
VAX 6000

VAX 6000 Model 500 System Technical User's Guide

Order Number: EK–650EA–TM–001

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, June 1991

Contents

INDEX

EXAMPLES

FIGURES

TABLES

PAGE xvi INTENTIONALLY LEFT BLANK

Preface

Intended Audience

This manual is written for Digital customer service engineers doing field-level repairs or programming and for OEMs who are writing specialized applications, such as their own operating systems.

Document Structure

This manual has four chapters.

- **Chapter 1** introduces the VAX 6000 Model 500 system and its parts.
- **Chapter 2** explains the KA65A CPU module.
- **Chapter 3** explains the FV64A vector processor.
- **Chapter 4** explains the MS65A memory module.
- The **Index** provides additional reference support.

VAX 6000 Series Documents

There are two sets of documentation: manuals that apply to all VAX 6000 series systems and manuals that are specific to one VAX 6000 model. Table 1 lists the manuals in the VAX 6000 series documentation set.

Table 1 VAX 6000 Series Documentation

Title	Order Number
Operation	
VAX 6000 Series Owner's Manual	EK–600EA–OM
VAX 6000 Series Vector Processor Owner's Manual	EK–60VAA–OM
VAX 6000 Vector Processor Programmer's Guide	EK–60VAA–PG
Service and Installation	
VAX 6000 Platform Technical User's Guide	EK–600EA–TM
VAX 6000 Series Installation Guide	EK–600EA–IN
VAX 6000 Installationsanleitung	EK–600GA–IN
VAX 6000 Guide d'installation	EK–600FA–IN
VAX 6000 Guia de instalacion	EK–600SA–IN
VAX 6000 Platform Service Manual	EK–600EA–MG

Table 1 (Cont.) VAX 6000 Series Documentation

Title	Order Number
Options and Upgrades	
VAX 6000: XMI Conversion Manual	EK–650EA–UP
VAX 6000: Installing MS65A Memories	EK–MS65A–UP
VAX 6000: Installing the H7236-A Battery Backup Option	EK–60BBA–IN
VAX 6000: Installing the FV64A Vector Option	EK–60VEA–IN
VAX 6000: Installing the VAXBI Option	EK–60BIA–IN

Manuals specific to models are listed in Table 2

Table 2 VAX 6000 Model Level Documentation

Title	Order Number
Models 200/300/400	
VAX 6000 Model 300 and 400 Service Manual	EK–624EA–MG
VAX 6000: Installing Model 200/300/400 Processors	EK–6234A–UP
VAX 6000 Model 200/300/400 Processor Console and Diagnostic ROM Upgrade Instructions	EK–60ROM–UP
VAX 6200–400 Options and Maintenance	EK–640EB–MG
VAX 6000–400 System Technical User's Guide	EK–640EB–TM
VAX 6200/6300 Options and Maintenance	EK–620AB–MG
VAX 6200/6300 System Technical User's Guide	EK–620AB–TM
Model 500	
VAX 6000 Model 500 Mini-Reference	EK–650EA–HR
VAX 6000 Model 500 Service Manual	EK–650EA–MG
VAX 6000 Model 500 System Technical User's Guide	EK–650EA–TM
VAX 6000: Installing Model 500 Processors	EK–KA65A–UP

Associated Documents

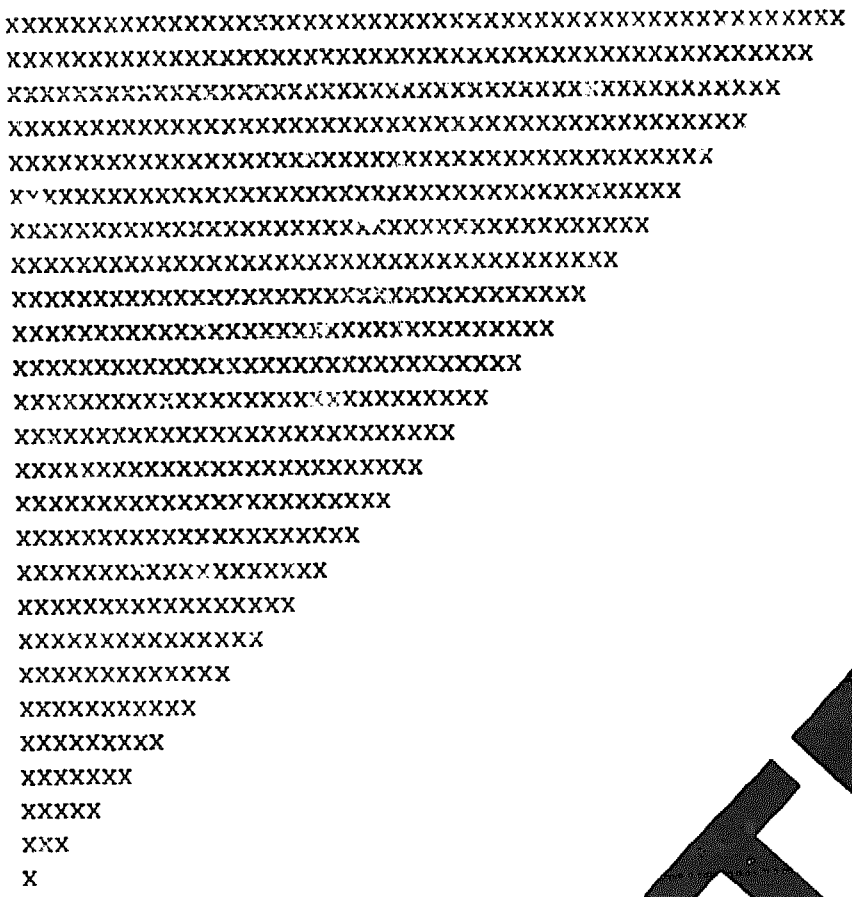
Table 3 lists other documents that you may find useful.

Table 3 Associated Documents

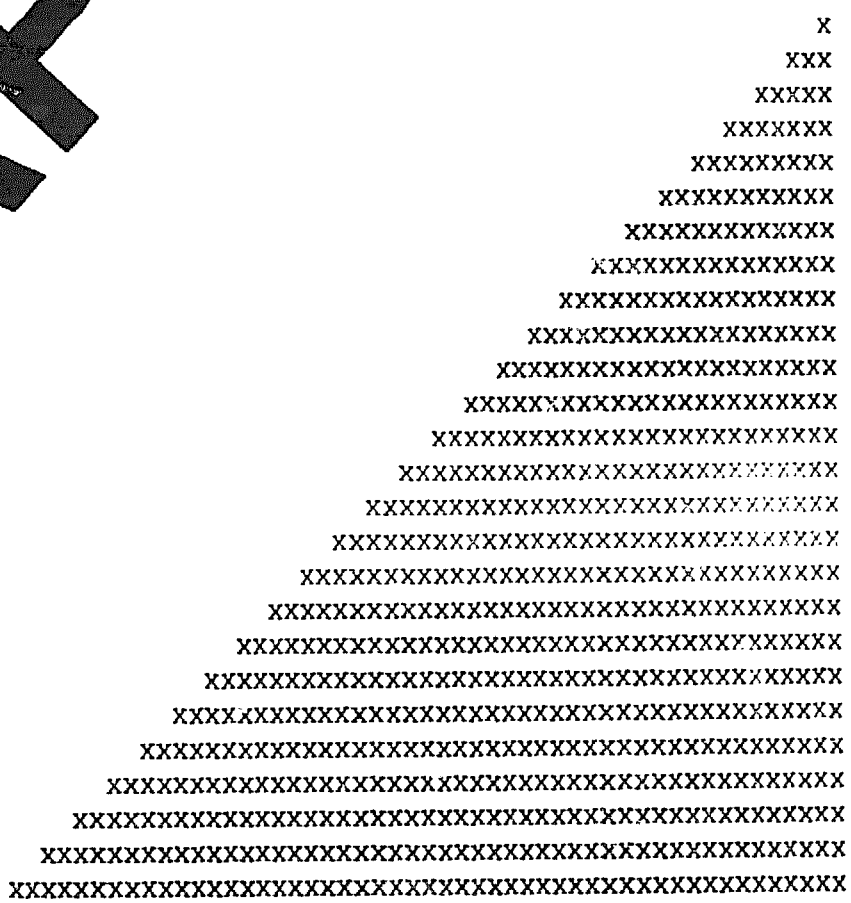
Title	Order Number
System Hardware Options	
VAXBI Expander Cabinet Installation Guide	EK–VBIEA–IN
VAXBI Options Handbook	EB–32255–46

Table 3 (Cont.) Associated Documents

Title	Order Number
System I/O Options	
CIBCA User Guide	EK–CIBCA–UG
CIXCD Interface User Guide	EK–CIXCD–UG
DEC LANcontroller 200 Installation Guide	EK–DEBNI–IN
DEC LANcontroller 400 Installation Guide	EK–DEMNA–IN
InfoServer 100 Installation and Owners Guide	EK–DIS1K–IN
KDB50 Disk Controller User's Guide	EK–KDB50–UG
KDM70 Controller User Guide	EK–KDM70–UG
RRD40 Disc Drive Owner's Manual	EK–RRD40–OM
RA90/RA92 Disk Drive User Guide	EK–ORA90–UG
SA70 Enclosure User Guide	EK–SA70E–UG
Operating System Manuals	
Guide to Maintaining a VMS System	AA–LA34A–TE
Guide to Setting Up a VMS System	AA–LA25A–TE
Introduction to VMS System Management	AA–LA24A–TE
ULTRIX–32 Guide to System Exercisers	AA–ME96B–TE
VMS Upgrade and Installation Supplement: VAX 6000 Series	AA–LB36C–TE
VMS Networking Manual	AA–LA48A–TE
VMS System Manager's Manual	AA–LA00A–TE
VMS VAXcluster Manual	AA–LA27B–TE
Peripherals	
HSC Installation Manual	EK–HSCMN–IN
H4000 DIGITAL Ethernet Transceiver Installation Manual	EK–H4000–IN
Installing and Using the VT320 Video Terminal	EK–VT320–UG
RV20 Optical Disk Owner's Manual	EK–ORV20–OM
SC008 Star Coupler User's Guide	EK–SC008–UG
TA78 Magnetic Tape Drive User's Guide	EK–OTA78–UG
TA90 Magnetic Tape Subsystem Owner's Manual	EK–OTA90–OM
TK70 Streaming Tape Drive Owner's Manual	EK–OTK70–OM
TU81/TA81 and TU/81 PLUS Subsystem User's Guide	EK–TUA81–UG
VAX Manuals	
VAX Architecture Reference Manual	EY–3459E–DP
VAX Systems Hardware Handbook — VAXBI Systems	EB–31692–46
VAX Vector Processing Handbook	EC–H0739–46



CHAPTER 1



1 The VAX 6000 Model 500 System

The VAX 6000 Model 500 computer system is designed for growth and can be configured for many different applications. Like other VAX systems, the VAX 6000 Model 500 system can support many users in a time-sharing environment. This system does the following:

- Supports a full range of VAX applications and operating systems
- Supports writeback caching which enhances system performance
- Functions as a standalone system, a member of a VAXcluster, 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 scalar and vector processors
- Implements symmetric multiprocessing where all processors have equal access to memory
- Uses a high-bandwidth system bus designed for multiprocessing
- Performs automatic self-test on power-up, reset, reboot, or system initialization

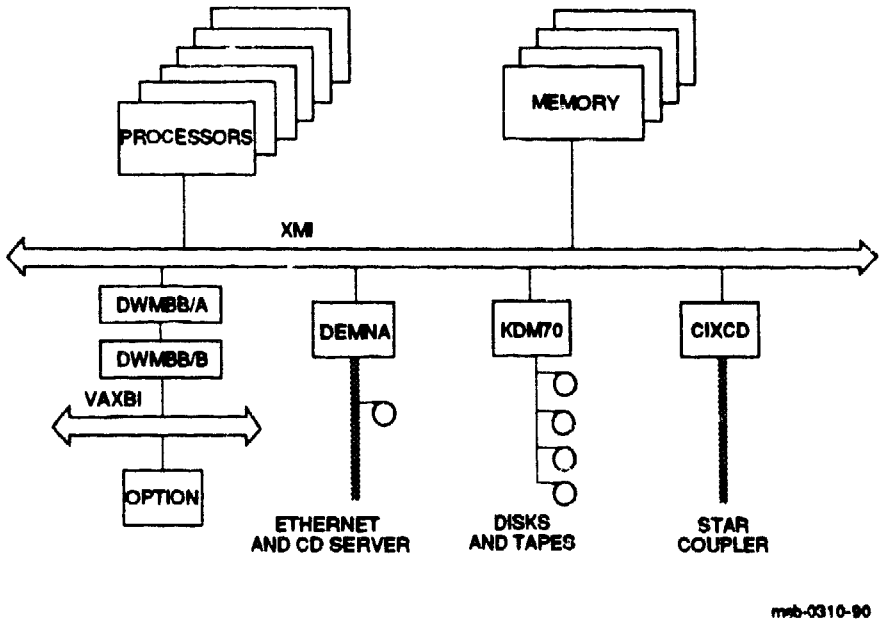
This chapter describes the system packages and introduces the location of components in the cabinet. Sections include:

- System Architecture
- Sample System
- System Front View
- System Rear View
- Supported Adapters

1.1 System Architecture

The high-speed XMI bus is used to interconnect processors, memory modules, and I/O adapters.

Figure 1–1 System Architecture



The XMI is the 64-bit system bus that interconnects the processors, memory modules, and I/O adapters.

The XMI bus uses the concept of a **node**. A node is a single functional unit that consists of one or more modules. The XMI has three types of nodes: processor nodes, memory nodes, and I/O adapters.

A **processor node** is a single-board scalar processor or a scalar/vector processor pair. The central processor unit (CPU) is comprised of two chips, one of which is a floating-point accelerator. A writeback cache subsystem improves system performance. In a multiprocessing system one scalar processor becomes the boot processor during power-up, and that boot processor loads the operating system and handles communication with the operator console. The other processors become secondary processors and receive system information from the boot processor.

Model 500 supports multiprocessing with up to six scalar processors and vector processing with up to four scalar/vector processor pairs. Symmetric multiprocessing is supported, allowing a program to execute on any processor.

A **memory node** is one memory module. Memory is a global resource equally accessible by all processors on the XMI. Each memory module may have 32, 64, or 128 Mbytes of memory consisting of ECC and control logic. The memories are automatically interleaved. An optional battery backup unit protects memory in case of power failure. The system supports up to eight memories.

I/O adapters are installed on the XMI bus (see Section 1.5). If the system has a VAXBI, then the DWMBB adapter is used to connect VAXBI I/O adapters to the XMI bus.

1.2 Sample System

A sample system has a system cabinet, a console load device—either in-cabinet or a compact disk server on the Ethernet—a console terminal and printer, an accessories kit, and a documentation set. The system may have additional storage devices and may be a member of a VAXcluster.

Figure 1–2 Sample System

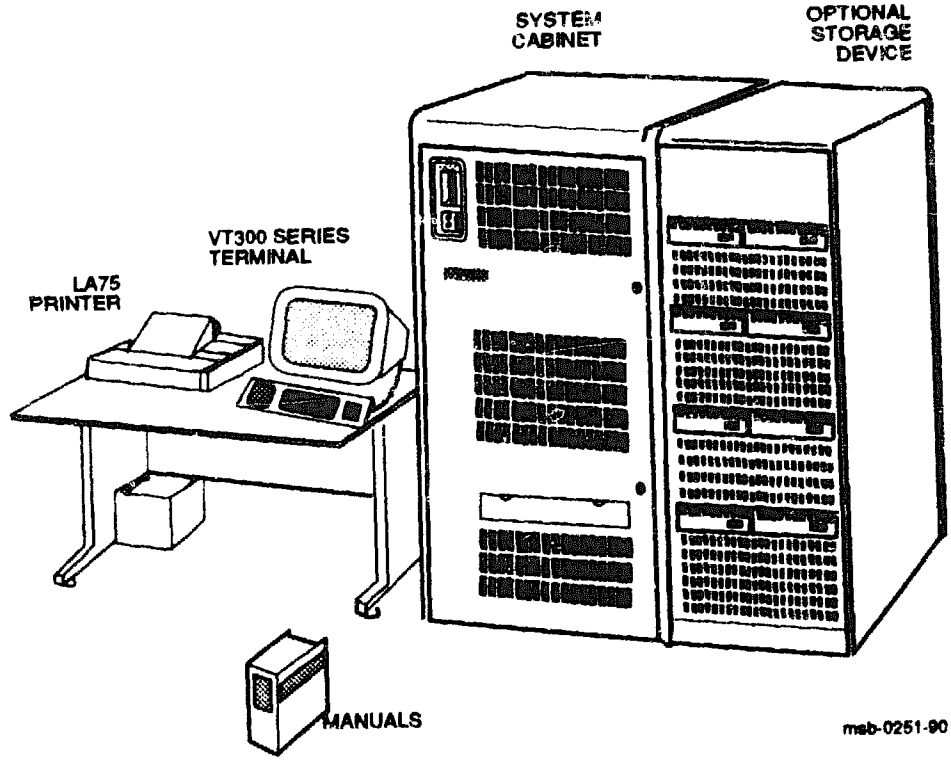


Table 1–1 System Components

Component	Function
System cabinet	Houses system components and optional storage
Console load device	Software distribution; stores and transfers data
Console terminal	Manages system and its resources
Console printer	Provides hardcopy of console transactions
Documentation	See the Preface for a full list of documentation related to VAX 6000 Model 500 systems
Storage cabinet	Provides additional storage capacity

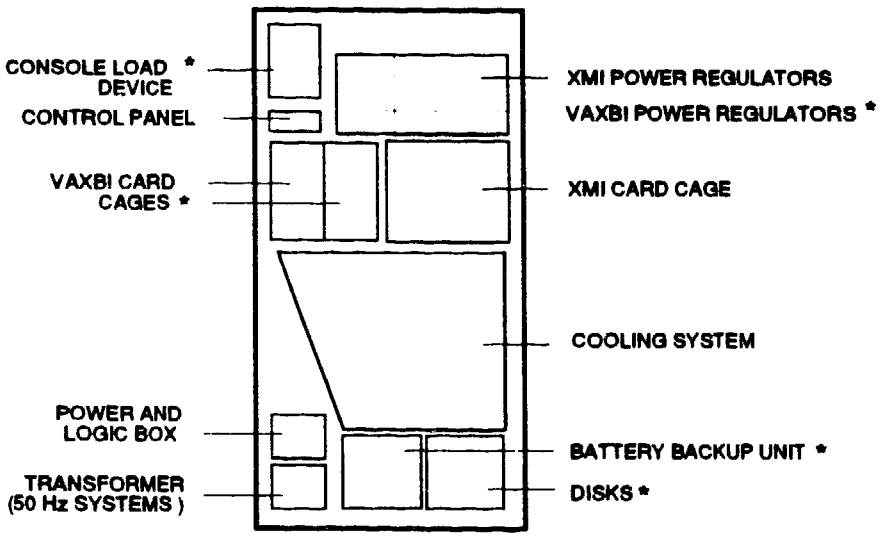
The VAX 6000 Model 500 components include:

- **The system cabinet** may house a console load device, the XMI card cage (which contains the processors, memories, and I/O adapters), two optional VAXBI card cages, optional disk drives, the control panel switches, status indicators, and restart controls.
- **The console load device** is used for installing operating systems, software, and some diagnostics. It may be an Ethernet-based compact disk server or an in-cabinet tape drive.
- **A storage cabinet** provides local storage and archiving capability.
- **The console terminal** is used for booting and for system management operations.
- **A system documentation kit** includes:
  - — *Installation Guide*
  - — *Owner's Manual*
  - — *Mini-Reference*

1.3 System Front View

The console load device, system control panel, and optional disk control panels are on the front of the system cabinet, accessible with the doors closed. With the front door open, Digital customer service engineers can access the power regulators, the XMI card cage and optional VAXBI card cages, the cooling system, and the optional battery backup unit.

Figure 1–3 System Front View



WARNING: The inside of the system cabinet is not designed to be accessed by the customer. The cabinet doors are to be opened only by Digital customer service engineers.

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

- Control panel
- XMI power regulators
- XMI card cage
- Cooling system
  One of the two blowers is visible from the front of the cabinet.
- Power and logic box

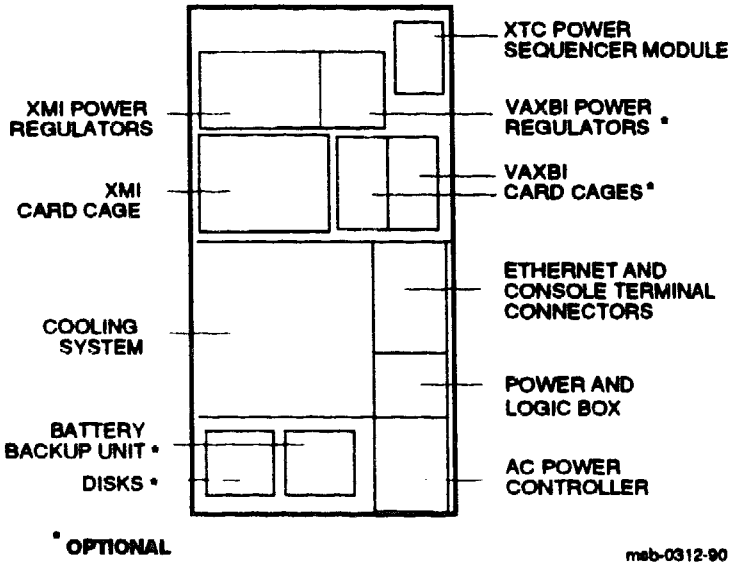
Optional components visible from the inside front of the cabinet include:

- Console load device
- VAXBI power regulators
- Two VAXBI card cages (one 12-slot channel)
- Battery backup unit
- Disks

1.4 System Rear View

With the rear door open, Digital customer service engineers can access the power sequencer module (XTC); the 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 System Rear View



WARNING: The inside of the system cabinet is not designed to be accessed by the customer. The cabinet doors are to be opened only by Digital customer service engineers.

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

- Power sequencer module (XTC) located on the back of the system control assembly and control panel unit
- XMI card cage (with I/O bulkhead unlatched)
- XMI power regulators
- I/O bulkhead space
  The panel covering the XMI and VAXBI areas is the I/O bulkhead panel and provides space for additional I/O connections.
- Ethernet and console terminal connectors
- Cooling system, with open grid over a blower
- Power and logic box
- AC power controller

Optional components visible from the inside rear of the cabinet include:

- VAXBI power regulators
- VAXBI card cages (with I/O bulkhead unlatched)
- Battery backup unit
- Disks

1.5 Supported Adapters

The system supports the following adapters: CIXCD, DEC LANcontroller 400 (DEMNA), DWMBB, KDM70, and the KFMSA.

Figure 1–5 Adapters

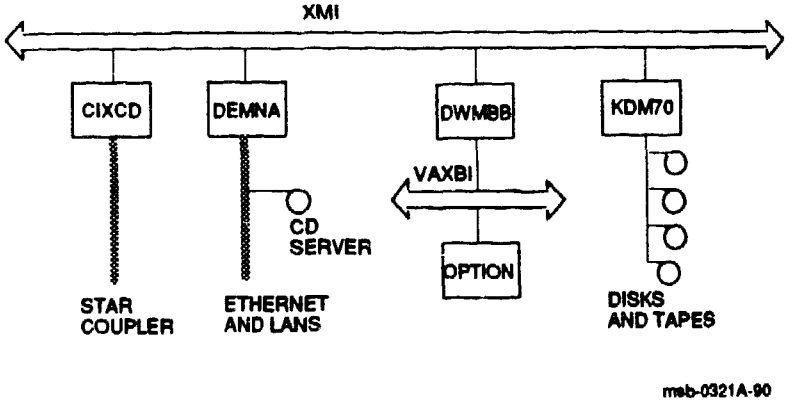
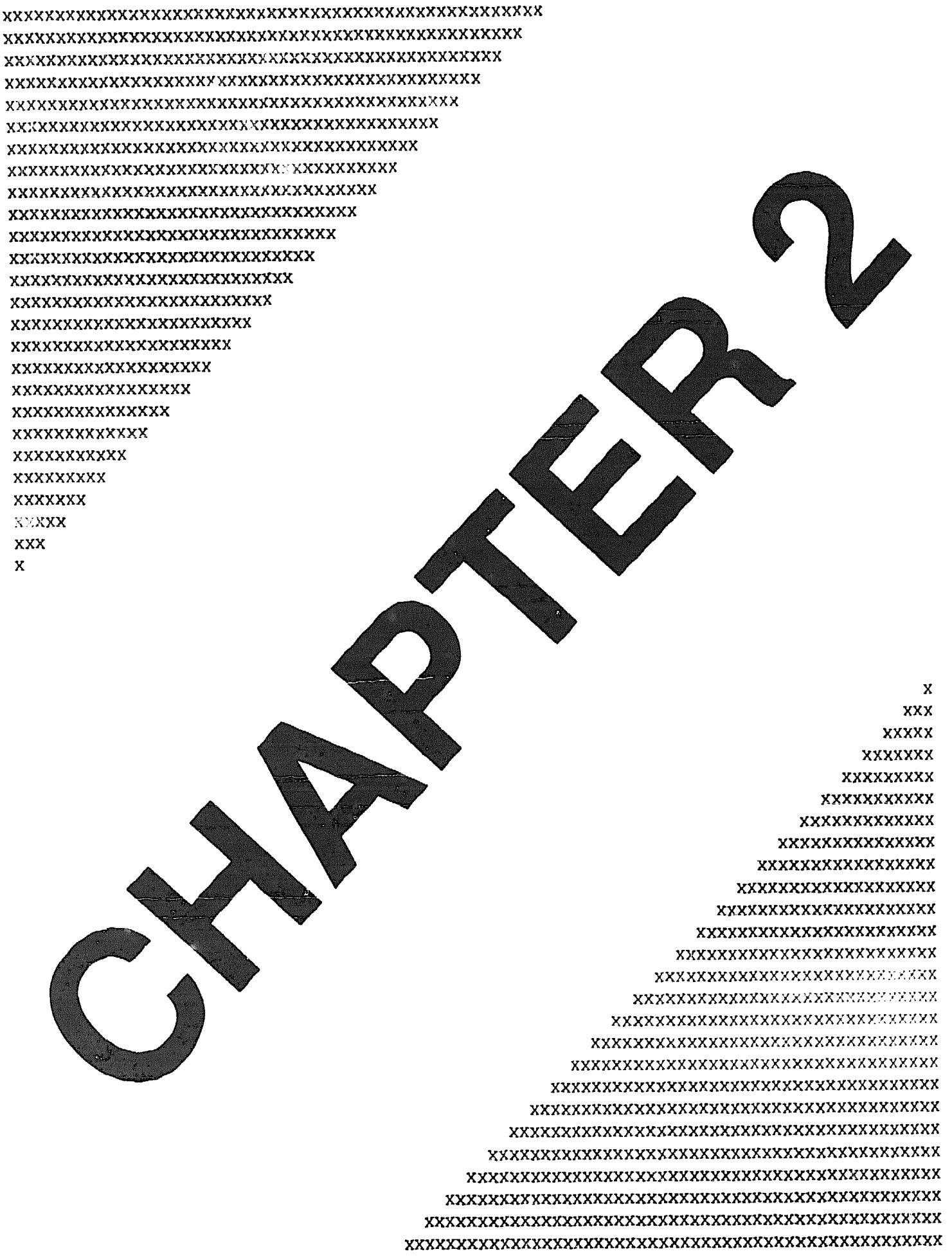


Table 1–2 lists some devices supported by the system. Some adapters have more than one module, requiring more than one slot on the XMI. For more information on adapters, see Digital's *Systems and Options Catalog* or the *VAX 6000 Platform Service Manual*.

Table 1–2 Adapters

Adapter	Std Opt'l	No. Slots	Function
CIXCD	O	1	CI port interface; connects the system to a Star Coupler.
DEMNA	S	1	Ethernet port interface; connects a system to a local area network.
DWMBB	O	1	VAXBI-to-XMI interface, a two-module set. The DWMBB/A is in the XMI card cage; the DWMBB/B is installed in the VAXBI card cage.
KDM70	O	2	Disk adapter; enables connection to RAxx disk drives.
KFMSA	O	1	Adapter; enables connection to RFxx and TFxx devices.


CHAPTER 2

2 KA65A CPU Module

This chapter describes the KA65A CPU module, the processor for the VAX 6000 Model 500 system.

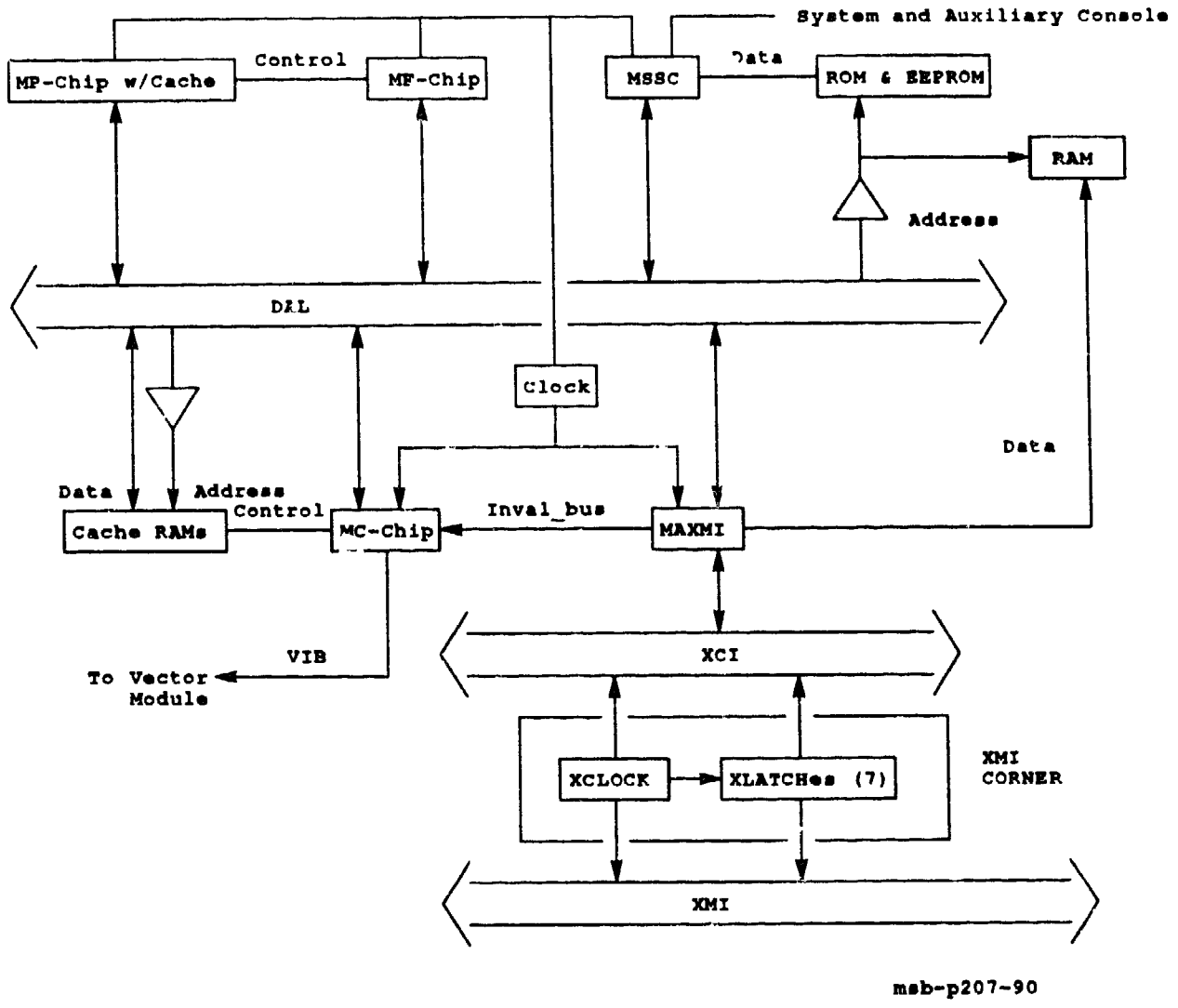
This chapter includes the following sections:

- Overview
- Block Diagram Description
- CPU Section
- Cache Memory
- System Support Chip
- XMI Interface
- Scalar/Vector Interaction
- KA65A CPU Module Registers
- Initialization, Self-Test, and Booting
- Interprocessor Communication Through the Console Program
- Error Handling

2.1 Overview

The KA65A CPU module is a single-board VAX CPU using five chips for the processor section and two chips for the MAXMI interface to the XMI bus.

Figure 2–1 KA65A CPU Module Block Diagram



The processor set consists of five chips:

- MP-chip (CPU-chip) (DC595)
- MF-chip (floating-point accelerator chip) (DC596)
- MC-chip (backup cache controller chip) (DC597)
- Clk-chip (clock distribution chip) (DC598)
- MSSC (system support chip) (DC224)

The MAXMI interface set consists of two chips:

- MDA (data path chip) and MCA (control chip)

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

- A VAX CPU that supports the 242-instruction VAX base instruction group, associated data types, and full VAX memory management.
- Support for 2 Gbytes of physical memory and 512 Mbytes of I/O space.
- 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 consisting of a 2-Kbyte primary cache and a 512-Kbyte backup cache.
- A writeback cache system that allows multiple read and write operations to be serviced which reduces XMI bus traffic.
- 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
- A bootstrap and diagnostic facility that provides:
  - Full microcode and macrocode power-up self-testing
  - Node initialization
  - Booting from various VAXBI and XMI devices
- An XMI interface that includes:
  - A writeback buffer with eight hexword entries.
  - Hexword cache fill logic that loads the backup cache with eight longwords of data on each cache miss.
  - XMI read/write monitoring logic

2.2 Block Diagram Description

The KA65A CPU module consists of four major blocks.

- CPU and floating-point accelerator
- MC-chip and backup cache
- System support chip
- XMI interface

Figure 2–1 shows the KA65A CPU module block diagram.

2.2.1 CPU and Floating-Point Accelerator

The MP-chip and the MF-chip cooperate as the CPU section of the KA65A CPU module. They implement the base instruction group of the VAX architecture.

The MP-chip is a pipelined CPU that provides the hardware and microcode needed to execute instructions, handle exceptions, and otherwise implement the VAX architecture. The MP-chip contains a 64-entry, fully associative translation buffer where both process- and system-space mappings are kept.

The MP-chip also includes the primary cache, which is 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 MF-chip is a pipelined execution unit that enhances the computation phase of floating-point and certain integer instructions. The MF-chip receives operands from the MP-chip, computes the results, and passes the result and status back to the MP-chip to complete the instructions.

2.2.2 MC-Chip and Backup Cache

The backup (or secondary cache) uses the MC-chip which contains the tag store and control logic for a 512-Kbyte backup cache using 64-Kbyte x 4 data RAMs on the module. The backup cache is direct-mapped and contains 4-Kbyte tags organized to provide a hexword fill size and a 4-hexword block size. Access to data in the backup cache takes three cycles longer than access to the primary cache.

The MC-chip includes a special address interface to check XMI write/read address against data in both caches. If the address corresponds to cached data, the MC-chip requests an invalidate or writeback cycle to purge the cached data.

The MC-chip also provides the operand and control interface between the KA65A CPU module and an optional vector module. This MC-chip issues vector instructions over the vector interface bus (VIB) to the vector module, which then executes the instructions, including all memory references needed to load or store vector registers.

2.2.3 System Support Chip

The MSSC system support chip provides support functions to the CPU in a system environment. Included in the MSSC is support for external ROM /EEPROM, 1 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.

2.2.4 XMI Interface

The XMI interface is between the DAL bus and the XMI bus and consists of the XMI Corner (one XCLOCK chip and seven XLATCH chips) and the MAXMI chipset (one MCA-chip and one MDA-chip).

2.3 CPU Section

The CPU section of the KA65A CPU module consists of the MP-chip and the MF-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
- Primary Cache
- Floating-Point Accelerator

2.3.1 Data Types

The KA65A 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.

2.3.2 Instruction Set

The KA65A 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 KA65A 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 KA65A 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.

2.3.3 Memory Management

The KA65A 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. The KA65A CPU module supports 25-bit page frame numbers (PFNs). 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).

2.3.3.1 Translation Buffer

The MP-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 31-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 bits <22:0> of the 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. Bits <24:23> of the PFN are ignored.

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, 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.

2.3.3.2 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 MP-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 a 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. An MFPR instruction to them causes a reserved operand fault.

2.3.4 Exceptions and Interrupts

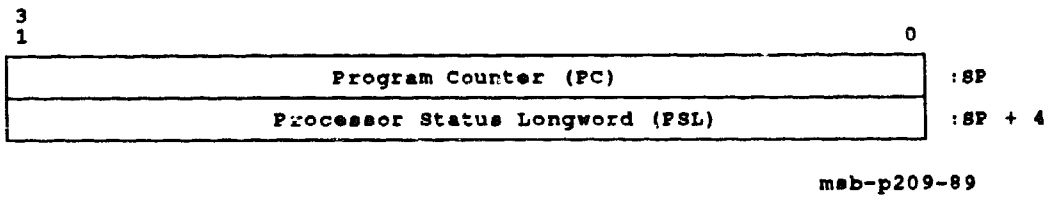
Sometimes events require execution of software routines outside the explicit flow of instructions.

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 2–2.

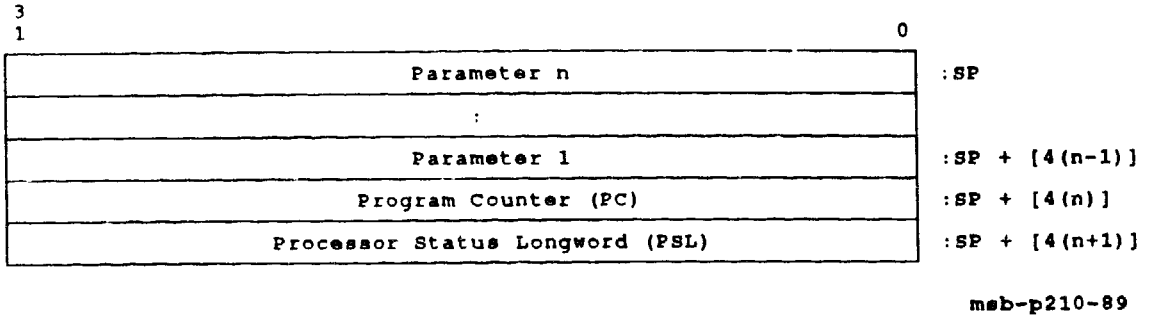
Figure 2–2 Minimum Stack Frame



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 2–3.

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

Figure 2–3 Large Stack Frame



2.3.4.1 Interrupts

A subset of the 31 VAX interrupt priority levels (IPLs) is implemented by the MP-chip. When an interrupt request is generated, the MP-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 MAXMI 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 2–1 lists the IPLs implemented by the KA65A CPU module.

Table 2–1 KA65A 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	MAXMI-detected errors in XBER	60
	MAXMI-detected errors in XBEER	60
	Vector module hard errors (VINTSR<2>), VINTSR<7>	60
	MC-chip-detected errors in BCSTS	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
	Parity error on an XMI cycle (XBER<23> set)	54
	Corrected XMI CNF error (XBER<27> set if XCR<6> not set)	54
	Correctable read data error (XBER<19> set if XCR<5> not set)	54
	Vector module soft errors (VINTSR<1> set)	54
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

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

Table 2–1 (Cont.) KA65A CPU Module Interrupts

Interrupt Level (hex)	Interrupt Condition	SCB Vector (hex)
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.

2.3.4.2 Exceptions

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 MP-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.

Table 2–2 lists the KA65A CPU module-specific instances of the seven classes of exceptions in the VAX.

Table 2–2 KA65A 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)	MP-cache read error abort
	DAL read error abort
	DAL write error abort
	MF-chip status error abort
	Translation buffer status error abort
	Internally detected inconsistency abort

2.3.4.3 Unique Exceptions

The following exceptions are unique to the MP-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 2–4 shows the arithmetic exception stack frame.

The exceptions are reported in the manner shown in Table 2–3.

Figure 2–4 Arithmetic Exception Stack Frame

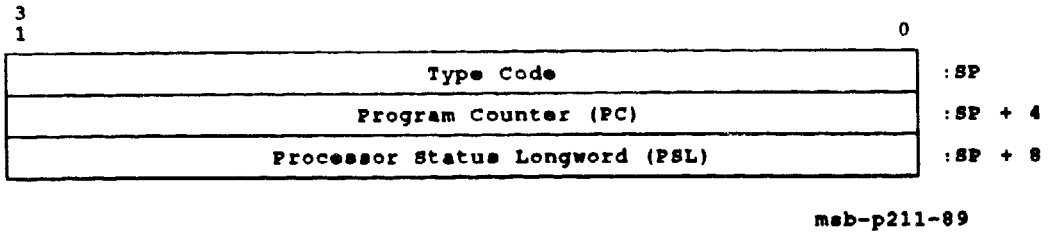


Table 2–3 Arithmetic Exceptions Type Codes

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

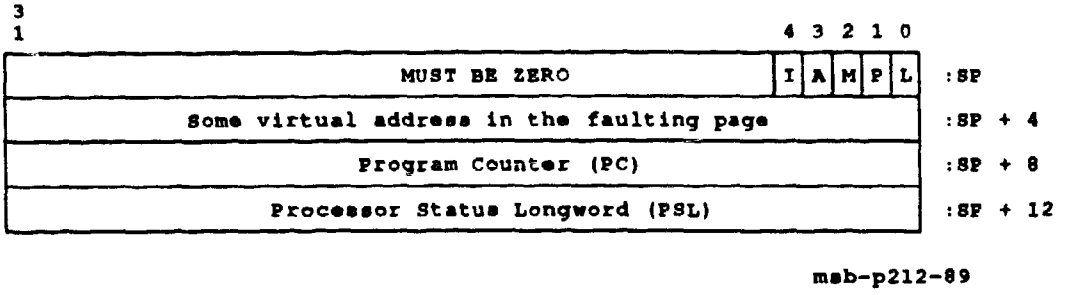
Memory management exceptions are detected during a memory reference and are always reported as faults. The memory management exceptions are listed in Table 2–4

Table 2–4 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 2–5.

Figure 2–5 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 MP-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 2–6 and the stack frame's parameters listed in Table 2–5.

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 2–7. In this case, PC is the opcode of the emulated instruction.

Figure 2–6 Emulated Instruction Trap

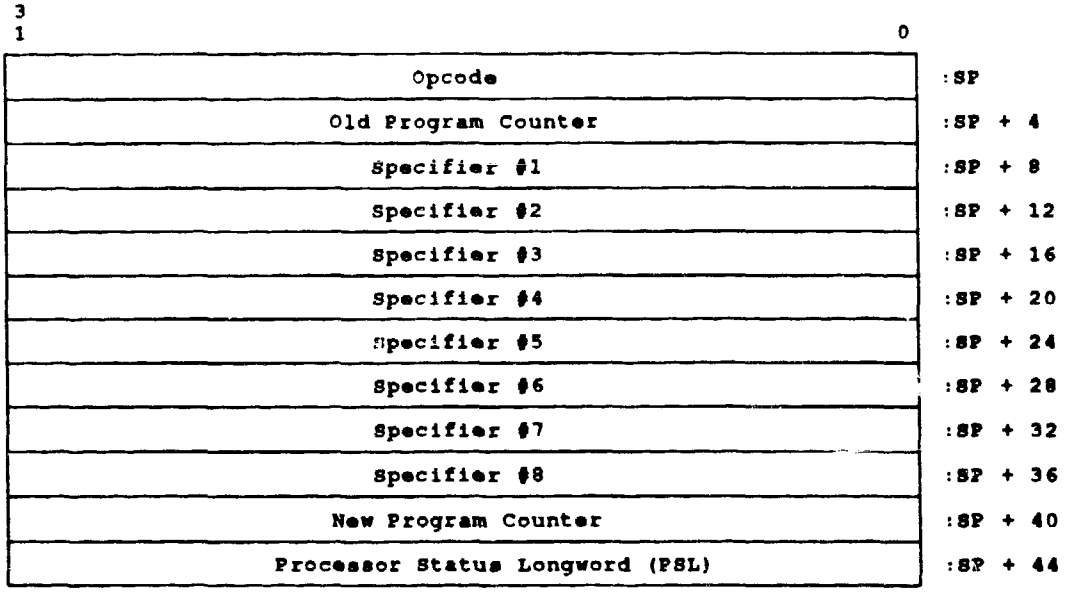
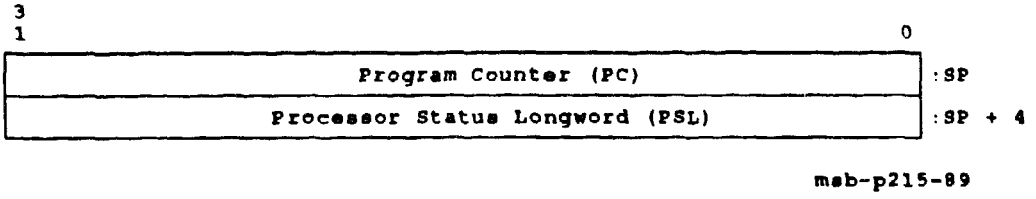


Table 2–5 Emulated Instruction Trap Stack Frame Parameters

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 2–7 Emulated Instruction Fault



Vector Module Disabled Fault

A vector module disable fault is initiated through SCB vector 68 (hex) if the MP-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 3.9.4.3).

Machine Check Exceptions

A machine check exception is reported through SCB vector 04 (hex) when the MP-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 2–8 and its parameters are described in Table 2–6. Table 2–7 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 2–8 Machine Check Stack Frame

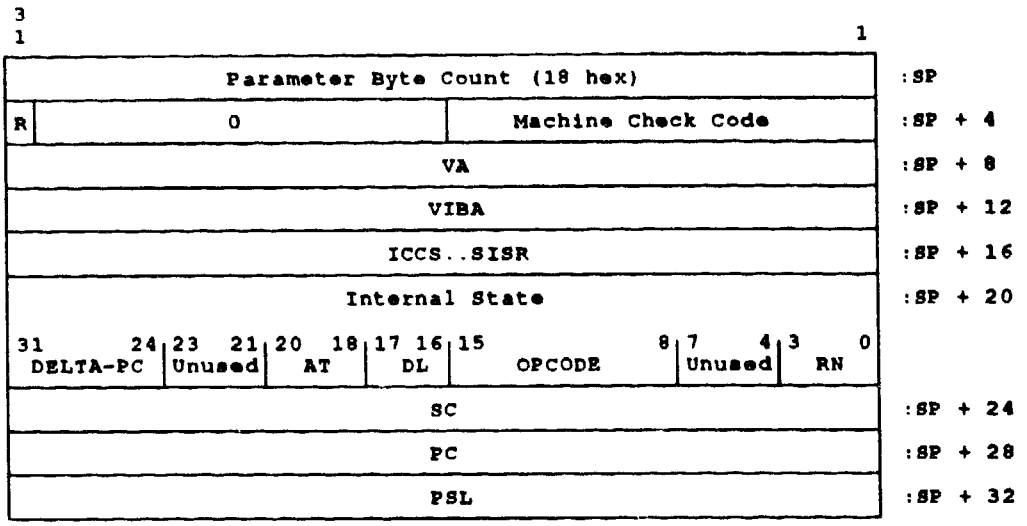


Table 2–6 Machine Check Stack Frame Parameters

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 NSCSR<20> (Unlock Write Pending, UWP.)
Machine check code (bits<15:0>)	Table 2–7 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 MP-chip and is not used by software to make recovery decisions.

Table 2–6 (Cont.) Machine Check Stack Frame Parameters

Parameter	Description
	Unused, bits<23:21>
	AT, bits<20:18> — The current setting of the E-box (the MP-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, MP-chip error
100	Unassigned, MP-chip error
101	Address
110	Variable bit
111	Branch

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 2–7 Machine Check Codes

Code (hex)	Mnemonic	Description	Restart Condition
01	MCHK_FP_PROTOCOL_ERROR	Protocol error during MF-chip operand/result transfer.	(R=1).(FPD=0).(UWP=0)
02	MCHK_FP_ILLEGAL_OPCODE	Illegal opcode detected by MF-chip.	(R=1).(FPD=0).(UWP=0)
03	MCHK_FP_OPERAND_PARITY	Operand parity error detected by MF-chip.	(R=1).(FPD=0).(UWP=0)
04	MCHK_FP_UNKNOWN_STATUS	Unknown status returned by MF-chip.	(R=1).(FPD=0).(UWP=0)
05	MCHK_FP_RESULTS_PARITY	Returned MF-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) (See Section 2.11.4.9)
0C	MCHK_UNKNOWN_IBOX_TRAP	Undefined trap code produced by the I-box (the MP-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	MP-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
15	MCHK_ERROR_ISTREAM	Error on I-stream read	((R=1)+(FPD=1)).(UWP=0).(TR2=0)

Where:

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

2.3.4.4 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 MP-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

The hardware restart sequence is as follows:

1 The MP-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 2–8.

4 The MP-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)
PC	E004 0000 (hex)
MAPEN	0
ACCS<31>	0
SCCS	0 after reset only (halt code = 3)
SISR	0 after reset only (halt code = 3)

State	Initialized Value
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 2–8 Halt Codes

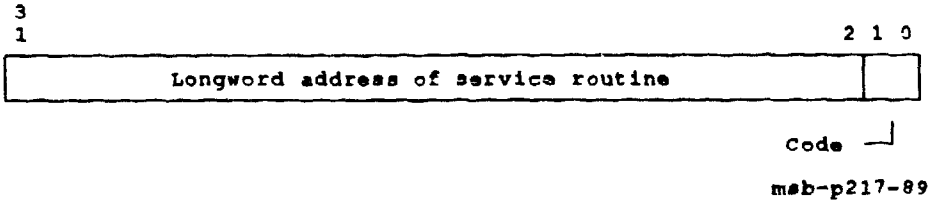
Code (hex)	Mnemonic	Description
02	ERR_HLTPIN	CTRL/P, break, or external halt
03	ERR_PWRUP	Initial power-up
04	ERR_INTSTK	Interrupt stack not valid during exception processing
05	ERR_DOUBLE	Machine check during exception processing
06	ERR_HLTINS	HALT instruction executed in kernel mode
07	ERR_ILLVEC	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 MP-chip. This can only happen during power-up.

2.3.5 System Control Block

The system control block (SCB) contains pages of vectors for servicing traps, faults, software interrupts, and exceptions. IPR17, the System Control Block Base Register, points to the SCB. Since SCBB contains a 32-bit physical address, the SCB can reside anywhere in memory space.

An SCB vector is an aligned longword in the SCB. The MP-chip's microcode dispatches interrupts and exceptions through the SCB vector, shown in Figure 2–9.

Figure 2–9 System Control Block Vectors



Bits <31:2> of each vector supply the virtual address of the 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 – Unimplemented; results in a console error halt.

11 – Unimplemented; results in a console error halt.

Table 2–9 shows the SCB layout.

Table 2–9 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	–	–	Compatibility mode in other VAXes
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

Table 2–9 (Cont.) System Control Block Layout

Vector (hex)	Name	Type	Number of Parameters	Notes
8C	Software level 3	Interrupt	None	Usually used for process scheduling
90 – BC	Software levels 4 through 15	Interrupt	None	
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

2.3.6 Process Structure

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

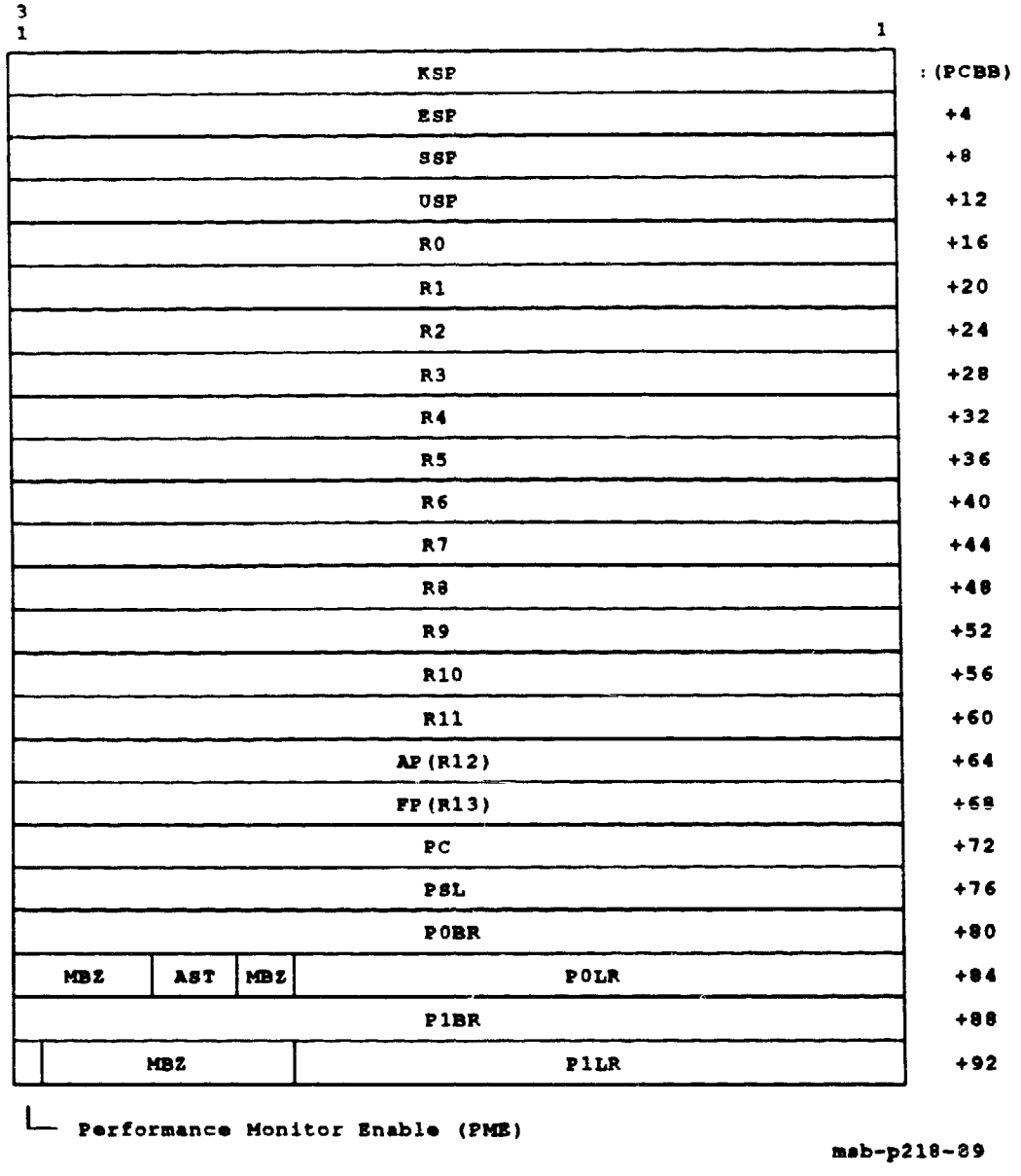
The physical address of the current PCB is changed by writing to the Process Control Block Register. PCBB contains a 32-bit physical address so that the PCB may be located anywhere in memory space. The LDPCTX instruction loads a process context from the PCB as described in the *VAX Architecture Reference Manual*, except for the PME (bit<31>) of PCB + 92, shown in Figure 2–10. 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*.

2.3.7 Primary Cache

The primary cache is located in the MP-chip. It works with the backup (or secondary) cache to boost system performance. The caches are discussed in Section 2.4.

Figure 2–10 Process Control Block



msb-p218-89

2.3.8 Floating-Point Accelerator

The KA65A CPU module includes the MF-chip floating-point accelerator that enhances the performance of floating-point and certain integer calculations.

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

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

The MF-chip is enabled/disabled by using the MF-Chip Present bit (Accelerator Control and Status Register, ACCS<1>). When the MF-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 MP-chip's microcode.

The MF-chip supports the following data types:

- F_floating
- D_floating
- G_floating

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

When the MF-chip receives the last operand for an instruction, it begins the computation of the result. In parallel, the MP-chip completes any instruction setup, such as parsing a destination specifier. The MP-chip next requests the result from the MF-chip and stalls until the result is returned. The MP-chip then stores the result where the destination

specifier indicates, either in a GPR or memory, and sets the PSL condition codes.

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

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

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

CAUTION: The MF-chip accepts memory operands in I/O space. Since the MF-chip executes longword-length integer multiply instructions, it might not produce correct results or might report operand parity errors if it is enabled during the execution of the console code from the boot ROM.

To avoid this, disable the MF-chip on any console entry by writing a zero to ACCS<1>, causing the MP-chip to execute the integer instructions in microcode and to invoke a reserved instruction fault for the floating-point instructions.

2.4 Cache Memory

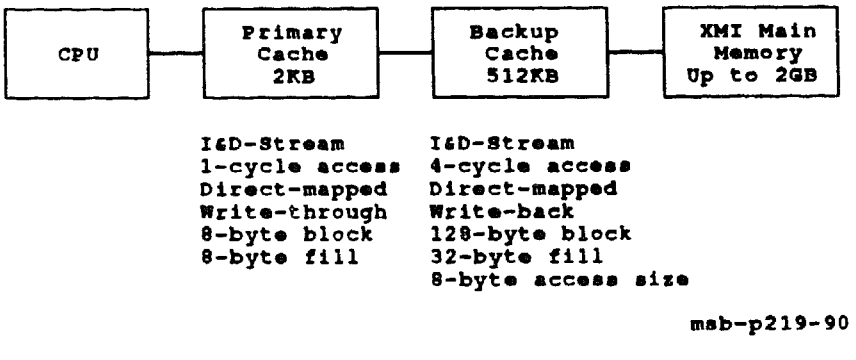
The KA65A CPU module has a two-level cache to maximize CPU performance. The first-level cache is a 2-Kbyte primary cache (P-cache) contained in the MP-chip. The second-level cache is a 512-Kbyte backup cache (B-cache) consisting of the MC-chip, for tag storage and cache control, and 32 64-K x 4 static RAMs, for data storage.

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 quadword data blocks. 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 512-Kbyte B-cache is direct mapped, with a hexword fill and a 4-hexword allocate (block) size. The B-cache is read allocate, write-allocate, and writeback. The MC-chip implements the B-cache tag store and provides control for the cache RAMs, which contain data, data parity, and ECC codes. The backup cache tag store contains 4096 tag entries and each tag entry contains a 12-bit tag, four valid bits, four dirty bits, and two parity bits. Dirty bits are defined as bits that have been written by the backup cache. The data RAMs are organized into 64-K locations of eight data bytes, with a parity bit for each data byte and 48 ECC bits, 6 bits for each data byte.

Figure 2–11 shows the memory hierarchy.

Figure 2–11 CPU Module Cache Memory



2.4.1 Writeback Cache

In a write-through cache, the processor writes to both the B-cache and memory simultaneously. In a writeback cache, data is written to the B-cache only. Since it is not written into memory, the amount of data on the XMI bus is reduced significantly.

To implement the writeback cache method, the KA65A processor uses a writeback protocol that includes the concept of ownership. The memory controller keeps a lock bit and an ownership bit for each block of memory corresponding to the 32-byte subblock B-cache. The memory controller uses the ownership bit to allow only one system component write or interlock access to a particular block at a time.

The KA65A processor issues an Ownership Read (OREAD) command to a hexword memory block when it has a cache miss for a location that is likely to be written, since the processor was actually performing a write to the location or a read with the "modify intent flag" signal asserted. If the memory block is not currently owned, the processor gets ownership.

Each cache subblock of the MC-chip has both a valid bit and a "dirty" bit. The KA65A CPU module's writeback protocol sets the dirty bit to indicate that the cache subblock is owned by this processor and is not accessible by any other processor or I/O adapter. At some later time, the cache block with its updated data will be written back to memory and ownership of the block will be given up. The updated data will then be available to all other system components.

Processor memory writes to cache subblocks that do not have the dirty bit set have to wait until ownership of the memory block can be obtained. The MC-chip uses the dirty bit for both ownership and interlock access.

The MC-chip monitors all memory accesses on the XMI for other system components wanting to gain read, write, ownership, or interlock access to memory. When other system components have read, write, ownership, or interlock requests for a particular memory block that is dirty, the MC-chip writes the cache subblock back to memory. This is called a writeback. The MC-chip then clears the dirty bit, giving up ownership of the block. On write, ownership, or interlock requests, the MC-chip also invalidates the subblock in the B-cache and P-cache. On read requests, the MC-chip keeps a read-only copy of the subblock. On write (ownership) or interlock requests, the MC-chip invalidates the cache subblock in the B-cache and P-cache if a read-only copy of the block is being cached.

