VAX 6000 Models 300 and 400 Service Manual

Order Number EK–624EA–MG–001

This manual is intended for Digital customer service engineers. It covers the <REFERENCE>(xyp) and <REFERENCE>(xrp) CPU options, the <REFERENCE>(xma2) memory, the <REFERENCE>(xrv) vector option, and the <REFERENCE>(xbi_plus) I/O adapter. This manual is to be used with the *VAX 6000 Platform Service Manual*.

digital equipment corporation
maynard, massachusetts

First Printing, December 1990

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software, if any, described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license. No responsibility is assumed for the use or reliability of software or equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Copyright ©1990 by Digital Equipment Corporation.

All Rights Reserved.
Printed in U.S.A.

The following are trademarks of Digital Equipment Corporation:

DEMNA	PDP	VAXcluster
DEC	ULTRIX	VAXELN
DEC LANcontroller	UNIBUS	VMS
DECnet	VAX	XMI
DECUS	VAXBI	digital™

FCC NOTICE: The equipment described in this manual generates, uses, and may emit radio frequency energy. The equipment has been type tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference, in which case the user at his own expense may be required to take measures to correct the interference.

Contents

Chapter 3 <REFERENCE>(xyp) Scalar Processor

Chapter 4 <REFERENCE>(XRP) Scalar Processor

Chapter 5 <REFERENCE>(XRV) Vector Processor

Chapter 6 MS65A Memory

Chapter 7 DWMBB I/O Adapter

Appendix A Console Error Messages for Model 300

Appendix B Console Error Messages for Model 400

Appendix C EEPROM Mismatches and the UPDATE Command

Appendix D Control Flags for Booting

Glossary

Index

Examples

Figures

Tables

Preface

The VAX 6000 Family

The first product in this midrange multiprocessing VAX family is the <REFERENCE>(calypso). Second, the <REFERENCE>(hyperion) offered higher performance, with a faster processor and a more efficient console tape drive (TK70). The third CPU in the series is <REFERENCE>(rigel), which uses advanced technology for more enhanced performance. All models have two versions — a traditional multiuser timesharing system and a VAXserver for diskless computers and workstations.

The VAX 6000 Model 500 introduces a new XMI backplane and power subsystem. While the older versions of the VAX 6000 series relied on a VAXBI and its options for I/O, the newer versions have several I/O adapters that provide I/O directly from the XMI. This manual covers the <REFERENCE>(xyp) and <REFERENCE>(xrp) CPUs in the newer cabinet. Please make sure you have the correct manuals for the systems you are servicing.

Intended Audience

This manual is written for Digital customer service engineers servicing the <REFERENCE>(2x) systems in the VAX 6000 series platform. The manual covers the second and third of the three CPUs in this series. The KA62A will not be shipped in the new <REFERENCE>(XMI2) being introduced with the VAX 6000 Model 500.

Document Structure

This manual uses a structured documentation design. There are many topics, organized into small sections for efficient reference. Each topic begins with an abstract. You can quickly gain a comprehensive overview by reading only the abstracts. Next is an illustration or example, which also provides quick reference. Last in the structure is descriptive text.

This manual has seven chapters and five appendixes:

- **Chapter 1, Introduction**, states why this manual is being written. It also describes how to tell what power and packaging variant you have from the console.
- **Chapter 2, Diagnostics**, describes self-test, general methods for running ROM-based diagnostics, and diagnostics run under the VAX Diagnostic Supervisor. Refer to specific chapters for information on running diagnostics on CPUs, memory, and options.
- **Chapter 3, <REFERENCE>(xyp) Scalar Processor**, describes the <REFERENCE>(hyperion) processor giving module specifications, configuration rules, functional descriptions, and register descriptions. Diagnostics and module replacement are also discussed.
- **Chapter 4, <REFERENCE>(xrp) Scalar Processor**, describes the <REFERENCE>(rigel) processor giving module specifications, configuration rules, functional descriptions, and register descriptions. Diagnostics and module replacement are also discussed.
- **Chapter 5, <REFERENCE>(xrv) Vector Processor**, describes the vector processor option available with the <REFERENCE>(rigel) system. Diagnostics and module replacement are also discussed.
- **Chapter 6, <REFERENCE>(XMA2) Memory**, gives module specifications, configuration rules, register descriptions, and functional descriptions of this memory module. Diagnostics, module replacement, interleaving, and memory addressing are also discussed.
- **Chapter 7, <REFERENCE>(xbi_plus) I/O Adapter**, describes the I/O adapter to the VAXBI option available on either the <REFERENCE>(hyperion) or the <REFERENCE>(rigel) systems.
- **Appendix A** lists the <REFERENCE>(hyperion) console error messages.
- **Appendix B** lists the <REFERENCE>(rigel) console error messages.
- **Appendix C** describes the results of executing the UPDATE command when different ROM revisions are in a system.
- **Appendix D** describes the boot flags used when booting either the VMS or ULTRIX operating systems.

VAX 6000 Series Documents

These documents apply to all VAX 6000 systems.

Title	Order Number
VAX 6000 Series Installation Guide	EK–600EA–IN
VAX 6000 Series Owner's Manual	EK–600EA–OM
VAX 6000 Platform Service Manual	EK–600EA–MG
VAX 6000 Series Platform Technical User's Guide	EK–600EA–TM

VAX 6000 Models 200 and 300 Documents

These documents should be used for systems shipped before the VAX 6000 Model 500. The VAX 6200/6300 documentation set includes the following:

Title: VAX 6200	Order Number
Options and Maintenance	EK–620AB–MG
System Technical User's Guide	EK–620AA–TM

<REFERENCE>(RIGEL) Documents

These documents should be used for systems shipped before the VAX 6000 Model 500. Documents in the VAX 6000–400 documentation set include:

Title	Order Number
VAX 6000–400 System Technical User's Guide	EK–640EB–TM
VAX 6000–400 Options and Maintenance	EK–640EB–MG
VAX 6000–400 Maintenance Advisory	EK–640EA–MA
VAX 6000 Series Upgrade Manual	EK–600EB–UP
VAX 6000 Series Vector Processor Owner's Manual	EK–60VAA–OM
VAX 6000 Series Vector Processor Programmer's Guide	EK–60VAA–PG

Associated Documents

Other documents that you may find useful include:

Table 1: Associated Documents

Title	Order Number
System Hardware Options	
VAXBI Expander Cabinet Installation Guide	EK–VBIEA–IN
VAXBI Options Handbook	EB–32255–46
System I/O Options	
CIBCA User Guide	EK–CIBCA–UG
CIXCD Interface User Guide	EK–CIXCD–UG
DEBNI Installation Guide	EK–DEBNI–IN
DEClancontroller 400 Installation Guide	EK–DEMNA–IN
InfoServer 100 Installation and Owner's Guide	EK–DIS1K–IN
KDB50 Disk Controller User's Guide	EK–KDB50–UG
KDM70 Disk Controller User's Guide	EK–KDM70–UG
KFMSA Module Installation and User Manual	EK–KFMSA–IM
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–KS95B–TE

Table 1 (Cont.): Associated Documents

Title	Order Number
Operating System Manuals	
VMS Installation and Operations: VAX 6000 Series	AA–LB36B–TE
VMS Networking Manual	AA–LA48A–TE
VMS System Manager's Manual	AA–LA00A–TE
VMS VAXcluster Manual	AA–LA27A–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

Introduction

With the introduction of the VAX 6000 Model 500 several changes were made to improve the platform in which the various VAX 6000 models reside. This chapter discusses the changes between the XMI-1 and XMI-2 platforms. Sections include:

- Why Read This Document
- System Functional Description
- Identifying the Platform Remotely
- Troubleshooting Flowcharts

1.1 Why Read This Document

The VAX 6000 series platform has changed to accommodate features of the VAX 6000 Model 500. Several XMI I/O adapters are also now available and can be installed in any XMI card cage. A new power regulator, capable of providing enough current at +3.3 volts for the VAX Model 500, is inhibited in the H9657 cabinet for Models 300 and 400. A more powerful battery backup unit (BBU) option, capable of providing power to the entire backplane, is available.

Table 1–1: VAX 6000 Platform Differences

Item	XMI-1 Platform	XMI-2 Platform
XMI backplane	XMI-1	XMI-2
Cabinet number	70-24900-XX	H9657-XX
XMI I/O adapters	DWMBA	CIXCD, DEMNA, KDM70, DWMBB
VAXBI	Required DWMBA adapter Two 6-slot channels	Optional[1] DWMBB adapter One 12-slot channel
Load device	TK50 or TK70	Optional[1]
XTC	20–29176-01	20–29176-02
Power regulators	+5V – H7214 +5VBB – H7214 -5, -2, ±12 – H7215	+5V – H7214 +5VBB – None[2] -5, -2, ±12 – H7215 +3.3V – H7242
Power logic	H7206-A	H7206-B
Battery backup	H7231-N	H7236-A[2]
Inhibit cable	None	17-02522-01

[1]Either a VAXBI with a TK70 or a CD server is required as a system load device.

