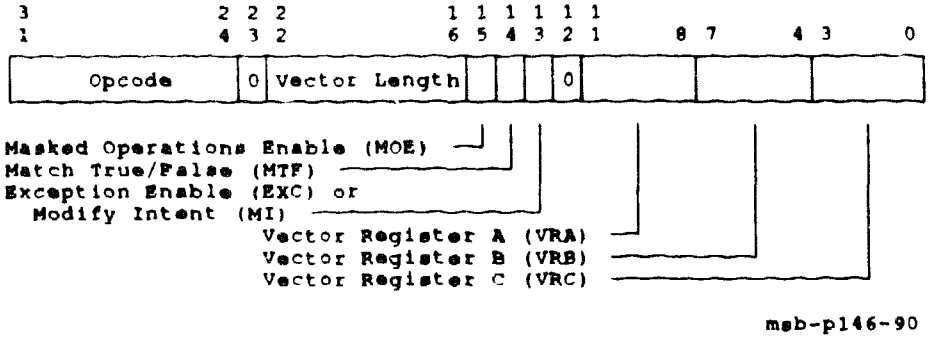
VCTL_CALU shows the arithmetic instruction being executed in the vector arithmetic pipeline. When the instruction completes, VCTL_CALU is loaded with the contents of the Deferred ALU Instruction Register.

A write to this register while an instruction is executing changes the contents of the register and causes UNPREDICTABLE results.

VCTL_CALU is accessed using the indirect address registers.

ADDRESS

480 (VECTL chip)



msb-p146-90

bits<31:24>

Name: Opcode

Mnemonic: None

Type: R/W, X

The first byte of the opcode is stripped by the scalar CPU during issue.

bit<23>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<22:16>

Name: Vector Length
Mnemonic: None
Type: R/W, X

Vector Length specifies the highest element operated upon.

bit<15>

Name: Masked Operations Enable
Mnemonic: MOE
Type: R/W, X

MOE, when set, enables masked operations. Only those elements that correspond to the Vector Mask Register bit that matches Match True/False (VCTL_CALU<14>) are enabled. If MOE is clear, all elements are operated upon.

MOE must be zero for VMERGE and IOTA instructions; otherwise, the results are UNPREDICTABLE.

bit<14>

Name: Match True/False
Mnemonic: MTF
Type: R/W, X

MTF controls the elements that are operated upon when masked operations are enabled. Only those elements that correspond to the Vector Mask Register bit that matches MTF are enabled.

bit<13>

Name: Exception Enable
Mnemonic: EXC
Type: R/W, X

EXC, when set, enables floating underflow for vector floating-point instructions and enables integer overflow for vector integer instructions. When EXC is clear, floating underflow and integer overflow are disabled.

For VLD/VGATH instructions, this bit is used and labeled differently.

alternate bit<13>

Name: Modify Intent
Mnemonic: MI
Type: R/W, X

For VLD/VGATH instructions, bit<13> has another meaning.

MI, when set, indicates to the vector processor that 75 percent or more of memory locations being loaded by a VLD/VGATH instruction will be modified later by VST/VSCAT instructions. This allows the vector processor to optimize the vector loads and stores performed on these locations.

bit<12>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bits<11:8>

Name: Vector Register A
Mnemonic: VRA
Type: R/W, X

VRA is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<7:4>

Name: Vector Register B
Mnemonic: VRB
Type: R/W, X

VRB is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<3:0>

Name: Vector Register C
Mnemonic: VRC
Type: R/W, X

VRC is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

Deferred ALU Instruction Register (VCTL_DALU)

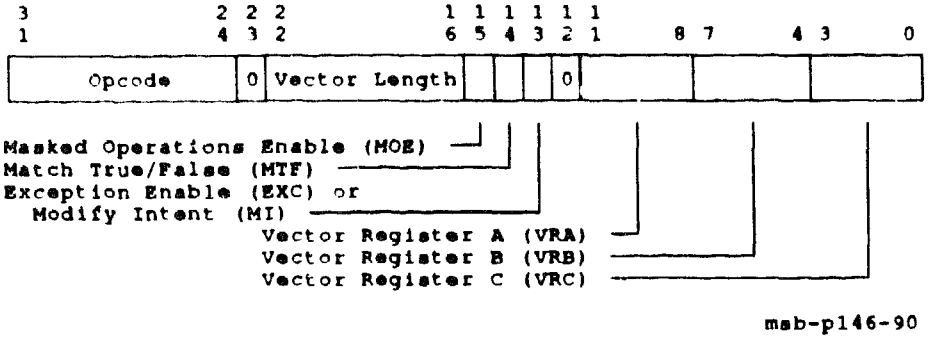
VCTL_DALU shows the arithmetic instruction to be executed next in the vector arithmetic pipeline.

A write to this register while an instruction is executing changes the contents of the register and causes UNPREDICTABLE results.

VCTL_DALU is accessed using the indirect address registers.

ADDRESS

481 (VECTL chip)



mab-p146-90

bits<31:24>

Name: Opcode

Mnemonic: None

Type: R/W, X

The first byte of the opcode is stripped by the scalar CPU during issue.

bit<23>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<22:16>

Name: Vector Length
Mnemonic: None
Type: R/W, X

Vector Length limits the highest element operated upon.

bit<15>

Name: Masked Operations Enable
Mnemonic: MOE
Type: R/W, X

MOE, when set, enables masked operations. Only those elements that correspond to the Vector Mask Register bit that matches Match True/False (VCTL_CALU<14>) are enabled. If MOE is clear, all elements are operated upon.

MOE must be zero for VMERGE and IOTA instructions; otherwise, the results are UNPREDICTABLE.

bit<14>

Name: Match True/False
Mnemonic: MTF
Type: R/W, X

MTF controls the elements that are operated upon when masked operations are enabled. Only those elements that correspond to the Vector Mask Register bit that matches MTF are enabled.

bit<13>

Name: Exception Enable
Mnemonic: EXC
Type: R/W, X

EXC, when set, enables floating underflow for vector floating-point instructions and enables integer overflow for vector integer instructions. When EXC is clear, floating underflow and integer overflow are disabled.

For VLD/VGATH instructions, this bit is used and labeled differently.

alternate bit<13>

Name: Modify Intent
Mnemonic: MI
Type: R/W, X

For VLD/VGATH instructions, bit <13> has another meaning.

MI, when set, indicates to the vector processor that 75 percent or more of memory locations being loaded by a VLD/VGATH instruction will be modified later by VST/VSCAT instructions. This allows the vector processor to optimize the vector loads and stores performed on these locations.

bit<12>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero

bits<11:8>

Name: Vector Register A
Mnemonic: VRA
Type: R/W, X

VRA is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<7:4>

Name: Vector Register B
Mnemonic: VRB
Type: R/W, X

VRB is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<3:0>

Name: Vector Register C
Mnemonic: VRC
Type: R/W, X

VRC is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

Current ALU Operand Low Register (VCTL_COP_LO)

Reading VCTL_COP_LO causes the contents of this register to be transferred to the indirect data registers.

VCTL_COP_LO is accessed using the indirect address registers.

ADDRESS

482 (VECTL chip)

msb-p147-90

bits<31:0>

Name: Current ALU Operand Low

Mnemonic: VCTL_COP_LO

Type: R/W, X

A read to VCTL_COP_LO causes the contents of this register to be transferred to the indirect data registers.

A write to VCTL_COP_LO while an arithmetic instruction is executing produces UNPREDICTABLE results.

Current ALU Operand High Register (VCTL_COP_HI)

Reading VCTL_COP_HI causes the contents of this register to be transferred to the indirect data registers.

VCTL_COP_HI is accessed using the indirect address registers.

ADDRESS

483 (VECTL chip)

mab-p148-90

bits<31:0>

Name: Current ALU Operand High

Mnemonic: VCTL_COP_HI

Type: R/W, X

A read to VCTL_COP_HI causes the contents of this register to be transferred to the indirect data registers.

A write to VCTL_COP_HI while an arithmetic instruction is executing produces UNPREDICTABLE results.

Deferred ALU Operand Low Register (VCTL_DOP_LO)

A read to VCTL_DOP_LO causes the contents of this register to be transferred to the indirect data registers.

VCTL_DOP_LO is accessed using the indirect address registers.

ADDRESS

484 (VECTL chip)

msb-p147-90

bits<31:0>

Name: Deferred ALU Operand Low

Mnemonic: VCTL_DOP_LO

Type: R/W, X

A read to VCTL_DOP_LO causes the contents of this register to be transferred to the indirect data registers.

A write to VCTL_DOP_LO while an arithmetic instruction is executing produces UNPREDICTABLE results.

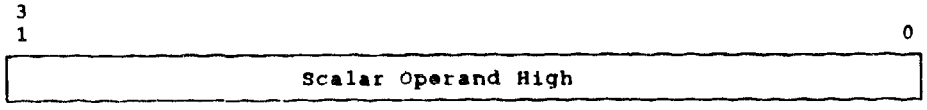
Deferred ALU Operand High Register (VCTL_DOP_HI)

A read to VCTL_DOP_HI causes the contents of this register to be transferred to the indirect data registers.

VCTL_DOP_HI is accessed using the indirect address registers.

ADDRESS

485 (VECTL chip)



mab-p148-90

bits<31:0>

Name: Deferred ALU Operand High

Mnemonic: VCTL_DOP_HI

Type: R/W, U

A read to VCTL_DOP_HI causes the contents of this register to be transferred to the indirect data registers.

A write to VCTL_DOP_HI while an arithmetic instruction is executing produces UNPREDICTABLE results.

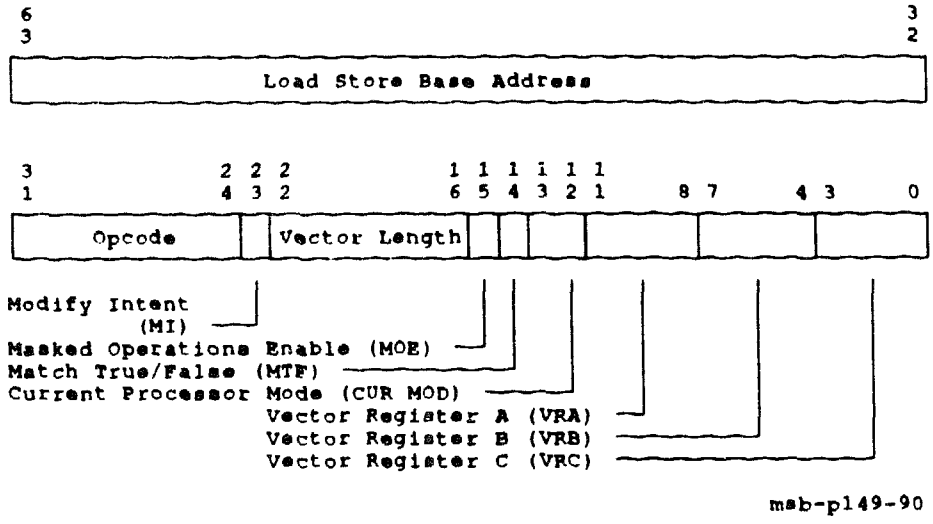
Load/Store Instruction Register (VCTL_LDST)

VCTL_LDST shows the load/store instruction being executed.

A write to this register changes the contents of the register and causes UNPREDICTABLE results.

VCTL_LDST is accessed using the indirect address registers.

ADDRESS *486 (VECTL chip)*



bits<63:32>

Name: Load/Store Base Address

Mnemonic: None

Type: RO, 0

Load/Store Base Address is the base address for this instruction.

bits<31:24>

Name: Opcode
Mnemonic: None
Type: R/W, X

The first byte of the opcode is stripped by the scalar CPU during issue.

bit<23>

Name: Modify Intent
Mnemonic: MI
Type: R/W, U

MI applies only to VLD/VGATH instructions. MI, when set, indicates to the vector processor that 75 percent or more of memory locations being loaded by a VLD/VGATH instruction will be modified later by VST/VSCAT instructions. This allows the vector processor to optimize the vector loads and stores performed on these locations.

bits<22:16>

Name: Vector Length
Mnemonic: None
Type: R/W, X

Vector Length specifies the highest element operated upon.

bit<15>

Name: Masked Operations Enable
Mnemonic: MOE
Type: R/W, X

When MOE is set, masked operations are enabled. Only those elements that correspond to the Vector Mask Register bit that matches Match True/False (VCTL_CALU<i4>) are enabled. If MOE is clear, all elements are operated upon.

MOE must be zero for VMERGE and IOTA instructions; otherwise, the results are UNPREDICTABLE.

bit<14>

Name: Match True/False
Mnemonic: MTF
Type: R/W, X

MTF controls the elements that are operated upon when masked operations are enabled. Only those elements that correspond to the Vector Mask Register bit that matches MTF are enabled.

bit<13:12>

Name: Current Processor Mode
Mnemonic: CUR MOD
Type: R/W, X

CUR MOD is used for all memory reference instructions.

bits<11:8>

Name: Vector Register A
Mnemonic: VRA
Type: R/W, X

VRA is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<7:4>

Name: Vector Register B
Mnemonic: VRB
Type: R/W, X

VRB is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<3:0>

Name: Vector Register C
Mnemonic: VRC
Type: R/W, X

VRC is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

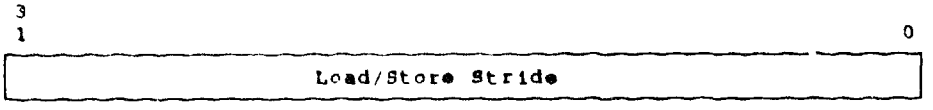
Load/Store Stride Register (VCTL_STRIDE)

VCTL_STRIDE contains the value of the stride that was sent as part of the last load/store instruction.

VCTL_STRIDE is accessed using the indirect address registers.

ADDRESS

487 (VECTL chip)



msb-p150-90

bits<31:0>

Name: Load/Store Stride

Mnemonic: VCTL_STRIDE

Type: R/W, X

VCTL_STRIDE contains the value of the stride that was sent as part of the last load/store instruction.

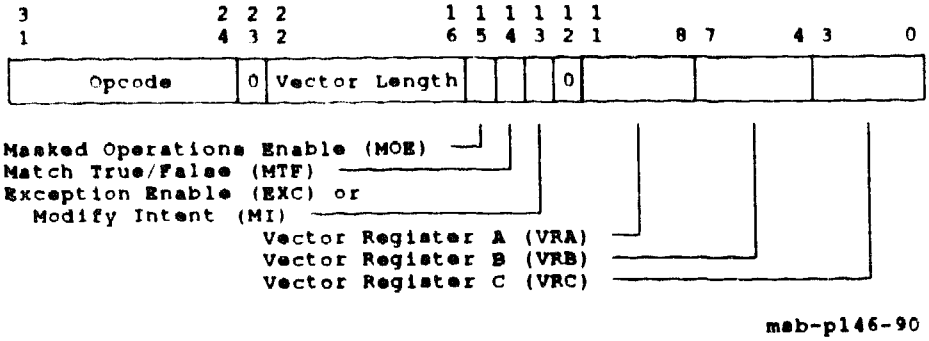
Illegal Instruction Register (VCTL_ILL)

VCTL_ILL stores any illegal instructions that are sent to the vector processor.

VCTL_ILL is accessed using the indirect address registers.

ADDRESS

488 (VECTL chip)



bits<31:24>

Name: Opcode

Mnemonic: None

Type: R/W, X

The first byte of the opcode is stripped by the scalar CPU during issue.

bit<23>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<22:16>

Name: Vector Length
Mnemonic: None
Type: R/W, X

Vector Length specifies the highest element operated upon.

bit<15>

Name: Masked Operations Enable
Mnemonic: MOE
Type: R/W, X

MOE, when set, enables masked operations. Only those elements that correspond to the Vector Mask Register bit that matches Match True/False (VCTL_CALU<14>) are enabled. If MOE is clear, all elements are operated upon.

MOE must be zero for VMERGE and IOTA instructions; otherwise, the results are UNPREDICTABLE.

bit<14>

Name: Match True/False
Mnemonic: MTF
Type: R/W, X

MTF controls the elements that are operated upon when masked operations are enabled. Only those elements that correspond to the Vector Mask Register bit that matches MTF are enabled.

bit<13>

Name: Exception Enable
Mnemonic: EXC
Type: R/W, X

EXC, when set, enables floating underflow for vector floating-point instructions and enables integer overflow for vector integer instructions. When EXC is clear, floating underflow and integer overflow are disabled.

For VLD/VGATH instructions, this bit is used and labeled differently.

alternate bit<13>

Name: Modify Intent
Mnemonic: MI
Type: R/W, X

For VLD/VGATH instructions, bit <13> has another meaning.

MI, when set, indicates to the vector processor that 75 percent or more of memory locations being loaded by a VLD/VGATH instruction will be modified later by VST/VSCAT instructions. This allows the vector processor to optimize the vector loads and stores performed on these locations.

bit<12>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bits<11:8>

Name: Vector Register A
Mnemonic: VRA
Type: R/W, X

VRA is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<7:4>

Name: Vector Register B
Mnemonic: VRB
Type: R/W, X

VRB is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

bits<3:0>

Name: Vector Register C
Mnemonic: VRC
Type: R/W, X

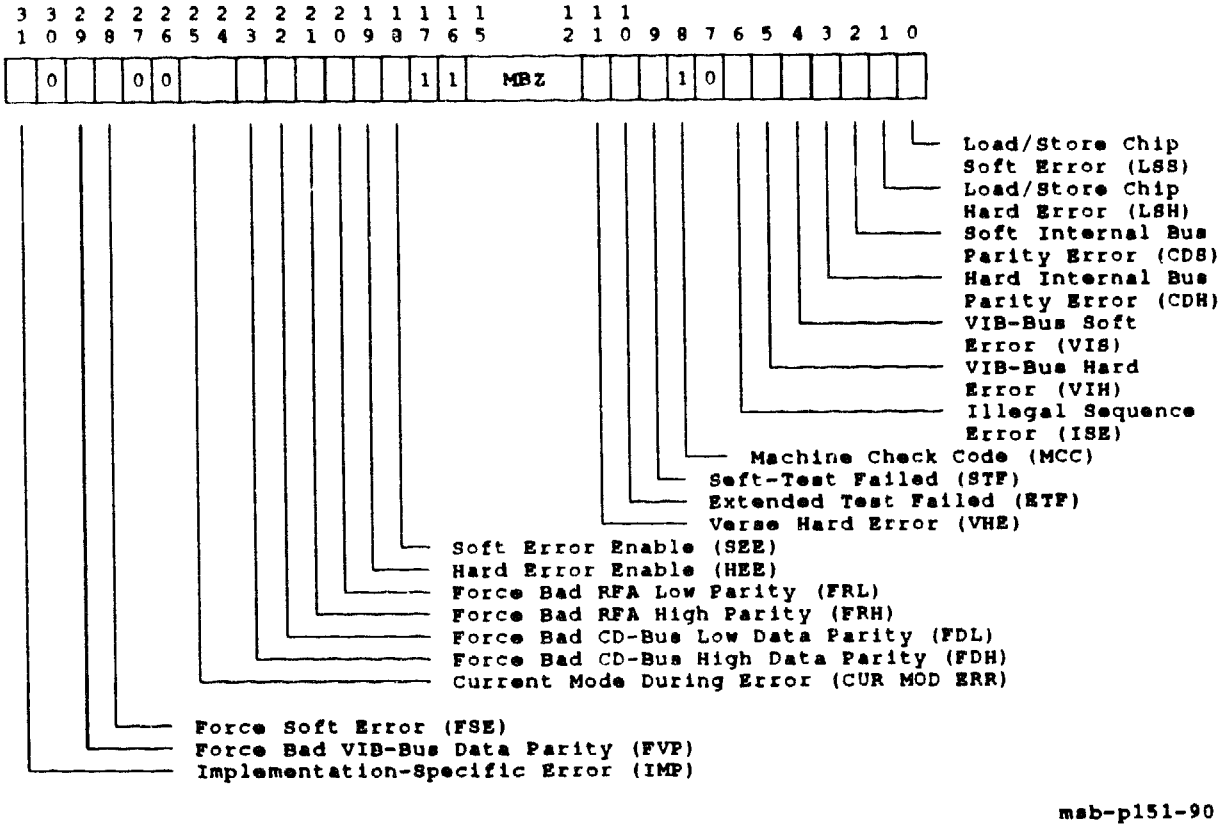
VRC is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

Status Register (VCTL_CSR)

VCTL_CSR gives status for errors within the vector processor.

VCTL_CSR is accessed using the indirect address registers.

ADDRESS *489 (VECTL chip)*



msb-p151-90

bit<31>

Name: Implementation-Specific Error

Mnemonic: IMP

Type: RO, 0

IMP is the "OR" of error bits 1, 3, 5, 6, and 11. It is also reported as bit <24> in VPSR. Writing a one to bit <1>, RST, or to bit <24>, IMP, in VPSR clears all soft error and hard error bits, but not the STF and ETF bits.

bit<30>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bit<29>

Name: Force Bad VIB-Bus Data Parity
Mnemonic: FVP
Type: R/W

FVP, when set, causes the VECTL chip to send bad parity on data transfers across the VIB bus. If bit <28> is set, the retry will be sent with good parity.

bit<28>

Name: Force Soft Error
Mnemonic: FSE
Type: R/W

FSE, when set, causes a forced error on the VIB or the CD bus to retry successfully, causing a soft bus error.

bits<27:26>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bits<25:24>

Name: Current Mode During Error
Mnemonic: CUR MOD ERR
Type: RO

The CUR MOD ERR field contains the complement of the CUR MOD bits in the load/store instruction that incurred the error. This permits the operating system machine check handler to determine what mode the processor was in when the instruction was issued.

bit<23>

Name: Force Bad CD-Bus High Data Parity
Mnemonic: FDH
Type: R/W

FDH, when set, causes the VECTL chip to assert bad parity on CD <63:32>. If bit <28> is clear, then the retry will also have bad CD data parity and this will result in a hard CD bus error. If bit <28> is set, the first transaction will have bad parity, but the retry will be successful causing a soft CD bus error to occur.

bit<22>

Name: Force Bad CD-Bus Low Data Parity
Mnemonic: FDL
Type: R/W

FDL, when set, causes the VECTL chip to assert bad parity on CD <31:0>. If bit <28> is clear, then the retry will also have bad CD data parity and this will result in a hard CD bus error. If bit <28> is set, the first transaction will have bad parity, but the retry will be successful causing a soft CD bus error to occur.

bit<21>

Name: Force Bad RFA High Parity
Mnemonic: FRH
Type: R/W

FRH, when set, causes the VECTL chip to assert bad parity on RFA <10:6>. If bit <28> is clear, then the retry will also have bad RFA parity and this will result in a hard CD bus error. If bit <28> is set, the first transaction will have bad parity, but the retry will be successful causing a soft CD bus error to occur.

bit<20>

Name: Force Bad RFA Low Parity
Mnemonic: FRL
Type: R/W

FRL, when set, causes the VECTL chip to assert bad parity on RFA <5:0>. If bit <28> is clear, then the retry will also have bad RFA parity and this will result in a hard CD bus error. If bit <28> is set, the first transaction will have bad parity, but the retry will be successful causing a soft CD bus error to occur.

bit<19>

Name: Hard Error Enable
Mnemonic: HEE
Type: R/W

HEE enables reporting of vector hard errors through the hard error reporting mechanism.

bit<18>

Name: Soft Error Enable
Mnemonic: SEE
Type: R/W

SEE enables reporting of vector soft errors through the soft error reporting mechanism.

bits<17:16>

Name: Reserved
Mnemonic: None
Type: RO, 1

Unused; must be one.

bits<15:12>

Name: Reserved
Mnemonic: None
Type: RO, 0

Unused; must be zero.

bit<11>

Name: Verse Hard Error
Mnemonic: VHE
Type: R/W1C

Detection: A Verse chip detects a hard error condition and reports it to the VECTL chip.

Correction: No corrective action is possible.

Reporting intent: Hard error interrupt.

bit<10>

Name: Extended Test Failed

Mnemonic: ETF

Type: R/W

Detection: This bit is set at power-up and clears after a successful self-test. The bit is not reset by VPSR bit <1>, RST. When ETF is set, the VPSR IMP bit is not set. This bit permits the scalar processor to detect whether the extended test passed, or causes the vector processor to be disabled if the extended test does not pass.

Correction: No corrective action is possible.

Reporting intent: Permanently disable.

bit<9>

Name: Self-Test Failed

Mnemonic: STF

Type: R/W

Detection: This bit is set at power-up and clears after a successful self-test. The bit is not reset by VPSR bit <1>, RST. When STF is set, the VPSR IMP bit is not set. This bit permits the scalar processor to detect whether self-test passed, or causes the vector processor to be disabled if self-test does not pass.

Correction: No corrective action is possible.

Reporting intent: Permanently disable.

bits<8:7>

Name: Machine Check Code

Mnemonic: None

Type: RO, 1, 0

The value 1,0 forces a machine check and is permanently set. If a hard error occurs in the machine while an IPR read is outstanding, the VECTL chip returns this CSR with an error response. Bits <8:7> determine the scalar action on receipt of the data.

bit<6>

Name: Illegal Sequence Error

Mnemonic: ISE

Type: W1C

Detection: When set, indicates that an illegal sequence occurred. The following sequences set this bit:

- Illegal instruction issue sequence
- Illegal VIB command
- Asserting Load/Store exception after MMOK

Correction: No correction is possible.

Reporting intent: Machine check if illegal read sequence; hard error interrupt otherwise.

bit<5>

Name: VIB-Bus Hard Error

Mnemonic: VIH

Type: R/W1C

Detection: The VECTL chip detected a hard parity error condition on the VIB bus.

Correction: The attempted retry failed.

Reporting intent: Hard error interrupt.

bit<4>

Name: VIB-Bus Soft Error

Mnemonic: VIS

Type: R/W1C

Detection: The VECTL chip detected a parity error in the instruction packet being sent by the scalar processor.

Correction: A retry will be attempted by the scalar processor.

Reporting intent: Soft error interrupt.

bit<3>

Name: Hard Internal Bus Parity Error

Mnemonic: CDH

Type: R/W1C

Detection: The VECTL chip detected or was informed of an unrecoverable parity error on the vector processor internal bus (the error was not recovered by retrying the transaction).

Correction: Unrecoverable error has occurred.

Reporting intent: Machine check on . ad within current context; hard error interrupt otherwise.

bit<2>

Name: Soft Internal Bus Parity Error

Mnemonic: CDS

Type: R/W1C, 0

Detection: The VECTL chip detected a parity error on the vector processor internal bus (the error was recovered by retrying the transaction, if bit <3> (CDH) is clear).

Correction: CD bus master has recovered the error by retry.

Reporting intent: Soft error interrupt.

bit<1>

Name: Load/Store Chip Hard Error

Mnemonic: LSH

Type: R/W1C, 0

Detection: A hard error condition occurred during an operation being performed by the Load/Store chip. The exact cause is stored in the status registers in the Load/Store chip.

Correction: Unrecoverabl Load/Store error has occurred.

Reporting intent: Machine check before MMOK; hard error interrupt after MMOK.

bit<0>

Name: Load/Store Chip Soft Error

Mnemonic: LSS

Type: R/W1C, 0

Dectection: A soft error or exception condition occurred during an operation being performed by the Load/Store chip. The cause is stored in the status registers in the Load/Store chip.

Correction: The Load/Store chip soft recovered the error.

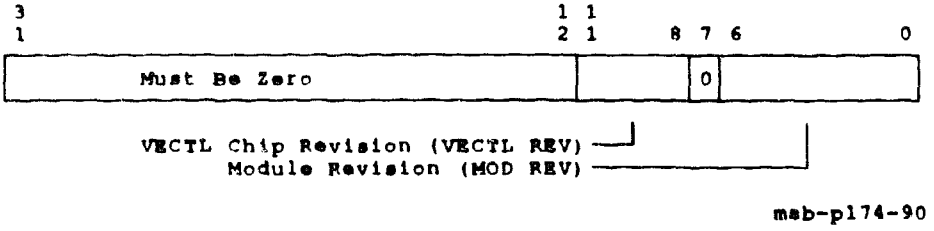
Reporting intent: Soft error interrupt to the scalar processor.