2.4.2 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 MAXMI interface uses the Inval-Bus (which connects the MAXMI and the MC-chip) to forward all invalidate requests it sees on the XMI to the MC-chip.

When the MAXMI interface sees a main memory write or read request on the XMI bus, it presents the address on the Inval-Bus and makes an Inval-Bus request to the MC-chip. These addresses are stored in a 4-entry invalidate queue in the MC-chip. The MC-chip uses otherwise idle cycles to look up the address in the B-cache to determine if any data is being cached for the address.

If the MC-chip finds data is being cached at the specified address, the operation of the MC-chip is determined by the type of invalidate request and the status of the cached data in the B-cache, as shown in Table 2–10. By knowing the invalidate type and cache state, a filtering mechanism for invalidates is used that avoids using the DAL except when an invalidate address is being cached.

Table 2–10 Invalidate Operations

Invalidate Type	Cache State	MC-Chip Action
Read access	Clean	Discard request.
Read access	Dirty	Write data back; keep a clean copy of data in the B-cache.
Write or ownership access	Clean	Invalidate data in both caches.
Write or ownership access	Dirty	Write data back; invalidate data in both caches.

2.4.3 Primary Cache

The primary cache (P-cache) resides in the MP-chip. The P-cache is arranged in 64 rows with four data quadwords and four tag entries per row, as shown in Figure 2–12. Each physical address is logically subdivided as shown in Figure 2–13. The organization of each tag entry is shown in Figure 2–14. The data array organization of each quadword is shown in Figure 2–15.

Figure 2–12 Primary Cache Organization

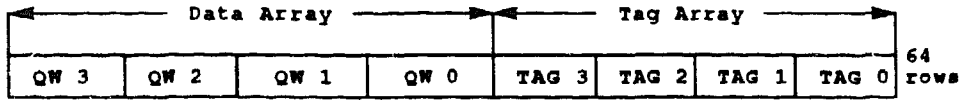
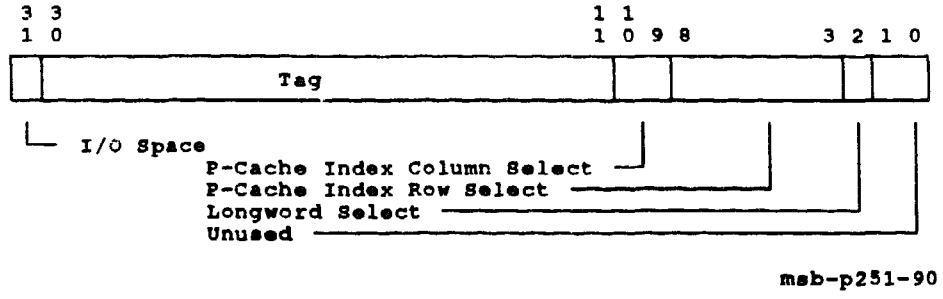
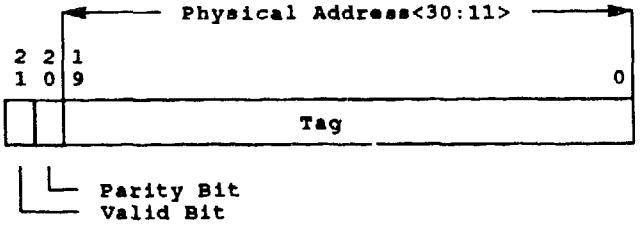


Figure 2–13 Primary Cache Physical Address



Bits <31:2>	The physical addresses supplied to the P-cache.	
	Bit <31>	Specifies I/O space. I/O space addresses are not cached.
	Bits <30:11>	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.

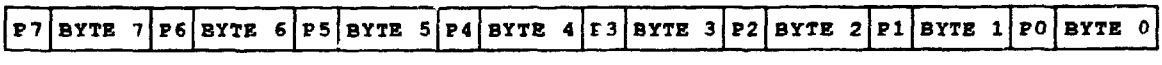
Figure 2–14 Tag Entry Organization



The tag consists of:

Bit <21>	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 <20>	The tag parity, which is parity computed by the P-cache over the 20 tag bits.
Bits <19:0>	The tag, bits <30:11> of the physical address.

Figure 2–15 Quadword Data Array Organization of the Primary Cache



Each P-cache entry consists of one quadword. Parity information is maintained separately for each byte. When an error is detected, the P-cache is automatically disabled and error information is read from the Primary Cache Status Register (PCSTS) and the Primary Cache Error Address Register (PCERR). The error is then corrected and PCSTS is cleared.

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.

Normal operational control of the P-cache is with writes to PCSTS, which has bits to enable the use of the P-cache and allow it to be flushed.

When an error is detected in the P-cache or in a DAL transaction, the P-cache latches error information. This information is available by reading PCSTS and PCERR. Bits in PCSTS can be written to tell the P-cache that error information has been read, enabling the P-cache to detect subsequent errors.

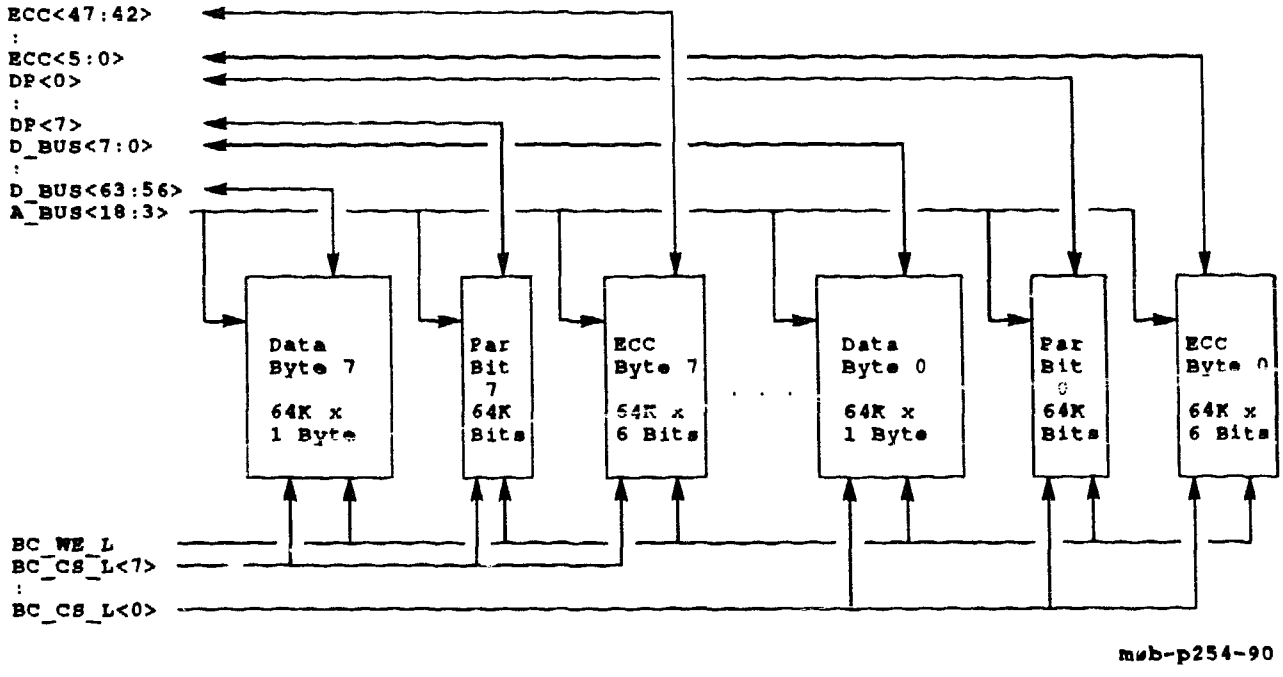
To maintain P-cache consistency, it is important to use the proper sequence of events for reenabling a tag store that has been disabled. Any state change to the CPU's P-cache must be reflected in the MC-chip copy of the primary cache tag store. During normal operation, changes are updated automatically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

Cache and memory error recovery are done with the P-cache disabled and the backup cache in error transition mode (ETM). To maintain cache coherency, the primary cache should be disabled when enabling the backup cache. Once the backup cache is enabled, the primary cache may be flushed and disabled.

2.4.4 Backup Cache

The backup cache (B-cache) is a 512-Kbyte cache, direct mapped, quadword access size, with a hexword fill (subblock) size and a 4-hexword allocate (block) size. The backup cache is read-allocate, write-allocate, and writeback. The backup cache sends quadwords of data to the primary cache. The backup cache is shown in Figure 2–16.

Figure 2–16 Backup Cache Organization



mob-p254-90

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.
- ECC<47:0>. 48 bits of data ECC. Six bits for each data byte.
- A_BUS<18:3>. Address for the cache RAMs; used as the cache index.
- BC_WE_L. Write enable. The MC-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 MC-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 written with returned data. During a memory write transaction, the MP-chip drives BM_L<7:0> as a byte mask for the data. The MC-chip asserts the corresponding BC_CS_L<7:0> lines so that only the intended bytes are written.

2.4.4.1 Backup Cache RAM Addressing

The VAX physical address, as used by the B-cache RAMs, is shown in Figure 2–17. 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 2–18 shows MC-chip tag store addressing.

Figure 2–17 Backup Cache RAM Addressing

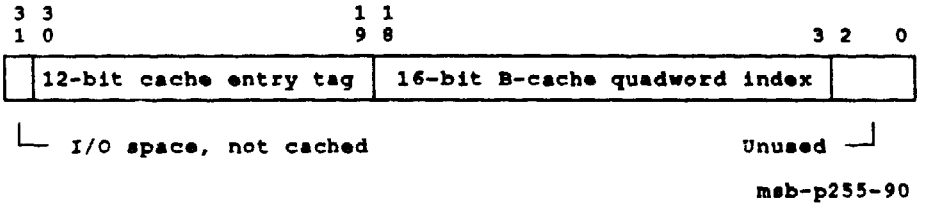
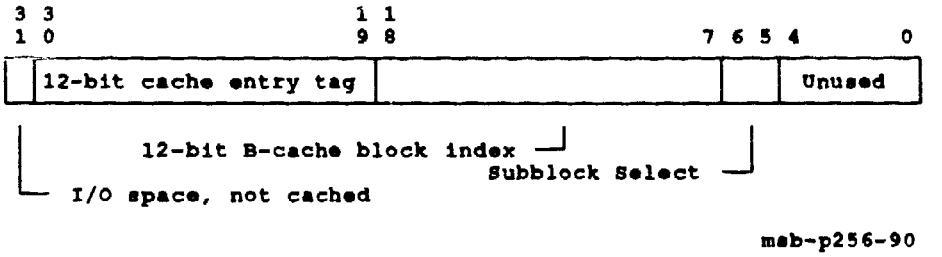


Figure 2–18 MC-Chip Tag Store Addressing

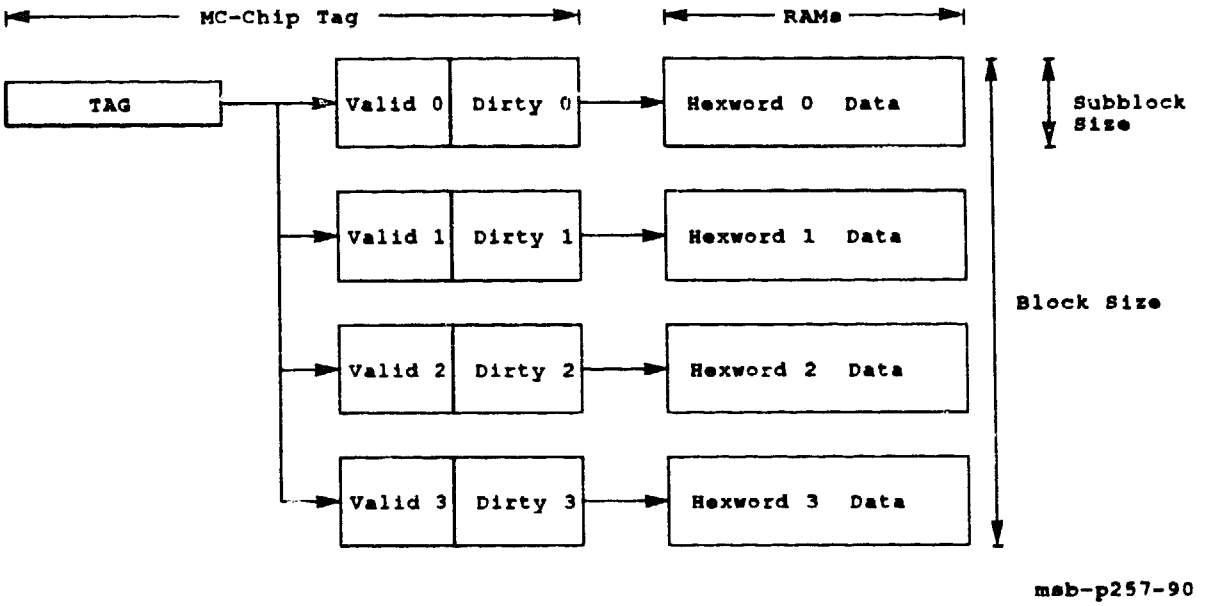


2.4.4.2 Backup Cache Tag Store Organization

The tag store is organized with one tag and four valid/dirty bit pairs corresponding to each four-hexword block of the cache. Each valid/dirty bit pair corresponds to one hexword subblock. When a cache tag miss occurs on a read, a block is allocated (which may require the deallocation of a block already in the cache), 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 2–19 shows the backup cache tag store organization.

Figure 2–19 Backup Cache Tag Store Organization



2.4.4.3 Backup Cache Internal Processor Registers

Several MC-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 MC-chip internal processor registers are:

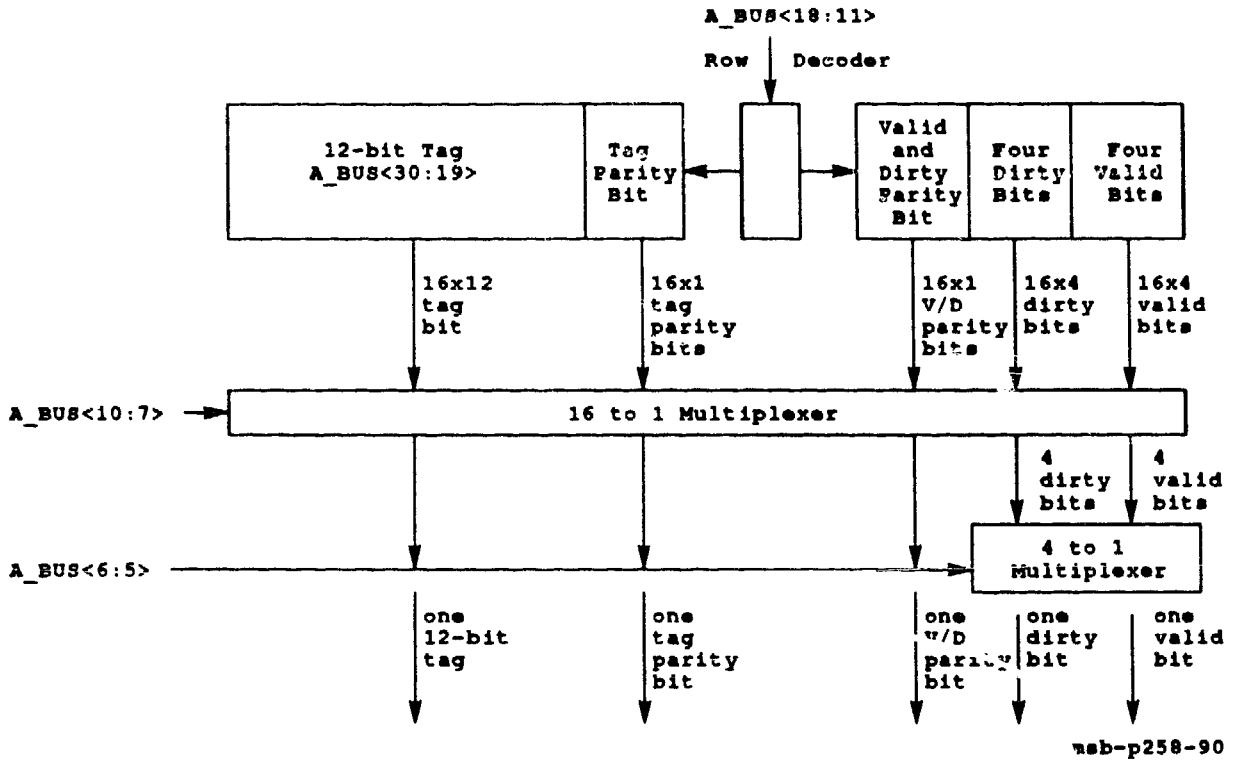
- Backup Cache Index, BCIDX (IPR112)
- Backup Cache Status, BCSTS (IPR113)
- Backup Cache Control, BCCTL (IPR114)
- Backup Cache Error Address, BCERA (IPR115)

- Backup Cache Tag Store, BCBTS (IPR116)
- Backup Cache Deallocate Tag, BCDET (IPR117)
- Backup Cache Error Tag, BCERT (IPR118)

2.4.4.4 Backup Cache Tag Store Block Diagram Description

Figure 2–20 is the backup cache tag store block diagram. Figure 2–21 shows the backup cache tag store addressing using a physical address.

Figure 2–20 Backup Cache Tag Store Block Diagram



A tag store entry (tag, valid bits, dirty bits, and parity bits) 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.

Figure 2–21 Backup Cache Tag Store Addressing Using Physical Address

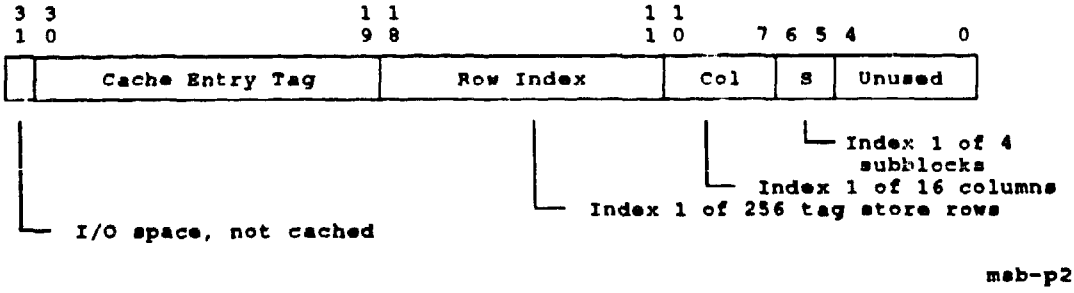
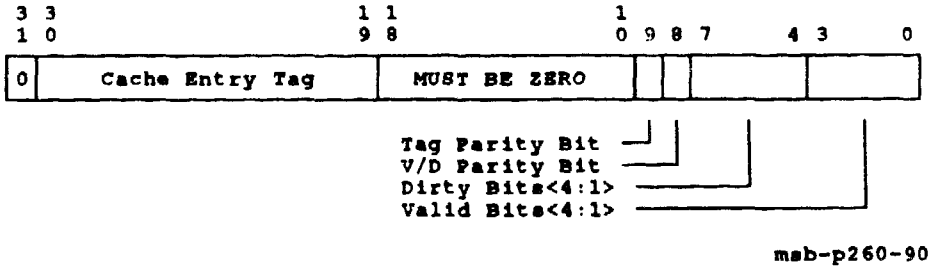


Figure 2–22 D_BUS Format to Access BCBTS



The MC-chip responds to an IPR read of the backup cache tag store register by driving the D_BUS as shown in Figure 2–22. 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 MC-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.

The cache entry tag contains the tag portion of the tag store entry, which is the high order of the physical address bits. The parity bit is for odd parity, as calculated on the tag. For more information, refer to the Backup Cache Tag Store Register (BCBTS).

2.4.4.5 Using Backup Cache Registers

The 10 MC-chip registers control the backup cache tag store and the MC-chip copy of the primary cache tag store. Operating system software reads these internal processor registers with Move To Processor Register (MTPR) instructions 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 MC-chip copy of the primary cache tag store is done with writes to the Backup Cache Control Register (BCCTL). The backup cache is flushed during normal operation. At power-up, the backup cache tag store is initialized 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. Both caches are left disabled.

Diagnostics

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

Backup Cache Status Register (BCSTS), IPR113

BCSTS is read using an IPR read. All bits are writable by the hardware, although some bits can be cleared using an IPR write.

BCSTS<25:22> (DAL CMD), BCSTS<26> (DMG L), BCSTS<27> (SYNC L), BCSTS<28> (AC PARITY), and BCSTS<29> (OREAD PENDING) are loaded at every transaction. In addition, BCSTS<20> (IBUS CYCLE), BCSTS<21> (IBUS CMD), BCSTS<15> (BTS HIT), BCSTS<16> (BTS COMPARE), BCSTS<17> (PPG), and BCSTS<19:18> (BTS PARITY) are loaded every time the backup tag store is accessed. This occurs for all memory read and write DAL transactions, DMA cache fills, Inval-Bus request lookups, invalidates, and writebacks from the DAL.

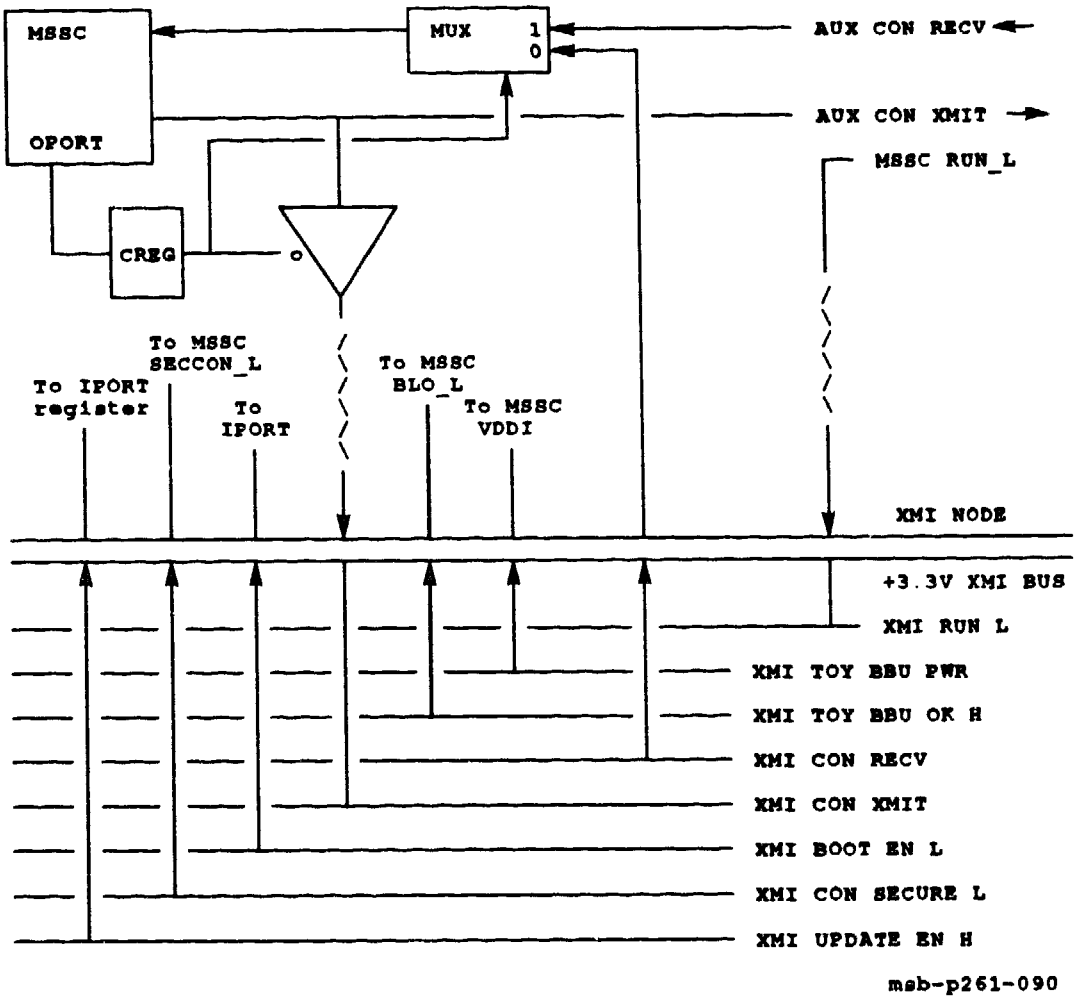
The BCSTS<0> (ERR SUMMARY) bit is set when any error bit (BCSTS<1> (BTS TPERR), (BCSTS<2> (BTS VDPERR), (BCSTS<5:4> (IPERR), (BCSTS<6> (FILL ABORT), (BCSTS<7> (AC PERR), or (BCSTS<8> (SECOND ERR) is set. The Backup Cache Status Register, Backup Cache Error Address Register, and Backup Cache Error Tag Register loads are also disabled to allow the MP-chip to examine the state of the MC-chip when the error occurred. If the MC-chip is not in ETM, all errors that occur within the same error-causing transaction are logged into the Backup Cache Status Register. If the MC-chip is in ETM, unrecoverable subsequent errors set BCSTS<8> (SECOND ERR). Other bits are not affected.

All error bits are cleared through an IPR write of the status register or during a reset. When all error bits are cleared, BCSTS<0> (ERR SUMMARY) is cleared and the MC-chip exits ETM.

2.5 System Support Chip

The system support chip (MSSC) provides functions to support the VAX 6000 Model 500 system environment, including the VAX console. The VAX console is the combined hardware-software interface that controls the system at power-up and when the system is halted.

Figure 2–23 Control Panel Connections Including Console Lines



2.5.1 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 2–23. 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:

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

2.5.1.1 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>, NSCSR0<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.

Each KA65A CPU module can drive and receive from the system console device. Only the boot processor communicates with the system console; the other KA65A CPU modules communicate with an auxiliary console through a serial transmit/receive pair that attaches to the I/O section of the KA65A CPU module.

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

2.5.1.2 CTRL/P Detection

The console serial line unit of the MSSC recognizes CTRL/P as a BREAK condition. When the MSSC detects a valid BREAK condition, RXDB<RBR> is set. If halts are enabled (XMI HALT EN L asserted, a function of a control panel key switch) when RXDB<RBR> sets, the CPU halts and transfers program control to ROM location E004 0000 (hex). RXDB<RBR> is cleared by reading RXDB.

2.5.1.3 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.

2.5.1.4 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).

2.5.2 Time-of-Year Clock and Timers

The KA65A 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.

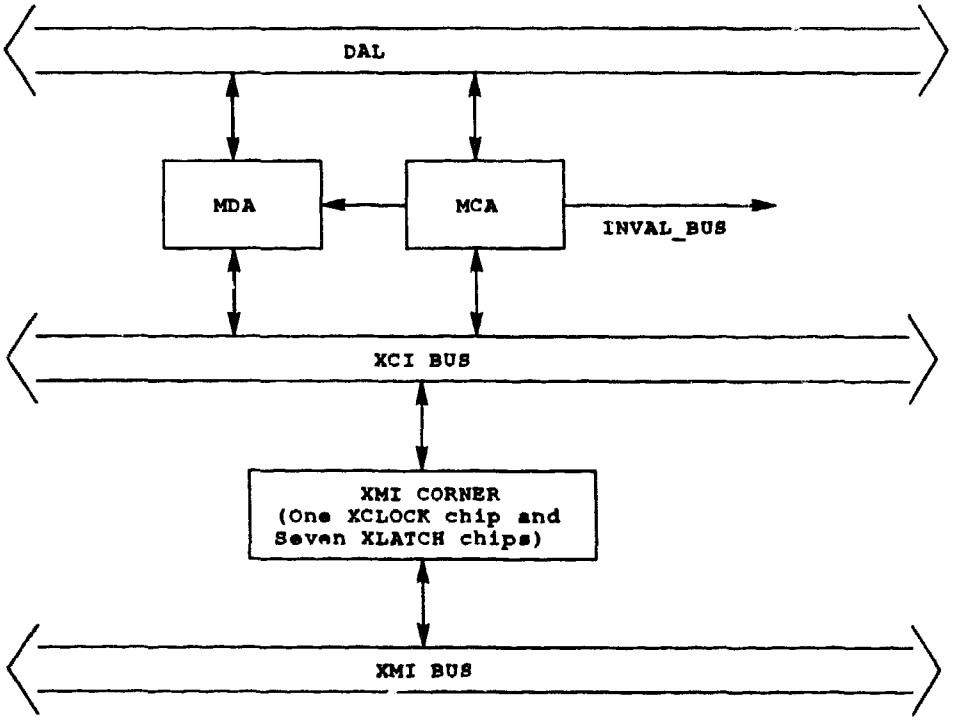
The time-of-year clock and timers include these registers (addresses are in hexadecimal):

- Time-of-Year Clock Register, TODR (IPR27)
- Interval Clock Control and Status Register, ICCS (IPR24)
- Timer Control Registers 0 and 1, TCR0 and TCR1 (E014 0160 and E014 0170, respectively)
- Timer Interval Registers 0 and 1, TIR0 and TIR1 (E014 0164 and E014 0174, respectively)
- Timer Next Interval Registers 0 and 1, TNIR0 and TNIR1 (E014 0168 and E014 0178, respectively)
- Timer Interrupt Vector Registers 0 and 1, TIVR0 and TIVR1 (E014 016C and E014 017C, respectively)

2.6 XMI Interface

The KA65A CPU module uses the MAXMI chipset to interface between the XCI bus and the CPU module's DAL bus. The XCI bus is the interface between the MAXMI chipset and the XMI Corner. The MAXMI block diagram is shown in Figure 2–24.

Figure 2–24 MAXMI Block Diagram



The MAXMI control array chip (MCA-chip) is responsible for the address path, DAL control logic, XMI control logic, and the data path chip (MDA). The MAXMI has 8 Kbytes of RAM located in node private space, from addresses E100 8000 through E100 9FFF (hex). This is used as local storage space by the console program and self-test. The MAXMI RAM is in addition to the 1 Kbyte of MSSC RAM. The MAXMI RAM supports byte, word, and longword references. The RAM array is not parity protected and the DAL bus parity is neither checked nor generated on reads or writes to the RAM.

The MAXMI supports ECC protection of the B-cache data. The ECC codes provide single-bit correction and multi-bit detection for each byte in the B-cache with a 6-bit code for each byte of data. The MAXMI writes ECC codes into the B-cache on all cache fills and, in parallel with the MP-chip, on MP-chip-generated write transactions. The MAXMI corrects all single-bit errors when data is being written back to memory from the B-cache. When multi-bit errors are detected, the main memory location is tagged bad by the MAXMI. This is done since the MAXMI detects multi-bit errors, but does not correct them.

The MCA-chip monitors XMI traffic and buffers up to 16 invalidate addresses for transfer to the MC-chip on the Inval-Bus. If an XMI read or write by another node hits the cache, the writeback or invalidate is performed by the MC-chip. Writeback control, an 8-entry writeback address queue, and DAL bus parity checking are also provided by the MCA-chip.

The MDA-chip implements the interface between the 64-bit DAL bus and the 64-bit XCI bus. It provides parity generation and checking for the XMI data path and parity and ECC generation and checking for the DAL data path. This ECC checking on the DAL bus provides correction of all single-bit errors and detection of multi-bit errors. The MDA-chip also supports a 4-quadword read queue and an 8-hexword writeback data queue.

2.6.1 KA65A XMI Private I/O Address Space Map and Transactions

Figure 2–25 KA65A CPU Module Private I/O Address Space Map

Byte Address		
E000 0000	Control Register (CREG) write-enable	
E000 0004 E000 3FFF	RESERVED	256 Kbytes
E004 0000 E009 FFFF	Self-Test/Console/Boot Code (halt protected) three (3) 128KB X 8 PROMs	384 Kbytes
E00A 0000 E00A 7FFF	Self-Test/Console/Boot Code (halt protected) one (1) 32KB X 8 EEPROM	32 Kbytes
E00A 8000 E00B FFFF	RESERVED	96 Kbytes
E00C 0000 E011 FFFF	Self-Test/Console/Boot Code (not halt protected, PROM)	384 Kbytes
E012 0000 E012 7FFF	Self-Test/Console/Boot Code (not halt protected, EEPROM)	32 Kbytes
E012 8000 E013 FFFF	RESERVED	96 Kbytes
E014 0000 E014 03FF	MSSC CSRs	1 Kbyte
E014 0400 E014 07FF	MSSC Battery-Backed-Up RAM	1 Kbyte
E014 0800 E0FF FFFF	RESERVED	14.7 Mbytes
E100 0000 E100 001F	MDA CSRs	32 Bytes
E100 0020 E100 7FFF	RESERVED	32 Kbytes
E100 8000 E100 9FFF	MAXMI RAM	8 Kbytes
E100 A000 E100 FFFF	RESERVED	24 Kbytes
E101 0000 E101 FFFF	Interprocessor IVINTR Generation "Virtual" Registers	64 Kbytes
E102 0000 E102 FFFF	Write Error IVINTR Generation "Virtual" Registers	64 Kbytes
E103 0000 E17F FFFF	RESERVED	7.75 Mbytes

mab-p263-90

Figure 2–25 shows the CPU module's implementation of its XMI private space. Addresses E000 0000 to E17F 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.

Table 2–11 describes the MAXMI generation and response for various MP-chip-to-XMI operations, while Table 2–12 describes the MAXMI generation and response for various XMI-to-MP-chip operations.

Table 2–11 MAXMI Transaction Generation/Response for MP-Chip-to-XMI Operations

MP-Chip Operation	Resulting XMI Operation
Memory Space References	
Read (misses both caches)	Hexword Read
Interlock/Modify Read (misses both caches)	Hexword Ownership Read
Write Mask (misses both caches)	Hexword Ownership Read
Unlock Write Mask (always hits the B-cache)	None
I/O Space References (outside of XMI Private Space)	
Read (forced to miss both caches)	Longword Read
Interlock Read (forced to miss both caches)	Longword Interlock Read
Write Mask	Longword Write
Unlock Write Mask	Longword Unlock Write Mask
Miscellaneous References	
Interrupt acknowledge	XMI IDENT (assuming that an XMI interrupt is pending and no MSSC or IP IVINTR interrupts are pending)
I/O space write to IVINTR generation space	XMI IVINTR
Clear write buffer	Complete any outstanding Ownership Reads and process all queued invalidates

Table 2–12 MAXMI Transaction Generation/Response for XMI-to-MP-Chip Operations

XMI Transaction	Resulting MP-Chip Operation
XMI memory space writes	Load invalidate queue, perform Inval-Bus lookup.
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 MP-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.

2.6.2 Invalidates

The MAXMI monitors all write traffic from other nodes to memory space to maintain cache coherency between the KA65A CPU module caches and main memory. This monitoring also allows other XMI nodes access to memory locations owned by the KA65A CPU module. When the MAXMI detects an XMI write to a currently cached memory location, the MAXMI performs an invalidate on the DAL. The MC-chip works with the MAXMI to determine if the data is currently cached. The MC-chip maintains a duplicate copy of the primary cache tag store whenever invalidate addresses are placed on the Inval-Bus. If the MC-chip detects an Inval-Bus address mismatch in either tag store, it notifies the MAXMI of the hit. This implements a filtering mechanism for invalidates that avoids using the DAL except when an address is found cached and needs to be disowned.

When the MAXMI 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 MAXMI requests the MC-chip to do a cache lookup. If this address misses both caches, the MAXMI deletes the entry from the invalidate queue and goes to the next entry. If the address hits in either cache, the MAXMI performs a DMA write transaction on the DAL to this address, invalidating the address in both caches.

The MAXMI's invalidate queue is 16 entries deep. The MAXMI uses the XMI suppress line (XMI SUP L) to suppress XMI transactions, keeping the invalidate queue from overflowing. The suppression of XMI commands allows the MAXMI and MC-chip to catch up on invalidate processing and to open up queue entries for future invalidate addresses.

The MAXMI processes all invalidates that arrived in the XMI cycles before the read command is acknowledged on the XMI bus in response to a read-type DAL command. The MAXMI accomplishes this by logically "marking" the valid entries in the invaldiate 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 bus.

While processing invalidates, the MC-chip might find an entry corresponding to data being cached. If this happens, the MC-chip requests a retry of the command to process the invalidate. When the MC-chip completes processing all invalidates, the MP-chip retries the read-type command. The MAXMI responds with the data that it buffered from the XMI.

If an invalidate address is received in the same cache subblock as an outstanding cacheable memory read, a cache miss occurs. The MP-chip stays in an internal state until the cache fill is completed. However, the MC-chip continues to process invalidates while waiting for the cache fill to complete.

The MAXMI 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 MAXMI retries the clear write buffer command and processes all invalidates that hit the DAL before letting the MP-chip retry the clear write buffer command.

2.6.3 Writeback Queue

The MAXMI contains an eight-entry writeback queue to hold cache subblocks that need to be written back from the cache to memory. Both the MC-chip and the MAXMI retry MP-chip commands that might not complete if the MC-chip has a writeback to perform. The XMI control logic in the MAXMI gives priority to writeback commands over MP-chip-generated requests.

2.6.4 Lockout Avoidance

The MAXMI supports a lockout avoidance function that prevents an XMI node from being denied access to an interlocked or owned memory location or to prevent the node from being denied access to I/O space.

2.6.5 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 MAXMI responds to all XMI interrupts and generates IVINTRs.

2.6.5.1 Device Interrupts (INTRs)

The four priority levels of XMI device interrupts are IPL 14 (hex) through IPL 17 (hex). The MAXMI keeps a set of 40 interrupt-pending bits (ten I/O-capable nodes with four IPL levels). If an interrupt command with an interrupt destination field matching the node ID of the CPU is received, the MAXMI sets the appropriate interrupt-pending bit, based on the IPL of the interrupt and the node ID of the commander. All ten bits at each interrupt level are ORed together and sent to the CPU on device interrupt lines IRQ L<3:0>.

2.6.5.2 Implied Vector Interrupts (IVINTRs)

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

The MAXMI 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 MP-chip.

Write error interrupts cause the MAXMI 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 MAXMI to generate an IVINTR command on the XMI bus. The addresses, in hexadecimal, for these IVINTR-generation "virtual" registers are:

- E101 0000 to E101 FFFF for IP IVINTRs
- E102 0000 to E102 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 MAXMI treats IVINTR transactions like normal I/O space writes on the DAL. The MAXMI 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).

2.6.5.3 Read Interrupt Vector and IDENT

The MP-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 MSSC, or the MAXMI (if an IP IVINTR is pending). The IPL of MSSC interrupts is programmable through control bits in the MSSC and the MAXMI, 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 MAXMI gives the read interrupt vector command priority in the order:

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

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

While the MAXMI waits for the vector offset from the XMI during an XMI IDENT, any invalidate that hits in either cache forces the MAXMI 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 MAXMI allows the MP-chip to retry the read interrupt vector transactions. The MAXMI then returns the vector it buffered from the XMI.

If the MAXMI receives a read interrupt vector transaction at the MSSC-programmed IPL and no MSSC interrupt is pending, it transmits an IDENT on the XMI. If an invalidate hit occurs, the MAXMI retries the read interrupt vector transaction so it can first process the invalidate. If an MSSC interrupt request is pending when the read interrupt vector transaction is retried, the MAXMI cannot allow the MSSC 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 MSSC interrupt is delayed until the original interrupt is completed.

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

2.6.6 XMI Registers

The MAXMI registers are located in the KA65A CPU module's nodespace and are directly accessible by all XMI nodes. A read or write access to any nonimplemented nodespace causes the MAXMI 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 MAXMI registers are as follows:

- Device Register, XDEV (nodespace + 0000 0000)
- Bus Error Register, XBER (nodespace + 0000 0004)
- Failing Address Register, XFADR (nodespace + 0000 0008)
- General Purpose Register, XGPR (nodespace + 0000 000C)
- Node-Specific Control and Status Register, NSCSR (nodespace + 0000 001C)
- XMI Control Register, XCR (nodespace + 0000 0024)

- Failing Address Extension Register, XFAER (nodespace + 0000 002C)
- Bus Error Extension Register, XBEER (nodespace + 0000 0034)
- Failing Address Register for Wback0, WFADR0 (nodespace + 0000 0040)
- Failing Address Register for Wback1, WFADR1 (nodespace + 0000 0044)

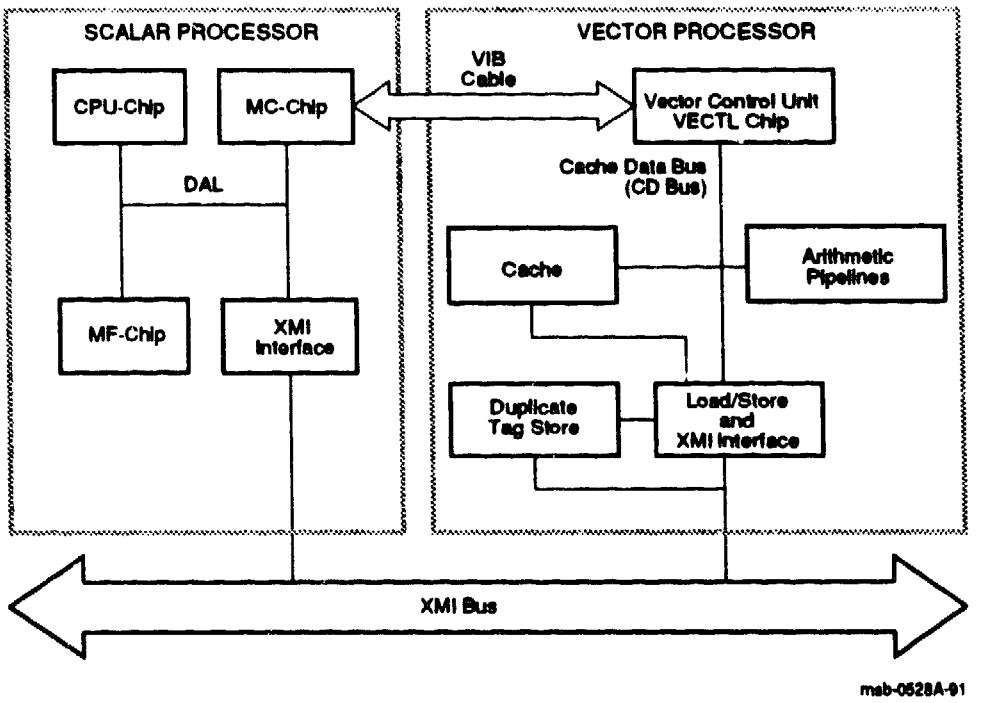
The MAXMI's registers have the following characteristics:

- The mask bits are ignored on writes to the MAXMI'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 MAXMI NO ACKs subsequent CSR references until the read data for the queued CSR read has been returned.

2.7 Scalar/Vector Interaction

The KA65A CPU module supports an optional attached vector module. For details on the FV64A vector processor, see Chapter 3.

Figure 2–26 Scalar/Vector Pair Block Diagram



The scalar module includes support for an optional vector module that executes VAX vector instructions. The interface between the MP-chip and the vector module is provided by the MC-chip (see Figure 2–26). Command and control information are transferred between the MC-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 MC-chip and the vector module itself. The MC-chip contains the Vector Interface Error Status Register (VINTSR), IPR123, which contains information about the scalar CPU side of the VIB (see Section 2.8.1 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 3.8.

2.7.1 Vector Instruction Execution

To an executing program, the scalar/vector pair of modules appear as one processor. The MP-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 MP-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.

2.7.2 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 MP-chip in response to the register read done at the end of the MP-chip instruction execution. The MP-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 vector 54 (hex). See Section 2.3.4 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 MP-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.

2.8 KA65A CPU Module Registers

The KA65A CPU module registers consist of internal processor registers, KA65A CPU module registers in XMI private space, and XMI required registers.

2.8.1 Internal Processor Registers

The processor state is stored in internal processor registers rather than in memory. See Table 2–13 and Table 2–14. 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 2–13 KA65A 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	MP-chip
1 (1)	Executive Stack Pointer	ESP	R/W	PROC	1	MP-chip
2 (2)	Supervisor Stack Pointer	SSP	R/W	PROC	1	MP-chip
3 (3)	User Stack Pointer	USP	R/W	PROC	1	MP-chip
4 (4)	Interrupt Stack Pointer	ISP	R/W	CPU	1	MP-chip
5 – 7 (5 – 7)	Reserved				4	

[1]See Table 2–14.

[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 KA65A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA65A CPU module.
3 = Implemented by the FV64 vector 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 KA65A CPU module reset (power-up, system reset, and node reset).

Table 2–13 (Cont.) KA65A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
8 (8)	P0 Base	P0BR	R/W	PROC	1	MP-chip
9 (9)	P0 Length	P0LR	R/W	PROC	1	MP-chip
10 (A)	P1 Base	P1BR	R/W	PROC	1	MP-chip
11 (B)	P1 Length	P1LR	R/W	PROC	1	MP-chip
12 (C)	System Base	SBR	R/W	CPU	1	MP-chip
13 (D)	System Length	SLR	R/W	CPU	1	MP-chip
14 – 15 (E – F)	Reserved				4	
16 (10)	Process Control Block Base	PCBB	R/W	PROC	1	MP-chip
17 (11)	System Control Block Base	SCBB	R/W	CPU	1	MP-chip
18 (12)	Interrupt Priority Level	IPL	R/W	CPU	1 Init	MP-chip
19 (13)	AST Level	ASTLVL	R/W	PROC	1 Init	MP-chip
20 (14)	Software Interrupt Request	SIRR	WO	CPU	1	MP-chip
21 (15)	Software Interrupt Summary	SISR	R/W	CPU	1 Init	MP-chip
22 – 23 (16 – 17)	Reserved				4	
24 (18)	Interval Clock Control and Status[4]	ICCS	R/W	CPU	2 Init	MP-chip
25 – 26 (19 — 1A)	Reserved				4	
27 (1B)	Time-of-Year Clock[5]	TODR	R/W	CPU	1	MSSC
28 (1C)	Console Storage Receiver Status	CSRS	R/W	CPU	6 Init	MSSC
29 (1D)	Console Storage Receiver Data	CSRD	RO	CPU	6 Init	MSSC
30 (1E)	Console Storage Transmitter Status	CSTS	R/W	CPU	6 Init	MSSC
31 (1F)	Console Storage Transmitter Data	CSTD	WO	CPU	6 Init	MSSC

[1]See Table 2–14.

[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 KA65A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA65A CPU module.
3 = Implemented by the FV64 vector 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 KA65A 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 2–13 (Cont.) KA65A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
32 (20)	Console Receiver Control/Status	RXCS	R/W	CPU	2 Init	MSSC
33 (21)	Console Receiver Data Buffer	RXDB	RO	CPU	2 Init	MSSC
34 (22)	Console Transmitter Control /Status	TXCS	R/W	CPU	2 Init	MSSC
35 (23)	Console Transmitter Data Buffer	TXDB	WO	CPU	2 Init	MSSC
36 – 37 (24 – 25)	Reserved				4	
38 (26)	Machine Check Error Summary	MCESR	WO	CPU	2	MP-chip
39 (27)	Reserved				4	
40 (28)	Accelerator Control and Status	ACCS	R/W	CPU	2 Init	MP-chip
41 (29)	Reserved				4	
42 (2A)	Console Saved Program Counter	SAVPC	RO	CPU	2	MP-chip
43 (2B)	Console Saved Processor Status Longword	SAVPSL	RO	CPU	2	MP-chip
44 – 46 (2C – 2E)	Reserved				4	
47 (2F)	Translation Buffer Tag	TBTAG	WO	CPU	2	MP-chip
48 – 54 (30 – 36)	Reserved				4	
55 (37)	I/O Reset	IORESET	WO	CPU	2	MSSC
56 (38)	Memory Management Enable	MAPEN	R/W	CPU	1 Init	MP-chip
57 (39)	Translation Buffer Invalidate All	TBIA	WO	CPU	1	MP-chip
58 (3A)	Translation Buffer Invalidate Single	TBIS	WO	CPU	1	MP-chip
59 (3B)	Translation Buffer Data	TBDATA	WO	CPU	2	MP-chip
60 – 61 (3C – 3D)	Reserved				4	
62 (3E)	System Identification	SID	RO	CPU	1	MP-chip
63 (3F)	Translation Buffer Check	TBCHK	WO	CPU	1	MP-chip
64 – 111 (40 – 6F)	Reserved				4	
112 (70)	Backup Cache Index	BCIDX	R/W	CPU	2	MC-chip

[1]See Table 2–14.

[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 KA65A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA65A CPU module.
3 = Implemented by the FV64 vector 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 KA65A CPU module reset (power-up, system reset, and node reset).

Table 2–13 (Cont.) KA65A CPU Module Internal Processor Registers

Address decimal (hex)	Register	Mnemonic	Type[1]	Scope[2]	Class[3]	Location
113 (71)	Backup Cache Status	BCSTS	R/W	CPU	2 Init	MC-chip
114 (72)	Backup Cache Control	BCCTL	R/W	CPU	2 Init	MC-chip
115 (73)	Backup Cache Error Address	BCERA	RO	CPU	2	MC-chip
116 (74)	Backup Cache Tag Store	BCBTS	R/W	CPU	2	MC-chip
117 (75)	Backup Cache Deallocate Tag	BCDET	WO	CPU	2	MC-chip
118 (76)	Backup Cache Error Tag	BCERT	RO	CPU	2	MC-chip
119 – 122 (77 – 7A)	Backup Cache Reserved	–	R/W	CPU	6	MC-chip
123 (7B)	Vector Interface Error Status	VINTSR	R/W	CPU	2	MC-chip
124 (7C)	Primary Cache Tag Array	PCTAG	R/W	CPU	2	MP-chip
125 (7D)	Primary Cache Index	PCIDX	R/W	CPU	2	MP-chip
126 (7E)	Primary Cache Error Address	PCERR	R/W	CPU	2	MP-chip
127 (7F)	Primary Cache Status	PCSTS	R/W	CPU	2 Init	MP-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
148 – 156 (94 – 9C)	Reserved				6	
157 (9D)	Vector Indirect Register 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 2–14.

[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 KA65A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA65A CPU module.
3 = Implemented by the FV64 vector 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 KA65A CPU module reset (power-up, system reset, and node reset).

NOTE: The vector internal processor registers IPR144 through IPR159 are described in Chapter 3.

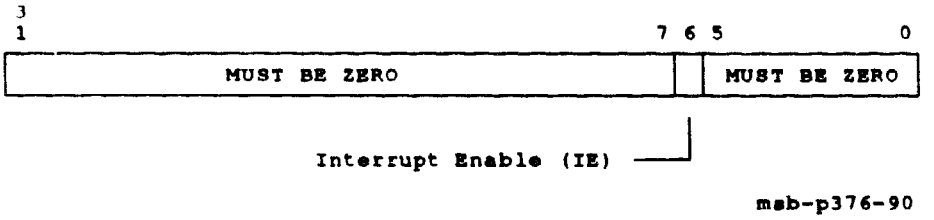
Table 2–14 Types of Registers and Bits

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 (MP-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>

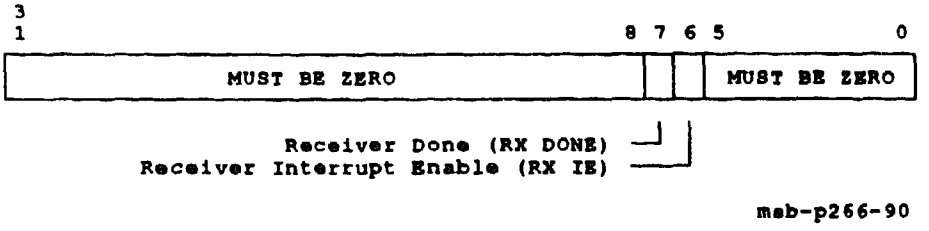
Name: Reserved
Mnemonic: None
Type: –

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 (MSSC)*



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

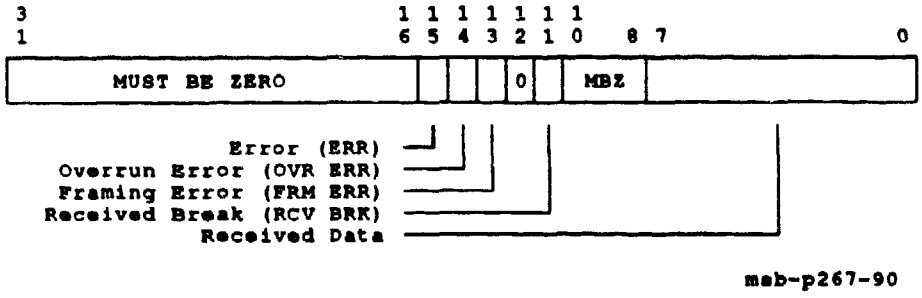
Type: –

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 (MSSC)*



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>

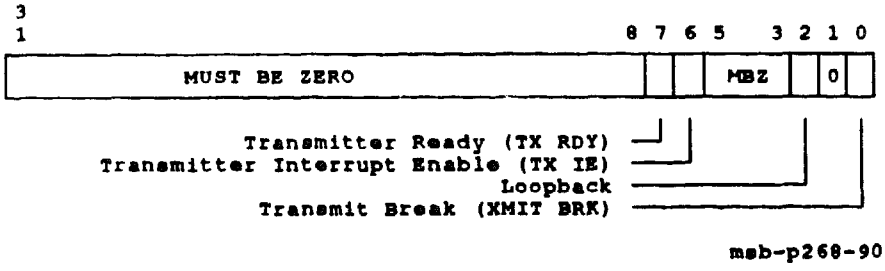
Name: Received Data
Mnemonic: None
Type: RO

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 (MSSC)*



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 RDY 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>

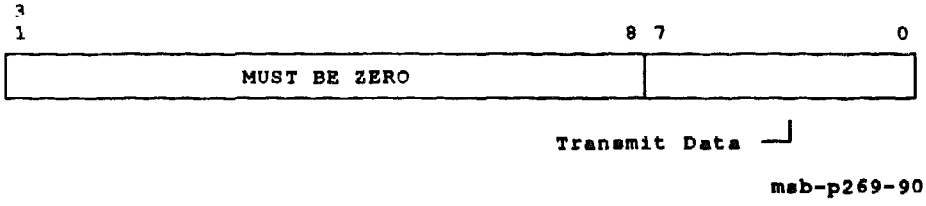
Name: Transmit Break
Mnemonic: XMIT BRK
Type: R/W, 0

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 (MSSC)*



bits<31:8>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<7:0>

Name: Transmit Data

Mnemonic: None

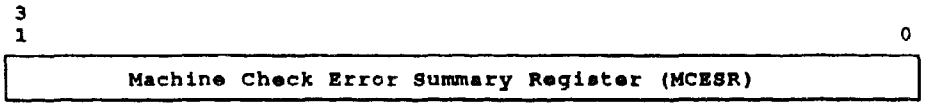
Type: WO

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 (MP-chip)*



mab-p270-90

bits<31:0>

Name: Machine Check Error Summary Register

Mnemonic: MCESR

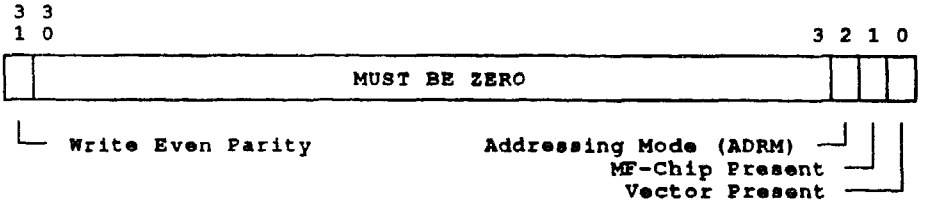
Type: WO

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 MF-chip, the optional vector module, and the generation of bad data parity on write operations.

ADDRESS *IPR40 (MP-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 MF-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<30:3>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<2>

Name: Addressing Mode

Mnemonic: ADRM

Type: R/W, 0

Addressing Mode controls the MP-chip's use of either 30- or 32-bit physical addressing. When ADRM is a zero, 30-bit addressing is used; when ADRM is a one, 32-bit addressing is used. Memory management must be disabled to change ADRM.

bit<1>

Name: MF-Chip Present

Mnemonic: None

Type: R/W, 0

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

bit<0>

Name: Vector Present

Mnemonic: None

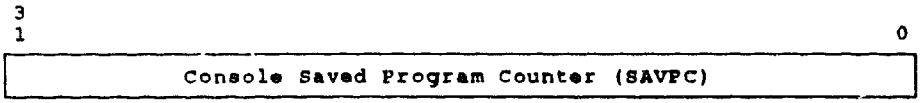
Type: R/W

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 (MP-chip)*



mab-p272-90

bits<31:0>

Name: Console Saved Program Counter

Mnemonic: SAVPC

Type: RO

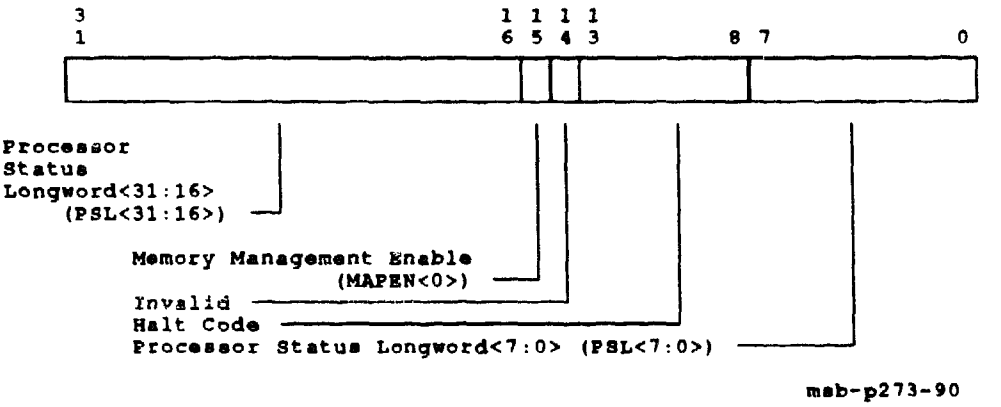
If the MP-chip microcode detects an inconsistent internal state, an incorrectly terminated DAL transaction, a kernel-mode HALT, a system reset, a node halt, a node reset, or the detection of CTRL/P, the microcode initiates a console halt (a hardware restart sequence which passes control to the console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

Console Saved Processor Status Longword (SAVPSL)

If the MP-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 (MP-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 one 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

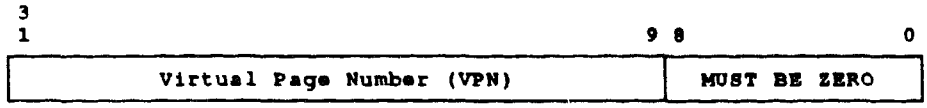
At the time of a console halt, PSL<7:0> is loaded into SAVPSL<7:0>.

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 register is not written to during normal operation.

ADDRESS

IPR47 (MP-chip)



mab-p274-90

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 (MSSC)*

msb-p275-90

bits<31:0>

Name: IORESET

Mnemonic: NONE

Type: R/W

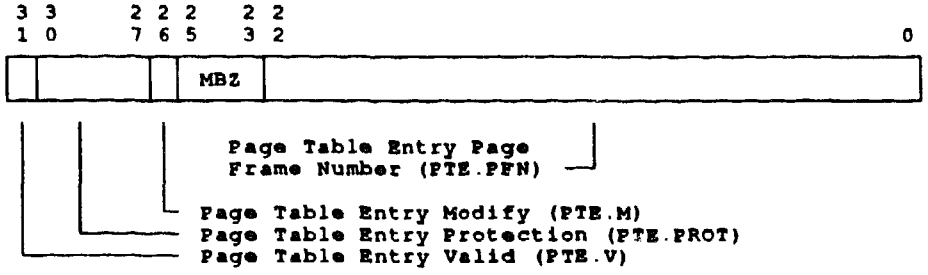
A KA65A CPU module can force a system reset by writing to IORESET with an MTPR (Move To Proce or Register) instruction.