[2]In the XMI-2 backplane +5V and what was +5VBB are tied together. Since the VAX 6000 Model 500 uses a writeback cache design, the CPU's cache also needed battery backup and the need for a separate +5VBB to back up only memory disappeared. The H7236-A BBU delivers more power than the H7231-N BBU.

Table 1–1 shows the major differences between the two platforms. Although this manual does not cover the differences in detail, it does cover the <REFERENCE>(2x) systems, the <REFERENCE>(xma2), and the DWMBB in the XMI-2 cabinet. Other documentation covers upgrades (see preface).

Power is distributed differently in the XMI-1 and XMI-2 backplanes. Therefore, power considerations are important when servicing these systems. Care must be exercised when replacing broken modules because of incompatibilities.

The H7242 power regulator provides +3.3 volts to the XMI-2 backplane to operate the VAX 6000 Model 500. The +3.3 volts is inhibited by a cable if either a <REFERENCE>(xyp) or a <REFERENCE>(XRP) is in the system.

All XMI adapters have been designed to work on both backplanes with the *exception* of the DWMBA. A new XMI-to-VAXBI adapter, the DWMBB, is used to communicate between the two buses.

There are several ways to tell which cabinet houses your <REFERENCE>(rigel) or <REFERENCE>(hyperion). Four are described here.

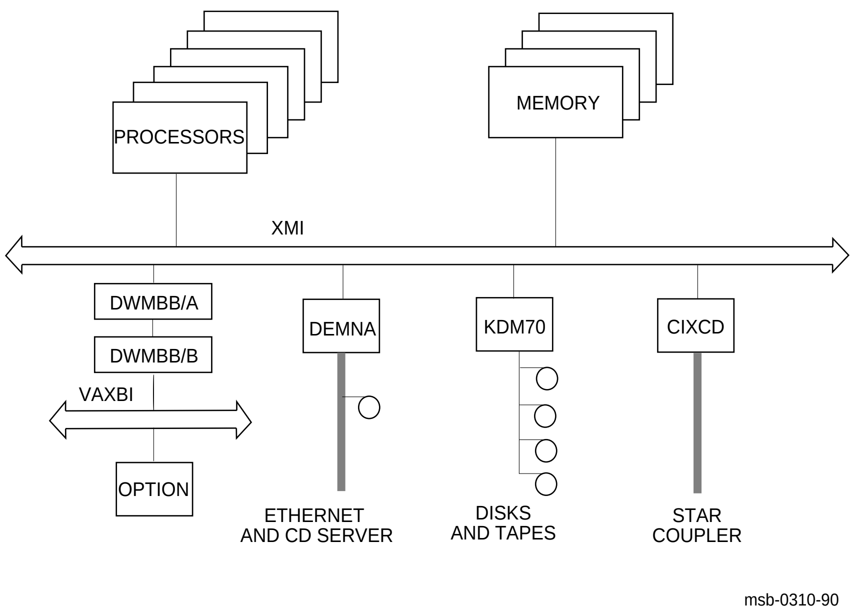
- Look at the cabinet number on the label on the bottom of the vertical frame member at the left rear of the cabinet. If this label is hidden from view by an SA70, you will have to use another method. If the label indicates a 70 class part number, you have the XMI-1 platform. If the label indicates an H9657, you have the XMI-2 platform.
- Open the rear cabinet door and look at the middle power regulator. If the regulator is an H7242, it is an XMI-2 platform. Otherwise, it is an XMI-1 platform.
- Open the rear cabinet door and look to see if the H7242 inhibit cable is installed. This cable has a large yellow tag on it and is easy to see at a glance.
- Open the rear cabinet door and look at the H7206 power logic unit just above the H405 AC controller. If there are two connectors on the H7206, the platform is an XMI-2. Otherwise, it is an XMI-1 platform.

For information on the XMI-1 power and packaging, see the *VAX 6000–400 Options and Maintenance* manual or the *VAX 6200/6300 VAXserver 6200/6300 Options and Maintenance* manual. For information on the XMI-2 power and packaging, see the *VAX 6000 Platform Service Manual*.

1.2 System Functional Description

The <REFERENCE>(2X) systems support multiprocessing with up to six <REFERENCE>(xyp) or <REFERENCE>(XRP) processors. The system uses a high-speed <REFERENCE>(XMI) system bus to connect its processors, its memory modules, and its I/O adapters.

Figure 1–1: <REFERENCE>(2X) System Architecture



The <REFERENCE>(XMI) is the 64-bit system bus connecting all major subsystems. It has a maximum throughput of 100 Mbytes per second.

Early versions of the VAX 6000 series had no direct XMI I/O. I/O was achieved through an interface between the XMI bus and the VAXBI bus which had direct I/O to the CI, disks, tapes, and other I/O devices. Since direct XMI I/O now exists, the VAXBI is now optional.

The VAXBI and <REFERENCE>(XMI) share the concept of a **node**. A node is a single functional unit that consists of one or more modules. The

<REFERENCE>(XMI) has three types of nodes: processor nodes, memory nodes, and I/O adapter nodes.

A **processor node**, consists of a single-board VAX processor. It contains a central processor unit (CPU) chip, a floating-point processor, a primary and secondary cache, a writable EEPROM for system parameters, and a communication path to the XMI bus. The <REFERENCE>(XRP) has a chip that communicates with an optional vector processor.

Processors communicate with main memory over the <REFERENCE>(XMI). The system supports multiprocessing with up to six processors. One processor becomes the boot processor during power-up and handles all system communication. The other processors become secondary processors and receive system information from the primary processor (see Section 3.4 and Section 4.4).

The <REFERENCE>(rigel) has an optional vector processor, the <REFERENCE>(xrv), which executes VAX vector instructions. This processor is tightly coupled to the <REFERENCE>(XRP) through a Vector Interface Bus (VIB) cable connecting the two modules. They occupy adjacent <REFERENCE>(XMI) slots.

A **memory node** is an <REFERENCE>(XMA2). In <REFERENCE>(2x) MS62A may also be installed. See Section 6.14, Mixing MS65A and MS62A Memory Modules for details. Memory is a global resource and may be reached by all processors on the <REFERENCE>(XMI). There are three variants of the <REFERENCE>(xma2); 32-Mbyte, 64-Mbyte, and 128-Mbyte. Each memory module has ECC and control logic. The console program automatically interleaves the memory for maximum performance. An optional battery backup unit protects all data in case of power failure.

The **XMI I/O adapters** include the DEMNA, the CIXCD, and the KDM70. The DEMNA is an Ethernet adapter, the CIXCD is a CI adapter, and the KDM70 is a disk adapter.

An optional <REFERENCE>(XMI)-to-VAXBI adapter, called a <REFERENCE>(XBI_plus), is a 2-module adapter that maps data between these two buses. I/O adapters on the VAXBI pass data between the system and peripheral devices. The <REFERENCE>(XBIA_plus) module is installed on the <REFERENCE>(XMI) bus; it communicates with the <REFERENCE>(XBIB_plus) module on the VAXBI using a 120-pin cable. The in-cabinet version of the VAXBI in the XMI-2 platform has 12 slots. Each VAXBI channel must have one <REFERENCE>(xbi_plus) connecting it to the <REFERENCE>(XMI).

System error messages and self-test results refer to the pair of DWMBB modules as XBI.

1.3 Identifying the Platform Remotely

Section 1.1 explained how to identify the platform by inspection. However, persons diagnosing systems remotely will want to be able to identify system power. While there is no foolproof method to do this on these systems, the method given below should work in almost all cases.

Examples of Power Identification Using Show Configuration

1.
```
>>> SHOW CONFIGURATION                  !Likely an XMI-1 system
      Type          Rev
  1+  KA64A   (8082) 0007
  3+  KA64A   (8082) 0007
  A+  MS62A   (4001) 0002
  B+  MS62A   (4001) 0002
  D+  DWMBA/A (2001) 0002❶
  E+  DWMBA/A (2001) 0002❶

  XBI D
  1+  DWMBA/B (2107) 0007
  2+  CIBCA   (0108) 41C1
  5+  DMB32   (0109) 210B
  6+  DEBNI   (0118) 0100

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

2.
```
>>> SHOW CONFIGURATION                  !Likely an XMI-2 system
      Type          Rev
  4+  KA64A   (8082) 0000
  8+  MS65A   (4001) 0084
  9+  MS65A   (4001) 0084
  A+  CIXCD   (0C05) 1611
  B+  DEMNA   (0C03) 0600
  E+  DWMBB/A (2002) 0001❷

  XBI E
  1+  DWMBB/B (210F) 000A
  4+  KDB50   (010E) 131C
  6+  KLESI-B (0103) 0006
  8+  TBK70   (410B) 0307❸
```

3.