Module Revision Register (MOD_REV)

MOD_REV contains the vector module revision and the VECTL chip revision.

MOD_REV is accessed using the indirect address registers.

ADDRESS *48A (VECTL chip)*



mab-p174-90

bits<31:12>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<11:8>

Name: VECTL Chip Revision

Mnemonic: VECTL REV

Type: RO, 0

VECTL REV contains the revision of the VECTL chip.

bit<7>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved to Digital.

bits<6:0>

Name: Module Revision

Mnemonic: MOD REV

Type: RO, 0

MOD REV contains the revision of the vector module.

Vector Copy–P0 Base Register (LSX_P0BR)

LSX_P0BR is the vector module's copy of the scalar CPU P0BR.

LSX_P0BR is accessed using the indirect address registers.

ADDRESS

RFA 500 (L/S)

mab-p129-90

bits<31:30>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<31:9>

Name: Vector Copy–P0 Base Register

Mnemonic: P0BR

Type: WO, X

P0BR is the vector module's copy of the scalar CPU P0BR. Both copies are updated at the same time. When P0BR is updated, the vector translation buffer is flushed.

bits<8:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

Vector Copy–P0 Length Register (LSX_P0LR)

LSX_P0LR is the vector module's copy of the scalar CPU P0LR.

LSX_P0LR is accessed using the indirect address registers.

ADDRESS

RFA 501 (L/S)

msb-p130~90

bits<31:22>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<21:0>

Name: Vector Copy–P0 Length Register

Mnemonic: P0LR

Type: WO, X

P0LR is the vector module's copy of the scalar CPU P0LR. Both copies are updated at the same time. When P0LR is updated, the vector translation buffer is flushed.

Vector Copy–P1 Base Register (LSX_P1BR)

LSX_P1BR is the vector module's copy of the scalar CPU P1BR.

LSX_P1BR is accessed using the indirect address registers.

ADDRESS

RFA 502 (L/S)

msb-p131-90

bits<31:30>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<29:9>

Name: Vector Copy–P1 Base Register

Mnemonic: P1BR

Type: WO, X

P1BR is the vector module's copy of the scalar CPU P1BR. Both copies are updated at the same time. When P1BR is updated, the vector translation buffer is flushed.

bits<8:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

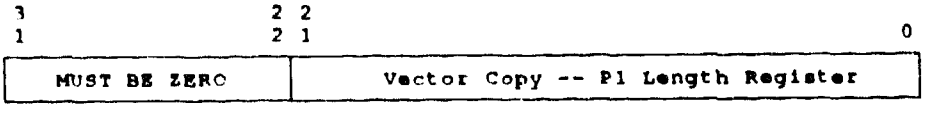
Vector Copy–P1 Length Register (LSX_P1LR)

LSX_P1LR is the vector module's copy of the scalar CPU P1LR.

LSX_P1LR is accessed using the indirect address registers.

ADDRESS

RFA 503 (L/S)



msb-p132-90

bits<31:22>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<21:0>

Name: Vector Copy–P1 Length Register

Mnemonic: P1LR

Type: WO, U

P1LR is the vector module's copy of the scalar CPU P1LR. Both copies are updated at the same time. When P1LR is updated, the vector translation buffer is flushed.

Vector Copy–System Base Register (LSX_SBR)

LSX_SBR is the vector module's copy of the scalar CPU SBR.

LSX_SBR is accessed using the indirect address registers.

ADDRESS

RFA 504 (L/S)

mab-p133-90

bits<31:30>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<31:9>

Name: Vector Copy–System Base Register

Mnemonic: SBR

Type: WO, X

SBR is the vector module's copy of the scalar CPU SBR. Both copies are updated at the same time. When SBR is updated, the vector translation buffer is flushed.

bits<8:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

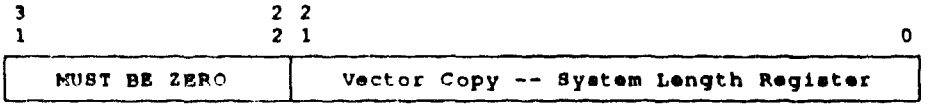
Vector Copy–System Length Register (LSX_SLR)

SLR is the vector module's copy of the scalar CPU SLR.

LSX_SLR is accessed using the indirect address registers.

ADDRESS

RFA 505 (L/S)



msb-p134-90

bits<31:22>

Name: Reserved

Mnemonic: None

Type: RO, 0

Unused; must be zero.

bits<21:0>

Name: Vector Copy–System Length Register

Mnemonic: SLR

Type: WO, X

SLR is the vector module's copy of the scalar CPU SLR. Both copies are updated at the same time. When SLR is updated, the vector translation buffer is flushed.

Load/Store Exception Register (LSX_EXC)

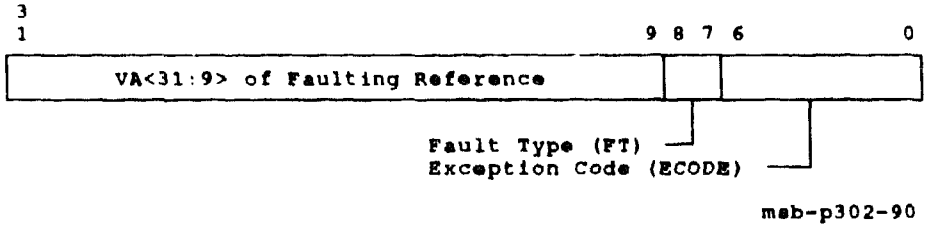
LSX_EXC contains the memory management fault parameters when an exception occurs during a vector load or store instruction.

When the Translation Buffer Control Register <MME> bit is clear, the vector translation buffer is turned off. The contents of LSX_EXC are UNPREDICTABLE in physical mode.

LSX_EXC is accessed using the indirect address registers.

ADDRESS

RFA 508 (L/S)



bits<31:9>

Name: Faulting VA Page Address

Mnemonic: VA<31:9>

Type: RO, 0

This field contains the virtual page address that caused the memory management fault being flagged. Memory management exceptions are not regarded as soft or hard errors by the Load/Store chip and are reported differently from hard/soft errors.

bits<8:7>

Name: Fault Type

Mnemonic: FT

Type: RO, 0

FT defines the type of fault that occurs.

<8:7>	Fault
0 1	Translation not valid fault (TNV) PTE<31>= 0 for a PTE.
1 1	Access control violation (ACV) Caused by one of the following: protection check error, vector alignment fault, I/O space reference, length check.

bits<6:0>

Name: Exception Code

Mnemonic: ECODE

Type: RO, X

A set bit indicates an abort and a memory management exception sent to the VECTL chip.

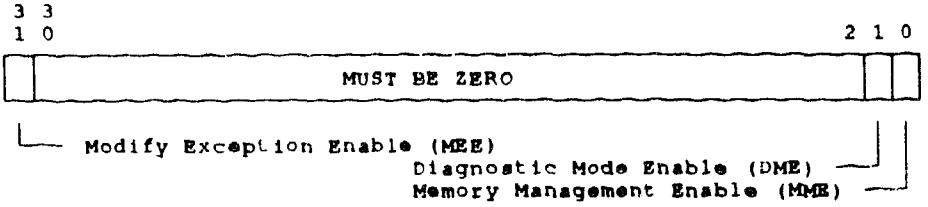
Bit	Name	Type	Cause
5	Modify Exception	Modify fault	The faulting reference attempted to write a page whose PTE<M> bit was not set. This bit indicates that the vector processor is running in virtual machine mode. To enable virtual machine mode, the MEE bit is set in TBCSR. This exception is accompanied by the TNV fault type code. This code should be ignored by scalar microcode and a modify exception taken.
4	I/O Space Reference (I)	ACV	The faulting reference attempted to access I/O space.
3	Alignment fault (A)	ACV	The faulting reference was unaligned.
2	Modify Intent (M)	ACV or TNV	The faulting reference was a write transaction. This bit indicates to the operating system that the faulting reference will set the M bit, which permits the operating system to preset the M bit in the PTE.
1	PTE Reference (P)	ACV or TNV	The fault occurred during a PTE fetch rather than a normal P0, P1, or system space reference. The bit is set either while attempting to fetch a system space PTE to translate a process space virtual address or while fetching a PTE that translates a system space reference.
0	Length Check (L)	ACV	The faulting reference was due to a length check.

Translation Buffer Control Register (LSX_TBCSR)

LSX_TBCSR controls the translation buffer (TB) during testing.

LSX_TBCSR is accessed using the indirect address registers.

ADDRESS *RFA 509 (L/S)*



bit<31>

Name: Modify Exception Enable

Mnemonic: MEE

Type: R/W, 0

Writing a one to MEE causes the Load/Store chip to generate modify exceptions whenever a write is attempted to a page whose PTE<M> bit is not set. The value of MEE is zero for normal operation.

bit<1>

Name: Diagnostic Mode Enable

Mnemonic: DME

Type: R/W, 0

Writing a one to the DME bit causes the TB to enter diagnostic mode.

bit<0>

Name: Memory Management Enable

Mnemonic: MME

Type: R/W, 1

Writing a one to the MME bit enables memory management. The vector processor memory management should be disabled only for diagnostic purposes.

Vector Copy–Memory Management Enable Register (LSX_MAPEN)

LSX_MAPEN is the vector module's copy of the scalar CPU MAPEN.

LSX_MAPEN is accessed using the indirect address registers.

ADDRESS

RFA 50A (L/S)

msb-p135-90

bits<31:0>

Name: Memory Management Enable

Mnemonic: MAPEN

Type: WO, X

When MAPEN is updated in the scalar CPU, the vector module copy is also written. MAPEN is a pseudo register, meaning that it only exists to permit a vector translation buffer flush to occur when MAPEN in the scalar CPU is changed.

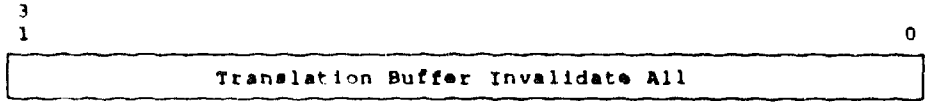
Vector Copy–Translation Buffer Invalidate All Register (LSX_TBIA)

LSX_TBIA is the vector module copy of the scalar CPU TBIA.

LSX_TBIA is accessed using the indirect address registers.

ADDRESS

RFA 50B (L/S)



msb-p136-90

bits<31:0>

Name: Translation Buffer Invalidate All

Mnemonic: TBIA

Type: WO, X

When TBIA is updated in the scalar CPU, the vector module copy is also written.

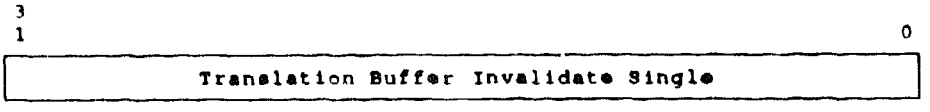
Vector Copy–Translation Buffer Invalidate Single Register (LSX_TBIS)

LSX_TBIS is the vector module copy of the scalar CPU TBIS.

LSX_TBIS is accessed using the indirect address registers.

ADDRESS

RFA 50C (L/S)



msb-p137-90

bits<31:0>

Name: Translation Buffer Invalidate Single

Mnemonic: TBIS

Type: WO, X

When TBIS is updated in the scalar CPU, the vector module copy is also written.

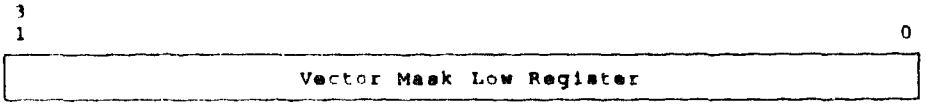
Vector Mask Low Register (LSX_MASKLO)

LSX_MASKLO is written with the contents of the mask register prior to a masked load/store operation taking place. The register is written whenever the Verse mask registers are read. The Load/Store chip detects the read request and loads the data into the high or low longword as appropriate.

LSX_MASKLO is accessed using the indirect address registers.

ADDRESS

RFA 510 (L/S)



mab-p304-90

bit<31:0>

Name: Vector Mask Low

Mnemonic: LSX_MASKLO

Type: R/W, X

LSX_MASKLO is written with the contents of the mask register prior to a masked load/store operation taking place. The register is written whenever the Verse mask registers are read. The Load/Store chip detects the read request and loads the data into the high or low longword as appropriate.

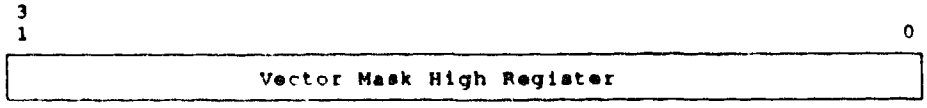
Vector Mask High Register (LSX_MASKHI)

LSX_MASKHI is written with the contents of the mask register prior to a masked load/store operation taking place. The register is written whenever the Verse mask registers are read. The Load/Store chip detects the read request and loads the data into the high or low longword as appropriate.

LSX_MASKHI is accessed using the indirect address registers.

ADDRESS

RFA 511 (L/S)



msb-p305-90

bit<31:0>

Name: Vector Mask High

Mnemonic: LSX_MASKHI

Type: R/W, X

LSX_MASKHI is written with the contents of the mask register prior to a masked load/store operation taking place. The register is written whenever the Verse mask registers are read. The Load/Store chip detects the read request and loads the data into the high or low longword as appropriate.

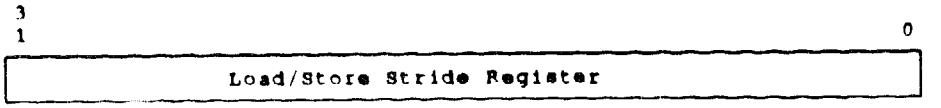
Load/Store Stride Register (LSX_STRIDE)

LSX_STRIDE contains the signed 32-bit stride that gives the address increments for each vector element. The stride must be loaded before the load or store instruction is issued.

LSX_STRIDE is accessed using the indirect address registers.

ADDRESS

RFA 512 (L/S)



mab-p306-90

bits<31:0>

Name: Load/Store Stride

Mnemonic: LSX_STRIDE

Type: R/W, X

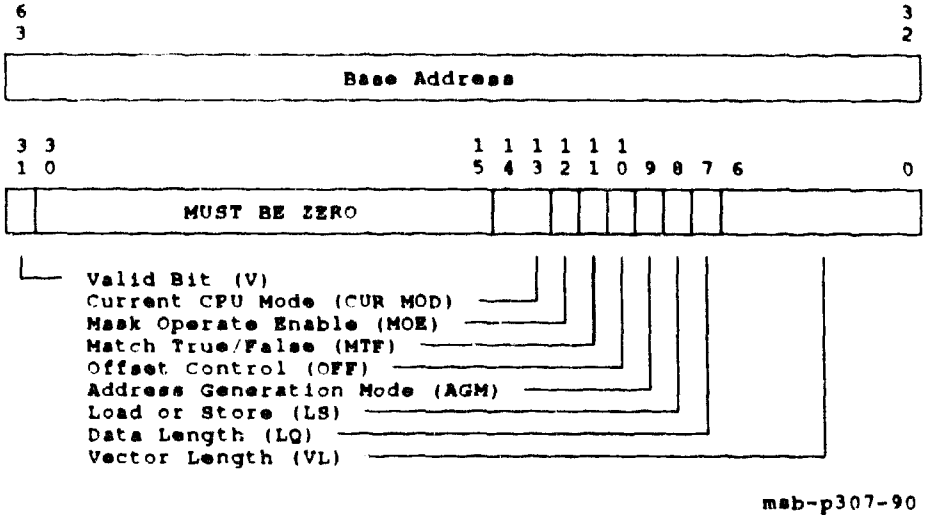
LSX_STRIDE contains the signed 32-bit stride that gives the address increments for each vector element. The stride must be loaded before the load or store instruction is issued.

Load/Store Instruction Register (LSX_INST)

LSX_INST is used to start a load/store operation. A write to this register causes the Load/Store chip to start executing the instruction. Before writing this register the LSX_STRIDE register must contain a valid stride.

LSX_INST is accessed using the indirect address registers.

ADDRESS *RFA 513 (L/S)*



msb-p307-90

bits<63:32>

Name: Base Address

Mnemonic: None

Type: R/W, X

This field gives the virtual base address for the load/store or

bit<31>

Name: Valid

Mnemonic: V

Type: WO, X

The V bit, when set, indicates that the instruction that was written is a valid instruction and should be executed. The VECTL chip sets this bit when it issues an instruction.

If this register is written with bit <31> = 1, the Load/Store chip will start executing an instruction that produces UNPREDICTABLE results. Diagnostics should always write bit <31> = 0 to prevent this problem.

bits<14:13>

Name: Current CPU Mode

Mnemonic: CUR MOD

Type: WO, X

CUR MOD correspond to the scalar CPU Current Mod bits in the PSL in effect at the time an instruction is issued.

bit<12>

Name: Mask Operate Enable

Mnemonic: MOE

Type: WO, X

When set, the mask register in the Load/Store chip is used to control the transaction.

bit<11>

Name: Match True/False

Mnemonic: MTF

Type: WO, X

MTF is used when bit <12> (MOE) is set to determine the sense of the mask operation. Only bits in the mask register corresponding to the sense of MTF are processed.

bit<10>

Name: Offset Control

Mnemonic: OFF

Type: WO, X

OFF is only valid if the AGM bit, bit <9> = 1. When OFF = 1, the Load/Store chip loads the offset register into an internal FIFO. Following this instruction the VECTL chip issues the same instruction with OFF = 0, which causes the Load/Store chip to perform the scatter/gather operation using the previously stored offsets.

bit<9>

Name: Address Generation Mode
Mnemonic: AGM
Type: WO, X

AGM specifies the address generation mode to be used. If AGM = 0, a strided write load/store operation will be executed. If AGM = 1, a scatter/gather operation using an offset vector register will take place.

bit<8>

Name: Load or Store
Mnemonic: LS
Type: WO, X

LS specifies the direction of transfer of data. LS = 0 causes a load or gather operation to take place; LS = 1 causes a store or scatter operation to take place.

bit<7>

Name: Data Length
Mnemonic: LQ
Type: WO, X

LQ specifies if the load/store or scatter/gather instruction transfers longwords (LQ = 0) or quadwords (LQ = 1).

bits<6:0>

Name: Vector Length
Mnemonic: VL
Type: WO, X

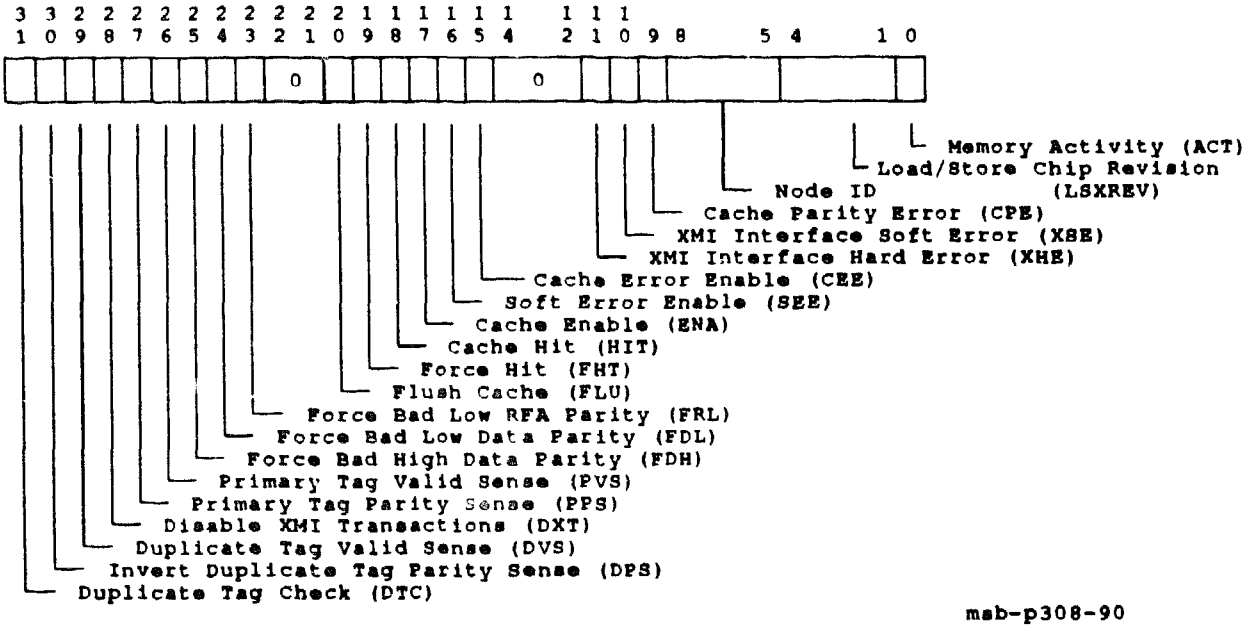
The VL field contains the contents of the Vector Length Register at the time the load/store instruction was received from the scalar processor. It can have values between 0–64. A value greater than 64 produces UNPREDICTABLE results.

Cache Control Register (LSX_CCSR)

LSX_CCSR provides control over the vector cache functions. Bit <28>, DXT, permits diagnostic features to be enabled or disabled. Bits <30>, DPS, and <29>, DVS, should only be changed when DXT = 0. Changing DPS or DVS while DXT = 1 can cause UNDEFINED operations.

LSX_CCSR is accessed using the indirect address registers.

ADDRESS *RFA 520 (L/S)*



bit<31>

Name: Duplicate Tag Check

Mnemonic: DTC

Type: R/W, 0

When DXT bit <28> is set and DTC is clear, duplicate tag locations can be written, as determined by the DVS <29> and DPS <30> bits. If DTC is set, the duplicate tag store is unchanged.

bit<30>

Name: Invert Duplicate Tag Parity Sense

Mnemonic: DPS

Type: R/W, 0

DPS specifies the duplicate tag parity used when diagnostics are enabled (DXT bit is set), and when bit <31>, DTC, is clear, and a store or scatter instruction is executed. If DPS is clear, odd parity is written. If DPS is set, even parity is written in the duplicate tag store.

During normal operation odd parity is used in the duplicate tag store.

bit<29>

Name: Duplicate Tag Valid Sense

Mnemonic: DVS

Type: R/W, 0

When DXT bit <28> is set, the DTC bit, <31>, is clear, and a store or scatter instruction is executed, the duplicate tag valid bit is written with the value in DVS.

bit<28>

Name: Disable XMI Transactions

Mnemonic: DXT

Type: R/W, 0

DXT, when set, permits diagnostics to perform operations in the cache without generating XMI transactions on cache miss and writes. Any physical address presented to the cache is truncated to a 19-bit longword-aligned address. When DXT is set, all tag comparisons that differ are flagged as tag parity errors.

Bits <30>, DPS, and <29>, DVS, should only be changed when DXT = 0. Changing DPS or DVS while DXT = 1 can cause UNDEFINED operations.

DXT, when clear, is the setting for normal machine operation.

bit<27>

Name: Primary Tag Parity Sense

Mnemonic: PPS

Type: R/W, 0

When bit <28> (DXT) is set, the sense of the parity is determined by PPS. If PPS = 1, even parity is used; if PPS = 0, odd parity is used. This inversion applies to both reads and writes of the primary tag store.

bit<26>

Name: Primary Tag Valid Sense
Mnemonic: PVS
Type: R/W, 0

When bit <28> (DXT) is set, the sense of the tag Valid bit is determined by PVS. When the tag is being compared, the V bit is compared to PVS. If they differ, a cache parity error is flagged. This inversion applies to both reads and writes of the primary tag store.

bit<25>

Name: Force Bad High Data Parity
Mnemonic: FDH
Type: R/W, 0

When set, FDH causes all Load/Store originated data transfers to have bad data parity on the internal bus, CD<63:32>.

bit<24>

Name: Force Bad Low Data Parity
Mnemonic: FDL
Type: R/W, 0

When set, FDL causes all Load/Store originated data transfers (cache fills) to have bad data parity on the internal bus, CD<31:0>.

bit<23>

Name: Force Bad Low RFA Parity
Mnemonic: FRL
Type: R/W, 0

When set, FRL causes all Load/Store originated low RFAs to have bad parity.

bits<22:21>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<20>

Name: Flush Cache
Mnemonic: FLU
Type: R/W, 0

Writing a one to FLU causes the cache flush cycle to take place. This bit remains set while the cache flush completes. Because of the cache flush mechanism, this bit can never be read in the one state; it will always appear as a zero. This bit must not be set until all error bits are clear, and the VMAC register has been read.

bit<19>

Name: Force Hit
Mnemonic: FHT
Type: R/W, 0

When set, FHT causes the cache to generate a hit on a cache reference. This is true for loads and stores, and this bit overrides the enable bit. If FHT = 1 and ENA = 0, the cache hit is still flagged.

bit<18>

Name: Cache Hit
Mnemonic: HIT
Type: R/W, 0

HIT is the latched output of the cache comparator. This bit is only updated on cache references from the Load/Store chip. HIT is not modified if bit <17> ENA is not set.

bit<17>

Name: Cache Enable
Mnemonic: ENA
Type: R/W, 1

ENA enables the data cache, when set, and disables the data cache when clear. When ENA is clear, all cache references are miss. The cache is disabled if any error bits (DCE, TPE) are set, even if ENA is set. Following a DCE or TPE error, the ENA bit will not change state.

bit<16>

Name: Soft Error Enable
Mnemonic: SEE
Type: R/W, 1

SEE enables reporting of XMI soft errors. This bit provides the mechanism for suppressing excessive memory CRDs or duplicate tag parity error interrupts.

bit<15>

Name: Cache Error Enable
Mnemonic: CEE
Type: R/W, 1

CEE enables reporting of cache errors. All cache errors are recovered by the Load/Store chip. This enable bit does not cover duplicate tag store parity errors. This bit provides the mechanism for suppressing excessive cache parity error interrupts.

bits<14:12>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<11>

Name: XMI Interface Hard Error
Mnemonic: XHE
Type: R/W1C, 0

When set, XHE indicates an XMI hard error has occurred. Writing a one to XHE clears it.

bit<10>

Name: XMI Interface Soft Error
Mnemonic: XSE
Type: R/W1C, 0

When set, XSE indicates an XMI soft error has occurred. The sources for this error are Duplicate Tag Parity Error or Corrected Read Data from memory. Writing a one to XSE clears it.

bit<9>

Name: Cache Parity Error
Mnemonic: CPE
Type: R/W1C, 0

CPE sets when a parity error is detected on a cache read from the data cache or when a tag parity error occurs on tag lookup. CPE is reset by writing a one to it.

bits<8:5>

Name: Node ID
Mnemonic: None
Type: RO, -

This field contains the XMI node ID.

bits<4:1>

Name: Load/Store Chip Revision
Mnemonic: LSXREV
Type: RO, X

This field contains the revision of the Load/Store chip.

bit<0>

Name: Memory Activity
Mnemonic: ACT
Type: RO, X

When set, ACT indictates that the vector processor is performing memory references. This bit is used by the VECTL chip to implement the MSYNC and MFPR #VMAC, dst functions.

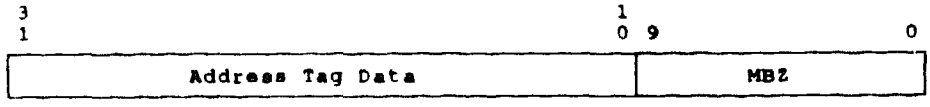
Translation Buffer Tag Register (LSX_TBTAG)

LSX_TBTAG is a pseudo-register. A write to this register causes the data to be written into the translation buffer (TB) location specified by the Translation Buffer Control Register (TBCSR) if the TBCSR DME bit <1> is set or written into the location specified by the replacement algorithm if the TBCSR DME bit is not set. A read operates in a similar fashion.

Writing different TB locations with the same tag causes contention and can damage the Load/Store chip.

LSX_TBTAG is accessed using the indirect address registers.