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 (MP-chip)



msb-p276-90

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<25:23>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<22:0>

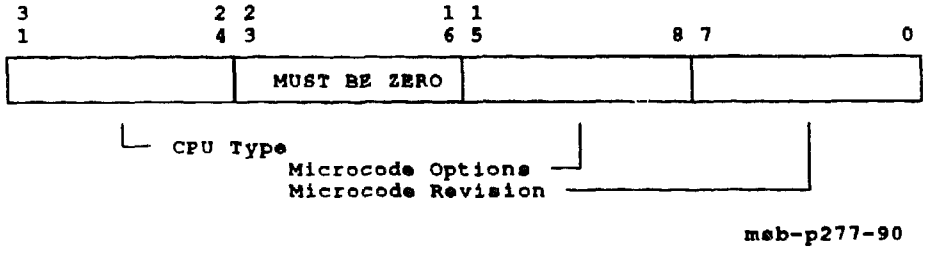
Name: Page Table Entry Page Frame Number
Mnemonic: PTE.PFN
Type: WO

This field is a repeat of the page frame number field of the VAX PTE. It contains the upper 23 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 (MP-chip)*



bits<31:24>

Name: CPU Type

Mnemonic: None

Type: RO

This field is always 18 (decimal), indicating the KA65A CPU module's MP-chip.

bits<23:16>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<15:8>

Name: Microcode Options

Mnemonic: None

Type: RO

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

bits<7:0>

Name: Microcode Revision

Mnemonic: None

Type: RO

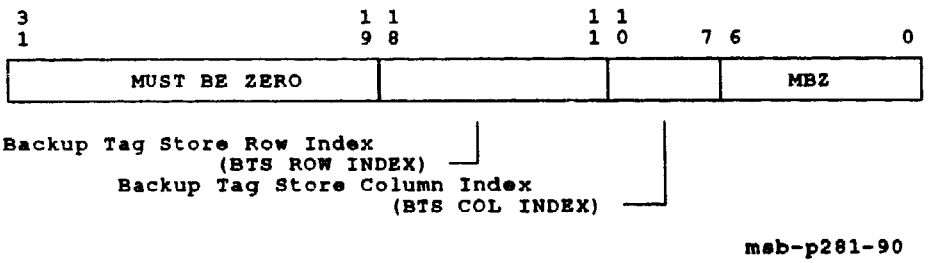
This field specifies the microcode revision of the MP-chip.

Backup Cache Index Register (BCIDX)

The Backup Cache Index Register accesses the Backup Cache Tag Store Register through IPR reads and writes.

ADDRESS

IPR112 (MC-chip)



bits<31:19>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<18:11>

Name: Backup Tag Store Row Index

Mnemonic: BTS ROW INDEX

Type: R/W, 0

BTS ROW INDEX stores the backup tag store row index.

bits<10:7>

Name: Backup Tag Store Column Index

Mnemonic: BTS COL INDEX

Type: R/W, 0

BTS COL INDEX stores the backup tag store column index.

bits<6:0>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

Backup Cache Status Register (BCSTS)

BCSTS is a status register that shows the results of tag stores, DAL transactions, and invalidates. BCSTS is read using an IPR read. Only hardware can write to BCSTS, but certain bits can be cleared with an IPR write. All error bits are cleared through an IPR write of the status register or during reset. When all error bits are cleared, ERR SUMMARY clears and the MC-chip exits error transition mode (ETM).

NOTE: The MC-chip is in ETM when either BCSTS<ERR SUMMARY> is set or both Backup Cache Control Register <ENABLE BTS bit <1> and Backup Cache Control Register <BTS ERROR TRAN bit <2> are set.

During normal operation, the following are loaded every transaction:

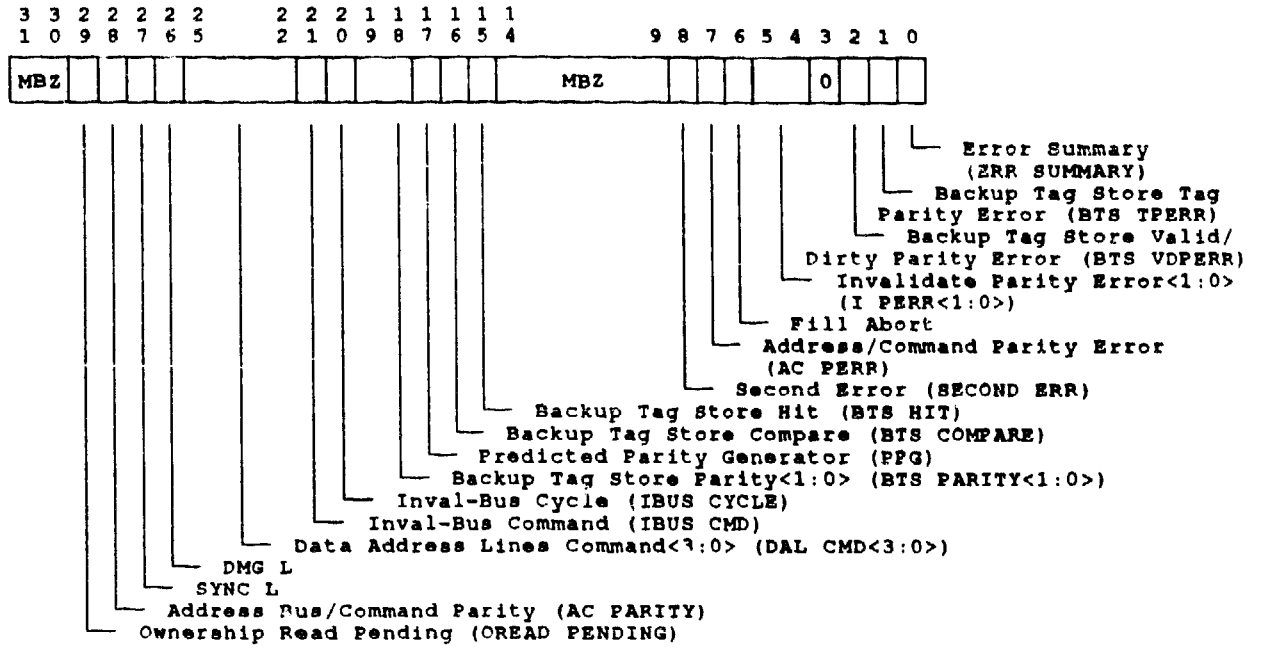
- DAL CMD <22:25>
- DMG L <26>
- SYNCH L <27>
- AC PARITY <28>
- OREAD PENDING <29>

Inval-Bus Cycle, bit <20>, and Inval-Bus Command, bit <21>, are loaded every time the backup tag store is accessed for:

- Memory read and write DAL transactions
- DMA cache fills
- Inval-Bus request lookups
- Invalidates and writebacks from the DAL

When ERR SUMMARY, bit <0> is set, the MC-chip is forced into error transition mode. The MC-chip cannot turn itself off when it encounters an error because there might be dirty subblocks (subblocks that have been written to) in the cache. The MC-chip must enter ETM. During ETM, the MC-chip will only respond to transactions that hit dirty subblocks. BCSTS, BCERA, and BCERT have their loads disabled to allow the MP-chip to examine the state of the MC-chip when the error occurred. If the MC-chip is in ETM, unrecoverable subsequent errors set SECOND ERR, bit <8>, but do not affect the other bits. When the MC-chip is not in ETM, all errors that occur in the same transaction are logged into the status register.

ADDRESS *IPR113 (MC-chip)*



mab-p282-90

bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<29>

Name: Ownership Read Pending

Mnemonic: OREAD PENDING

Type: RO, 0

OREAD PENDING sets when there is a pending Ownership Read during the last DAL transaction. It is used with the DMG L and SYNC L signals to determine if a cache fill sequence was due to a pending Ownership Read. If IBUS CYCLE, bit <20>, is set, OREAD PENDING is unpredictable.

bit<28>

Name: Address Bus/Command Parity

Mnemonic: AC PARITY

Type: RO, 0

AC PARITY stores the parity for the Address Bus and Command fields of the last DAL transaction. If IBUS CYCLE, bit <20>, is set, AC PARITY is unpredictable.

bit<27>

Name: SYNC L

Mnemonic: None

Type: RO, 0

SYNC L contains the value of the SYNC L signal from the last DAL transaction. It is used with DMG L to determine if a hexword cache fill sequence is in progress. If IBUS CYCLE, bit <20>, is set, SYNC L is unpredictable.

bit<26>

Name: DMG L

Mnemonic: None

Type: RO, 0

DMG L contains the value of the DMG L signal from the last DAL transaction. It is used to distinguish between MP-chip generated transactions and DMA DAL transactions. It is also used, with SYNC L, to determine if a hexword cache fill sequence is in progress. If IBUS CYCLE, bit <20>, is set, DMG L is unpredictable.

bits<25:22>

Name: Data Address Lines Command<3:0>

Mnemonic: DAL CMD<3:0>

Type: RO, 0

DAL CMD<3:0> contains the last DAL command. If IBUS CYCLE, bit <20>, is set, DAL CMD<3:0> is unpredictable.

bit<21>

Name: Inval-Bus Command
Mnemonic: IBUS CMD
Type: RO, 0

IBUS CMD is loaded during the same cycle that IBUS CYCLE, bit <20> sets. IBUS CMD is the invalidate command signal that was received for the XMI for the Inval-Bus request being serviced by the tag store. If IBUS CYCLE <20> is not set, IBUS CMD is unpredictable.

bit<20>

Name: Inval-Bus Cycle
Mnemonic: IBUS CYCLE
Type: RO, 0

IBUS CYCLE sets when this register is loaded, during the servicing of an Inval-Bus request.

bits<19:18>

Name: Backup Tag Store Parity<1:0>
Mnemonic: BTS PARITY<1:0>
Type: RO, 0

BTS PARITY<1> contains the parity generated from the tag on the last tag store access.

BTS PARITY<0> contains the parity generated from the valid/dirty bits on the last tag store access.

bit<17>

Name: Predicted Parity Generator
Mnemonic: PPG
Type: RO, 0

PPG contains the output of the predicted parity generator for the tag on the last backup tag store access. PPG is the parity generated on A BUS<18:7>.

bit<16>

Name: Backup Tag Store Compare
Mnemonic: BTS COMPARE
Type: RO, 0

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

bit<15>

Name: Backup Tag Store Hit
Mnemonic: BTS HIT
Type: RO, 0

BTS HIT sets when an Inval-Bus invalidate request hits a valid subblock in the B-cache, a memory read hits a valid subblock in the B-cache, or a memory write completes by either hitting a valid subblock in the B-cache or by performing a write-and-run.

bits<14:9>

Name: Reserved
Mnemonic: None
Type: –

Unused; must be zero.

bit<8>

Name: Second Error
Mnemonic: SECOND ERR
Type: R/W1C, 0

SECOND ERR sets if the MC-chip is in ETM and an unrecoverable subsequent error occurs on another transasction. No other state changes when SECOND ERR sets.

bit<7>

Name: Address/Command Parity Error
Mnemonic: AC PERR
Type: R/W1C, 0

AC PERR sets if there is an address/command parity error as the result of the last DAL transaction. It is loaded for all modes of MC-chip operation.

bit<6>

Name: Fill Abort
Mnemonic: None
Type. RO, 0

Fill Abort sets when a cache fill abort command is received from the XMI. Fill Aborts sets only when Backup Cache Control Register <ENABLE BTS bit <1> is set and Backup Cache Control Register <FORCE BHIT bit <0> is not set.

bits<5:4>

Name: Invalidate Bus Parity Error<1:0>
Mnemonic: I PERR<1:0>
Type: R/W1C, 0

This field contains the result of the last invalidate lookup. I PERR<1> sets if the second half of the Inval-Bus transaction caused a parity error. I PERR<0> sets if the first half of the Invalidate Bus transaction caused a parity error. This field is not loaded until the invalidate request has done the tag store lookup. I PERR<1:0> is always loaded, even if the MC-chip is disabled.

bit<3>

Name: Reserved
Mnemonic: None
Type: –

Unused; must be zero.

bit<2>

Name: Backup Tag Store Valid/Dirty Parity Error
Mnemonic: BTS VDPERR
Type: R/W1C, 0

BTS VDPERR sets when a parity error occurs in the valid/dirty (written into) part of a tag entry as a result of the last access of the tag store. This bit sets only when Backup Cache Control Register <ENABLE BTS bit <1> is set and Backup Cache Control Register <FORCE BHIT bit <0> is not set.

bit<1>

Name: Backup Tag Store Tag Parity Error
Mnemonic: BTS TPERR
Type: R/W1C, 0

BTS TPERR sets when a parity error occurs in the tag entry as a result of the last access of the tag store. This bit sets only when Backup Cache Control Register <ENABLE BTS bit <1> is set and Backup Cache Control Register <FORCE BHIT bit <0> is not set.

bit<0>

Name: Error Summary
Mnemonic: ERR SUMMARY
Type: RO, 0

ERR SUMMARY sets whenever any of the following sets:

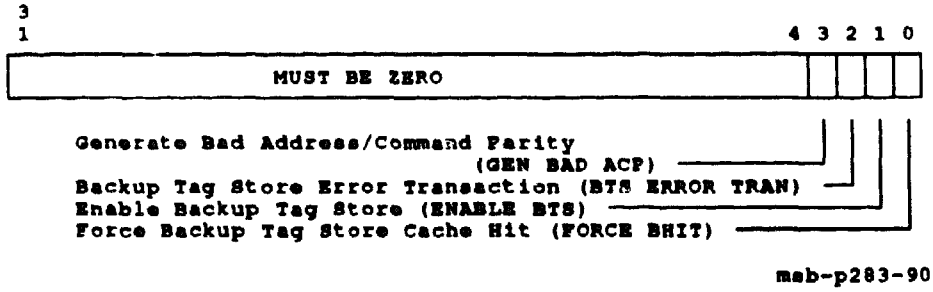
- BTS TPERR (bit <1>)
- BTS VDPERR (bit <2>)
- I PERR <1:0> (bits <5:4>)
- FILL ABORT (bit <6>)
- AC PERR (bit <7>)
- SECOND ERR (bit <8>)

When ERR SUMMARY sets, the MC-chip is forced into ETM. ERR SUMMARY clears when all the error bits are cleared.

Backup Cache Control Register (BCCTL)

Logic external to the MC-chip uses BCCTL to control the MC-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.

ADDRESS *IPR114 (MC-chip)*



mab-p283-90

bits<31:4>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<3>

Name: Generate Bad Address/Command Parity

Mnemonic: GEN BAD ACP

Type: R/W, 0

GEN BAD ACP, when set, causes the MC-chip to generate incorrect address/command parity (ACP) for its transactions. GEN BAD ACP is used for diagnostic purposes.

bit<2>

Name: Backup Tag Store Error Transaction

Mnemonic: BTS ERROR TRAN

Type: R/W, 0

If ENABLE BTS is zero, BTS ERROR TRAN is ignored. When BTS ERROR TRAN and ENABLE BTS are both set, the backup cache is forced into error transition mode (ETM). While in ETM, all MP-chip memory space references are treated as though the backup cache is disabled, except for those references that have a valid and dirty subblock in the B-cache. References that hit a dirty subblock are serviced from the B-cache. All invalidate requests from the Inval-Bus are forwarded to the P-cache as writeback-and-invalidate commands for dirty hits or as invalidate-only commands for all other invalidate requests. The MC-chip services forwarded writeback-and-invalidate commands and ignores invalidate-only commands.

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 memory reads produce a cache miss, and no memory writes to the cache are done.
- The MC-chip response to cache fill transactions is unpredictable.
- IPR read and write transactions continue normally.
- All Inval-Bus lookups are forwarded to the MP-chip as invalidate-only. The MC-chip ignores the forwarded invalidate-only.
- No B-cache tag store parity errors and no cache fill abort commands are reported.

ENABLE BTS should be cleared by software only when BTS ERROR TRAN, bit <2>, is set. However, software may clear ENABLE BTS independently for diagnostic purposes.

CAUTION: Software should not clear ENABLE BTS until all cache fills are completed for outstanding memory reads or writes.

bit<0>

Name: Force Backup Tag Store Cache Hit

Mnemonic: FORCE BHIT

Type: R/W, 0

When FORCE BHIT is set, all memory space backup tag store accesses produce a cache hit, including read lock accesses. The cache RAM data is written for each memory space write and read on every memory space read. The MC-chip response is unpredictable to cache fill transactions. All Inval-Bus invalidates are forwarded to the MP-chip as invalidate-only. The MC-chip ignores forwarded invalidate-only commands. IPR read and write transactions continue normally, but the state of the backup tag store is unpredictable and must be flushed before returning to normal operations. No backup tag store parity errors nor cache fill abort commands are reported.

If ENABLE BTS is clear and FORCE BHIT is set, the backup tag store response is unpredictable. The MC-chip response is also unpredictable if the MC-chip is in ETM and FORCE BHIT is set.

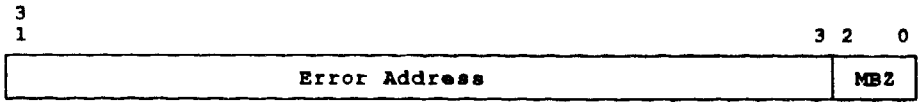
CAUTION: Software should not clear FORCE BHIT until all cache fills are completed for outstanding memory reads or writes.

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

Backup Cache Error Address Register (BCERA)

BCERA contains the address of the current transaction. If the MC-chip goes into error transition mode (ETM), BCERA locks against further writes, regardless of subsequent errors, until the MC-chip exits ETM. BCERA is read using an IPR read. IPR writes are ignored.

ADDRESS *IPR115 (MC-chip)*



msb-p279-90

bits<31:3>

Name: Backup Cache Error Address
Mnemonic: BCERA
Type: RO, 0

BCERA is loaded by hardware for every new DAL transaction and every Inval-Bus request lookup. BCERA contains the address of the current transaction. There are two sources for BCERA: the Address bus for normal DAL transactions and the Inval-Bus for invalidate request lookups.

bits<2:0>

Name: Reserved
Mnemonic: None
Type: –

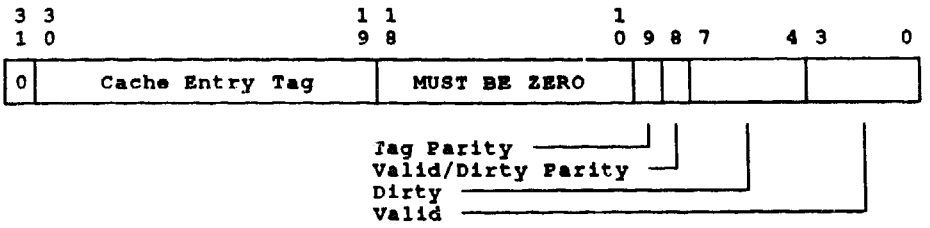
Unused; must be zero.

Backup Cache Tag Store Register (BCBTS)

BCBTS is used to access the tag store, which contains 4096 entries of 22 bits each. Each entry consists of a tag parity bit, one 12-bit tag (address<30:19>), four dirty (or written) bits, four valid bits, and one dirty/valid parity bit for each 4-hexword 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 the Backup Cache Index Register (BCIDX).

ADDRESS

IPR116 (MC-chip)



bit<31>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<30:19>

Name: Cache Entry Tag

Mnemonic: None

Type: R/W

The Cache Entry Tag field contains bits <30:19> of the backup cache address.

bits<18:10>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<9>

Name: Tag Parity
Mnemonic: None
Type: R/W

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

bit<8>

Name: Valid/Dirty Parity
Mnemonic: None
Type: R/W

The Valid/Dirty Parity bit contains odd parity for BCBTS<7:0>, the Valid and Dirty bits. The user supplies correct parity when writing to BCBTS.

bit<7:4>

Name: Dirty
Mnemonic: None
Type: R/W

Each of the four Dirty bits is associated with one hexword subblock of the 4-hexword block. Bits <6:5> of the backup cache address are used to select the valid bit corresponding to the addressed hexword, as shown:

BCBTS Bit	A_bus<6:5> Value	Subblock Number
4	0 0	1
5	0 1	2
6	1 0	3
7	1 1	4

bits<3:0>

Name: Valid

Mnemonic: None

Type: R/W

Each of the four Valid bits is associated with one hexword subblock of the 4-hexword block. Bits <6:5> of the backup cache address are used to select the valid bit corresponding to the addressed hexword, as shown:

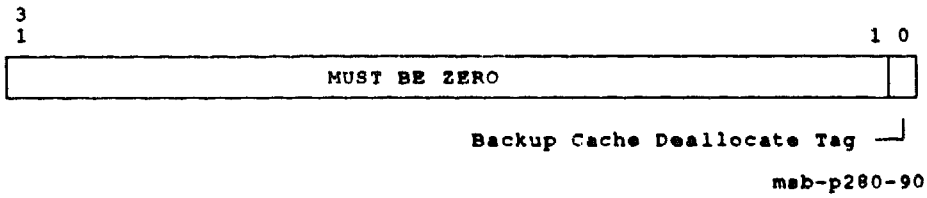
BCBTS Bit	A_bus<6:5> Value	Subblock Number
0	0 0	1
1	0 1	2
2	1 0	3
3	1 1	4

Backup Cache Deallocate Tag Register (BCDET)

BCDET is a write-only register that provides software with a means of flushing the backup cache by sequentially deallocating each block.

ADDRESS

IPR117 (MC-chip)



bits<31:1>

Name: Reserved
Mnemonic: None
Type: –

Unused; must be zero.

bit<0>

Name: Deallocate Tag
Mnemonic: None
Type: WO, X

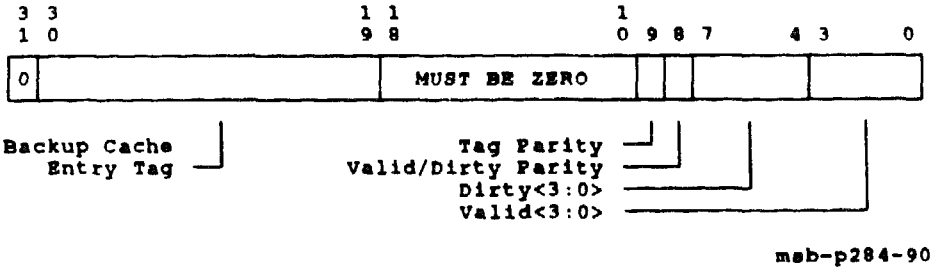
Deallocate Tag provides software with a means of flushing the backup cache by sequentially deallocating each block. The backup tag store row index and the backup tag store column index fields of BCIDX are used to index the tag array. BCIDX must have been previously written using an IPR write to ensure predictable results when performing the deallocate. The MC-chip immediately invalidates or writes back the current block being deallocated, if required.

Backup Cache Error Tag Register (BCERT)

BCERT is loaded by hardware every time the backup tag store is accessed for all memory reads and write DAL transactions, for all DMA cache fills, for all Inval-Bus request lookups, and for all invalidates and writebacks from the DAL. If the MC-chip enters ETM, BCERT locks against further writes, regardless of subsequent errors, until the MC-chip exits error transition mode. BCERT is read using an IPR read. IPR writes are ignored.

ADDRESS

IPR118 (MC-chip)



bit<31>

Name: Reserved

Mnemonic: None

Type: –

Unused; must be zero.

bits<30:19>

Name: Backup Cache Entry Tag

Mnemonic: None

Type: RO, 0

Backup Cache Entry Tag contains the tag.

bit<18:10>

Name: Reserved

Mnemonic: None

Type: –

Unused; must be zero.

bit<9>

Name: Tag Parity
Mnemonic: None
Type: RO, 0

Tag Parity contains the parity bit for the tag.

bit<8>

Name: Valid/Dirty Parity
Mnemonic: V/D PARITY
Type: RO, 0

V/D PARITY contains the valid/dirty parity bit for the tag.

bits<7:4>

Name: Dirty<3:0>
Mnemonic: None
Type: RO, 0

Dirty<3:0> contains the four dirty bits for the tag.

bits<3:0>

Name: Valid<3:0>
Mnemonic: None
Type: RO, 0

Valid<3:0> contains the four valid bits for the tag.

Vector Interface Error Status Register (VINTSR)

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

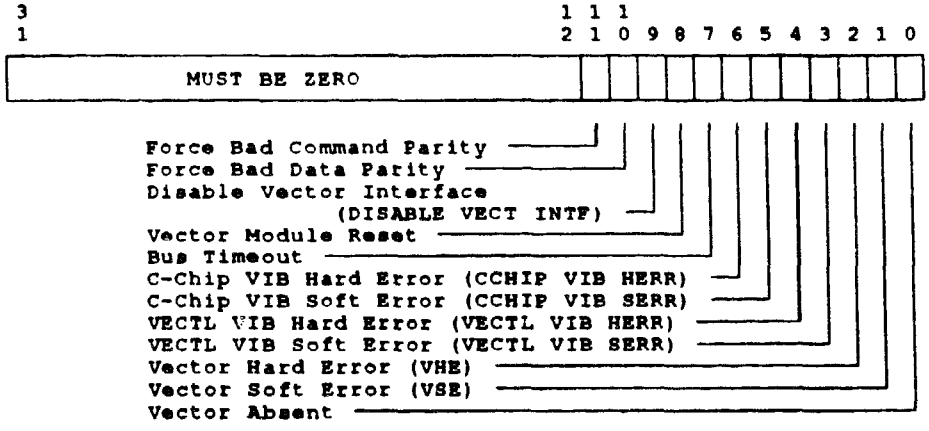
Software reads this register to determine if VECTL VIB HERR, CCHIP VIB HERR, Bus Timeout, or Vector Absent are set before accessing the vector module over the VIB. If any of these bits is set, the VIB is not functioning properly, and it is impossible to communicate with the vector module without first 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 (MC-chip)



mab-p175-90

bits<31:12>

Name: Reserved

Mnemonic: None

Type: –

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 MC-chip to generate bad command parity on all subsequent VIB operations. Clearing Force Bad Command Parity returns the MC-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 MC-chip to generate bad data parity on all subsequent VIB operations. Clearing Force Bad Data parity returns the MC-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 MC-chip. The MC-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 MC-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 MC-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the MP-chip. Reads and writes of VINTSR still function normally. The MC-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 MC-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 MC-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 MC-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 MC-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 MC-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 MC-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 MC-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the MP-chip. Reads and writes of VINTSR still function normally. The MC-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 MC-chip interface logic resets to the idle state and asserts the VECT INTERFACE RESET L signal, causing VECTL to reset its VIB interface state.

If a read operation caused the timeout error, the MC-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 MC-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 MC-chip begins to drive these lines again in this same cycle. The MC-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 MC-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 MC-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 MC-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the MP-chip. Reads and writes of VINTSR still function normally. The MC-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 MC-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 MC-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 MC-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 MC-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 MC-chip returns the VIB machine check code (D BUS H<8:7> = 00 (binary)) to the MP-chip. Reads and writes of VINTSR still function normally. The MC-chip also deasserts the VECTOR UNIT ENABLED L signal and asserts the VECT INTERFACE RESETL 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 MC-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 MC-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 MC-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 MC-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 MC-chip has write data in its instruction buffer, the MC-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 MC-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 MC-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 module. When VSE sets, the MC-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 MC-chip detects the deassertion of the VECT PRESENT L signal on the VIB, which the MC-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 MC-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 MC-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 MC-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 MC-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 MC-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 MC-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.

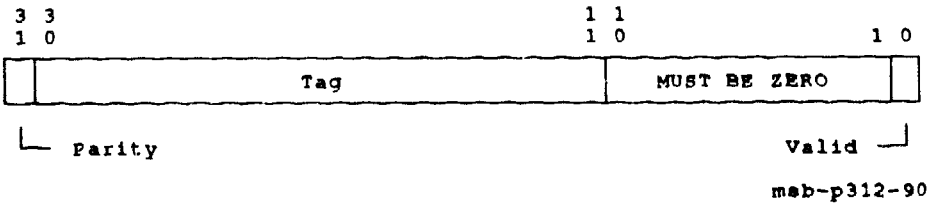
If Vector Absent is set during a pending VIB transaction, the MC-chip VIB interface returns to the home state. If the pending transaction is a read, it is terminated with the ERR L signal on the scalar DAL asserted, and the VIB machine check code (D BUS H<8:7> = 00 (binary)) is returned to the CPU.

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 (MP-chip)



bit<31>

Name: Parity
Mnemonic: None
Type: R/W

Parity bit (odd parity) for tag.

bits<30:11>

Name: Tag
Mnemonic: None
Type: R/W

The primary cache tag contents.

bits<10:1>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<0>

Name: Valid
Mnemonic: None
Type: R/W

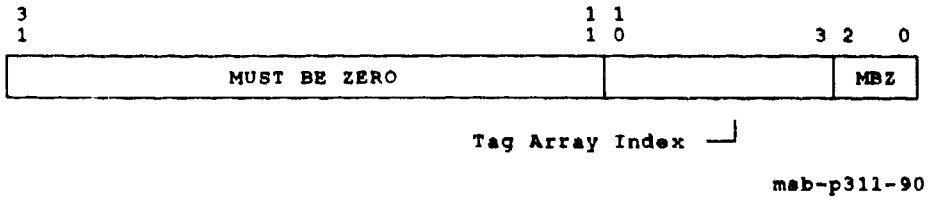
Valid bit for tag.

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 (MP-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: –

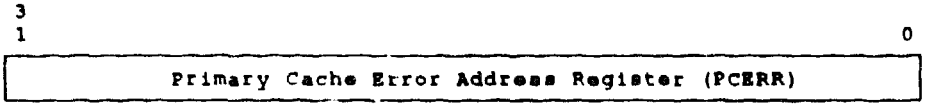
Reserved; must be zero.

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 remains locked until Trap1 is cleared. PCERR is valid only if Trap1 is set.

ADDRESS

IPR126 (MP-chip)



mab-p286-90

bits<31:0>

Name: Primary Cache Error Address Register

Mnemonic: PCERR

Type: RO

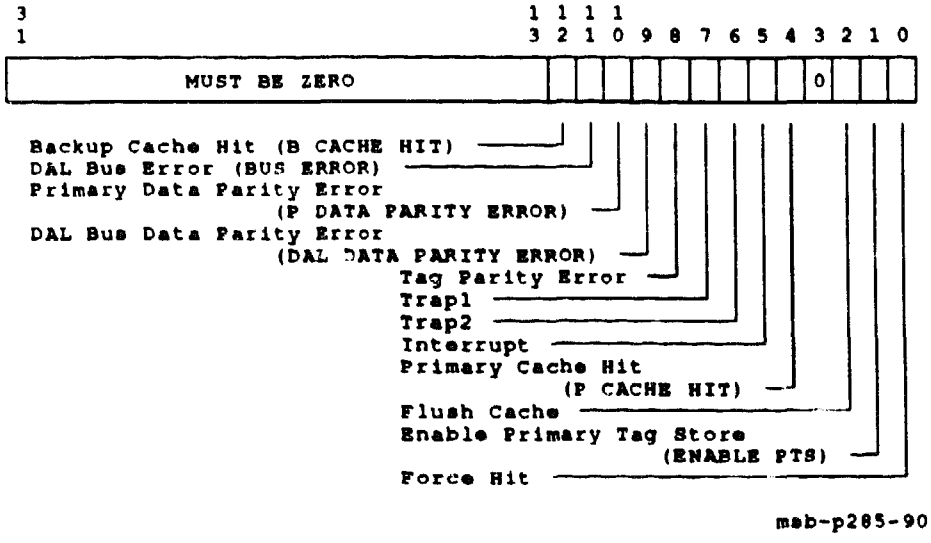
PCERR latches and holds the physical address of an error when PCSTS<7>, Trap1, sets for read commands. Since write errors are asynchronous to the instruction pipeline, the address latched for write commands is not the address of the error and the write error address is not available.

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 (MP-chip)



bits<31:13>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<12>

Name: Backup Cache Hit

Mnemonic: B CACHE HIT

Type: RO, 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 memory 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: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<2>

Name: Flush Cache
Mnemonic: None
Type: R/W1C, 0

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 errors 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.

NOTE: When ENABLE PTS is zero (primary cache is disabled) and Force Hit is set, the operation of the primary cache is unpredictable.

2.8.2 XMI Registers

In addition to the internal processor registers, the KA65A CPU module contains registers in XMI private space and XMI required registers in XMI nodespace. These registers are listed in Table 2–15 and Table 2–16.

The KA65A CPU module's XMI registers have the following characteristics:

- The mask bits are ignored on writes to the KA65A 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 XMI responder queue is only one deep so the KA65A CPU module will NO ACK subsequent CSR references until the read data for the queued CSR read has been returned.
- Write transactions directed at read-only registers are accepted and acknowledged but no action is taken and the register's value is not changed.

Read or write access to any unimplemented nodespace regions result in a NO ACK for the XMI transaction

Table 2–15 KA65A CPU Module Registers in XMI Private Space

Register	Mnemonic	Address	Location
Control Register 0	CREG0	none	A "virtual" register
Control Register 1	CREG1	none	A "virtual" register
CREG Write Enable	CREGWE	E000 0000	MSSC
Console ROM (halt protected)		E004 0000 – E009 FFFF	Console ROM
Console EEPROM (halt protected)		E00A 0000 – E00A 7FFF	Console EEPROM
Console ROM (not halt protected)		E00C 0000 – E011 FFFF	Console ROM
Console EEPROM (not halt protected)		E012 0000 – E012 7FFF	Console EEPROM
MSSC Base Address	SSCBAR	E014 0000	MSSC
MSSC Configuration	SSCCNR	E014 0010	MSSC
MSSC Bus Timeout Control	SSCBTR	E014 0020	MSSC
MSSC Output Port	OPORT	E014 0030	MSSC
MSSC Input Port	IPORT	E014 0040	MSSC
CREG Base Address	CRBADR	E014 0130	MSSC
CREG Address Decode Mask	CRADMR	E014 0134	MSSC
EEPROM Base Address	EEBADR	E014 0140	MSSC
EEPROM Address Decode Mask	EEADMR	E014 0144	MSSC
Timer 0 Control	TCR0	E014 0160	MSSC

Table 2–15 (Cont.) KA65A CPU Module Registers in XMI Private Space

Register	Mnemonic	Address	Location
Timer 0 Interval	TIR0	E014 0164	MSSC
Timer 0 Next Interval	TNIR0	E014 0168	MSSC
Timer 0 Interrupt Vector	TIVR0	E014 016C	MSSC
Timer 1 Control	TCR1	E014 0170	MSSC
Timer 1 Interval	TIR1	E014 0174	MSSC
Timer 1 Next Interval	TNIR1	E014 0178	MSSC
Timer 1 Interrupt Vector	TIVR1	E014 017C	MSSC
Interval Counter	SSCICR	E014 01F8	MSSC
MSSC Internal RAM		E014 0400 – E014 07FF	MSSC
DAL Diagnostic Register	DCSR	E100 0000	MDA-chip
Failing DAL Register 0	FDAL0	E100 0020	MDA-chip
Failing DAL Register 1	FDAL1	E100 0028	MDA-chip
Failing DAL Register 2	FDAL2	E100 0030	MDA-chip
Failing DAL Register 3	FDAL3	E100 0038	MDA-chip
MAXMI RAM		E100 8000 – E100 9FFF	MAXMI RAM
Interprocessor Implied Vector Interrupt Generation	IPIVINTR	E101 0000 – E101 FFFF	MAXMI RAM
Write Error Implied Vector Generation	WEIVINTR	E102 0000 – E102 FFFF	MAXMI RAM

Table 2–16 XMI Registers for the KA65A CPU Module

Register	Mnemonic	Address	Location
Device Register	XDEV	BB[1] + 0000 0000	MCA-chip
Bus Error	XBER	BB + 0000 0004	MCA-chip
Failing Address	XFADR	BB + 0000 0008	MCA-chip
XMI General Purpose	XGPR	BB + 0000 000C	MCA-chip
Node-Specific Control and Status	NSCSR	BB + 0000 001C	MCA-chip
XMI Control Register	XCR	BB + 0000 0024	MCA-chip
Failing Address Extension	XFAER	BB + 0000 002C	MCA-chip
Bus Error Extension	XBEER	BB + 0000 0034	MCA-chip
Writeback 0 Failing Address	WFADR0	BB + 0000 0040	MCA-chip
Writeback 1 Failing Address	WFADR1	BB + 0000 0044	MCA-chip

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

Control Register 0 (CREG0)

CREG0 controls some KA65A CPU module functions.

Writing to CREG0 is a multistep process:

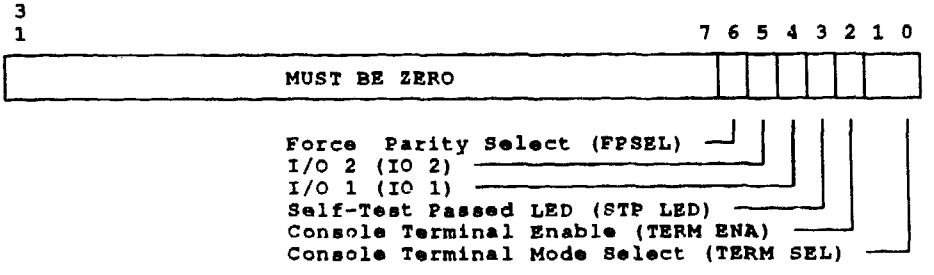
1 Write OPORT<9:2> (CREG DATA) with the data to be written to CREG0.

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)*



msb-p313-90

bits<31:7>

Name: *Reserved*
Mnemonic: None
Type: –

Reserved; must be zero.

bit<6>

Name: Force Parity Select
Mnemonic: FPSEL
Type: WO, 0

FPSEL is used with the XMI Control Register bits <21:19> to force bad parity on the XMI. Setting FPSEL enables the use of XMI Control Register bits <21:19>.

bit<5>

Name: I/O 2
Mnemonic: IO 2
Type: WO, 0

IO 2 is used by selt-test as a trigger. The value written to IO 2 is inverted before it is driven directly to a pin on the I/O section of the backplane.

bit<4>

Name: I/O 1
Mnemonic: IO 1
Type: WO, 0

IO 1 is used by selt-test as a trigger. The value written to IO 1 is inverted before it is driven directly to a pin on the I/O section of the backplane.

bit<3>

Name: Self-Test Passed LED
Mnemonic: STP LED
Type: WO, 0

STP LED sets to drive the self-test-passed LED that indicates that self-test was successful.

bit<2>

Name: Console Terminal Enable
Mnemonic: TERM ENA
Type: WO, 0

TERM ENA enables the KA65A 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
0 0	Auxiliary Console	The auxiliary console line is connected to the MSSC console terminal input.
0 1	System Console	The XMI backplane console line is connected to the MSSC console terminal input.
1 0	Auxiliary Console Loopback	The auxiliary console output is connected back to the MSSC console terminal input.
1 1	System Console Loopback	The XMI console output is connected back to the MSSC 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:

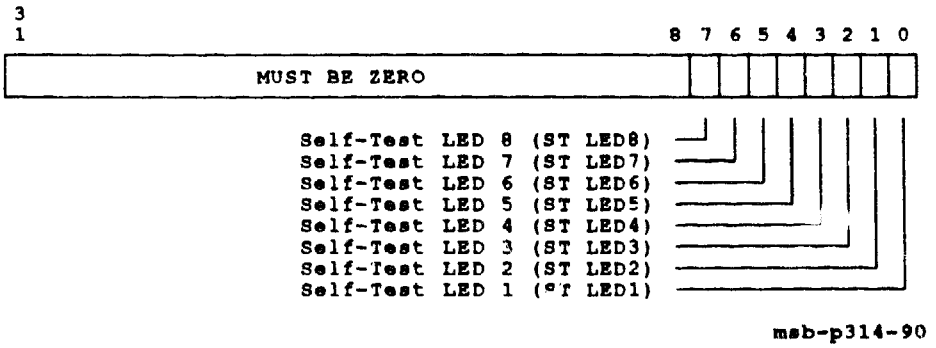
1 Write OPORT<9:2> (CREG DATA) with the data to be written to CREG1.

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: –

Reserved; must be zero.

bits<7:0>

Name: Self-Test LED 8 through Self-Test LED 1

Mnemonic: ST LED*n*

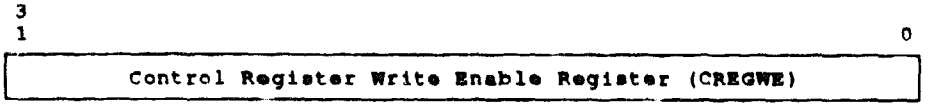
Type: WO, 0

These eight bits drive the eight 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, tc write to CREG0 and CREG1.

ADDRESS *E000 0000 (MSSC)*



msb-p008-89

bits<31:0>

Name: Control Register Write Enable

Mnemonic: CREGWE

Type: WO, 0

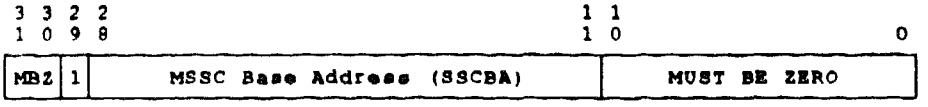
CREGWE enables writes to Control Register 0 and Control Register 1 (CREG0 and CREG1). The data that is in OPORT<9:2> (CREG DATA) is sent to the selected Control Register (OPORT<0>, OPORT<1>) whenever anything is written to CREGWE.

MSSC 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 MSSC. Hardware reset initializes SSCBAR to E014 0000 (hex).

ADDRESS

E014 0000 (MSSC)



msb-p322-90

bits<31:30>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<29>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be one.

bits<28:11>

Name: MSSC Base Address
Mnemonic: SSCBA
Type: R/W

SSCBAR controls the base address of the 2-Kbyte block and is set to E014 0000 (hex) by console code during processor initialization.

bits<10:0>

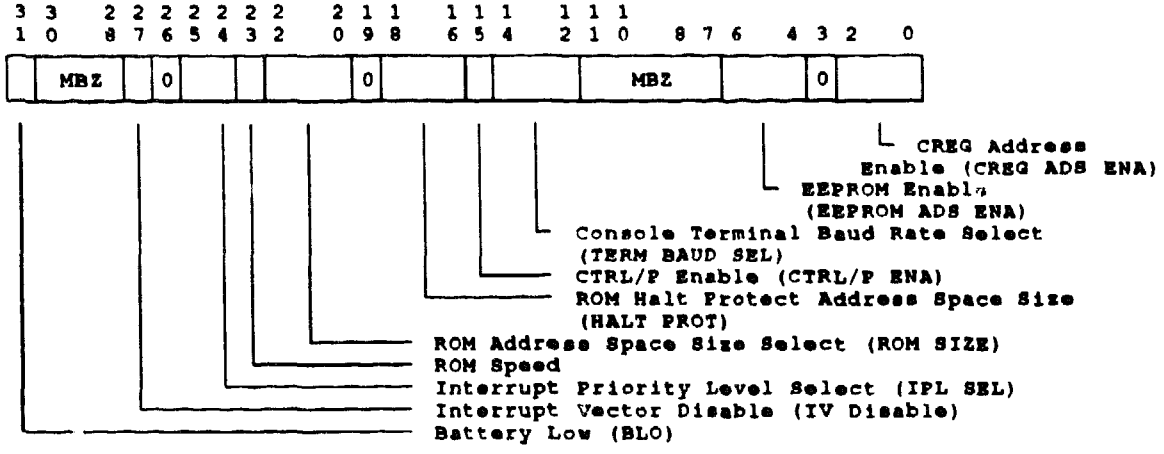
Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

MSSC Configuration Register (SSCCNR)

SSCCNR provides configuration information for MSSC-controlled features.

ADDRESS *E014 0010 (MSSC)*



mab-p323-90

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: –

Reserved; must be zero.

bit<27>

Name: Interrupt Vector Disable
Mnemonic: IV Disable
Type: R/W, 0

When IV Disable is set, the MSSC is disabled from returning an interrupt vector in response to a read interrupt vector transaction from the MC-chip. IV Disable is clear for normal operation.

bit<26>

Name: Reserved
Mnemonic: None
Type: –

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.

CAUTION: IPL SEL must match XCR<MSCIPL> to ensure correct operation. If these values are different, the operation of the interrupt system is undefined.

bit<23>

Name: ROM Speed
Mnemonic: None
Type: R/W, 0

ROM Speed selects the access time of the ROM/EEPROM, which is a function of the CPU cycle time. Setting ROM Speed to zero selects 250 ns while setting it to one selects 150 ns. ROM Speed must be set to zero for normal operation.

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 set to zero during reset and must be set to 111 (binary) before self-test, yielding an address range of 1 Mbyte (E004 0000 to E013 FFFF).

<22:20>	Size (Kbytes)	Address Range (hex)
000	8	E004 0000 – E004 1FFF
001	16	E004 0000 – E004 3FFF
010	32	E004 0000 – E004 7FFF
011	64	E004 0000 – E004 FFFF
100	128	E004 0000 – E005 FFFF
101	256	E004 0000 – E007 FFFF
110	512	E004 0000 – E00B FFFF
111	1024	E004 0000 – E013 FFFF

CAUTION: Before the console code attempts to access any ROM or EEPROM code outside the first 8 Kbytes of ROM, console code must first set ROM SIZE to 111 (binary) since hardware resets ROM SIZE to 000 (binary).

bit<19>

Name: Reserved

Mnemonic: None

Type: –

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 MSSC 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	E004 0000 – E004 1FFF
001	16	E004 0000 – E004 3FFF
010	32	E004 0000 – E004 7FFF
011	64	E004 0000 – E004 FFFF
100	128	E004 0000 – E005 FFFF
101	256	E004 0000 – E007 FFFF
110	512	E004 0000 – E00B 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 MSSC baud clock runs about 1.75% fast.

bit<11:7>

Name: Reserved

Mnemonic: None

Type: –

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 the 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: –

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

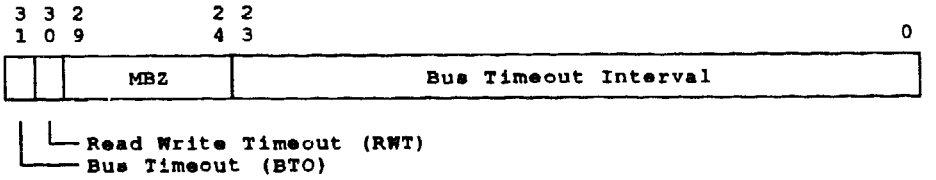
CREG ADS ENA is set to 101 by the console program during initialization.

MSSC Bus Timeout Control Register (SSCBTR)

SSCBTR contains the timeout value used to terminate DAL bus transactions. When a DAL transaction is started, the MSSC starts a timer that increments every microsecond until the transaction completes. If the time reaches the value contained in Bus Timeout Interval, the MSSC asserts the ERR L signal to terminate the transaction, causing a machine check and preventing system hangs.

ADDRESS

E014 0020 (MSSC)



mab-p324-90

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, 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 MSSC or with the MAXMI.

bits<29:24>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<23:0>

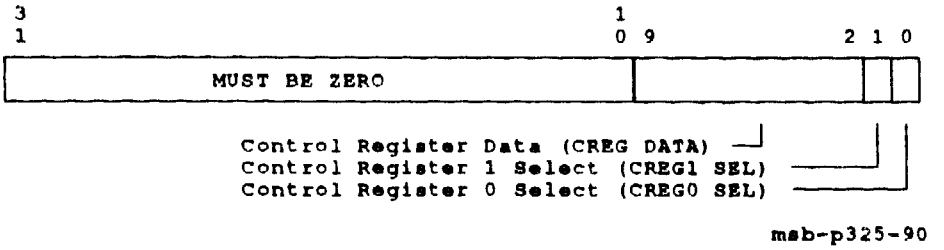
Name: Bus Timeout Interval
Mnemonic: None
Type: R/W, 0

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.

MSSC Output Port Register (OPORT)

OPORT selects the desired control register and writes it with data.

ADDRESS *E014 0030 (MSSC)*



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

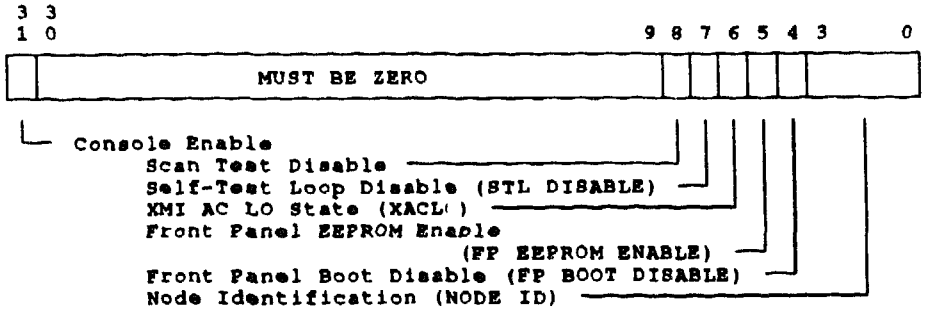
Type: R/W, 0

When set, CREG0 SEL selects Control Register 0 as the target to write the data in CREG DATA.

MSSC Input Port Register (IPORT)

IPORT gives KA65A CPU module state information.

ADDRESS *E014 0040 (MSSC)*



msb-p326-90

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: None

Type: <RO, 1

Scan Test Disable indicates the state of the module's I/O pin.

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 *E014 0130 (MSSC)*

msb-p327-90

bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

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: –

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

E014 0134 (MSSC)

mab-p328-90

bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

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: –

Reserved; must be zero.

EEPROM Base Address Register (EEBADR)

EEBADR supplies the base address for EEPROM write enables.

ADDRESS *E014 0140 (MSSC)*

mab-p329-90

bits<31:30>

Name: Reserved
Mnemonic: None
Type: –

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 200A 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 E00A 0000 to E00A 7FFF (hex)).

bits<1:0>

Name: Reserved
Mnemonic: None
Type: –

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

E014 0144 (MSSC)

msb-p330-90

bits<31:30>

Name: Reserved

Mnemonic: None

Type: –

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

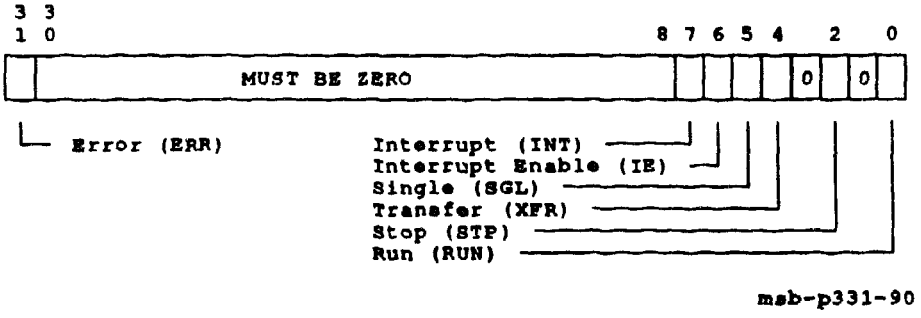
Type: –

Reserved; must be zero.

Timer Control Register 0 (TCR0)

TCR0 controls timer 0.

ADDRESS *E014 0160 (MSSC)*



msb-p331-90

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: –

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: –

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: –

Reserved; must be zero.

bit<0>

Name: Run
Mnemonic: RUN
Type: R/W, 0

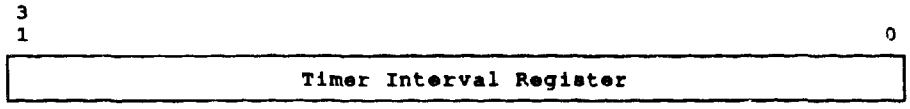
When RUN is set, the Timer Interval Register is incremented once every microsecond. INT is set when the timer overflows. If STP is set at overflow, RUN is cleared by the hardware at overflow and counting stops. When RUN is clear, the Timer Interval Register is not incremented automatically.

Timer Interval Register 0 (TIR0)

TIR0 contains the interval count for timer 0.

ADDRESS

E014 0164 (MSSC)



bits<31:0>

Name: Timer Interval Register 0

Mnemonic: TIR0

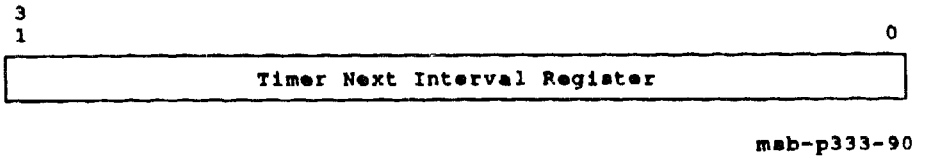
Type: RO, 0

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 *E014 0168 (MSSC)*



bits<31:0>

Name: Timer Next Interval Register 0

Mnemonic: TNIR0

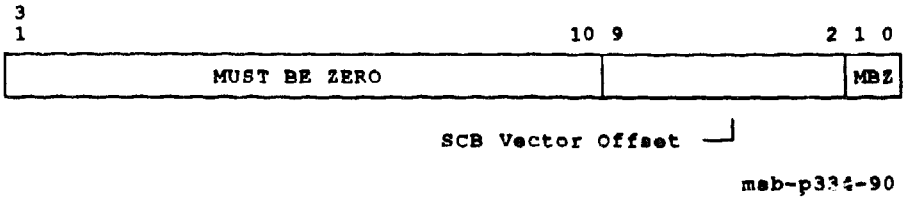
Type: R/W, C

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 *E014 016C (MSSC)*



bits<31:10>

Name: Reserved
Mnemonic: None
Type: –

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>

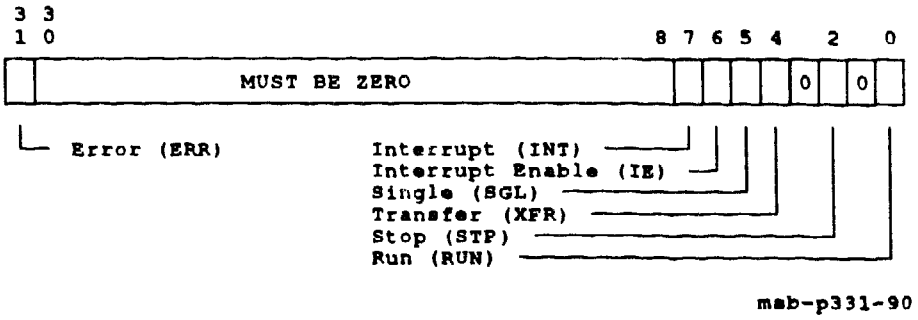
Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

Timer Control Register 1 (TCR1)

TCR1 controls timer 1, which is used by the console program.

ADDRESS *E014 0170 (MSSC)*



mab-p331-90

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: –

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: –

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: –

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

E014 0174 (MSSC)

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

E014 0178 (MSSC)

bits<31:0>

Name: Timer Next Interval Register 1

Mnemonic: TNIR1

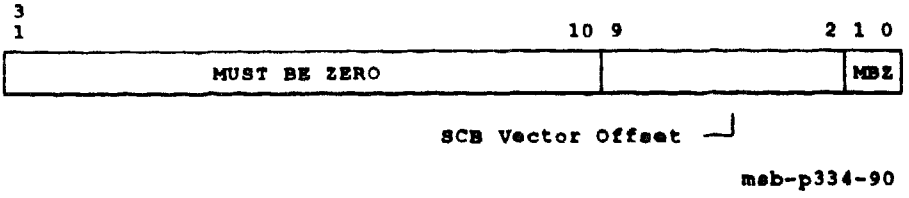
Type: R/W, 0

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 *E014 017C (MSSC)*



bits<31:10>

Name: Reserved

Mnemonic: None

Type: –

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: –

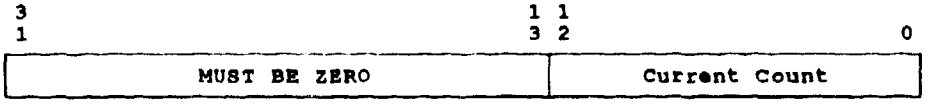
Reserved; must be zero.

Interval Counter Register (SSCICR)

SSCICR controls transitions of the INT TIM L pin on the MP-chip during diagnostic testing.

ADDRESS

E014 01F8 (MSSC)



mab-p335-90

bits<31:13>

Name: Reserved

Mnemonic: None

Type: –

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 MP-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.

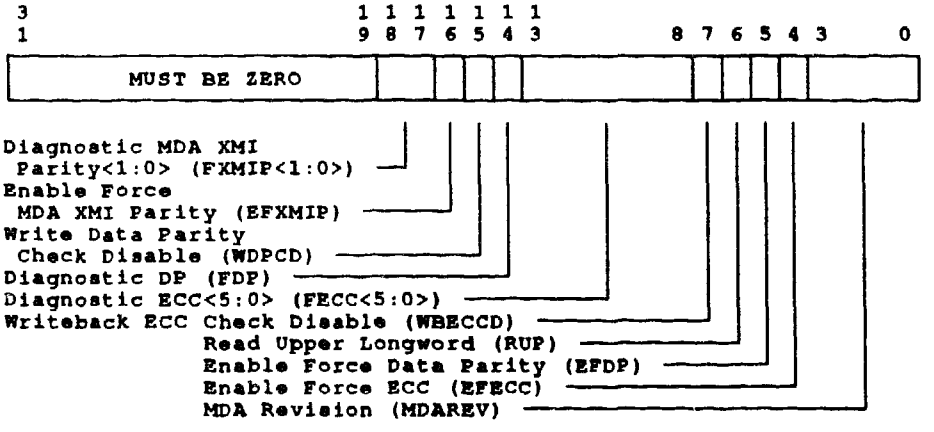
On reset, Current Count clears to zero, so the first interval is 8192 microseconds rather than the expected value of 5000 microseconds. SSCICR is used only for diagnostic purposes and is not written during normal system operation.

DAL Diagnostic Register (DCSR)

DCSR holds information used to test or disable various MDA-chip functions.

ADDRESS

E100 0000 (MDA-chip)



msb-p336-90

bits<31:19>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<18:17>

Name: Diagnostic MDA XMI Parity<1:0>

Mnemonic: FXMIP<1:0>

Type: R/W, 0

FXMIP<1:0>, when enabled by setting DCSR<EFXMIP>, forces bad parity on the corresponding XMI P<1:0> signals when driven by the MDA-chip.

bit<16>

Name: Enable Force MDA XMI Parity

Mnemonic: EFXMIP

Type: R/W, 0

EFXMIP, when set, enables the forcing of bad parity on the XMI from the MDA-chip.

bit<15>

Name: Write Data Parity Check Disable
Mnemonic: WDPCD
Type: RO, 0

WDPCD, when set, disables the checking of parity on write data cycles on the DAL.

bit<14>

Name: Diagnostic DP
Mnemonic: FDP
Type: RO, 0

FDP, when set with DCSR<EFDP>, forces bad parity for each byte instead of generated MAXMI DP<7:0>.

bits<13:8>

Name: Diagnostic ECC<5:0>
Mnemonic: FECC<5:0>
Type: RO, 0

FECC<5:0>, when set with DCSR<EFECC>, forces bad ECC bits. FECC is used for each byte instead of the generated MAXMI ECC<47:0>.

bit<7>

Name: Writeback ECC Check Disable
Mnemonic: WBECCD
Type: RO, 0

WBECCD, when set, disables the detection and correction of writeback data errors on the DAL.

bit<6>

Name: Read Upper Longword
Mnemonic: RUP
Type: RO, 0

When set, read data on XMID<63:32> is mapped to XMID<31:0> while remaining the same on XMID<63:32>. RUP must not be set during memory-space XMI reads.

bit<5>

Name: Enable Force Data Parity
Mnemonic: EFDP
Type: RO, 0

EFDP, when set, causes DCSR<FDP> to be driven on DP<7:0> in place of the MAXMI-generated parity during cache fills. All eight bytes are given the same FDP parity value.

bit<4>

Name: Enable Force ECC
Mnemonic: EFECC
Type: RO, 0

EFECC, when set, drives DSCR<FECC<5:0> on ECC<47:0> in place of the MAXMI-generated ECC codes during cache fills and memory-space writes. All eight bytes are given the same 6-bit FECC value.

bits<3:0>

Name: MDA Revision
Mnemonic: MDAREV
Type: RO, 0

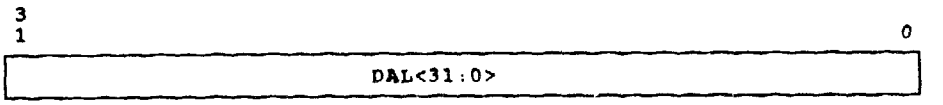
MDAREV identifies the revision level of the MDA-chip.

Failing DAL Register 0 (FDAL0)

This register contains DAL<31:0> and is loaded on all quadwords of a writeback and on memory-space reads. FDAL0 locks when a corrected or uncorrected writeback error occurs.

ADDRESS

E100 0020 (MDA-chip)



bits<31:0>

Name: Failing DAL Register 0

Mnemonic: FDAL0

Type: RO, 0

FDAL0 contains DAL<31:0> and is loaded on all quadwords of a writeback and on memory-space reads (hits or misses). Cache data is sampled at the beginning of the DAL read transaction. For read misses, the data is indeterminate.

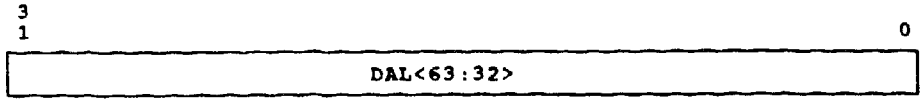
FDAL0 locks if a writeback error occurs (XBEER<WFDQ0> or XBEER<WFDQ1> set), whether corrected or uncorrected. It is not locked on read data errors.