```
>>> SHOW CONFIGURATION                    !Likely an XMI-2 system
     Type           Rev
  1+  <REFERENCE>(XRP)   (8082) 0007
  2+  <REFERENCE>(XRP)   (8082) 0007
  8+  MS65A   (4001) 0084
  9+  MS65A   (4001) 0084
  B+  KDM70   (0C22) 1811
  D+  CIXCD   (0C05) 1611
  E+  DEMNA   (0C03) 0600
```

System configuration provides clues to help identify the power and packaging of <REFERENCE>(hyperion) and <REFERENCE>(rigel) systems. The examples show output from a SHOW CONFIGURATION command on three systems.

System 1. The first system is an XMI-1 based system because there are two DWMBAs at nodes D and E (see ❶ in the example). A DWMBA is not allowed on an XMI-2 based system. Therefore, the proper power and related FRUs are found in the second column of Table 1–1. There are also no XMI I/O devices, which is an indication of the XMI-1. The system also has MS62A memory modules. XMI-2 based systems ship with MS65A memories.

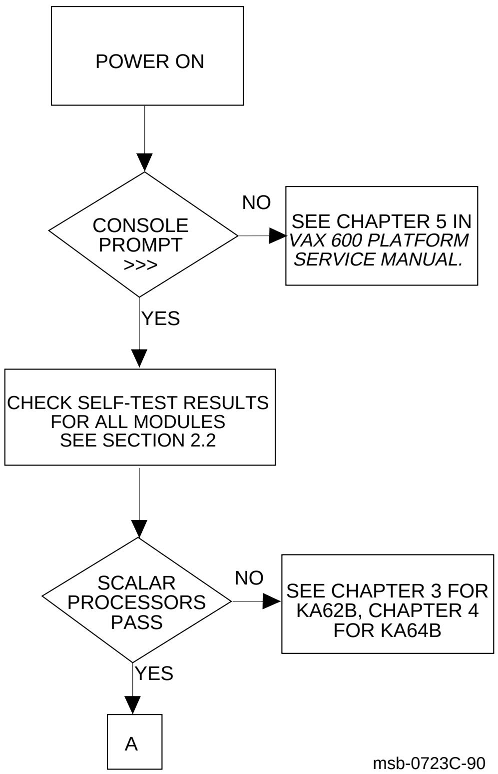
System 2. The second system is an XMI-2 based system because the XMI-to-VAXBI adapter is a DWMBB at node E. A DWMBB is not likely to be found in an XMI-1 based system. Note also that the VAXBI has node numbers above six, indicating a 12-slot VAXBI channel. Therefore, the proper power and related FRUs are found in the third column of Table 1–1.

System 3. The third system is an XMI-2 based system. There is no VAXBI, the memories are all MS65As, and there is no system load device shown. Therefore, the proper power and related FRUs are found in the third column of Table 1–1.

1.4 Troubleshooting Flowcharts

These flowcharts reference sections in this manual.

Figure 1–2: Power-Up



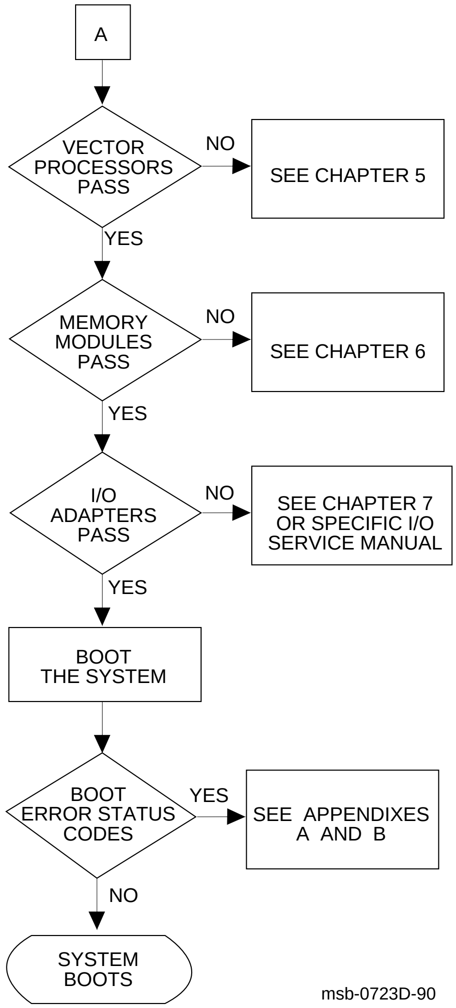
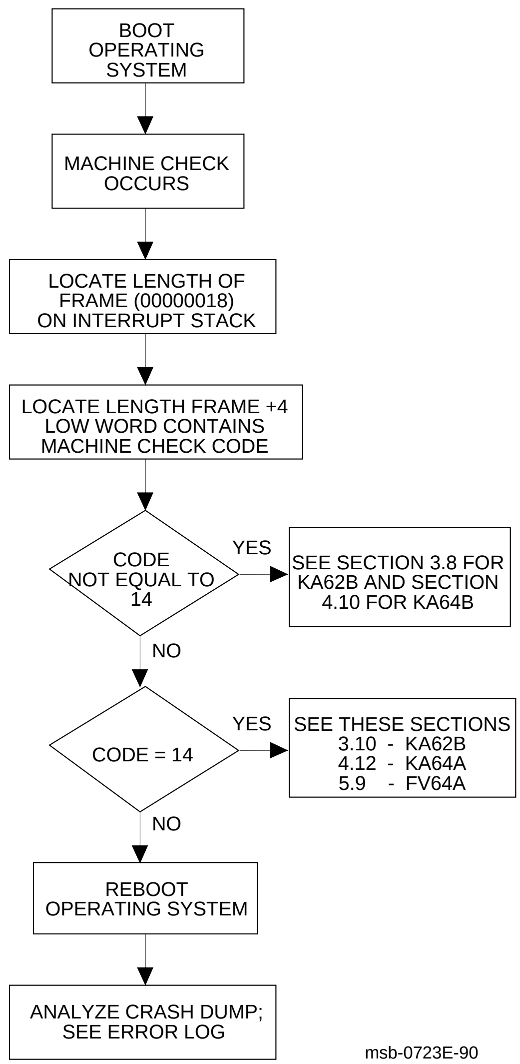

A
VECTOR PROCESSORS PASS
NO
SEE CHAPTER 5
YES
MEMORY MODULES PASS
NO
SEE CHAPTER 6
YES
I/O ADAPTERS PASS
NO
SEE CHAPTER 7 OR SPECIFIC I/O SERVICE MANUAL
YES
BOOT THE SYSTEM
BOOT ERROR STATUS CODES
YES
SEE APPENDIXES A AND B
NO
SYSTEM BOOTS
msb-0723D-90

Figure 1–3: Booting the Operating System



Chapter 2
Diagnostics

This chapter describes diagnostics for the VAX 6000 Model 300 and 400 systems. Sections include:

- Diagnostic Overview
- Self-Test and Additional Power-Up Tests
- ROM-Based Diagnostic Monitor and Its Control

    ROM-Based Diagnostic Programs
    START Command
    START Command Qualifiers
    RBD Test Printout, Passing
    RBD Test Printout, Failing
    SUMMARY Command
    Sample RBD Session
    Running ROM-Based Diagnostics on VAXBI Devices

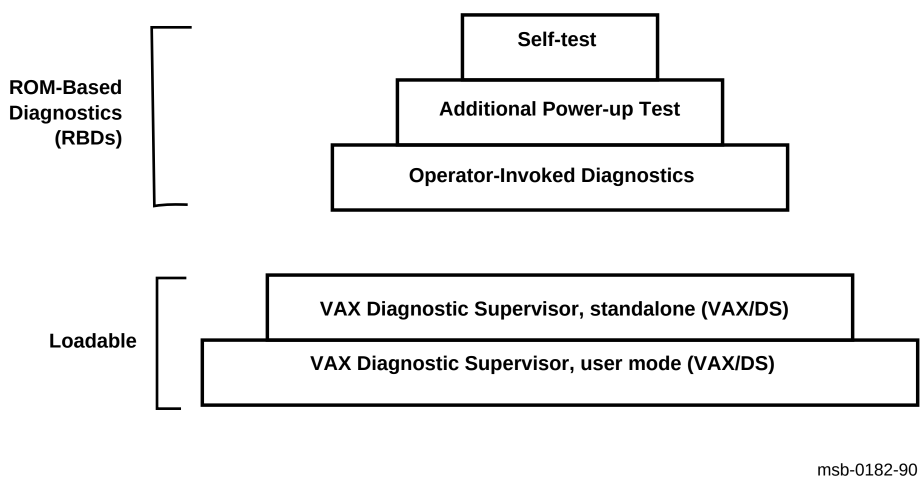
- VAX Diagnostic Supervisor Programs

    Booting the Diagnostic Supervisor from a CD Server
    Running VAX/DS
    Sample VAX/DS Session
    VAX/DS Diagnostics

2.1 Diagnostic Overview

The <REFERENCE>(6000) systems are tested with two types of diagnostics: ROM-based and loadable. The ROM-based diagnostics (RBD) include self-tests, additional power-up tests, and callable diagnostics (from the RBD monitor). The loadable diagnostics run under the VAX Diagnostic Supervisor (VAX/DS) in standalone mode or in user mode (see Figure 2–1).

Figure 2–1: Diagnostics Design



Self-Tests

Each module on the <REFERENCE>(XMI) and VAXBI buses has its own self-test resident in ROM, except for the now optional <REFERENCE>(xbi_plus) modules. At power-up, initialization, booting, or system reset, each module runs its own self-test. The processor self-test completes within 10 seconds. The memory test completes in less than 60 seconds. Results of self-test are printed on the STF line of the console self-test display.

Other Power-Up Tests

Following the modules' self-tests, two additional tests are run and reported in the self-test display: CPU/memory interaction tests and <REFERENCE>(xbi_plus) tests.

All CPUs that have passed self-test run the CPU/memory interaction test. The CPU/memory interaction test checks that the processors can access memory. Memory also has a self-test that tests actual memory locations. The CPU/memory interaction test is the second test for memory and serves as a check on the memory's <REFERENCE>(XMI) interface and on some CPU logic that can be tested only by accessing memory. Results are printed on the ETF line of the self-test display.

The optional DWMBB modules are tested by the boot processor before it queries the VAXBI options for the results of their self-tests. Results from both are printed on the XBI lines on the self-test printout (see Example 2–1).

Operator-Invoked ROM-Based Diagnostics

From the console prompt, you can enter RBD mode and run ROM-based diagnostics. There are five diagnostics for the <REFERENCE>(xyp) and four for the <REFERENCE>(XRP). Common tests include self-tests, CPU/memory interaction tests, DWMBB tests, and memory tests; the <REFERENCE>(xyp) has an additional secondary cache test, and the <REFERENCE>(XRP) has an additional vector interaction test. In RBD mode, you have the capability of running tests other than those in the default suite, running multiple passes of tests, and receiving an error report with information about any failing tests.

VAX Diagnostic Supervisor (VAX/DS)

From the console prompt, you can boot VAX/DS in standalone mode from the TK tape, a CD server, or other media and run level 3 diagnostics. From the operating system, you can run VAX/DS in user mode and run level 2R diagnostics. Level 2 VAX/DS diagnostics may be run either in standalone or user mode. See Table 2–9 for a listing of VAX/DS diagnostics.

2.2 Self-Test and Additional Power-Up Tests

The self-test diagnostics reside in ROM on the processors and on most modules. These diagnostics check each module at power-up, when the system is reset, and during a boot. Self-test results are written to the console terminal, as shown in Example 2–1.

Example 2–1: Sample Self-Test Results