ADDRESS *RFA 530 (L/S)*



mab-p309-90

bits<31:10>

Name: Address Tag Data

Mnemonic: None

Type: R/W, X

The address tag data corresponds to bits <31:10> of the virtual address.

bits<9:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

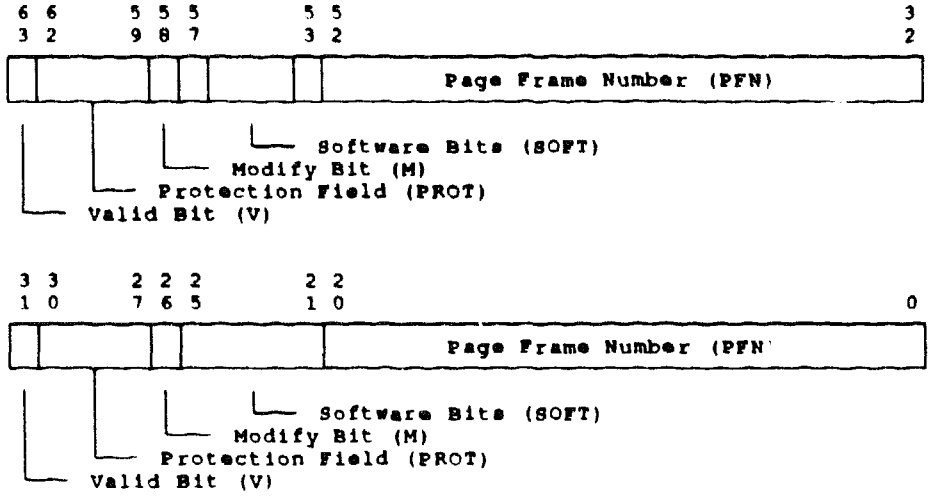
Translation Buffer PTE Register (LSX_PTE)

LSX_PTE is a 64-bit pseudo-register. A write to LSX_PTE causes the PTE to be written. The location written is specified either by the Translation Buffer Control Register (if the TBCSR<DME> bit is enabled) or by a previous translation buffer hit. A read works in a similar fashion.

LSX_PTE is accessed using the indirect address registers.

ADDRESS

RFA 531 (L/S)



msb-p310-90

bit<63>

Name: Valid

Mnemonic: V

Type: R/W, X

V indicates the validity of <58> (M) and <52:32> (PFN). V = 1 for valid, and V = 0 for invalid.

bits<62:59>

Name: Protection

Mnemonic: PROT

Type: R/W, X

PROT contains the protection code for referencing the page. This field is valid even if V = 0.

bit<58>

Name: Modify
Mnemonic: M
Type: R/W, X

If M = 1, the page has been written and any permissible writes can take place. If M = 0 and write access is permitted, a write attempt to this page will cause the Load/Store chip to enter its modify flows and the write is blocked. When the V bit <63> is clear, the M bit is reserved for use by Digital software. When the V bit is set, the M bit indicates the status of the page.

bits<57:53>

Name: Software
Mnemonic: SOFT
Type: R/W, X

This field is used to record software information for each PTE so that there is a record whenever the Modify bit <58> is set. The translation buffer does not keep these bits.

bits<52:32>

Name: Page Frame Number
Mnemonic: PFN
Type: R/W, X

PFN gives the upper 21 bits of the physical address of the base of the page. Memory management uses the value of PFN only if the Valid bit <63> is set (V = 1).

bit<31>

Name: Valid
Mnemonic: V
Type: R/W, X

This bit indicates the validity of <26> (M) and <20:0> (PFN). V = 1 for valid, and V = 0 for invalid.

bits<30:27>

Name: Protection
Mnemonic: PROT
Type: R/W, X

PROT contains the protection code for referencing the page. This field is valid even if V = 0.

bit<26>

Name: Modify
Mnemonic: M
Type: R/W, X

If M = 1, the page has been written and any permissible writes can take place. If M = 0 and write access is permitted, a write attempt to this page will cause the Load/Store chip to enter its modify flows and the write is blocked. When the V bit <31> is clear, the M bit is reserved for use by Digital software. When the V bit is set, the M bit indicates the status of the page.

bits<25:21>

Name: Software
Mnemonic: SOFT
Type: R/W, X

This field is used to record software information for each PTE so that there is a record whenever the Modify bit <26> is set. The translation buffer does not keep these bits.

bits<20:0>

Name: Page Frame Number
Mnemonic: PFN
Type: R/W, X

PFN gives the upper 21 bits of the physical address of the base of the page. Memory management uses the value of PFN only if the Valid bit <31> is set (V = 1).

4.9 Error Handling

The vector processor provides extensive error detection and reporting. Errors and interrupts are reported to the scalar processor. Soft errors are automatically retried, and hard errors cause the vector processor to disable itself.

Table 4–4 Vector Processor Hardware-Detected Errors

Error	Description
Machine check	A hardware error occurred synchronously with the CPU-chip's execution of instructions. Instruction-level recovery and retry may be possible.
Hard error interrupt	A hardware error occurred asynchronously with the CPU-chip's execution of instructions. Not recoverable.
Soft error interrupt	A hardware error occurred that was not fatal to the process or system. System error software should be able to recover and continue.

The FV64A vector processor has been designed to provide extensive error detection. Parity detection is the major technique used on the buses and the large RAM structures. The VECTL chip also detects illegal instructions and sequences.

The vector module provides for:

- Standard XMI error detection
- Parity on all cache data and tag stores
- Parity coverage on the internal data bus, both address and data
- Parity protection on the control and data signals on the VIB bus
- Instruction consistency checks

The vector module at times can detect an error immediately, or it may take several cycles. The vector processor attempts to correct all errors that occur. Some errors are fatal, and the instruction cannot be retried. When an error occurs, the last atomic transaction that contained the error is retried if possible.

The vector processor attempts to report all errors in a standard fashion. Soft errors are reported by interrupts. Hard errors are reported either by a hard error interrupt or by a machine check. The vector processor consistently reports errors, but the system-level error result depends on the instruction stream. An interrupt or machine check can vary because of the timing between the error and the reaction of the scalar processor to the error.

A hard error interrupt can cause a disable fault flow to be started because of the way the scalar microcode handles vector instructions. If this occurs, the interrupt is taken and by the time the scalar processor returns to the disable flow the error condition has been reported. The operating system must recognize this condition.

Soft errors may be followed by hard errors. Therefore, the soft error interrupt handler may receive no valid data because the hard error interrupt service routine has already taken the interrupt.

An error is treated either as a fault from which there is no recovery or as a transient error with recovery possible after a single retry. The FV64A vector processor retries atomic transactions. An atomic transaction is defined as a collection of one or more small transactions that can be restarted within the machine. An atomic transaction can be as small as an internal bus transfer or as large as a complete load/store instruction. Examples of atomic transactions are as follows:

- Single transaction error retry. A parity error is detected during a data transfer on the internal bus. The bus master retries the transaction.
- A cache parity error occurs during a load. The load instruction is restarted.

The operating system handles error reporting in three ways (see Table 4–5):

- Hard errors are flagged as machine checks through SCB vector 04 (hex).
- Hard errors interrupt through SCB vector 60 at IPL 1D (hex).
- Soft errors interrupt through SCB vector 54 at IPL 1A (hex).

Table 4–5 Vector Processor Error Reporting

Reporting Intent	System-Level Error Result	Conditions
Machine check	Machine check SCB vector 04 (hex)	Hard error within current read context
Hard error interrupt	Hard error interrupt SCB vector 60 IPL 1D (hex)	Hard error interrupt only
	Disable fault followed by hard error interrupt	Hard error interrupt
	Soft error interrupt followed by hard error interrupt	Hard error interrupt
	Machine check SCB vector 04 (hex)	Read after hard error flagged by C-chip
Soft error interrupt	Soft error interrupt SCB vector 54 IPL 1A (hex)	Soft error interrupt only

4.9.1 Machine Checks

Machine checks occur when the operation requested by the scalar processor cannot be completed because a hard error has occurred. Because there are several sources of machine checks, the error registers are conditionally logged to prevent further machine checks.

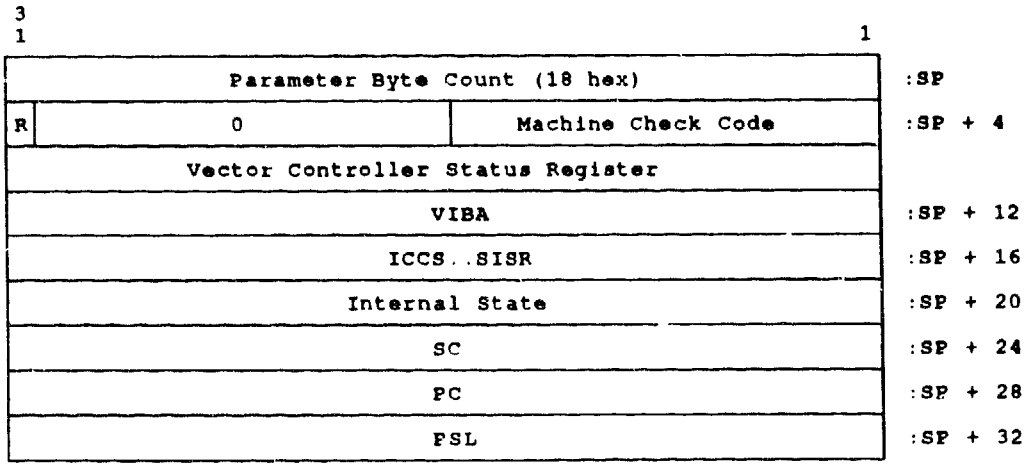
The following error conditions cause machine checks:

- Hard errors on read requests that occur within the context of the read
- Load/store hard errors during the memory management phase
- Any read passed to the C-chip while the VINTSR register is locked
- Any EPR read to the VECTL for an outstanding MFVP instruction or load/store operation that cannot be completed
- Any read request to the Vector Memory Activity Check Register (VMAC) after a hard error has occurred

An error within the context of a read means that the error is associated with the read data. An example is a CD bus parity error when reading a Load/Store register. An XMI error while the scalar processor is reading a Load/Store register is an example of an error outside the context of the read request.

Figure 4–14 shows the frame pushed onto the stack by a machine check related to the vector module (MCHK_VECTOR_STATUS 14 (hex)).

Figure 4–14 Machine Check Stack Frame



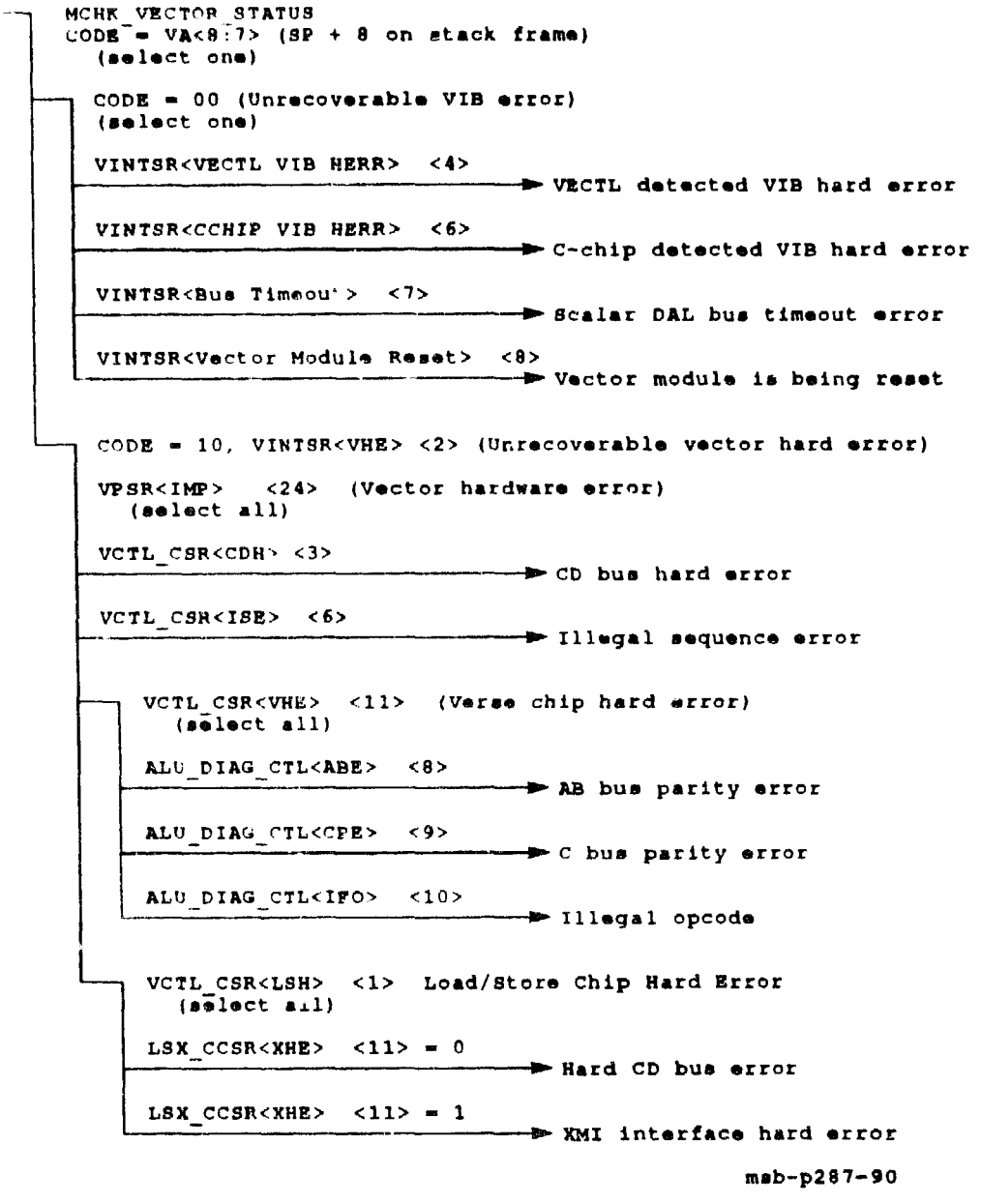
The operating system must guarantee that all errors are reported within the context of the vector processor's current process. To permit this, the vector processor generates a machine check when a hard error occurs during a read of the VMAC register.

The programmer must do the following to maintain synchronization.

CAUTION: To ensure that the barrier synchronization operates correctly, the programmer must read the Vector Processor Status Register (VPSR), IPR144, and spin on reading this register until the VPSR Busy bit is clear. Then reading the Vector Memory Activity Check Register (VMAC), IPR146, *twice* guarantees synchronization and that all errors have been reported.

Depending on the error, the machine check error is parsed as shown in Figure 4–15.

Figure 4–15 Machine Check Parse Tree



4.9.2 Hard Error Interrupt

When the vector processor encounters a hard error asynchronously to the scalar processor, it disables itself and flags a hard error interrupt to the scalar processor through SCB vector 60 at IPL 1D (hex). A hard error pushes the PC/PSL pair to the stack.

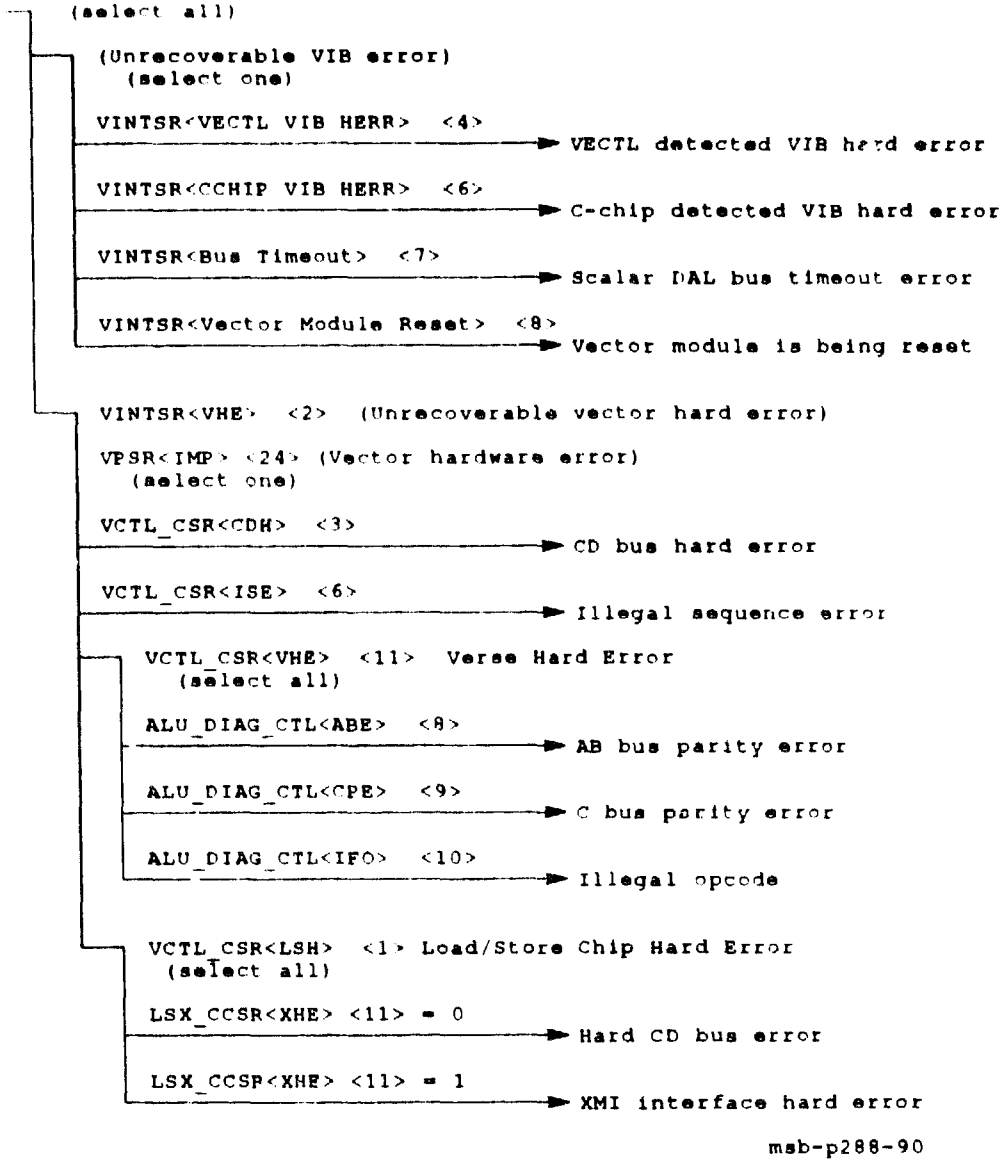
When a hard error occurs that is asynchronous to the currently executing instruction stream, a hard error interrupt is posted. Examples of this are:

- Any hard error on write transactions
- Any hard error on load/store operations following the memory management phase when the scalar processor has been released
- Parity errors on the Verse/Favor chip interfaces
- Errors that occur asynchronously to the current operation

When a hard error is flagged, the vector processor is disabled. If another instruction is then issued, it is ignored unless it is an MxPR instruction. The C-chip locks the VIB interface and any attempt to read registers across the VIB will be machine checked until the VINTSR IPR in the scalar processor's C-chip is unlocked. After unlocking the VINTSR register, the scalar processor can examine the state of the vector processor to determine the cause of the error. Because there are several sources of hard errors, the error registers are conditionally logged to prevent hard errors or machine checks.

Figure 4–16 shows how hard errors are parsed.

Figure 4–16 Hard Error Interrupt Parse Tree



4.9.3 Soft Error Interrupt

Soft errors are reported by soft error interrupts. Soft error interrupts are the result of errors that are successfully retried.

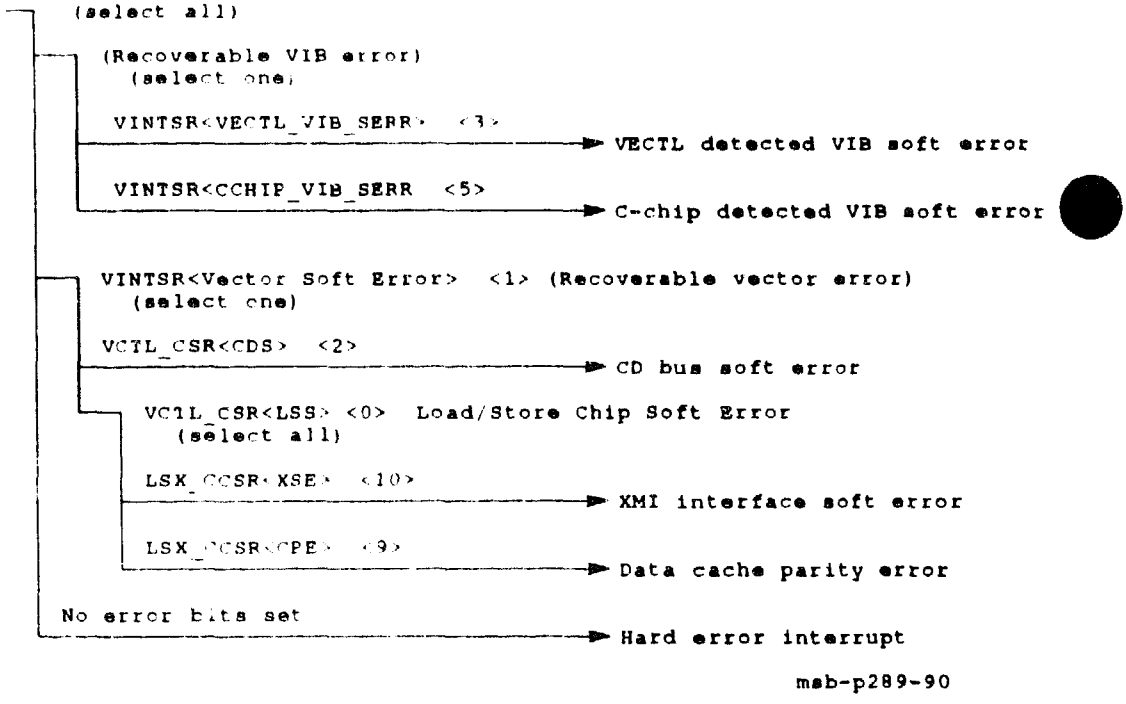
If a soft error and a hard error occur simultaneously, the soft error interrupt is posted, but the hard error interrupt takes precedence. There are many examples where the first occurrence causes a soft error interrupt, and the subsequent error during retry causes a hard error interrupt. The operating systems must be able to support both soft and hard error interrupts for the same error.

When the vector processor encounters a soft error, it posts a soft error interrupt to the scalar processor through SCB vector 54 at IPL 1A (hex). A soft error pushes the PC/PSL pair to the stack.

A soft error interrupt may be followed by a hard error interrupt. Because of the relative priority, the hard error interrupt will interrupt the soft error interrupt flow before it has completed. This will cause the soft error interrupt routine to see all error bits clear. The operating system must provide for such dual interrupts.

Figure 4–17 shows how soft errors are parsed.

Figure 4–17 Soft Error Interrupt Parse Tree



4.9.4 Exceptions

The vector processor generates two kinds of exceptions: memory management exceptions and arithmetic exceptions. Memory management exceptions are dealt with by the operating system, whereas arithmetic exceptions disable the vector processor.

4.9.4.1 Memory Management Exceptions

A memory mangement exception occurs if a fault is detected during the virtual-to-physical translation phase of a memory reference instruction. The SCB vector depends on the fault type.

The memory mangement exception flows are entered by a return status of ERR L on the VIB bus following the read of the Vector Memory Activity Check (VMAC) Register.

NOTE: **Vector memory management exceptions are serviced in the same way as scalar memory management exceptions. See Section 3.3.4 for more on memory management exceptions.**

The vector processor produces precise exceptions for memory management faults. When a memory management exception occurs, microcode and operating system handlers are used to fix the exception.

The vector processor cannot service translation-not-valid and access-control violation faults. To handle these exceptions, the vector processor passes sufficient state back to the scalar processor so that if a memory management fault occurs, then microcode can build a vector exception frame on the stack to permit vector processor memory management exceptions to be handled precisely and restart the faulting instruction.

To enforce synchronous operation, when a vector load/store operation is issued by the scalar processor, the scalar processor will not issue more instructions until the memory management has completed. To reduce this delay from issue of a load/store instruction to issue of the next vector instruction, the load/store unit has logic to predict when memory management can proceed fault free. When the load/store unit knows that it can perform all virtual-to-physical translations without incurring a memory management fault, it issues a signal called MMOK (memory management OK) to the VECTL chip, which then releases the scalar processor to issue more instructions while the load/store unit completes the remainder of the data transfers. With this mechanism, the overhead of providing precise memory management faults is reduced.

4.9.4.2 Vector Arithmetic Exceptions

Vector arithmetic exceptions are called *imprecise*, because the program counter in the scalar processor is pointing farther down the instruction stream and cannot be backed up to point at the failing instruction. In this case to report the condition, the vector processor disables itself so that the scalar processor will take a vector disable fault when it attempts to send a vector instruction. The vector disable fault handler then determines the cause.

Imprecise exceptions are caused by typical arithmetic problems such as divide by zero and underflow. Because the vector processor executes instructions out of order, it is not possible to determine the instruction that caused the exception from the updated PC.

When an imprecise exception occurs, the VECTL chip sets a bit in the register mask in the Vector Arithmetic Exception Register (VAER) to indicate the destination vector register that received data from the exception. It then informs the scalar processor of the exception.

NOTE: To find the precise instruction causing a problem, you can include a SYNC instruction after each arithmetic instruction. Each instruction then completes before the next is attempted. The machine, however, runs much slower than if imprecise exceptions are permitted.

4.9.4.3 Disable Faults

If the scalar processor issues a vector instruction when the vector processor is disabled, it will take a vector module disable fault through SCB vector 68 (hex). It is possible that a disable fault for a hard error interrupt could occur before the interrupt request is serviced. This causes the error to be logged and cleared before the service flow is complete. The operating system must be aware that this sequence is possible.

A disable fault pushes the PC/PSL pair to the stack.

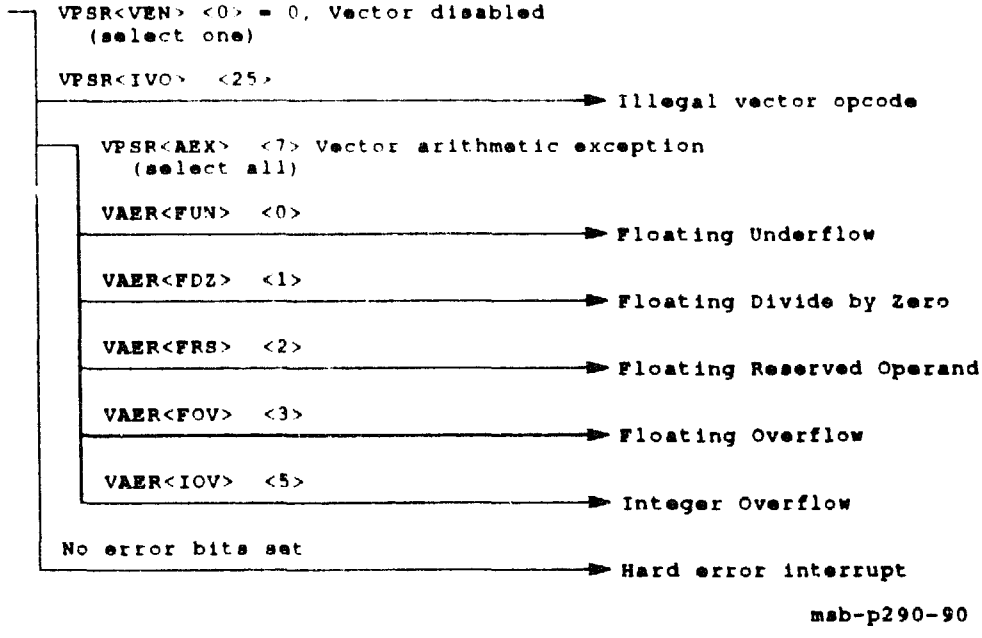
Table 4–6 shows the registers logged for an arithmetic exception.