Failing DAL Register 1 (FDAL1)

This register contains DAL<63:32> and is loaded on all quadwords of a writeback and on memory-space reads. FDAL1 locks when a corrected or uncorrected writeback error occurs.

ADDRESS

E100 0028 (MDA-chip)



mab-p338-90

bits<31:0>

Name: Failing DAL Register 1

Mnemonic: FDAL1

Type: RO, 0

FDAL1 contains DAL<63:32> and is loaded on all quadwords of a writeback and on memory-space reads (hits or misses). Cache data is sampled at the beginning of the DAL read transaction. For read misses, the data is indeterminate.

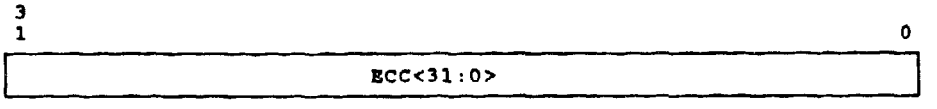
FDAL1 locks if a writeback error occurs (XBEER<WFDQ0> or XBER<WFDQ1> set), whether corrected or uncorrected. It is not locked on read data errors.

Failing DAL Register 2 (FDAL2)

This register contains ECC<31:0> and is loaded on all quadwords of a writeback and on memory-space reads. FDAL2 locks when a corrected or uncorrected writeback error occurs.

ADDRESS

E100 0030 (MDA-chip)



msb-p339-90

bits<31:0>

Name: Failing DAL Register 2

Mnemonic: FDAL2

Type: RO, 0

FDAL2 contains ECC<31:0> related to DAL<63:0> and is loaded on all quadwords of a writeback and on memory-space reads (hits or misses). Cache data is sampled at the beginning of the DAL read transaction. For read misses, the data is indeterminate.

FDAL2 locks if a writeback error occurs (XBEER<WFDQ0> or XBEER<WFDQ1> set), whether corrected or uncorrected. It is not locked on read data errors.

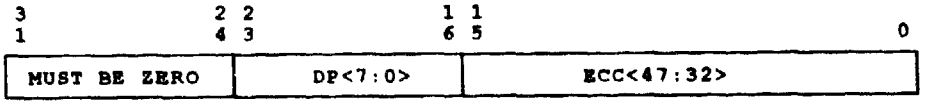
Failing DAL Register 3 (FDAL3)

FDAL3 is loaded on all quadwords of a writeback and on memory-space reads (hits or misses). Cache data is sampled at the beginning of the DAL read transaction. For read misses, the data is indeterminate.

FDAL3 locks if a writeback error occurs (XBEER<WFDQ0> or XBEER<WFDQ1> set), whether corrected or uncorrected. It is not locked on read data errors.

ADDRESS

E100 0038 (MDA-chip)



mab-p340-90

bits<31:24>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<23:16>

Name: Data Parity<7:0>

Mnemonic: DP<7:0>

Type: RO, 0

DP<7:0> contain the data parity bits related to DAL<63:0>.

bits<15:0>

Name: Error Correction Code<47:32>

Mnemonic: ECC<47:32>

Type: RO, 0

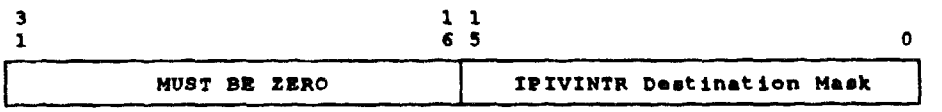
ECC<47:32> contains the error correction code bits related to DAL<63:0>.

Interprocessor Implied Vector Interrupt Generation Register (IPIVINTR)

The MAXMI has a generic procedure for generating interprocessor XMI IVINTR transactions. A byte-length I/O space write, using a byte-length instruction such as MOVB or CLRB, causes the MAXMI to generate an IVINTR command on the XMI.

ADDRESS

E101 0000–E101 FFFF
(A "virtual" register in MAXMI RAM)



msb-p341-90

bits<31:16>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<15:0>

Name: IPIVINTR Destination Mask
Mnemonic: None
Type: RO, 0

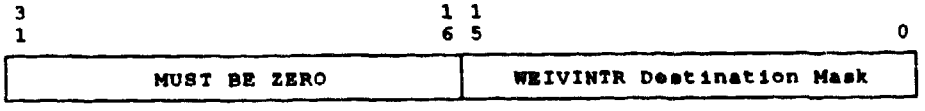
The low 16 bits of the address are used as the XMI destination mask during the IVINTR command. The MAXMI treats IVINTR transactions as normal I/O space writes until after the ACK for the IVINTR is received.

Write Error Implied Vector Interrupt Generation Register (WEIVINTR)

The MAXMI has a generic procedure for generating write error XMI IVINTR transactions. A byte-length I/O space write, using a byte-length instruction such as MOVB or CLRB, causes the MAXMI to generate an IVINTR command on the XMI.

ADDRESS

E102 0000–E102 FFFF
(A "virtual" register in MAXMI RAM)



msb-p342-90

bits<31:16>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<15:0>

Name: WEIVINTR Destination Mask
Mnemonic: None
Type: RO, 0

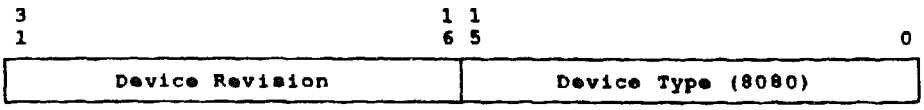
The low 16 bits of the address are used as the XMI destination mask during the IVINTR command. The MAXMI treats IVINTR transactions as normal I/O space writes until after the ACK for the IVINTR is received.

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 (MCA-chip)



bits<31:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W, 0

Identifies the revision level of the module in hexadecimal. The DREV field always reflects the letter revision of the module as follows:

KA65A 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

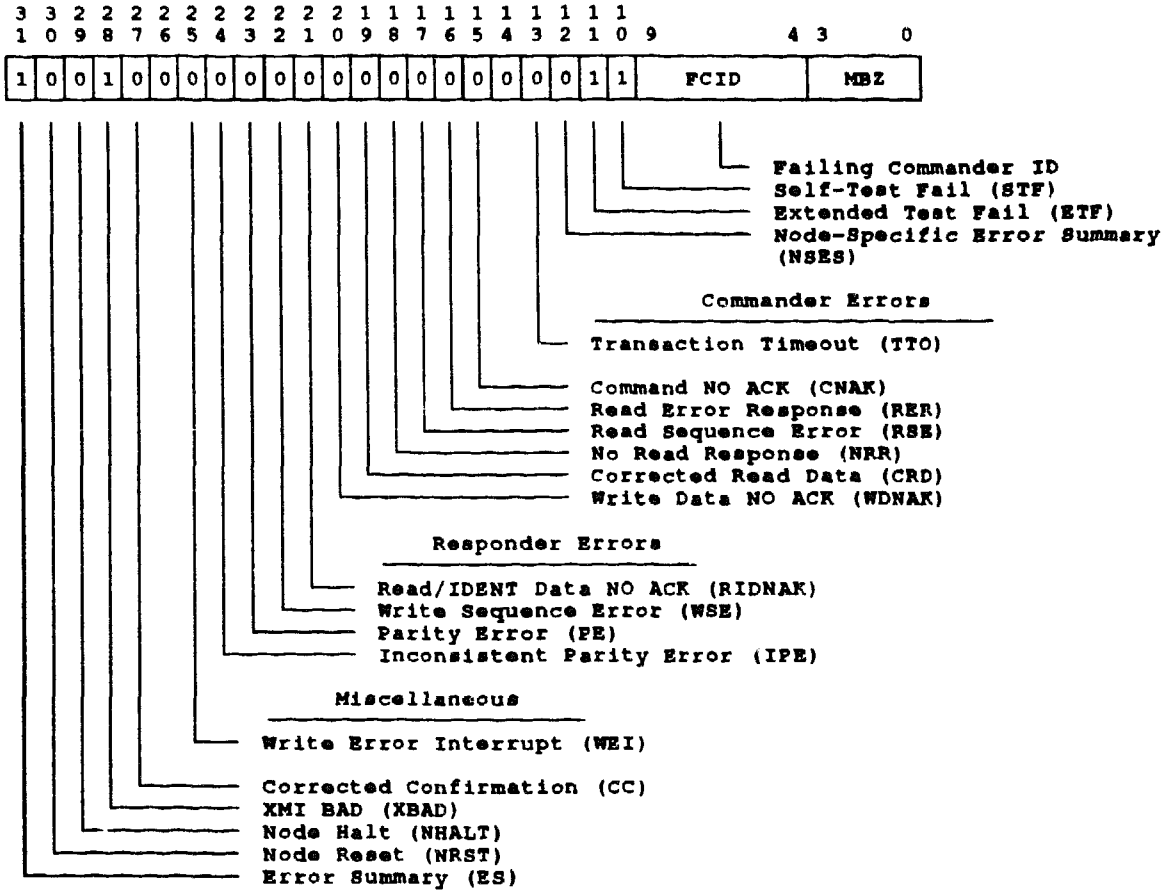
Type: R/W, 0

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 8080 (hex) for the KA65A CPU module.

Bus Error Register (XBER)

The Bus Error Register contains error status on a failed XMI transaction. This status includes the failed 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 (MCA-chip)*



msbp-343-90

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, 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 KA65A 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 KA65A 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: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<25>

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

When set, WEI indicates that the KA65A CPU module received a write error interrupt IVINTR transaction. 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 KA65A CPU module detected a parity error on an XMI cycle and the confirmation for the erroneous cycle was ACK. This indicates that at least one node (the responder) detected good parity during the cycle time that this KA65A CPU module detected a parity error. If this was a successful write to memory, it could leave the cache incoherent. The KA65A CPU module checks all XMI transactions, not just those it generates. When IPE sets, at least one of XBEER<MDAXPE> and XBEER<MCAXPE> sets and 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 MAXMI detected a parity error on an XMI cycle. When PE sets, at least one of XBEER<MDAXPE> and XBEER<MCAXPE> set and 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 KA65A CPU module, the MAXMI 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 KA65A 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 KA65A CPU module received a NO ACK confirmation. WDNACK 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 KA65A CPU module received a CRD read response, meaning that memory saw a parity error when reading data out of memory and corrected it. When CRD sets, Error Summary (ES) also sets.

bit<18>

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

When set, NRR indicates that a transaction initiated by this KA65A CPU module failed due to a read response timeout (no LOC, RER, CRD, or GRD). 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 KA65A CPU module failed due to a read sequence error, since the MAXMI 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 KA65A 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: –

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 KA65A CPU module failed due to a transaction timeout. The timeout counter is started when the MAXMI requests the XMI for a transaction. This bit is set only if retries fail. Write Data NO ACK (WDNAK), No Read Response (NRR), Command NO ACK (CNAK), and XBEER<OLR> indicate the cause of the timeout. If none of the bits are set, the MAXMI 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 the implemented bits in XBEER except XBEER<UWP> and <UUW>. 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 KA65A CPU module has not yet passed its extended test. This bit is cleared by console code when the KA65A CPU module passes its extended test. When ETF sets, Error Summary (ES) also sets.

bit<10>

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

When set, STF indicates that the KA65A CPU module has not yet passed its self-test. This bit is cleared by console code when the KA65A CPU module passes its self-test. When STF sets, Error Summary (ES) also sets.

bits<9:4>

Name: Failing Commander ID
Mnemonic: FCID
Type: RO

FCID holds the commander ID of a failing transaction.

bits<3:0>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

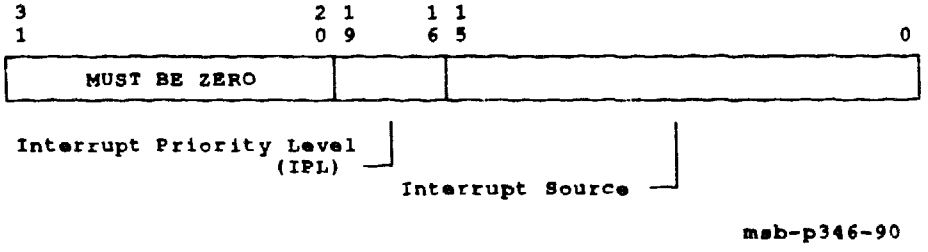
Failing Address Register (XFADR)

The Failing Address Register logs address and length information associated with a failing transaction. XFADR has an undetermined value on power-up. XFADR 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), XBER<13> (TTO), or XBEER<1> (SEO) are set at the beginning of the transaction. There are three interpretations of XFADR, depending on the XMI command. XFAER<31:28> determine the hex value of the XMI command.

ADDRESS

Nodespace base address + 0000 0008 (MCA-chip)

XFADR, when the XMI command 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>

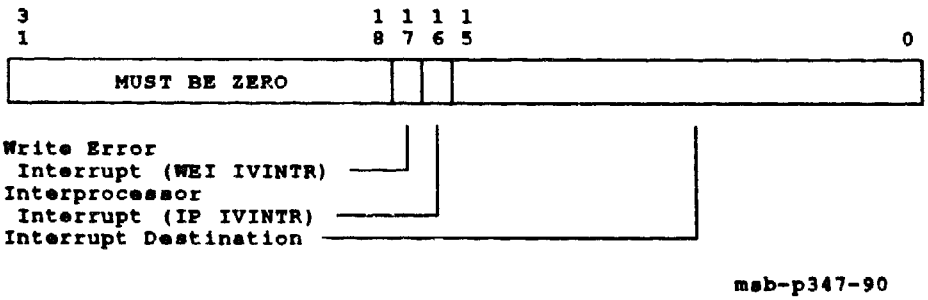
Name: Interrupt Source

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 the XMI command is F (hex), an IVINTR transaction:



bits<31:18>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bit<17>

Name: Write Error Interrupt
Mnemonic: WEI IVINTR
Type: RO

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

bit<16>

Name: Interprocessor Interrupt
Mnemonic: IP IVINTR
Type: RO

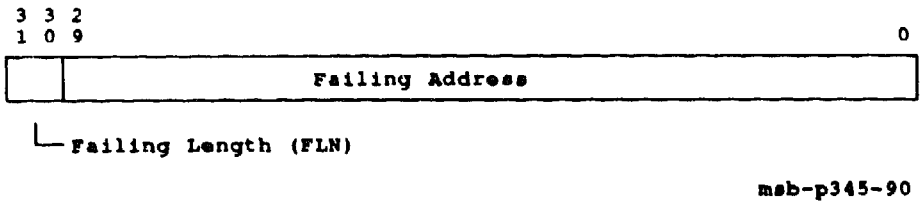
IP IVINTR sets if the implied vector interrupt request was for an interprocessor interrupt.

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 the XMI command is neither an IDENT transaction nor an IVINTR transaction:



msb-p345-90

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 (MCA-chip)

msb-p201-89

bits<31:0>

Name: XMI General Purpose Register

Mnemonic: XGPR

Type: R/W, 0

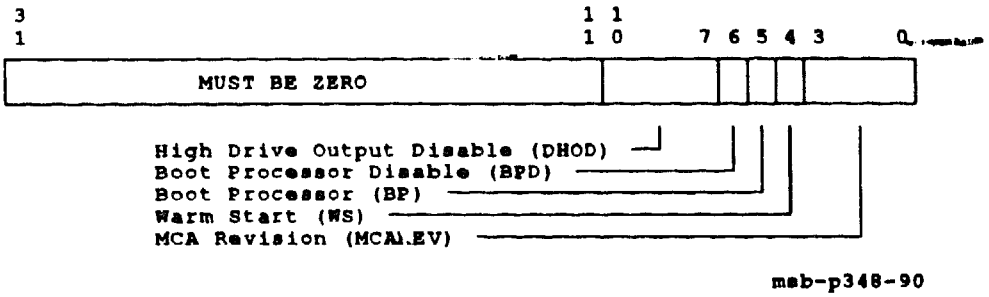
The general purpose register is used by self-test and during ROM-based diagnostics.

Node-Specific Control and Status Register (NSCSR)

NSCSR provides the KA65A CPU module control and status to the XMI bus.

ADDRESS

Nodespace base address + 0000 001C (MCA-chip)



bits<31:11>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<10:7>

Name: High Drive Output Disable

Mnemonic: HDOD

Type: R/W, 0

HDOD, when set, selectively disables individual stages of the 48 milliamp output drivers. Each bit corresponds to one stage. The output stage is disabled when set to a one. HDOD is used only for testing and must be zero for normal operation.

bit<6>

Name: Boot Processor Disable

Mnemonic: BPD

Type: R/W, 0

BPD is set by console code on power-up to indicate that this node is not eligible to become the boot processor. Following self-test, all KA65A 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<5>

Name: Boot Processor
Mnemonic: BP
Type: R/W, 0

BP is set by console code to indicate that this node is the boot processor.

bit<4>

Name: Warm Start
Mnemonic: WS
Type: RO, 0

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 used after a node reset.

bits<3:0>

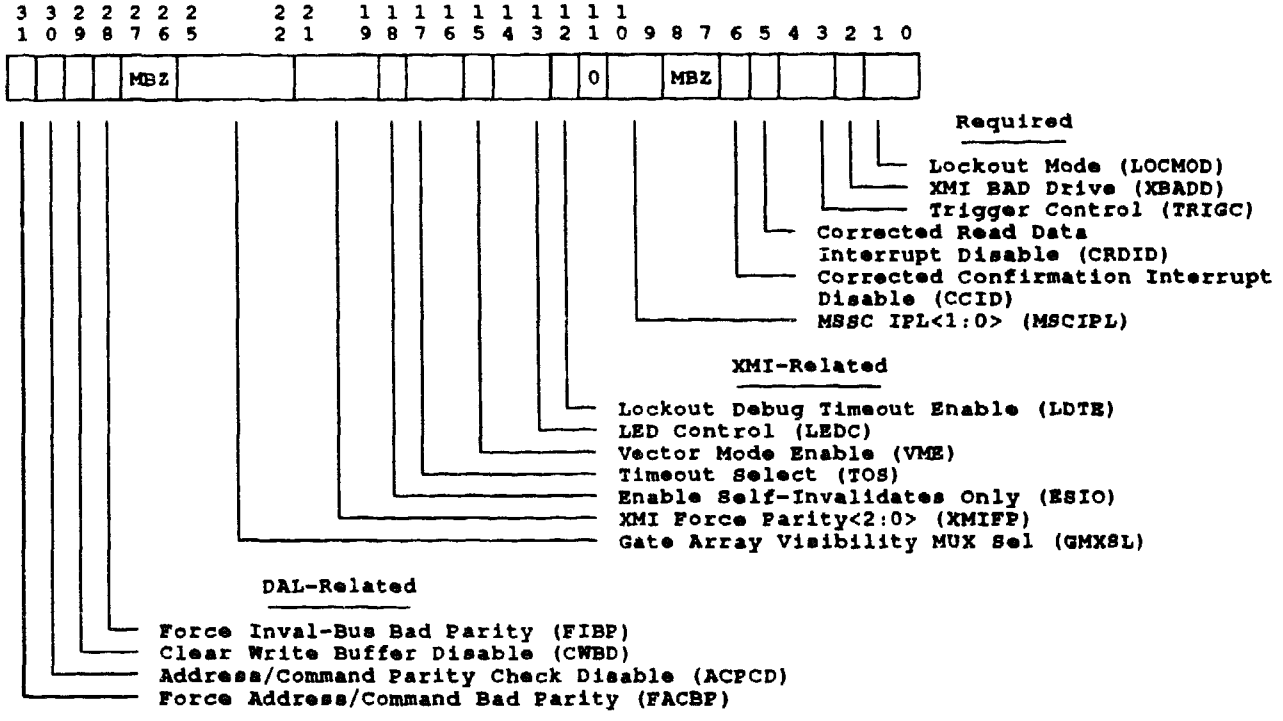
Name: MCA Revision
Mnemonic: MCAREV
Type: RO, 0

MCAREV contains the revision of the MCA-chip.

XMI Control Register (XCR)

The XMI Control Register contains toggles for various XMI and processor functions.

ADDRESS *XMI nodespace base address + 0000 0024 (MCA-chip)*



mab-p349-90

bit<31>

Name: Force Address/Command Bad Parity

Mnemonic: FACBP

Type: R/W, 0

FACBP, when set, forces bad parity on the Address/Command Parity (ACP) signals during all cache fills.

bit<30>

Name: Address/Command Parity Check Disable
Mnemonic: ACPCD
Type: R/W, 0

ACPCD, when set, disables parity checking on XMI Address<31:3> and CMD<3:0> for all DAL transactions.

bit<29>

Name: Clear Write Buffer Disable
Mnemonic: CWBD
Type: R/W, 1

CWBD, when set, causes the MAXMI to return RDY synchronous to the MP-chip without flushing the MAXMI invalidate queue. When CWBD is clear, the clear write buffer command proceeds normally.

bit<28>

Name: Force Inval-Bus Bad Parity
Mnemonic: FIBP
Type: R/W, 0

When FIBP is set, the invalidate addresses are forwarded from the MAXMI to the MC-chip with bad Inval-bus parity and must be clear during normal operation.

bits<27:26>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bits<25:22>

Name: Gate Array Visibility Multiplexer Select
Mnemonic: GMXSL
Type: RO, 0

GMXSL contains multiplexer selection lines to permit access to internal gate array signals.

bits<21:19>

Name: XMI Force Bad Parity<2:1>
Mnemonic: XMIFP
Type: R/W, 0

Each bit of XMIFP corresponds to the XMI P<2:0> parity bits. When an XMIFP bit is set, it forces bad parity on the XMI. Bad parity can be transmitted on XMI P<2:0>, during command/address cycles, and on XMI P<2> only, during data cycles.

bit<18>

Name: Enable Self-Invalidates Only
Mnemonic: ESIO
Type: R/W, 0

ESIO, when clear, causes the MAXMI to process invalidates from other nodes. ESIO, when set, causes the MAXMI to process invalidates for memory reads and writes generated by this node only. This allows a single CPU node to verify the operation of the invalidate logic. ESIO is set for diagnostic purposes only and must be clear during normal operation.

bits<17:16>

Name: Timeout Select
Mnemonic: TOS
Type: R/W, 0

TOS selects one of four timeout values used to detect both response and reattempt timeout conditions for commands and writebacks. TOS is for debug and diagnostic purposes only and must be cleared during normal operation. The timeout values are as follows:

TOS<1:0>	Timeout (Time/XMI Cycles)
00	16 ms/250K
01	16.38 μsec/256
10	4.01 μsec/64
11	1.92 μsec/32

bit<15>

Name: Vector Mode Enable
Mnemonic: VME
Type: R/W, 0

VME, when set, causes the MAXMI to drive a null cycle after every Ownership Read command address cycle. VME is set by console code when a vector processor is on the XMI.

bit<14:13>

Name: LED Control

Mnemonic: LEDC

Type: R/W, 0

LEDC controls the status selection of the KA65A CPU module LEDs as follows:

LEDC <1:0>	Description
0 0	The module's LED is lit when the Error Summary bit (XBER<ES>) is set.
0 1	The module's LED is lit if the backup cache is off or is in error transition mode.
1 0	The module's LED is lit when either the Error Summary bit (XBER<ES>) is set or if the backup cache is off or is in error transition mode.
1 1	Off

bit<12>

Name: Lockout Debug Timeout Enable

Mnemonic: LDTE

Type: R/W, 0

When LDTE is set and lockout is enabled (XCR<LOCMOD> = 00, 01, or 10 (binary)) both the lockout assertion timer (LAT) and the lockout de assertion timer (LDT) are forced to one microsecond. When lockout is disabled (XCR<LOCMOD> = 11 (binary)), LDTE has no effect.

bit<11>

Name: Reserved

Mnemonic: None

Type: –

Reserved; must be zero.

bits<10:9>

Name: MSSC Interrupt Priority Level<1:0>
Mnemonic: MSCIPL
Type: R/W, 0

MSCIPL selects the interrupt priority level for MSSC interrupt requests as follows:

MSSC IPL<1:0>	IPL (hex)
00	14
01	15
10	16
11	17

MSCIPL is set to 01 (binary) by console code for normal operation.

CAUTION: SSCCNR<IPL SEL> must match MSCIPL to ensure correct operation. If these values are different, the interrupt system operation is UNDEFINED.

bits<8:7>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<6>

Name: Corrected Confirmation Interrupt Disable
Mnemonic: CCID
Type: R/W, 0

CCID controls the generation of interrupts caused by corrected confirmations. A zero enables interrupts; a one disables interrupts.

bit<5>

Name: Corrected Read Data Interrupt Disable
Mnemonic: CRDID
Type: RO, 0

CRDID controls the generation of interrupts caused by corrected read data. A zero enables interrupts; a one disables interrupts.

bits<4:3>

Name: Trigger Control
Mnemonic: TRIGC
Type: RO, 0

TRIGC controls the setting of the XMI TRIGGER L signal. The default code of zero means that the signal is never asserted. The codes of one, two, or three are undefined.

bit<2>

Name: XMI BAD Drive
Mnemonic: XBADD
Type: R/W, 1

On reads, XBADD indicates the state of the module's driver for the XMI BAD L signal, which could be different from XBER<XBAD>. This is because XBADD is used to assert or deassert XMI BAD L, while XBER bit <25> XBAD only shows the state of XMI BAD L. Writes to XBADD cause the state to be driven on the wired-OR XMI BAD L signal by this node. Writing a one asserts XMI BAD L; a zero deasserts the node's driver for XMI BAD L, which deasserts the signal if no other nodes are asserting it.

bits<1:0>

Name: Lockout Mode
Mnemonic: LOCMOD
Type: R/W, 0

LOCMOD determines the lockout assertion timer (LAT) and lockout deassertion timer (LDT).

Failing Address Extension Register (XFAER)

The Failing Address Extension Register logs command, address, and write mask information associated with a failing transaction.

XFAER is the higher 32-bits of a 64-bit register formed by concatenating XFADR and XFAER. The 64-bit register is used to log command, address, length, and write mask information associated with a failing transaction.

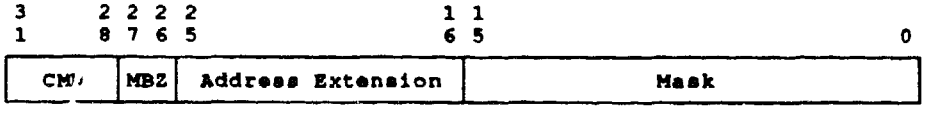
XFADR and XFAER latch on the first XMI bus error. The following rules govern the overwriting of the information in the registers:

- If no error information is in the registers, they are written on the first hard or soft error.
- If soft error information is being latched, the registers are not changed on subsequent soft errors.
- If soft error information is being latched, the registers are overwritten by a hard error.
- If hard error information is being latched, the information is not changed on subsequent errors.

Setting of one or more of the following XBER and XBEER bits are hard errors and force the latching of XFADR and XFAER: XBER<20> (WDNAK), XBER<18> (NRR), XBER<17> (RSE), XBER<16> (RER), XBER<15> (CNAK), XBER<13> (TTO), and XBEER<1> (SEO).

ADDRESS

Nodespace base address + 0000 002C (MCA-chip)



msb-p200-89

bits<31:28>

Name: Command

Mnemonic: CMD

Type: RO

CMD logs the value of XMI D<63:60> during the command cycle of a failing transaction. The field contains the command code of the transactions during the command cycle.

bits<27:26>

Name: Reserved
Mnemonic: None
Type: –

Unused; must be zero.

bits<25:16>

Name: Address Extension
Mnemonic: None
Type: RO

The Address Extension field logs the value of XMI D<57:48> during the command cycle of a failing transaction. Address Extension contains address bits<38:29> of the specified address in read and write transactions.

bits<15:0>

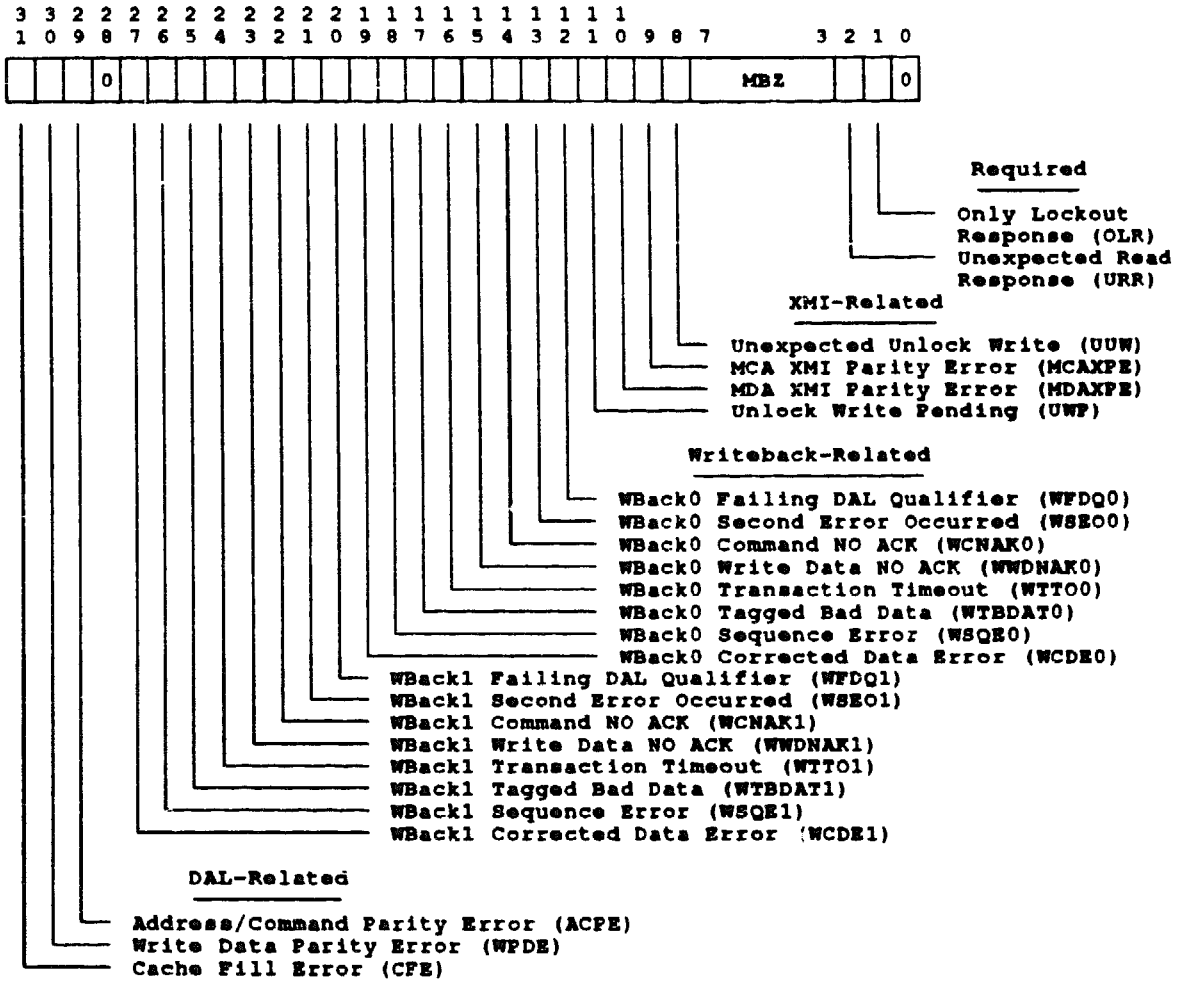
Name: Mask
Mnemonic: None
Type: RO

The Mask field logs the value of XMI D<47:32> during the command cycle of a failing transaction. It contains the write mask for write transactions and is undefined for other transactions.

Bus Error Extension Register (XBEER)

The Bus Error Extension Register contains error status on XMI-related, writeback-related, and DAL-related transactions. This status includes parity errors, NO ACKs, and timeouts.

ADDRESS *Nodespace base address + 0000 0034 (MCA-chip)*



mab-p350-90

bit<31>

Name: Cache Fill Error
Mnemonic: CFE
Type: R/W 1C

When this bit is set, an error occurred in the second, third, or fourth quadword of an MP-chip read request or in any quadword of an MC-chip read request (write and run). The cause of the error can be found in XBER <RSE, RER, TTO, and NRR>. A cache fill abort is sent to the MC-chip as a result of this error. If CFE is set, XBER <NSES> is also set.

bit<30>

Name: Write Data Parity Error
Mnemonic: WDPE
Type: R/W 1C

WDPE sets when a parity error is detected on the data bus during a cache-off memory space write or I/O space write transaction. An XMI write is not performed. This detection is disabled if DCSR<WDPCD> is set. If WDPE is set, XBER <NSES> is also set.

bit<29>

Name: Address/Command Parity Error
Mnemonic: ACPE
Type: R/W 1C

ACPE sets when the MAXMI detects a parity error on ABUS<31:3> and CMD<3:0>.

bit<28>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<27>

Name: Writeback1 Corrected Data Error
Mnemonic: WCDE1
Type: R/W 1C

WCDE1 sets when a single-bit ECC error is detected and corrected by the MAXMI before the Disown Write is issued on the XMI bus. If WCDE1 is set, XBER <NSES> is also set.

bit<26>

Name: Writeback1 Sequence Error

Mnemonic: WSQE1

Type: R/W 1C

WSQE1 is set during a writeback involving writeback queue 1, if the second, third, or fourth quadword transaction on the data address line is not completed due to an address/command parity error. The writeback is issued on the XMI bus with a tagged bad data cycle. If WSQE1 is set, XBER <NSES> is also set.

bit<25>

Name: Writeback1 Tagged Bad Data

Mnemonic: WTBDAT1

Type: R/W 1C

If this bit is set, an uncorrectable (multiple-bit) ECC error was detected by the MAXMI during a data address line data cycle. The MAXMI issues the Tag Bad Data command instead of a Disown Write on the XMI bus for the corrupted hexword. If WTBDAT1 is set, XBER <NSES> is also set.

bit<24>

Name: Writeback1 Transaction Timeout

Mnemonic: WTTO1

Type: R/W 1C

When a writeback transaction is initiated from writeback queue 1 by the MAXMI, a timeout counter is started. If the transaction does not complete before the timeout interval expires, this bit is set. WCNAK1 and WWDNAK1 indicate the cause of the timeout. If neither bit is set, the MAXMI was never granted the XMI bus for the transaction. If XBEER <WTTO1> is set, XBER <NSES> is also set.

bit<23>

Name: Writeback1 Write Data NO ACK

Mnemonic: WWDNAK1

Type: R/W 1C

If the transaction timeout period (WTTO1) expires while the MAXMI is issuing the Disown Write command and is receiving a NO ACK in response to the transmission of any data cycles, this bit is set. XBEER <WTTO1> should also be set. If WWDNAK1 is set, XBER <NSES> is also set.

bit<22>

Name: Writeback1 Command NO ACK
Mnemonic: WCNAK1
Type: R/W 1C

This bit is set if the transaction timeout period (WTTO1) expires while the MAXMI is receiving a NO ACK in response to transmission of the Disown Write command cycle. XBEER <WTTO1> should also be set. If WCNAK1 is set, XBER <NSES>is also set.

bit<21>

Name: Writeback1 Second Error Occurred
Mnemonic: WSEO1
Type: RO, 0

When set, this bit indicates that a second writeback hard error occurred while another one was being reported.

bit<20>

Name: Writeback Failing DAL Qualifier
Mnemonic: WFDQ
Type: R/W 1C

This bit is set with XBEER0 <WTBDAT1> or XBEER0 <WBCDE1> if the corresponding error locked the FDAL register. If this bit is set, writing a one to it clears the bit and enables the reloading of the FDAL register.

bit<19>

Name: Writeback0 Corrected Data Error
Mnemonic: WCDE0
Type: R/W 1C

If this bit is set, a single-bit ECC error was detected on the data address lines and corrected by the MAXMI before the Disown Write was issued on the XMI bus. If WCDE0 is set, XBER <NSES> is also set.

bit<18>

Name: Writeback0 Sequence Error
Mnemonic: WSQE0
Type: R/W 1C

During a writeback involving writeback queue 0, this bit is set if the second, third, or fourth quadword transaction on the data address lines does not complete successfully due to an address/command parity error. The writeback is issued on the XMI bus with a tagged bad data cycle. If WSQE0 is set, XBER <NSES> is also set.

bit<17>

Name: Writeback0 Tagged Bad Data
Mnemonic: WTBDAT0
Type: R/W 1C

If this bit is set, an uncorrectable (multiple-bit) ECC error was detected by the MAXMI on a data address line data cycle. The MAXMI issues the Tag Bad Data command instead of a Disown Write on the XMI bus for the corrupted hexword. If WTBDAT0 is set, XBER <NSES> is also set.

bit<16>

Name: Writeback0 Transaction Timeout
Mnemonic: WTTO0
Type: R/W 1C

When a writeback transaction is initiated from writeback queue 0 by the MAXMI, a timeout counter is started. If the transaction does not complete before the timeout interval expires, this bit is set. WCNAK0 and WWDNAK0 indicate the cause of the timeout. If neither bit is set, the MAXMI was never granted the XMI bus for the transaction. If XBEER <WTTO0> is set, XBER <NSES> is also set.

bit<15>

Name: Writeback0 Write Data NO ACK
Mnemonic: WWDNAK0
Type: R/W 1C

When set, this bit indicates that Disown Write data cycles were continually NO ACKed and the transaction eventually timed out (causing WTT0 <16> to be set). If WWDNAK0 is set, XBER <NSES> is also set.

bit<14>

Name: Writeback0 Command NO ACK
Mnemonic: WCNAK0
Type: R/W 1C

This bit is set if the transaction timeout period (WTTO0) expires while the MAXMI is receiving NO ACKs during transmission of the Disown Write command cycle. XBEER <WTTO0> should also be set. If WCNAK0 is set, XBER <NSES> is also set.

bit<13>

Name: Writeback0 Second Error Occurred
Mnemonic: WSEO0
Type: RO, 0

When set, this bit indicates that a second writeback hard error occurred while another one was being reported.

bit<12>

Name: Writeback Failing DAL Qualifier
Mnemonic: WFDQ
Type: R/W 1C

This bit is set with XBEER0 <WTBDAT0> or XBEER0 <WBCDE0> if the corresponding error locked the FDAL register. If this bit is set, writing a one to this bit clears the bit and enables the reloading of the FDAL register.

bit<11>

Name: Unlock Write Pending
Mnemonic: UWP
Type: R/W 1C

The MAXMI sets this bit when an Interlock Read is generated by this node. It clears the bit when a subsequent Unlock Write is generated. The setting and clearing of this bit is not gated by the successful completion of the XMI transaction.

bit<10>

Name: MDA XMI Parity Error
Mnemonic: MDAXPE
Type: R/W 1C

This bit is set whenever the MDA chip detects a parity error on XMI D<63:0>. If MDAXPE is set, XBER <PE> is also set.

bit<9>

Name: MCA XMI Parity Error
Mnemonic: MCAXPE
Type: R/W 1C

This bit is set whenever the MCA detects a parity error on XMI D<63:0>, XMI F<3:0>, or XMI ID<5:0>. If MCAXPE is set, XBER <PE> is also set.

bit<8>

Name: Unexpexcted Unlock Write
Mnemonic: UUW
Type: R/W 1C

This bit is set when the MAXMI issues an Unlock Write without the corresponding Interlock Read having preceded it.

bits<7:3>

Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

bit<2>

Name: Unexpected Read Response
Mnemonic: URR
Type: RO, 0

When set, this bit indicates that the CPU received a read response when it was not expecting one. A node is defined as not expecting a response when there are no outstanding reads or identifiers for that particular commander ID.

bit<1>

Name: Only Lockout Response
Mnemonic: OLR
Type: RO, 0

When set, this bit indicates that the CPU received only lockout responses while attempting a read-type transaction. This identifies the timeout case due to failure of the lockout mechanism to provide timely access to a resource such as interlock or ownership of a memory location. If OLR is set, XBER <TTO> is also set.

bit<0>

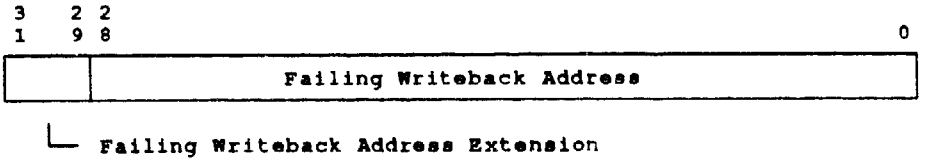
Name: Reserved
Mnemonic: None
Type: –

Reserved; must be zero.

Writeback 0 Failing Address Register (WFADR0)

WFADR0 is loaded at the start of every XMI writeback transaction. It is set as the result of Transaction Timeout (XBER<TTO> bit <13>), Writeback0 Tagged Bad Data (XBEER<WTBDAT0> bit <17>), and Writeback0 Sequence Error (XBEER<WSQE0> bit <18>) being set. WFADR0 unlocks when these error bits clear.

ADDRESS *Nodespace base address + 0000 0040 (MCA-chip)*



msb-p351-90

bits<31:29>

Name: Failing Writeback Address Extension

Mnemonic: None

Type: RO, 0

The Failing Writeback Address Extension field logs the writeback queue 0 address extension during the command cycle of a failing writeback transaction as follows:

WFADR0	Value
<31>	XMI D<29>
<30:29>	XMI D<49:48>

bits<28:0>

Name: Failing Writeback Address

Mnemonic: None

Type: RO, 0

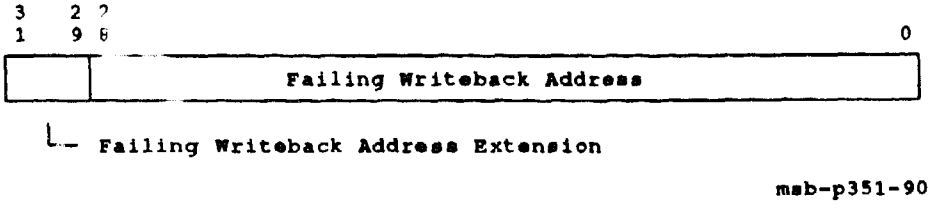
The Failing Writeback Address field logs the writeback queue 0 XMI D<28:0> during the command cycle of a failing writeback transaction.

Writeback 1 Failing Address Register (WFADR1)

WFADR1 is loaded at the start of every XMI writeback transaction. It is set as the result of Transaction Timeout (XBER<TTO> bit <13>), Writeback1 Tagged Bad Data (XBEER<WTBDAT1> bit <25>), and Writeback1 Sequence Error (XBEER<WSQE1> bit <26>) being set. WFADR1 unlocks when these error bits clear.

ADDRESS

Nodespace base address + 0000 0044 (MCA-chip)



msb-p351-90

bits<31:29>

Name: Failing Writeback Address Extension

Mnemonic: None

Type: RO, 0

The Failing Writeback Address Extension field logs the writeback queue 1 address extension during the command cycle of a failing writeback transaction as follows:

WFADR1	Value
<31>	XMI D<29>
<30:29>	XMI D<49:48>

bits<28:0>

Name: Failing Writeback Address

Mnemonic: None

Type: RO, 0

The Failing Writeback Address field logs the writeback queue 1 XMI D<28:0> during the command cycle of a failing writeback transaction.

2.9 KA65A CPU Module Initialization, Self-Test, and Booting

This section gives the KA65A CPU module initialization overview, describes the results of initialization, and discusses the bootstrapping or restarting of the operating system.

2.9.1 Initialization Overview

The three ways to reset the KA65A CPU module are:

- Power-Up Sequence—When the VAX 6000 Model 500 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 2.11 for a discussion of error handling. The *VAX 6000 Series Owner's Manual* discusses the operation of the system console.

In response to a power-up or system reset, the KA65A 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.)

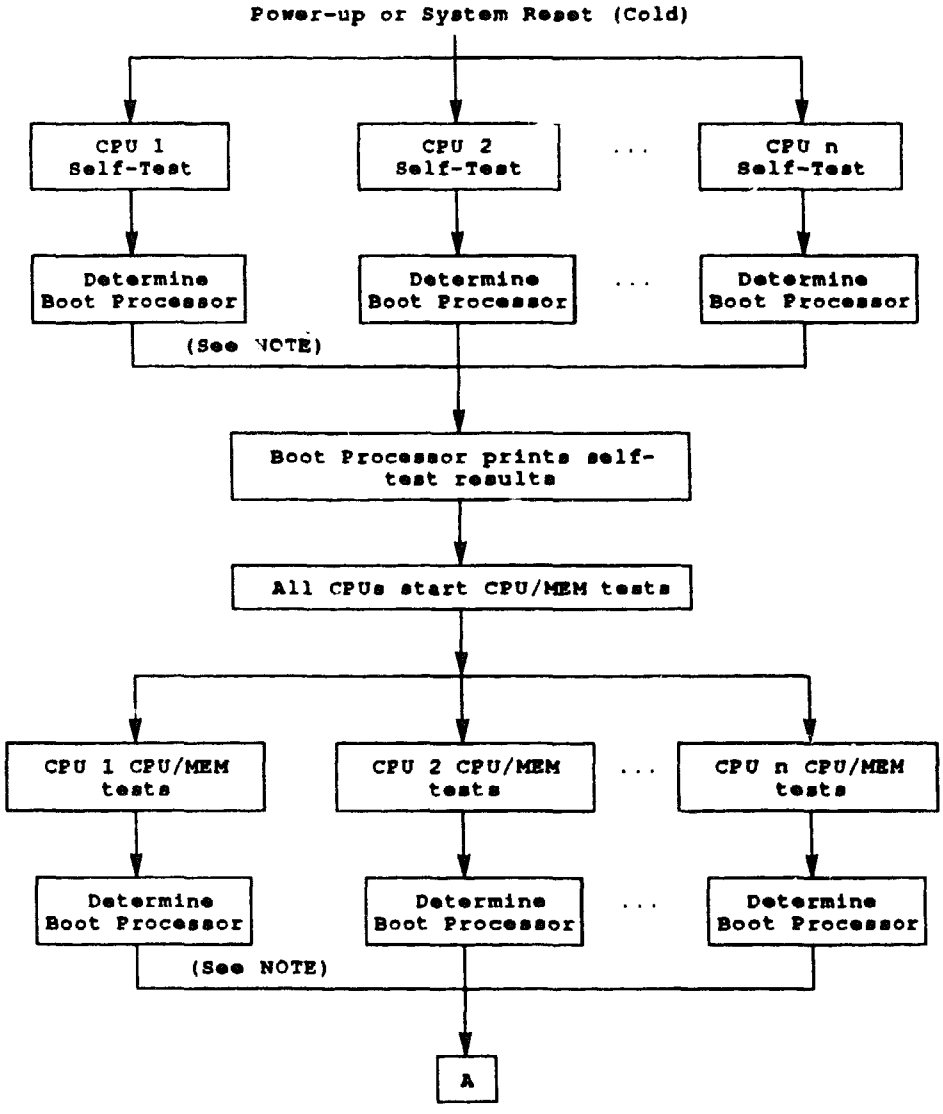
2 All CPUs start executing the console program at E004 0000 in ROM. The console program initializes the registers and executes a complete ROM-based diagnostic (RBD) self-test and extended tests.

3 The operating system initialization code performs the final system initialization.

2.9.2 Detailed Initialization Description

The following is a flowchart and summary of the initialization process.

Figure 2–27 Initialization Flowchart

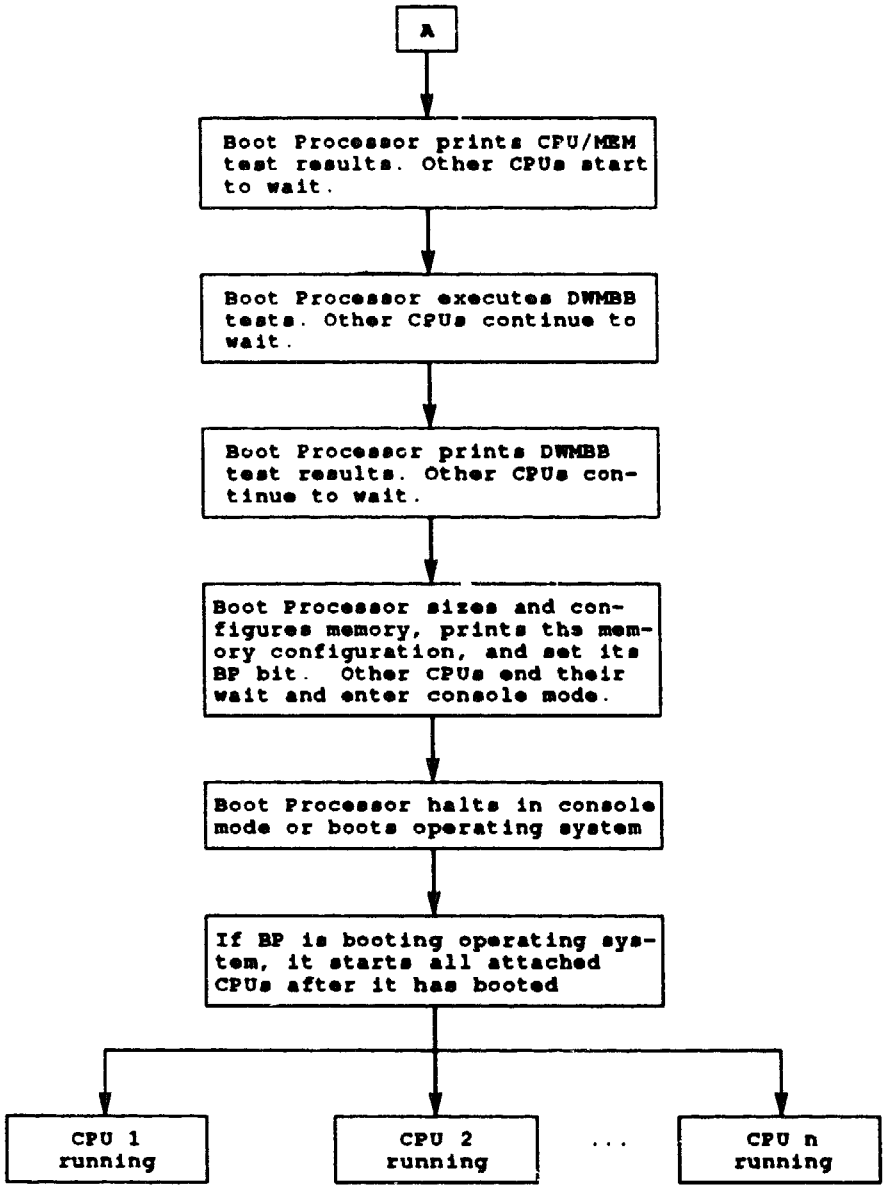


NOTE: The second determination of the Boot Processor occurs even if the original Boot Processor passes all memory tests.

mab-p352-90

Figure 2–27 Cont'd on next page

Figure 2–27 (Cont.) Initialization Flowchart



When a CPU module resets, it first performs a microcode self-test that briefly checks the MP-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 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 DWMBBs in the system. If the DWMBB tests are successful, the BP lights the DWMBB/A's yellow LED and the DWMBB /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 DWMBB tests are performed. If a warm restart occurs after the operating system has elected a new primary processor (via a CCA field), a third boot processor is selected.

On node reset, the CPU initializes its state and runs self-test. The console program performs an additional CPU initialization before the operating system starts the processor running.

When the console initialization is completed, the boot processor either restarts the operating system (if a warm restart), boots the operating system, or halts in console mode. The secondary processors enter console mode and wait for the operating system to issue console commands that start them running. See Table 2–17 for the console-initialized state of the CPU module's registers.

2.9.2.1 Initialization State Summary

When a KA65A 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 2–14. The console then runs its self-test, and in the process loads new states into many of the registers. The KA65A 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.

Table 2–17 CPU Module Register's 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	0000
VINTSR[9]	0000	0000	0000	0000	0000	0000	0000	000x
PCSTS[10]	0000	0000	0000	0000	000x	xxxx	000x	x000
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	0000	0101
SSCBTR	0000	0000	0000	0000	1001	0000	0000	0000
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	1010	0000	0000	0000	0000
EEADMR	0000	0000	0000	0000	0111	1111	1111	1100
XDEV[14]	xxxx	xxxx	xxxx	xxxx	1000	0000	1000	0000
XBER[15]	0000	0000	0000	0000	0000	00xx	xxxx	xxxx
XBEER	0000	0000	0000	0000	0000	0000	0000	0000
XCR	0000	0000	0000	0000	x000	0010	0000	0000

Table 2–17 (Cont.) CPU Module Register's Console-Initialized State

Register	Bit Settings[1]							
NSCSR[16]	0000	0000	0000	0000	0000	0111	1xxxx	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. MF-Chip Present, bit<1>, should always be 1 unless the MF-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 MC-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 KA65A CPU module.
[15]XBER<FCID> and XBER<FCMD> are latched during self-test and extended test. The final values are unpredictable.
[16]NSCSR<BP>, NSCSR<BPD>, and NSCSR<WS> are set as appropriate.

2.9.2.2 Power-Up Initialization

Entry to the initialization sequence starts at physical address E004 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 MSSC 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 NSCSR<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 NSCSR<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 2.9.2.3 for the warm start initialization and to Section 2.9.2.4 for a node reset.

Significant events of a power-up initialization include the following:

- Self-test is run.

- SSCCNR is configured.
- TIR1 is started.
- SSCBTR is set for its bus timeout interval.
- The interrupt vectors for the programmable timers are set.
- The primary cache and backup cache 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 are disabled.
- EEADMR is set.
- The contents of the EEPROM are verified by checksum.
- SSCCNR is set to allow the CTRL/P key to be the console break character.
- The initialization of the console scratch area is completed.
- If appropriate, the NSCSR<Boot Processor Disable> bit is set.
- The MAXMI is initialized:
  - — XDEV is loaded with the appropriate module revision and device type values.
  - — All error bits in NSCSR and XCR are cleared and those bits that require initialized state are initialized.
  - — SSCCNR<IPL SEL> and XFADR<IPL> are set to the 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 2.9.2.5. The BP prints the initial self-test results on the console terminal and sets its NSCSR<BP>.
- All CPU nodes that pass initial self-test execute the CPU/MEM tests and the multiprocessor exerciser tests.

- If these tests are successful, the CPU again clears XBER<XBAD> and NSCSR<BP>.
- Vector registers are initialized (if present) and the vector processor is enabled if it passed self-test.
- The BP is redetermined. This newly determined BP prints the results of the CPU/MEM test and the multiprocessor exerciser tests on the console terminal.
- The BP runs the DWMBB self-test and prints the results on the console terminal. The status of each VAXBI node is printed.
- The BP configures system memory, as described in Section 2.9.2.6 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 2.9.2.7.
- The BP's NSCSR<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 2.9.3, is entered.

2.9.2.3 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 DWMBB 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.

2.9.2.4 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.
- DWMBB self-test is not run. No initialization of the DWMBB 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.

2.9.2.5 Boot Processor Determination

Each processor examines all other CPU XBER<STF> (Self-Test Failed), XBER<ETF> (Extended Test Failed), and NSCSR<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 NSCSR<BP>. All nonboot processors wait for the designated BP to set its NSCSR<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 NRSCR<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.

2.9.2.6 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 NRCSR<Boot Processor Disabled> bit, causing the nonboot processors, if any, to determine a new BP. The new BP repeats the memory configuration attempt.

2.9.2.6.1 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.

The VAX 6000 Model 500 supports interleave sets of mixed densities. Interleave sets are built from like-sized memory modules or memory modules whose cumulative value is equal to the largest memory module in the set. For example, a 64-Mbyte memory module can be combined with another 64-Mbyte memory module to form a two-way interleave set. If another 64-Mbyte memory module is not present, two 32-Mybte memory modules can be used to complete the interleave set.

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.

2.9.2.6.2 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.

2.9.2.7 DWMBB Configuration

The console program performs minimal initialization of the DWMBBs following self-test. The initialization performed for each DWMBB 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 (under 512 Mbytes).

2 The BICSR (bb+04) has its BI Broke bit cleared.

3 The DMA-B Transmit Buffer is disabled under these conditions:

- The DWMBB/A module Device Register shows less than 10 (decimal).
- The system contains five or more XMI commanders and the DWMBB/B module device register shows a revision of less than 0A (hex).

If a DWMBB/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.

2.9.3 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 processors do not restart system software unless the control panel lower key switch is set to Auto Start.

2.9.3.1 Operating System Restart

The boot processor console program attempts to start/restart the operating system whenever one of the following events occurs:

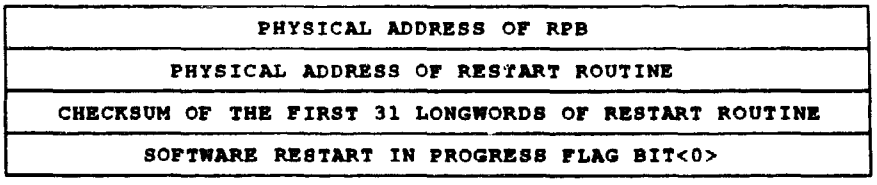
- Power is restored to the processor. If the memory was kept valid by the optional battery backup, it is a warm start; otherwise, it is a cold start.
- A system reset occurs. This is treated as a cold start.
- The running processor halts due to an error halt. This is a restart. A CTRL/P from the console 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 2–28 shows the RPD format.

Figure 2–28 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.

2.9.3.2 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.

2.9.3.3 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

2.9.3.4 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 Model 500 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 MSSC 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 or the boot block 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.

2.9.3.5 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 MSSC 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 2–17.

10 Transfer control to the boot primitive.

11 Transfer control to the memory image of VMB or the boot block 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.

2.9.3.6 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 2–18 shows how the registers are used.

Table 2–18 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 boot adapter
	<3:0>	VAXBI node number (if necessary)
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

2.10 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 Model 500 system.

The console program runs on each processor of a multiprocessor VAX 6000 Model 500 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 Model 500. There is no communication between secondary (nonboot) processors.

2.10.1 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 nonboot 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.

2.10.2 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 for each node. The header contains status information that must be visible systemwide. The buffers, one pair for each XMI node, are used for passing messages between processors.

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 MSSC 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 the existing CCA is as follows:

1 Next = highest memory address in the 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.

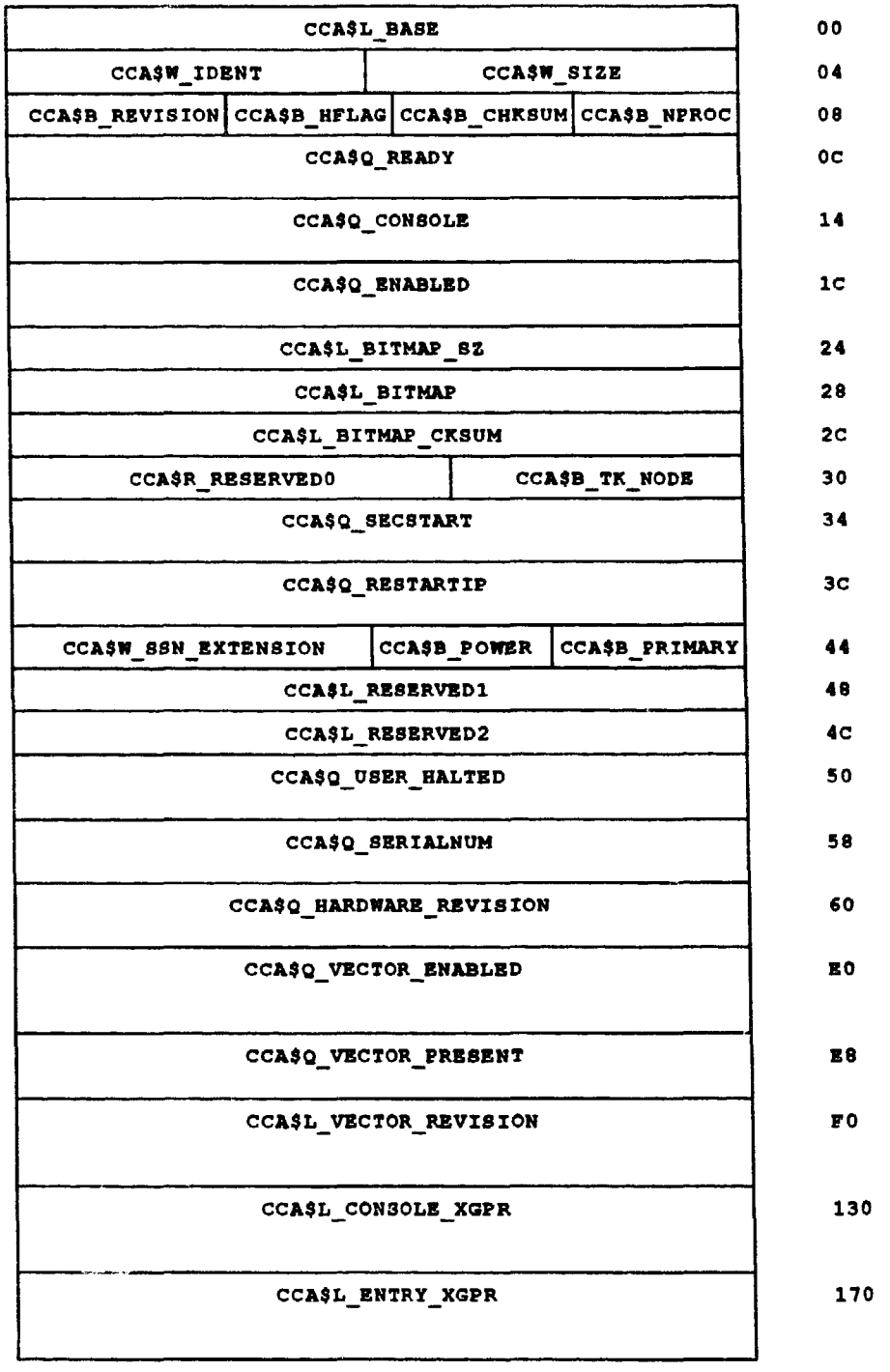
6 If sum = (next + CCA$B_CHKSUM), then "Exit with CCA found at next."