```
#123456789 0123456789 0123456789 01234567#  ! Trace for KA64A in slot 1

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   .   .   M   M   M   M   P   P   P   P   P   P       TYP
    o   +   .   .   +   +   +   +   +   +   +   +   +   +       STF    ❶
    .   .   .   .   .   .   .   .   E   E   E   E   E   B       BPD
    .   .   .   .   .   .   .   .   +   +   +   +   +   +       ETF    ❷
    .   .   .   .   .   .   .   .   E   E   E   E   E   B       BPD

.   .   .   .   .   .   .   .   .   +   +   .   +   .   +   .   XBI E +❸

    .   .   .   .   A4  A3  A2  A1  .   .   .   .   .   .       ILV
    .   .   .   .   64  64  64  64  .   .   .   .   .   .       256Mb

ROM0 = V3.00  ROM1 = V3.00  EEPROM = 2.03/3.00   SN = SG01234567  ❹

>>>
```

The callouts in Example 2–1 refer to Table 2–1.

Self-test is invoked and results are written to the console under several circumstances:

- At power-up
- When the control panel Restart button is pressed
- During a boot
- When the console command INITIALIZE is issued

On a <REFERENCE>(rigel) the first line of the self-test printout is the progress trace. This line indicates how the <REFERENCE>(XRP) at node 1 is functioning during self-test. If there is no processor in node 1, no trace appears.

The remainder of the printout is the self-test display. Table 2–1 describes the tests run during self-test. The callouts in Table 2–1 refer to Example 2–1.

Table 2–1: Self-Test Components

Test	Description
Processors	Each processor runs its own self-test resident in its own ROM. A plus sign (+) on the STF line ❶ means the processor passed. Each processor also tests interaction with memory. A plus sign on the ETF line ❷ means the test passed.
Memory	Each memory runs its own on-board self-test. A plus sign on the STF line ❶ means the memory passed.
<REFERENCE>(XBI PLUS)	The <REFERENCE>(XMI)-to-VAXBI adapter is tested by the boot processor. A plus sign at the right of the XBI line ❸ means both the DWMBB/A and DWMBB/B modules passed.
VAXBI	Each VAXBI bus on the system is checked, and each VAXBI node runs its own self-test. A plus sign in column 0 through F of the XBI line ❸ means the VAXBI node passed.

The self-test printout in Example 2–1 reflects a specific configuration. A detailed explanation of self-test results is available by typing HELP SELF at the console prompt, or see the *VAX 6000 Series Owner's Manual*.

The tests run during self-test can be individually invoked in RBD mode using the ROM-based diagnostics monitor program. Here you can examine each test more closely and determine which test is failing.

Two different ROM revision lines are possible. Example 2–1 shows the KA64A revision line (see ❹). The KA62B revision line follows:

```
ROM = 6.0     EEPROM = 2.0/6.0     SN = SG01234567
```

2.3 ROM-Based Diagnostic Monitor and Its Control

The ROM-Based Diagnostic Monitor program is accessed through the console program. Type T/R at the console prompt to enter RBD mode. RBD mode has three commands with qualifiers and a set of control characters that assist the user when entering commands or running tests.

Table 2–2: RBD Monitor Commands

Command	Function
ST[ART]*n*	Starts RBD *n*, where *n* is the number of the RBD program listed in Table 2–4 and Table 2–5
SU[MMARY]	Prints a summary report of the last RBD program run
QU[IT]	Exits the RBD monitor and returns control to the console program

Table 2–3: RBD Monitor Control Characters

Character	Environment	Function
CTRL/C	Test running	Stops the execution of an RBD test, executes cleanup code, and returns to the RBDn> prompt.
DELETE	RBD command line	Use for deleting erroneous characters entered on the command line.
CTRL/Q	Test running	Resumes output to terminal that was suspended with CTRL/S.
CTRL/R	At RBD prompt	Refreshes the command line; useful when characters are deleted.
CTRL/S	Test running	Suspends output to the terminal until CTRL/Q is typed.
CTRL/U	At RBD prompt	Disregards previous input.
CTRL/Y	Test running	Stops the execution of an RBD test and does not execute any cleanup code.
CTRL/Z	At RBD prompt	Exits RBD monitor program and enters console program; same effect as the QUIT command.

To enter the RBD monitor, at the console prompt type:

```
>>>  T/R              !  This is the abbreviation for TEST/RBD.
                      !  Early versions of the console only accept T/R.
RBDn>                 !  RBD prompt appears signifying entrance into
                      !  RBD mode, where n is the XMI node number of
                      !  the processor running the RBD monitor program.
```

The RBD commands are explained here and in Sections 2.3.2 and 2.3.6. Table 2–2 gives the commands, their abbreviations, and functions.

Four programs run from the ROM-based diagnostics (RBD) monitor program. The programs are CPU self-test, CPU/memory interaction tests, the DWMBB tests, and memory RBD tests. Each of these programs has several tests, as shown in Table 2–4 and Table 2–5. A fifth program is available for the <REFERENCE>(hyperion) and tests it's secondary cache. The RBDs are designed for use by Digital customer service personnel.

The RBD monitor responds to several control characters. These characters manage both the diagnostic and the monitor as shown in Table 2–3. When CTRL/C is entered from the console terminal, the diagnostic stops execution, runs cleanup code, and returns control to the RBD monitor program. This happens immediately when running RBD 0, RBD 1, or RBD 2; there may be a wait of up to one minute for a response when RBD 3 is running. If CTRL/C is typed at the RBD monitor prompt, it has the same effect as CTRL/U.

When you use the DELETE key (or rubout key), characters being deleted are preceded by a backslash (\ ) and print as they are rubbed out. When the next valid character is typed, it is preceded by a backslash (\ ) to delineate the deleted characters. You can use CTRL/R to refresh the line.

When the RBD monitor program receives a CTRL/U, the program disregards all previous input typed and returns the RBD prompt. If a test is running when CTRL/U is entered, CTRL/U is ignored.

When a CTRL/Y is received by the RBD monitor program from the console, the diagnostic stops execution and returns control to the RBD monitor program. No cleanup code is run, and the unit under test is left in an indeterminate state. A CTRL/Y entered at the RBD monitor prompt has the same effect as CTRL/U.

When the RBD monitor program receives a CTRL/Z, the program exits and control is returned to the console program (>>>). CTRL/Z has the same effect as the QUIT command. If CTRL/Z is entered while an RBD test is running, CTRL/Z has the same effect as CTRL/C: it halts the test and executes cleanup code.

2.3.1 ROM-Based Diagnostic Programs

Table 2–4 and Table 2–5 lists the diagnostic programs available for both VAX 6000 Model 300 and Model 400 systems.

Table 2–4: ROM-Based Diagnostic Programs for KA62B

RBD Program	Test Totals	Default (Power-Up)	Default (Callable)	Description
0	34	34	34	Runs CPU tests
1	20	20	20	Runs CPU/memory interaction tests
2	26	21	20	Runs DWMBB tests
3	12	0	7	Sizes and runs additional tests on main memory
4	8	0	0	Miscellaneous tests of second-level cache

Table 2–5: ROM-Based Diagnostic Programs for KA64A

RBD Program	Test Totals	Default (Power-Up)	Default (Callable)	Description
0	37	37	37	Runs CPU tests
	49	49	49	Runs scalar and vector tests
1	13	12	13	Runs CPU/memory interaction tests
	17	16	17	Runs scalar/vector CPU/memory interaction tests
2	26	20	20	Runs DWMBB tests
3	12	0	7	Sizes and runs additional tests on main memory

Table 2–4 and Table 2–5 show the number of tests available. Column 1 contains the numbers used to invoke the test in RBD mode.

```
RBD2> ST1         !Starts the CPU memory interaction test
```

The second column gives the total number of tests in the RBD. The third column indicates that there may be some tests not run during power up, and the fourth column indicates the number of test run when the RBD is invoked by the START command with no qualifiers. All tests may be run. Some, like a number of the memory diagnostics, modify data or take a long time and are not run by default. These tests require the operator to invoke them specifically.

For the <REFERENCE>(xyp) each RBD has a default number of tests that run when the test is invoked. The CPU and CPU/memory tests (RBD 0 and 1) run all their tests when invoked. RBD 4, the secondary cache tests, are not run by default and must be specifically requested to run. The <REFERENCE>(XRP) does not have a separate RBD for its secondary cache.

For the <REFERENCE>(XRP) each RBD has a default number of tests that run at power-up, and another default number of tests that run when the program is called from the RBD monitor (see Table 2–4 and Table 2–5). The CPU diagnostic (RBD 0) runs all its tests in both modes. The power-up default for the CPU/memory interaction diagnostic (RBD 1) is 12 (scalar only—16 with the vector tests), and the callable default is all tests.

The DWMBB diagnostic (RBD 2) runs 20 or 21 of the 26 tests as the default in both modes; tests 2, 3, 4, 10, 11, and 26 must be invoked by qualifier in callable mode.

To run tests other than the default suite from the RBD monitor, issue a command such as the following, which invokes all DWMBB tests:

```
RBDn> ST2/T=1:26 D
```

See Section 7.4 for more details.

RBD 3, the Memory diagnostic, does not run on power-up. In callable mode, 7 of 12 tests run when invoked; tests 2 through 8 are defaults. See Section 6.11 for information on calling additional tests.

It is helpful to use the trace qualifier, /TR, with the RBD START command. (See Section 2.3.3.) This qualifier shows each individual test as it is run. If a test fails, the program displays error messages. By default, the RBDs continue testing after an error is encountered. Adding the halt-on-error qualifier, /HE, causes the program to halt when the first error is encountered. Testing can be aborted at any time by typing CTRL/C.

To exit RBD mode, type QUIT at the RBD prompt.

2.3.2 START Command

The RBD monitor START command invokes a specific RBD program. It takes an argument indicating the RBD program to be run, and can take any of 13 qualifiers.

Example 2–2: START Command

```
>>> T/R                       ! Command to enter RBD monitor program.

RBD3>                         ! RBD monitor prompt, where 3 is the hexa-
                              ! decimal node number of the processor
                              ! that is currently receiving your input.

RBD3> ST2/TR E                ! Runs the default XBI tests, testing the
                              ! <REFERENCE>(xbi_plus) at <REFERENCE>(XMI) node number E. Test results
                              ! are written to the console terminal.

RBD3> ST1/HE/IE/BE            ! Runs the CPU/memory interaction RBD, halting
                              ! on the first error encountered, inhibiting
                              ! error output, ringing the bell when the
                              ! first error is  encountered.
```

The START command syntax is:

ST*n*[/qualifier] [parameter]

where:

- *n* is the RBD to be run (see Table 2–4 and Table 2–5).
- [/qualifier] is one of those listed in Section 2.3.3.
- [parameter] is a program-specific value used in RBD 2 or RBD 3. (For the meaning of this parameter in RBD 2, see Section 7.4, and in RBD 3, see Section 6.11.)

2.3.3 START Command Qualifiers

The START command is the primary RBD program command. Its qualifiers act as switches, allowing you to control the output of the tests—to run portions of a test, to run nondefault tests, and to loop on tests.

Table 2–6: START Command Qualifiers

Qualifier	Default	Function
/BE	Disabled	Bell sounds when an error is encountered
/C	Disabled	Destructive test confirmation
/DS	Disabled	Disable status reports
/HE	Disabled	Halt on the test that incurs a hard error
/HS	Disabled	Halt on the test that incurs a soft error (<REFERENCE>(XRP) only)
/IE	Disabled	Inhibit all error output
/IS	Disabled	Inhibit summary reports
/LE	Disabled	Loop on the test that incurs a hard error
/LS	Disabled	Loop on the test that incurs a soft error (<REFERENCE>(XRP) only)
/P=*n*	Enabled	Make *n* passes of the test or tests indicated
/QV	Disabled	Quick verify mode
/T=*n[:m]*	Enabled	/T=*n* runs test *n*; /T=*n:m* runs a range of tests from *n* through *m*
/TR	Disabled	Print a trace of test numbers, as they run

NOTE: *A qualifier is valid only for the command with which it is issued. Qualifiers do not remain in effect for the session.*

See Example 2–2 for examples and a description of the START command syntax.

/BE causes a bell on the console terminal to ring when an error is encountered. This is useful when error printout is inhibited and a loop on intermittent error is set (/LE).

/C enables execution of destructive tests. See Example 2–7 and Section 6.11 for information on the destructive tests.

/DS disables printout of the diagnostics test results. The summary report is run, unless it is specifically disabled.

/HE halts on hard error and stops execution of tests as soon as the first hard error is encountered. (In this context, a hard error is defined as a recoverable, repeatable error, for example, a ROM checksum error. This differs from a fatal error, which is an unrecoverable fault, for example, an unexpected interrupt or exception. A fatal error is always cause for program abortion, regardless of the state of the /HE or /LE qualifier.) The test number is printed, and a summary indicating failure of the RBD is printed to the console terminal. Also the RBD monitor prompt is returned. Continue on error is the default condition, so if you want a halt on error, you must specifically invoke it in your command line.

/HS halts on soft error and stops execution of tests as soon as the first soft error is encountered. This qualifier works only on the <REFERENCE>(XRP) module. (In this context, a soft error is defined as a recoverable error that goes away after retry, for example, a corrected read data memory error.) The test number is printed, and a summary indicating failure of the RBD is printed to the console terminal. Also the RBD monitor prompt is returned. Continue on soft error is the default condition, so if you want to halt on soft error, you must specifically invoke it in your command line.

/IE inhibits all error output, suppressing printing of RBD results. This qualifier is used primarily for module repair, in conjunction with the /LE or /LS qualifier. Errors are counted even when the printing is disabled.

/IS suppresses printout of RBD summary after the end of the last pass performed by the RBD.

/LE loops on the test where the first hard error is detected. Even if the error is intermittent, looping continues on the test indicated. To terminate /LE, enter CTRL/C, CTRL/Z, or CTRL/Y. After entering one of these control characters, a summary report is printed. A fatal error causes the program to abort, regardless of the state of this qualifier.

/LS loops on the test where the first soft error is detected. This qualifier works only on the <REFERENCE>(XRP) module. Even if the error is intermittent, looping continues on the test indicated. To terminate /LS, enter CTRL/C, CTRL/Z, or CTRL/Y. After entering one of these control characters, a summary report is printed.

/P=*n* runs *n* passes of the RBD test invoked; *n* is a decimal number. If *n* is 0, all selected tests run for an infinite number of passes. If the /P qualifier is not used, the program defaults to one pass of the test invoked. When used with the /T=*n:m* qualifier, you run a range of tests. To terminate /P=*n*, enter CTRL/C, CTRL/Z, or CTRL/Y. After entering one of these control characters, a summary reports prints out and the RBD monitor prompt returns.

/QV selects the quick verify version of any selected test that supports this mode.

/T=*n[:m]* selects individual tests (/T=*n*) or a range of tests (/T=*n:m*) where *n* and *m* are decimal numbers. For example, to run tests T0005 through T0008, use /T=5:8. If no /T qualifier is used, the diagnostic runs its default suite of tests.

/TR prints each test number as it is completed. This qualifier allows you to trace the progress of the diagnostic as it runs. Without the /TR qualifier, just the summary line is printed.

For both the <REFERENCE>(xyp) and the <REFERENCE>(XRP) a single **parameter field** can be appended to the START command string to control aspects of the diagnostic that are not covered by the qualifiers. The parameter must be appended after any qualifiers and separated from them by a space. The format of the parameter field is 4 hex characters. The use of a parameter field is implementation specific.

2.3.4 RBD Test Printout, Passing

The RBD printout results are different when the RBD tests pass and when they fail. Example 2–3 shows a passing printout, and Example 2–4 is a sample failure printout.

Example 2–3: RBD Test Printout, Passing