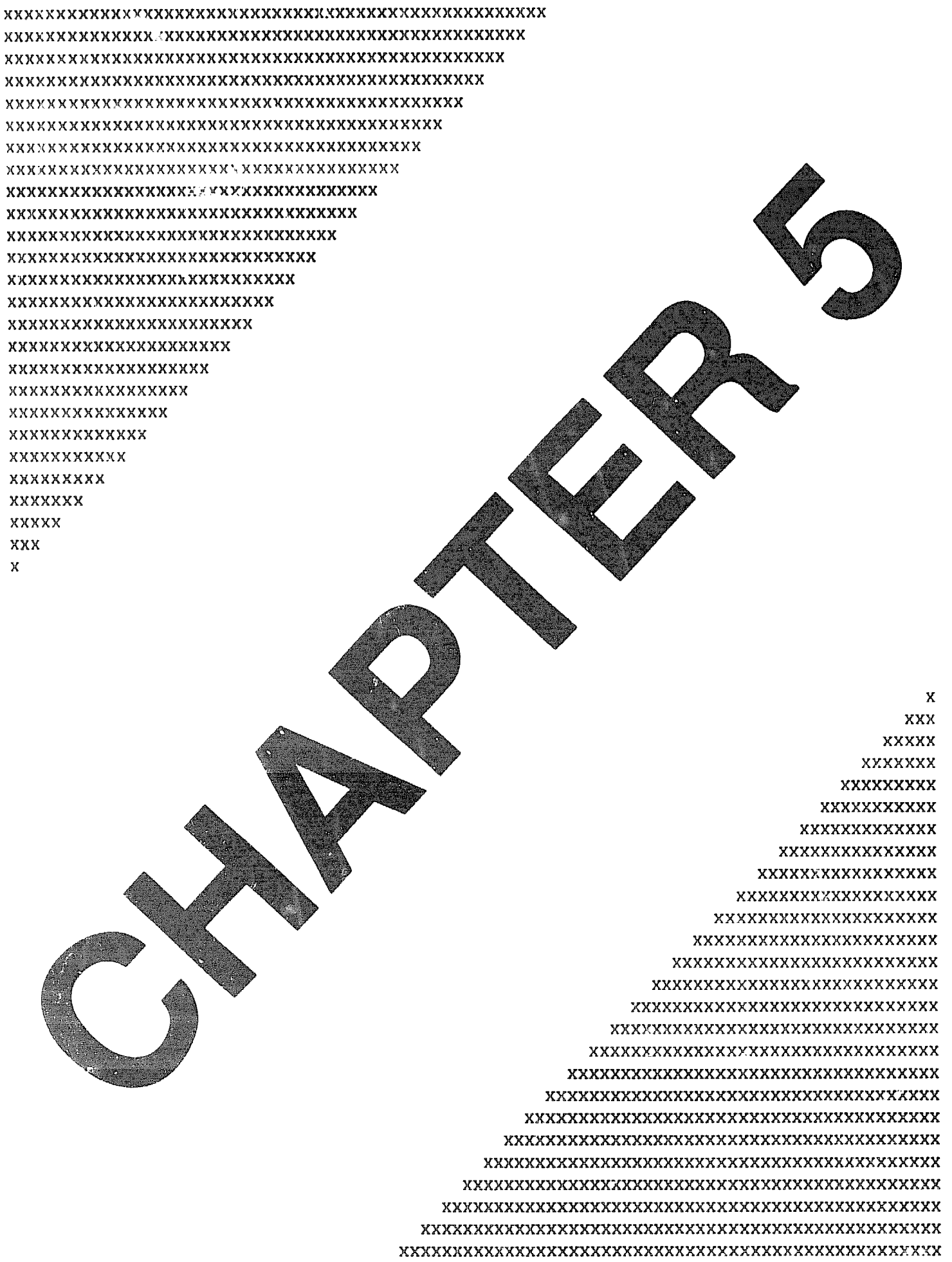
Table 4–6 Disable Faults

Error/Exception	Registers Logged	Description
Arithmetic	VPSR VAER	An arithmetic exception caused the disable fault; generally results in image rundown.

Figure 4–18 shows how disable faults are parsed.

Figure 4–18 Disable Fault Parse Tree




CHAPTER 5

5 MS62A Memory Module

The MS62A memory module is a metal-oxide semiconductor (MOS), dynamic random access memory (DRAM), that provides 32 Mbytes of data storage. The memory array is designed for use in the VAX 6000–400 system and communicates over the XMI bus.

This chapter contains the following sections:

- Features
- Technical Description
- Self-Test and Initialization
- Starting Address and Interleaving
- Control and Status Registers
- Error Handling and Command Responses

5.1 Module Features

The MS62A memory module is a dynamic random access memory (DRAM) that communicates through the XMI bus to provide VAX 6000–400 system memory.

The MS62A memory module has the following features:

- The memory module contains MOS dynamic RAM (DRAM) arrays, a CMOS gate array (that contains error correction code (ECC) logic and control logic), and an XMI interface (the XMI Corner).
- Storage arrays are made up of four banks of 72 DRAMs.
- ECC logic detects single-bit and double-bit errors and corrects single-bit errors.
- Memory self-test checks all RAMS, the data path, and control logic on power-up.
- Quadwords, octawords, and hexwords can be read from memory.
- Quadwords and octawords can be written to memory.
- The memory can be configured by the system for 1-, 2-, 4-, 8-way or no interleaving.

5.2 Technical Description

The MS62A memory module uses XMA logic, DRAM arrays, and a PROM to provide 32 Mbytes of memory to the VAX 6000–400 system.

The MS62A memory module consists of the following major components:

- XMI Corner
- XMA gate array
- Address and control logic
- DRAMs

The **XMI Corner** is the module's interface to the XMI bus and contains CMOS gate arrays and interface logic. Its primary purpose is to transfer data between the MS62A memory module and the XMI bus.

The **XMA gate array** transfers data between the XMI Corner and the DRAMs. The gate array also controls address multiplexing, command decoding, arbitration, and CSR logic functions.

Address and control logic modifies address bits received from the XMI Corner. These modified address bits are used to control the selection of the DRAMs during reading and writing.

All power for the XMI memory array is supplied from +5VBB. If the optional battery backup unit (BBU) is not installed and VAX 6000–400 system power is lost, memory is lost as well.

If the optional BBU is installed, it takes over, ensuring that no data is lost during the power interruption. The BBU supplies power to memory for approximately 10 minutes.

5.3 Self-Test and Initialization

The MS62A memory module performs an initialization and self-test sequence on a cold power-up or when the sequence is requested by a console command.

During a cold power-up the gate array chip is initialized, all memory locations are tested, and the control and status registers are initialized.

A warm power-up occurs when the system, excluding memory if a battery backup unit (BBU) is present, loses power. During a warm power-up, self-test is not run and memory contents are unmodified. However, any data in the data path is lost.

Memory self-test may take up to 60 seconds to run. While self-test runs, the Fault light on the system front panel is on. When self-test completes, the Fault light goes off and the console printout of self-test begins. For details on the self-test console printout, refer to the *VAX 6000–400 Owner's Manual*.

5.4 Starting Address and Interleaving

On power-up the VAX 6000–400 console firmware loads the Starting and Ending Address Register (SEADR) with the starting address, the interleave mode, and the ending address. The following paragraphs describe how to set the SEADR for proper system operation. Section 5.5 gives a description of the SEADR.

5.4.1 Starting and Ending Addresses

The memory responds to starting addresses on any 2-Mbyte boundary. The ending address is also on any 2-Mbyte boundary. The ending address must be greater than the starting address to ensure that data will not be overwritten. The ending address minus the starting address must be equal to or less than the memory size multiplied by the number of ways interleaved.

EA—SA = Memory Size * (# of ways interleaved)

Starting addresses for memory can be in the range from 0 to 510 Mbytes and ending addresses in the range from 0 to 512 Mbytes. Ending addresses greater than 512 Mbytes are not permitted. The area above 512 Mbytes is I/O space.

5.4.2 Interleaving

Interleaving achieves greater throughput to memory by optimizing memory access time and increasing the effective memory transfer rate. This is done by operating memory modules in parallel.

The memory array supports 1-way (no interleaving), 2-way, 4-way, or 8-way interleaving at the system level. Up to eight memory array modules can be interleaved. Interleaving is done on hexword boundaries.

5.5 Control and Status Registers

The CSR names and their relative addresses are shown in Table 5–1. Descriptions of the CSRs are also included in this section.

Table 5–1 MS62A Memory Module Control and Status Registers

CSR Name	Mnemonic	Address
Device Register	XDEV	BB[1] + 0000 0000
Bus Error Register	XBER	BB + 0000 0004
Starting and Ending Address Register	SEADR	BB + 0000 0010
Memory Control Register 1	MCTL1	BB + 0000 0014
Memory ECC Error Register	MECER	BB + 0000 0018
Memory ECC Error Address Register	MECEA	BB + 0000 001C
Memory Control Register 2	MCTL2	BB + 0000 0030
TCY Register	TCY	BB + 0000 0034
Interlock Flag Status Registers	IFLG*n*	BB + 0000 000*n*[2]

[1] "BB" refers to the base address of an XMI node (2180 0000 + (node ID x 8000)).

[2] Refer to the Interlock Flag Status Register description for the relative address of the Interlock Flag Status Registers.

The memory contains 24 control and status registers (CSRs) to control the memory and log errors. All CSRs are 32 bits long and respond only to longword read and write transactions. When writing to the CSRs, only full writes are performed. If a parity error occurs during a write operation, the operation is aborted and the contents of the CSRs are unchanged.

Some bits in the registers are cleared on power-up, while others need a one written to them to clear. Only the Interlock Flag Status Registers initialize on a warm start.

The CSRs start at an address dependent upon the node ID. All CSR addresses are designated as BB + *n*, where *n* is the relative offset of the register.

The following definitions apply to the descriptions of the control and status registers.

CRD error – A correctable single-bit error.

RDS error – An uncorrectable double-bit error that occurs when the syndrome bits represent an unused ECC code.

RER error – A general uncorrectable double-bit error indicator that includes an RDS error, a row parity error, a column parity error, or a byte write error.

RO – Indicates a read-only register.

RO, 0 – Indicates a read-only register, cleared on power-up.

R/W – Indicates a read and write register.

R/W, 0 – Indicates a read and write register, cleared on power-up.

R/W1C – Indicates a read and write register, write a one to clear.

R/W1C, 0 – Indicates a read and write register, write a one to clear, and cleared on power-up.

R/W1C, 1 – Indicates a read and write register, write a one to clear, and set on power-up.

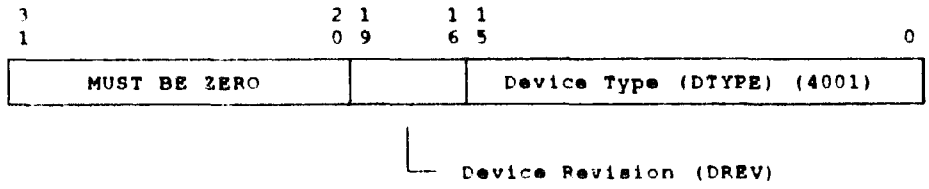
W/O, 0 – Indicates a write only register, cleared on power-up.

Device Register (XDEV)

The Device Register contains information to identify the MS62A memory module. Both fields are loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS

Nodespace base address + 00000 0000



bits<31:20>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<19:16>

Name: Device Revision

Mnemonic: DREV

Type: RO

Identifies the revision level of the MS62A memory module. The use of the Device Revision field is implementation dependent. The field does not indicate the hardware revison level, only the functional level.

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

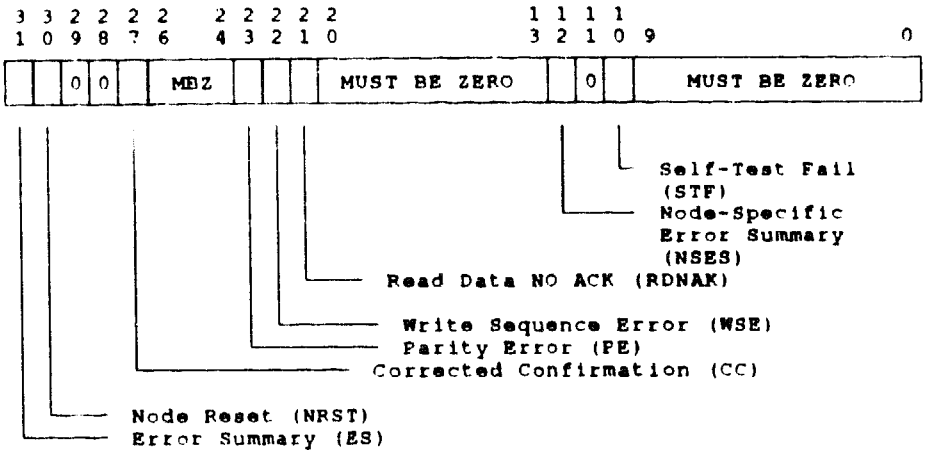
Type: RO

Identifies the type of node. The device type for an MS62A memory module is 4001 (hex). This value is hardwired in the Device Register

Bus Error Register (XBER)

The Bus Error Register records error and status information about the XMI bus.

ADDRESS *Nodespace base address + 0000 0004*



bit<31>

Name: Error Summary

Mnemonic: ES

Type: RO, 0

This bit state represents the logical OR of the error bits in this register.

bit<30>

Name: Node Reset

Mnemonic: NRST

Type: W/O, 0

Writing a one to this location initiates a complete node reset, including self-test.

bits<29:28>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<27>

Name: Corrected Confirmation
Mnemonic: CC
Type: R/W1C, 0

This bit is set when the XMI Corner interface (XCI) bus detects a single-bit error on the XMI CNF bits.

bits<26:24>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<23>

Name: Parity Error
Mnemonic: PE
Type: R/W1C, 0

This bit is set when the node detects a parity error on an XMI cycle

bit<22>

Name: Write Sequence Error
Mnemonic: WSE
Type: R/W1C, 0

When set, indicates that the node aborted a write transaction due to one or more missing data cycles.

bit<21>

Name: Read Data NO ACK
Mnemonic: RDNAK
Type: R/W1C, 0

When set, indicates that the node received a NO ACK confirmation for a data cycle it transmitted.

bits<20:13>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<12>

Name: Node-Specific Error Summary
Mnemonic: NSES
Type: RO, 0

When set, this bit indicates that a node-specific error condition has been detected. The exact nature of the error is located in the memory error status registers.

bit<11>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<10>

Name: Self-Test Fail
Mnemonic: STF
Type: R/W1C, 1

While set, this bit indicates that the node has not yet passed its self-test. This bit is cleared when self-test successfully completes. This bit also drives XMI BAD (an XMI bus signal that reports node failures). Clearing this bit also clears XMI BAD.

bits<9:0>

Name: Reserved
Mnemonic: None
Type: RO

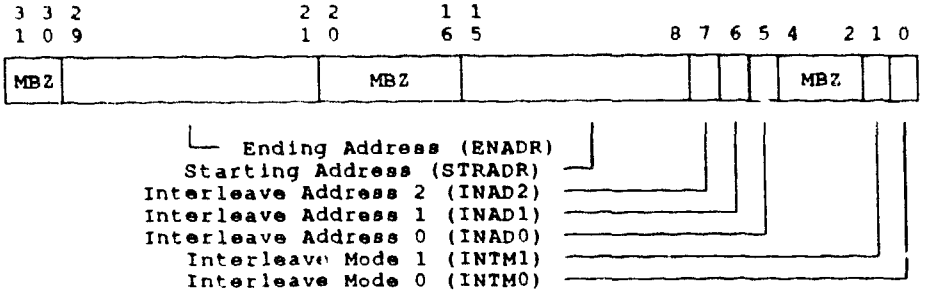
Reserved; must be zero.

Starting and Ending Address Register (SEADR)

The Starting and Ending Address Register contains the memory starting and ending addresses. See Section 5.4.1 for a description of the rules that must be followed when setting these addresses. This register also sets the interleave mode.

ADDRESS

Nodespace base address + 0000 0010



bits<31:30>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bits<29:21>

Name: Ending Address

Mnemonic: ENDADR

Type: R/W, 0

The Ending Address for the memory on 2-Mbyte boundaries. The memory is enabled if the ending address is greater than the starting address. The ending address range is from 0 to (510 Mbytes + 2 Mbytes).

bits<20:16>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<15:8>

Name: Starting Address
Mnemonic: STRADR
Type: R/W, 0

The Starting Address for the memory on 2-Mbyte boundaries. The starting address range is from 0 to 510 Mbytes.

bits<7:5>

Name: Interleave Address
Mnemonic: INADn
Type: R/W, 0

The address field used for interleaving. This address determines to what address the module will respond.

bits<4:2>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<1:0>

Name: Interleave Mode
Mnemonic: INTLMn
Type: R/W, 0

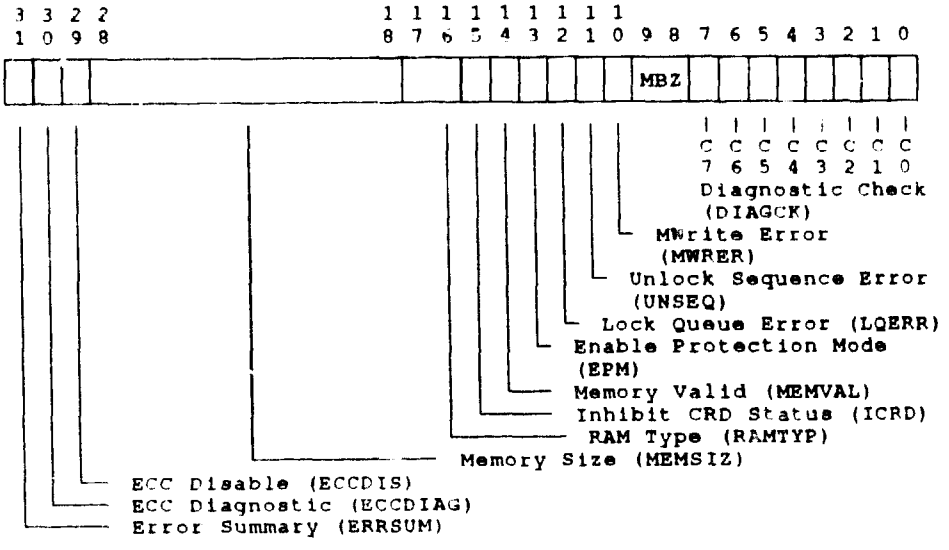
This field shows how many ways the module is being interleaved and is used to determine the addresses that the module will respond to.

Memory Control Register 1 (MCTL1)

The Memory Control Register 1 along with the Memory Control Register 2 contains memory-specific control, status, and error bits. The MCTL1 Register also controls the diagnostic modes of the memory module.

ADDRESS

Nodespace base address + 0000 0014



bit<31>

Name: Error Summary

Mnemonic: ERRSUM

Type: RO

This bit contains the ORed sum of error bits in MCTL1, MCTL2, and Memory ECC Error Registers.

bit<30>

Name: ECC Diagnostic
Mnemonic: ECCDIAG
Type: R/W, 0

This bit is used for diagnostic purposes.

bit<29>

Name: ECC Disable
Mnemonic: ECCDIS
Type: R/W, 0

This bit is used for diagnostic purposes.

bits<28:18>

Name: Memory Size
Mnemonic: MEMSIZ
Type: RO

This field contains the memory module size in 256-Kbyte increments, where 00000011000=6 Mbytes, 00000100000=8 Mbytes, and 00010000000=32 Mbytes.

bits<17:16>

Name: RAM Type
Mnemonic: RAMTYP
Type: RO

This field contains the size of the RAM.

bit<15>

Name: Inhibit CRD Status
Mnemonic: ICRD
Type: R/W, 0

This bit inhibits the reporting of CRD status to the commander on read cycles. When this bit is set, any CRD response is changed to a GRD response. The CRD errors are logged in the MECEA and MECER registers. RER errors are logged and reported normally.

bit<14>

Name: Memory Valid
Mnemonic: MEMVAL
Type: RO, 0

This bit indicates that valid data is stored in memory. The bit is set on the first write to the module memory space.

bit<13>

Name: Enable Protection Mode
Mnemonic: EPM
Type: R/W, 0

When this bit is set, the operation of the ECC Diagnostic<30> and ECC Disable <29> bits are inhibited in the first 2 Mbytes of memory space, starting address to starting address plus 2 Mbytes.

bit<12>

Name: Lock Queue Error
Mnemonic: LQERR
Type: R/W1C, 0

This bit is set if a data word is sent as a response to an Interlock Read and no lock is pending in the memory.

bit<11>

Name: Unlock Sequence Error
Mnemonic: UNSEQ
Type: R/W1C, 0

This bit is set if an Unlock Write transaction is accepted and no corresponding matching location is marked as locked. Either an Interlock Read was never performed to this location, the lock did not set, or the lock might have been cleared by another source.

bit<10>

Name: MWrite Error
Mnemonic: MWRER
Type: R/W1C, 0

This bit is set on an RDS error during a partial write cycle.

bits<9:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<7:0>

Name: Diagnostic Check
Mnemonic: DIAGCK
Type: R/W, 0

This field is used during ECC diagnostic mode as substitute check bits.

Memory ECC Error Register (MECER)

The Memory ECC Error Register logs ECC error status. The MECER also logs uncorrectable error codes for row parity error, column parity errors, and byte write errors. The MECER logs ECC error information during read cycles only. If an RER error occurs during a Write Mask cycle, the MWRER bit in the MCTL1 Register is set.

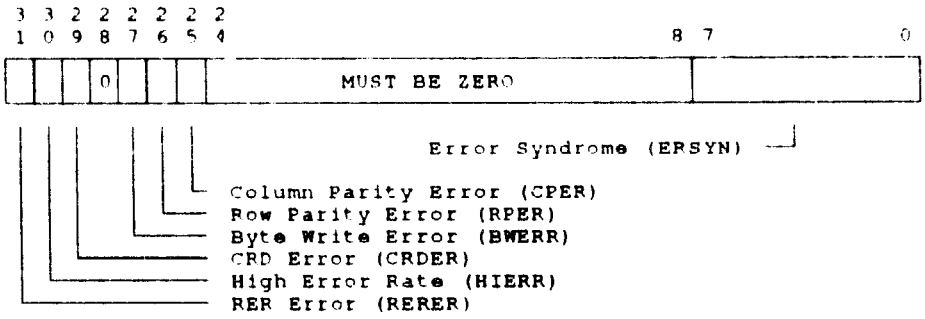
This register logs ECC error type and error syndrome information when correctable and uncorrectable errors occur during Read transactions. During a Write Mask transaction, only the MWRER bit logs the fact that the ECC error occurred.

For read accesses, the register logs the first correctable error and holds it until either an uncorrectable error occurs or the error is cleared. Additional correctable errors are only reported and are not logged. An uncorrectable error will overwrite a logged correctable error. A correctable error will not overwrite a logged uncorrectable error or a previously logged correctable error until the error has been cleared.

This register logs errors during module self-test.

ADDRESS

Nodespace base address + 0000 0018



bit<31>

Name: Uncorrectable Double-Bit (RER) Error

Mnemonic: RERER

Type: R/W1C, 0

This bit indicates that an uncorrectable error occurred during a read transaction. The Error Address and Error Syndrome are valid for the uncorrectable double-bit error. If the Column Parity Error bit, the Row Parity Error bit, and the Byte Write Error bit are not all set, then the uncorrectable double-bit error is an RDS error.

bit<30>

Name: High Error Rate
Mnemonic: HIERR
Type: R/W1C, 0

This bit indicates that another error, RER or CRD, occurred before the previous one was cleared from the register.

bit<29>

Name: CRD Error
Mnemonic: CRDER
Type: R/W1C, 0

This bit indicates that a CRD error occurred during a read transaction. This includes a single-bit error in the check bits, even though no correction is done on the data bits. The error address and error syndrome are valid if no RER error log exists.

bit<28>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<27>

Name: Byte Write Error
Mnemonic: BWERR
Type: RO, 0

This bit indicates that the RER error was due to reading a location that was marked bad during a partial write cycle that had previously detected an RER error. Cleared when MECER<31> is cleared.

bit<26>

Name: Row Parity Error
Mnemonic: RPER
Type: RO, 0

This bit indicates that the RER error is due to a row address parity error. Cleared when MECER<31> is cleared.

bit<25>

Name: Column Parity Error
Mnemonic: CPER
Type: RO, 0

This bit indicates that the RER error is due to a column address parity error. Cleared when MECER<31> is cleared.

bits<24:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<7:0>

Name: Error Syndrome
Mnemonic: ERSYN
Type: RO, 0

This field is the syndrome bits of the location in an RER or CRD error.

Memory ECC Error Address Register (MECEA)

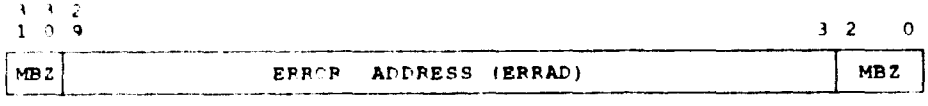
The Memory ECC Error Address Register logs the address of correctable and uncorrectable errors logged in the Memory ECC Error Register.

For read accesses, this register logs the address of the first corrected read data (CRD) error and holds it until a double-bit uncorrectable error (RER) occurs or the error is cleared. An RER error causes a logged CRD error address to be overwritten. A CRD will not overwrite a logged RER error address. If multiple RER errors occur, only the first error address is logged.

This register logs errors during self-test.

ADDRESS

Nodespace base address + 0000 001C



bits<31:30>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero

bits<29:3>

Name: Error Address

Mnemonic: ERRAD

Type: RO, 0

The error address of the RER or CRD error logged in the Memory ECC Error Register. This register is valid only if the RER or CRD Error log bits are set in the Memory ECC Error Register. This address is the bus address of the cycle that was being performed at the time of the error.

bits<2:0>

Name: Reserved

Mnemonic: None

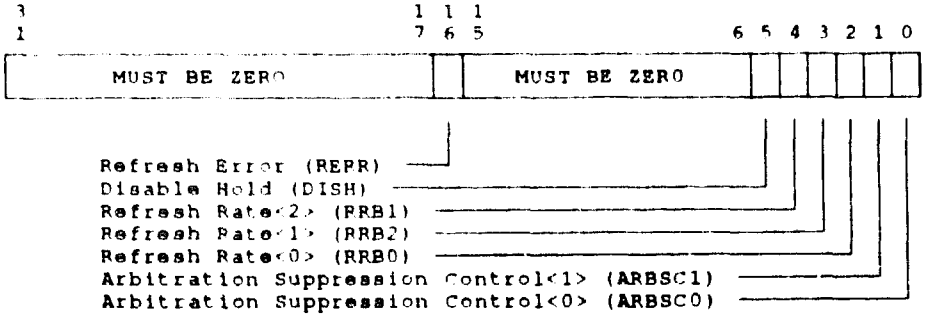
Type: RO

Reserved; must be zero.

Memory Control Register 2 (MCTL2)

The second memory control register contains additional control and error status information.

ADDRESS *Nodespace base address + 0000 0030*



bits<31:17>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bit<16>

Name: Refresh Error

Mnemonic: RERR

Type: R/W1C, 0

This bit is set if a refresh request is set, and a second refresh request is asserted before the first one is implemented, meaning that a refresh was missed.

bits<15:6>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bit<5>

Name: Disable Hold
Mnemonic: DISH
Type: R/W, 0

This bit is used by memory arbitration logic to disable the use of XMI HOLD L.

bits<4:2>

Name: Refresh Rate
Mnemonic: RRB
Type: R/W

This field controls the module's DRAM refresh rate.

bits<1:0>

Name: Arbitration Supression Control
Mnemonic: ARBSCn
Type: R/W, 0

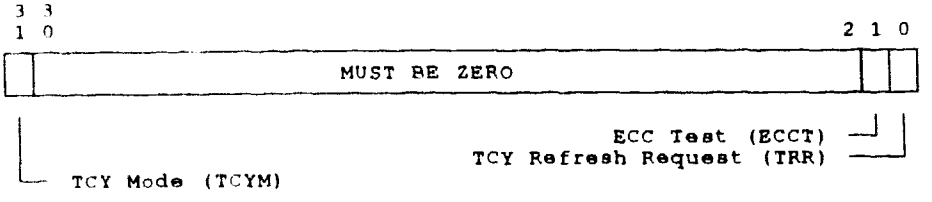
This field controls the Arbitration Supression mode.