7 Next = next–512.

8 Goto Step 2.

The overall layout of the CCA is shown in Figure 2–29. The contents of the fields are described in Table 2–19.

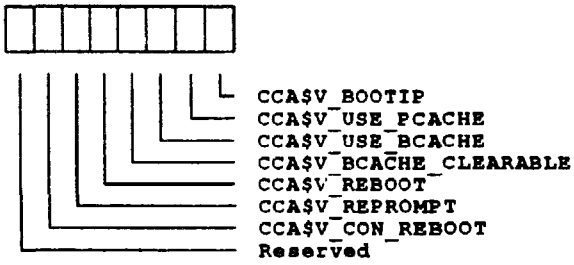
Figure 2–29 CCA Layout



msb-p355-90

Table 2–19 CCA Fields

Field	Description
CCA$L_BASE	Physical address of the base of the CCA.
CCA$W_IDENT	The ASCII characters "CC".
CCA$W_SIZE	The size, in bytes, of the CCA.
CCA$B_REVISION	The revision number for the CCA.
CCA$B_HFLAG	Systemwide status flags:



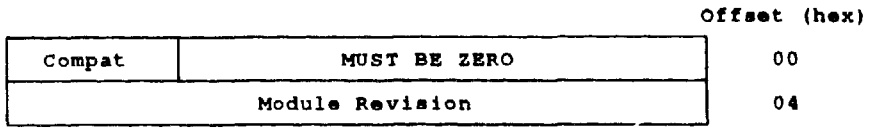
CCA$V_CON_ REBOOT	This bit is used by the console program to trigger a reboot following a failed restart.
CCA$V_REPROMPT	This bit is used by the console program to support the SET CPU command.
CCA$V_REBOOT	This bit is tested whenever the console is entered as a result of a CTRL/P or a node halt. If the bit is set, the operating system is requesting a reboot. The system is rebooted from the default boot device. The front panel lower key switch does not inhibit such a reboot. This bit is ignored if the key switch is in the Secure position.
CCA$V_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$V_USE_ BCACHE	When set, the external (backup) cache is to be enabled by the operating system. This bit is set based on a value in the EEPROM. The default setting is a one.
CCA$V_USE_ PCACHE	When set, the MP-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_BOOTIP	When set, a bootstrap is being attempted. This prevents repeated attempts to bootstrap after a failure.

Table 2–19 (Cont.) CCA Fields

Field	Description
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$B_NPROC	The number of processors supported by the CCA. The normal value is 16. The maximum value is 64, limited by the bitmasks in the other fields.
CCA$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$Q_CONSOLE	A bitmask indicating the processors known to be in console mode. The appropriate bit is set and cleared by each processor as it enters and leaves console mode.
CCA$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$L_BITMAP_ CKSUM	Reserved; not used.
CCA$R_RESERVED0	Reserved; not used.
CCA$B_TK_NODE	This field is used to pass to the operating system the XMI (in bits <7:4>) and VAXBI (in bits <3:0>) node numbers of the adapter that controls the TK tape drive. This field is set initially from a value stored in the EEPROM. If the initially specified node does not contain a TK tape drive adapter, the console program searches each VAXBI for a suitable adapter. The serach starts with the highest XMI and from the highest VAXBI node ID on each VAXBI. The field sets to the location of the adapter, or zero if no adapter is found.
CCA$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$Q_RESTARTIP	A bitmask indicating which processors are currently attempting restarts. Multiple flags are maintained to allow simultaneous error restarts to be performed. The operating system clears these fields if restart or boot succeeds.
CCA$W_SSN_ EXTENSION	The highest two bytes of the serial number.
CCA$B_POWER	An ASCII character representing the type of power system.
CCA$B_PRIMARY	The XMI node number of the primary processor. The console selects a primary or boot processor, which may be changed by the operating system. If a reboot occurs, the console references these bits to reboot the operating system.
CCA$L_RESERVED1	Reserved; not used.
CCA$L_RESERVED2	Reserved; not used.

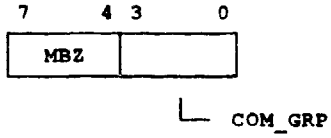
Table 2–19 (Cont.) CCA Fields

Field	Description
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 contains 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.

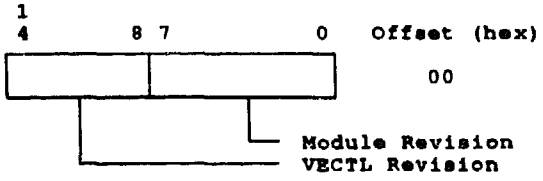
The layout of the Compat field is:



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.

Table 2–19 (Cont.) CCA Fields

Field	Description
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_ENTRY_XGPR	The XGPR value used by the 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* = Base address of CCA + 512 + (*n* * 168)

The layout of the buffer area is shown in Figure 2–30, and the buffer fields are described in Table 2–20.

Figure 2-30 Layout of XMI Node Buffers

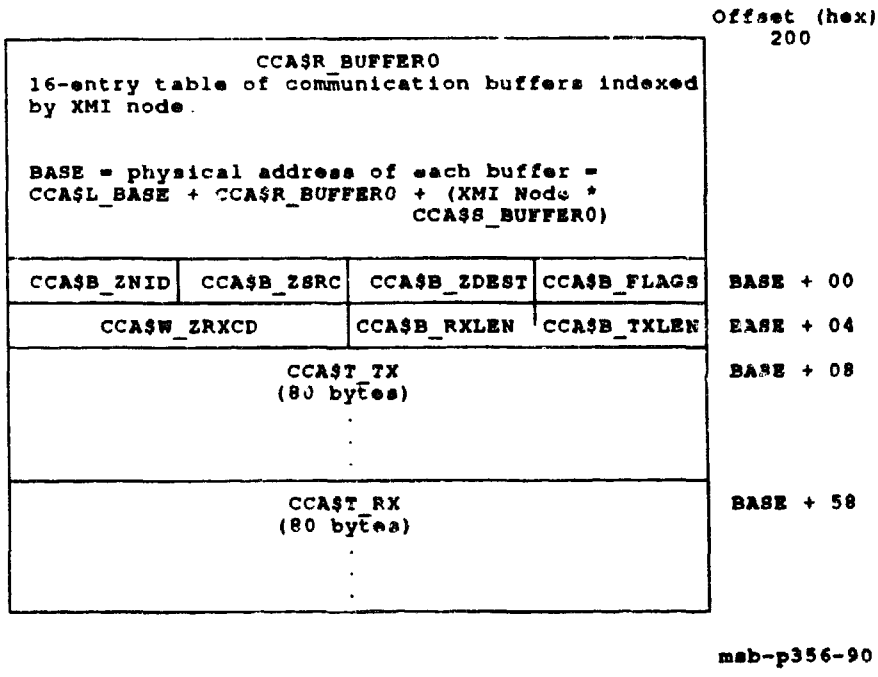


Table 2-20 Buffer Fields

Field	Description
CCA$R_BUFFER0	A 16-entry table of communication buffers indexed by the XMI node. See Figure 2–30 for more information.
CCA$B_ZNID	If CCA$B_ZNID is set, this field contains the XMI node number of the originator of the Z connection.
CCA$B_ZSRC	If CCA$B_ZSRC is set, this field contains the XMI node number of the node transmitting "Z" command data to this node.
CCA$B_ZDEST	When CCA$V_ZDEST is set, this field contains the XMI node number of the node receiving the "Z" command data that this node is sending. If the low four bits of this field identify a node that is a DWMBB, the high order four bits contain the destination VAXBI node number.
CCA$B_FLAGS	Status flags:

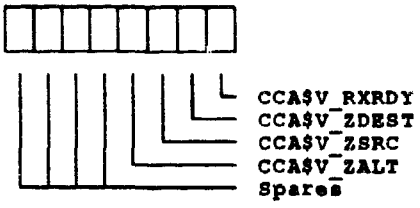
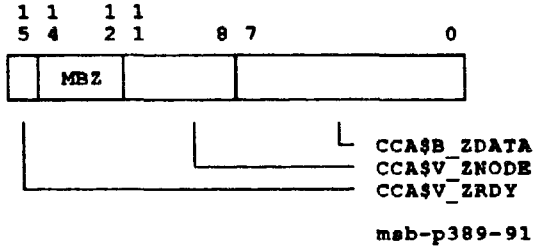


Table 2–20 (Cont.) Buffer Fields

Field	Description	
	CCA$V_ZALT	When set, the target of the current "Z" command cannot communicate through the CCA. The target is either a non-processor XMI node or a VAXBI node and must be accessed using alternate RXCD protocol, as described in the *VAXBI System Reference Manual*.
	CCA$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_ZDEST	When set, this node is sending "Z" command data to the node listed in CCA$B_ZDEST.
	CCA$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$Q_READY of the CCA header.
CCA$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$V_ZRDY	When this bit is set, there is valid data in the other CCA$W_ZRXCD fields.
	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$B_ZDATA.
	CCA$B_ZDATA	When CCA$V_ZRDY is set, this field contains one byte of "Z" command data being sent to this node.
CCA$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$T_RX.	
CCA$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$T_TX	This buffer is used by the node to transmit a response to the BP. Only response data is passed through this buffer since a nonboot processor does not send commands to the boot processor.	
CCA$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.	

2.10.3 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 go to 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 go to 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.

2.11 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 2–21 lists the internally generated system control block (SCB) entry points. The complete list of supported SCB vectors is in Section 2.3.5. Table 2–22 describes the levels of hardware-detected errors by level of severity. Table 2–23 lists the categories of errors, organized by entry point.

Refer to these sections:

- Section 2.3.5 for an explanation of the SCB.
- Section 2.3.4 for an explanation of exceptions and interrupts.

Table 2–21 MP-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

[1]This section describes the entry-point vector in detail.

Table 2-21 (Cont.) MP-Chip Internally Generated SCB Entry Points

Mnemonic	SCB Index (hex)	Description
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 2–22 Hardware-Detected Errors

Error	Description
Console halt	A halt to console mode is caused by one of the errors listed in Table 2–8. 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 MP-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 MP-chip's execution of instructions. Instruction-level recovery and retry may be possible.

Table 2–23 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
		MAXMI-to-MP-chip parity error
		ERR L signal asserted
		MC-chip tag store parity errors
		XBER-notified errors
		MSSC DAL timeout
		DAL error on memory write or flush write buffers
		ERR L signal asserted
		MC-chip tag/VD parity error
		XBER-notified errors (backup cache off)
		MSSC DAL timeout
		Vector module errors
08	Kernel stack not valid	
0C	Power fail	
54	Soft error interrupt	P-cache errors
		MAXMI-detected errors
		Vector module errors
60	Hard error interrupt	MC-chip tag store parity error
		Inval-Bus parity errors
		Address/command bus parity errors
		Cache fill errors
		XBER-notified errors
		XBEER-notified errors
		Vector module errors

2.11.1 General Error Detection and Reporting Characteristics

The following sections describe the general operation of error handling and recovery not associated with specific errors.

2.11.1.1 MAXMI Error Handling

The MAXMI 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, NSCSR, and XFADR. If retry is enabled, the normal state, the MAXMI 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 a hard error interrupt at the IPL 1D (hex), or as a soft error interrupt at IPL 1A (hex).

Software uses the MAXMI error summary bits XBER<ES> (general XMI errors) and XBER<NSES> (MAXMI node-specific errors) to determine if the MAXMI 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 MAXMI 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 MP-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 MP-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 MP-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 nondemand data, such as cache fill data, result in 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 MAXMI generates and checks XMI parity. Both the MDA-chip and the MCA-chip check parity across the XCI D<63:0> lines.

The MAXMI 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 MAXMI participates, such as read data, the MAXMI either posts a hard error interrupt or asserts ERR L, 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.

The MAXMI generates parity on read data that it drives on the DAL bus. The MAXMI detects DAL parity errors using one parity bit for each data byte across the 64 bits. Table 2–24 shows the MAXMI parity coverage.

Table 2–24 MAXMI Parity Coverage

Side of MAXMI	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			
		MAXMI passive			Soft error interrupt
		MAXMI active Write			Hard error interrupt
		MAXMI active Read			ERR L asserted
DAL	8 total	1 per data byte	Yes	Yes	Hard error interrupt

The MAXMI generates ECC on all cache fill and cache write operations. Six bits of ECC are generated per byte. The MAXMI checks ECC on all writeback operations. Single-bit errors are corrected and reported as soft errors. Multiple-bit errors result in a Tag Bad Data XMI transaction to mark the memory location as bad and are reported as hard errors. The MAXMI generates bad ECC by flipping the six ECC bits if a DAL parity error is detected on a memory space write. This error is reported as a hard error when the cache subblock is written back to memory.

Assertion of the ERR L signal causes a 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 MP-chip and the MP-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 MF-chip reports all errors through protocol signals on the bus between the MP-chip and the MF-chip. Certain severe protocol errors between the MP-chip and the MF-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 MF-chip is present.

2.11.1.2 Parity Generation and Detection

Parity generation and check characteristics of the KA65A CPU module are as follows:

- The MP-chip generates parity on write data and checks parity on memory transaction read data. The MP-chip does not generate parity on command/address information.
- An MP-chip I-stream parity error is reported as an interrupt with the appropriate bits set in the P-cache status register. This error may also result in the MCHK_ERROR_ISTREAM machine check.
- The MP-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 MP-chip during reads.
- The MAXMI detects DAL parity errors on memory writes when the backup cache is off.
- The MAXMI generates and checks the three XMI parity bits, which cover both data and command information.
- The MF-chip generates parity for MF-chip results and checks parity on D_BUS floating operands.
- The MSSC does not support parity so the internal battery-backed-up one Kbyte of RAM is not protected, nor are any of the MSSC internal registers.
- The MAXMI CSR registers are not parity protected.
- The MC-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.

2.11.1.3 Microcode-Detected Errors

Errors detected by microcode checks that result in console halts and machine checks are listed in Table 2–25. Microcode checks also detect the kernel stack not valid exception.

Table 2–25 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 MP-chip
Machine Checks		
01	MCHK_FP_PROTOCOL_ERROR	Protocol error during MF-chip operand/result transfer
02	MCHK_FP_ILLEGAL_OPCODE	Illegal opcode detected by MF-chip
03	MCHK_FP_OPERAND_PARITY	Operand parity error detected by MF-chip
04	MCHK_FP_UNKNOWN_STATUS	Unknown status returned by MF-chip
05	MCHK_FP_RESULT_PARITY	Returned MF-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 MP-chip's instruction fetch and decode unit)

Table 2–25 (Cont.) Microcode-Detected Errors

Code (hex)	Mnemonic	Description
0D	MCHK_UNKOWN_CS_ADDR	Undefined control store address reached
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 on write or clear write buffer
13	MCHK_UNKNOWN_BUSERR_ TRAP	Undefined bus error microtrap
14	MCHK_VECTOR_STATUS	Vector module error
15	MCHK_ERROR_ISTREAM	Error on I-stream read

2.11.1.4 Self-Test-Detected Errors

There are two levels of self-test errors: power-up MP-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 BAD L signal
- XBER<STF> not clearing
- The module's Self-Test Passed (STP) LED not turned on

2.11.2 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

2.11.2.1 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:

- Bus Error Register (XBER)
- XMI Failing Address Register (XFADR)

- XMI Failing Address Extension Register (XFAER)
- Bus Error Extension Register (XBEER)
- Writeback 0 Failing Address Register (WFADR0)
- Writeback 1 Failing Address Register (WFADR1)
- Failing DAL Register 0 (FDAL0)
- Failing DAL Register 1 (FDAL1)
- Failing DAL Register 2 (FDAL2)
- Failing DAL Register 3 (FDAL3)
- MSSC Bus Timeout Control Register (SSCBTR)
- Backup Cache Control Register (BCCTL)
- Backup Cache Status Register (BCSTS)
- Backup Cache Error Address Register (BCERA)
- Backup Cache Error Tag Register (BCERT)
- Primary Cache Status Register (PCSTS)
- Primary Cache Error Address Register (PCERR)
- Vector Interface Error Status Register (VINTSR)

Pseudocode 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 XBER would be saved in the variable s_xber.

2.11.2.2 Error Analysis

The error condition is analyzed with the error state obtained during the collection process. See Section 2.11.4, Section 2.11.6, and Section 2.11.7 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 the primary cache disabled and the backup cache in error transistion mode (ETM) to minimize the possibility of nested errors.

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.

2.11.2.3 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 the primary cache disabled and the backup cache in ETM:

  PCSTS := $\overline{\text{ENABLE\_BTS}}$ + $\overline{\text{FORCE\_HIT}}$;

  BCCTL := ENABLE_BTS + BTS_ERROR_TRAN + $\overline{\text{FORCE\_BHIT}}$

  The primary cache must be disabled when enabling the backup cache to maintain cache coherency. Once the backup cache is enabled, the primary cache is flushed and enabled.

- Error recovery is performed starting with the most distant component and then working toward the CPU. MAXMI errors are to be processed first, followed by MSSC errors, MC-chip errors, and P-cache errors.

- MAXMI errors are cleared by writing the read/write-one-to-clear (R/W1C) bits in XBER and XBEER. The suggested procedure is to write the values saved during error state collection back to registers:

  XBER := s_xber

  XBEER := s_xbeer

- MSSC 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;

- MC-chip internal parity errors are recovered by reconstructing the tag and valid/dirty bits for dirty subblocks from the following system state:

  Backup Cache Status Register (BCSTS)

  Backup Cache Error Address Register (BCERA)

  Backup Cache Error Tag Register (BCERT)

  Memory controller block state (using memory block state registers BSCTL and BSADR)

  Any of the four cache subblocks indexed by the failing tag could be dirty and, therefore, contain the only "good" copy of that data. These subblocks must be identified, the tag and valid/dirty bits rewritten into the tag store, and then written back to memory. If the failing tag

contained no dirty subblocks, the tag is rewritten with good parity and all valid/dirty bits cleared.

Error bits BCSTS<BTS TPERR> and BCSTS<BTS VDPERR> indicate if the tag and/or the valid/dirty bits have parity errors. If only the valid/dirty bit contained the parity error, the error address can be obtained by concatenating the following fields:

```
test_addr<31> := 0;

test_addr<30:19> := s_bcert<31:19>;

test_addr<18:7> := s_bcera<18:7>;

test_addr<6:0> := 0;
```

The correct tag can then be created as shown below:

```
new_tag<30:9> := s_bcert<30:9>;

new_tag<8:0> := 0;

For i := 0 to 3 DO
    BEGIN
    IF memory_state(test_addr) = (OWNED AND (ID = KA65A
    CPU Module)) THEN
        BEGIN
        new_tag<i> := 1;
        new_tag<i+4> := 1;
        END;
    test_addr<6:5> := test_addr<6:5> + 1;
    END;

new_tag<8> := odd_parity(new_tag<7:0>)
```

The tag is now written back into the tag array and the block written back to memory.

```
BCIDX := s_bcerr;

BCBTS := new_tag;

BCDET := 0;
```

If the tag had bad parity and all dirty/valid bits are clear, the tag may be rewritten with good parity:

```
BCIDX := s_bcerr;

BCBTS := %x00000300;
```

If the tag had bad parity and one or more dirty bits are set, or the valid/dirty bits also contain a parity error, a number of memory locations must be searched to re-create the correct address. Thirteen possible addresses (12 tag bits, 1 parity bit) are created using the search method shown below:

```
test_addr<31> := 0;
```

```
test_addr<30:19> := s_bcert<30:19>;
test_addr<18:7> := s_bcera<18:7>;
test_addr<6:0> := 0;
```

The correct tag is now created:

```
new_tag<31:0> := 0;
tag_found := false;
For j := 0 to 3 DO
    BEGIN
    IF memory_state(test_addr) = (OWNED AND (ID = KA65A
    CPU Module)) THEN
        BEGIN
        tag_found := true;
        new_tag<j> := 1;
        new_tag<j+4> := 1;
        END;
    test_addr<6:5> := test_addr<6:5> + 1;
    END;
i := 19
 While (i ≤ 30) AND ~tag_found~ DO
    BEGIN
    test_addr<i> := ~test_addr<i>~
    For j := 0 to 3 DO
        BEGIN
        If memory_state(test_addr) = (OWNED AND (ID = KA65A
        CPU Module)) THEN
            BEGIN
            tag_found := true;
            new_tag<i> := 1;
            new_tag<i+4> := 1;
            END;
        test_addr<6:5> := test_addr<6:5> + 1;
        END;
    IF ~tag_found~ Then test_addr<i> := ~test_addr<i>~;
    END; (While)
 new_tag<8> := odd_parity(new_tag<7:0>)
```

If an owned block was found during the memory search, the created tag is now written back into the tag array and the block written back to memory. If an owned block was not found, it is assumed that the cache block was not dirty and the tag is cleared as shown below:

```
IF tag_found THEN
    BEGIN
    BCIDX := s_bcerr;
    BCBTS := new_tag;
    BCDET := 0;
    END
ELSE
    BEGIN
    BCIDX := s_bcerr;
    BCBTS := %x00000300;
    END;
```

MC-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 := %x80000000;
        END;
```

P-cache errors are cleared as part of the reenabling of the cache process, which requires the following sequence:

```
BCSTS := s_bcsts; For i := 0 to 4095 DO
    BEGIN
    BCIDX := 1*128;
    BCDET := 0;
    BCBTS := %x00000300;
    END;
BCCTL := ENABLE_BTS + FORCE_BHIT + BTS_ERROR_TRAN
PCSTS := s_pcsts OR
(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 as shown above.

System performance degrades if one or both caches are disabled.

2.11.2.4 Vector Error Recovery

Nested machine checks during the analysis and recovery of vector errors must be prevented. Bits in VINSTR determine the error analysis and recovery procedure as follows:

1 If VINTSR<Vector Absent> (VINTSR<0>) is set, the vector processor module is not connected to the VIB cable. Immediately discontinue using the vector module.

2 If VINTSR<VECTL VIB Hard Error>, <C-Chip VIB Hard Error>, or <Bus Timeout> (VINTSR<4>, <6>, or <7>, respectively) is set, an unrecoverable VIB error was detected and no other registers should be read.

3 If VINTSR<Vector Hard Error> (VINTSR<2>) is set, an unrecoverable error was detected by the vector module. All VINTSR error bits must be cleared to determine the error source. Do this by writing the values saved during the error state collection back to the register:

```
VINTSR := s_vintsr;
```

VECTL CSR must be read to determine the vector module error source. This register can be written to the VA longword of the stack frame for vector machine checks. Depending on the error conditions indicated by VECTL CSR, read additional vector module registers. See Chapter 3 for more information on vector module errors.

4 If an unrecoverable vector module error occurred, the vector module must be reset by setting VINTSR<Vector Module Reset> (VINTSR<8>) for at least 100 CPU cycles and then clearing it. Image exit must be forced for the process that was executing in the vector module at the time of the error.

All error bits in VINTSR must be cleared, using the above technique, if the vector module is to be used after reset. Perform a self-test before returning the vector module to service. Software must not continue using a vector module that fails self-test or has reported more than one unrecoverable error.

To discontinue using the vector module, the VIB must be disabled by setting VINTSR<Disable Vector Interface> (VINTSR<9>) and execution of vector instructions must be disabled by clearing ACCS<Vector Present> (ACCS<0>).

5 If VINTSR<Vector Soft Error>, <VECTL VIB Soft Error>, <MC-chip VIB Soft Error> (VINTSR<1>, <3>, or <5>, respectively) is set, a recoverable error has occurred. Error state must be cleared in the vector module and VINTSR error bits must be cleared, using the above technique, to recover from these errors.

Software can count the number of reported vector module soft errors and, if the count exceeds a threshold, save process state and discontinue using the vector module.

2.11.2.5 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.

2.11.3 Console Halt and Halt Interrupt

A console halt is not an exception but is a transfer of control by the MP-chip microcode directly into console program macrocode in the boot ROM at address E004 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 MSSC.

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 SAVPSL<13:8>.

Table 2–8 lists and describes the console halt codes.

2.11.4 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 (XBEER<UWP>)
- State of the Backup Cache Second Error bit (BCSTS<SECOND ERR>)
- 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 externally detected hardware error, such as a memory parity error.

A machine check is technically an aborted macro instruction. The MP-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 2–8, and its parameters are described in Table 2–6. These parameters are parsed by the error handling macrocode to determine what caused the machine check.

Figure 2–31 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 2–31 Machine Check Parse Tree

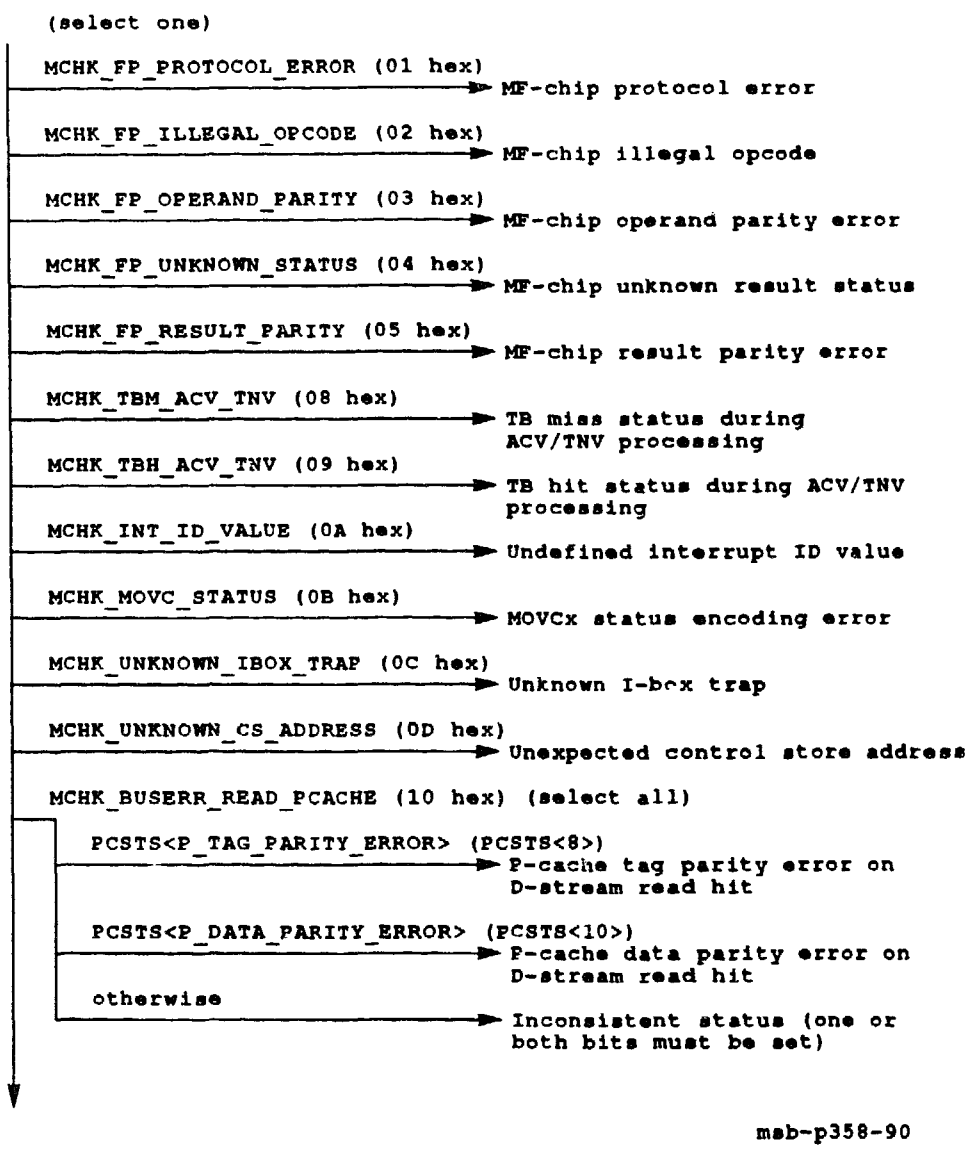


Figure 2–31 Cont'd on next page

Figure 2–31 (Cont.) Machine Check Parse Tree

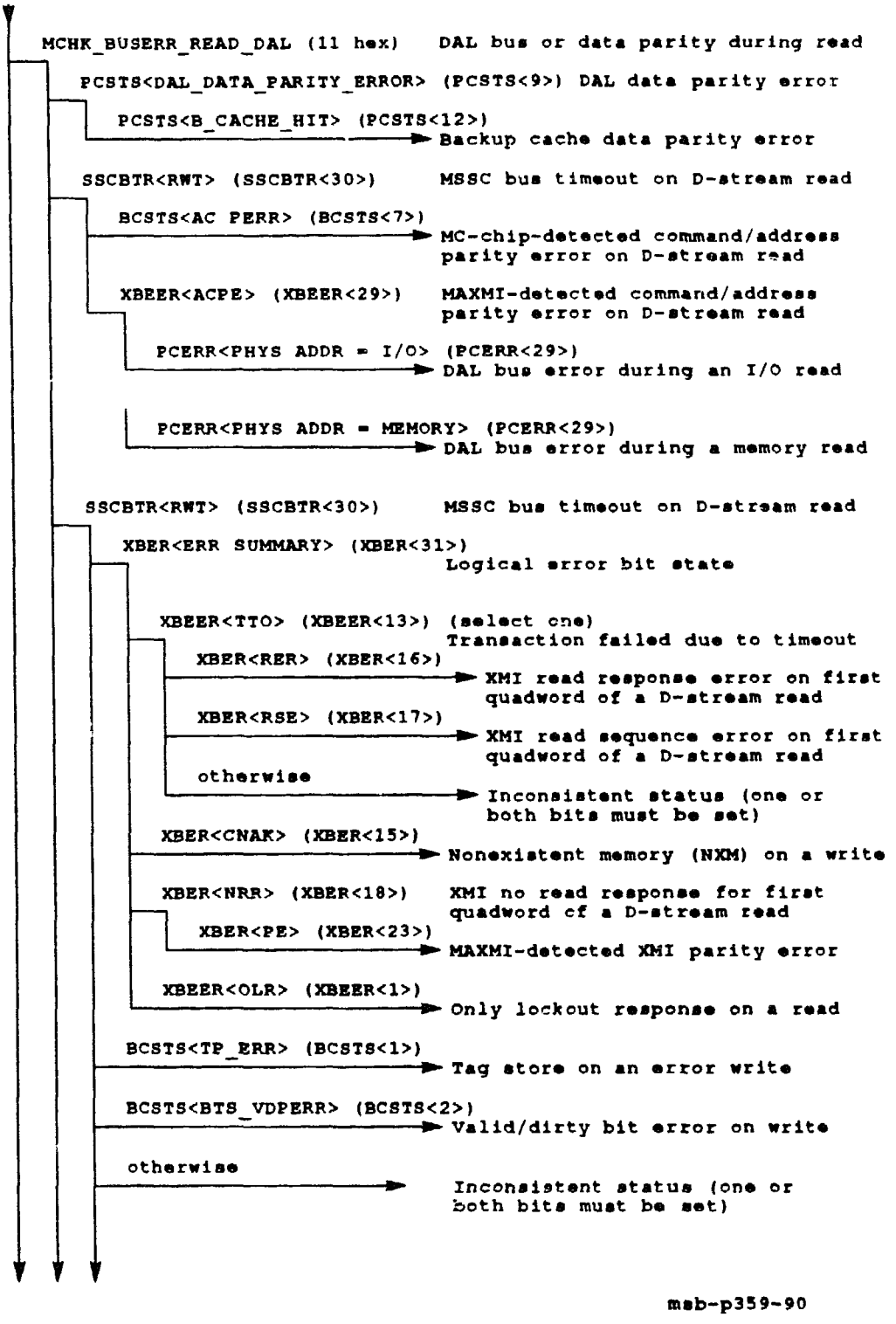


Figure 2–31 Cont'd on next page

Figure 2-31 (Cont.) Machine Check Parse Tree

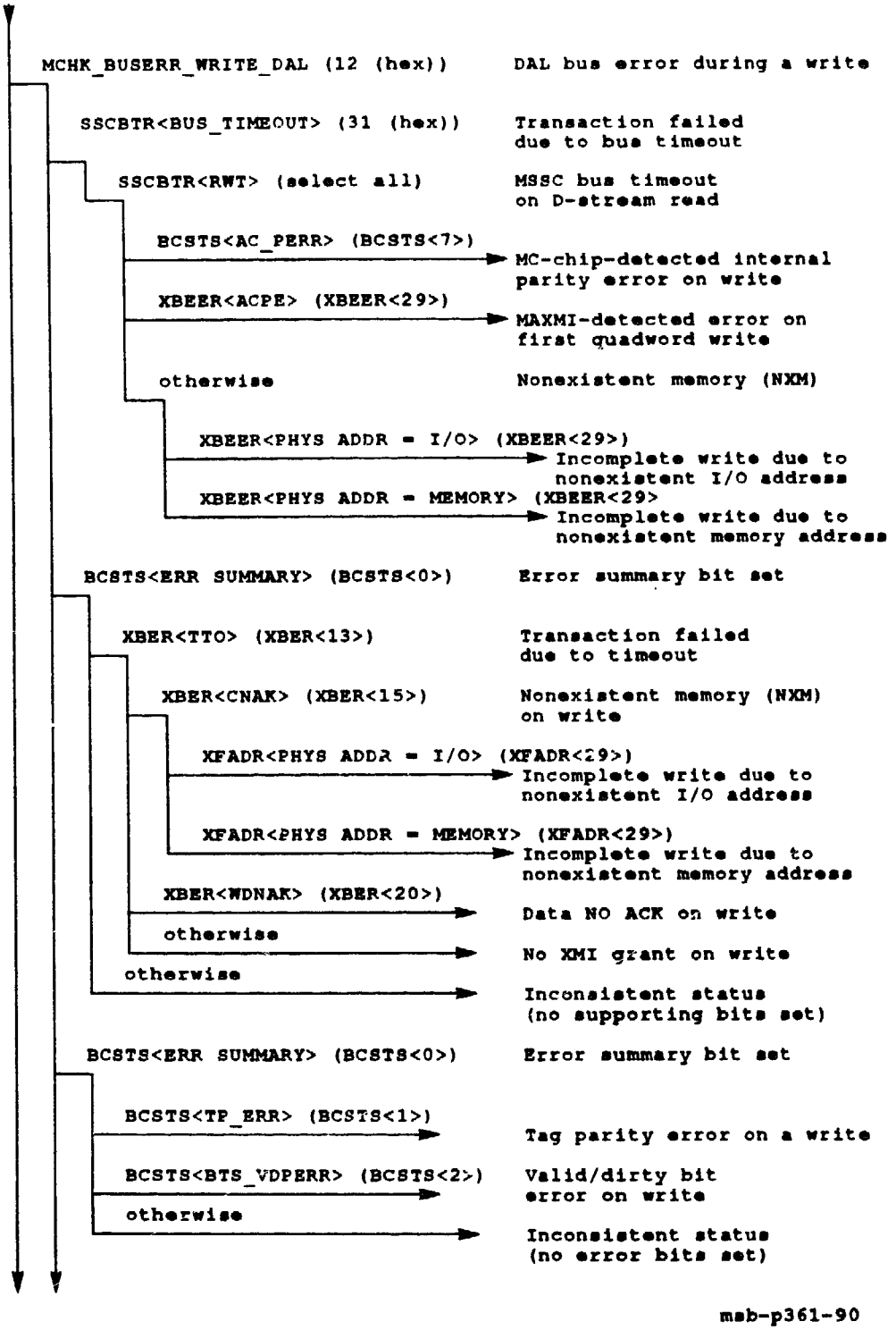


Figure 2-31 Cont'd on next page

Figure 2–31 (Cont.) Machine Check Parse Tree

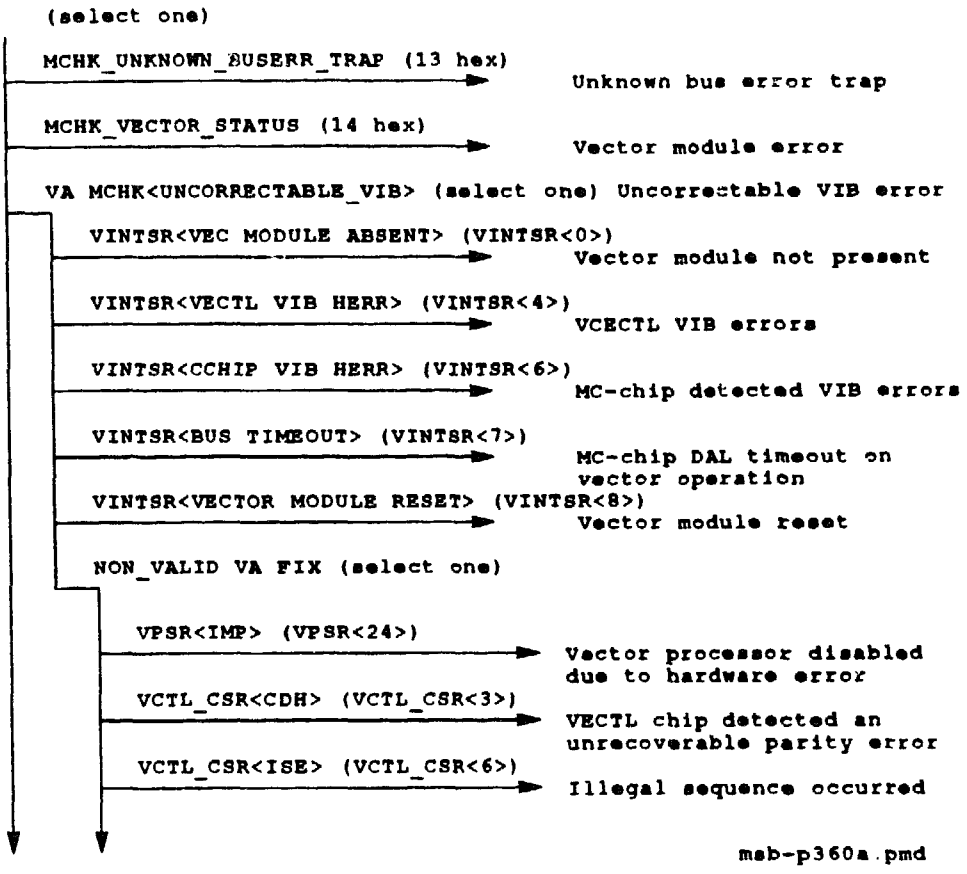


Figure 2–31 Cont'd on next page

Figure 2–31 (Cont.) Machine Check Parse Tree

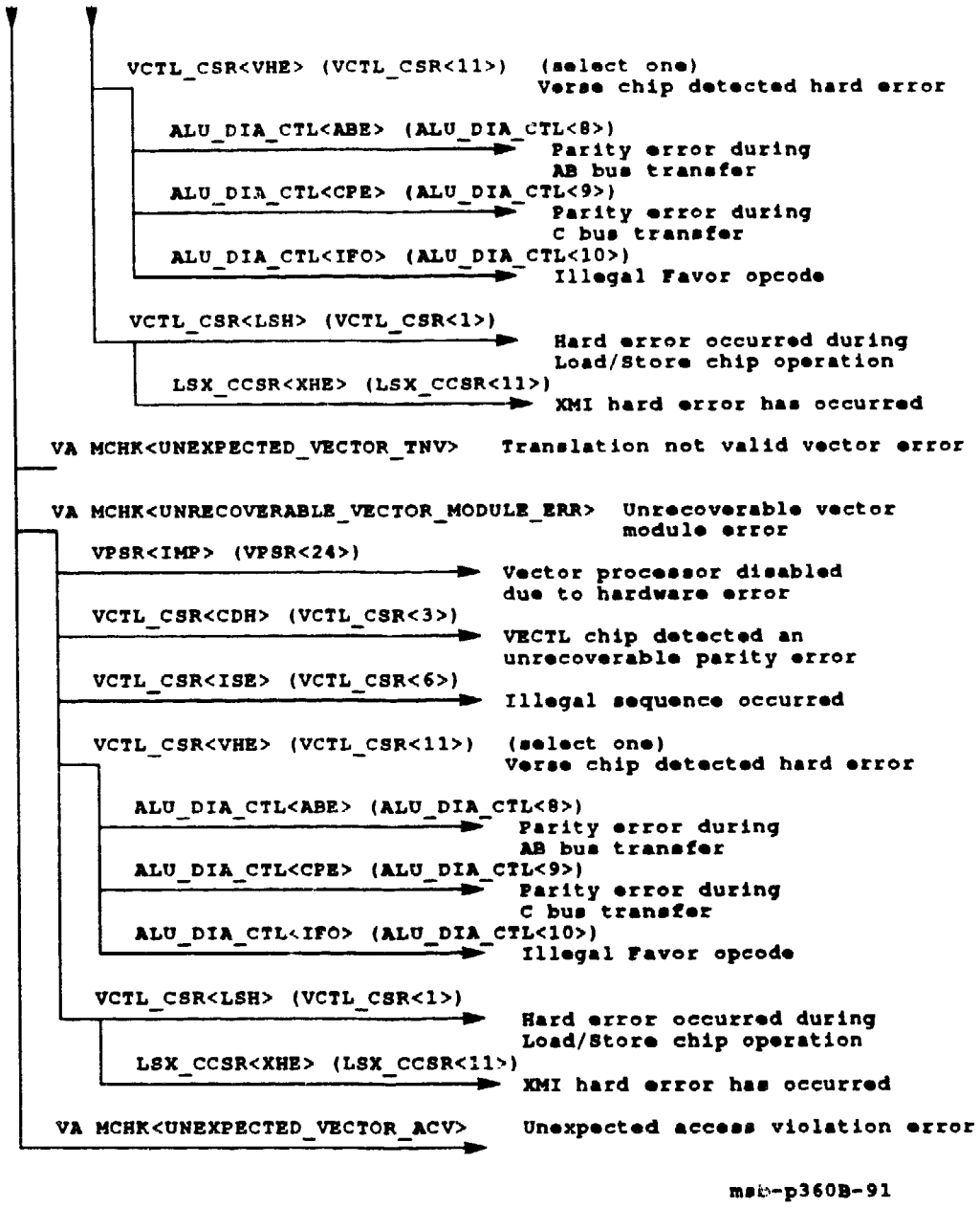
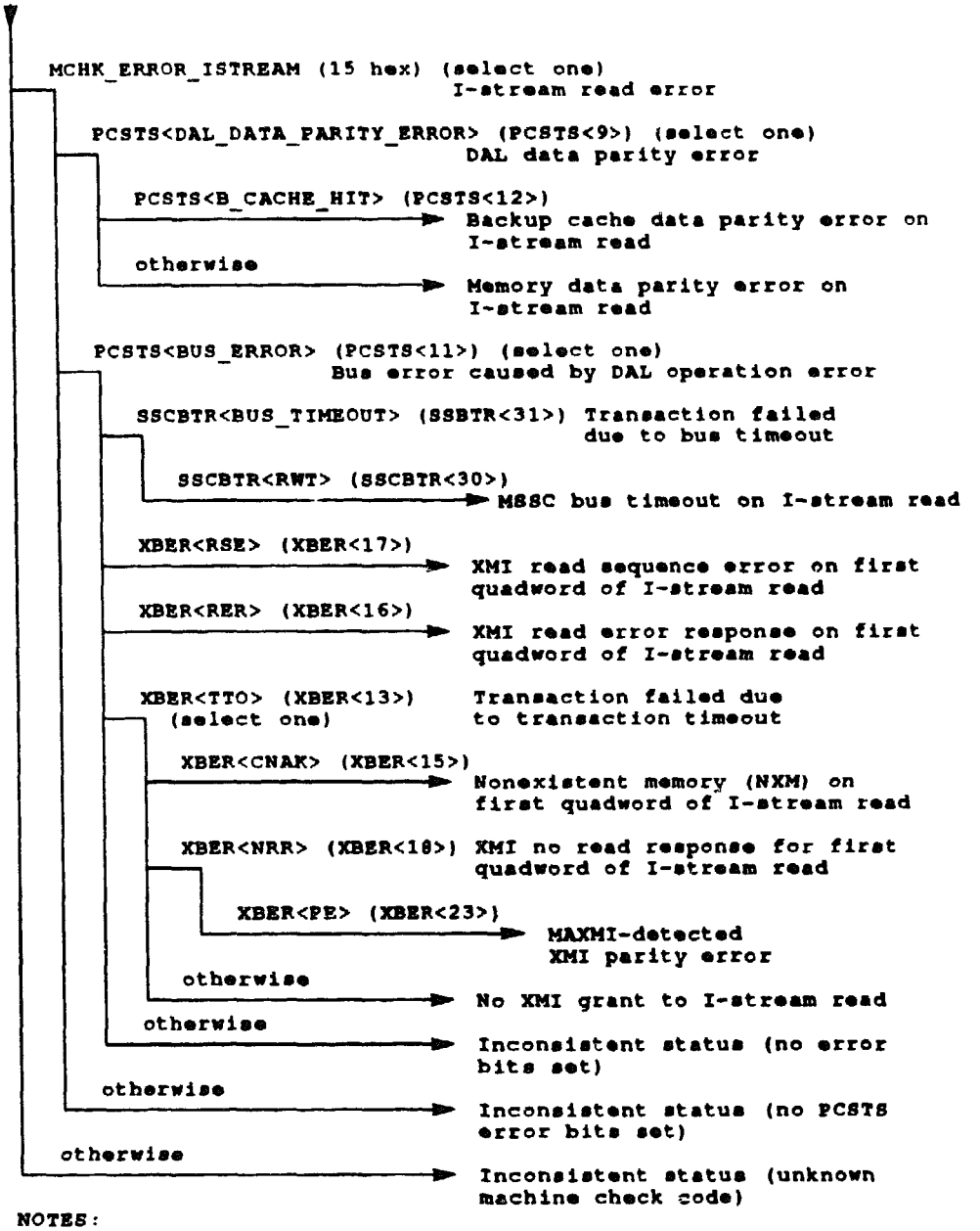


Figure 2–31 Cont'd on next page

Figure 2–31 (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.

msb-p362-90

2.11.4.1 MCHK_FP_PROTOCOL_ERROR

Description: A protocol error was detected by either the MP-chip or the MF-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	MP-chip
11	xxx	MP-chip
10	000	MF-chip

The error is probably due to a bit flipped on either the CPSTA or CPDAT signal lines between the MP-chip and the MF-chip during an opcode, operand, or result transfer.

Recovery procedure: No explicit error recovery is required. If the error reoccurs, disable the MF-chip by writing a zero to ACCS<MF-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an MF-chip instruction, The restart bit (R) in the stack frame should always be one, PSL<FPD> should always be zero, and XBEER<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (XBEER<UWP>=0)

2.11.4.2 MCHK_FP_ILLEGAL_OPCODE

Description: An illegal opcode was detected by the MF-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 MF-chip.

Recovery procedure: No explicit error recovery is required. If the error reoccurs, disable the MF-chip by writing a zero to ACCS<MF-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an MF-chip instruction, R should always be one, PSL<FPD> should always be zero, and XBEER<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (XBEER<UWP>=0)

2.11.4.3 MCHK_FP_OPERAND_PARITY

Description: A parity error was detected by the MF-chip during an operand transfer and reported during a result return. Backup cache or memory parity errors are also detected by the MC-chip, resulting instead in a MCHK_BUSERR_READ_DAL machine check. This machine check indicates that the parity error was detected only by the MF-chip, implying that the MP-chip generated bad parity, or that the MP-chip and the MF-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 an MP-chip parity checking error, an MF-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 MF-chip operand is read from an I/O space location. MP-chip parity checking is disabled for I/O space reads but the MF-chip checks parity for all operands. Also see Section 2.3.8.

Recovery procedure: No explicit error recovery is required. If the error reoccurs, disable the MF-chip by writing a zero to ACCS<MF-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an MF-chip instruction, R should always be one, PSL<FPD> should always be zero, and XBEER<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (XBEER<UWP>=0)

2.11.4.4 MCHK_FP_UNKNOWN_STATUS

Description: An unassigned status code was returned by the MF-chip. This is caused when CPSTA=10 and CPDAT<2:0>=111 appear with the returned result (from MF-chip to MP-chip). This is probably due to a bit flipped on the CPSTA or CPDAT lines during result transfer to the MP-chip.

Recovery procedure: No explicit error recovery is required. If the error reoccurs, disable the MF-chip by writing a zero to ACCS<1>.

Restart condition: Because this error is detected during the execution flow of an MF-chip instruction, R should always be one, PSL<FPD> should always be zero, and XBEER<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (XBEER<UWP>=0)

2.11.4.5 MCHK_FP_RESULT_PARITY

Description: A result data parity error was detected by the MP-chip during MF-chip result transfer. This is probably due to a bit flipped on the D-bus or parity lines.

Recovery procedure: No explicit error recovery is required. If the error reoccurs, disable the MF-chip by writing a zero to ACCS<MF-Chip Present>.

Restart condition: Because this error is detected during the execution flow of an MF-chip instruction, R should always be one, PSL<FPD> should always be zero, and XBEER<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (XBEER<UWP>=0)

2.11.4.6 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 procedure: 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 (XBEER<UWP>=0)

2.11.4.7 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 procedure: 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 (XBEER<UWP>=0)

2.11.4.8 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 procedure: 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 (XBEER<UWP>=0)

2.11.4.9 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 procedure: 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 (XBEER<UWP>=0)

2.11.4.10 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 procedure: 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 XBEER<UWP> should always be zero. Retry if:

(R=1) AND (PSL<FPD>=0) AND (XBEER<UWP>=0)

2.11.4.11 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 procedure: No explicit error recovery is required.

Restart condition: Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0)

2.11.4.12 MCKH_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.

2.11.4.12.1 PCSTS<P_TAG_PARITY_ERROR>

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 procedure: Write all P-cache tags with good parity and cleared valid bits, and perform the full memory error recovery procedures in Section 2.11.2.3. If the error reoccurs, disable the P-cache.

Restart condition: Retry if

((R=1) OR (PSL<FPD>=0)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.12.2 PCSTS<P_DATA_PARITY_ERROR>

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 procedure: Perform the full memory error recovery procedures in Section 2.11.2.3. If the error reoccurs, disable the P-cache.

Restart condition: Retry if

((R=1) OR (PSL<FPD>=0)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.13 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.

2.11.4.13.1 PCSTS<DAL_DATA_PARITY_ERROR>

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.

2.11.4.13.2 PCSTS<B_CACHE_HIT>

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 procedure: Perform the full memory error recovery procedures in Section 2.11.2.3. If the error reoccurs, disable the backup cache.

Restart condition: Retry if

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.13.3 SSCBTR<RWT>

Description: The MSSC timed out on a D-stream read. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, and PCSTS<TRAP1> should all be set. PCERR contains the physical address of the error. This error is due to either a reference to a reserved address or an address parity error. On an address parity error, BCSTS<AC PERR> and/or XBEER<ACPE> will be set.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures in Section 2.11.2.3.

Restart condition: This error may cause other XMI nodes to experience XMI timeouts because writeback and invalidates cannot be performed by the MC-chip while it is waiting for the MSSC to time out a DAL transaction. Retry may or may not be desirable. Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.13.3.1 BCSTS<AC PERR>

Description: The MC-chip detected an internal parity error on a D-stream read. PCSTS<BUS ERROR>, PCSTS<TRAP1>, BCSTS<ERR SUMMARY>, and either BCSTS<BTS TPERR> and/or BCSTS<BTS VDPERR> should be set. BCSTS<DAL CMD> must be a read, BCSTS<DMG L> and BCSTS<SYNC L> must be set, BCSTS<OREAD PENDING> must be clear, and BCERA must match PCERR. Otherwise, the status is inconsistent and the error should not be retried. BCERA and PCERR contain the physical address of the error.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the error recovery procedure in Section 2.11.2.3.

Restart condition: Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.13.3.2 XBEER<ACPE>

Description: The MAXMI and/or MC-chip detected an address/command parity error on a DAL command. XBEER<ACPE> and/or BCSTS<AC PERR> should be set. Probably due to a bit flipped on the A bus or command lines. When either the MAXMI or the MP-chip detect an address /command parity error, they ignore the transaction. This error can also be reported as an MSSC bus timeout. The source of the error is determined by the state of these bits:

PCERR<PHYS ADDR = I/O>	A DAL problem exists while attempting to do a read operation. The MP-chip requested an I/O location from the XMI bus. A valid I/O location indicates a CPU module or node problem. An invalid I/O location indicates a CPU module or software problem.
PCERR<PHYS ADDR = MEMORY>	A DAL problem exists while attempting to do a read operation. The MP-chip requested a memory location from the XMI bus. A valid I/O location indicates a CPU module or memory problem. An invalid I/O location indicates a CPU module or software problem.

Recovery procedure: Clear all error bits in XBEER and BCSTS.

Restart condition: Retry is not possible.

2.11.4.13.4 SSCBTR<RWT>

Description: The MSSC timed out on a D-stream read. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, and PCSTS<TRAP1> should all be set. PCERR contains the physical address of the error. This error is due to either a reference to a reserved address or an address parity error. On an address parity error, BCSTS<AC PERR> and/or XBEER<ACPE> will be set.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures in Section 2.11.2.3.

Restart condition: This error may cause other XMI nodes to experience XMI timeouts because writeback and invalidates cannot be performed by the MC-chip while it is waiting for the MSSC to time out a DAL transaction. Retry may or may not be desirable. Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.13.4.1 XBER<ERR SUMMARY>

Description: The state of this bit represents the logical OR of the error bits in this register. This bit is asserted if any error bit is asserted and cleared when all error bits are cleared.

The source of the error is determined by the state of these bits:

XBER<TTO> — The first quadword of read data was not returned before the MAXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set in the normal case. If neither bit is set, the MAXMI was never granted the XMI for the read command. PCSTS<BUS ERROR>, PCSTS<TRAP1>, BCSTS<ERR SUMMARY>, and BCSTS<Fill Abort> 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<ERR SUMMARY>, and BCSTS<Fill Abort> should all be set.

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<ERR SUMMARY>, and BCSTS<Fill Abort> 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>, PCSTS<BUS ERROR>, PCSTS<TRAP1>, BCSTS<ERR SUMMARY>, 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 MAXMI 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<ERR SUMMARY>, and BCSTS<Fill Abort> should all be set.

XBER<PE> The MAXMI detected a parity error on an XMI cycle. The command sent was acknowledged, but data was never returned to the CPU. XBEER<MDAXPE> or XBEER<MCAXPE> and XBER>BITMAP>(ES) should be set. If appropriate, XBER>BITMAP>(IPE) is also set.

XBEER<OLR> Lockout responses were received while attempting to do a read-type transaction. This is due to faliure of the lockout mechanism to provide access to an interlock or the ownership of a memory location. If OLR is set, XBER<TTO> is also set.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3.

Restart condition: Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

unless stated otherwise above.

2.11.4.13.4.2 BCSTS<TP_ERR>

Description: The backup (secondary) cache chip detected a tag parity error. This bit sets when a parity error occurs in the tag entry as a result of the last access of the tag store. This bit sets only when BCCTL<ENABLE BTS> is set and BCCTL<FORCE BHIT> is not set.

Recovery procedure: Clear BCSTS<TP_ERR> and BCCTL<ENABLE BTS>.

Restart condition: Do not attempt retry.

2.11.4.13.4.3 BCSTS<BTS_VDPERR>

Description: The backup (secondary) cache chip detected a valid/dirty parity error. This bit sets when a parity error occurs in the valid/dirty part of the tag entry as a result of the last access of the tag store. This bit sets only when BCCTL<ENABLE BTS> is set and BCCTL<FORCE BHIT> is not set.

Recovery procedure: Clear BCSTS<TP_ERR> and BCCTL<ENABLE BTS>.

Restart condition: Do not attempt retry.

2.11.4.14 MCHK_BUSERR_WRITE_DAL

Description: A DAL write or clear write buffer transaction was terminated with the ERR L signal asserted. The MSSC timed out a write or clear write buffer transaction. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, and PCSTS<TRAP1>, should all be set. This error is due to either a reference to a reserved address or an address parity error. On an address parity error, BCSTS<AC PERR> and/or XBEER<ACPE> will be set.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures described in Section 2.11.2.3.

Restart condition: Do not attempt retry. The error is reported after the instruction that issued the write is completed. Since no address is available, the error is considered to be systemwide.

2.11.4.14.1 SSCBTR<BUS_TIMEOUT>

Description: A CPU transaction timed out and terminated with the ERR L signal asserted. If a CPU read, write, or clear write buffer command is timed out, this bit and SSCBTR<RWT> are set. For IPR reads and writes, only this bit sets.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures in Section 2.11.2.3.

Restart condition: This error may cause other XMI nodes to experience XMI timeouts because writeback and invalidates cannot be performed by the MC-chip while it is waiting for the MSSC to time out a DAL transaction. Retry may or may not be desirable. Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.14.1.1 SSCBTR<RWT>

Description: The MSSC timed out on a D-stream read. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, and PCSTS<TRAP1> should all be set. PCERR contains the physical address of the error. This error is due to either a reference to a reserved address or an address parity error. On an address parity error, BCSTS<AC PERR> and/or XBEER<ACPE> will be set.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures in Section 2.11.2.3.

Restart condition: This error may cause other XMI nodes to experience XMI timeouts because writeback and invalidates cannot be performed by the MC-chip while it is waiting for the MSSC to time out a DAL transaction. Retry may or may not be desirable. Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.14.1.1.1BCSTS<AC PERR>

Description: The MC-chip detected a command/address parity error on a DAL command. Either XBEER<ACPE> and/or BCSTS<AC PERR> will be set. Probably due to a flipped bit on the A bus or CMD lines. The MC-chip will ignore the transaction when this error occurs. This error may also be reported as an MSSC bus timeout.

Recovery procedure: Clear all error bits in XBEER and BCSTS.

Restart condition: No retry is possible because this error could indicate a failed write.

2.11.4.14.1.1.2XBEER<ACPE>

Description: The MAXMI and/or MC-chip detected an address/command parity error on a DAL command. XBEER<ACPE> and/or BCSTS<AC PERR> should be set. Probably due to a bit flipped on the A bus or command lines. When either the MAXMI or the MP-chip detect an address /command parity error, they ignore the transaction. This error can also be reported as an MSSC bus timeout. The source of the error is determined by the state of these bits:

XBEER<PHYS ADDR = I/O>	A CPU write command did not complete its operation. Since there were no internal bus timeouts, a nonexistent address was accessed. A valid I/O location indicates a node problem. An invalid I/O location indicates a CPU address or software problem.

XBEER<PHYS ADDR = MEMORY>	A CPU write command did not complete its operation. Since there were no internal bus timeouts, a nonexistent address was accessed. A valid memory location indicates a node problem. An invalid memory location indicates a CPU address or software problem.

Recovery procedure: Clear all error bits in XBEER and BCSTS.

Restart condition: Retry is not possible.

2.11.4.14.2 BCSTS<ERR_SUMMARY>

Description: ERR SUMMARY sets whenever any of the following sets:

- BCSTS BTS TPERR (bit <1>)
- BCSTS BTS VDPERR (bit <2>)
- BCSTS I PERR <1:0> (bits <5:4>)
- BCSTS FILL ABORT (bit <6>)
- BCSTS AC PERR (bit <7>)
- BCSTS SECOND ERR (bit <8>)

When ERR SUMMARY sets, the MC-chip is forced into ETM. ERR SUMMARY clears when all the error bits are cleared.

Recovery procedure: Clear all error bits to clear BCSTS<ERR SUMMARY>.

Restart condition: Retry is not possible.