```
>>> T/R                         ! Command to enter RBD monitor program at
                                ! console prompt.
RBD3>                           ! RBD monitor prompt, where 3 is the hexa-
                                ! decimal node number of the processor
                                ! that is currently receiving your input.

RBD3> ST2/TR E                  ! Runs the XBI self-test, testing the <REFERENCE>(xbi_plus)
                                ! at <REFERENCE>(XMI) node number E. Test results
                                ! written to the console terminal:

; XBI_TEST❶           3.00❷

; T0001  T0005  T0006  T0007  T0008  T0009  T0012  T0013  T0014  T0015
; T0016  T0017  T0018  T0019  T0020  T0021  T0022  T0023  T0024  T0025❸

;        P❹       3❺   8082❻      1❼
; 00000000 00000000 00000000 00000000 00000000 00000000 00000000❽


RBD3> QU❾                       ! RBD prompt returns; test ran successfully.
                                ! Exit RBD program.
>>>
```

The callouts in Example 2–3 are explained below.

❶ This entry designates which test is being run. Here it is the XBI_TEST, the self-test for the DWMBB. Below is a list of some XMI RBDs.

XRP/V_ST	RBD 0, the CPU tests for <REFERENCE>(XRP)
XCPST	RBD 0, the CPU tests for <REFERENCE>(xyp)
CPUMEM	RBD 1, the CPU/memory interaction tests
XBI_TEST	RBD 2, the DWMBB tests
XMA_RBD	RBD 3, the memory tests
SLCRBD	RBD4, the second-level cache tests for <REFERENCE>(xyp)

❷ This field lists the revision number of the RBD program.

❸ These T00*nn* fields appear only with the /TR qualifier; each entry corresponds to a test being run and prints out as the test starts running. In a passing RBD, the final T00*nn* number corresponds to the last test run.

Note that T0002 through T0004 and T0010, T0011, and T0026 are not executed. These tests are not part of the default selection and must be individually invoked by qualifier. For a list of the tests for each RBD and the definition of the tests, see the chapter for each module in this manual.

❹ This field indicates whether the RBD passed or failed; P for passed, F for failed.

❺ This field is the XMI node number of the boot processor executing the RBD. It matches the number in your RBD prompt.

❻ This field is 8082 for the <REFERENCE>(XRP) and 8001 for the <REFERENCE>(xyp)—the device type of the boot processor.

❼ This field displays the total number of passes (in decimal) executed by the RBD. The default number of passes is 1. If you use the START command with the qualifier /P=5, for example, then this field will show 5, indicating 5 passes were completed.

❽ This line contains the summary of the RBD failures. In a successful RBD run, the line will contain all zeros as shown here. Currently only the second and third fields are used. The second field contains the number of hard errors detected during the run. The third field contains the number of soft errors detected during the run.

❾ The console prompt is usually returned in response to the RBD QUIT command, as shown in this example. However, when some tests are run, the response to QUIT is a system reset. Self-test is then run, and the self-test results are printed. The tests that cause a system reset are: Test 1 and 22 of RBD 1; Tests 2, 3, 4, 10, and 11 of RBD 2; and Tests 4 and 8 of RBD 3.

2.3.5 RBD Test Printout, Failing

The RBD printout results are different when the RBD passes and when it fails. Example 2–4 is a sample failure printout, and Example 2–3 shows a passing printout.

Example 2–4: RBD Test Printout, Failing

```
>>> T/R                         ! Command to enter RBD monitor program at
                                ! console prompt.
RBD2>                           ! RBD monitor prompt, where 2 is the hexa-
                                ! decimal node number of the processor
                                ! that is currently receiving your input.
RBD2> ST0/TR                    ! Execute RBD 0 (CPU test) and trace results.

; XRP/V_ST       3.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018❶

;        F❷         2❸      8082❹        1❺
;       HE❻    REX520❼       XX❽     T0018❾
;       10❿ AAAAAAAA⓫ A8AAAAAA⓬ 00000000⓭ 000004AC⓮ 2006451F⓯ 01⓰
; T0019  T0020  T0021  T0022  T0023  T0024  T0025  T0026  T0027  T0028
; T0029  T0030  T0031  T0032  T0033  T0034  T0035  T0036  T0037⓱

;        F        2       8082       1⓲
; 00000000 00000001⓳ 00000000 00000000 00000000 00000000 00000000

RBD2>                           ! RBD prompt returns; test completed.
RBD2> QUIT                      ! Exit RBD program.
>>>                             ! Console prompt reappears.
```

The callouts in Example 2–4 are explained below. (See also Example 2–3 for explanation of other fields of the printout.)

❶ These T00*nn* fields appear only with the /TR qualifier; each entry corresponds to a test being run. The entry prints out as the test starts running. This T00*nn* number is the number of the failing test and is followed by a failure report. In this example, test 18 failed. The /HE qualifier was not used, so testing continues.

❷ F indicates failure of the previous test listed, test 18.

❸ This field is the XMI node number of the boot processor executing the RBD. It matches the number in your RBD prompt.

❹ This field is the device type of the boot processor.

❺ This field displays the total number of passes (in decimal) executed by the RBD. The default number of passes is 1.

❻ The class of error is displayed here. *HE* indicates that the error was a hard error. *SE* means the error was a soft error, and *FE* indicates a fatal error. (See Section 2.3.3 for a definition of these errors.)

❼ This field describes the failing logic for the <REFERENCE>(XRP) and gives an abbreviated test name for the <REFERENCE>(xyp). Here, the processor chip of a <REFERENCE>(XRP), device type 8082, is the failing logic.

❽ This field is the unit number used in memory and DWMBB tests.

❾ This field lists the number of the test that failed; test 18 failed here.

❿ This is a two-digit (decimal) generic error code.

⓫ The expected data is listed here. AAAAAAAA is the data test 18 expected.

⓬ The received data is listed here. A8AAAAAA is the data test 18 received.

⓭ This field shows any unexpected interrupt vectors.

⓮ This is the address in memory where the referenced error is found.

⓯ This is the address of the failing PC at the time of error.

⓰ This is the error number within the failing test. In this example, the failure was detected at the first possible failure point in T0018. This is a decimal field.

⓱ This final T00*nn* number corresponds to the last test run.

⓲ This entire line is the summary line, and a repeat of the failure summary. It lists the pass/fail code (P or F), the node number and device type number of the boot processor executing the RBD, and the number of passes of the RBD.

⓳ This is the number of hard errors detected.

2.3.6 SUMMARY Command

The RBD monitor SUMMARY command displays a summary of the last diagnostic run.

Example 2–5: SUMMARY Command

```
>>> T/R                        ! Command to enter RBD monitor program at
                               ! console prompt.
RBD1> ST0/IE/IS/P=100          ! Execute RBD 0 (CPU test), inhibiting
                               ! error outputs and summary report.
; XRP/V_ST     3.00


RBD1> SU                       ! Request a summary.