TCY Tester Register (TCY)

The TCY Tester Register contains control bits to implement manufacturing tests.

ADDRESS

Nodespace base address + 0000 0034

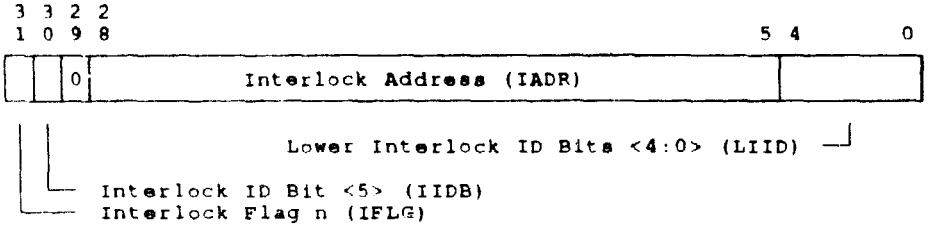


Interlock Flag Register (IFLG*n*)

The Interlock Flag *n* Register (IFLG*n*) (where *n* is 0 to 15) holds the address and ID of the last interlock flag only if all lower interlock flags are set. The locations of IFLG*n* flags are shown in the relative address table.

ADDRESS

Nodespace base address + (relative address)



where n is the number of the Interlock Flag Register Number (0-15)

Interlock Flag Register	Relative Address
Interlock Flag 0 Status Register	BB + 20
Interlock Flag 1 Status Register	BB + 24
Interlock Flag 2 Status Register	BB + 28
Interlock Flag 3 Status Register	BB + 2C
Interlock Flag 4 Status Register	BB + 40
Interlock Flag 5 Status Register	BB + 44
Interlock Flag 6 Status Register	BB + 48
Interlock Flag 7 Status Register	BB + 4C
Interlock Flag 8 Status Register	BB + 80
Interlock Flag 9 Status Register	BB + 84
Interlock Flag 10 Status Register	BB + 88
Interlock Flag 11 Status Register	BB + 8C
Interlock Flag 12 Status Register	BB + 100
Interlock Flag 13 Status Register	BB + 104
Interlock Flag 14 Status Register	BB + 108
Interlock Flag 15 Status Register	BB + 10C

bit<31>

Name: Interlock Flag *n*
Mnemonic: IFLGn
Type: R/W1C, 0

This bit is Interlock Flag *n*, where n = (0-15). If asserted, the Interlock Address and Interlock ID are valid and the lock is set. The lock cannot be set by writing directly to IFLGn. Writing a one to IFLG*n* clears the lock.

bit<30>

Name: Interlock ID Bit <5>
Mnemonic: IIDB
Type: RO, 0

IIDB is the most significant commander ID bit of the Interlock Read transaction. This bit is valid only if Interlock Flag *n* is set.

bit<29>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<28:5>

Name: Interlock Address
Mnemonic: IADR
Type: RO, 0

IADR gives the address of the Interlock Read transaction. It is valid only if Interlock Flag *n* is set.

bits<4:0>

Name: Lower Interlock ID Bits <4:0>
Mnemonic: LIID
Type: RO, 0

LIID are the lower five commander ID bits of the Interlock Read transaction. This field is valid only if Interlock Flag *n* is set.

5.6 Error Handling and Command Responses

The following paragraphs describe how the memory responds to an error condition. The memory performs single-bit correction and double-bit detection on the data stored.

5.6.1 Read Errors

If no errors occur during a read operation, a Good Read Data (GRDn) function code is returned with the data. If a correctable error occurs during the read operation, a Corrected Read Data (CRDn) function code is returned. If an uncorrectable error occurs, a Read Error Response (RER) is returned in place of the data.

The lock bit is not set if:

- An RER error occurs during an Interlock Read transaction.
- The confirmation of Interlock Read data is missing or bad.

A locked response is sent if:

- The address of an Interlock Read transaction matches a locked hexword.
- All locks are set and memory receives an Interlock Read request.

5.6.2 Full Write Errors

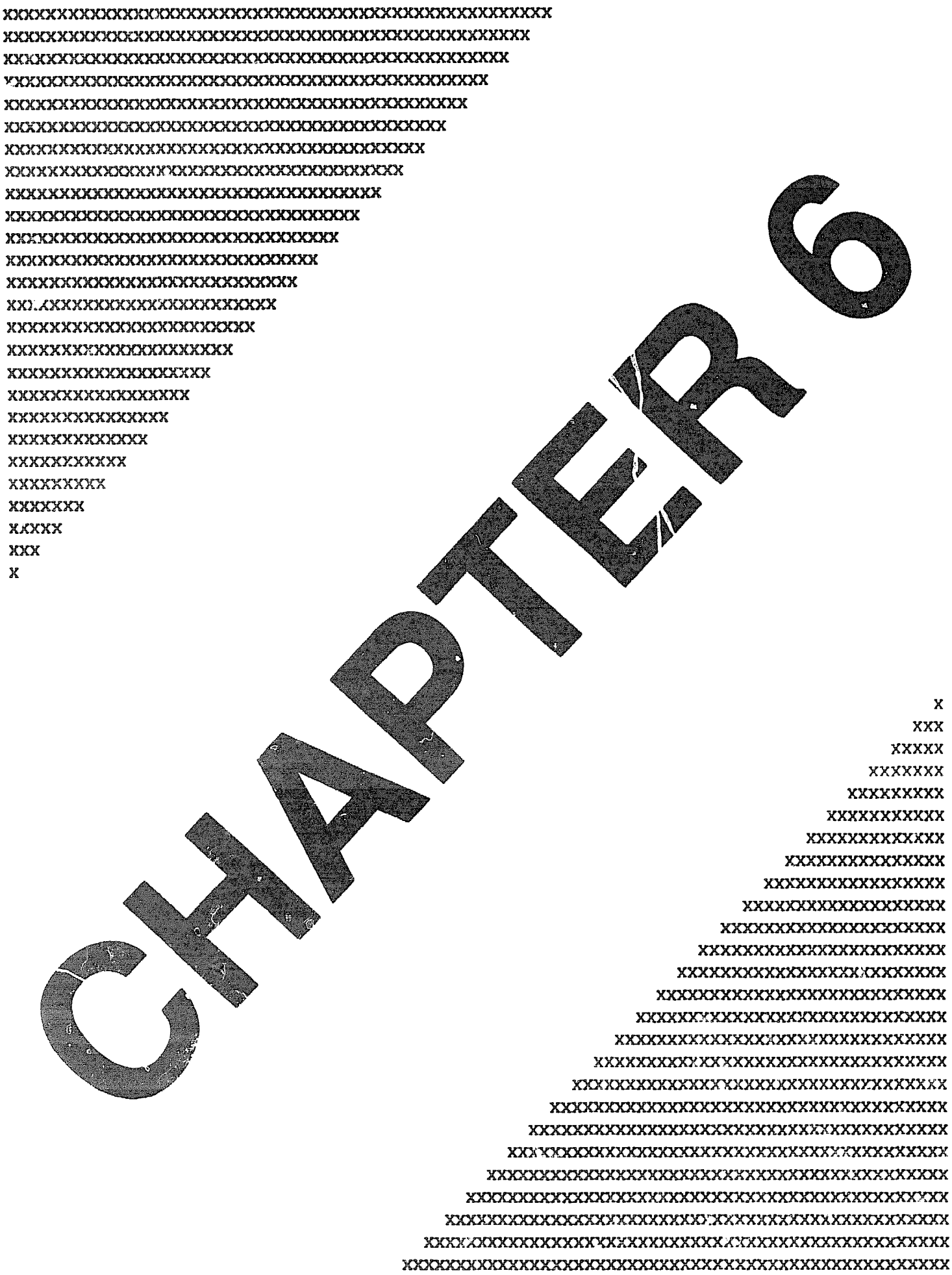
A full write is performed on a quadword or octaword, dependent on the number of mask bits that are set. If mask bits <47:32> are set, an octaword write transaction takes place. If mask bits <39:32> are set, a quadword write transaction takes place. Write data is written into memory with the generated ECC check bits. The write transaction does not begin until all the write data is received from the XMI bus and checked for parity.

If an XMI parity error occurs on one or more quadwords of received data, the write will not begin and a NO ACK response is returned.

5.6.3 Partial Write Errors

If the mask bits for a quadword or octaword are not all set, a partial write is performed. After write data is merged with read data, the write data is written into memory. If the read data is correct, the write is completed. If a correctable read error occurs, the write continues to completion with the corrected data. Uncorrectable read data causes the old data to be rewritten with a Byte Write Error ECC code to mark the location defective. If the cycle is an Unlock Write cycle, an uncorrectable error causes the location to be marked bad and the interlock flag cleared.

If an XMI parity error occurs on one or more quadwords of received data, the write does not begin. If the parity error occurs during an Unlock Write command or data cycle, the lock is not reset.


CHAPTER 6

6 DWMBA XMI-to-VAXBI Adapter

The DWMBA XMI-to-VAXBI adapter provides an information path between the XMI bus and I/O devices on the VAXBI bus.

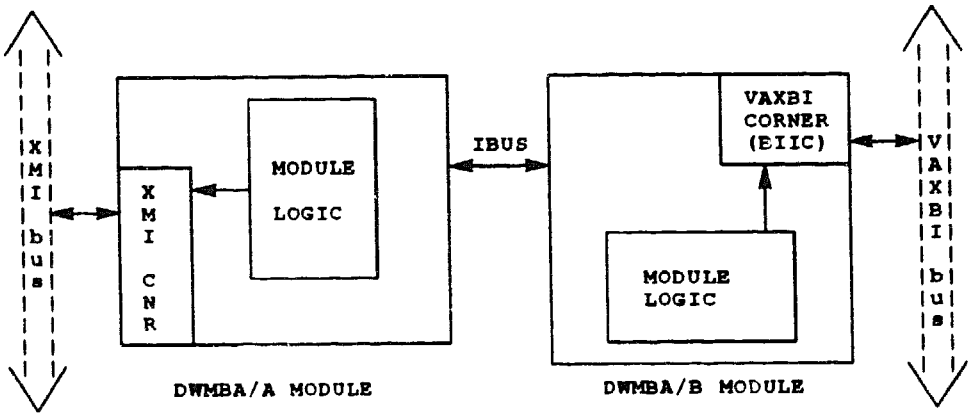
This chapter contains the following sections:

- DWMBA Overview
- CPU Transactions
- DMA Transactions
- DWMBA Registers
- Interrupts
- Error Reporting
- DWMBA Initialization, Self-Test, and Booting

6.1 DWMBA Overview

The DWMBA XMI-to-VAXBI adapter provides an information path between the XMI bus and I/O devices on the VAXBI bus. The DWMBA consists of two modules: the DWMBA/A XMI module and the DWMBA/B VAXBI module. The IBUS connects the two modules.

Figure 6–1 DWMBA XMI-to-VAXBI Adapter Block Diagram



The DWMBA/A module contains an XMI Corner, register files, XMI required registers, DWMBA-specific registers, and control sequencers for the XMI interface.

The DWMBA/B module contains a BIIC, interconnect drivers, control sequencers to handle the control of the data transfer, status bits to/from the DWMBA/A module's register files and the BIIC, DWMBA/B module specific registers, decode logic for DMA operations, and VAXBI clock generation circuitry.

These two modules are connected by four cables of 30 wires each. The 120 wires make up the IBUS, which transfers data and control information between the two modules.

The DWMBA uses CPU and DMA transactions to exchange information. CPU transactions originate from the KA64A CPU module(s) and are presented to the DWMBA from the XMI bus with the CPU as the XMI commander and the DWMBA as the XMI responder.

DMA transactions originate from VAXBI nodes that select the DWMBA as the VAXBI slave. These are read or write transactions targeted to XMI memory space or are VAXBI-generated interrupt transactions that target a KA64A CPU module. For DMA transactions, the DWMBA is the XMI commander and the MS62A memory module is the XMI responder.

Write transactions, whether DMA or CPU, are always disconnected. This means that as soon as either the CPU or the VAXBI master issues the write, it waits for an ACK confirmation that the command and write data was accepted but not necessarily completed at the destination. If the write fails, a write error IVINTR is returned.

The VAX 6000–400 system uses a 30-bit physical address. Chapter 2 describes the XMI address space. The *VAXBI Options Handbook* describes the VAXBI address space. The DWMBA can be both a master and a slave on the VAXBI. As a master, it carries out transactions requested by its XMI devices. As a slave, it responds to VAXBI transactions that select its node.

6.2 CPU Transactions

The DWMBA XMI-to-VAXBI adapter translates XMI transactions into equivalent VAXBI transactions. Regardless of whether the transaction is a read, write, or IDENT, software need not concern itself with the details, as the XMI transaction behaves as it would if it were directed to memory or other XMI devices.

Table 6–1 XMI-to-VAXBI Command Translations

XMI	VAXBI
Longword Read	Longword Read
Quadword Read	Illegal
Octaword Read	Illegal
Hexword Read	Illegal
Longword Interlock Read	Longword Interlock Read (iRCI)
Quadword Interlock Read	Illegal
Octaword Interlock Read	Illegal
Hexword Interlock Read	Illegal
Longword Mask Write	Longword Write Mask (WMCI)
Quadword Mask Write	Illegal
Octaword Unlock Write Mask	Illegal
Longword Unlock Write Mask	Longword Unlock Write Mask (UWMCI)
Quadword Unlock Write Mask	Illegal
Octaword Unlock Write Mask	Illegal
Interrupt Request (INTR)	Illegal
Indentify (IDENT)	IDENT
Implied Vector Interrupt (IVINTR)	Illegal

6.2.1 General Operation

The DWMBA responds to XMI longword transactions. When an XMI commander issues a Read, Interlock Read, Write Mask, Unlock Write Mask, or IDENT targeting the DWMBA, the XMI commander arbitrates for the XMI bus, wins the bus, sends out the function, command, address, ID, and parity. The targeted DWMBA recognizes its ID and returns ACK or NO ACK (for busy, an error, or illegal transaction). Once the DWMBA accepts a CPU transaction from an XMI commander, it asserts the NO ACK confirmation code to all subsequent XMI commanders that attempt a CPU transaction until the current transaction completes.

For Read transactions, the DWMBA decodes the XMI command and determines if the address references VAXBI I/O space or a DWMBA register. If VAXBI address space is referenced, the DWMBA generates a VAXBI Read transaction and waits for the return of read data from the VAXBI. Upon receiving the read data from either the VAXBI or a DWMBA register, the DWMBA arbitrates for the XMI bus as a responder and returns the requested data to the commander. The XMI commander sends confirmation of the receipt of data back to the DWMBA. If the Read fails, the XMI commander retries the Read

Interlock Read transactions are handled the same as Reads except:

- DWMBA registers do not support Interlock Reads and handle them the same as Reads.
- If the Interlock Read command that targets the VAXBI bus gets a RETRY CNF from the VAXBI, the DWMBA returns the Locked Response back to the XMI commander.

Write transactions to the VAXBI are disconnected. The CPU continues on after the DWMBA/A ACKs the Mask Write and Unlock Write Mask transaction if the command/address (C/A) and data received from the XMI bus is error free. The DWMBA decodes the XMI command and determines if the address references VAXBI I/O space or a DWMBA register. If VAXBI address space is referenced, the DWMBA generates the corresponding VAXBI write transaction. If a DWMBA register is referenced, it is written with the write data. Write errors cause write error IVINTRs to be returned to the CPU.

6.2.2 VAXBI I/O Space Reads

The two XMI read transactions are Read and Interlock Read. The XMI Interlock Read is translated to a VAXBI IRCI transaction while the XMI Read is translated to a VAXBI Read transaction.

The length of the generated VAXBI transaction must be a longword (D<31:30> = 01 in the VAXBI command/address cycle). XMI address bits<28:25> are forced to zero to map XMI addresses to VAXBI addresses and passed onto the VAXBI. The DWMBA ignores with a NO ACK confirmation any targeted transaction longer than a longword.

If the VAXBI issues a RETRY on an XMI Interlock Read request to VAXBI I/O address space due to the resource being locked by a previous Interlock Read request, the DWMBA issues a Locked Response to the XMI commander.

6.2.3 VAXBI I/O Space Writes

The two XMI writes are Mask Write and Unlock Write Mask. The Mask Write is translated to a VAXBI Write Mask with Cache Intent (WMCI), while the Unlock Write Mask is translated to a VAXBI Unlock Write Mask with Cache Intent (UWMCI).

The length of the generated VAXBI transaction must be a longword (D<31:30> = 01 in the VAXBI command/address cycle). XMI address bits<28:25> are forced to zero and passed onto the VAXBI. The DWMBA ignores with a NO ACK confirmation any targeted XMI transaction longer than a longword. The DWMBA supports interlocked instructions even though the KA64A CPU module never issues interlocked instructions to I/O space.

6.2.4 Interrupts

6.2.4.1 XMI IDENT to VAXBI IDENT

When an XMI CPU issues an XMI IDENT, the DWMBA issues a VAXBI IDENT if the DWMBA does not have a pending interrupt at the IDENT level. The DWMBA/B module fetches the IDENT command from the DWMBA/A module's register file and clears the corresponding level and interrupt sent flip-flops that were previously set by the VAXBI-initiated interrupt, providing that no IBUS parity errors are detected.

The DWMBA/B module writes the received vector data into the CPU read data buffer and notifies the DWMBA/A module that the vector is available. The DWMBA/A module then issues an IDENT response cycle on the XMI (with a Good Read Data response where the function code = 100 and the vector is in bits<15:2>).

6.2.4.2 XMI IDENT with DWMBA Adapter Pending Interrupt

If an XMI IDENT is decoded with an IPL matched by the DWMBA/B module while the DWMBA's interrupt-pending flip-flop is set, the interrupt vector of the DWMBA is issued to the XMI. The IDENT clears both the IPL level 17 sent flip-flop and the DWMBA interrupt-pending flip-flop. The corresponding level 17 VAXBI interrupt-pending flip-flop, if also set, is not cleared, resulting in the DWMBA issuing another XMI INTR transaction.

6.2.4.3 Passive Release of VAXBI Interrupts

If the requesting VAXBI node aborts its interrupt request before the XMI CPU generates an IDENT transaction at that level, the resulting IDENT on the VAXBI gets NO ACKed. The DWMBA then issues a Read Error Response (RER) to the XMI commander.

If an XMI CPU issues an IDENT to the VAXBI and the DWMBA has no pending flip-flops set, the DWMBA issues the IDENT to the VAXBI. The resulting IDENT on the VAXBI gets NO ACKed. The DWMBA then issues a Read Error Response (RER) to the XMI commander and sets the IDENT Error bit in the DWMBA/B module's Error Summary Register (BESR<1>).

6.3 DMA Transactions

The DWMBA XMI-to-VAXBI adapter translates a VAXBI transaction into an XMI bus transaction when a VAXBI node selects the DWMBA as the slave node for a VAXBI transaction. The XMI bus transaction is serviced by a memory node, and the requested data is then read from or written to XMI memory.

Table 6–2 VAXBI-to-XMI Command Translations

VAXBI	XMI
Read	Read
Interlock Read with Cache Intent	Interlock Read
Read with Cache Intent	Read
Write (LW)	Write Mask on the unused longword within the XMI quadword
Write (QW)	Write Mask (QW)
Write (OW)	Write Mask (OW)
Write with Cache Intent (LW)	Write Mask on the unused longword within the XMI quadword
Write with Cache Intent (QW)	Write Mask
Write with Cache Intent (OW)	Write Mask
Unlock Write Mask with Cache Intent (LW)	Unlock Write Mask
Unlock Write Mask with Cache Intent (QW)	Unlock Write Mask
Unlock Write Mask with Cache Intent (OW)	Unlock Write Mask
Write Mask with Cache Intent (LW)	Write Mask on the unused longword within the XMI quadword
Write Mask with Cache Intent (QW)	Write Mask (QW)
Write Mask with Cache Intent (OW)	Write Mask (OW)

Table 6–2 (Cont.) VAXBI-to-XMI Command Translations

VAXBI	XMI
Interrupt (INTR)	Interrupt
Identify (IDENT)	Not supported (NO ACK to VAXBI)[1]
Invalidate (INVAL)	Not supported (NO ACK to VAXBI)
Broadcast (BDCST)	Not supported (NO ACK to VAXBI)
Interprocessor Interrupt (IPINTR)[2]	Interrupt at IPL 16
Stop	Not supported (NO ACK to VAXBI)

[1]The DWMBA does not process VAXBI IDENTs onto the XMI bus but the DWMBA's BIIC responds to VAXBI IDENTs that are directed to it if:

- The BIIC detects an error condition that results in a generated interrupt.
- The user sets the force interrupt bits in the appropriate BIIC register.
- External logic such as the IPINTR decode logic asserts the BCI INT signal (pins<7:4> on the BIIC).

[2]See Section 6.3.4.

A VAXBI transaction can reference an address between the addresses in the Starting and Ending Address Registers in the DWMBA's BIIC. VAXBI transactions cannot access DWMBA-specific registers.

6.3.1 VAXBI-to-XMI Memory Space Reads

The DWMBA has two sets of register files, DMA-A and DMA-B, which allows the DWMBA to accept either a second VAXBI write transaction or a VAXBI read transaction before the previous XMI write completes. The DWMBA performs the operations on the XMI in the order that the VAXBI issues the transactions to ensure that out-of-order sequences do not occur.

If the incoming VAXBI transaction is a read-type transaction and the address falls between the address in the DWMBA's BIIC Starting and Ending Address Registers, the slave sequencer determines if a DMA buffer is available for use. If so, the slave sequencer moves the C/A data to the DMA(x) buffer, where x indicates either DMA-A or DMA-B, and notifies the DWMBA/A module that VAXBI C/A data has been loaded and the DWMBA/A module should request the XMI bus. The slave sequencer then issues a STALL response to the VAXBI until the transaction completes.

Later, the DWMBA/A module receives a Read response cycle from XMI memory with the requested data. The DWMBA/A module loads the data into the DMA data buffer and notifies the slave sequencer in the DWMBA/B module that the requested data is available. The slave sequencer then moves the data to the VAXBI, completing the request.

The DWMBA does not support the caching of memory on VAXBI nodes so VAXBI reads are always answered with the VAXBI "don't cache" read status.

6.3.2 VAXBI-to-XMI Memory Space Interlock Reads

VAXBI interlock reads (IRCI) behave the same as reads except if a VAXBI node references a location in XMI memory that is locked. Then the memory returns a Locked Response (LOC) to the DWMBA, and the DWMBA issues a RETRY confirmation code to the VAXBI commander, which then releases the VAXBI. The DWMBA returns to idle and awaits the next VAXBI request.

6.3.3 VAXBI-to-XMI Memory Writes

The disconnected write mode of operation is used for VAXBI-to-XMI memory writes, allowing use of the VAXBI by other devices while the DWMBA completes the write on the XMI.

The DWMBA's slave sequencer moves the C/A and write data (whether longword, quadword, or octaword) to an available DMA buffer location when the incoming write-type VAXBI transaction's address falls between the addresses in the DWMBA'S Starting and Ending Address Registers in its BIIC. The slave sequencer then issues an ACK confirmation to the VAXBI.

When the buffer load completes, the slave sequencer notifies the DWMBA/A module's XMI transmit logic that it should request the XMI bus. Upon receiving an XMI grant, the DWMBA transmits the write data transaction and waits for an ACK response.

If a third VAXBI write transaction occurs before the first and second XMI writes complete, the DWMBA stalls this VAXBI transaction until the first XMI write completes successfully.

6.3.4 VAXBI-Generated Interrupts

Interrupts can either be (1) generated by the DWMBA if there is a status change or an error condition or (2) passed through the DWMBA to the XMI bus if generated by various I/O devices on the VAXBI bus. These interrupts are translated into the appropriate XMI interrupt transactions. If a DWMBA and a VAXBI device interrupt are both pending at the same IPL when an XMI IDENT transaction is issued, the DWMBA returns its vector to ensure that DWMBA error interrupts are serviced first.

6.4 DWMBA XMI-to-VAXBI Adapter Registers

Two sets of registers are used by the DWMBA: DWMBA registers (residing on both modules of the DWMBA) and VAXBI registers (residing in the BIIC). The DWMBA registers include the XMI required registers and DWMBA-specific registers in DWMBA private space.

Table 6–3 lists the DWMBA/A module XMI module registers. Table 6–4 lists the DWMBA/B module VAXBI module registers. Table 6–5 lists the VAXBI registers. See Chapter 5 of the *VAXBI Options Handbook* for a description of the VAXBI registers, except for the VAXBI Device Register. The remainder of Section 6.4 gives detailed descriptions of the DWMBA registers. The DWMBA/A module registers are presented first, followed by the DWMBA/B module registers and the VAXBI Device Register.

See Section 2.2.2.3 for details of I/O addressing.

Table 6–3 XMI Registers on the DWMBA/A Module

Name	Mnemonic[1]	Address[2]
Device Register	XDEV	BB+0000 0000
Bus Error Register	XBER	BB+0000 0004
Failing Address Register	XFADR	BB+0000 0008
Responder Error Address Register	AREAR	BB+0000 000C
Error Summary Register	AESR	BB+0000 0010
Interrupt Mask Register	AIMR	BB+0000 0014
Implied Vector Interrupt Destination/Diagnostic Register	AIVINTR	BB+0000 0018
Diag 1 Register	ADG1	BB+0000 001C

[1]The first letter of the mnemonic indicates the following:

X=XMI register, resides on the DWMBA/A XMI module
A=Resides on the DWMBA/A XMI module
B=Resides on the DWMBA/B VAXBI module

[2]The abbreviation "BB" refers to the base address of an XMI node (the address of the first location of the nodespace).

Table 6–4 XMI Registers on the DWMBA/B Module

Name	Mnemonic[1]	Address[2]
Control and Status Register	BCSR	BB+0000 0040
Error Summary Register	BESR	BB+0000 0044
Interrupt Destination Register	BIDR	BB+0000 0048
Timeout Address Register	BTIM	BB+0000 004C
Vector Offset Register	BVOR	BB+0000 0050
Vector Register	BVR	BB+0000 0054
Diagnostic Control Register 1	BDCR1	BB+0000 0058
Reserved Register	–	BB+0000 005C

[1]The first letter of the mnemonic indicates the following:

X=XMI register, resides on the DWMBA/A module
A=Resides on the DWMBA/A module
B=Resides on the DWMBA/B module

[2]The abbreviation "BB" refers to the base address of an XMI node (the address of the first location of the nodespace).

Table 6–5 VAXBI Registers

Name	Mnemonic	Address[1]
Device Register	DTYPE[2]	bb+00
VAXBI Control and Status Register	VAXBICSR	bb+04
Bus Error Register	BER	bb+08
Error Interrupt Control Register	EINTRSCR	bb+0C
Interrupt Destination Register	INTRDES	bb+10
IPINTR Mask Register	IPINTRMSK	bb+14
Force-Bit IPINTR/STOP Destination Register	FIPSDES	bb+18
IPINTR Source Register	IPINTRSRC	bb+1C
Starting Address Register	SADR	bb+20
Ending Address Register	EADR	bb+24
BCI Control and Status Register	BCICSR	bb+28
Write Status Register	WSTAT	bb+2C
Force-Bit IPINTR/STOP Command Register	FIPSCMD	bb+30
User Interface Interrupt Control Register	UINTRCSR	bb+40
General Purpose Register 0	GPR0	bb+F0
General Purpose Register 1	GPR1	bb+F4
General Purpose Register 2	GPR2	bb+F8
General Purpose Register 3	GPR3	bb+FC
Slave-Only Status Register	SOSR	bb+100
Receive Console Data Register	RXCD	bb+200

[1]The abbreviation "bb" refers to the base address of a VAXBI node (the address of the first location of the nodespace).