2.11.4.15 Other MAXMI-Detected Errors

Description: The MAXMI detected an error during an XMI transaction. The state of these bits determine the various cases:

XBER<TTO>	The first quadword of read data was not returned before the MAXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set in the normal case. If neither bit is set, the MAXMI was never granted the XMI for the read command. PCSTS<BUS ERROR>, PCSTS<TRAP1>, BCSTS<ERR SUMMARY>, and BCSTS<Fill Abort> 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>, PCSTS<BUS ERROR>, PCSTS<TRAP1>, BCSTS<ERR SUMMARY>, 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.

XFADR<PHYS ADDR = I/O>	A CPU write command did not complete its operation. Since there were no internal bus timeouts, a nonexistent address was accessed. A valid I/O location indicates a node problem. An invalid I/O location indicates a CPU address or software problem.
XFADR<PHYS ADDR = MEMORY>	A CPU write command did not complete its operation. Since there were no internal bus timeouts, a nonexistent address was accessed. A valid memory location indicates a node problem. An invalid memory location indicates a CPU address or software problem.
XBER<WDNAK>	The write data cycle was NO ACKed by the XMI and all retries failed. XBER<TTO> should also be set.

Recovery procedure: Clear all error bits in XBER.

2.11.4.15.1 BCSTS<ERR_SUMMARY>

Description: ERR SUMMARY sets whenever any of the following sets:

- BCSTS BTS TPERR (bit <1>)
- BCSTS BTS VDPERR (bit <2>)
- BCSTS I PERR <1:0> (bits <5:4>)
- BCSTS FILL ABORT (bit <6>)
- BCSTS AC PERR (bit <7>)
- BCSTS SECOND ERR (bit <8>)

When ERR SUMMARY sets, the MC-chip is forced into ETM. ERR SUMMARY clears when all the error bits are cleared.

Recovery procedure: Clear all error bits to clear BCSTS<ERR SUMMARY>.

Restart condition: Retry is not possible.

2.11.4.15.1.1 BCSTS<TP_ERR>

Description: The backup (secondary) cache chip detected a tag parity error. This bit sets when a parity error occurs in the tag entry as a result of the last access of the tag store. This bit sets only when BCCTL<ENABLE BTS> is set and BCCTL<FORCE BHIT> is not set.

Recovery procedure: Clear BCSTS<TP_ERR> and BCCTL<ENABLE BTS>.

Restart condition: Do not attempt retry.

2.11.4.15.1.2 BCSTS<BTS_VDPERR>

Description: The backup (secondary) cache chip detected a valid/dirty parity error. This bit sets when a parity error occurs in the valid/dirty part of the tag entry as a result of the last access of the tag store. This bit sets only when BCCTL<ENABLE BTS> is set and BCCTL<FORCE BHIT> is not set.

Recovery procedure: Clear BCSTS<TP_ERR> and BCCTL<ENABLE BTS>.

Restart condition: Do not attempt retry.

2.11.4.16 MCHK_UNKNOWN_BUSERR_TRAP

Description: The bus interface unit (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 procedure: No explicit error recovery is required.

Restart condition: Retry should not be attempted because this error may be masking a write error.

2.11.4.17 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 MC-chip and vector processor related errors. See also Section 3.9.1.

Recovery procedure: In general, a MCHK_VECTOR_STATUS machine check is not recoverable.

Restart condition: Retry should not be attempted. Reset and initialize the vector processor module.

2.11.4.17.1 VA MCHK<UNCORRECTABLE_VIB>

Description: The CPU detected an uncorrectable VIB error and flagged it executing machine checks. The state of this error is determined by the errors listed below.

2.11.4.17.1.1 VINTSR<VEC MODULE ABSENT>

Description: The MC-chip has detected that the vector module is not installed properly or is not present.

Recovery procedure: This error is caused by one of three components: the vector module, the CPU module, or the vector cable. Check the vector cable first to be sure that it is connected properly. Then examine the CPU module and/or vector module for defects.

Restart condition: Retry can be attempted if the recovery procedure is successful.

2.11.4.17.1.2 VINTSR<VECTL VIB HERR>

Description: The VECTL detected an unrecoverable command or write data parity error during a VIB transaction. VINTSR<VECTL VIB HERR> should be set.

Recovery procedure: Perform the full vector error recovery described in Section 2.11.2.3.

Restart condition: Since hardware retry already failed, communication with the vector module is no longer possible and no retry should be attempted. An image exit should be forced on the process executing in the vector module at the time of the error.

2.11.4.18 VINTSR<CCHIP VIB HERR>

Description: The MC-chip detected an unrecoverable data parity error during a VIB read transaction. VINTSR<CCHIP VIB HERR> should be set.

Recovery procedure: Perform the full vector error recovery described in Section 2.11.2.3.

Restart condition: Since hardware retry already failed, communication with the vector module is no longer possible and no retry should be attempted. An image exit should be forced on the process executing in the vector module at the time of the error.

2.11.4.19 VINTSR<BUS TIMEOUT>

Description: The MC-chip detected the assertion of the ERR L signal while holding a vector EPR read transaction on the DAL. This error may be due to an insufficient timeout value in the MSSC timeout register. VINTSR<Bus Timeout> and SSCBTR<BTO> should both be set.

Recovery procedure: Perform the full vector error recovery described in Section 2.11.2.3. Then clear SSCBTR<BTO>.

Restart condition: Since a DAL vector EPR transaction was terminated before the corresponding VIB transaction completed, the VIB is in an inconsistent state and no retry should be attempted. An image exit should be forced on the process executing in the vector module at the time of the error.

2.11.4.19.1 VINTSR<VECTOR MODULE RESET>

Description: The MC-chip has detected that an external processor read operation on the Vector Interface Bus (VIB) was in progress while the vector module was reset.

Recovery procedure: This problem can be caused by the vector not initializing properly, the reset function failing, or the software not waiting long enough between cycles to attempt vector transactions. The first two problems can be fixed by replacing the CPU or vector modules. The software problem requires additional software support.

Restart condition: Retry can be attempted if the recovery procedure is successful.

2.11.4.19.2 NON_VALID VA FIX

Description: The CPU detected vector errors and flagged them by executing machine checks. The state of these bits determine the various cases:

VPSR<IMP>	The vector processor is disabled due to a hardware error.
VCTL_CSR<CDH>	The VECTL chip found an unrecoverable parity error on the VIB. The error is not recovered by retrying the transaction.
VCTL_CSR<ISE>	An illegal sequence occurred. The following sequences set this bit: Illegal instruction issue sequence Illegal VIB command Asserting Load/Store exception after MMOK
VCTL_CSR<VHE>	The Verse chip detected a hard error condition and reported it to the VECTL chip.
ALU_DIA_CTL<ABE>	A parity error was detected during an AB bus transfer.
ALU_DIA_CTL<CPE>	A parity error was detected during a C bus transfer.
ALU_DIA_CTL<IFO>	An illegal Favor opcode was transmitted.
VCTL_CSR<LSH>	A hard error 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.
LSX_CCSR<XHE>	An XMI hard error has occurred.

Recovery procedure: Clear all error bits to clear the errors.

Restart condition: Retry is not possible.

2.11.4.19.3 VA MCHK<UNEXPECTED_VECTOR_TNV>

Description: The MC-chip detected an unexpected translation not valid (TNV) vector error. This is a programming error.

Recovery procedure: Clear all error bits to clear this error.

Restart condition: Retry is possible.

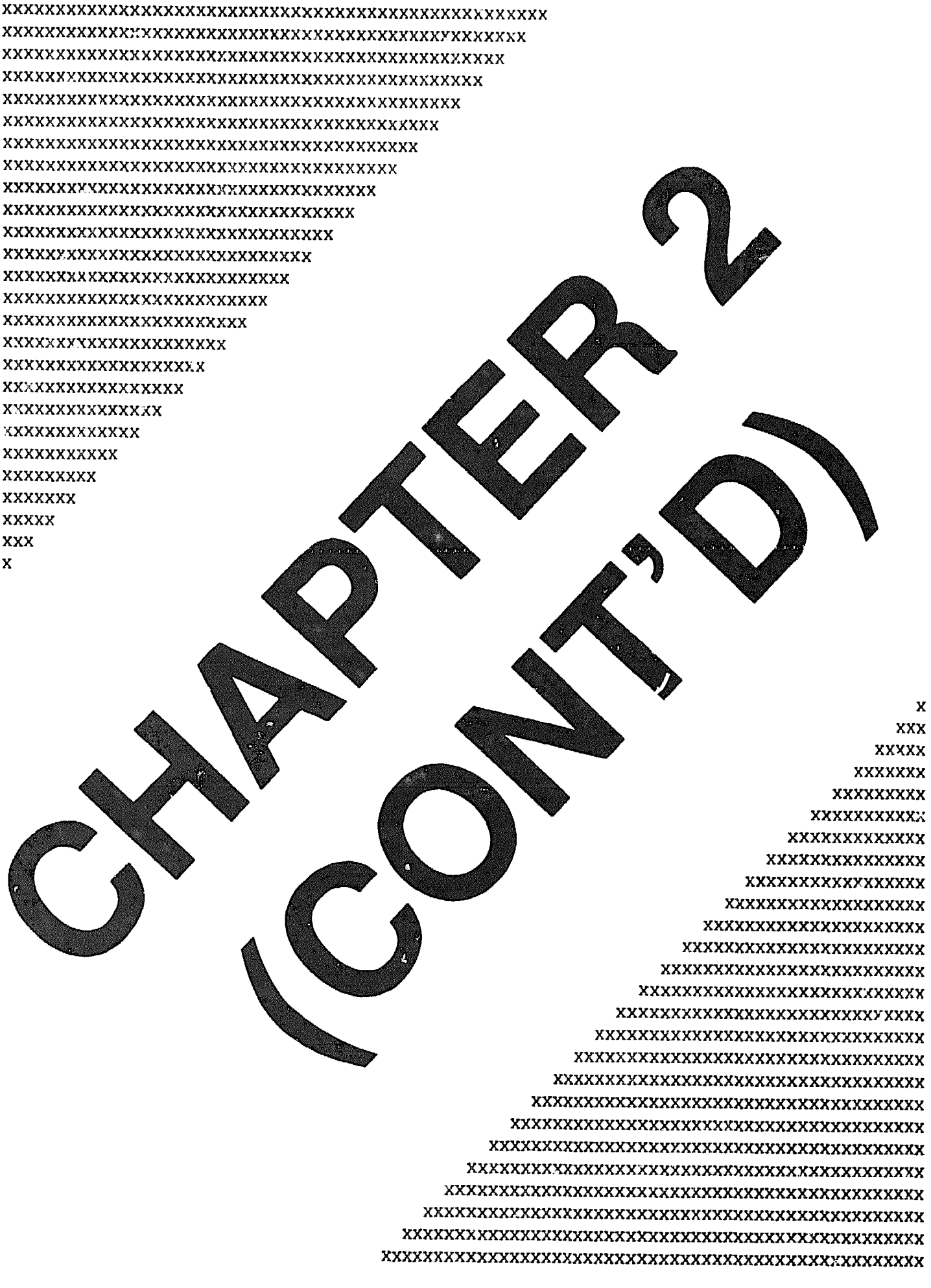
2.11.4.19.4 VA MCHK<UNRECOVERABLE_VECTOR_MODULE_ERR>

Description: The vector module detected an unrecoverable internal error. VINTSR<VECTOR HERR> should be set. The VECTL CSR is returned in bits<31:9> and bits<6:0> of the VA longword in the machine check stack frame and indicates the source of the error. Other vector module registers may contain additional information. Refer to Chapter 3 for a list of possible errors. The state of these bits determine the various cases:

VPSR<IMP>	The vector processor is disabled due to a hardware error.
VCTL_CSR<CDH>	The The VECTL chip found an unrecoverable parity error on the VIB. The error is not recovered by retrying the transaction.
VCTL_CSR<ISE>	An illegal sequence occurred. The following sequences set this bit: Illegal instruction issue sequence Illegal VIB command Asserting Load/Store exception after MMOK
VCTL_CSR<VHE>	The Verse chip detected a hard error condition and reported it to the VECTL chip.
ALU_DIA_CTL<ABE>	A parity error was detected during an AB bus transfer.
ALU_DIA_CTL<CPE>	A parity error was detected during a C bus transfer.
ALU_DIA_CTL<IFO>	An illegal Favor opcode was transmitted.
VCTL_CSR<LSH>	A hard error 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.
LSX_CCSR<XHE>	An XMI hard error has occurred.

Recovery procedure: Clear all error bits to clear the errors.

Restart condition: Retry is not possible.


CHAPTER 2
(CONT'D)

2.11.4.19.5 VA MCHK<UNEXPECTED_VECTOR_ACV>

Description: The MC-chip detected an unexpected access violation (ACV) vector error. This is a programming error.

Recovery procedure: Clear all error bits to clear this error.

Restart condition: Retry is possible.

2.11.4.20 MCHK_ERROR_ISTREAM

Description: The MP-chip has detected an I-stream read error. This error is generated internally in the CPU module.

Recovery procedure: No explicit error recovery is required. Replacing the CPU module should fix the problem.

Restart condition: Retry is possible.

2.11.4.20.1 PCSTS<DAL_DATA_PARITY_ERROR>

A data parity error was detected during an I-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.

2.11.4.20.1.1 PCSTS<B_CACHE_HIT>

Description: A data parity error was detected during an I-stream read hit in the backup cache. PCSTS<TRAP1>, PCSTS<DAL DATA PARITY ERROR>, and PCSTS<B CACHE HIT> should all be set.

Recovery procedure: Perform the full memory error recovery procedures in Section 2.11.2.3. If the error reoccurs, disable the backup cache.

Restart condition: Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.20.1.2 Memory Data Parity Error on I-Stream Read

Description: A memory data parity error was detected during an I-stream read. An actual memory parity error would be reported as a bus error, described next. This error implies that the parity went bad between the MAXMI and the MP-chip. PCSTS<TRAP1> and PCSTS<DAL DATA PARITY ERROR> should both be set. PCSTS<B CACHE HIT> should be cleared.

Recovery procedure: Perform the full memory error recovery procedures in Section 2.11.2.3.

Restart condition: Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.20.2 PCSTS<BUS_ERROR>

A bus error was detected during an I-stream read. The I-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 a non-I/O space address for BCSTS to log the error.

2.11.4.20.2.1 SSCBTR<BUS_TIMEOUT>

Description: A CPU transaction timed out and terminated with the ERR L signal asserted. If a CPU read, write, or clear write buffer command is timed out, this bit and SSCBTR<RWT> are set. For IPR reads and writes, only this bit sets.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures in Section 2.11.2.3.

Restart condition: This error may cause other XMI nodes to experience XMI timeouts because writeback and invalidates cannot be performed by the MC-chip while it is waiting for the MSSC to time out a DAL transaction. Retry may or may not be desirable. Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

2.11.4.20.2.2 SSCBTR<RWT>

Description: The MSSC timed out on an I-stream read. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, and PCSTS<TRAP1> should all be set. PCERR contains the physical address of the error. This error is due to either a reference to a reserved address or an address parity error. On an address parity error, BCSTS<AC PERR> and/or XBEER<ACPE> will be set.

Recovery procedure: Clear SCCBTR<RWT> and SSCBRT<BTO>. Then perform the full memory error recovery procedures in Section 2.11.2.3.

Restart condition: This error may cause other XMI nodes to experience XMI timeouts because writeback and invalidates cannot be performed by the MC-chip while it is waiting for the MSSC to time out a DAL transaction. Retry may or may not be desirable. Retry if:

((R=1) OR (PSL<FPD>=1)) AND (XBEER<UWP>=0) AND (PCSTS<TRAP2>=0)

The source of the error is determined by the state of these bits:

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<ERR SUMMARY>, and BCSTS<Fill Abort> 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<ERR SUMMARY>, and BCSTS<Fill Abort> should all be set.

XBER<TTO> The first quadword of read data was not returned before the MAXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set in the normal case. If neither bit is set, the MAXMI was never granted the XMI for the read command. PCSTS<BUS ERROR>, PCSTS<TRAP1>, BCSTS<ERR SUMMARY>, and BCSTS<Fill Abort> 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>, PCSTS<BUS ERROR>, PCSTS<TRAP1>, BCSTS<ERR SUMMARY>, 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 MAXMI 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<ERR SUMMARY>, and BCSTS<Fill Abort> should all be set.

XBER<PE> The MAXMI detected a parity error on an XMI cycle. The command sent was acknowledged, but data was never returned to the CPU. XBEER<MDAXPE> or XBEER<MCAXPE> and XBER>BITMAP>(ES) should be set. If appropriate, XBER>BITMAP>(IPE) is also set.

2.11.5 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 2–2.

The VAX 6000 Model 500 supports the standard XMI time of 4 milliseconds to execute the software necessary to save processor state for systems without a battery backup unit; software has 500 milliseconds to save processor state in systems with battery backup.

Software must flush the cache to memory in the power fail service routine since backup cache state is not saved across a power fail. This routine is:

```
BCCTL := ENABLE_BTS + BTS_ERROR_TRAN + FORCE_BHIT

FOR i := 0 to 4095 DO
    BEGIN
    BCIDX := i * 128;
    BCDET := 0;
    BCBTS := %x00000300;
    END;
```

In a system without a battery backup unit, software may not have time to flush the entire cache to memory.

2.11.6 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 2–2.

Figure 2–32 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, the procedure to recover, and the conditions for restarting the operation.

Figure 2–32 Hard Error Interrupt Parse Tree

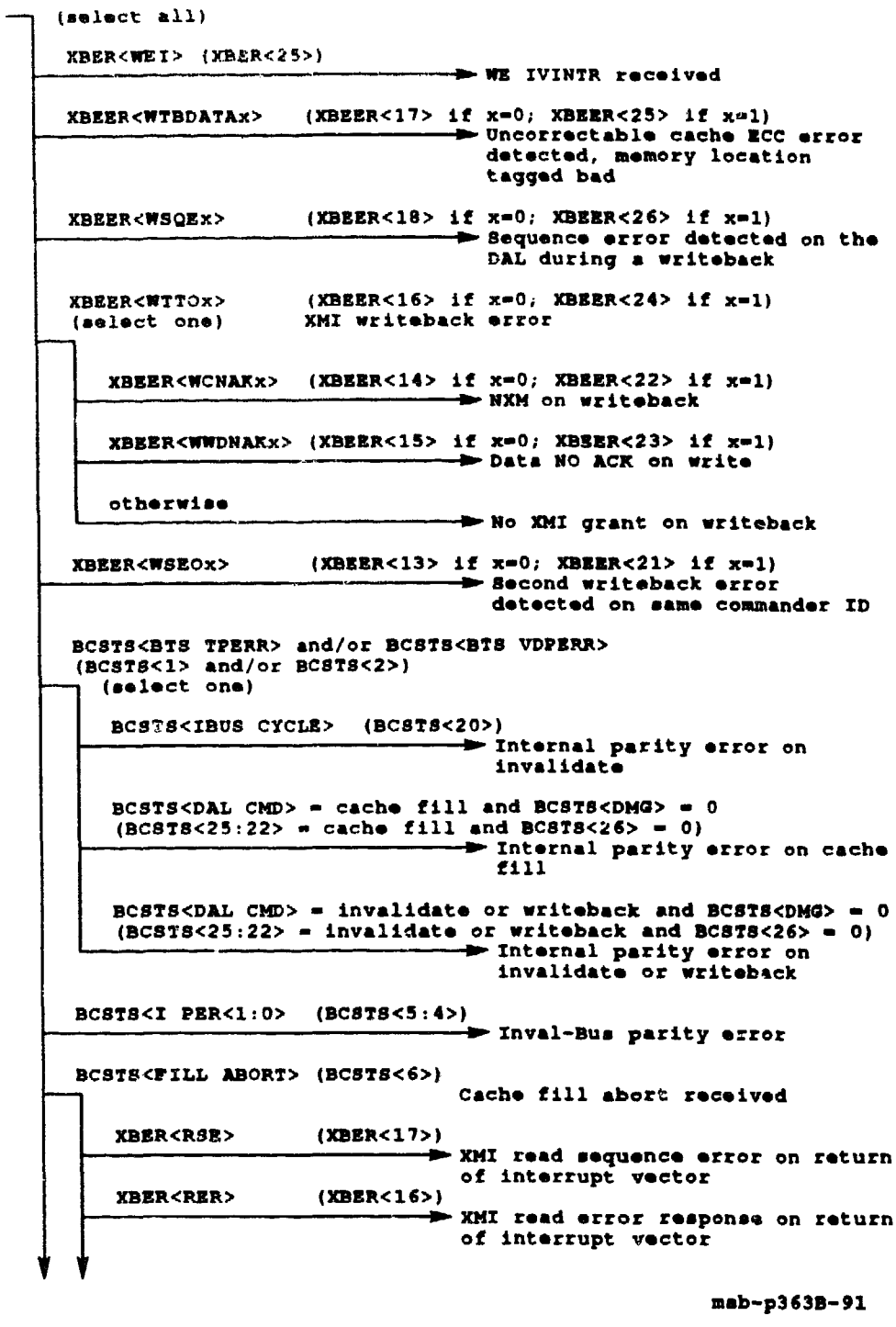


Figure 2–32 Cont'd on next page

Figure 2–32 (Cont.) Hard Error Interrupt Parse Tree

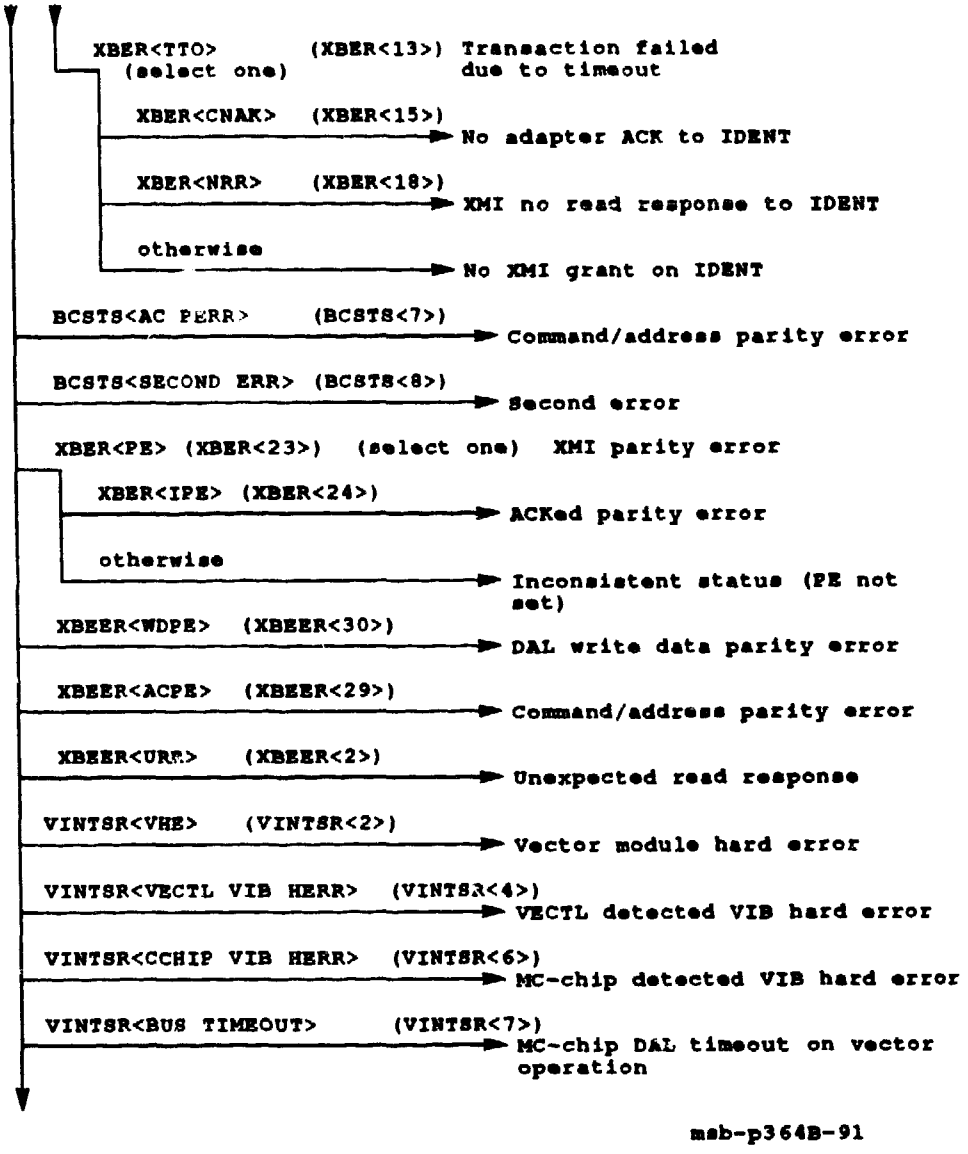
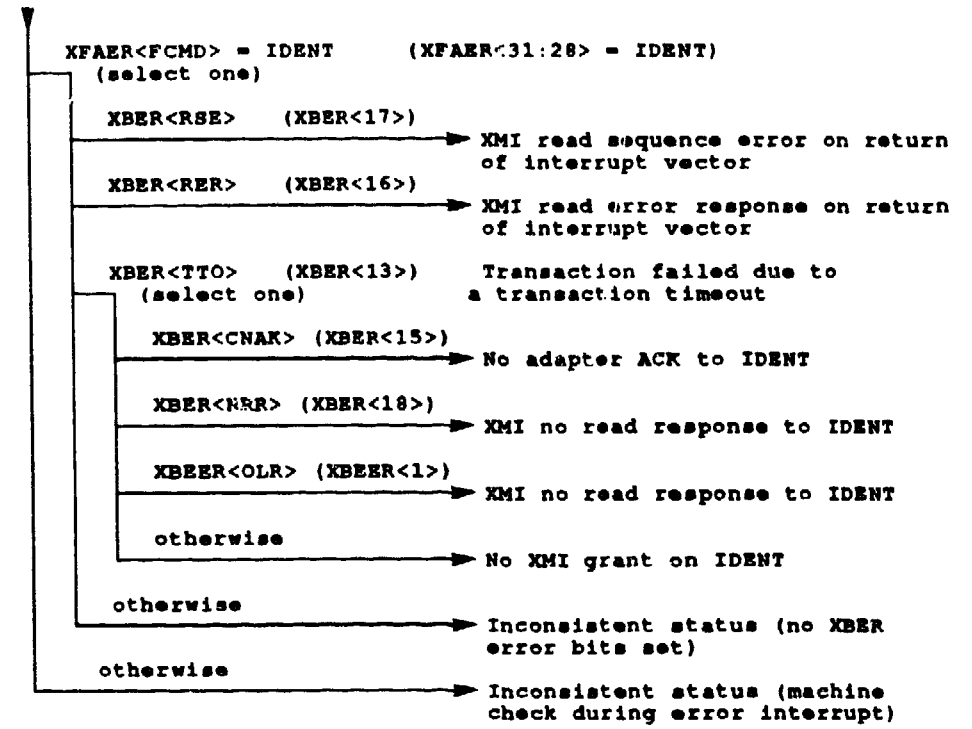


Figure 2–32 Cont'd on next page

Figure 2–32 (Cont.) Hard Error Interrupt 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.

msb-p365B-91

2.11.6.1 XBER<WEI>

Description: An XMI write error IVINTR command was received by the MAXMI. XBER<WEI> indicates this condition. WEIs are generated by other nodes, so the interrupt source (for example, the DWMBB) must be examined to determine if recovery is possible.

Recovery procedure: Clear all error bits in XBER.

Restart condition: Retry is not possible.

2.11.6.2 XBEER<WTBDATAx> and XBEER<WSQEx>

Description: The MAXMI detected an error on the DAL during a writeback. The source of the error is distinguished by bits in XBEER, as shown:

XBEER<WTBDATA*x*>	An uncorrectable (multiple-bit) ECC error was detected by the MAXMI on a writeback transaction. This error is due to a cache RAM failure, an interconnect failure, or a parity error on an MP-chip write. Parity errors on writes can be distinguished from other errors since the MAXMI inverts all six ECC bits for any byte with a parity error on a write and allows the bad parity to be written into the cache.
XBEER<WSQE*x*>	The second, third, or fourth quadword of a writeback on the DAL did not complete successfully due to a command/ address parity error.

XBEER<WFDQ0> and XBEER<WFDQ1> indicate where the data, parity, and ECC for this error are stored in the FDAL*n* registers. WFADR*n* contains the physical address of the write that caused the error. The MAXMI issues an XMI Tag Bad Data transaction to memory to mark this memory location bad.

Recovery procedure: Clear all error bits in XBEER.

Restart condition: No retry is possible because this error indicates a failed write. Software must determine if the error is limited to a process or if the entire system is affected.

2.11.6.3 XBEER<WTTOx>

Description: The MAXMI detected an XMI error during an XMI Disown Write transaction. The source of the error is distinguished by bits in XBEER, as follows:

XBEER<WTTO*x*>	A Disown Write was not completed on the XMI before the MAXMI timeout expired. XBEER<WCNAK*x*> or XBEER<WWDNAK*x*> should also be set. If neither bit is set, the MAXMI was never granted the XMI for the Disown Write.
XBEER<WCNAK*x*>	The Disown Write command was NO ACKed on the XMI and retry failed. This does not indicate an NXM since the block must have been read from memory in order for the MC-chip to start a writeback. XBEER<TTO> should also be set.
XBEER<WWDNAK*x*>	The write data cycle was NO ACKed by the XMI and all retries failed. XBEER<TTO> should also be set.

WFADR*n* contains the physical address of the write that caused the error.

Recovery procedure: Clear all error bits in XBEER.

Restart condition: No retry is possible because this error indicates a failed write. Software must determine if the error is limited to a process or if the entire system is affected.

2.11.6.4 XBEER<WSEOx>

Description: The MAXMI detected a second writeback error before the first error was serviced. XBEER<WSEO*n*> will be set.

Recovery procedure: Clear all bits in XBEER.

Restart condition: This error indicates that multiple writes have failed, and retry is not possible.

2.11.6.5 BCSTS<BTS TPERR>

Description: The backup (secondary) cache chip detected a tag parity error. This bit sets when a parity error occurs in the tag entry as a result of the last access of the tag store. This bit sets only when BCCTL<ENABLE BTS> is set and BCCTL<FORCE BHIT> is not set.

Recovery procedure: Clear BCSTS<TP_ERR> and BCCTL<ENABLE BTS>.

Restart condition: Do not attempt retry.

2.11.6.6 BCSTS<BTS VDPERR>

Description: The backup (secondary) cache chip detected a valid/dirty parity error. This bit sets when a parity error occurs in the valid/dirty part of the tag entry as a result of the last access of the tag store. This bit sets only when BCCTL<ENABLE BTS> is set and BCCTL<FORCE BHIT> is not set.

Recovery procedure: Clear BCSTS<TP_ERR> and BCCTL<ENABLE BTS>.

Restart condition: Do not attempt retry.

2.11.6.6.1 BCSTS<IBUS CYCLE>

Description: The MC-chip detected an internal parity error on an invalidate lookup. BCSTS<BTS TPERR> and/or BCSTS<BTS VDPERR> will be set. BCSTS<IBUS CYCLE> distinguishes this error from other internal parity errors. BCERA contains the physical address of the lookup, and BCERT contains the failing tag entry.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3. If the error reoccurs, disable the backup cache.

Restart condition: Other XMI nodes may experience XMI timeouts if a writeback is delayed because of this error.

2.11.6.6.2 BCSTS<DAL CMD> = Cache Fill

Description: The MC-chip detected an internal parity error while attempting to set the valid and/or dirty bit at the end of a cache fill sequence. BCSTS<BTS TPERR> and/or BCSTS<BTS VDPERR> will be set. BCSTS<DAL CMD> = cache fill and BCSTS<DMG> = 0 distinguish this error from other internal parity errors. BCERA contains the physical address of the fill, and BCERT contains the failing tag entry. Correct data is always loaded into the cache during this error.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3. If the error reoccurs, disable the backup cache.

Restart condition: Other XMI nodes may experience XMI timeouts if a writeback is delayed because of this error.

2.11.6.6.3 BCSTS<DAL CMD> = Invalidate or Writeback

Description: The MC-chip detected an internal parity error while attempting to clear the valid and/or dirty bit at the end of a cache invalidate or writeback sequence. BCSTS<BTS TPERR> and/or BCSTS<BTS VDPERR> will be set. BCSTS<DAL CMD> = write and BCSTS<DMG> = 0 distinguish this error from other internal parity errors. This invalidate or writeback will complete and the valid and/or dirty bit will be cleared in the tag array. BCERA contains the physical address of the invalidate or writeback, and BCERT contains the failing tag entry. The valid, dirty, and VD parity bits in BCERT are unpredictable during this error.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3. If the error reoccurs, disable the backup cache.

Restart condition: Other XMI nodes may experience XMI timeouts if a writeback is delayed because of this error.

2.11.6.7 BCSTS<I PER<1:0>

Description: The MC-chip detected an Inval-Bus parity error on an invalidate request transaction. BCSTS<IBUS CYCLE> and BCSTS<I PER<0> and/or BCSTS<I PER<1> will be set. BCSTS<IBUS CMD> contains the value of the invalidate command line received on the Inval-Bus. BCERA contains the physical address that the MC-chip received on the Inval-Bus.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3. If the error reoccurs, disable the backup cache.

Restart condition: Other XMI nodes may experience XMI timeouts if a writeback is delayed because of this error.

2.11.6.8 BCSTS<FILL ABORT>

Description: The MC-chip received a cache fill abort command from the MAXMI, indicating that an error occurred on the second, third, or fourth quadword of a read transaction or an error occurred on any quadword of an Ownership Read for a write transaction. The source of the error is distinguished by bits in XBER, as shown:

XBER<RSE>	A quadword of read data was returned with the wrong sequence number. Probably due to a parity error on the returned data, in which case XBER<PE> will also be set.
XBER<RER>	A quadword of read data was returned with an RER, indicating that the memory got a double-bit error reading the array. Probably not recoverable, but it is possible.
XBER<TTO>	The first quadword of read data was not returned before the MAXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set in the normal case. If neither bit is set, the MAXMI was never granted the XMI for the read command.
XBER<CNAK>	The read data command was NO ACKed by the XMI and retry failed (if retry was enabled). Probably indicates an NXM (nonexistent memory). XBER<TTO> should also 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, do so under the conditions stated above.
XBER<NRR>	The first quadword of read data was not returned before the MAXMI timeout expired. The read command was ACKed on the XMI, indicating that this is not an NXM. Probably due to a parity error on the returned data (if so, XBER<PE> will also be set). XBER<TTO> should also be set.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3.

Restart condition: Restart is not possible if XBER<FCMD> = OREAD, since a write may have been lost.

2.11.6.9 BCSTS<AC PERR>

Description: The MAXMI and/or MC-chip detected a command/address parity error on a DAL command. Either XBEER<ACPE> and/or BCSTS<AC PERR> will be set. Probably due to a flipped bit on the A bus or CMD lines. The MAXMI and the MC-chip will ignore the transaction when this error occurs. This error may also be reported as an MSSC bus timeout.

Recovery procedure: Clear all error bits in XBEER and BCSTS.

Restart condition: No retry is possible because this error could indicate a failed write.

2.11.6.10 BCSTS<SECOND ERR>

Description: The MC-chip detected an error while a previous error was still being serviced. BCSTS<SECOND ERR> will be set.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3.

Restart condition: Retry is not possible since the second error state is unknown.

2.11.6.11 XBER<PE>

Description: During every XMI cycle, the MAXMI 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 procedure: Clear all error bits in XBER.

Restart condition: Retry is not possible since this is a systemwide error.

2.11.6.12 XBER<IPE>

Description: The KA65A CPU module detected a parity error on an XMI cycle. At least one node detected good parity during the cycle time that the KA65A CPU module detected a parity error. When this bit is set, at least one bit (XBEER<MDAXPE> XBEER<MCAXPE>) sets and both XBER<PE> and XBER<ES> are set.

Recovery procedure: Clear all error bits in XBER and XBEER.

Restart condition: Retry is not possible since this is a systemwide error.

2.11.6.13 XBEER<WDPE>

Description: The MAXMI detected a parity error on the DAL data for a write from the MP-chip. XBEER<WDPE> should be set. Probably due to a bit flipped on the D bus or to intentional/unintentional bad parity generated by the MP-chip. The XMI transaction is suppressed. This bit is only used for I/O writes and memory space writes when the backup cache is off or in ETM.

Recovery procedure: Clear all error bits in XBEER.

Restart condition: Retry is not possible.

2.11.6.14 XBEER<ACPE>

Description: The MAXMI and/or MC-chip detected an address/command parity error on a DAL command. XBEER<ACPE> and/or BCSTS<AC PERR> should be set. Probably due to a bit flipped on the A bus or command lines. When either the MAXMI or the MP-chip detect an address /command parity error they ignore the transaction. This error can also be reported as an MSSC bus timeout.

Recovery procedure: Clear all error bits in XBEER and BCSTS.

Restart condition: Retry is not possible.

2.11.6.15 XBEER<URR>

Description: The MAXMI received a read response on the XMI when it was not expecting one. XBEER<URR> is set. This indicates a communication problem between the MAXMI and other XMI devices. Other information about this error is not saved.

Recovery procedure: Clear all error bits in XBEER.

Restart condition: Restart from this error is not possible since the processor state might have been corrupted.

2.11.6.16 VINTSR<VHE>

Description: The vector module detected an unrecoverable internal error. VINTSR<VECTOR HARD ERROR> should be set. The VECTL CSR indicates the source of the error. Other vector module registers may contain additional information. Refer to Chapter 3 for a list of possible errors.

Recovery procedure: Perform the full vector error recovery described in Section 2.11.2.3.

Restart condition: Since hardware retry failed, vector state may be corrupted and no retry should be attempted. An image exit should be forced on the process executing in the vector module at the time of the error.

2.11.6.17 VINTSR<VECTL VIB HERR>

Description: The VECTL detected an unrecoverable command or write data parity error during a VIB transaction. VINTSR<VECTL VIB HERR> should be set.

Recovery procedure: Perform the full vector error recovery described in Section 2.11.2.3.

Restart condition: Since hardware retry already failed, communication with the vector module is no longer possible and no retry should be attempted. An image exit should be forced on the process executing in the vector module at the time of the error.

2.11.6.18 VINTSR<CCHIP VIB HERR>

Description: The MC-chip detected an unrecoverable data parity error during a VIB read transaction. VINTSR<CCHIP VIB HERR> should be set.

Recovery procedure: Perform the full vector error recovery described in Section 2.11.2.3.

Restart condition: Since hardware retry already failed, communication with the vector module is no longer possible and no retry should be attempted. An image exit should be forced on the process executing in the vector module at the time of the error.

2.11.6.19 VINTSR<BUS TIMEOUT>

Description: The MC-chip detected the assertion of the ERR L signal while holding a vector EPR read transaction on the DAL. This error may be due to an insufficient timeout value in the MSSC timeout register. VINTSR<Bus Timeout> and SSCBTR<BTO> should both be set.

Recovery procedure: Perform the full vector error recovery described in Section 2.11.2.3. Then clear SSCBTR<BTO>.

Restart condition: Since a DAL vector EPR transaction was terminated before the corresponding VIB transaction completed, the VIB is in an inconsistent state and no retry should be attempted. An image exit should be forced on the process executing in the vector module at the time of the error.

2.11.6.20 XFAER<FCMD> = IDENT

Description: The MAXMI detected an XMI error while processing an IDENT command. XFAER<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 MAXMI terminated the DAL command with the ERR L signal asserted, which is then reported as a hard error.

The source of the error is determined by the state of these bits:

XBER<RSE>	The interrupt vector was returned with the wrong sequence number. The adapter returned an RER response.
XBER<RER>	A quadword of read data was returned with an RER, indicating that the memory got a double-bit error reading the array. Probably not recoverable, but it is possible.
XBER<TTO>	The interrupt vector was not returned before the MAXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set. If neither bit is set, the MAXMI was never granted the XMI for the IDENT.
XBER<CNAK>	The IDENT command was NO ACKed by the adapter and retry failed, if retry is enabled. This may indicate that the target for the IDENT is not present. XBER<TTO> should also be set.
XBER<NRR>	The interrupt vector was not returned before the MAXMI timeout expired. The read command was ACKed on the XMI, so the adapter received the IDENT. XBER<TTO> should also be set.
XBEER<OLR>	Lockout responses were received while attempting to do a read-type transaction. This is due to failure of the lockout mechanism to provide access to an interlock or the ownership of a memory location. If OLR is set, XBER<TTO> is also set.

Recovery procedure: Clear all error bits in XBER.

Restart condition: 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.

2.11.7 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 2–2. Figure 2–33 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.

2.11.7.1 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.

2.11.7.2 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.

2.11.7.3 PCSTS<P 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 procedure: Write all P-cache tags with good parity and clear valid bits. Then perform the full memory error recovery procedures described in Section 2.11.2.3. If the error reoccurs, disable the P-cache.

Figure 2–33 Soft Error Interrupt Parse Tree

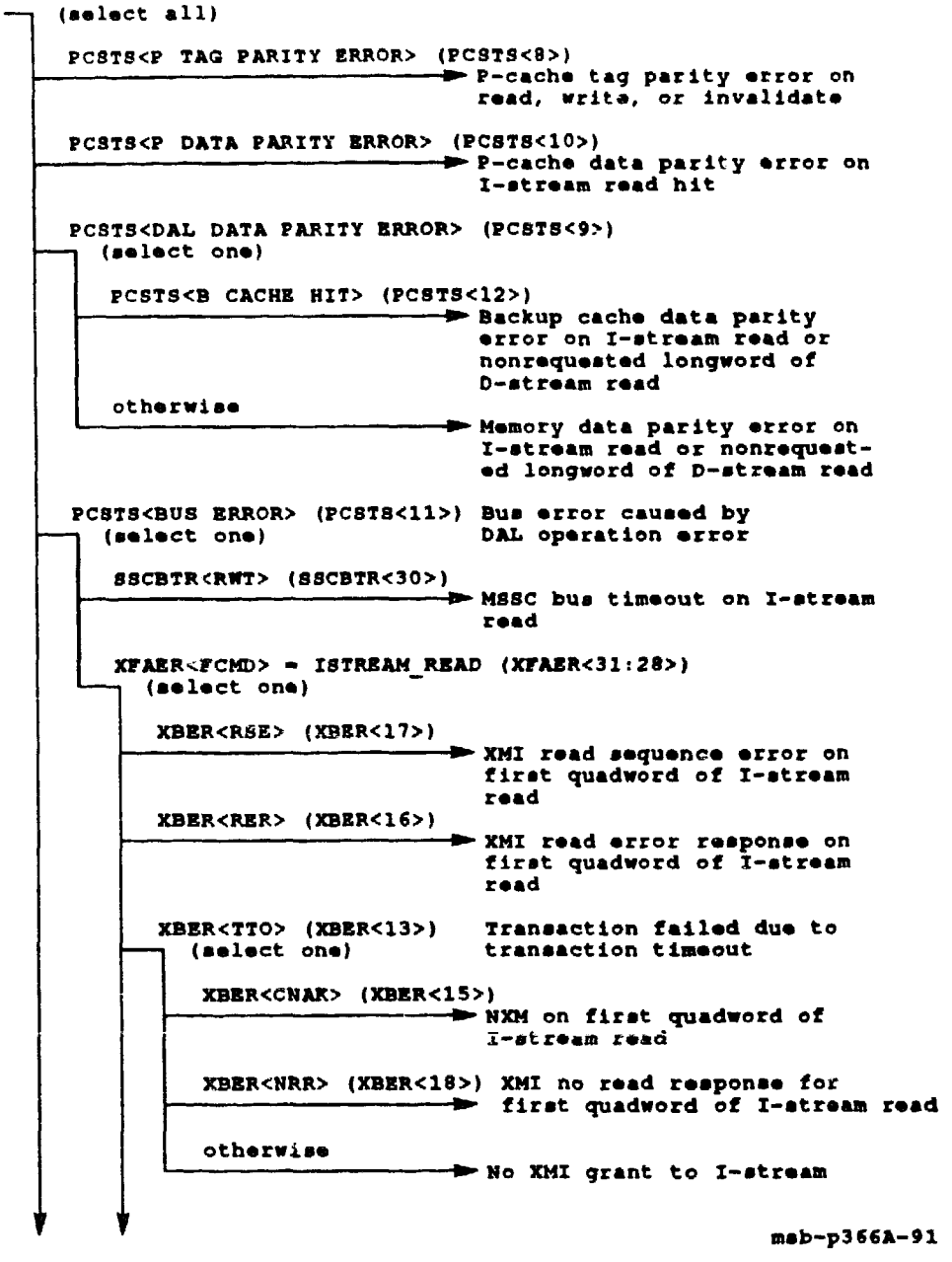
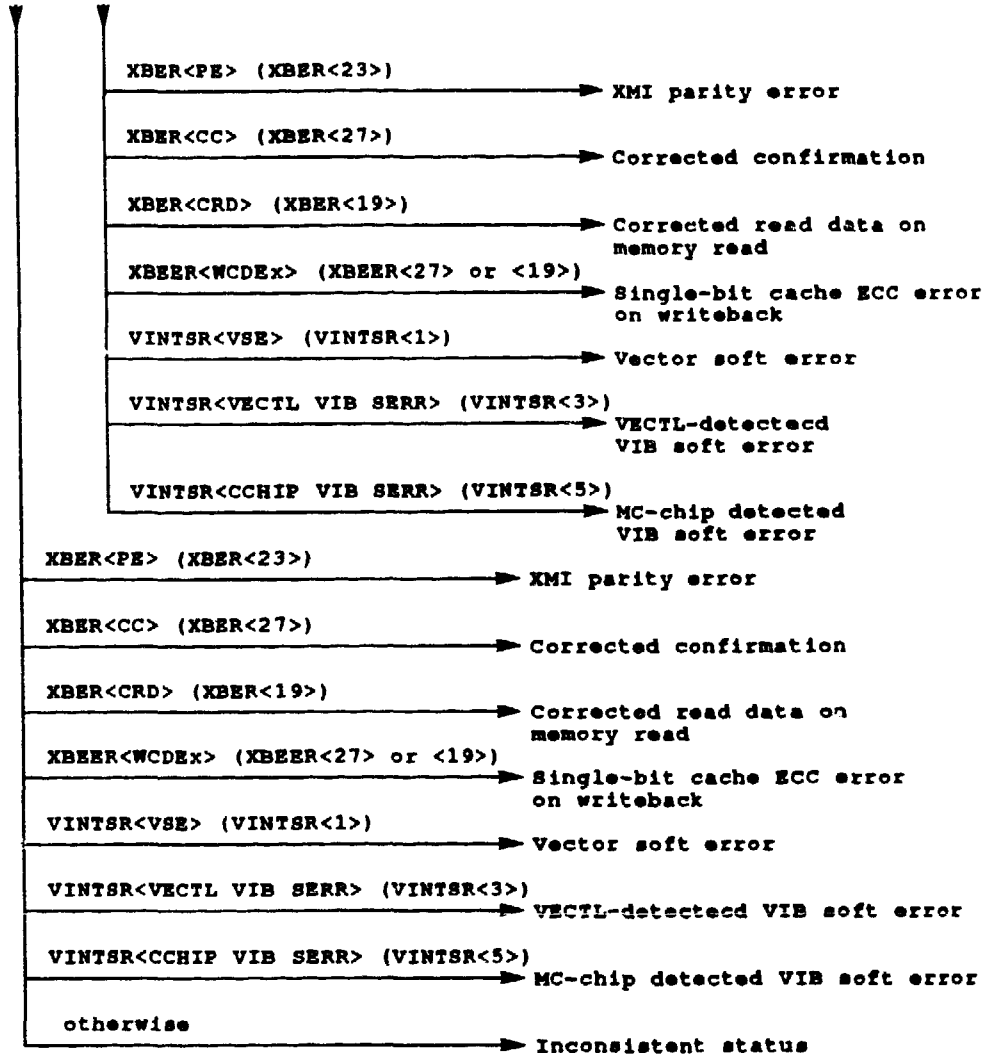


Figure 2–33 Cont'd on next page

Figure 2–33 (Cont.) Soft Error Interrupt 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.

msb-p367A-91

2.11.7.4 PCSTS<P DATA PARITY ERROR>

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 procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3. If the error reoccurs, disable the P-cache.

2.11.7.5 PCSTS<DAL DATA PARITY ERROR>

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>.

2.11.7.5.1 PCSTS<B CACHE HIT>

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 procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3. If the error reoccurs, disable the backup cache.

2.11.7.5.2 Memory Data Parity Error

Description: A memory data parity error was detected during an I-stream read or in the nonrequested longword of a D-stream read. An actual memory parity error would be detected by the MAXMI and reported as a memory read error. A Memory Data Parity Error implies that the parity went bad between the MAXMI and the MP-chip. PCSTS<Interrupt> and PCSTS<DAL DATA PARITY ERROR> should both be set. PCSTS<B CACHE HIT> should be clear.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3.

2.11.7.5.3 PCSTS<BUS ERROR>

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 a non-I/O space address for BCSTS to log the error.

2.11.7.5.3.1 SSCBTR<RWT>

Description: The MSSC timed out on an I-stream read. SSCBTR<RWT>, SSCBTR<BTO>, PCSTS<BUS ERROR>, and PCSTS<Interrupt> should all be set.

Recovery procedure: Clear SSCBTR<RWT> and SSCBTR<BTO>. Then perform the full memory error recovery procedures described in Section 2.11.2.3.

Restart condition: This error should never occur unless the reference is to an unimplemented node private space address to which the MAXMI does not respond since the MSSC timeout should be longer than the MAXMI timeout.

2.11.7.5.3.2 XFAER<FCMD> = ISTREAM_READ

Description: The MAXMI detected an error on the first quadword of an I-stream read. XFAER<FMCD> should be a read. XFADR and XFAER contain the physical address of the error.

The source of the error is determined by the state of these bits:

XBER<RSE> The first quadword of read data was returned with the wrong sequence number. PCSTS<BUS ERROR> and PCSTS<Interrupt> should be set.

XBER<RER> 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> and PCSTS<Interrupt> should be set.

XBER<TTO> The first quadword of read data was not returned before the MAXMI timeout expired. XBER<NRR> or XBER<CNAK> should also be set. If neither bit is set, the MAXMI was never granted the XMI for the read command. PCSTS<BUS ERROR> and PCSTS<Interrupt> should 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. PCSTS<BUS ERROR> and PCSTS<Interrupt> should be set.

XBER<NRR> The first quadword of read data was not returned before the MAXMI 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>, and PCSTS<Interrupt> should be set.

Recovery procedure: Perform the full memory error recovery procedures described in Section 2.11.2.3.

2.11.7.6 Other MAXMI-Detected Errors

Description: The MAXMI detected an error during an XMI transaction. The state of these bits 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 XCR<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 XCR<CRDID> is set.

XBEER<WCDEx> A single-bit ECC error was detected on the data address lines and corrected by the MAXMI before the Disown Write was issued on the XMI bus. XBER <NSES> will also be set.

VINTSR<VSE> The vector module detected a recoverable internal error. VINTSR<VSE> should be set and the vector module registers should indicated the exact error. Refer to Chapter 3 for a list of possible errors.

VINTSR<VECTL VIB SERR> The VECTL detected a recoverable command or write data parity error during a VIB transaction. VINTSR<VECTL VIB SERR> should be set, and the VECTL CSR should indicate the exact error.

VINTSR<CCHIP VIB SERR> The MC-chip detected a recoverable data parity error during a VIB read transaction. VINTSR<CCHIP VIB SERR> should be set.

Recovery procedure: Clear all error bits in XBER.

2.11.7.7 XBER<PE>

Description: During every XMI cycle, the MAXMI 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 procedure: Clear all error bits in XBER.

Restart condition: Retry is not possible since this is a systemwide error.

2.11.7.8 XBER<CC>

Description: The XMI Corner corrected a single-bit error in a confirmation code. This interrupt is posted unless XCR<CCID> is set.

Recovery procedure: Clear all error bits in XBER and XCR.

Restart condition: Retry is possible.

2.11.7.9 XBER<CRD>

Description: 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 XCR<CRDID> is set.

Recovery procedure: Clear all error bits in XBER and XCR.

Restart condition: Retry is not possible.

2.11.7.10 XBEER<WCDEx>

Description: A single-bit ECC error was detected on the data address lines and corrected by the MAXMI before the Disown Write was issued on the XMI bus. XBER<NSES> will also be set.

Recovery procedure: Clear all error bits in XBER and XBEER.

Restart condition: Retry is possible.

2.11.7.11 VINTSR<VSE>

Description: The vector module detected a recoverable internal error. VINTSR<VSE> should be set and the vector module registers should indicate the exact error. Refer to Chapter 3 for a list of possible errors.

Recovery procedure: Perform the full vector error recovery procedures described in Section 2.11.2.3. If the error reoccurs, discontinue use of the vector module.

2.11.7.12 VINTSR<VECTL VIB SERR>

Description: The VECTL detected a recoverable command or write data parity error during a VIB transaction. VINTSR<VECTL VIB SERR> should be set, and the VECTL CSR should indicate the exact error.

Recovery procedure: Perform the full vector error recovery procedures described in Section 2.11.2.3. If the error reoccurs, discontinue use of the vector module.

2.11.7.13 VINTSR<CCHIP VIB SERR>

Description: The MC-chip detected a recoverable data parity error during a VIB read transaction. VINTSR<CCHIP VIB SERR> should be set.

Recovery procedure: Perform the full vector error recovery procedures described in Section 2.11.2.3. If the error reoccurs, discontinue use of the vector module.

2.11.8 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 2–2.

2.11.9 Errors with No Notification

One category of MAXMI-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.

2.11.9.1 CSR Read Data NO ACK

Description: When another XMI node reads a MAXMI CSR, the MAXMI responds with the requested data. If the data transfer is not ACKed by the receiving node, the MAXMI sets XBER<RIDNAK>. This will usually be reported on the receiving node as a no read response error.

Recovery procedure: Clear all error bits in XBER.

2.11.9.2 CSR Write Sequence Error

Description: When another XMI node writes a MAXMI CSR, the MAXMI 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 procedure: Clear all error bits in XBER.