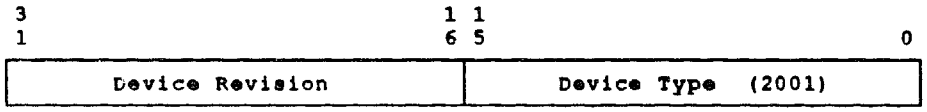
[2]Described in this section.

Device Register (XDEV)

The Device Register contains information to identify the node and is loaded during node initialization. A zero value indicates an uninitialized node.

ADDRESS

XMI nodespace base address + 0000 0000



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: RO, 0

Identifies the functional revision level of the module in hexadecimal. The DREV field always reflects the letter revision of the module as follows:

DWMBA/A Adapter Revision	DREV (decimal)	DREV (hex)
A0	1	0001
A1	1	0001
B0	2	0002
B1	2	0002
.		
.		
.		
Z0	26	001A

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

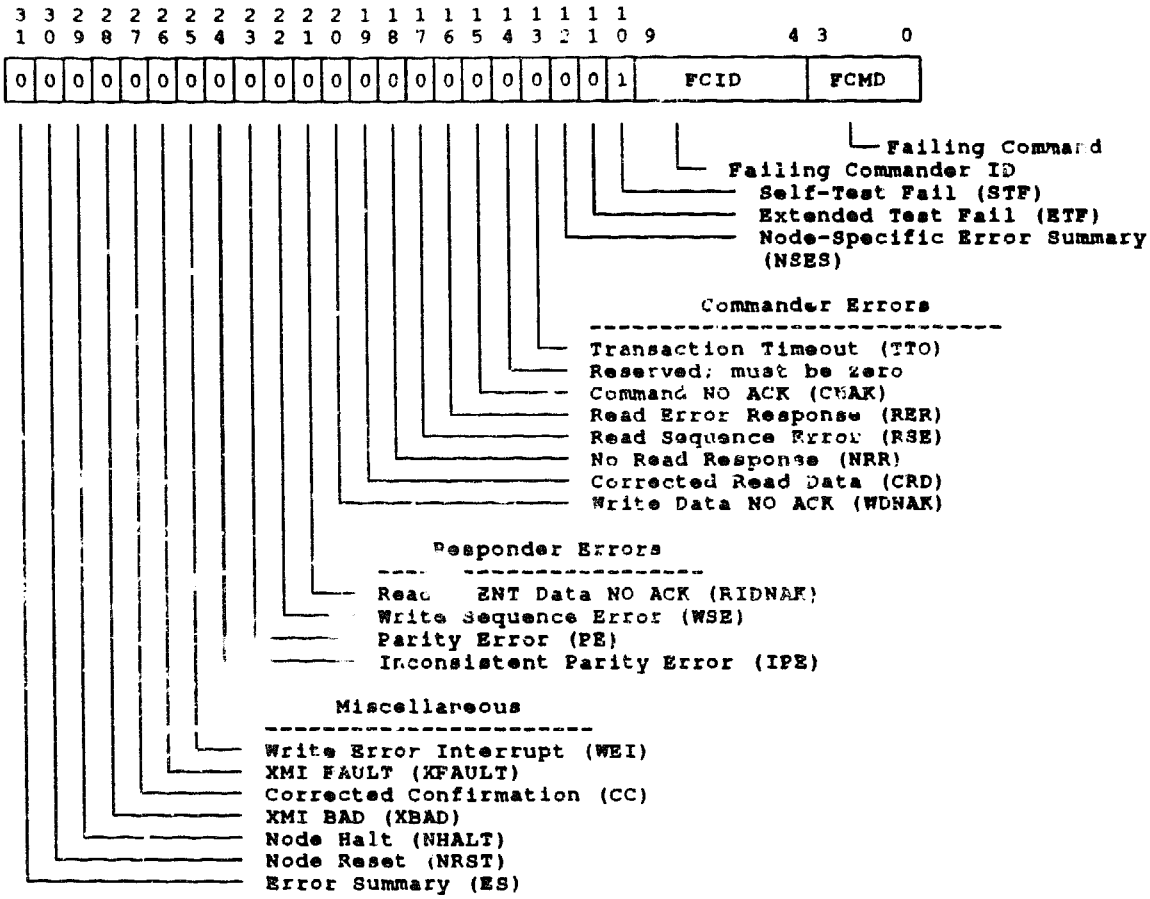
Type: RO, 0

Identifies the type of node. DTYPE is 2001 (hex) for the DWMBA/A module.

Bus Error Register (XBER)

The Bus Error Register contains error status on a failed XMI transaction. This status includes the failed command, commander ID, and an error bit that indicates the type of error that occurred. This status remains locked up until software resets the error bit(s).

ADDRESS *XMI nodespace base address + 0000 0004*



bit<31>

Name: Error Summary
Mnemonic: ES
Type: RO, 0

ES represents the logical OR of the error bits in this register. Therefore, ES asserts whenever any error bit asserts.

bit<30>

Name: Node Reset
Mnemonic: NRST
Type: R/W, 0

Writing a one to NRST initiates a power-up reset of the system. Reads to this bit location return zero. When NRST has a one written to it, the DWMBA:

- Resets all logic on the DWMBA/A module to an initialized (power-up) state.
- Asserts the RESET control signal to the DWMBA/B module, sequencing the VAXBI power supply(s). The assertion of RESET to the DWMBA/B causes the DWMBA/B to sequence BI AC LO, and BI DC LO. The assertion of BI DC LO causes the DWMBA/B module to reset to an initialized (power-up) state.

When NRST is set, it remains asserted for six to eight XMI cycles, after which it is cleared by logic on the DWMBA/A module. During the time that the DWMBA is performing its node reset, it does not affect the operation of the XMI bus.

bit<29>

Name: Node Halt
Mnemonic: NHALT
Type: R/W, 0

Unused; must be zero.

bit<28>

Name: XMI BAD
Mnemonic: XBAD
Type: R/W, 0

Unused; must be zero.

bit<27>

Name: Corrected Confirmation
Mnemonic: CC
Type: R/W1C, 0

CC sets when the DWMBA detects a single-bit CNF error. Single-bit CNF errors are automatically corrected by the XCLOCK chip in the XMI Corner.

bit<26>

Name: XMI FAULT
Mnemonic: XFAULT
Type: R/W1C, 0

Unused; must be zero.

bit<25>

Name: Write Error Interrupt
Mnemonic: WEI
Type: R/W1C, 0

Unused; must be zero.

bit<24>

Name: Inconsistent Parity Error
Mnemonic: IPE
Type: R/W1C, 0

Unused; must be zero.

bit<23>

Name: Parity Error
Mnemonic: PE
Type: R/W1C, 0

When set, PE indicates that the DWMBA detected a parity error on an XMI cycle.

bit<22>

Name: Write Sequence Error
Mnemonic: WSE
Type: R/W1C, 0

When set, WSE indicates that the DWMBA aborted a write transaction directed to it due to missing data cycles.

bit<21>

Name: Read/IDENT Data NO ACK

Mnemonic: RIDNAK

Type: R/W1C, 0

When set, RIDNAK indicates that a Read or IDENT data cycle (GRDn, CRDn, LOC, RER) transmitted by the DWMBA received a NO ACK confirmation.

bit<20>

Name: Write Data NO ACK

Mnemonic: WDNAK

Type: R/W1C, 0

When set, WDNAK indicates that a Write data cycle (GRDn, CRDn, LOC, RER) transmitted by the DWMBA received a NO ACK confirmation.

bit<19>

Name: Corrected Read Data

Mnemonic: CRD

Type: R/W1C, 0

When set, CRD indicates that the DWMBA received a CRDn read response.

bit<18>

Name: No Read Response

Mnemonic: NRR

Type: R/W1C, 0

When set, NRR indicates that a read transaction initiated by the DWMBA failed due to a read response timeout.

bit<17>

Name: Read Sequence Error

Mnemonic: RSE

Type: R/W1C, 0

When set, RSE indicates that a transaction initiated by the DWMBA failed due to a read sequence error.

bit<16>

Name: Read Error Response
Mnemonic: RER
Type: R/W1C, 0

When set, RER indicates that a DWMBA received a Read Error Response.

bit<15>

Name: Command NO ACK
Mnemonic: CNAK
Type: R/W1C, 0

When set, CNAK indicates that a command cycle transmitted by the DWMBA received a NO ACK confirmation caused by either a reference to a nonexistent memory location or a command cycle parity error. This bit is set only if the reattempts fail.

bit<14>

Name: Reserved
Mnemonic: None
Type: R/W, 0

Reserved; must be zero.

bit<13>

Name: Transaction Timeout
Mnemonic: TTO
Type: R/W1C, 0

When set, TTO indicates that a transaction initiated by the DWMBA failed due to a transaction timeout. This bit is set only if the reattempts fail.

bit<12>

Name: Node-Specific Error Summary
Mnemonic: NSES
Type: RO, 0

When set, NSES indicates that a node-specific error condition was detected. The exact nature of the error is contained in DWMBA-specific registers.

bit<11>

Name: Extended Test Fail
Mnemonic: ETF
Type: R/W1C, 0

Unused; must be zero.

bit<10>

Name: Selt-Test Fail
Mnemonic: STF
Type: R/W1C, 1

When set, STF indicates that the DWMBA has not yet passed its self-test. This bit is cleared by the CPU node that executed the DWMBA self-test when the DWMBA passes its self-test.

bits<9:4>

Name: Failing Commander ID
Mnemonic: FCID
Type: RO

The Failing Commander ID field logs the commander ID of a failing transaction. FCID sets only if the retried transaction fails.

bits<3:0>

Name: Failing Command
Mnemonic: FCMD
Type: RO

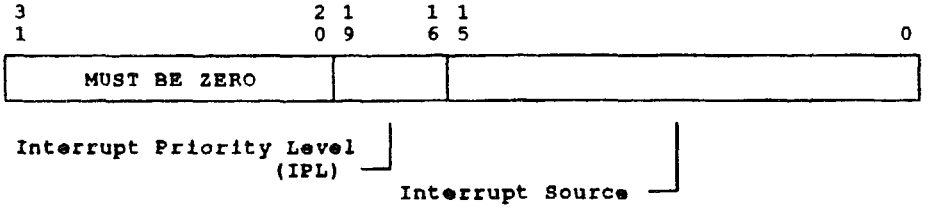
The Failing Command field logs the command code of a failing transaction. FCMD sets only if the retried transaction fails.

Failing Address Register (XFADR)

The Failing Address Register logs address and length information associated with a failing transaction. There are three interpretations of XFADR, depending on XBER<FCMD>.

ADDRESS *Nodespace base address + 0000 0008*

XFADR, when XBER<3:0> (FCMD) is 9 (hex), an IDENT transaction:



bits<31:20>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<19:16>

Name: Interrupt Priority Level
Mnemonic: IPL
Type: RO

IPL is a bit mask that specifies the interrupt priority level for the IDENT command as shown below:

Bit	IPL (hex)
19	17
18	16
17	15
16	14

bits<15:0>

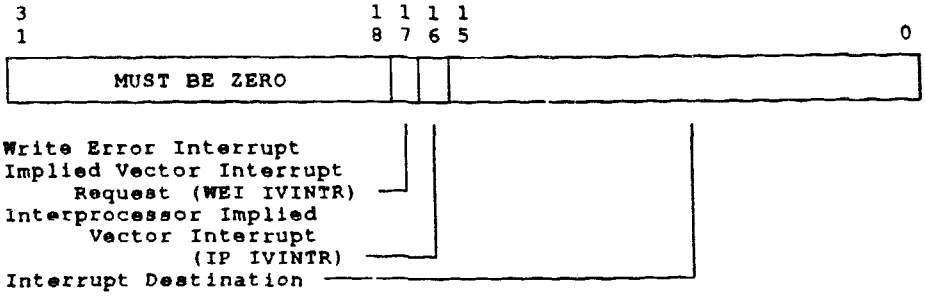
Name: Interrupt Source

Mnemonic: None

Type: RO

The Interrupt Source field is a bit mask that specifies the IDENT command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1.

XFADR, when XBER<3:0> (FCMD) is F (hex), on IVINTR transaction:



bits<31:18>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<17>

Name: Write Error Interrupt Implied Vector Interrupt Request

Mnemonic: WEI IVINTR

Type: RO

WEI IVINTR sets if the implied vector interrupt request was for a write error interrupt.

bit<16>

Name: Interprocessor Implied Vector Interrupt

Mnemonic: IP IVINTR

Type: RO

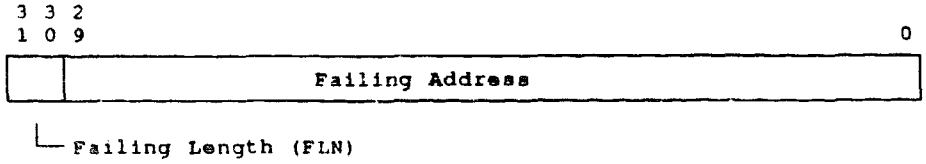
IP IVINTR sets if the implied vector interrupt request was for an interprocessor interrupt.

bits<15:0>

Name: Interrupt Destination
Mnemonic: None
Type: RO

The Interrupt Destination field is a bit mask that specifies the IVINTR command target-node ID. Bit<14> corresponds to node E, bit<13> corresponds to node D, . . . bit<1> corresponds to node 1

XFADR, when XBER<3:0> (FCMD) is neither an IDENT transaction nor an IVINTR transaction:



bits<31:30>

Name: Failing Length
Mnemonic: FLN
Type: RO

FLN logs the value of XMI D<31:30> during the commar d cycle of a failing transaction.

bits<29:0>

Name: Failing Address
Mnemonic: None
Type: RO

The Failing Address field logs the value of XMI D<29:0> during the command cycle of a failing transaction.

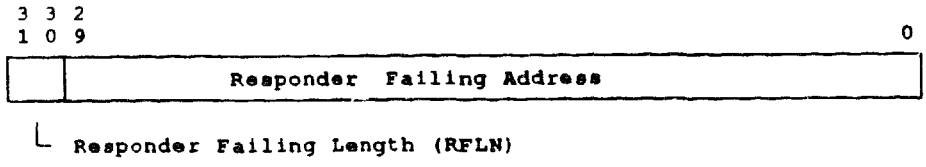
Responder Error Address Register (AREAR)

AREAR logs the failing address received from a CPU node initializing an I/O write, read, or IDENT transaction to the DWMBA or the VAXBI. AREAR is loaded when the DWMBA/A module ACKs the XMI's C/A cycle.

AREAR is locked when the DWMBA is unable to complete the requested operation, either a CPU write transaction that fails, resulting in the I/O Write Failure bit in the DWMBA/A module's Error Summary Register being set or a CPU read or IDENT transaction that results in the setting of the Data NO ACK bit in the DWMBA/A module's XBER register.

ADDRESS

XMI nodespace base address + 0000 000C



bits<31:30>

Name: Responder Failing Length

Mnemonic: RFLN

Type: RO

RFLN logs the value of XMI D<31:30> during the cycle that the DWMBA accepts the C/A cycle for the XMI commander.

bits<29:0>

Name: Responder Failing Address

Mnemonic: None

Type: RO

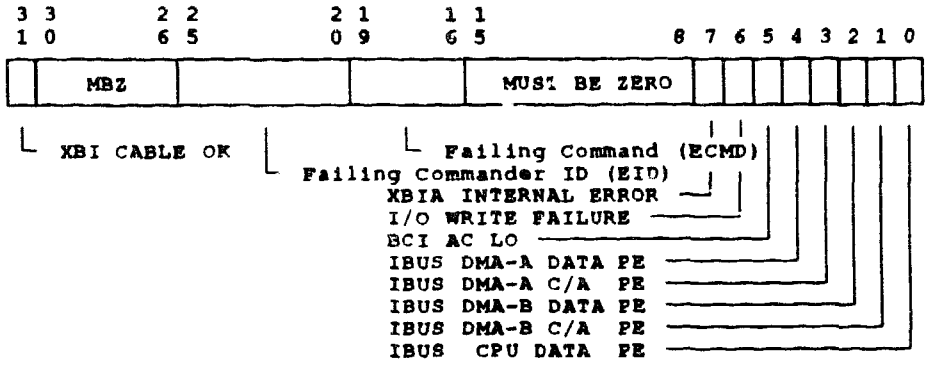
The Responder Failing Address field logs the value of XMI D<29:0> during the cycle that the DWMBA accepts the C/A cycle from the XMI commander.

Error Summary Register (AESR)

AESR is used to capture DWMBA/A module-related error conditions.

ADDRESS

XMI nodespace base address + 0000 0010



bit<31>

Name: XBI Cable OK

Mnemonic: None

Type: RO

XBI Cable OK sets to one on initialization if the four IBUS cables are correctly connected and if the DWMBA/B module has DC power from the VAXBI backplane. If XBI Cable OK clears and the DWMBA/B module has VAXBI DC power, then one or more of the cables is not connected or is incorrectly installed.

bits<30:26>

Name: Reserved

Mnemonic: None

Type: RO, O

Reserved; must be zero.

bits<25:20>

Name: Failing Commander ID

Mnemonic: EID

Type: RO

EID logs the XMI commander ID of a failed DWMBA I/O write, I/O read, or XMI IDENT transaction. The DWMBA will load this register after it ACKs the XMI commander's C/A cycle. EID locks if the DWMBA is unable to complete the requested operation as follows:

1 A failing CPU write transaction that sets the I/O Write Failure bit in the DWMBA/A module's Error Summary Register.

2 A CPU read or IDENT transaction that sets the Data NO ACK bit in the DWMBA/A module's Bus Error Register (XBER).

The lock on EID clears when both of the locking error conditions clear.

bits<19:16>

Name: Failing Command

Mnemonic: ECMD

Type: RO

ECMD logs the XMI commander command of a failed DWMBA I/O write, I/O read, or XMI IDENT transaction. The DWMBA loads this register after it ACKs the XMI commander's C/A cycle. ECMD locks if the DWMBA is unable to complete the requested operation as follows:

1 A failing CPU write transaction that sets the I/O Write Failure bit in the DWMBA/A module's Error Summary Register.

2 A CPU read or IDENT transaction that sets the Data NO ACK bit in the DWMBA/A module's Bus Error Register (XBER).

The lock on EID clears when the locking error conditions clear for both ECMD and EID.

bits<15:8>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<7>

Name: XBIA Internal Error

Mnemonic: None

Type: R/W1C, 0

The XBIA Internal Error bit sets to indicate that an UNEXPLAINED internal error to the DWMBA/A module gate array was detected, generally a hardware problem where control logic encountered UNDEFINED conditions. The DWMBA/A module issues an IVINTR transaction with "memory write error" set in the Type field when XBIA Internal Error sets.

bit<6>

Name: I/O Write Failure During CPU Write Transaction

Mnemonic: I/O Write Failure

Type: R/W1C, 0

I/O Write Failure During CPU Write transaction sets if the DWMBA/B module is unable to complete a CPU write transaction to either its register space or to VAXBI address space. Its assertion coincides with the generation of an IVINTR transaction due to this error condition. The DWMBA issues an IVINTR with "mem write error" set in the Type field when I/O Write Failure During CPU Write Transaction is asserted. Software uses this bit and other error bits to determine the cause of a DWMBA-generated IVINTR transaction.

When I/O Write Failure During CPU Write Transaction sets, the contents of the DWMBA/A module Responder Error Address Register, the Failing Commander ID bits, and the Failing Command bits lock.

bit<5>

Name: BCI AC LO

Mnemonic: None

Type: R/W1C, 1

The BCI AC LO bit sets when VAXBI power falls below specifications, as indicated by an asserted BCI AC LO L signal (asserted = one). The DWMBA issues an IVINTR with "mem write error" set in the Type field when BCI AC LO is asserted so that software can determine the cause of this IVINTR transaction. Software then clears BCI AC LO as part of the interrupt service routine that executes as a result of the IVINTR.

The DWMBA self-test program clears BCI AC LO.

bit<4>

Name: IBUS DMA-A Data Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-A Data Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-A data buffer location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-A Data Parity Error asserts.

bit<3>

Name: IBUS DMA-A C/A Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-A C/A Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-A C/A location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-A C/A Parity Error asserts and the failing DMA transaction is a write or interrupt. The DWMBA issues an error interrupt if this error bit is set and the appropriate mask bit is also set.

bit<2>

Name: IBUS DMA-B Data Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-B Data Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-B data buffer location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-B Data Parity Error asserts.

bit<1>

Name: IBUS DMA-B C/A Parity Error

Mnemonic: None

Type: R/W1C, 0

IBUS DMA-B C/A Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading a DMA-B C/A location. The DWMBA issues an IVINTR with "mem write error" set in the Type field when IBUS DMA-B C/A Parity Error asserts and the failing DMA transaction is a write. The DWMBA issues an error interrupt if this error bit is set and the appropriate mask bit is also set.

bit<0>

Name: IBUS CPU DATA Parity Error

Mnemonic: None

Type: R/W1C, 0

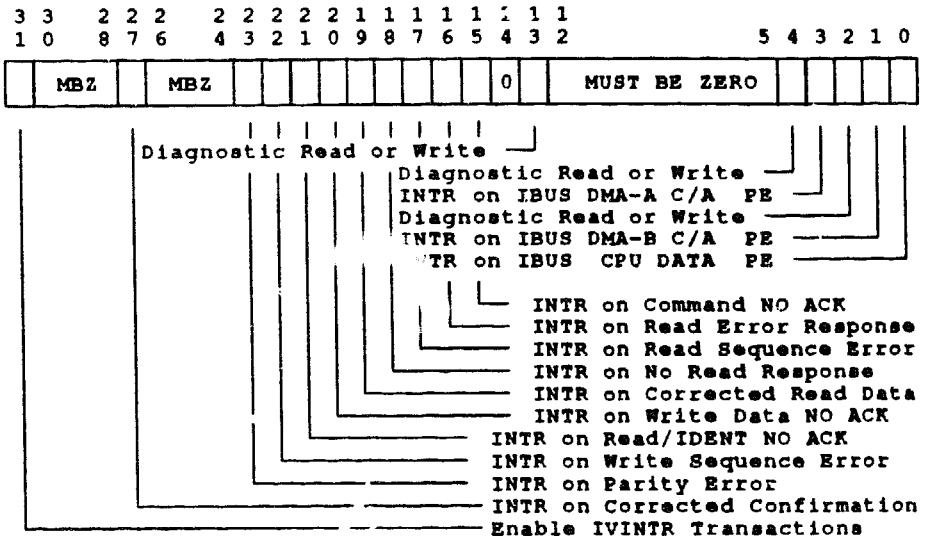
IBUS CPU DATA Parity Error sets when the DWMBA/A module detects a parity error on the IBUS when the DWMBA/B module was loading CPU DATA location during a CPU-initiated I/O read or IDENT. The DWMBA issues a Read Error Response (RER) to the commander when IBUS CPU DATA Parity Error asserts. The DWMBA issues an error interrupt to the XMI if this error bit is set and the appropriate mask bit is also set.

Interrupt Mask Register (AIMR)

AIMR enables/disables the generation of an error interrupt transaction when the corresponding error bit in both the DWMBA/A module's XMI Bus Error Register (XBER) and the DWMBA/A module's Error Summary Register (AESR) is set.

ADDRESS

XMI nodespace base address + 0000 0014



bit<31>

Name: Enable IVINTR Transactions

Mnemonic: None

Type: R/W, 0

When Enable IVINTR Transactions is set and the IVINTR Destination Register is properly configured, IVINTRs are enabled and can be issued on the XMI bus.

CAUTION: The Enable IVINTR Transactions bit MUST be set to ensure proper error reporting in the case of asynchronous write failures and to report the occurrence of a pending VAXBI power-fail not initiated by XMI AC LO, XMI DC LO, or XBI Node Reset.

bits<30:28>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero

bit<27>

Name: INTR on Corrected Confirmation
Mnemonic: None
Type: R/W, 0

When INTR on Corrected Confirmation sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<23> (PE) is set.

bits<26:24>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<23>

Name: INTR on Parity Error
Mnemonic: None
Type: R/W, 0

When the INTR on Parity Error bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<23> (PE) is set.

bit<22>

Name: INTR on Write Sequence Error
Mnemonic: None
Type: R/W, 0

When the INTR on Write Sequence Error bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBFR<22> (WSE) is set.

bit<21>

Name: INTR on Read/IDENT NO ACK

Mnemonic: None

Type: R/W, 0

When the INTR on Read/IDENT NO ACK sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<21> (RIDNAK) is set.

bit<20>

Name: INTR on Write Data NO ACK

Mnemonic: None

Type: R/W, 0

When the INTR on Write Data NO ACK sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<20> (WDNAK) is set.

bit<19>

Name: INTR on Corrected Read Data

Mnemonic: None

Type: R/W, 0

When the INTR on Corrected Read Data bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<19> (CRD) is set.

bit<18>

Name: INTR on No Read Response

Mnemonic: None

Type: R/W, 0

When the INTR on No Read Response bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<18> (NRR) is set.

bit<17>

Name: INTR on Read Sequence Error

Mnemonic: None

Type: R/W, 0

When the INTR on Read Sequence Error bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<17> (RSE) is set.

bit<16>

Name: INTR on Read Error Response
Mnemonic: None
Type: R/W, 0

When the INTR on Read Error Response bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<16> (RER) is set.

bit<15>

Name: INTR on Command NO ACK
Mnemonic: None
Type: R/W, 0

When the INTR on Command NO ACK bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if XBER<15> (CNAK) is set.

bit<14>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<13>

Name: Diagnostic Read or Write
Mnemonic: None
Type: RO, X

Diagnostic Read or Write is used by diagnostic tests.

bits<12:5>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<4>

Name: Diagnostic Read or Write
Mnemonic: None
Type: RO, X

Diagnostic Read or Write is used by diagnostic tests.

bit<3>

Name: INTR on IBUS DMA-A C/A PE

Mnemonic: None

Type: R/W, 0

When the INTR on IBUS DMA-A CA PE bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if a parity error was detected on the IBUS when the DWMBA/B module was loading a DMA-A C/A location.

bit<2>

Name: Diagnostic Read or Write

Mnemonic: None

Type: RO, X

Diagnostic Read or Write is used by diagnostic tests.

bit<1>

Name: INTR on IBUS DMA-B C/A PE

Mnemonic: None

Type: R/W, 0

When the INTR on IBUS DMA-B C/A PE bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if a parity error was detected on the IBUS when the DWMBA/B module was loading a DMA-B C/A location.

bit<0>

Name: INTR on IBUS CPU DATA PE

Mnemonic: None

Type: R/W, 0

When the INTR on IBUS CPU DATA PE bit sets, the DWMBA/A module asserts the IR XMI ERR BIT SET L line of the IBUS, which generates an interrupt request if a parity error was detected on the IBUS when the DWMBA/B module was loading the CPU data location.

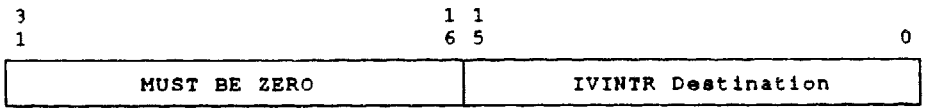
Implied Vector Interrupt Destination/Diagnostic Register (AIVINTR)

The AIVINTR is used during diagnostics and DWMBA-initiated IVINTR transactions.

ADDRESS

XMI nodespace base address + 0000 0018

AIVINTR, when used during DWMBA-initiated IVINTR transactions:



bits<31:16>

Name: Reserved

Mnemonic: None

Type: R/W

Not used; must be zero.

bits<15:0>

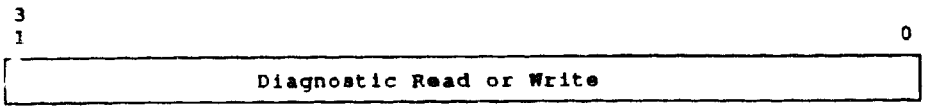
Name: IVINTR Destination

Mnemonic: None

Type: R/W, 0

The IVINTR Destination field determine which nodes on the XMI will be targeted by the DWMBA when it issues an Implied Vector Interrupt transaction. Each of the 16 bits corresponds to one of the 16 XMI nodes (only 14 nodes are used in the VAX 6000–400). When a bit is set, the selected node will be the target. For example, if bit<12> becomes set, then XMI node 12 is the node that the DWMBA selects to participate in the IVINTR transaction. Any number of bits can be set.

AIVINTR, when used during diagnostics:



bits<31:0>

Name: Diagnostic Read or Write

Mnemonic: None

Type: R/W

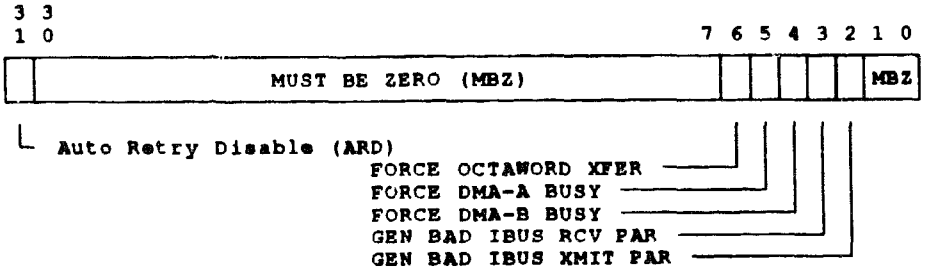
The Diagnostic Read or Write field is used by diagnostic routines to verify the integrity of the DWMBA/A module's main data path inside the DWMBA/A module gate array. When used in this manner, diagnostics need to raise the processor's IPL level above IPL 30 so that, should an error occur causing the DWMBA/A module to issue an IVINTR transaction, an unexpected interrupt will not occur.

Diag 1 Register (ADG1)

ADG1 is used by diagnostics to test parity and other features in the DWMBA/A module and the IBUS.

ADDRESS

XMI nodespace base address + 0000 001C



bit<31>

Name: Auto Retry Disable

Mnemonic: ARD

Type: R/W, 0

Setting Auto Retry Disable disables reattempts of failed XMI commander transfers. XMI error indications (NO ACKs) are immediately logged in the XMI Bus Error Register, and the appropriate action is taken.

CAUTION: **A NO ACK confirmation is a legal response that an XMI node may issue if it is currently unable to respond to the requested transaction because it is busy. If the user sets Auto Retry Disable, the user must ensure that either a "busy" NO ACK cannot be issued by the targeted node on the XMI or the DWMBA has the capability to handle a transaction that may not complete.**

bits<30:7>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<6>

Name: Force Octaword Transfers

Mnemonic: None

Type: R/W, 0

When Force Octaword Transfers is set, the DWMBA/A module generates octaword DMA transactions regardless of the length code that the DWMBA/B module loaded into the DMA buffer. The Force Octaword Transfers bit is used with Force DMA-A/B Busy (ADG1<5:4>), Flip FADR bit 1 (BDCR1<6>), and Flip Bit 29 (BDCR1<4>) to allow diagnostics to test the DWMBA's DMA buffer memory using CPU loopback transactions to XMI memory.

CAUTION: When Flip Bit 29 (BDCR1<4>) has been set to use the diagnostic feature "DMA loopback mode," only LEGAL addresses are permitted. ILLEGAL addresses result in UNDEFINED data. The CPU-generated address must be either 2xxx xxx0 or 2xxx xxx4 to be legal. The following are ILLEGAL addresses: 2xxx xxx8 and 2xxx xxxC.

bit<5>

Name: Force DMA-A Buffer Busy

Mnemonic: None

Type: R/W, 0

When set, the Force DMA-A Buffer Busy bit forces the DMA buffer control logic to place the DMA-A buffer into the BUSY state, forcing all DMA traffic through the DMA-B buffer.

CAUTION: If both ADG1<5> and ADG1<4> are set, all DMA transactions (VAXBI transactions that select the DWMBA as the slave and whose address falls within the bounds of the Starting and Ending Address Registers) will stall.

bit<4>

Name: Force DMA-B Buffer Busy

Mnemonic: None

Type: R/W, 0

When set, the Force DMA-B Buffer Busy bit forces the DMA buffer control logic to place the DMA-B buffer into the BUSY state, forcing all DMA traffic through the DMA-A buffer.

CAUTION: If both ADG1<5> and ADG1<4> are set, all DMA transactions (VAXBI transactions that select the DWMBA as the slave and whose address falls within the bounds of the Starting and Ending Address Registers) will stall.

bit<3>

Name: General Bad IBUS Receiver Parity

Mnemonic: GEN BAD IBUS RCV PAR

Type: R/W, 0

Setting GEN BAD IBUS RCV PAR causes the parity check bit on the DWMBA/A module for IBUS parity to be a one, regardless of the data that is loaded onto the buffer. Diagnostic routines use this bit and specific data patterns to force IBUS parity check errors on the DWMBA/A module when the DWMBA/B module loads the contents of the C/A or data buffers contained in the DWMBA/A module gate array.

bit<2>

Name: General Bad IBUS Transmit Parity

Mnemonic: GEN BAD IBUS XMIT PAR

Type: R/W, 0

Setting GEN BAD IBUS XMIT PAR causes the parity bit sent to the DWMBA/B module for IBUS parity to be a one, regardless of the data that resides in the buffer. Diagnostic routines use this bit and specific data patterns to force IBUS parity errors on the DWMBA/B module when the DWMBA/B module fetches the contents of the C/A or data buffers contained in the DWMBA/A module gate array.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

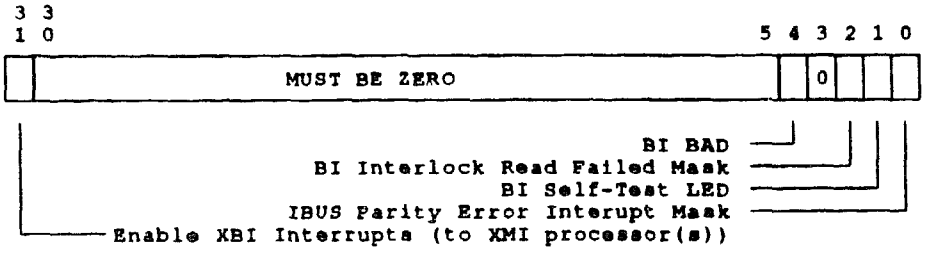
Reserved; must be zero.

Control and Status Register (BCSR)

BCSR contains DWMBA/B module operational control and status bits.

ADDRESS

XMI nodespace base address + 0000 0040



bit<31>

Name: Enable XBI Interrupts

Mnemonic: None

Type: R/W, C

Setting Enable XBI Interrupts enables the DWMBA to generate XMI interrupt requests in response to DWMBA-generated or VAXBI-generated interrupts. The appropriate interrupt mask bits must also be set for interrupts to be generated.

bits<30:5>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<4>

Name: BI BAD
Mnemonic: None
Type: RO

The initial state of the BI BAD bit on power-up or reset reflects the state of the BI BAD L line on the VAXBI by monitoring the line. It is used by console initialization software and error handling software to detect faulty VAXBI nodes. The assertion of BI BAD L on a VAXBI node results in the assertion of the XMI BAD line.

The BI BAD bit sets to logic level one when the VAXBI BI BAD L deasserts. When the BI BAD bit sets, it indicates that all VAXBI nodes have passed self-test, except for the DWMBA/B module, which does not assert BI BAD L.

bit<3>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<2>

Name: BI Interlock Read Failed Mask
Mnemonic: None
Type: R/W, 0

Setting BI Interlock Read Failed Mask to a one causes the DWMBA to generate an error interrupt request if BESR<2> (BI Interlock Read Failed) is set.

bit<1>

Name: BI Self-Test LED
Mnemonic: None
Type: R/W, 0

The BI Self-Test LED bit is set by the XMI boot processor node when the DWMBA self-test completes without error. If any portion of the DWMBA self-test fails, this bit does not set. When the BI Self-Test LED bit sets, the VAXBI Self-Test LED lights on the DWMBA/B module.

NOTE: The BI Self-Test LED bit has NO EFFECT on the operation of the XMI Self-Test LED on the DWMBA/A module. The XMI Self-Test LED is controlled by XBER<10>, Self-Test Fail.

bit<0>

Name: IBUS Parity Error Interrupt Mask

Mnemonic: None

Type: R/W, 0

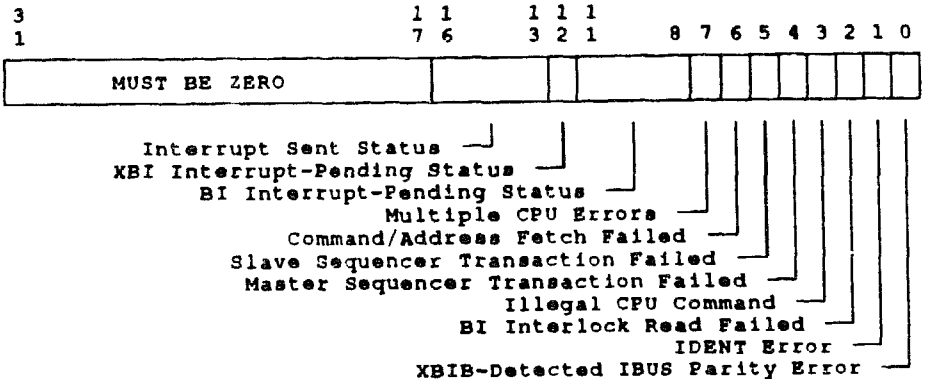
Setting IBUS Parity Error Interrupt Mask to one causes the DWMBA to generate an error interrupt request if BESR<0> (XBIB-Detected IBUS Parity Error) is set.

Error Summary Register (BESR)

The BESR contains status bits for errors detected by the DWMBA/B module.

ADDRESS

XMI nodespace base address + 0000 0044



bits<31:17>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<16:13>

Name: Interrupt Sent Status

Mnemonic: None

Type: RO, 0

The Interrupt Sent Status field corresponds to the 4-bit interrupt sent flops internal to the gate array, with BESR<16> corresponding to IPL<17>, BESR<15> corresponding to ILP<16>, etc. The interrupt sent status flops and BSER<12:8> determine the current interrupt-pending status.

bit<12>

Name: XBI Interrupt-Pending Status

Mnemonic: None

Type: RO, 0

The XBI Interrupt-Pending Status bit is a direct read of the XBI interrupt-pending flip-flop. A one indicates that a DWMBA interrupt is pending.

bits<11:8>

Name: BI Interrupt-Pending Status

Mnemonic: None

Type: RO, 0

The BI Interrupt-Pending Status field sets to indicate that one or more of the VAXBI interrupt-pending flip-flops is set. When asserted, they indicate that a VAXBI-generated interrupt targeting the DWMBA was successfully received and that a CPU IDENT at the correct IPL has not yet been received. This field is a direct read of the VAXBI interrupt-pending flip-flops, with BESR<11> corresponding to IPL<17> and BESR<8> corresponding to IPL<14>.

bit<7>

Name: Multiple CPU Errors

Mnemonic: None

Type: R/W1C, 0

Multiple CPU Errors sets when BESR<4> and BESR<0> have previously set due to a CPU transaction IBUS parity error when C/A or data is removed from the CPU buffer. This indicates that an error occurred on a subsequent CPU transaction before software had acknowledged a previously failed CPU transaction. This bit does not set on a parity error on write data accompanying the command/address on which an error was detected since the transaction has already been recorded as having failed.

bit<6>

Name: Command/Address Fetch Failed

Mnemonic: C/A Fetch Failed

Type: RO, 0

C/A Fetch Failed, when set with BESR<0> set, indicates that the DWMBA/B module detected an IBUS parity error on the C/A fetch from the CPU C/A buffer. C/A Fetch Failed will NOT set on a DWMBA/B module detected IBUS parity error when write data is fetched from the CPU Write Data buffer.

bit<5>

Name: Slave Sequencer Transaction Failed

Mnemonic: None

Type: RO, 0

Slave Sequencer Transaction Failed sets with BESR<0> to indicate that an IBUS parity error occurred while the slave sequencer had control of the IBUS during a read data fetch from the DMA read buffer.

bit<4>

Name: Master Sequencer Transaction Failed

Mnemonic: None

Type: RO, 0

Master Sequencer Transaction Failed sets with BESR<0> to indicate that an IBUS parity error occurred while the master sequencer had control of the IBUS during a C/A or write data fetch from the CPU buffer.

NOTE: This bit will be set but NOT VALID unless bit<0> in this register is also set.

bit<3>

Name: Illegal CPU Command

Mnemonic: None

Type: RO

Illegal CPU Command sets to indicate that an illegal CPU command was decoded by the DWMBA/B module. This error occurs only if an undetected multi-bit parity error happened during the time when the DWMBA/B module fetches the commmand/address from the CPU buffer. The error results in the master sequencer terminating the transaction and signaling the DWMBA/A module that the transaction failed.

The Illegal CPU Command bit does NOT generate an error interrupt.

bit<2>

Name: BI Interlock Read Failed

Mnemonic: None

Type: R/W1C, 0

BI Interlock Read Failed sets to indicate that a VAXBI-to-XMI memory Interlock Read operation failed to successfully complete on the VAXBI. When this error occurs, it is highly probable that the lock set in XMI memory will not be unlocked by the VAXBI device that issued the Interlock Read. The contents of the Timeout Address Register and the setting of BI Interlock Read Failed can be used to determine the locked address in XMI memory. The operating system can clear the lock in XMI memory by writing to a specific CSR in XMI memory.

BI Interlock Read Failed sets whenever a VAXBI Interlock Read command has been decoded and the summary EV code Illegal CNF Received for Slave Data (ICRSD) is decoded during a VAXBI Interlock Read transaction. Setting BI Interlock Read Failed locks the contents of the Timeout Address Register. Writing a one to BI Interlock Read Failed clears both the bit and its lock on the register.

When BI Interlock Read Failed is set with its corresponding mask bit, an error interrupt request is generated.

bit<1>

Name: IDENT Error

Mnemonic: None

Type: R/W1C, 0

IDENT Error sets to indicate that the DWMBA received an XMI IDENT transaction and no VAXBI nor DWMBA interrupt requests were pending at the IDENTed IPL. A set IDENT Error indicates an error condition on the XMI bus with multiple IDENTs being issued on the XMI for the same interrupt transaction. (Only one XMI IDENT is issued on the XMI if a single interrupt targets multiple CPUs.) All other CPUs that are waiting for an XMI bus grant to issue their XMI IDENTs will cancel their IDENT transactions if they see an IDENT transaction that matches the node ID and IPL of the IDENT that they are waiting to issue.

IDENT Error sets if a CPU IDENT command is decoded and no interrupts are pending in the DWMBA/B module gate array.

The setting of IDENT Error does NOT generate a DWMBA error interrupt.

bit<0>

Name: XBIB-Detected IBUS Parity Error

Mnemonic: None

Type: R/W1C, 0

XBIB-Detected IBUS Parity Error sets if the DWMBA/B module detects an IBUS parity error on a CPU transaction's C/A cycle, on a write data cycle when the data is removed from the CPU buffer by the master sequencer, or on a DMA transaction read data cycle when the read data is removed from the DMA read buffer by the slave sequencer. When XBIB-Detected IBUS Parity Error sets, the appropriate bit of BESR<6:4> sets.

The Timeout Address Register also locks on IBUS parity errors detected during DMA read data fetches from the buffer.

Writing a one to XBIB-Detected IBUS Parity Error also clears BESR<6:4> and the lock on the Timeout Address Register.

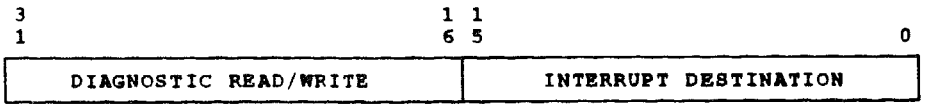
When the XBIB-Detected IBUS Parity Error bit is set with its corresponding mask bit, an error interrupt request is generated.

Interrupt Destination Register (BIDR)

BIDR is used by the DWMBA module to determine the targeted nodes on the XMI for an interrupt transaction. BIDR is used by both VAXBI-initiated and DWMBA error/status-initiated interrupts.

ADDRESS

XMI nodespace base address + 0000 0048



bits<31:0>

Name: Diagnostic Read/Write

Mnemonic: None

Type: R/W

The Diagnostic R/W field is used by diagnostics to verify much of the data path integrity of the DWMBA/B module gate array.

bits<15:0>

Name: Interrupt Destination

Mnemonic: None

Type: R/W, 0

The Interrupt Destination field determines the nodes on the XMI that are targeted by the DWMBA when it issues an interrupt transaction. Each bit in the 16-bit field corresponds to one of the 16 XMI nodes (only 14 nodes are used in the VAX 6000–400). When a bit is set to one, the selected node is the targeted node that the DWMBA will interrupt. Multiple bits can be set to interrupt as many XMI nodes as the user desires.

During diagnostics, bits<15:0> are used as part of the Diagnostic Read/Write bits<31:0>, as described above.

Timeout Address Register (BTIM)

The Timeout Address Register is loaded each time a VAXBI command/address is latched off the VAXBI. BTIM locks when (1) a VAXBI-to-XMI memory Interlock Read fails, causing the BI Interlock Read Failed bit (BESR<2>) to set, or (2) a VAXBI-to-XMI memory read-type fails, causing the XBIB-Detected IBUS Parity Error bit (BESR<0>) to set.

ADDRESS

XMI nodespace base address + 0000 004C

bits<31:0>

Name: BI DMA Failing Address

Mnemonic: None

Type: RO

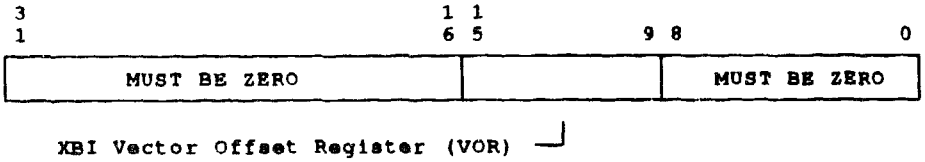
The BI DMA Failing Address contains the longword physical address (bits<29:0>) and length of the received VAXBI-to-XMI transaction (bits<31:30>.) If no errors are detected, the register reads back the last VAXBI transaction. The register logically locks upon error and unlocks when that error clears.

Vector Offset Register (BVOR)

BVOR contains a value that is concatenated with the VAXBI device-supplied vector, if bits<13:9> of the VAXBI-supplied vector are equal to zero.

ADDRESS

XMI nodespace base address + 0000 0050



bits<31:16>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<15:9>

Name: XBI Vector Offset Register

Mnemonic: VOR

Type: R/W, 0

BVOR is a 7-bit register loaded by software upon system initialization. BVOR contains a value that is concatenated with the VAXBI device-supplied vector, providing that bits<13:9> of the VAXBI-supplied vector are equal to zero, ensuring that multiple DWMBA/VAXBIs with the same devices on each bus will have a unique entry point into the SCB.

bits<8:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

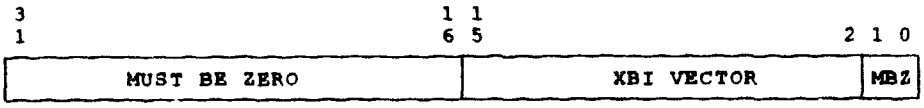
Reserved; must be zero.

Vector Register (BVR)

BVR is loaded by software upon system initialization. BVR contains the DWMBA vector that will be transmitted to the IDENTing XMI node when the DWMBA has a pending interrupt request that matches the interrupt source and IPL sent during the XMI IDENT transaction.

ADDRESS

XMI nodespace base address + 0000 0054



bits<31:16>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<15:2>

Name: XBI Vector

Mnemonic: None

Type: R/W, 0

The XBI vector is transmitted to the IDENTing XMI node when the DWMBA has a pending interrupt request that matches the interrupt source and IPL sent during the XMI IDENT transaction. This vector is NOT sent for any VAXBI-generated interrupts or BIIC interrupts due to error conditions.

bits<1:0>

Name: Reserved

Mnemonic: None

Type: RO, 0

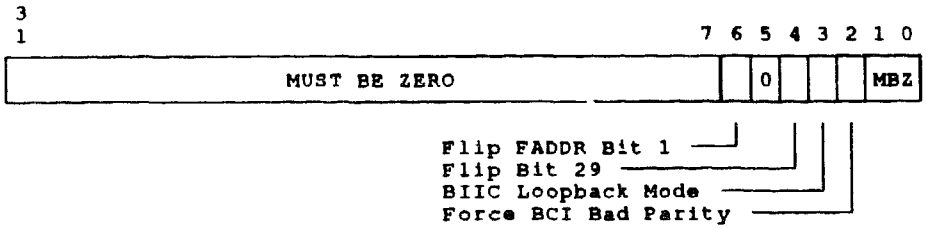
Reserved; must be zero.

Diagnostic Control Register 1 (BDCR1)

The BDCR1 is used by diagnostics to perform various diagnostic functions on the DWMBA/B module, ensuring that its hardware operates properly.

ADDRESS

XMI nodespace base address + 0000 0058



bits<31:7>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bit<6>

Name: Flip Failing Address Bit 1

Mnemonic: Flip FADDR Bit 1

Type: R/W, 0

The Flip FADDR Bit 1, used with Force DMA-A/B Busy bits (ADG1<5:4>) and Flip Bit 29, enables diagnostics to test the DWMBA's DMA buffer memory using CPU loopback transactions to XMI memory. When Flip FADDR Bit 1 is set, the invert state of FADDR Bit 1 is used to address the data words in the buffer, allowing diagnostics to use the buffer locations that normally would only be used for transfers greater than a quadword.

Setting Flip FADDR Bit 1 only affects FADDR bit 1 when the DWMBA/B module logic accesses data locations in the buffer. During the cycle when the C/A is addressed in the buffer, the setting of Flip FADDR Bit 1 has no effect on the buffer address.

bit<5>

Name: Reserved
Mnemonic: None
Type: RO, 0

Reserved; must be zero.

bit<4>

Name: Flip Bit 29
Mnemonic: None
Type: R/W, 0

Setting Flip Bit 29 inverts the state of bit 29 and BCI parity after the CPU C/A has been fetched and decoded by the master sequencer. The new address, which now resides in XMI memory space, is issued to the VAXBI. The DWMBA is the selected slave for the transaction, which processes this transaction like any other VAXBI-initiated DMA longword transaction, allowing diagnostic programs executing on the XMI to issue a CPU transaction to the DWMBA, which then converts it into a DMA transaction.

bit<3>

Name: BIIC Loopback Mode
Mnemonic: None
Type: R/W, 0

All requests to the master port of the BIIC become loopback requests whenever BIIC loopback mode is set, allowing the master sequencer to make loopback requests to access BIIC registers. The loopback mode prevents the BIIC from initiating VAXBI cycles to access the BIIC registers. When the BIIC is in loopback mode, it ignores the node ID portion of the address presented to it.

bit<2>

Name: Force BCI Bad Parity
Mnemonic: None
Type: R/W, 0

When Force BCI Bad Parity is set, bad parity is forced onto the BCI bus to the VAXBI during CPU C/A, CPU data cycles, and DMA read data cycles.

bits<1:0>

Name: Reserved
Mnemonic: None
Type: RO, 0

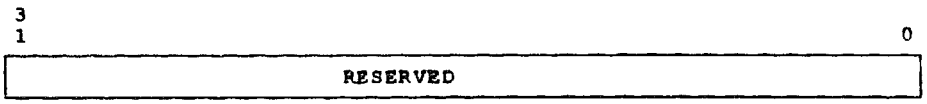
Reserved; must be zero.

Reserved Register

The Reserved Register is an undefined register that is reserved for future use. Reads to this register return UNDEFINED data with correct parity. Writes to this register appear to complete successfully.

ADDRESS

XMI nodespace base address + 0000 005C



bits<31:0>

Name: Reserved Register

Mnemonic: None

Type: Undefined

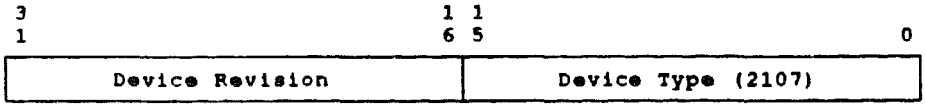
The reserved register bits are reserved for future use.

Device Register (DTYPE)

The VAXBI Device Register is loaded during self-test by console code with the DWMBA VAXBI device type and by the revision select logic with the revision level.

ADDRESS

VAXBI nodespace base address + 0000 0000



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

Identifies the revision level of the device. The revision level is loaded by hardware during BCI DC LO. For revision H, the DREV field contains 7 (hex). There is no revision I. Starting with revision J, the DREV field reflects the letter revision of the module as follows:

DWMBA/B Revision	DREV (decimal)	DREV (hex)
J0	10	000A
J1	10	000A
K0	11	000B
K1	11	000B
.		
.		
.		
Z0	26	001A

bits<15:0>

Name: Device Type

Mnemonic: DTYPE

Type: R/W, 0

Identifies the type of VAXBI node The processor's console code loads DTYPE with 2107 (hex) after successful completion of self-test.

6.5 Interrupts

The DWMBA XMI-to-VAXBI adapter implements two mechanisms for generating interrupts to XMI CPUs. One is in response to interrupts from the VAXBI bus and one in response to errors detected on the XMI bus. The BIIC also generates error interrupts on the VAXBI in response to errors on the VAXBI.

6.5.1 DWMBA XMI-to-VAXBI Adapter Vector Formats and Requirements

Interrupt vectors returned by VAXBI nodes, as seen by the XMI IDENT transactions, fall into three categories:

- XMI bus device interrupt vectors
- UNIBUS device interrupt vectors
- VAXBI bus device interrupt vectors

Figure 6–2 XMI Bus Vector Format

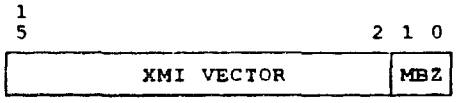


Figure 6–3 UNIBUS Vector Format

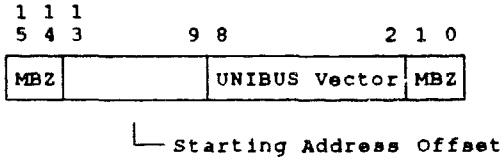
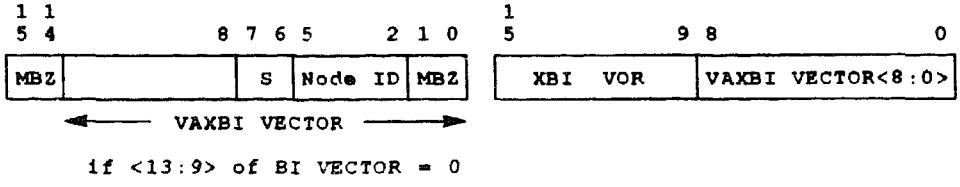


Figure 6–4 VAXBI Node Bus Vector Format



6.5.1.1 XMI Bus Vector Format

XMI device-initiated interrupts return vectors in the format shown in Figure 6–2 as a response to an XMI IDENT transaction. It is the responsibility of the operating system software to assign vector values to any vector register(s) that may exist on XMI devices that are capable of generating interrupt requests.

6.5.1.2 Offsettable Bus Vectors

There are several interrupt vectors returned by offsettable devices, including the BUA (VAXBI to UNIBUS Adapter) and the KLESI-B (VAXBI to Low-End Storage Interconnect). These other buses support devices that generate interrupts that must be differentiated from vectors generated by VAXBI devices. Figure 6–3 shows an example of the UNIBUS vector.

The UNIBUS vector field is an architecturally fixed vector returned by UNIBUS devices. Bits<8:0> cannot be modified by software. The SAO field must be a non-zero software assemble offset value to be used to index into the SCB with a unique vector.

6.5.1.3 VAXBI Node Vectors

The VAXBI node vector format has bits<15:9> as non-zero and are assigned a value by the operating system during initialization. The offset value, contained in XBI VOR (Vector Offset Register or BVOR) on the DWMBA/B module is concatenated with the vector value returned by a VAXBI node, bits<8:2>, providing that bits<13:9> of the VAXBI vector are zero. This new value is returned to the XMI commander during XMI IDENT cycles when a VAXBI node generates the interrupt request. If bits<13:9> of the VAXBI vector are non-zero, the vector will not be concatenated with the BVOR and will be passed to the XMI commander unchanged.