2.11.10 Error Recovery Coding Examples

Example 2–1 is pseudocode 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 2–1 Error Recovery Coding

```
PROCEDURE collect_error_state;

! This support procedure collects error state from the
! appropriate KA65A CPU module registers.

    BEGIN

    s_xber   := XBER;                    ! save MAXMI XBER register
    s_xfadr  := XFADR;                   ! save MAXMI XFADR register
    s_xfaer  := XFAER;                   ! save MAXMI XFAER register
    s_xbeer  := XBEER;                   ! save MAXMI XBEER register
    s_wfadr0 := WFADR0;                  ! save MAXMI WFADR0 register
    s_wfadr1 := WFADR1;                  ! save MAXMI WFADR1 register
    s_fdal0  := FDAL0;                   ! save MAXMI FDAL0 register
    s_fdal1  := FDAL1;                   ! save MAXMI FDAL1 register
    s_fdal2  := FDAL2;                   ! save MAXMI FDAL2 register
    s_fdal3  := FDAL3;                   ! save MAXMI FDAL3 register
    s_sscbtr := SSCBTR;                  ! save MSSC SSCBTR register
    s_bcctl  := BCCTL;                   ! save MC-chip BCCTL register
    s_bcsts  := BCSTS;                   ! save MC-chip BCSTS register
    s_bcera  := BCERA;                   ! save MC-chip BCERA register
    s_bcert  := BCERT;                   ! save MC-chip BCERT register
    s_pcsts  := PCSTS;                   ! save P-cache PCSTS register
    s_pcerr  := PCERR;                   ! save P-cache PCERR register
    s_vintsr := VINTSR;                  ! save MC-chip VINTSR register

    END;  ! collect_error_state

PROCEDURE disable_cache;

! This support procedure disables the P-cache and forces the backup
! cache into ETM during error recovery.

    BEGIN

! Disable the P-cache and make sure
! force hit is off, while leaving the WC bits alone.

    PCSTS := ^ENABLE_PTS+^FORCE_HIT;

! Put the backup cache into ETM.

    BCCTL := ENABLE_BTS+BTS_ERROR_TRAN+^FORCE_BHIT;

! Initialize flags used to determine if cache should be reenabled.
! If either cache is already disabled, they are not reenabled on restart.

    pcache_disable := s_pcsts<ENABLE_PTS> AND s_bcctl<ENABLE_PTS>;
    bcache_disable := s_bcctl<ENABLE_BTS>;

    END;  ! disable_cache
```

Example 2–1 Cont'd on next page

Example 2–1 (Cont.) Error Recovery Coding

```
PROCEDURE enable_cache;

! This support procedure conditionally reenables both caches if they
! were enabled before, and no error thresholds have been exceeded. The
! backup cache must be enabled before the P-cache to ensure cache coherency.

    BEGIN

! Assume that both caches will be enabled.

    bc_enable := ENABLE_BTS+^BTS_ERROR_TRAN+^FORCE_BHIT;

    pc_enable := 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 P-cache shouldn't be enabled, clear enable bit.

    IF pcache_disable THEN
        BEGIN
        pc_enable := pc_enable AND NOT ENABLE_PTS;
        END;

! Flush all tag stores and reenable the caches as appropriate.

    For i := 0 to 4095 DO
        BEGIN
        BCIDX := i*128;
        BCDET := 0;
        BCBTS := %x00000300;
        END;

    BCCTL := bc_enable;
    PCSTS := pc_enable;

    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 KA65A CPU module
! registers.  This restores the "M" bits to their orignal state
! and clears the "WC" error bits.

    BEGIN

! Clear MAXMI errors.

    XBER := s_xber;                 ! clear XBER error
    XBEER := s_xbeer;               ! clear XBEER error

! Clear MSSC bus timeout errors.

    SSCBTR := s_sscbtr;             ! clear MSSC timeout errors
```

Example 2–1 Cont'd on next page

Example 2–1 (Cont.) Error Recovery Coding

```
! Check for MC-chip backup tag store parity error.
! Correct tag store and write any dirty blocks back to memory.
! If error threshold is exceeded, disable the cache on exit.

    IF ^s_bcsts<BTS_TPERR> AND s_bcsts<BTS_VDPERR> THEN
    BEGIN

! Tag address is good, valid/dirty bits have error. Check memory
! controller for dirty subblocks, rewrite, and then deallocate tag.

    test_addr<31> := 0;
    test_addr<30:19> := s_bcert<30:19>;
    test_addr<18:7> := s_bcera<18:7>;
    test_addr<6:0> := 0;

    new_tag<30:9> := s_bcert<30:9>;
    new_tag<8:0> := 0;

    For i := 0 to 3 DO
       BEGIN
       IF memory_state(test_addr) = (OWNED AND (ID = KA65A CPU Module)) THEN
          BEGIN
          new_tag<i> := 1;
          new_tag<i+4> := 1;
          END;
       test_addr<6:5> := test_addr<6:5> + 1;
       END;

    new_tag<8> := odd_parity(new_tag<7:0>)
    BCIDX := s_bcerr;
    BCBTS := new_tag;
    BCDET := 0;

    END;

IF s_bcsts<BTS_TPERR> AND ^s_bcsts<BTS_VDPERR> AND
s_bcert<7:4> = 0 THEN
    BEGIN

! Valid/dirty bits are good and subblocks are not owned. Rewrite tag
! with good parity.

    BCIDX := s_bcerr;
    BCBTS := %x00000300;

    END;

IF (s_bcsts<BTS_TPERR> AND s_bcsts<BTS_VDPERR>) OR
   (s_bcsts<BTS_TPERR> AND ^s_bcsts<BTS_VDPERR> AND
s_bcert<7:4> <> 0) THEN

! Both address and valid/dirty bits have parity errors, or address has
! parity error and at least 1 subblock is dirty. Search 13 possible
! block addresses for dirty subblocks. If any dirty subblocks are found,
! rewrite and deallocate tag. If no dirty subblocks are found, rewrite
! the tag with good parity.
```

Example 2–1 Cont'd on next page

Example 2–1 (Cont.) Error Recovery Coding

```
    BEGIN

    test_addr<31> := 0;
    test_addr<30:19> := s_bcert<30:19>;
    test_addr<18:7> := s_bcera<18:7>;
    test_addr<6:0> := 0;

    new_tag<31:0> := 0;
    tag_found := false;

    For j:= 0 to 3 DO

        BEGIN
        IF memory_state(test_addr) = (OWNED AND (ID = KA65A CPU module)) THEN
           BEGIN
           tag_found := true;
           new_tag<j> := 1;
           new_tag<j+4> := 1;

        test_addr<6:5> := test_addr<6:5> + 1;
        END;

    i := 19
    While (i <= 30) AND ^tag_found DO
       BEGIN
       test_addr<i> := ^test_addr<i>

       For j := 0 to 3 DO

           BEGIN
        IF memory_state(test_addr) = (OWNED AND (ID = KA65A CPU module)) THEN
              BEGIN
              tag_found := true;
              new_tag<i> := 1;
              new_tag<i+4> := 1;
              END;
           test_addr<6:5> := test_addr<6:5> + 1;
           END;
       IF ^tag_found THEN test_addr<i> := ^test_addr<i> ;
       END;(While)

    new_tag<8> := odd_parity(new_tag<7:0>)

    IF tag_found THEN
       BEGIN
       BCIDX := s_bcerr;
       BCBTS := new_tag;
       BCDET := 0;
       END
    ELSE
       BEGIN
       BCIDX := s_bcerr;
       BCBTS := %x00000300;
       END;

    END;

       IF (backup tag store error threshold exceeded) THEN
           bcache_disable := TRUE;
       END;

! Clear MC-chip errors.

    BCSTS := s_bcsts;

! Check for P-cache tag parity errors, which require rewriting the
! entire tag store.  If error threshold is exceeded, disable the cache
! on exit.
```

Example 2–1 Cont'd on next page

Example 2–1 (Cont.) Error Recovery Coding

```
    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 := %X80000000;
            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)
    IF PCSTS<ENABLE_PTS> THEN
        pc_enable := pc_enable OR ENABLE_PTS;

    END;

PROCEDURE machine_check_handler;

! This procedure is invokved 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 KA65A CPU module registers and the machine check frame.

    collect_error_state;                 ! read KA65A CPU module 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;

! 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

           ! MF-chip related error
           ! If the MF-chip error threshold is exceeded, disable
           ! the MF-chip by clearing ACCS<MF_CHIP_PRESENT>

           IF (MF-chip error threshold exceeded) THEN
              ACCS := ACCS AND NOT MF_CHIP_PRESENT;
           retry := s_r AND (NOT s_psl<FPD>) AND (NOT s_XBEER<UWP>);

           END;
```

Example 2–1 Cont'd on next page

Example 2–1 (Cont.) Error Recovery Coding

```
    MCHK_TBM_ACV_TNV,
    MCHK_TBH_ACV_TNV,
    MCHK_INT_ID_VALUE,
    MCHK_UNKNOWN_IBOX_TRAP,
    MCHK_UNKNOWN_CS_ADDR:
        BEGIN

        ! Internal MP-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_XBEER<UWP>) AND
                 (NOT disable_retry);

        END;

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_XBEER<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,
MCHK_ERROR_ISTREAM;
    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_XBEER<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;


    MCHK_VECTOR_STATUS:
        BEGIN

    ! Vector module error
    ! Perform full vector error recovery
```

Example 2–1 Cont'd on next page

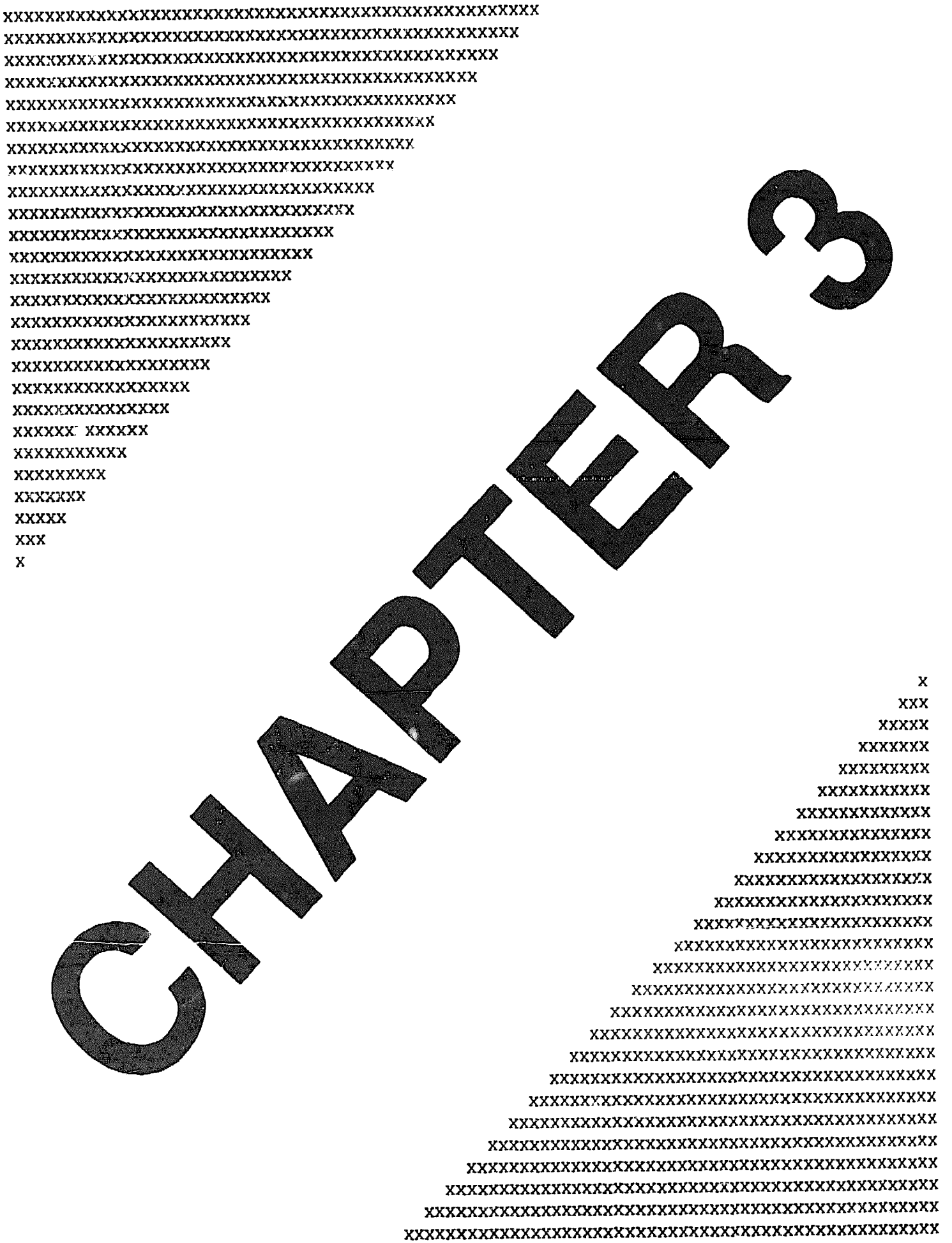
Example 2–1 (Cont.) Error Recovery Coding

```
            recover_from_vector_errors;
            retry := FALSE;

            END;

        OTHERWISE
            BEGIN

        ! Unknown machine check code

            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.

    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 KA65A CPU module registers.

    collect_error_state;

! If cache-related error, disable cache during processing.

    IF s_pcsts<INTERRUPT> THEN
        disable_cache;

! Recover from errors by performing the memory error recovery procedures.

    recover_from_memory_subsystem_errors;

! Recover from soft vector errors by performing the vector
! error recovery procedures.

    recover_from_vector_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
```

Example 2–1 Cont'd on next page

Example 2–1 (Cont.) Error Recovery Coding

```
    IF s_pcsts<INTERRUPT> THEN
        enable_cache;

! Soft error interrupts should be retried.

    REI;

    END;  ! soft_error_interrupt_handler


PROCEDURE hard_error_interrupt_handler;

! This procedure is invoked via SCB vector 60 (hex) to process
! hard error interrupts at IPL 1D (hex).

    BEGIN

! Collect error state from KA65A CPU module registers.

    collect_error_state;

! Disable P-cache and backup cache to minimize the possibility of
! generating a machine check while servicing this interrupt.

    disable_cache;

! Hard error interrupts are recovered by performing the memory
! error recovery procedures.

    recover_from_memory_subsystem_errors;

! Recover from any hard vector errors by performing the
! vector error recovery procedures.

    recover_from_vector_errors;

! Log all registers and the PC/PSL pair in the error log for later
! analysis.  It is important to log everything as this will aid
! in the analysis of unexpected errors.

    {log error state and interrupt stack frame};

! Conditionally reenable caches

    enable_cache;

! Restart from this interrupt is a function of the exact cause of
! the error.  XFAULT, WEI, and IPE are usually fatal to the system.
! Write errors are fatal to the process or the system depending on
! the write address.  IDENT errors may potentially be retried at
! the device level.  Vector errors are fatal to the process.

    {perform appropriate restart action};

    REI;

    END;  ! hard_error_interrupt_handler;
```


CHAPTER 3

3 FV64A Vector Processor Module

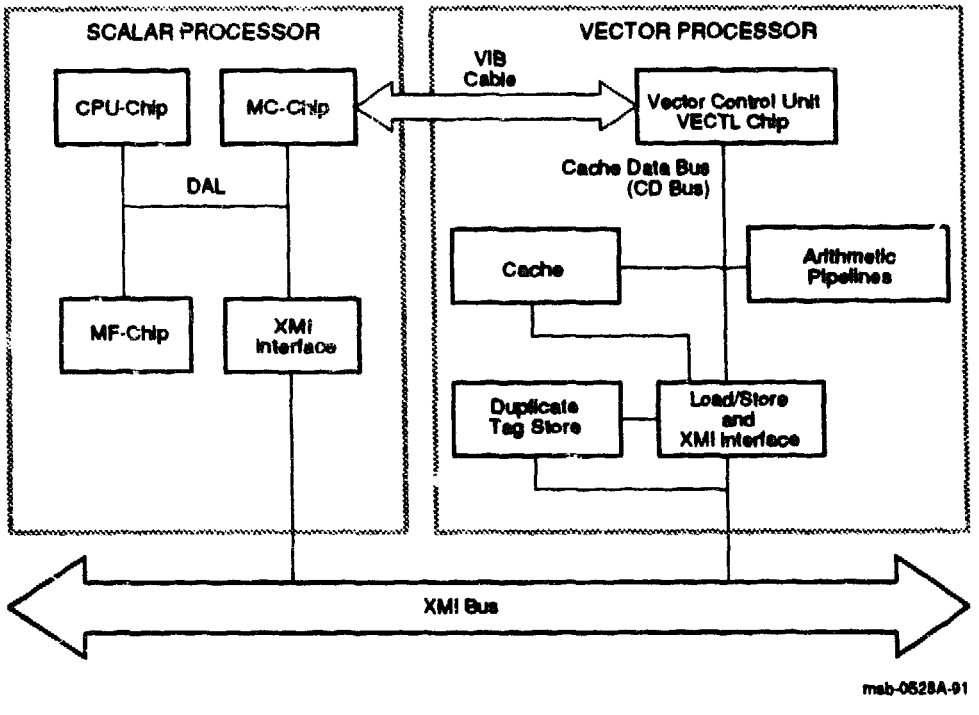
This chapter describes the FV64A vector processor module. The chapter includes the following sections:

- Overview
- Functional Units
- Block Diagram Description
- Vector Control Unit
- Arithmetic Unit
- Load/Store Unit
- Cache Memory
- Vector Processor Registers
- Error Handling

3.1 Overview

The FV64A vector processor is a single-board option that implements the VAX vector instruction set. Figure 3–1 is a block diagram of a scalar/vector pair.

Figure 3–1 Scalar/Vector Pair Block Diagram



The FV64A vector processor requires a scalar KA65A CPU module (KA64A CPU module for the Model 400 system) 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 MC-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.

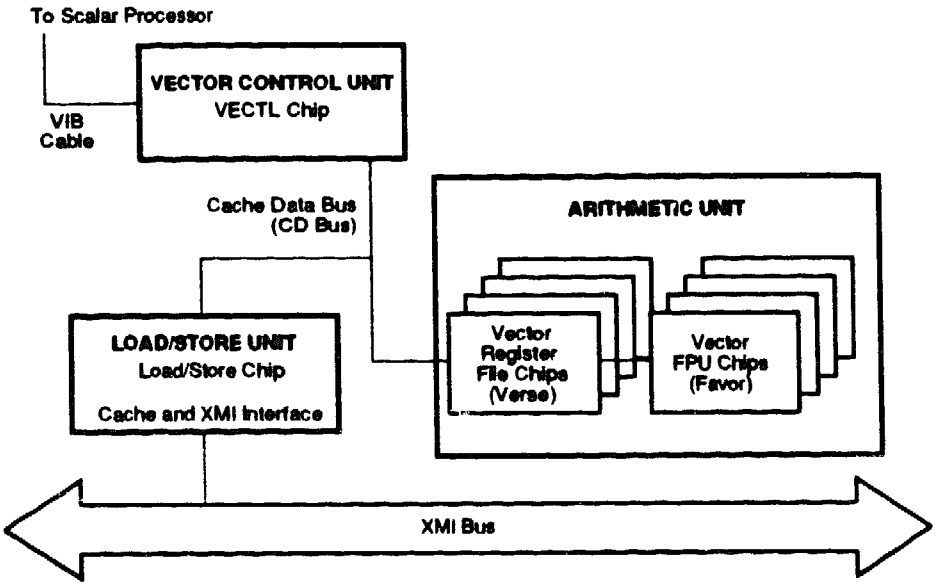
The FV64A vector processor is a standard XMI module but it is not a full-fledged XMI node. The vector processor functions only as an XMI commander. It performs memory transactions over the XMI bus without intervention from the scalar module. The FV64A vector processor does not issue transactions to I/O space or respond to XMI transactions directed to it.

A multiprocessor system can have four scalar/vector pairs. For optimal performance, two memory modules are required with a scalar/vector pair. Four memory modules are required with two or more scalar/vector pairs.

3.2 Functional Units

Figure 3–2 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 3–2 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.

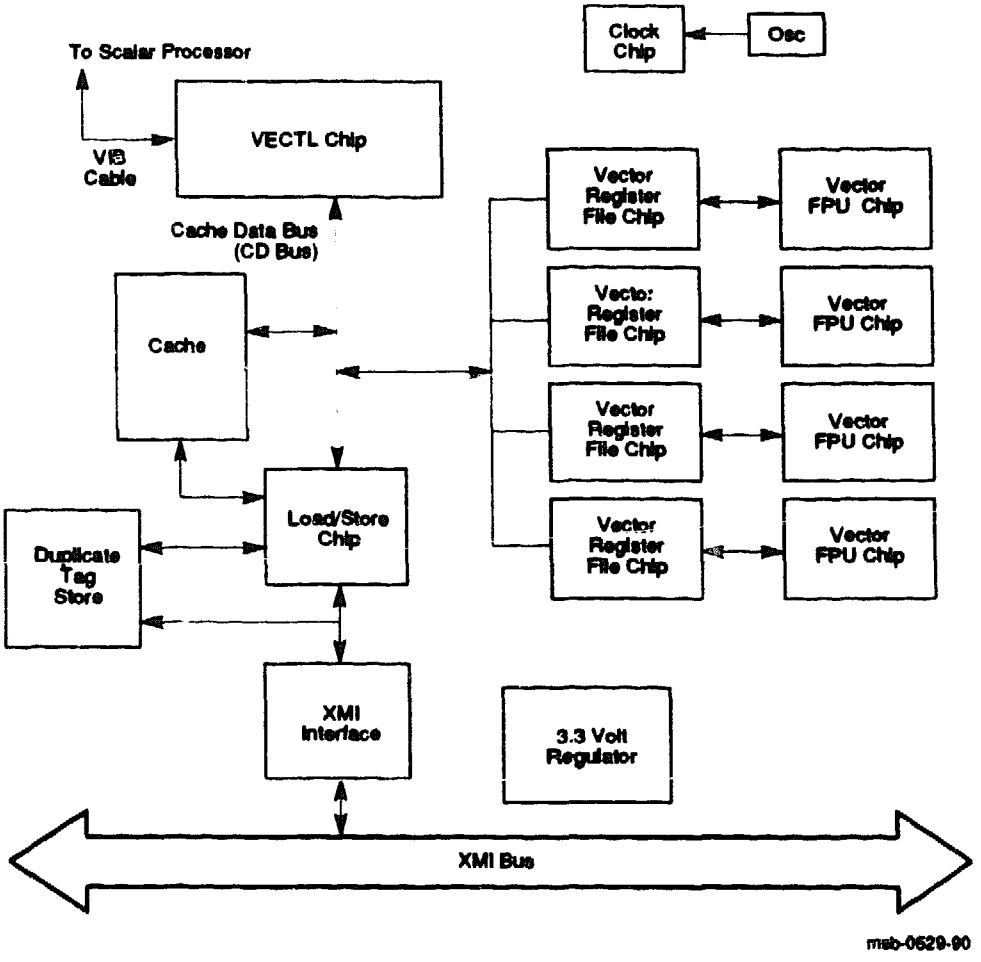
The three functional units are described in more detail in these sections:

- Vector control unit — Section 3.4
- Arithmetic unit — Section 3.5
- Load/store unit — Section 3.6

3.3 Block Diagram Description

Figure 3–3 is a block diagram of the vector module. This section summarizes the major chips on the module.

Figure 3–3 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

The module also has a regulator that converts 5 volts to 3.3 volts.

3.3.1 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).

3.3.2 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.

3.3.3 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.
- Five-stage pipelined data path.

3.3.4 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.

3.3.5 Clock Chip

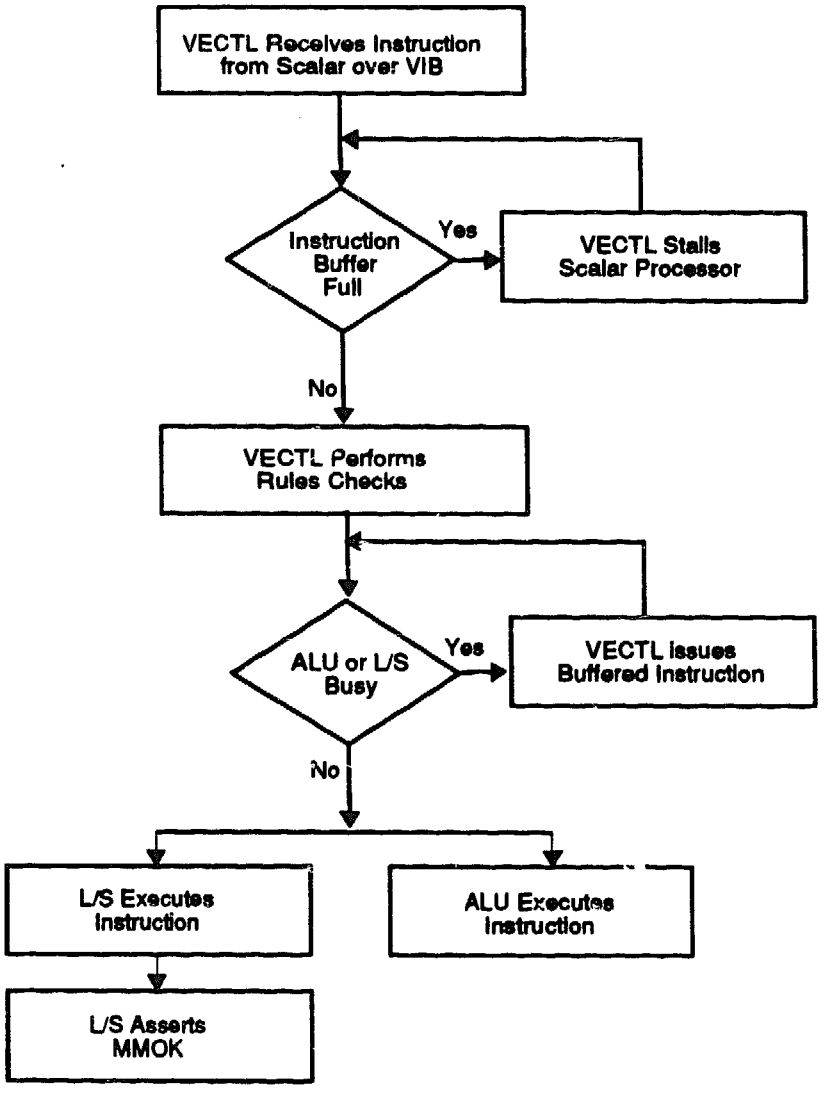
The clock chip (CLK-chip, DC598) provides the main system clocks to all chips on the vector module except for the Favor chips, which get their clocks from the Verse chips.

3.4 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 3–4 VECTL Chip Instruction Flow



3.4.1 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 3–4 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 3.4.2). 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.*

3.4.2 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.

3.4.3 Data Types

The FV64A vector processor supports five data types:

- Longword
- Quadword
- F_floating
- G_floating
- D_floating

3.4.4 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.

3.4.4.1 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.

3.4.4.2 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.

3.4.4.3 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.

3.4.4.4 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.

3.4.4.5 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.

3.4.4.6 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.

3.4.4.7 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.

3.4.4.8 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.

3.4.4.9 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.

3.4.4.10 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.

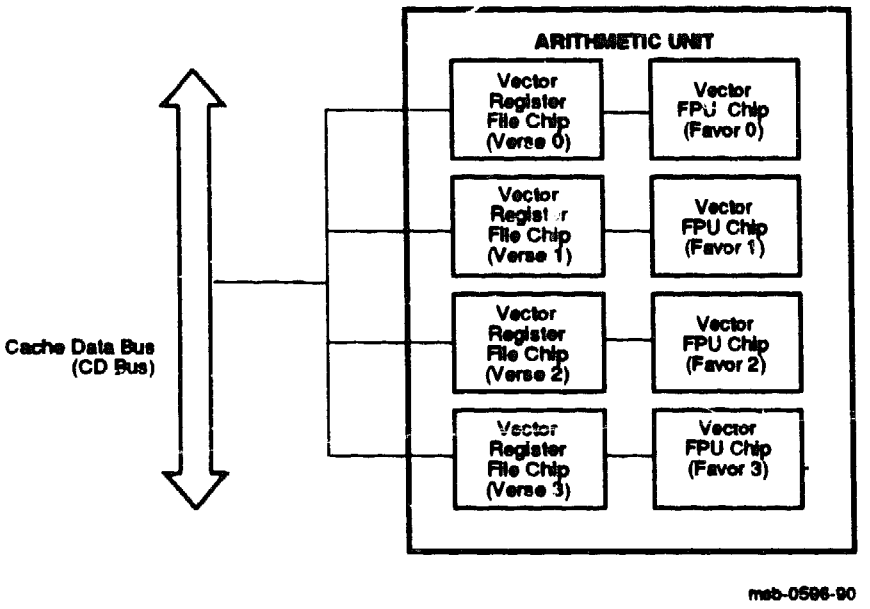
3.4.4.11 VSYNC Instruction

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.

3.5 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 3–5 Vector Arithmetic Unit



All register-to-register vector instructions are handled by the arithmetic unit (ALU). Figure 3–5 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.

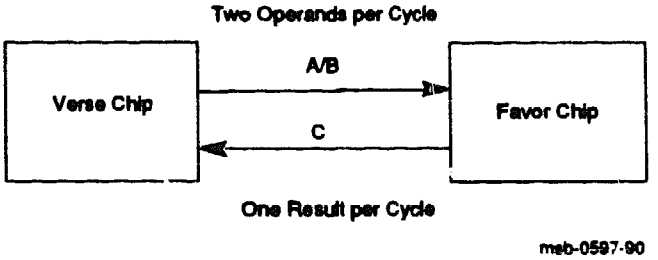
All instructions, except floating-point divide instructions, are fully pipelined. For increased performance all arithmetic instructions are executed by four parallel pipelines. After an initial startup time, the fully pipelined instructions can produce four results each cycle for single-precision instructions 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 3–6). 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 3–6 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.

3.6 Load/Store Unit

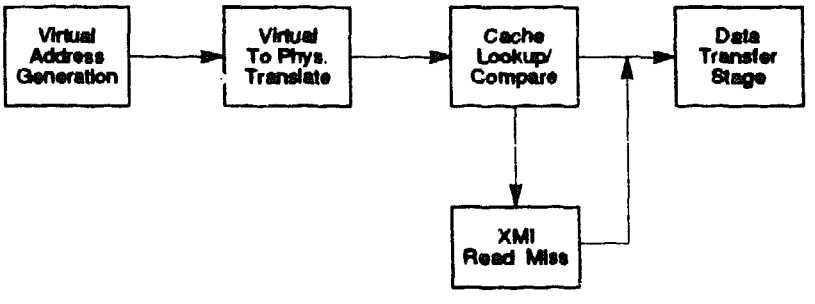
Vector instructions are received by the vector control unit (the VECTL chip) from the scalar processor and then issued to either the load/store unit or the arithmetic unit. The Load/Store chip controls the load/store unit, which consists of a 1-Mbyte cache and the XMI interface logic. There is a duplicate 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 3–7 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 3–7 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.

3.6.1 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.

3.6.1.1 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.

3.6.1.2 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.

3.6.1.3 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)

3.6.2 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.

3.7 Cache Memory

The vector module implements a single-level, direct-mapped cache. The cache is write through, and the size is one Mbyte.

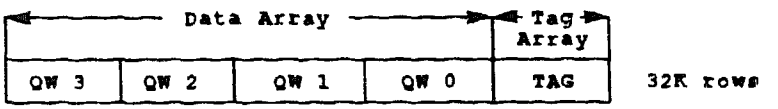
The vector processor implements a 1-Mbyte cache, direct-mapped, with a fill of a hexword (block) and a hexword allocate (block size). The cache is read allocate, no-write allocate, and write through. There are 32K tags, and each tag maps one hexword block. Each tag contains one block valid bit, a 9-bit 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.

3.7.1 Cache Organization

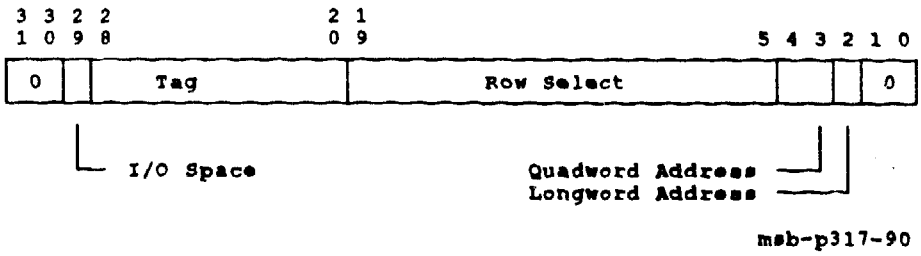
The cache is a 1-Mbyte, direct mapped, write-through cache. The cache is arranged in 32K rows with four quadwords and one tag entry per row. Figure 3–8 shows the cache arrangement. Figure 3–9 shows how the physical address is divided.

Figure 3–8 Cache Arrangement



mab-p316-90

Figure 3–9 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 3–10 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 3–10 Tag Entry Organization

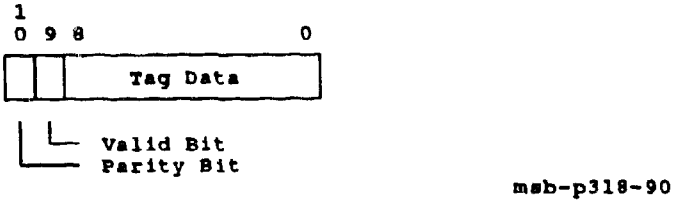
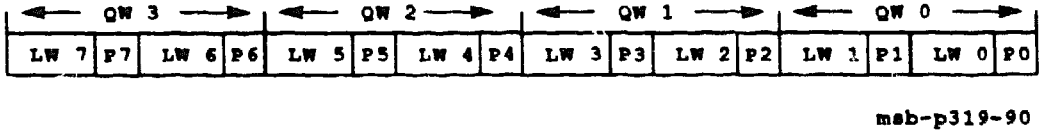


Figure 3–11 shows the organization of the cache data. Each cache block contains four quadwords, with eight longword parity bits.

Figure 3–11 Cache Data Organization



3.7.2 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.

3.7.3 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 that all transactions have finished.

- Issuing the MSYNC instruction—used by user programs
- Reading the Vector Memory Activity Check Register—used by operating systems

3.7.3.1 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.

3.7.3.2 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.

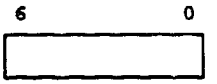
NOTE: It is not possible to monitor the VPSR<BSY> bit to determine instruction completion. This bit is cleared as soon as the instruction completes, not when the memory transfer completes.

3.8 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.

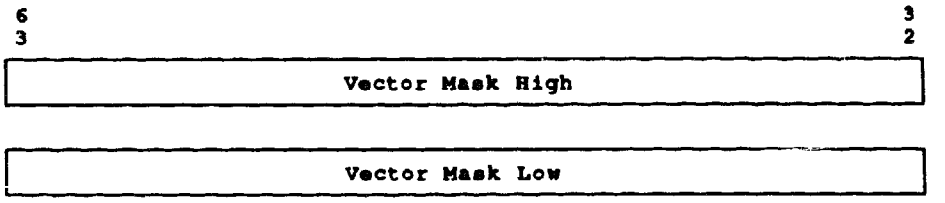
The vector procesor has 16 data registers, each containing 64 elements numbered 0 through 63. Each element is 64 bits wide. The *VAX Architecture Reference Manual* specifies the registers used to access the data registers. Other registers used with the data registers are the Vector Length, Vector Count, and Vector Mask Registers (see Figure 3–12 and Figure 3–13).

Figure 3–12 Vector Length and Vector Count Registers



msb-p320-90

Figure 3–13 Vector Mask Register



msb-p321-90

- 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 3–2 lists the internal processor registers (IPRs), and Table 3–3 lists the vector indirect registers.

3.8.1 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.

3.8.2 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 3–2 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 3–1 lists the codes used to categorize the registers and bits in the following register descriptions.

Table 3–1 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 3–2 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 3–1.

[2]Key to Classes:

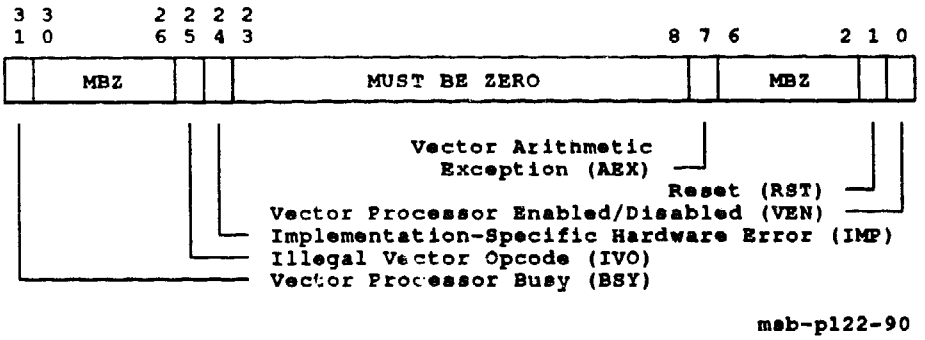
1 = Implemented by the KA65A CPU module as specified in the *VAX Architecture Reference Manual.*
2 = Implemented uniquely by the KA65A CPU module.
3 = Implemented by the FV64A vector processor module.
4 = Implemented in the KA65A CPU module with a copy in the FV64A vector processor module.
n Init = The register is initialized on a KA65A 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)



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

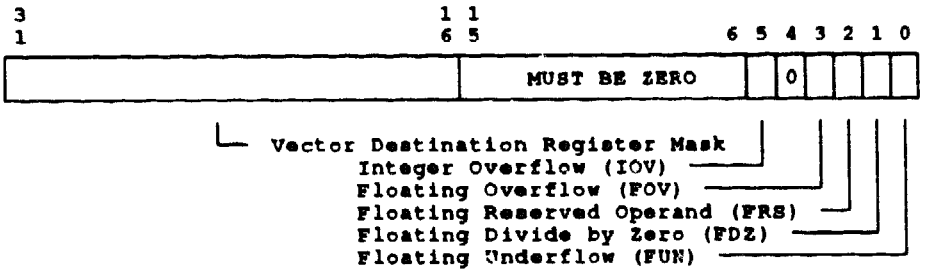
Enable the vector processor by writing a one to Vector Processor Enabled/Disabled. Disable the vector processor by writing a zero to VEN. VEN sets if the processor encounters a disabling fault. When this bit is zero, only MFPR/MTPR instructions can access the internal registers; the scalar CPU blocks 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)*



msb-p123-90

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

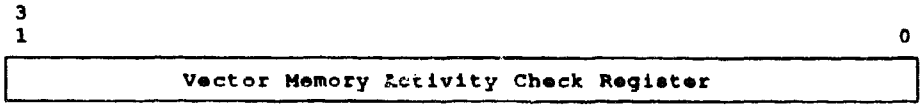
FUN sets when an F_, D_, or G_floating arithmetic or conversion operation underflows the destination exponent.

Vector Memory Activity Check Register (VMAC)

VMAC facilitates memory synchronization between the scalar and vector modules.

ADDRESS

IPR146 (VECTL chip)



msb-p124-90

bits<31:0>

Name: Vector Memory Activity Check

Mnemonic: VMAC

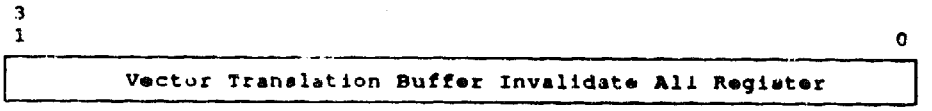
Type: RO, X

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)*



msb-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:

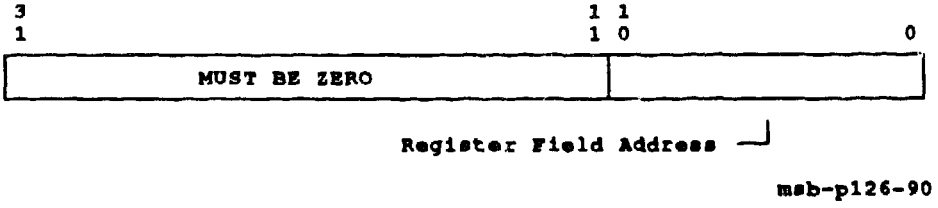
To examine the contents of ALU_OP at address 440:

```
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)

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.

3.8.3 Vector Indirect Registers

The vector module has 16 Kbytes of indirect address space. Table 3–3 lists the registers implemented in that space.

Table 3–3 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 3–3 (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

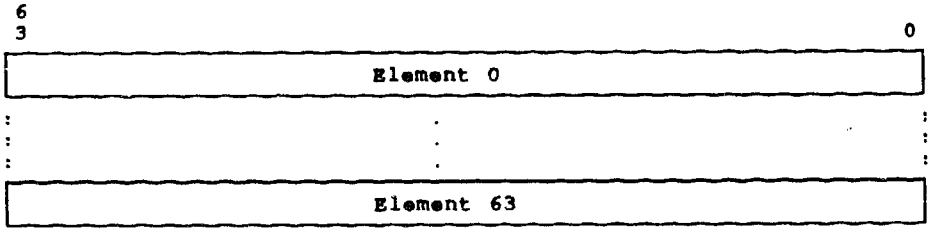
Vector Register *n* (VREG*n*)

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.

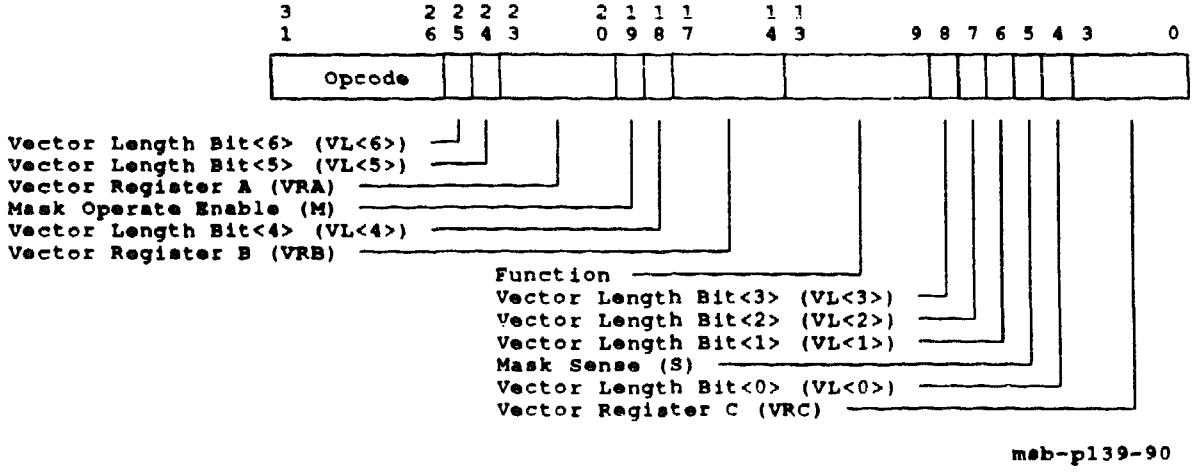
Arithmetic Instruction Register (ALU_OP)

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)*



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.

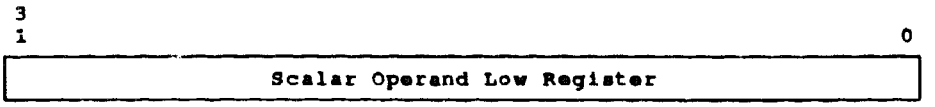
Scalar Operand Low Register (ALU_SCOP_LO)

ALU_SCOP_LO is used to store the scalar operand.

ALU_SCOP_LO is accessed using the indirect address registers.

ADDRESS

448 (Verse chip)



msb-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.

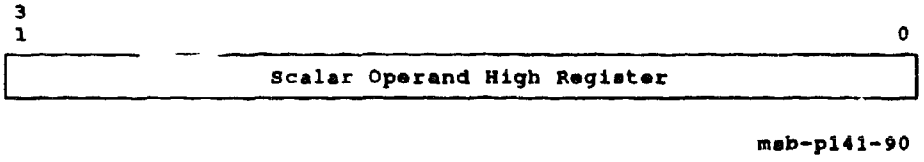
Scalar Operand High Register (ALU_SCOP_HI)

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)

msb-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)

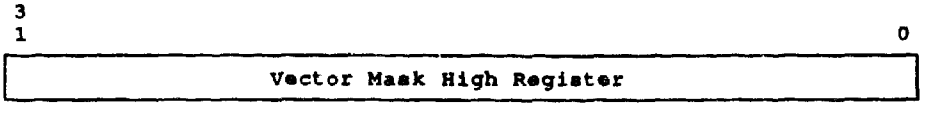
ALU_MASK_HI, when read, causes the high 32 bits of the vector mask to be read. When written, the appropriate mask bits are loaded.

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.

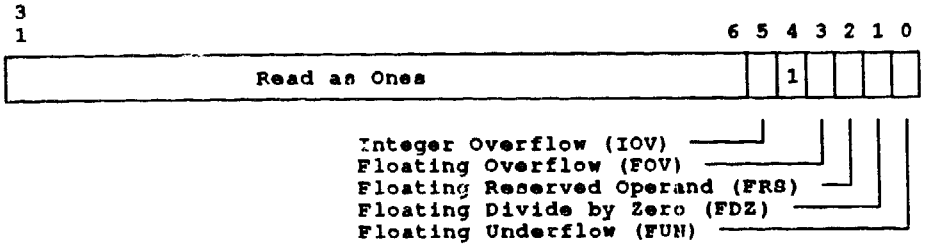
Exception Summary Register (ALU_EXC)

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)



mab-p144-90

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 from 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.

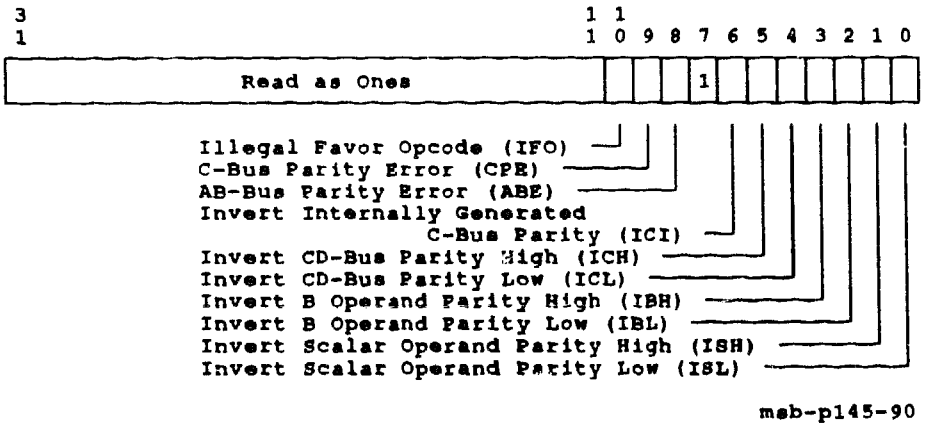
Diagnostic Control Register (ALU_DIAG_CTL)

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)

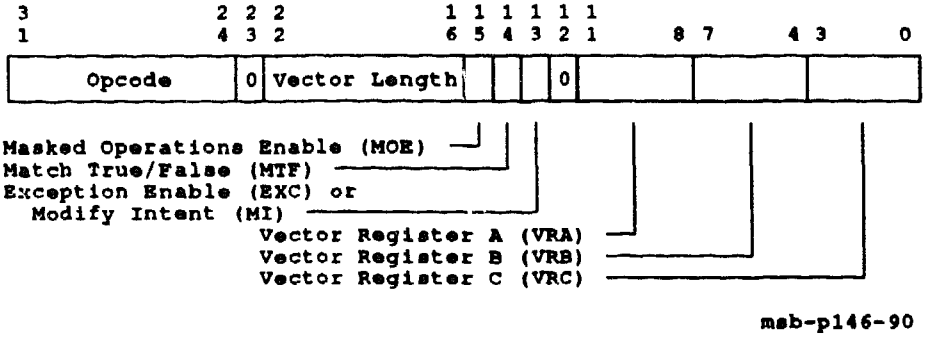
VCTL_CALU shows the arithmetic instruction being executed in the vector arithmetic pipeline. When the instruction completes, VCTL_CALU is loaded with the contents of the Deferred ALU Instruction Register.

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)

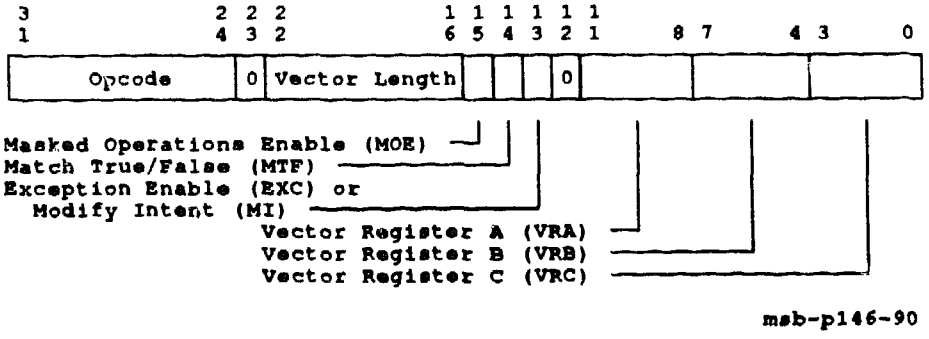
VCTL_DALU shows the arithmetic instruction to be executed next in the vector arithmetic pipeline.

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)



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 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.

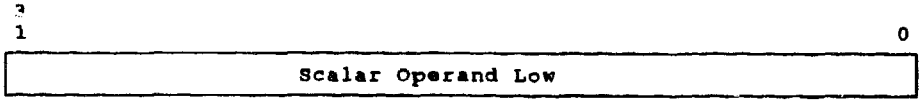
Current ALU Operand Low Register (VCTL_COP_LO)

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)



mab-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-pl48-90

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)

mab-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.

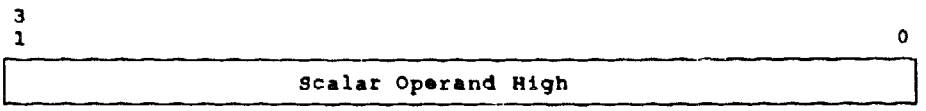
Deferred ALU Operand High Register (VCTL_DOP_HI)

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)

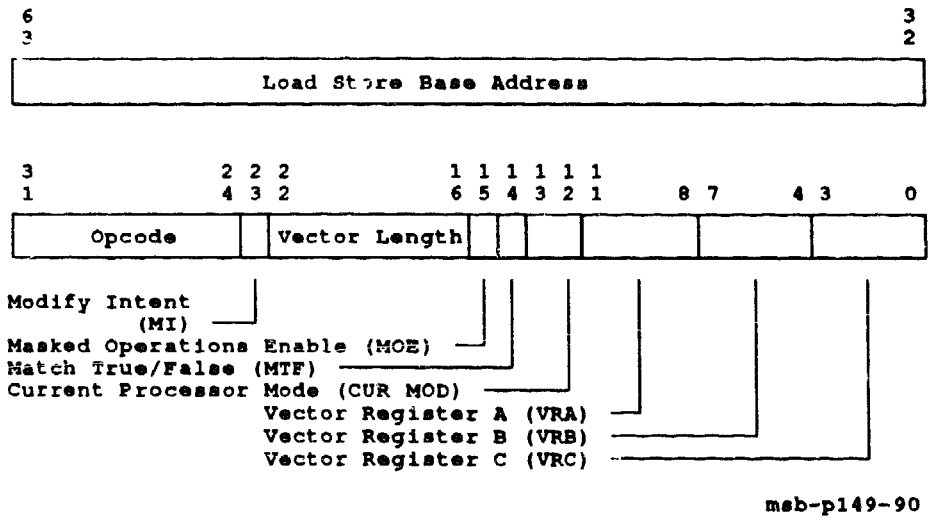
VCTL_LDST shows the load/store instruction being executed.

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)



msb-p149-90

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<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<13:12>

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.

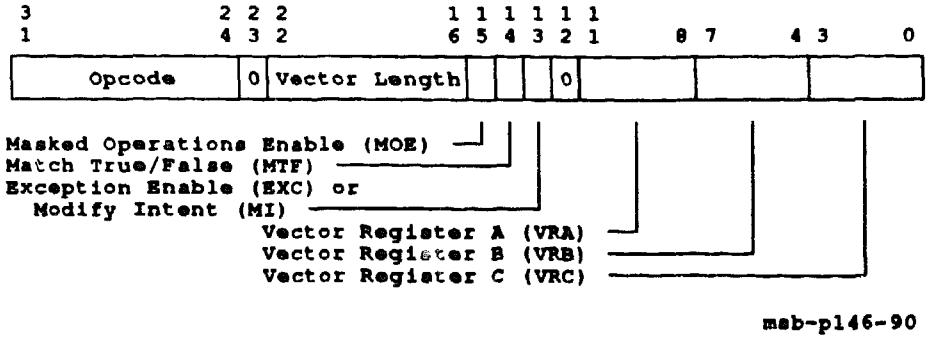
Illegal Instruction Register (VCTL_ILL)

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)



mab-pl46-90

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

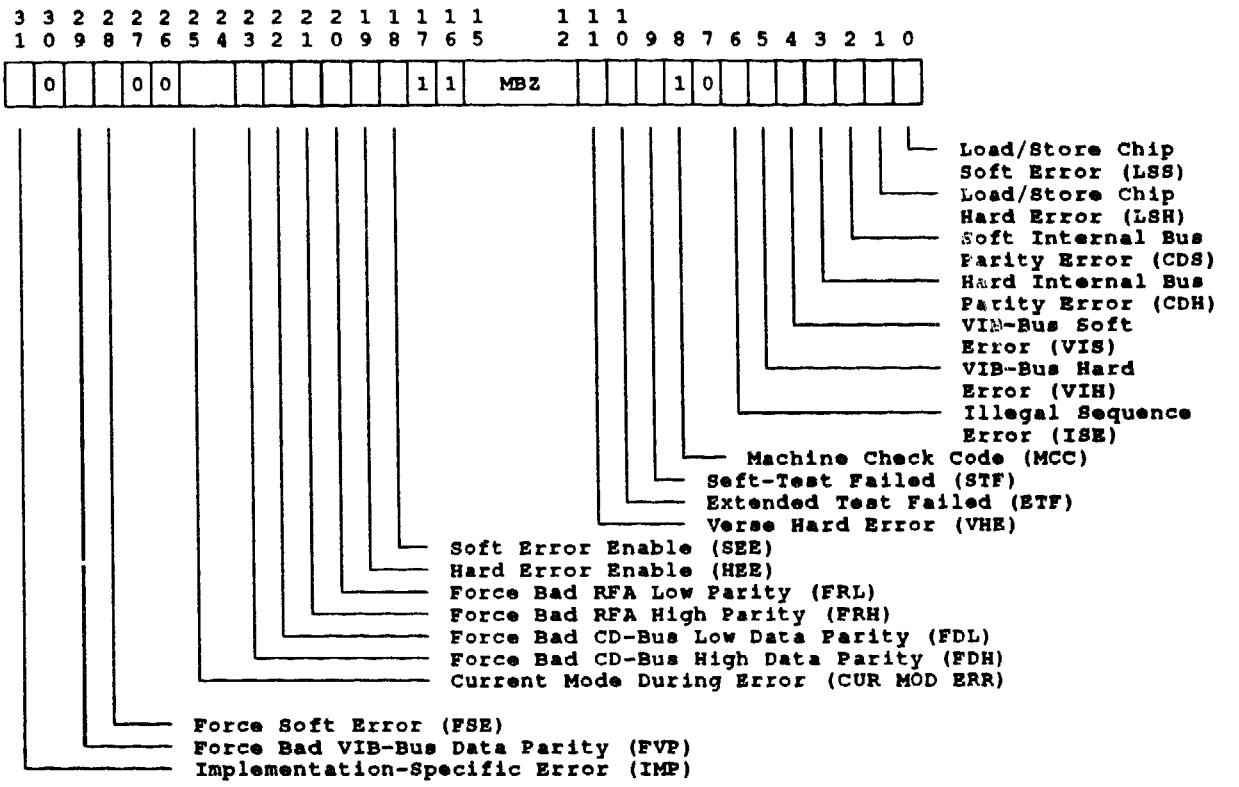
VRC is one of three vector registers. Some vector instructions use these fields to encode instruction-specific information.

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 read 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: Unrecoverable 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.

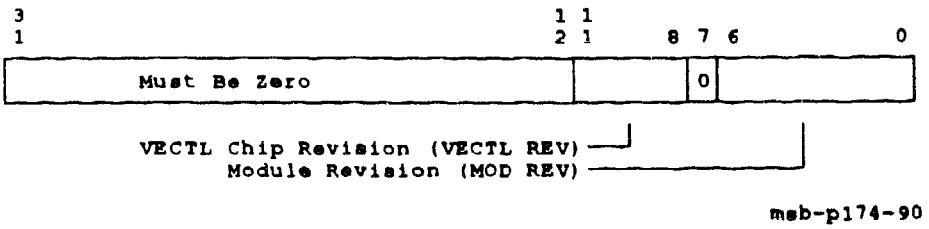
Module Revision Register (MOD_REV)

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)



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.

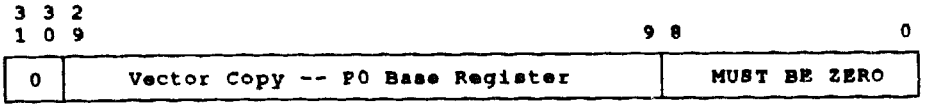
Vector Copy–P0 Base Register (LSX_P0BR)

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.

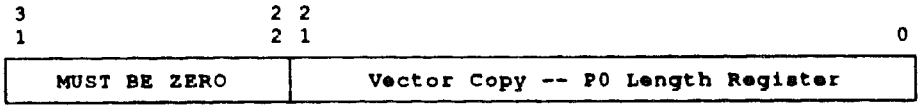
Vector Copy–P0 Length Register (LSX_P0LR)

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)

mab-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.

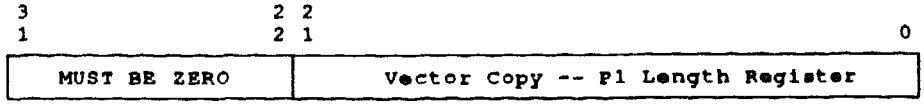
Vector Copy–P1 Length Register (LSX_P1LR)

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)*

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.

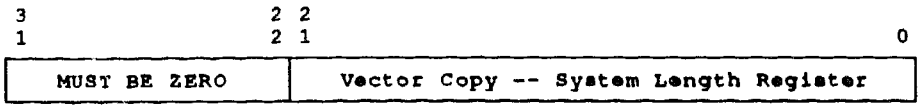
Vector Copy–System Length Register (LSX_SLR)

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)



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.

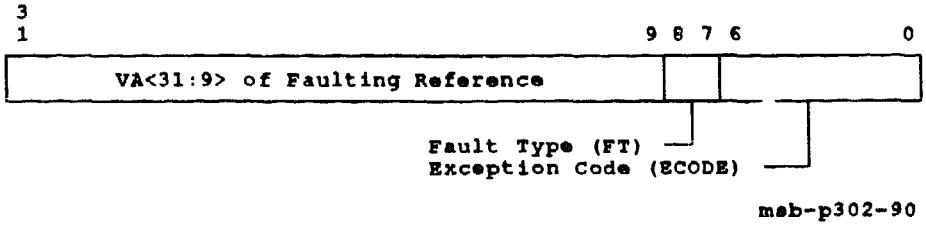
Load/Store Exception Register (LSX_EXC)

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.

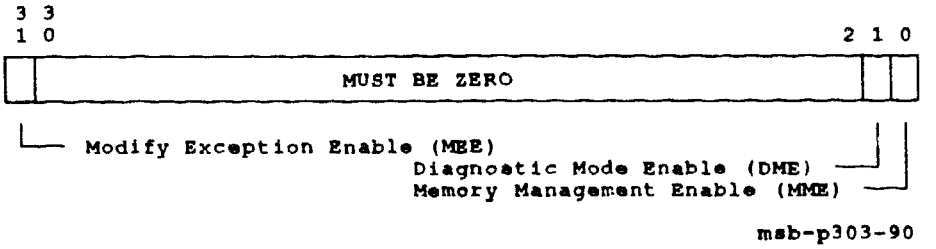
Bit	Name	Type	Cause
5	Modify Exception	Modify fault	The faulting reference attempted to write a page whose PTE<M> bit was not set. This bit indicates that the vector processor is running in virtual machine mode. To enable virtual machine mode, the MEE bit is set in TBCSR. This exception is accompanied 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

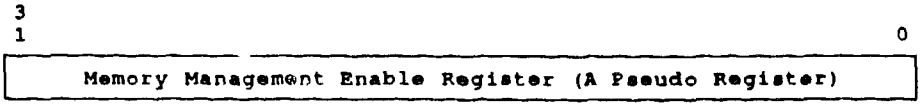
Writing a one to the MME bit enables memory management. The vector processor memory management should be disabled only for diagnostic purposes.

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.

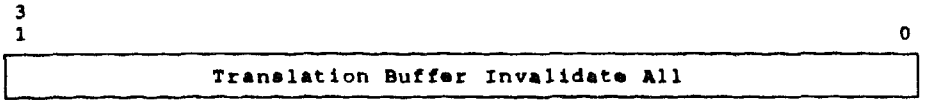
Vector Copy–Translation Buffer Invalidate All Register (LSX_TBIA)

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.

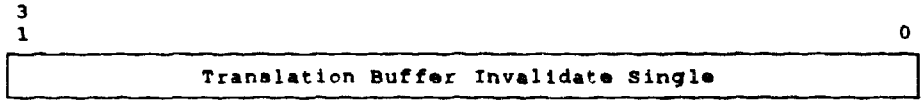
Vector Copy–Translation Buffer Invalidate Single Register (LSX_TBIS)

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)



mab-p137-90

bits<31:0>

Name: Translation Buffer Invalidate Single

Mnemonic: TBIS

Type: WO, X

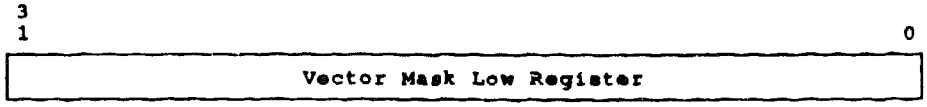
When TBIS is updated in the scalar CPU, the vector module copy is also written.

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)*



msb-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.

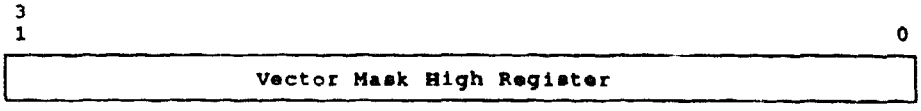
Vector Mask High Register (LSX_MASKHI)

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)*

msb-p306-90

bits<31:0>

Name: Load/Store Stride

Mnemonic: LSX_STRIDE

Type: R/W, X

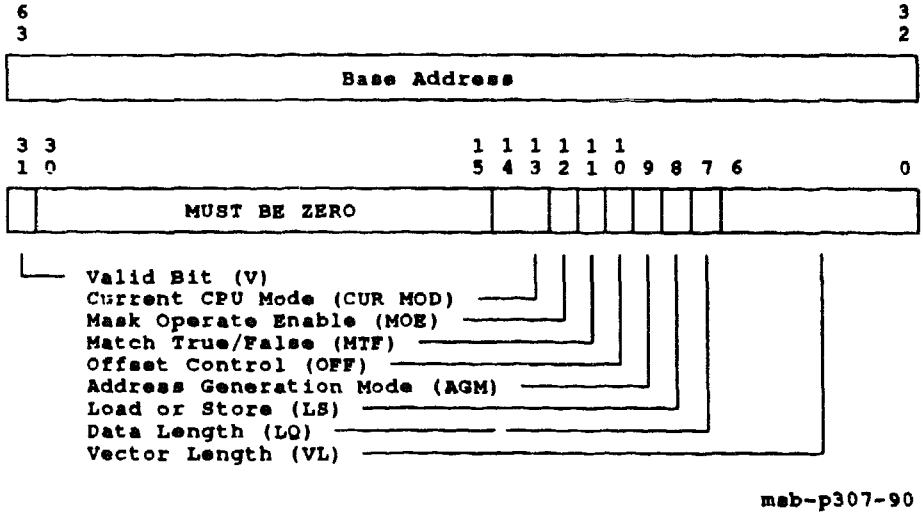
LSX_STRIDE contains the signed 32-bit stride that gives the address increments for each vector element. The stride must be loaded before the load or store instruction is issued.

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)*



mab-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 gather /scatter instruction.

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

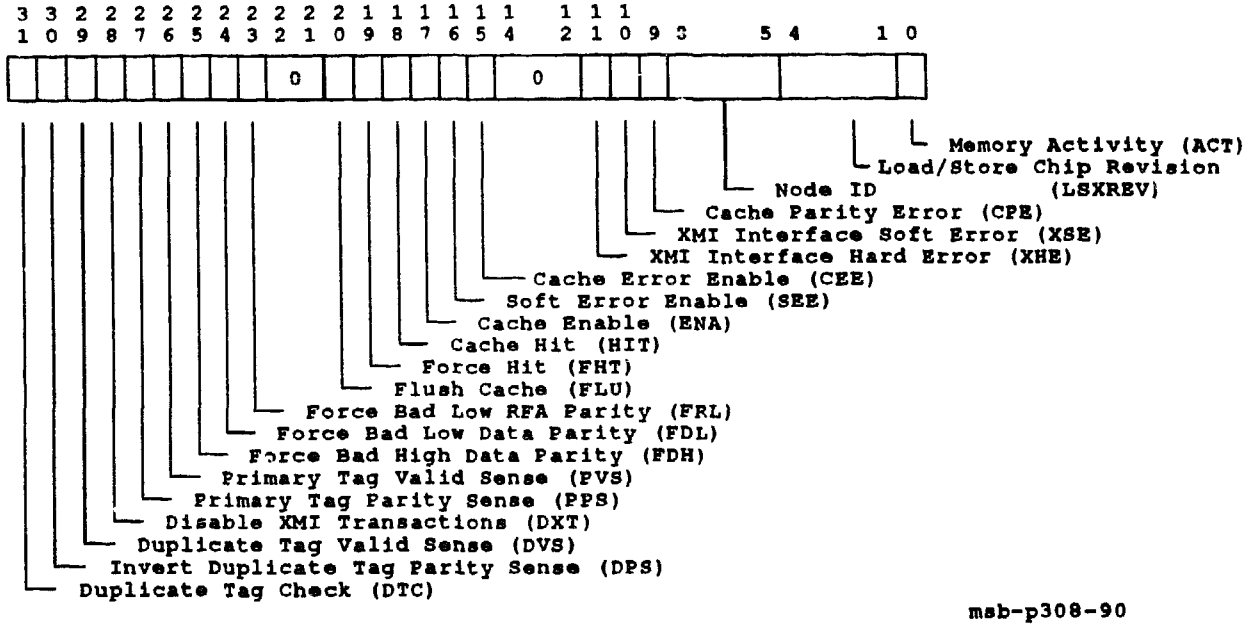
The VL field contains the contents of the Vector Length Register at the time the load/store instruction was received from the scalar processor. It can have values between 0–64. A value greater than 64 produces UNPREDICTABLE results.

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.

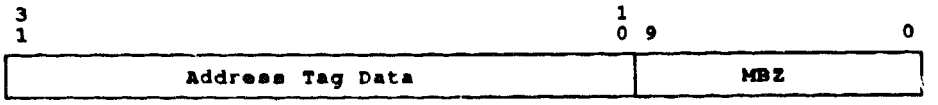
Translation Buffer Tag Register (LSX_TBTAG)

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.

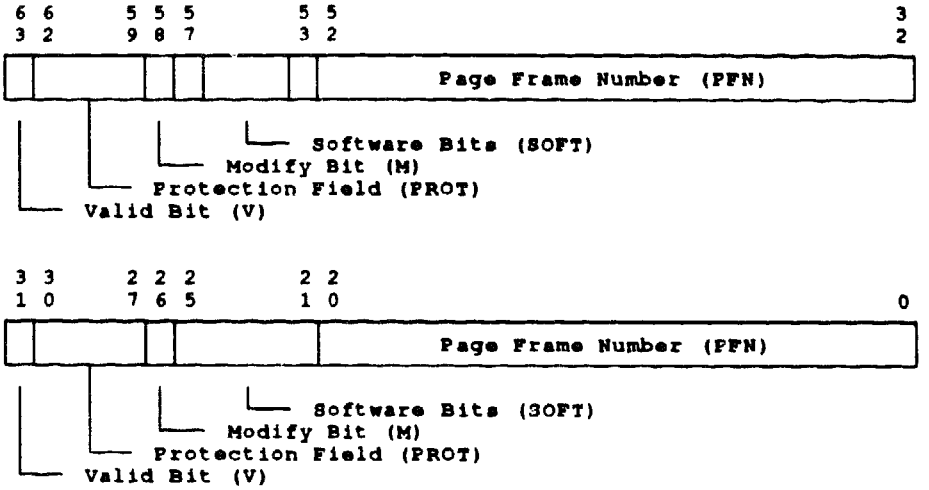
Translation Buffer PTE Register (LSX_PTE)

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).

3.9 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 3–4 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 3–5):

- 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 3–5 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

3.9.1 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.

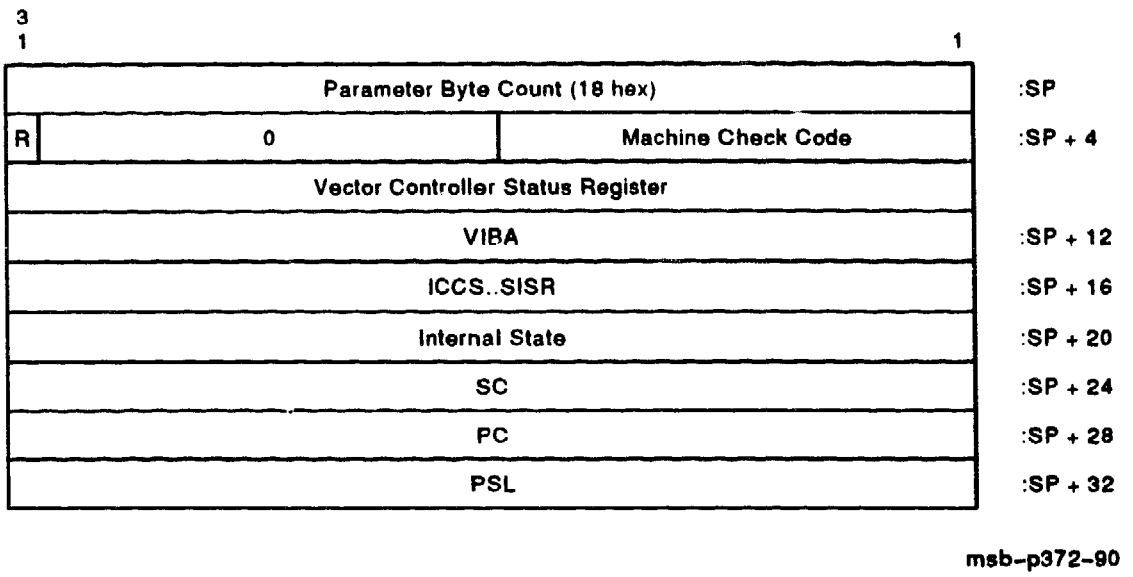
The following error conditions cause machine checks:

- 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 3–14 shows the frame pushed onto the stack by a machine check related to the vector module (MCHK_VECTOR_STATUS 14 (hex)).

Figure 3–14 Machine Check Stack Frame



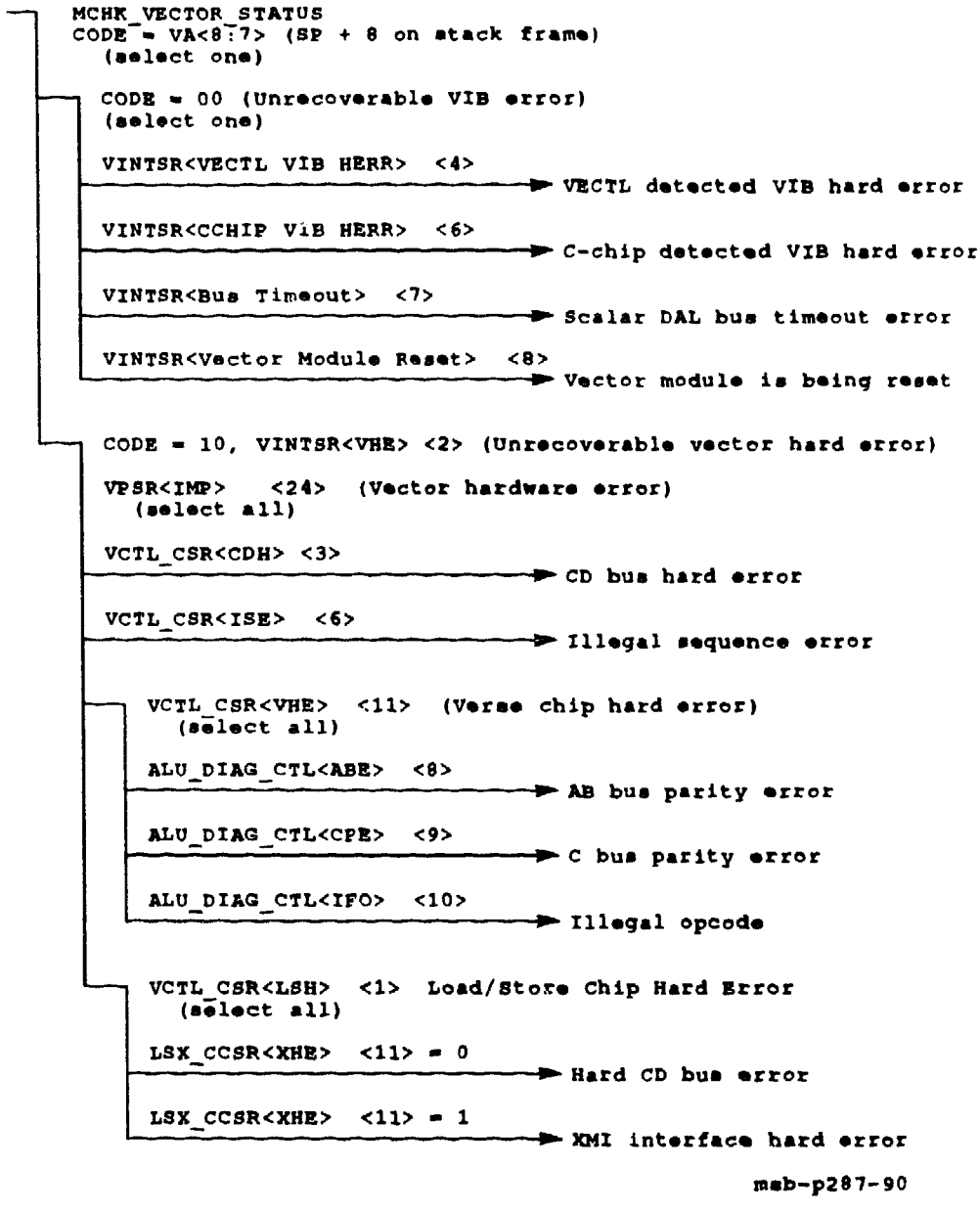
The operating system must guarantee that all errors are reported within the context of the vector processor's current process. To permit this, the vector processor generates a machine check when a hard error occurs during a read of the VMAC register.

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 3–15.

Figure 3–15 Machine Check Parse Tree



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.

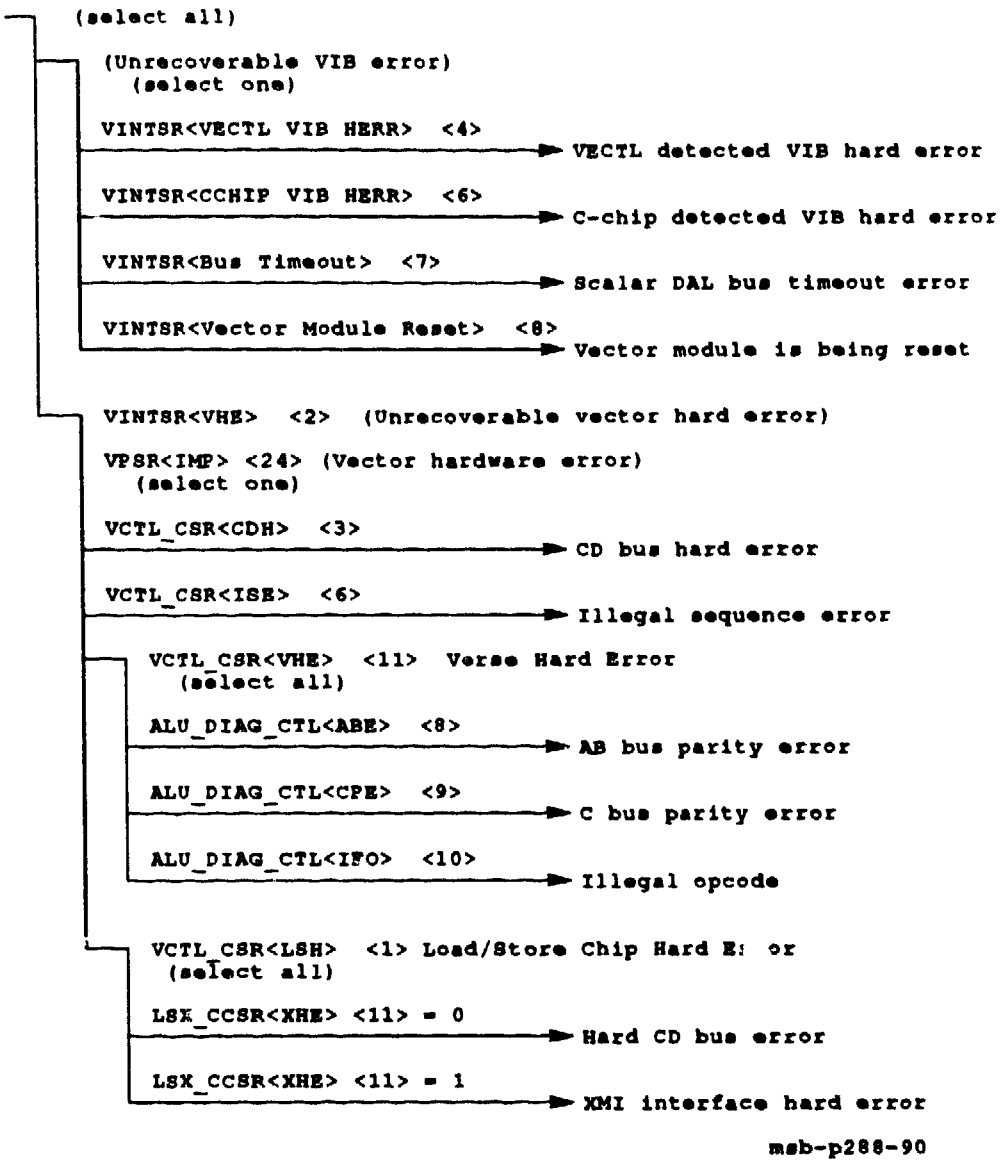
When a hard error occurs that is asynchronous to the currently executing instruction stream, a hard error interrupt is posted. Examples of this are:

- 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 3–16 shows how hard errors are parsed.

Figure 3–16 Hard Error Interrupt Parse Tree



3.9.3 Soft Error Interrupt

Soft errors are reported by soft error interrupts. Soft error interrupts are the result of errors that are successfully retried.

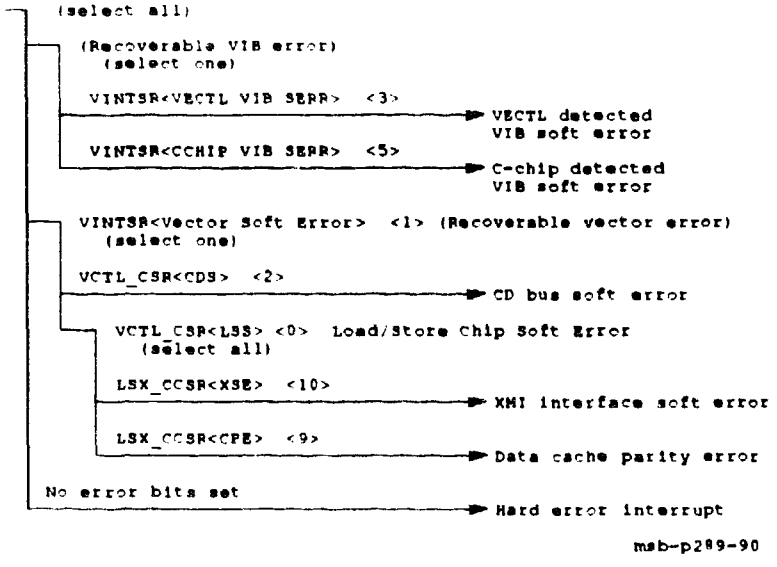
If a soft error and a hard error occur simultaneously, the soft error interrupt is posted, but the hard error interrupt takes precedence. There are many examples where the first occurrence causes a soft error interrupt, and the subsequent error during retry causes a hard error interrupt. The operating 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 3–17 shows how soft errors are parsed.

Figure 3–17 Soft Error Interrupt Parse Tree



3.9.4 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.

3.9.4.1 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 2.3.4 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.

3.9.4.2 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.

3.9.4.3 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.

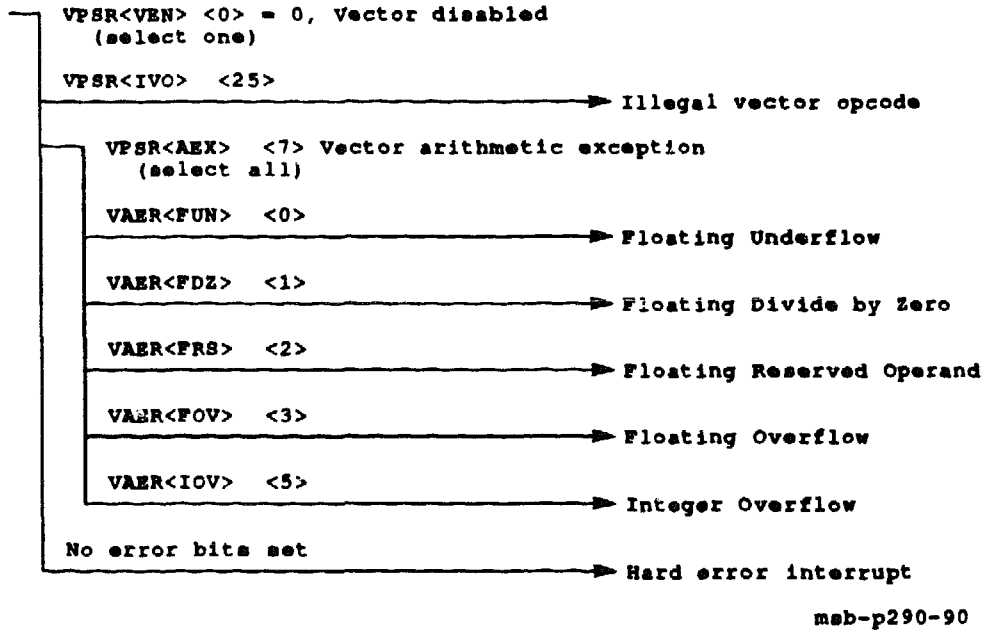
Table 3–6 shows the registers logged for an arithmetic exception.

Table 3–6 Disable Faults

Error/Exception	Registers Logged	Description
Arithmetic	VPSR VAER	An arithmetic exception caused the disable fault; generally results in image rundown.

Figure 3–18 shows how disable faults are parsed.

Figure 3–18 Disable Fault Parse Tree




CHAPTER 4

4 MS65A Memory Module

The MS65A memory module is a metal-oxide semiconductor (MOS), dynamic random access memory (DRAM), that provides up to 128 Mbytes of data storage. The memory array can be used in any VAX 6000 system and communicates over the XMI bus.

This chapter contains the following sections:

- Module Description
- Self-Test and Initialization
- Control and Status Registers
- Error Handling

4.1 Module Description

The MS65A memory module is a dynamic random access memory (DRAM) that communicates through the XMI bus to provide system memory.

The MS65A memory module consists of the following major components:

- XMI Corner
- Memory control gate array
- Memory storage array

The MS65A memory module has the following features:

- The memory module contains MOS dynamic RAM (DRAM) arrays, a CMOS memory control gate array (that contains error correction code (ECC) logic and control logic), an EEPROM storage element, and an XMI interface (the XMI) Corner.
- Storage arrays are made up of two or four banks with 155 or 299 DRAMs.
- ECC logic detects single-bit and double-bit errors and corrects single-bit errors.
- Memory self-test checks all DRAMS, the data path, and control logic on power-up.
- Quadwords, octawords, and hexwords can be read from memory.
- Quadwords, octawords, and hexwords can be written to memory.
- The memory can be configured by the system for 2-, 4-, 8-way or no interleaving.

The **XMI Corner** is the module's interface to the XMI bus and contains CMOS memory control gate arrays and interface logic. Its primary purpose is to transfer data between the MS65A memory module and the XMI bus.

The **memory control gate array** controls the MS65A memory module and transfers data between the XMI Corner and the DRAMs. The memory control gate array also controls address multiplexing, command decoding, arbitration, CSR logic functions, and the EEPROM.

The **memory storage array** within the memory control gate array includes address and control logic to modify address bits received from the XMI Corner. These modified address bits are used to control the selection of the DRAMs during reading and writing.

A block is defined as a contiguous 32-byte quantity of data, also known as a hexword. The MS65A memory module uses an indicator to represent the state of each block. This indicator is known as the *block state*. The block state is used to determine the proper read or write response on commands.

All power for the XMI memory array is supplied from the +5V rail. If the optional battery backup unit (BBU) is not installed and system power is lost, memory is lost as well.

If the optional BBU is installed, it supplies power to the system regulators in the event of a power failure, ensuring that no data is lost. The BBU supplies power to the XMI for up to 10 minutes.

4.2 Self-Test and Initialization

The MS65A memory module performs an initialization and self-test sequence on a cold power-up or when the sequence is requested by a console command. On power-up the console firmware loads the Starting Address Register (STADR) with the starting address, the Segment/Interleave Register (INTLV) with the interleave mode, and the Ending Address Register (ENADR) with the ending address.

During a cold power-up the memory control gate array 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 not modified.

Memory self-test takes about 5 seconds for a 32-Mbyte memory module, 10 seconds for a 64-Mbyte memory module, and 20 seconds for a 128-Mbyte module. 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 Chapter 6 in the *VAX 6000 Series Owner's Manual*.

4.2.1 Starting and Ending Addresses

The memory responds to starting addresses on any 16-Mbyte boundary. The ending address is also on any 16-Mbyte boundary. The ending address must be greater than the starting address, otherwise commands are not acknowledged. 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 7F FF00 0000 (16 Mbytes to 512 Gbytes in 16-Mbyte multiples). Ending addresses can range from 0 to 80 0000 0000 (0 to 512 Gbytes in 16-Mbyte multiples). Ending addresses greater than 80 0000 0000 are not permitted. The area above 80 0180 0000 is reserved for CSR addresses.

4.2.2 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 2-way, 4-way, 8-way or no interleaving at the system level. Up to eight memory array modules can be interleaved. Interleaving is done on hexword boundaries.

4.3 Control and Status Registers

The CSR names and their relative addresses are shown in Table 4-1. Detailed descriptions of the CSRs are also included in this section.

Table 4–1 MS65A Memory Module Control and Status Registers

CSR Name	Mnemonic	Address
Device Register	XDEV	BB[1] + 0000 0000
Bus Error Register	XBER	BB + 0000 0004
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 Tester Register	TCY	BB + 0000 0034
Block State ECC Error Register	BECER	BB + 0000 0038
Block State ECC Address Register	BECEA	BB + 0000 003C
Starting Address Register	STADR	BB + 0000 0050
Ending Address Register	ENADR	BB + 0000 0054
Segment/Interleave Control Register	INTLV	BB + 0000 0058
Memory Control Register 3	MCTL3	BB + 0000 005C
Memory Control Register 4	MCTL4	BB + 0000 0060
Block State Control Register	BSCTL	BB + 0000 0068
Block State Address Register	BSADR	BB + 0000 006C
EEPROM Control Register	EECTL	BB + 0000 0070
Timeout Control/Status Register	TMOER	BB + 0000 0074

[1]"BB" refers to the base address of an XMI node (E180 0000 + (node ID x 8000)).

The memory contains 19 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. Only full writes are performed. If a parity error occurs during a write, 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.

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.

RER error – A general uncorrectable multiple-bit error indicator that includes an RDS error (a hard unrecoverable 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/CW, 0 – Indicates a read conditional write 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.

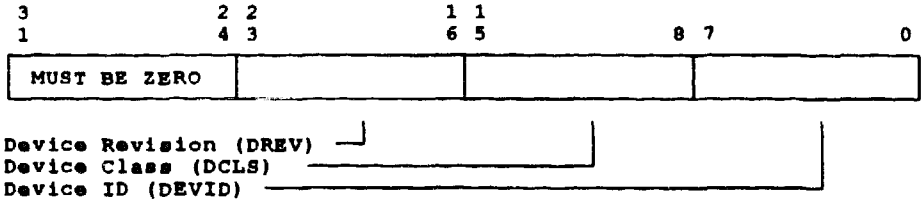
WO, 0 – Indicates a write-only register, cleared on power-up.

Device Register (XDEV)

The Device Register contains information to identify the MS65A memory module. The fields are loaded during node initialization. A zero value indicates an uninitialized node. The device class and device ID fields match those of the device type code of the MS62A memory module.

ADDRESS

Nodespace base address + 0000 0000



msb-p245-90

bits<31:24>

Name: Reserved

Mnemonic: None

Type: RO, 0

Reserved; must be zero.

bits<23:16>

Name: Device Revision

Mnemonic: DREV

Type: R/W

Identifies the revision level of the MS65A 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<15:8>

Name: Device Class

Mnemonic: DCLS

Type: RO

Identifies the type of node. The device type for an MS65A memory module is 40 (hex) and 4001(hex) for an MS62A memory module. This value is set permanently in the Device Register.

bits<7:0>

Name: Device ID

Mnemonic: DEVID

Type: R/W

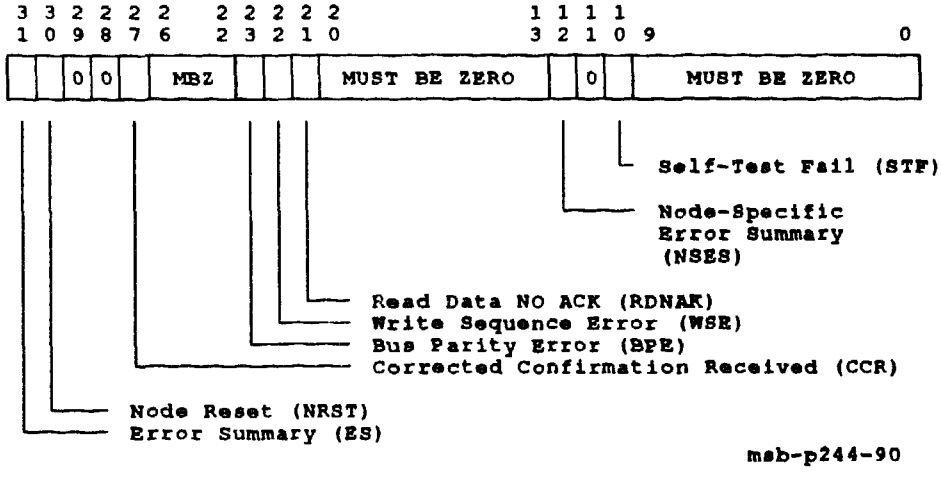
Identifies the ID of node. The ID for an MS65A memory module is 0000 0001 (binary) or 01 (hex).

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: WO, 0

When set, this bit 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 Received
Mnemonic: CCR
Type: R/W1C, 0

When set, the XMI Corner Interface (XCI) bus detected a single-bit error on any of the three XMI confirmation lines. This error is always logged, whether the MS65A memory module is responsible for the faulty bus cycle or not.

bits<26:24>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<23>

Name: Bus Parity Error
Mnemonic: BPE
Type: R/W1C, 0

When set, the memory detected a parity error on the XMI data, ID, fault, or parity lines. The MS65A does not acknowledge any cycle that contains a bus parity error. Once the cycle is not acknowledged, all other bus cycles associated with the transaction are flushed from memory. No log is kept of which parity fields had the errors. This error is always logged, whether the MS65A memory module is responsible for the faulty bus cycle or not.

bit<22>

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

When set, indicates that the memory aborted a write transaction, due to one or more missing data cycles. All command and write data entries associated with this error are flushed from queues.

bit<21>

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

When set, indicates that the memory failed to receive a response for a LOC, RER, CRD, or GRD data response cycle. Any remaining quadwords of the data packet being transmitted are sent normally.

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. A hierarchy of these registers is shown in Figure 4–1.

bit<11>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<10>

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

This bit is set when self-test starts and is cleared when self-test successfully completes.

bits<9:0>

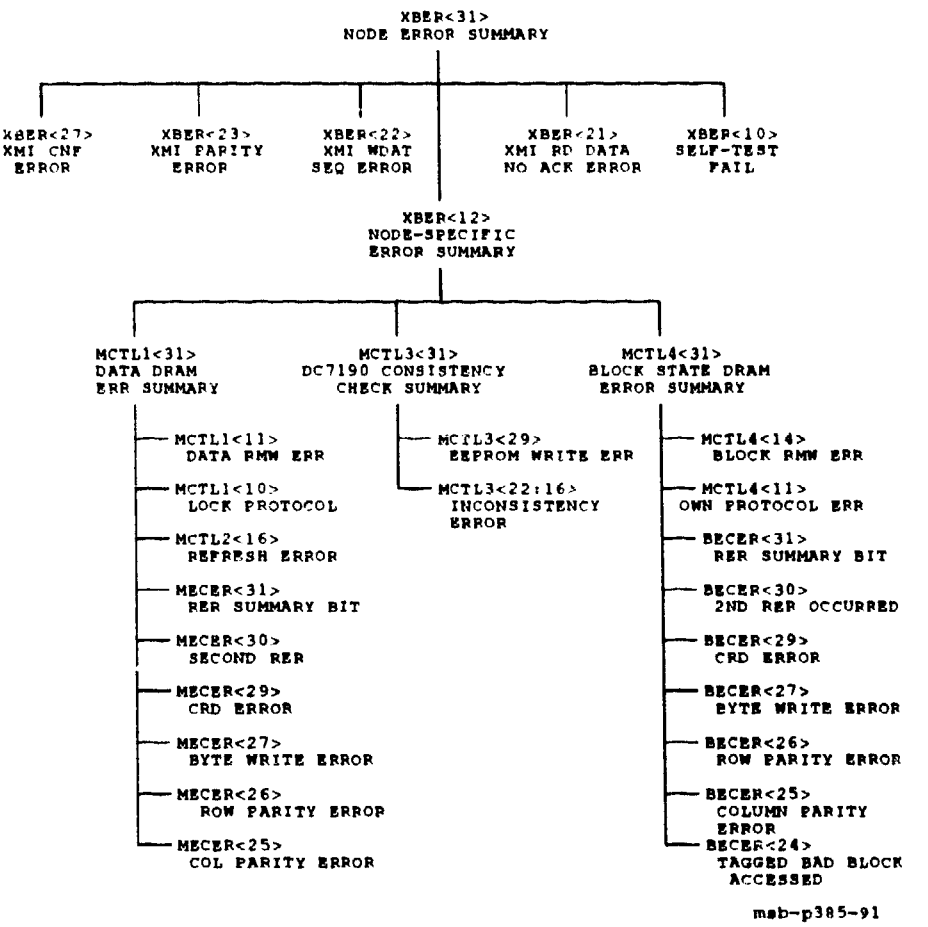
Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

Figure 4–1 Error Bit Hierarchy

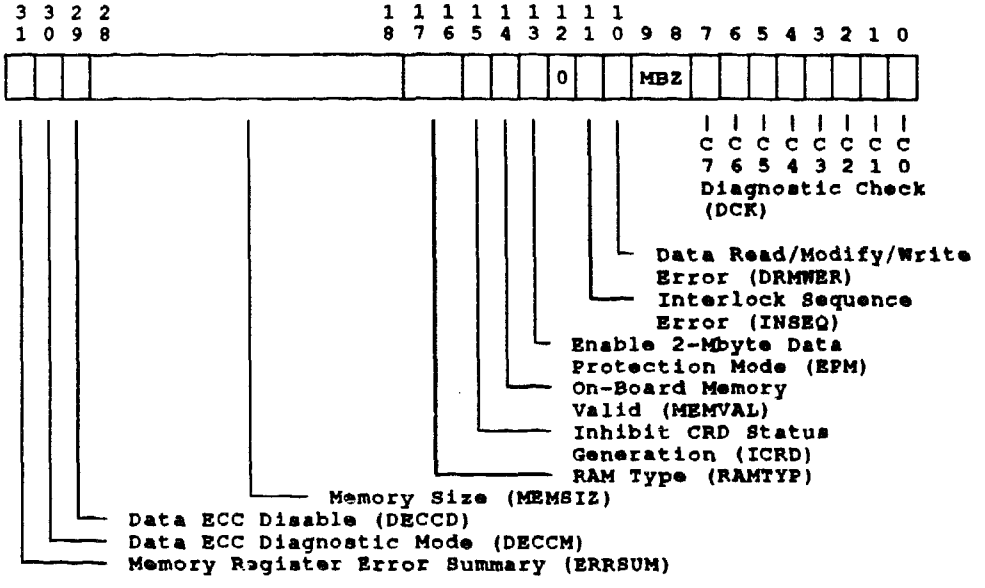


Memory Control Register 1 (MCTL1)

The Memory Control Register 1 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



msb-p231-90

bit<31>

Name: Memory Register 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: Data ECC Diagnostic Mode

Mnemonic: DECCM

Type: R/W, 0

This bit is used for diagnostic purposes.

bit<29>

Name: Data ECC Disable
Mnemonic: DECCD
Type: R/W, 0

This bit is used for diagnostic purposes.

bits<28:18>

Name: Memory Size
Mnemonic: MEMSIZ
Type: R/W

These bits specify the memory module size in 256-Kbyte increments. An upper extension of these bits is located in the MCTL4 Memory Size <28:18> field.

bits<17:16>

Name: RAM Type
Mnemonic: RAMTYP
Type: R/W

These bits show the DRAM type used. They are used in conjunction with <28:18> and Memory Control Register 4 (MCTL4) <17:16>.

bit<15>

Name: Inhibit CRD Status Generation
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. CRD errors are still logged, and RER errors are logged and reported.

bit<14>

Name: On-Board 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 2-Mbyte Protection Mode
Mnemonic: EPM
Type: R/W, 0

When set, the operation of the MCTL1 ECC Diagnostic<30> MCTL1 ECC Disable <29>, MCTL4 Block State Diagnostic Mode <30>, and MCTL4 Block State ECC Disable <29> bits are inhibited in the first 2 Mbytes of memory space.

bit<12>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<11>

Name: Interlock Sequence Error
Mnemonic: INSEQ
Type: R/W1C, 0

When set, this bit indicates that a system protocol violation has been observed around an executed UWMASK to the on-board memory.

bit<10>

Name: Data Read Modify Write Error
Mnemonic: DRMWER
Type: R/W1C, 0

When set, an uncorrectable error was detected in the data DRAM field during a Read Modify Write cycle.

bits<9:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<7:0>

Name: Diagnostic Check
Mnemonic: DCK
Type: R/W, 0

These bits are used in ECC diagnostic mode as substitute check bits.

Memory ECC Error Register (MECER)

The Memory ECC Error Register logs ECC error status and syndrome information for the data DRAMs. The MECER logs ECC error information during read cycles only. If an RER error occurs during a Write Mask cycle, DRMWER bit<10> in the MCTL1 Register is set. Errors associated with block state DRAMs are logged in MCTL4.

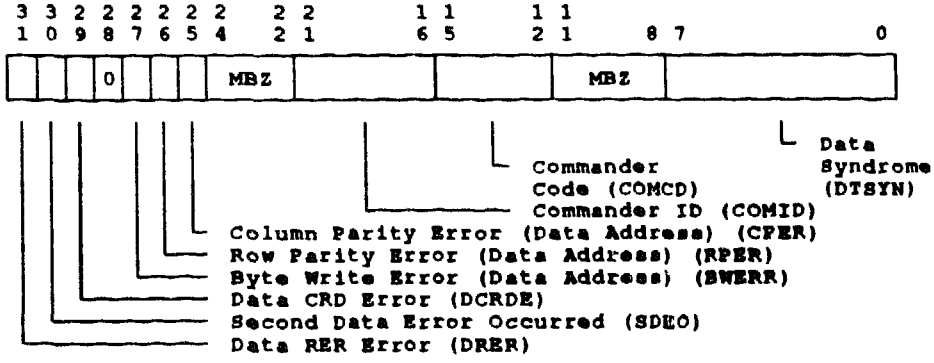
This register logs ECC error type and error syndrome information when correctable and uncorrectable errors occur during Read transactions. During a Write Mask transaction, only DRMWER bit<10> in the MCTL1 Register logs the fact that the ECC error occurred.

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



mab-p236-90

bit<31>

Name: Data RER Error

Mnemonic: DRER

Type: R/W1C, 0

When set, an uncorrectable error occurred during a read transaction.

bit<30>

Name: Second Data Error Occurred
Mnemonic: SDEO
Type: R/W1C, 0

When set, an RER or CRD error occurred before the previous one was cleared from the register. The error information is logged only if the second error is an RER type error and the first error was a CRD. A second CRD error after the initial CRD error will set this bit.

bit<29>

Name: Data CRD Error
Mnemonic: DCRDE
Type: R/W1C, 0

When set, 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 (Data Address)
Mnemonic: BWERR
Type: RO, 0

When set, an 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. The setting of this bit also sets MECER<31>.

bit<26>

Name: Row Parity Error (Data Address)
Mnemonic: RPER
Type: RO, 0

When set, an RER error occurs due to a row address parity error. Cleared when MECER<31> is cleared. The setting of this bit also sets MECER<31>.

bit<25>

Name: Column Parity Error (Data Address)
Mnemonic: CPER
Type: RO, 0

When set, an RER error occurs due to a column address parity error. Cleared when MECER<31> is cleared. The setting of this bit also sets MECER<31>.

bits<24:21>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<21:16>

Name: Commander ID
Mnemonic: COMID
Type: RO, 0

This field logs the 6-bit commander ID (ID <5:0>) that was involved in the transaction that caused the logging of MECER errors. A CRD response is sent back to the commander and the CRD occurrence is logged in the MECER. The MECER error log shows which CPU or I/O module encountered the error. These bits are not cleared when the error status bits are reset by a system commander. Therefore, they hold the ID of the last MECER error logged (or a zero if no errors occurred since the last cold start).

bits<15:12>

Name: Commander Code
Mnemonic: COMCD
Type: RO, 0

This field logs the 4-bit command code associated with the logged failure. The commander code identifies the command type associated with the error that occurred. Knowing the type of command helps the user to determine why the memory error occurred. These bits are not cleared when the error status bits are reset. Therefore, they hold the command code of the last MECER error logged (or a zero if no errors occurred since the last cold start).

bits<7:0>

Name: Data Syndrome

Mnemonic: DTSYN

Type: RO, 0

These bits are the syndrome bits of the location in an RER or CRD error, when the memory module is not in diagnostic mode.

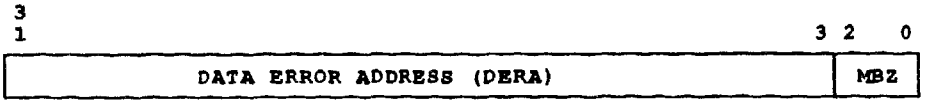
Memory ECC Error Address Register (MECEA)

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*



msb-p235-90

bits<31:3>

Name: Data Error Address

Mnemonic: DERA

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

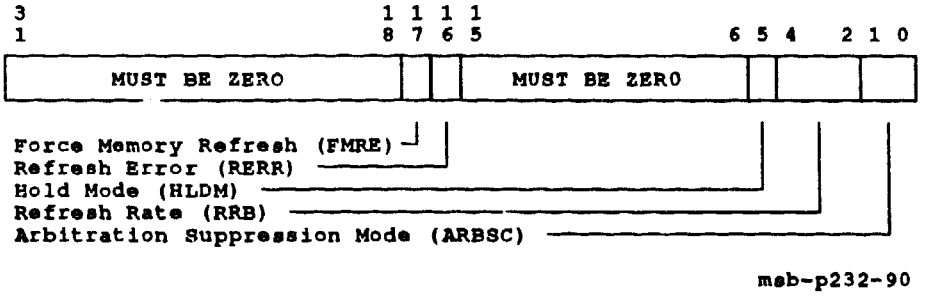
Reserved; must be zero.

Memory Control Register 2 (MCTL2)

The second memory control register contains additional control and error status information.

ADDRESS

Nodespace base address + 0000 0030



bits<31:18>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bit<17>

Name: Force Memory Refresh

Mnemonic: FMRE

Type: WO, 0

When set by a commander, the gate array initiates a single memory refresh operation. This causes a refresh cycle to be generated for all on-board DRAMs.

bit<16>

Name: Refresh Error

Mnemonic: RERR

Type: R/W1C, 0

When set, a second refresh request was asserted before the first one was implemented, meaning that a refresh was missed.

bits<15:6>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<5>

Name: Hold Mode
Mnemonic: HLDM
Type: RO, 0

This bit informs the gate array which hold mode is needed for read-data response transfers.

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 Mode
Mnemonic: ARBSC
Type: R/W, 0

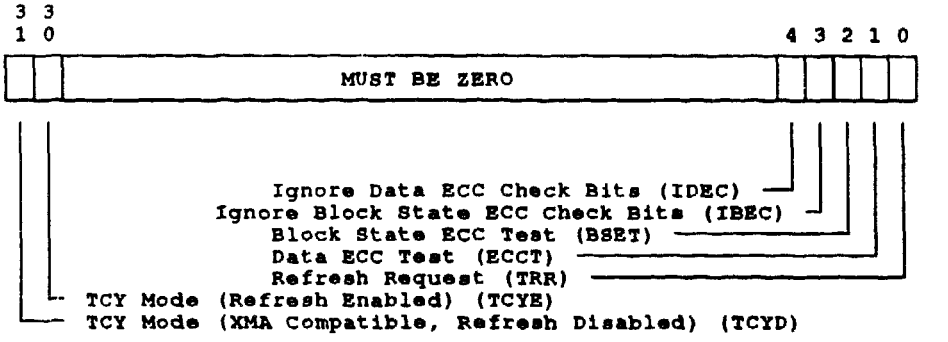
These field controls the Arbitration Supression mode.

TCY Tester Register (TCY)

The TCY Tester Register contains control bits to implement manufacturing tests.

ADDRESS

Nodespace base address + 0000 0034



msb-p242-90

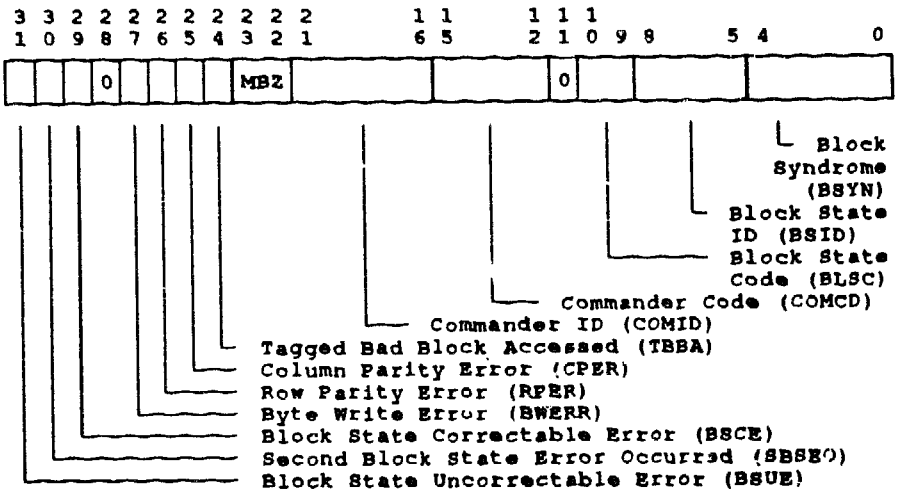
Block State ECC Error Register (BECER)

The Block State ECC Error Register is used to log ECC error status and syndrome information, as well as tagged bad accesses, for the block state DRAMs. This register logs errors during self-test and normal operation. Diagnostic information is provided if self-test fails and Bus Error Register (XBER) bit <10> is set.

Bits in this register are used for logging block state errors, byte write errors, parity bit errors, commander IDs, commander codes, block state codes, block state IDs, and block syndrome bits.

ADDRESS

Nodespace base address + 0000 0038



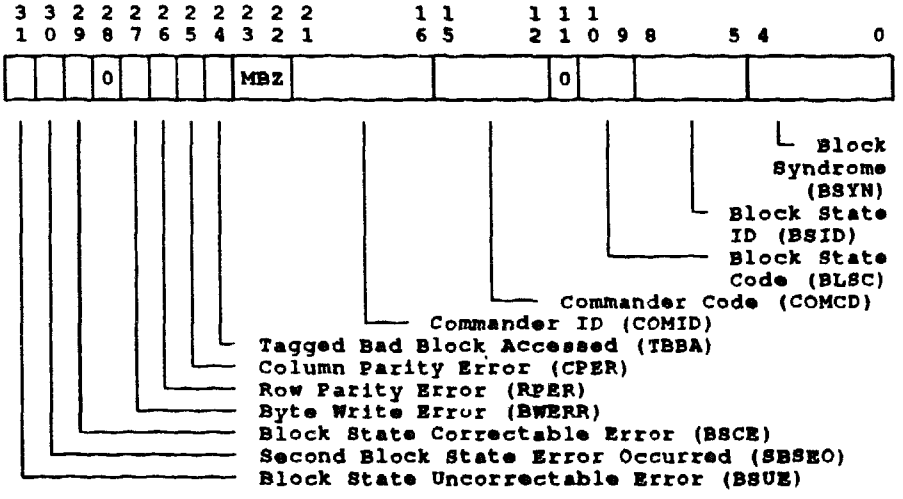
mab-p224-90

Block State ECC Error Register (BECER)

The Block State ECC Error Register is used to log ECC error status and syndrome information, as well as tagged bad accesses, for the block state DRAMs. This register logs errors during self-test and normal operation. Diagnostic information is provided if self-test fails and Bus Error Register (XBER) bit <10> is set.

Bits in this register are used for logging block state errors, byte write errors, parity bit errors, commander IDs, commander codes, block state codes, block state IDs, and block syndrome bits.

ADDRESS *Nodespace base address + 0000 0038*

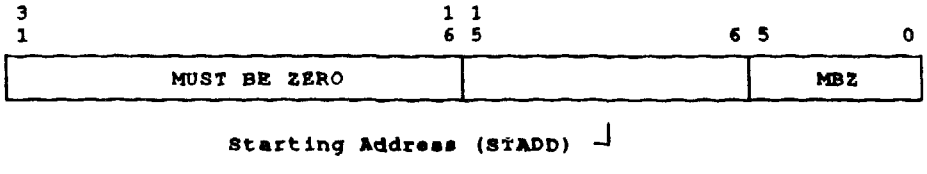


mab-p224-90

Starting Address Register (STADR)

The Starting Address Register configures the starting memory address of the MS65A memory module with a total memory address space less than or equal to 512 Mbytes. It is used to place an MS65A memory module above the 512-Mbyte limit in the address space. It can also be used for configuring systems with less than 512 Mbytes of total memory.

ADDRESS *Nodespace base address + 0000 0050*



mab-p241-90

bits<31:16>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<15:6>

Name: Starting Address
Mnemonic: STADD
Type: R/W, 0

This field contains the 10-bit memory starting address.

bits<5:0>

Name: Reserved
Mnemonic: None
Type: RO

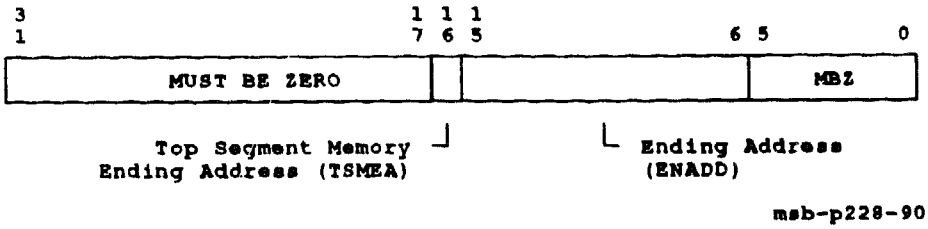
Reserved; must be zero.

Ending Address Register (ENADR)

The Ending Address Register configures the ending memory address of an MS65A memory module with a total memory address space less than or equal to 512 Mbytes. It is also used to place an MS65A memory module above the 512-Mbyte limit in the address space. MS65A memory modules use this register along with the Starting Address Register (STADR) and the Segment/Interleave Register (INTLV) to configure memory addresses.

ADDRESS

Nodespace base address + 0000 0054



bits<31:17>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bit<16>

Name: Top Segment Memory Ending Address

Mnemonic: TSMEA

Type: R/W, 0

This bit configures the ending address as the top byte of the 16-Gbyte memory address segment.

bits<15:6>

Name: Ending Address
Mnemonic: ENADD
Type: R/W, 0

This field contains the 10-bit memory ending address.

bits<5:0>

Name: Reserved
Mnemonic: None
Type: RO

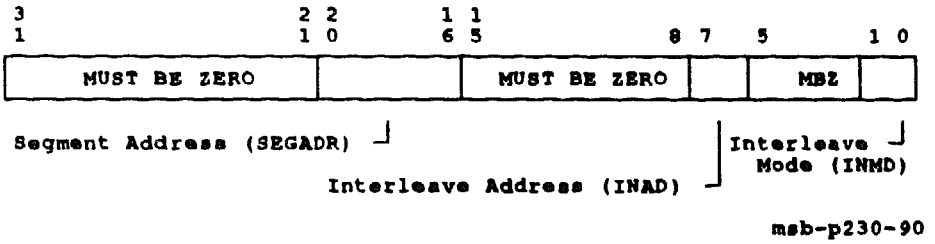
Reserved; must be zero.

Segment/Interleave Register (INTLV)

The Segment/Interleave Register gives memory interleave information. This register is used to establish the interleave mode and allows the system to program in a non-zero segment value. It sets up an optional segment comparison address for the upper bits of the memory bus address.

ADDRESS

Nodespace base address + 0000 0058



bits<31:21>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bits<20:16>

Name: Segment Address

Mnemonic: SEGADR

Type: R/W, 0

This field is optionally configured to place memory address space at a 16-Gbyte offset in the memory bus address space. Reading these bits tells the user the address of the failure. A zero in the field leaves the memory segment based in the lowest 16-Gbyte memory address space.

bits<15:8>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bits<7:5>

Name: Interleave Address

Mnemonic: INAD

Type: R/W, 0

This field contains the address used for interleaving. The interleaving address determines the address in which the memory module will respond.

bits<4:2>

Name: Reserved

Mnemonic: None

Type: RO

Reserved; must be zero.

bits<1:0>

Name: Interleave Mode

Mnemonic: INMD

Type: R/W, 0

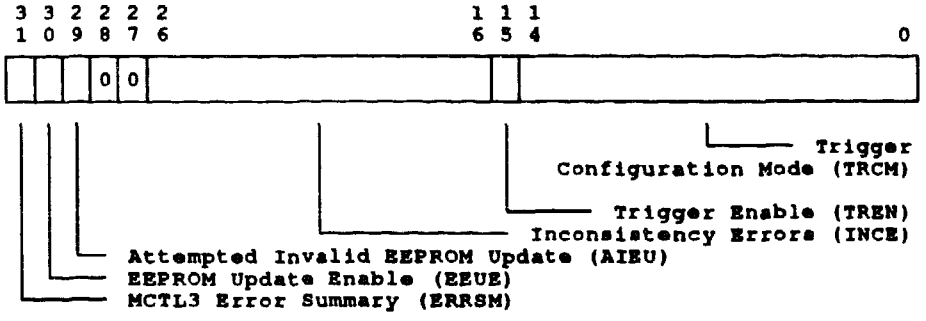
This field shows how many ways the module is interleaved (none, 1-way, 2-way, 4-way, or 8-way interleaving). The Interleave Mode bits are also used to determine the address in which the memory module will respond.

Memory Control Register 3 (MCTL3)

The third memory control register contains control and error status information.

ADDRESS

Nodespace base address + 0000 005C



mab-p233-90

bit<31>

Name: MCTL3 Error Summary

Mnemonic: ERRSM

Type: RO, 0

This bit represents the error summary status for Memory Control Register 3. When clear, all contributing error bits have been reset.

bit<30>

Name: EEPROM Update Enable

Mnemonic: EEUE

Type: R/W, 0

This bit enables EEPROM updates. When it is set, selected locations of the EEPROM can be updated.

bit<29>

Name: Attempted Invalid EEPROM Update

Mnemonic: AIEU

Type: R/W1C, 0

This status bit is set if an invalid EEPROM update was attempted.

bits<28:23>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<22:16>

Name: Inconsistency Errors
Mnemonic: INCE
Type: R/W1C, 0

This field is used to report internal consistency errors. Set bits usually indicate that the gate array has a fault.

bit<15>

Name: Trigger Enable
Mnemonic: TREN
Type: R/W, 0

This bit controls trigger enabling. It is dependent on the state of TRCM bit <14>.

bits<14:0>

Name: Trigger Configuration Mode
Mnemonic: TRCM
Type: R/W

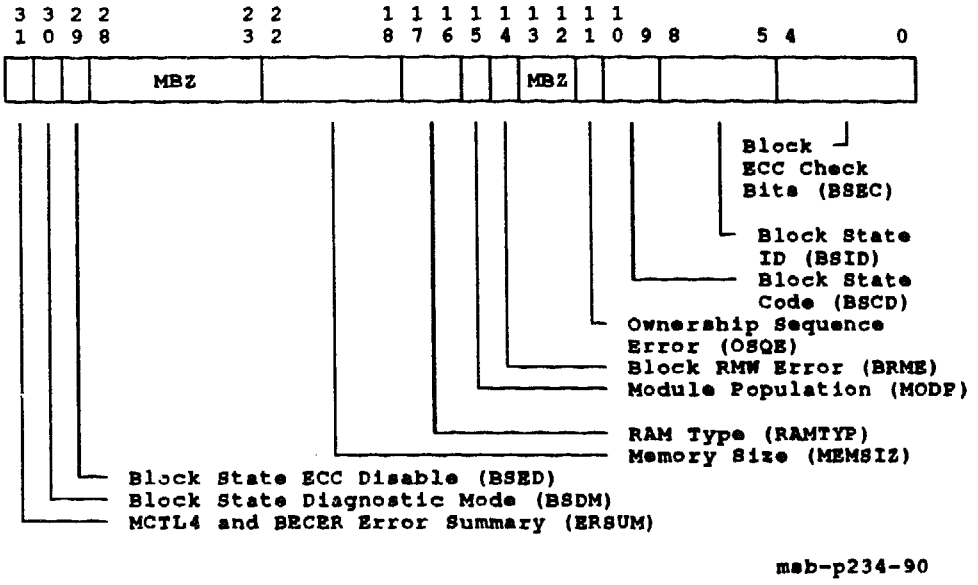
This field is used to enable or disable trigger generation.

Memory Control Register 4 (MCTL4)

The fourth memory control register contains additional control and error status information.

ADDRESS

Nodespace base address + 0000 0060



mab-p234-90

bit<31>

Name: MCTL4 Error Summary

Mnemonic: ERSUM

Type: RO, 0

This bit represents the error summary bit for Memory Control Register 4 and the Block State ECC Error Register. When clear, all contributing error bits have been reset.

bit<30>

Name: Block State Diagnostic Mode

Mnemonic: BSDM

Type: R/W, 0

This bit is used for diagnostic purposes.

bit<29>

Name: Block State ECC Disable
Mnemonic: BSED
Type: R/W, 0

This bit is used for diagnostic purposes.

bits<28:23>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<22:18>

Name: Memory Size
Mnemonic: MEMSIZ
Type: R/W, 0

This field provides an upper extension to the MCTL1 Memory Size <28:18> field.

bits<17:16>

Name: RAM Type
Mnemonic: RAMTYP
Type: R/W, 0

This field shows the type of DRAM used.

bit<15>

Name: Module Population
Mnemonic: MODP
Type: R/W, 0

This bit shows the DRAM population status of the MS65A memory module.

bit<14>

Name: Block Read Modify Write Error
Mnemonic: BRME
Type: R/W1C, 0

This bit is set if an error is found on the block state DRAM field.

bits<13:12>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bit<11>

Name: Ownership Sequence Error
Mnemonic: OSQE
Type: R/W 1C

This bit is set when a bus protocol violation has occurred.

bits<10:9>

Name: Block State Code
Mnemonic: BSCD
Type: R/W, 0

This bit is used for diagnostic purposes.

bits<8:5>

Name: Block State ID
Mnemonic: BSID
Type: R/W, 0

This field is used for diagnostic purposes.

bits<4:0>

Name: Block ECC Check Bits
Mnemonic: BSEC
Type: R/W, 0

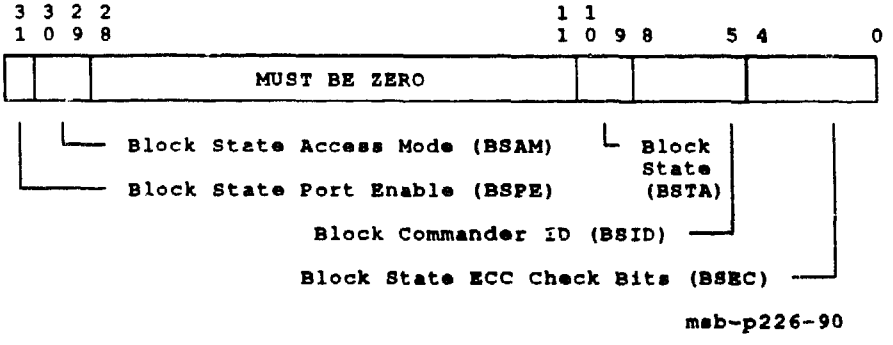
This field is used for diagnostic purposes.

Block State Control Register (BSCTL)

The Block State Control Register is used to access a block state DRAM field directly in diagnostic or error recovery modes of operation. This register is used with the Block State Address Register (BSADR).

ADDRESS

Nodespace base address + 0000 0068



bit<31>

Name: Block State Port Enable
Mnemonic: BSPE
Type: R/CW, 0

bits<30:29>

Name: Block State Access Mode
Mnemonic: BSAM
Type: R/CW, 0

bits<28:11>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<10:9>

Name: Block State
Mnemonic: BSTA
Type: RO, 0

bits<8:5>

Name: Block Commander ID
Mnemonic: BCID
Type: RO, 0

bits<4:0>

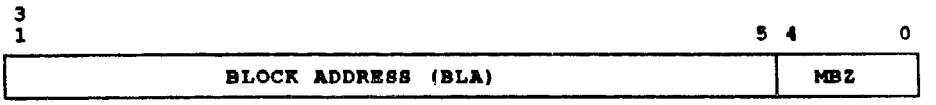
Name: Block State ECC Check Bits
Mnemonic: BSECB
Type: RO, 0

Block State Address Register (BSADR)

The Block State Address Register is used to set up a block address for commander access to the block state DRAM field. This register is used with the Block State Control Register (BSCTL).

ADDRESS

Nodespace base address + 0000 006C



mab-p225-90

bits<31:5>

Name: Block Address
Mnemonic: BLA
Type: R/W, 0

These bits contain the block address.

bits<4:0>

Name: Reserved
Mnemonic: None
Type: RO

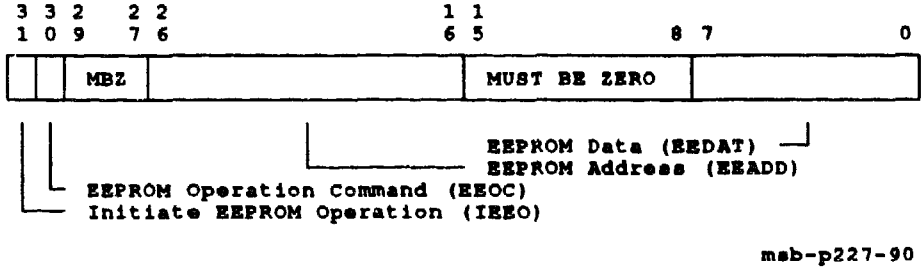
Reserved; must be zero.

EEPROM Control Register (EECTL)

The EEPROM Control Register is used to control EEPROM access modes. This register holds the address and data associated with EEPROM operations. It also contains the read/write operation initiation control bits. EEPROM update enable and status bits are located in Memory Control Register 3 (MCTL3). This register is used for diagnostic purposes only.

ADDRESS

Nodespace base address + 0000 0070



bit<31>

Name: Initiate EEPROM Operation

Mnemonic: IEEO

Type: R/CW, 0

This bit is used to initiate an EEPROM operation.

bit<30>

Name: EEPROM Operation Command

Mnemonic: EEOC

Type: R/W1C, 0

This bit is used to specify an EEPROM operation.

bits<29:27>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<26:16>

Name: EEPROM Address
Mnemonic: EEADD
Type: R/W1C, 0

This field contains the EEPROM address.

bits<15:8>

Name: Reserved
Mnemonic: None
Type: RO

Reserved; must be zero.

bits<7:0>

Name: EEPROM Data
Mnemonic: EEDAT
Type: R/CW, 0

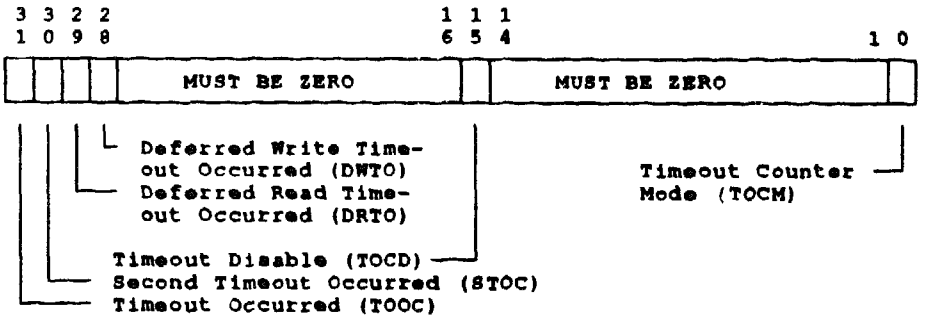
This field contains the EEPROM data.

Timeout Control/Status Register (TMOER)

The Timeout Control/Status Register is used with timeout counters and to log the status of timed out commands.

ADDRESS

Nodespace base address + 0000 0074



mab-p243-90

4.4 Error Handling

The memory module performs single-bit correction and double-bit detection on the data stored. Two error schemes are used: one for the data DRAMs and another for the block state DRAMs.

Memory logic detects and corrects single-bit DRAM errors. Double-bit errors are detected, but not corrected. All ones or all zeros are considered to be uncorrectable errors. For more information, refer to the Bus Error Register (XBER) bit descriptions.

Block state DRAM ECC errors are handled in the following way. Each 6-bit block state/commander ID location of the block state field has a 5-bit ECC check field. Each block state access operation uses an 11-bit pattern that is transferred between the memory control gate array and the block state DRAM array. The ECC check bit pattern is a function of the 6-bit block state commander ID pattern and the hexword block address.


INDEX

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X