VAXBI device-initiated interrupts return vectors in the format shown in Figure 6–4 as a response to an XMI IDENT transaction. *Node ID* is the VAXBI node ID of the interrupt node. *S* is the interrupt vector number, which can be one of four possible interrupt vectors per node. *BVOR* must be a non-zero software assemble offset value to be used to index into the SCB with a unique vector for multiple VAXBI devices. BVOR bits<15:9> may be supplied by the DWMBA. The BVOR is necessary as the XMI is capable of supporting multiple DWMBA nodes, where the same device may exist on multiple VAXBIs. Since some VAXBI nodes might have fixed vectors that are unchangeable by software, the BVOR is used to ensure that multiple VAXBI devices with fixed vectors have a unique entry point into the SCB.

6.5.2 Interrupt Levels and Vectors

Table 6–6 lists the interrupt conditions used by the DWMBA adapter.

Table 6–6 DWMBA Adapter Interrupt Levels and Vectors

IPL (hex)	Name	Vector (hex)
17	DWMBA VAXBI Error/Status Change	XMI-7
17	VAXBI Level 7 Interrupt	VAXBI-7
16	VAXBI IPINTR 6 Interrupt	BIIC UINTRCSR REG-6[1]
16	VAXBI Level 6 Interrupt	VAXBI-6
15	VAXBI Level 5 Interrupt	VAXBI-5
14	VAXBI Level 4 Interrupt	VAXBI-4

[1]The DWMBA treats IPINTR as an error. The IPINTR value is written in the UINTRCSR as a generic VAXBI interrupt. For example, if bits<13:0> of the vector value equals zero, then the DWMBA will logically "OR" the contents of the BVOR (Vector Offset Register) with the value contained in bits<8:0> of the vector.

6.5.3 Types of Interrupts

Two types of interrupts are generated or passed through the DWMBA to the XMI bus. They are the interrupts generated by the DWMBA due to a status change or error condition and those interrupts generated on the VAXBI bus by I/O devices. The VAXBI interrupts are translated into XMI interrupt transactions.

6.5.3.1 DWMBA-Generated Interrupts

The DWMBA generates two types of interrupts: error interrupts and power-fail interrupts.

Errors detected by the DWMBA logic set bits in the DWMBA/A module and DWMBA/B module error summary registers. If the corresponding interrupt mask bit is enabled, an interrupt at level 7 (IPL 17) is requested by the DWMBA. A DWMBA error interrupt request is cleared when an XMI IDENT transaction is received at IPL 17.

The DWMBA generates an IVINTR transaction when it detects that a power failure is about to take place on the VAXBI. When BCI AC LO is asserted, the DWMBA/A module generates an IVINTR transaction with "mem write error" set in the Type field that targets the XMI node(s) specified in the Destination field of the command. During power-up and initialization, the DWMBA does not issue IVINTR transactions.

6.5.3.2 VAXBI-Generated Interrupts

Interrupts directed at the DWMBA node are passed on to the XMI bus. The BIIC handles INTR transactions directed at the DWMBA node and sets one of four interrupt level flip-flops, which store the acceptance of an INTR transaction at the given level. The INTR transaction causes the DWMBA/B module to issue an XMI interrupt command, at the corresponding IPL, to be posted on the XMI.

The BIIC generates INTR transactions on the VAXBI in response to errors detected on the VAXBI. The user has control of this mechanism via the BIIC Error Interrupt Control Register. The DWMBA's BIIC is configured to select itself as a destination node for INTR transactions, thereby informing an XMI CPU of VAXBI-related errors.

Interprocessor interrupts generated by VAXBI nodes targeting the DWMBA are supported. For the DWMBA to receive interprocessor interrupts, the software must set the DWMBA/B module's IPINTR Mask Register and enable the IPINTREN bit in the DWMBA/B module's BCI Control and Status Register.

The DWMBA handles interprocessor interrupts by asserting the BCI INT 6 signal on the DWMBA/B module's BIIC, causing the BIIC to generate an IPL 16 interrupt. The DWMBA/B module's BIIC Interrupt Destination Register configures to select itself as the destination of the interrupt transaction, thus causing this interrupt to be received by the DWMBA/B module as a generic VAXBI IPL 16 interrupt. When the DWMBA/B module receives an IDENT transaction from the XMI, it issues the IDENT onto the VAXBI. If no other interrupts are pending on the VAXBI, the DWMBA/B module's BIIC issues the vector that had been previously written by software during initialization onto the BIIC's UINTRCSR register.

The interprocessor interrupt vector value written in the UINTRCSR is treated by the DWMBA hardware as a generic VAXBI interrupt. If bits<13:9> of the vector value are zero, then the DWMBA logically ORs the contents of the BVOR with the value contained in bits<8:0> of the vector.

6.5.4 XMI IDENT to VAXBI IDENT

There are two XMI to VAXBI IDENT transactions for the DWMBA: one when the DWMBA has no interrupts pending and one when the DWMBA has an interrupt pending.

6.5.4.1 XMI to VAXBI IDENT

The DWMBA issues a VAXBI IDENT when an XMI CPU issues an XMI IDENT unless the DWMBA has a pending interrupt at the IDENTed level.

The DWMBA issues an IDENT response cycle on the XMI (Good Read Data response—function code = 1000 with the vector in bits<15:2> of the data field) upon receiving a vector from the VAXBI.

The VAXBI interrupt-pending flip-flop(s) and the INTR Sent Flip-Flop(s) that correspond to the IDENTed IPL are cleared when BCI RAK L is asserted, after the DWMBA/B module makes a VAXBI request.

If the requesting VAXBI node aborts its interrupt request before the XMI CPU generates an IDENT transaction at that level, the resulting IDENT on the VAXBI gets NOACKed. The DWMBA then issues a Read Error Response (RER) to the XMI commander and sets the IDENT Error bit in the DWMBA/B module's Error Summary Register.

6.5.4.2 XMI to VAXBI IDENT (DWMBA Interrupt Pending)

If the DWMBA has its interrupt-pending flip-flop set and it decodes an XMI IDENT transaction with IPL 17 set in D<19:16> of the IDENT command, it responds by issuing the DWMBA's vector that is located in BVOR. When the vector has been written into the DWMBA/A module's register file by the DWMBA/B module's master sequencer (state machine controller), the DWMBA's interrupt-pending and sent flip-flops clear.

If an XMI CPU issues an IDENT to the DWMBA and the DWMBA has no interrupt-pending flip-flops set, the DWMBA issues the IDENT on the VAXBI. There is a direct mapping of the XMI IDENT IPL (D<19:16> to that of the VAXBI D<19:16>). No remapping is required.

6.6 Error Reporting

The DWMBA adapter uses two mechanisms for detecting and reporting errors. One mechanism is the BIIC for VAXBI-related errors and the other mechanism deals with DWMBA-internal and XMI-related errors.

6.6.1 VAXBI Errors

The BIIC implements error checking and reporting features that deal with the VAXBI. These errors are reported to an XMI CPU via BIIC registers where bus errors are reported: the Bus Error Register, Error Interrupt Control Register, and the Interrupt Destination Register.

6.6.2 DWMBA Errors

Error generation and checking is performed on the DWMBA, both ports of the CPU, DMA-A and DMA-B register files, and the IBUS data path between the modules.

A specific error is flagged in one of the two Error Summary Registers (AESR and BESR) so that errors can be traced by software and diagnostics. When an error occurs, the DWMBA locks its error and address registers to ensure that a subsequent transaction will not change any states in the DWMBA until software services the error condition(s).

Even though an error causes the DWMBA/A module to issue a write error IVINTR, any pending DMA or CPU transactions that are error free are processed to completion, even if a previous transaction was halted due to an error.

6.6.3 DWMBA XMI-to-VAXBI Adapter Error Response Matrix

Table 6–7 XMI Errors During DMA Transactions (VAXBI to XMI Memory)

XMI Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
XMI Fault	–	–	–	–
Corrected Read Data	–	DWMBA generates interrupt	–	–
Corrected Confirmation	DWMBA generates interrupt	–	DWMBA generates interrupt	DWMBA generates interrupt
Read Error Response	–	DWMBA generates interrupt (NO ACK to VAXBI)	–	–
Inconsistent Parity	–	–	–	–
Parity Error	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt
Write Data NO ACK	–	–	–	DWMBA generates interrupt
Command NO ACK	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Write Sequence Error	–	–	–	–
Read Sequence Error	–	DWMBA generates interrupt (NO ACK to VAXBI)	–	–
Transaction Timeout	DWMBA/A module generates IVINTR	–	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
No Read Response	–	DWMBA generates interrupt (NO ACK to VAXBI)	–	–
Write Error Interrupt	–	–	–	–
Read/IDENT Data NO ACK	–	–	–	–

Table 6–8 XMI Errors During CPU I/O Transactions (XMI to VAXBI)

XMI Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
XMI Fault	–	–	–	–
Corrected Read Data	–	–	–	–
Corrected Confirmation	–	DWMBA generates interrupt	–	–
Read Error Response	–	–	–	–
Inconsistent Parity	–	–	–	–
Parity Error	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt
Write Data NO ACK	–	–	–	–
Command NO ACK	–	–	–	–
Write Sequence Error	–	–	–	DWMBA generates interrupt
Read Sequence Error	–	–	–	–
Transaction Timeout	–	–	–	–
No Read Response	–	–	–	–
Write Error Interrupt	–	–	–	–
Read/IDENT Data NO ACK	–	DWMBA generates interrupt	–	–

Table 6–9 DWMBA Errors During DMA Transactions (VAXBI to XMI Memory)

DWMBA Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
		DWMBA/A XMI Module		
I/O Write Failure	–	–	–	–
BCI AC LO	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
IBUS DMA-A Data Parity Error	–	–	–	DWMBA/A module generates IVINTR
IBUS DMA-A C/A Parity Error	DWMBA/A module generates interrupt (NO ACK to VAXBI)	–	DWMBA/A module generates IVINTR	–
IBUS DMA-B Data Parity Error	–	–	–	DWMBA/A module generates IVINTR
IBUS DMA-B C/A Parity Error	DWMBA/A module generates interrupt (NO ACK to VAXBI)	–	DWMBA/A module generates IVINTR	–
IBUS CPU Data Parity Error	–	–	–	–
		DWMBA/B VAXBI Module		
Multi-CPU Errors	–	–	–	–
Interlock Read Error	–	DWMBA generates interrupt Lock Time Register	–	–
IDENT Error	–	–	–	–
IBUS Data Parity Error	–	DWMBA generates interrupt. Bad Data/Parity to VAXBI.	–	–
Illegal CPU Command	–	–	–	–

Table 6–10 DWMBA Errors During CPU I/O Transactions (XMI to VAXBI)

DWMBA Error	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
		DWMBA/A XMI Module		
I/O Write Failure	–	–	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
BCI AC LO	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
IBUS DMA-A Data Parity Error	–	–	–	–
IBUS DMA-A C/A Parity Error	–	–	–	–
IBUS DMA-B Data Parity Error	–	–	–	–
IBUS DMA-B C/A Parity Error	–	–	–	–
IBUS CPU Data Parity Error	–	DWMBA generates interrupt. RER to XMI.	–	–
		DWMBA/B VAXBI Module		
Multi-CPU Errors	DWMBA generates interrupt. RER to XMI	–	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
Interlock Read Error	–	–	–	–
IDENT Error	–	RER to XMI	–	–
IBUS Data Parity Error	DWMBA generates interrupt. RER to XMI.	–	DWMBA/A module generates IVINTR	DWMBA/A module generates IVINTR
Illegal CPU Command	DWMBA generates interrupt. RER to XMI	–	DWMBA/A module generates IVINTR	–

Table 6–11 VAXBI Errors During DMA Transactions (VAXBI to XMI Memory)

VAXBI Error (DWMBA/B module's BIIC)	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
NO ACK to Multi-responses	–	–	–	
Master Xmit Error	–	–	–	–
Control Xmit Error	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt	DWMBA generates interrupt
Master Parity Error	–	–	–	–
Interlock Sequence Error	–	–	DWMBA generates interrupt	–
Transmitter During Fault	–	DWMBA generates interrupt	–	–
IDENT Vector Error	–	–	–	–
Command Parity Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Slave Parity Error	–	–	–	DWMBA generates interrupt
Read Data Substitute	–	–	–	–
Retry Timeout	–	–	–	–
Stall Timeout	–	–	–	–
Bus Timeout	–	–	–	–
Nonexistent Address	–	–	–	–
Illegal Confirmation Error	–	DWMBA generates interrupt	–	–
ID Parity Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Corrected Read Data	–	–	–	–
Null Bus Parity Error	–	–	–	–

Table 6–12 VAXBI Errors During CPU I/O Transactions (XMI TO VAXBI)

VAXBI Error (DWMBA/B module's BIIC)	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
NO ACK to Multi-responses	DWMBA generates interrupt	–	–	–
Master Xmit Error	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR.	DWMBA generates interrupt. DWMBA/A module generates IVINTR.
Control Xmit Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Master Parity Error	–	DWMBA generates interrupt. RER to XMI	–	–
Interlock Sequence Error	–	–	DWMBA generates interrupt	DWMBA generates interrupt
Transmitter During Fault	DWMBA generates interrupt	–	DWMBA generates interrupt	DWMBA generates interrupt
IDENT Vector Error	–	DWMBA generates interrupt	–	–
Command Parity Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–
Slave Parity Error	–	–	–	DWMBA generates interrupt
Read Data Substitute	–	DWMBA generates interrupt. RER to XMI	–	–
Retry Timeout	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR.	–
Stall Timeout	–	–	–	–
Bus Timeout	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR.	–
Nonexistent Address	DWMBA generates interrupt. RER to XMI	–	DWMBA generates interrupt. DWMBA/A module generates IVINTR	–
Illegal Confirmation Error	DWMBA generates interrupt. RER to XMI	DWMBA generates interrupt. RER to XMI	DWMBA generates interrupt. DWMBA/A module generates IVINTR	DWMBA generates interrupt. DWMBA/A module generates IVINTR
ID Parity Error	DWMBA generates interrupt	–	DWMBA generates interrupt	–

Table 6–12 (Cont.) VAXBI Errors During CPU I/O Transactions (XMI TO VAXBI)

VAXBI Error (DWMBA/B module's BIIC)	Read C/A Cycle	Read Data Cycle	Write C/A Cycle	Write Data Cycle
Corrected Read Data	–	DWMBA generates interrupt	–	–
Null Bus Parity Error	-	–	–	–

6.7 DWMBA Initialization, Self-Test, and Booting

This section discusses the DWMBA adapter initialization and diagnostics.

6.7.1 DWMBA Initialization

The three ways to reset the DWMBA are:

- Power-Up Sequence—When the VAX 6000–400 is powered up, XMI AC LO L and XMI DC LO L are sequenced so that all XMI nodes are reset.
- System Reset—The XMI emulates a power-up sequence by asserting the XMI RESET L line, causing the power supply to sequence XMI AC LO L and XMI DC LO L as in a "real" power-up. Software asserts XMI RESET L by writing to IPR55. The XMI does not differentiate between a "real" power-up and a system reset. The console INITIALIZE command generates a system reset if no argument is given.
- Node Reset—A DWMBA is "node reset" by setting its XBER<30> (NRST) bit. The console INITIALIZE command generates a node reset if a node ID argument is provided. For the KA64A CPU module the differences between the node reset and a system reset are as follows:
  - XMI AC LO L is not sequenced during node reset.
  - VAXBI "self-test" is not run during node reset.

When initialized, the DWMBA performs as follows:

- All DWMBA logic resets to a known state.
- The DWMBA asserts XMI STF L until self-test completes successfully.
- The DWMBA registers are initialized to a known value by self-test.

The VAXBI subsystem of the DWMBA resets as would any VAXBI system whenever the XMI resets. Each VAXBI backplane in a VAX 6000–400 is connected to power, and each DWMBA/B module has logic that controls the VAXBI backplane.

Setting XBER<30> (NRST) initiates a node reset, which resets both the DWMBA/A module and the DWMBA/B module. When NRST is written to a one, the DWMBA/B module sequences the BI AC LO and BI DC LO signals, causing each VAXBI node to reset its logic.

A DWMBA/B module and its VAXBI subsystem, when powered down, has no effect on the DWMBA/A module and the XMI bus. When the VAXBI is powered down, a read to the VAXBI gives a RER, while a write gives a hard error interrupt.

6.7.2 DWMBA Self-Test and Diagnostics

The two diagnostic control registers are used to force bad parity internal to the DWMBA, for performing a loopback in the BIIC, and to act as temporary storage registers for diagnostic routines.

6.7.2.1 Loopback

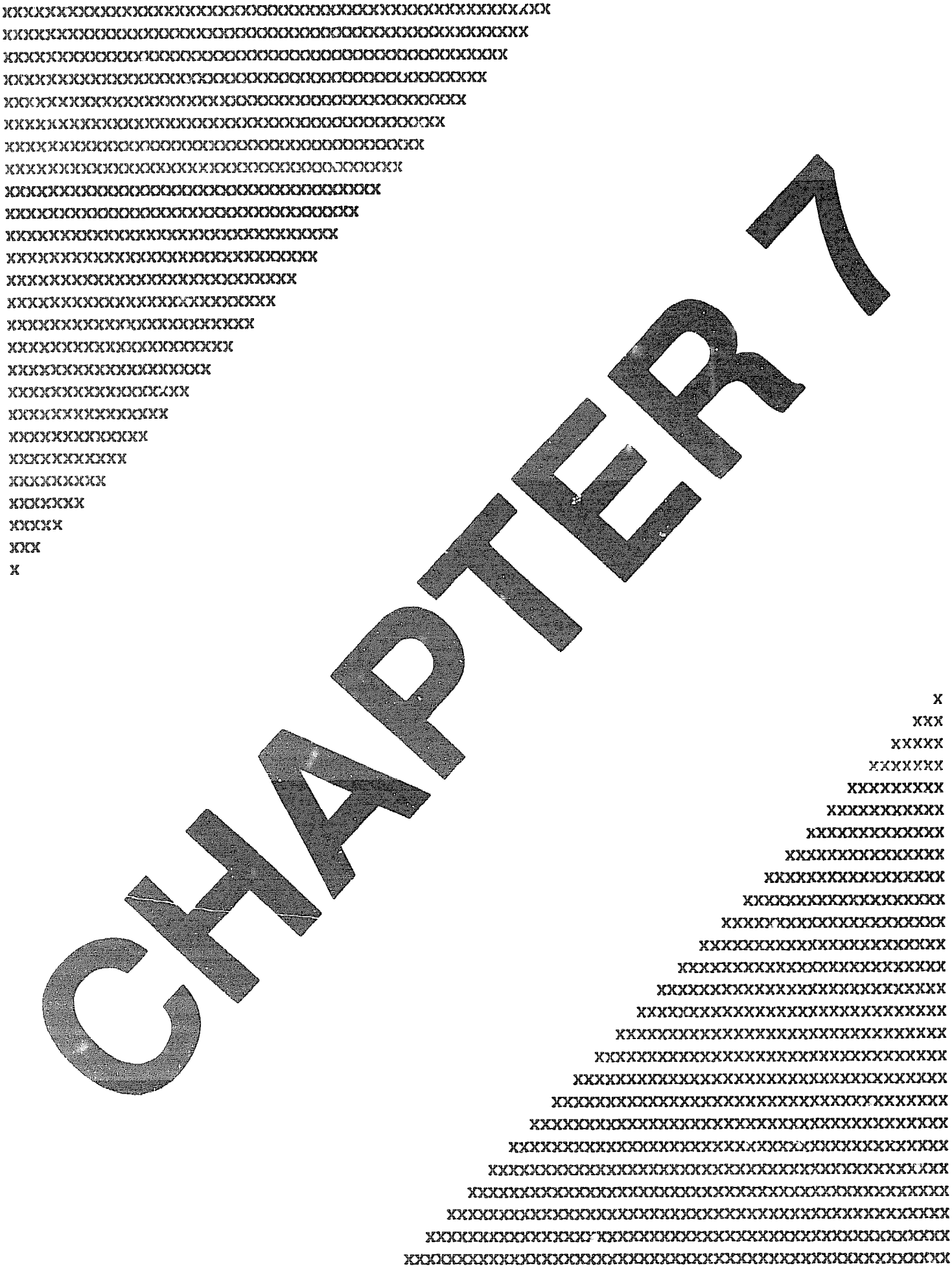
Two diagnostic loopbacks are implemented on the DWMBA: the BIIC loopback of the VAXBI and a transformation of a CPU transaction into a DMA transaction.

The BIIC loopback of the VAXBI occurs when the DWMBA/B module's BDCR1<3> (Force BIIC Loopback Mode) sets to a one.

When BDCR1<4> (Flip Bit 29) sets to a one, the DWMBA/B module inverts the state of address bit<29> and BCI parity when they are sent to the BIIC, allowing a VAXBI I/O space request to be converted into a DMA request that targets the DWMBA as the selected slave. This causes a CPU transaction to be transformed into a DMA transaction (longword only) that accesses XMI memory.

6.7.2.2 Self-Test

DWMBA self-test is executed by the boot processor on the XMI using the processor's resident ROM.


CHAPTER 7

7 Power and Cooling Systems

The VAX 6000–400 power system consists of an AC power controller, the power and logic unit, five power regulators, an optional battery backup unit, and a temperature sensor. The cooling system consists of two blower units and an airflow sensor, with the airflow path through the XMI and VAXBI card cages. See the *VAX 6000–400 Options and Maintenance* manual for more on power components.

7.1 Power System

The power system contains the following components:

- An H405-E AC power controller for 60 Hz systems; for 50 Hz, an H405-F and a high-voltage autotransformer
- An H7206 power and logic unit (PAL)
- Two H7215 power regulators, one for the XMI card cage and one for the VAXBI card cages
- Three H7214 power regulators, two for the XMI card cage and one for the VAXBI card cages
- An XTC power sequencer
- A temperature sensor and an airflow sensor
- An optional H7231 battery backup unit (BBU)

7.1.1 Input Power

The input power is five-wire (three-phase AC, neutral, and ground). 208V 60 Hz AC enters the H405-E AC power controller. Either 380 or 416V 50 Hz AC inputs the H405-F AC power controller and then enters the high-voltage autotransformer, which reduces the voltage to 208.

The H405 AC power controllers suppress conducted emissions. The AC power controller has a contactor that closes when the control panel upper key switch is in any position except "0," allowing AC power to the H7206, and opens if the cabinet's temperature sensor detects an excessive temperature.

7.1.2 H7206 Power and Logic Unit

The H7206 PAL:

- Rectifies the three-phase power into 300V DC for the DC-to-DC power regulators
- Develops regulated +14V DC for both internal use and the DC-to-DC power regulators
- Develops 110 watts of 24V DC for the cooling system blowers and its own internal fan
- Controls the interface between power regulators
- Controls the interface between the power regulators and the rest of the VAX 6000–400 system

A red LED on the front face of the PAL lights to indicate that an inhibit (shutdown) latch has been set.

Two green LEDs light to indicate the presence of the +14V DC for internal use and the presence of 300V DC.

7.1.3 H7214 Power Regulator

The H7214 inputs 300V DC and +14V bias. A 30 kHz clock synchronizes this to all other power components. Outputs are 120 A of +5V DC and 0.5 A of +13.5V DC for Ethernet transceivers. A green LED lights to indicate that the +5V output is present.

7.1.4 H7215 Power Regulator

The H7215 inputs 300V DC and outputs 20 A of –5V DC, 7 A of –2V DC, 4 A of +12V DC, and 2.5 A of –12V DC. A green LED lights to indicate that the outputs are present. An internal overtemperature switch asserts the OVERTEMP signal when necessary.

7.1.5 XTC Power Sequencer

The XTC power sequencer contains:

- XMI reset timing control logic
- Time-of-year (TOY) clock power circuits
- EIA RS-232/RS-423-compatible console line driver and receiver

7.1.5.1 XMI Reset Timing Control Logic

The XMI reset timing control logic handles these sequences:

- Cold start power-up
- Warm start power-up
- Loss of AC power followed by a cold start power-up
- Reset, which mimics a power-down and then a cold start power-up

7.1.5.2 TOY Circuits

The TOY circuits consist of a battery charger circuit that trickle charges the TOY clock battery and a voltage-level detection circuit that monitors the TOY BBU battery voltage.

7.1.5.3 Console Line Driver and Receiver

The XTC power sequencer contains the system console line driver and receiver, which are EIA RS-232/RS-423 compatible.

7.1.6 Power System Signals

Power system signals are partitioned so that a failure of power supply 1 shuts down only the XMI side or a failure of power supply 2 shuts down only the VAXBI side.

The power system signals are described in Table 7–1.

Table 7–1 Power System Signals

Name	Origin	Destination	Description
ON SENSE L	Control panel	XTC	Asserts when the control panel upper key switch is in any position except "0."
PNL RESET L	Control panel	XTC	Asserts while the control panel Restart button is pressed. Causes the XTC to start the reset sequence.
STANDBY CMD L	Control panel	H7206	Asserts when the control panel upper key switch is in any position except "0."
ON CMD L	Control panel	H7206	Asserts when the control panel upper key switch is in either the Enable or Secure position. Applies DC power to entire VAX 6000–400.
PB REQ L	Control panel	H7206, then from H7206 to DEC power bus and AC power controller	Asserts when STANDBY CMD L asserts to close a contactor in the AC power controller, applying AC power to H7206 and DC power to cooling system and memory. Controls all peripherals tied to the DEC power bus.
DEC Power Bus	Control panel	H405	Safety Extra Low Voltage (SELV) circuit that allows the VAX 6000–400 to turn other equipment on and off.
DCOK H	H7206	XTC	Asserts to indicate that the DC outputs from the power regulators are OK. Used by the XTC power sequencer to start the power-up/power-down sequence.
ACOK H	H7206	XTC	Asserts to indicate that the AC input voltage is adequate. It deasserts when the H7206's 300V DC output level reaches a level that guarantees 4.2 milliseconds of acceptable 300V DC prior to the deassertion of DCOK H. Used by the XTC power sequencer during the power-up/power-down sequence.
BBU STATUS	BBU	Control panel	Controls the green Battery LED.
MODULE ENABLE L	BBU	H7206	Asserts to indicate that the BBU is supplying 300V DC to the H7206, which causes only the memory modules to receive low voltage DC power.
BATTERY BACKUP ENABLE H (BBUE H)	H7206	BBU	Asserts to indicate that the memory module's power regulators are operational.

Table 7–1 (Cont.) Power System Signals

Name	Origin	Destination	Description
BATTERY BACKUP REQUEST H (BBUR H)	H7206	BBU	Asserts when ACOK deasserts to tell the BBU to start supplying 300V DC.
CHANNEL *n* OK (CH *n* OK)	Power regulator *n*	H7206	Asserts to tell the H7206 that the power regulator specified by the number *n* is OK.
OVER TEMPERATURE *n*	H7215	H7206	Asserts to tell the H7206 that the H7215 temperature is above specification, causing an orderly system shutdown followed by a latched inhibit of the appropriate outputs.
INTERLOCK *n* INHIBIT H	Cabinet interlock switch	H7206	Asserts to tell the H7206 that an interlock switch has been thrown, causing an orderly system shutdown followed by a latched inhibit of the appropriate outputs.
BLOWER FAULT H	Cooling system	H7206	Asserts to indicate an airflow sensor has detected a loss of airflow. When asserted for more than 30 seconds, an orderly system shutdown occurs followed by a latched inhibit of the outputs.
CHANNEL *n* INHIBIT	H7206	Power regulator *n*	Asserts to command the respective power regulator to turn off and reset to a ready state so that output power restores as the signal deasserts.
SYNC	H7206	Power regulator	A pulse train used to synchronize dependent power regulators.

7.2 Cooling System

The cooling system consists of two identical blowers, one for the front of the cabinet, the other for the back. An airflow sensor signals a loss of airflow.

The H7206 PAL unit has an internal fan.


INDEX

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X