; XRP/V_ST     3.00

;        P❶      1❷   8082❸     100❹
; 00000000 00000000 00000000 00000000 00000000 00000000 00000000❺

RBD1>
```

The callouts in Example 2–5 are explained below.

❶ This field indicates whether the RBD passed or failed; P for passed, F for failed.

❷ This field is the XMI node number of the boot processor executing the RBD. It will match the number in your RBD prompt, which also indicates the node number of your boot processor.

❸ This field is the device type number of the boot processor executing the RBD.

❹ This field displays the total number of passes executed by the RBD.

❺ This line contains the summary of the RBD failures. Presently only the second and third fields are used. The second field contains the number of hard errors detected during the run. The third field contains the number of soft errors detected during the run.

2.3.7 Sample RBD Session

Examples 2–6 and 2–7 show a sample RBD session.

Example 2–6: Sample RBD Session, Part 1 of 2

```
>>>  T/R❶
RBD1> ST0/TR❷

;XRP/V_ST       3.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018  T0019  T0020
; T0021  T0022  T0023  T0024  T0025  T0026  T0027  T0028  T0029  T0030
; T0031  T0032  T0033  T0034  T0035  T0036  T0037

;       P        1     8082        1
;00000000 00000000 00000000 00000000 000000000 00000000 00000000

RBD1> ST1/TR/HE❸

;CPUMEM       3.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013

;       P        1     8082        1
;00000000 00000000 00000000 00000000 000000000 00000000 00000000

RBD1> ST2 5❹

;XBI_TEST     3.00

;       F        1     8082        1
;      FE  No_Unit       05    T0000
;      00 00000000 00000000 00000000 00000000 200705E7 01

;       F        1     8082        1
;00000000 00000001 00000000 00000000 00000000 00000000 00000000

RBD1>  ST2/TR/T=2:4/P=3 E❺

;XBI_TEST     3.00

; T0002  T0003  T0004  T0002  T0003  T0004  T0002  T0003  T0004

;       P        1     8082        3
;00000000 00000000 00000000 00000000 000000000 00000000 00000000
```

❶ Enter RBD mode from console mode. The RBD prompt appears and indicates you are operating from the boot processor at node 1.

❷ Run RBD 0 and trace the tests. The CPU test runs all 37 tests successfully.

❸ Run RBD 1, trace it and halt on the first error found. All CPU/memory interaction RBD tests run and pass.

❹ Run RBD 2, testing the DWMBB at XMI node 5. The value NO_UNIT on the third line of output indicates that the node value of node 5 is not correct; no DWMBB was found at this node.

❺ Run 3 passes of tests 2 through 4 of RBD 2 on node E with Trace Set.

Note that the T00*nn* line lists each of the three tests three times, since the /P=3 called for 3 passes of the tests. And the final parameter in the summary line is a 3, indicating that 3 passes completed.

Example 2–7: Sample RBD Session, Part 2 of 2

```
RBD1> ST3/TR/T=1❻

RBD1> ST3/TR/T=1

RBD1> ST3/TR/T=1 /C❼

;XMA_RBD      3.00

; T0001

;       P        1     8082       1
;00000000 00000000 00000000 00000000 000000000 00000000 00000000

RBD1> QU❽

          [self-test results may be displayed here]

>>> SET CPU 2❾
>>> T/R

RBD2> ST0/TR❿

;XRP/V_ST       3.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018  T0019  T0020
; T0021  T0022  T0023  T0024  T0025  T0026  T0027  T0028  T0029  T0030
; T0031  T0032  T0033  T0034  T0035  T0036  T0037

;       P        2     8082        1
;00000000 00000000 00000000 00000000 000000000 00000000 00000000
```

❻ Run RBD 3, trace it, and run only test 1 of this RBD. This test is one of the memory tests that is not part of the default suite of tests. Test 1 corrupts memory. You must add a /C qualifier (Confirm) to the START command, to indicate that you do indeed intend to run this destructive test.

The /C qualifier was not given in this example. The command line is echoed, waiting for /C to be typed.

On a <REFERENCE>(rigel) you can either press Return and reenter the command with the /C qualifier, or you can type the /C qualifier followed by Return.

On a <REFERENCE>(hyperion) you must enter the command line again adding the /C qualifier.

❼ Run RBD 3, trace the tests as they run, run only test 1, and /C allows the test to run. In this example, the test completed with no errors.

❽ Exit from RBD mode. Enter console mode.

❾ Make the next processor the primary processor so that RBD 0 can be run on it.

❿ Run RBD 0 and trace the tests. Here the CPU runs all 37 <REFERENCE>(XRP) tests successfully.

2.3.8 Running ROM-Based Diagnostics on VAXBI Devices

Some VAXBI devices can be tested from the console terminal with their on-board ROM-based diagnostics. The Z console command is used to send commands to these VAXBI nodes.

Example 2–8: VAXBI RBD Session

```
>>> SHOW CONFIGURATION❶
      Type           Rev
  1+  <REFERENCE>(XRP)   (8082) 0007
  A+  MS65A   (4001) 0002
  E+  DWMBB/A (2002) 0001

  XBI E
  1+  DWMBB/B (210F) 000A
  5+  DMB32   (0109) 210B
  6+  DEBNI   (0118) 0100
  7+  KDB50   (010E) 0F1C
  8+  TBK70   (410B) 0307

>>> Z/BI:6 E❷
?33 Z connection successfully started

T/R❸

RBD6> ST 0/TR❹

;DEBNI_ST     1.02

; T01  T02  T03  T04  T05  T06  T07  T08

;       P        6     0118 00000020
;00000000 00000000 00000000 00000000 00000000 00000000 00000000

;   PUDR: FFFFxxxx

RBD6> QUIT❺
^P
?31 Z connection terminated by ^P

>>> Z/BI:8 E❻
?33 Z connection successfully started

T/R
RBD8> ST 0/TR

;T1035_St     1.00
```

Example 2–8 Cont'd on next page

Example 2–8 (Cont.): VAXBI RBD Session

```
; T01  T02  T03  T04  T05  T06  T07  T08  T09  T10  T11  T12  T13  T14
; T15  T16  T17

;       P        8     410B 00000001
;00000000 00000000 00000000 00000000 00000000 00000000 00000000

;   PUDR: 5FF43FDF

RBD5> QUIT
^P
?31 Z connection terminated by ^P

>>>
```

The callouts in Example 2–8 are explained below.

❶ The SHOW CONFIGURATION console command shows that this system includes a DEBNI at node 6 of the VAXBI attached at XMI node E, and a TBK70 at node 8 of the VAXBI attached at XMI node E.

❷ The Z command is typed at the console prompt. A connection is established to node 6 (/BI:6) of the VAXBI connected at XMI node E (E). The console returns a message confirming that the connection has been made.

❸ After the console message is returned in ❷, no prompt is printed. Typing T/R invokes the RBD monitor on the VAXBI adapter being tested and returns the RBD monitor prompt. Note that the "6" in the RBD prompt refers to the VAXBI node.

❹ The START command for VAXBI RBDs requires a space before the 0. When run with the /TR qualifier, test traces are printed. The last line of the summary report indicates the contents of the Power-Up Diagnostic Register. Refer to the technical manual for the device being tested to interpret the contents reported.

❺ The QUIT command exits the RBD monitor. The Z connection remains until CTRL/P is entered.

❻ Steps ❷ through ❺ are repeated to run the RBD of the TBK70 at node 8 of the VAXBI attached at XMI node E.

2.4 VAX Diagnostic Supervisor Programs

The VAX Diagnostic Supervisor (VAX/DS) is a monitor that controls operation of a diagnostic program. You can use VAX/DS in one of two modes: standalone mode (exclusive use of the system) or user mode (under the VMS operating system).

Table 2–7: VAX Diagnostic Program Levels

Level	Type of Test	Run-Time Environment
1	System exercisers	Runs under the VMS operating system without VAX/DS
2R	Function tests of peripheral devices	Runs under the VMS operating system with VAX/DS
2	Exercisers and function tests of peripheral devices and processors	Runs under VAX/DS in user mode and standalone mode
3	Function tests and logic tests of peripheral devices and processors	Runs under VAX/DS in standalone mode

Table 2–8: VAX/DS Documentation

Document	Order Number
VAX Diagnostic Supervisor User's Guide	AA–FK66A–TE
VAX Diagnostic Software Handbook	AA–F152A–TE
VAX Diagnostic Design Guide	AA–FK67A–TE
VAX Systems Hardware Handbook	EB–31692–46

The VAX Diagnostic Supervisor (VAX/DS) can be run in interactive mode. You type commands in response to the VAX/DS program prompt:

```
DS>
```

VAX/DS lets you load diagnostic programs into system memory, select devices to be tested, and run the programs. The VAX/DS command language also lets you control the execution of diagnostic programs; you can specify which tests or sections of a program should run, and how many passes it should run. You can also show the current state of parameters that affect the operation of diagnostic programs. The programs report their results through VAX/DS to the terminal.

VAX/DS supports three types of diagnostic programs:

- Logic tests
  Test a specific section of a device's logic circuitry. Logic tests provide the greatest degree of detail in determining the location of faulty hardware.
- Function tests
  Test the functions of the device. For example, a function test for a disk drive would test the drive's reading and writing capabilities. Function tests can detect the location of faulty hardware, although the results may be less exact than those of a logic test.
- Exercisers
  Test entire systems or subsystems and verify that a system can function properly over a period of time. Exercisers can detect both hardware faults resulting from the simultaneous use of a system's numerous devices and intermittent faults occurring only once or twice over a long period of time.

Table 2–9 lists the VAX/DS programs available for the <REFERENCE>(rigel) and <REFERENCE>(hyperion) systems. Each program has a HELP file available. To access the help files for any diagnostic, at the VAX/DS prompt, type:

```
DS>   HELP [VAX/DS diagnostic program name]
```

2.4.1 Booting the Diagnostic Supervisor from a CD Server

To boot the Diagnostic Supervisor from a CD server, set an additional control flag in R5 and enter ISL_LVAX at the filename prompt.

Example 2–9: Booting the Diagnostic Supervisor over the Ethernet

```
>>> BOOT /XMI:E /BI:C /R5:110 ET0 ❶

 Initializing system

          [Self-test display prints]

 Loading system software

Filename: ISL_LVAX  ❷

          [Several establishing link messages]

  Ethernet Initial System Load Function

   FUNCTION        FUNCTION
     ID
     1     -       Display Menu
     2     -       Help
     3     -       Choose Service ❸
     4     -       Stop

  Enter a function Id value: 3 ❸

  Service options:
   1 = Find Services
   2 = Enter Known Service Name
   =>1  ❹

  Working

  Servers found: 1

  Service Name Format:
  Service Name
  Server Name
  Ethernet ID

 #1
 VMS054
 ESS_08002B15FCE1
 08-00-2B-15-FC-E1
```

Example 2–9 Cont'd on next page

Example 2–9 (Cont.): Booting the Diagnostic Supervisor over the Ethernet

```
#2
NSS_SYSDISK
ESS_08002B15FCE1
08-00-2B-15-FC-E1

#3
6000_DIAG_A
ESS_08002B15FCE1
08-00-2B-15-FC-E1

 Enter a number =>3 ❺

    [Diagnostic Supervisor Banner prints]

DS>
```

After placing a CDROM cartridge labeled 6000_DIAG_* in a CD server on the network, your system can access and boot the Diagnostic Supervisor from the Ethernet.

❶ At the console prompt enter a boot command identifying a path to the CD server.

Should the Ethernet be attached directly to the XMI through a DEMNA, you will not need the /BI qualifier and you will need to change the device name to EX*nn*.

R5 = 110 sets two flags for VMB. Bit 4 specifies the Diagnostic Supervisor. Bit 8 prompts for a secondary loader.

❷ At the prompt, "Filename" type ISL_LVAX — the secondary loader. The system loads software and identifies various functions available.

❸ At the prompt, "Enter a function Id value:" enter a 3 to Choose Service.

❹ Assuming you do not know the Service Name or number, at the "=>" prompt enter a 1 to Find Services, otherwise enter a 2.

The available services are then located and shown. You should see the name of the CD you just put into the CD server.

❺ At the "Enter a number =>" prompt, enter the number of the server that has the CD with the Diagnostic Supervisor on it. The Diagnostic Supervisor then boots.

2.4.2 Running VAX/DS

You can use VAX/DS in one of two modes: standalone mode (exclusive use of the system) or user mode (under VMS). There are several methods of booting the VAX/DS in standalone mode any one of which can be used. Section 2.4.1 shows one, Example 2–10 shows another, and Example 2–12 shows a third.

Example 2–10: Running Standalone VAX/DS from a TK70

```
>>> BOOT/R5:10 CSA1           ! Enter BOOT command designating the
                              ! TK tape drive as input device; /R5:10
                              ! is the boot flag indicating the
                              ! VAX/DS program.
                 [self-test results print]
Loading system software.
                    VAX DIAGNOSTIC SOFTWARE
                          PROPERTY OF
                 DIGITAL EQUIPMENT CORPORATION
              ***CONFIDENTIAL AND PROPRIETARY***

Use Authorized Only Pursuant to a Valid Right-to-Use License
Copyright, Digital Equipment Corporation, 1990.  All Rights Reserved.

DIAGNOSTIC SUPERVISOR. ZZ-ERSAA-14.X-NNNN  1-JAN-1990 12:01:45
DS>                           ! System boots VAX/DS and displays banner.
                              ! Run VAX/DS level 3 or 2 programs.
DS> ^P                        ! Enter CTRL/P to exit VAX/DS
?02 External halt (CTRL/P, break, or external halt)
    PC  = 00027056
    PSL = 00000200
    KSP = 0004D210

>>>                           ! Console prompt returns.
```

Example 2–11: Running VAX/DS in User Mode

```
$                              ! At the operating system prompt, run
$ RUN ERSAA                    ! the VAX/DS program.
          [VAX/DS banner prints, as in example above]
DS>                            ! VAX/DS prompt appears.
                               ! Run VAX/DS level 2R or 2 programs.
DS> EXIT                       ! Type EXIT to exit VAX/DS
$                              ! Operating system prompt returns.
```

Table 2–7 describes the levels of VAX/DS programs. Check Table 2–9 for the programs you wish to run, and determine if you will run VAX/DS in standalone or user mode.

To run VAX/DS in standalone mode on systems with a TK70 insert a TK tape containing the VAX/DS program into the TK tape drive on the system. At the console prompt, boot VAX/DS from the TK tape using the /R5:10 qualifier:

```
>>> BOOT /R5:10 CSA1
```

where CSA1 is the device name for the TK tape drive, and /R5:10 sets the boot flag designating the VAX/DS program (see Example 2–10). To run VAX/DS in standalone mode from a CD server, see Section 2.4.1.

To run VAX/DS in user mode under VMS, you use the RUN command under your operating system (see Example 2–11).

In either standalone or user mode, VAX/DS functions the same way. Typically a diagnostic running in user mode provides less detailed results than one running in standalone mode. For more information on VAX/DS, see the documents listed in Table 2–8.

2.4.3 Sample VAX/DS Session

When you run the VAX/DS programs, run the system autosizer program EVSBA first. This program, which takes several minutes to execute, will save you time as you proceed with other tests. Certain conditions cause the generation of an unexpected trap or interrupt. Use the method shown to avoid these conditions.

Example 2–12: Sample VAX/DS Session, Part 1 of 3

```
>>> SET BOOT DIAG /XMI:B /R5:10  DUA0        !booting through a KDM70

>>> BOOT DIAG❶

             [self-test results print]

Loading system software.

                    VAX DIAGNOSTIC SOFTWARE
                          PROPERTY OF
                 DIGITAL EQUIPMENT CORPORATION
              ***CONFIDENTIAL AND PROPRIETARY***

Use Authorized Only Pursuant to a Valid Right-to-Use License
Copyright, Digital Equipment Corporation, 1990.  All Rights Reserved.

DIAGNOSTIC SUPERVISOR. ZZ-ERSAA-14.X-NNNN  1-JAN-1990 12:01:45
DS> RUN EVSBA❷

.. Program: EVSBA - AUTOSIZER   level 3   X6.6, revision 6.6, 3 tests,
   at 17:52:20.21.

.. End of run, 0 errors detected, pass count is 1,
   time is 31-DEC-1989 17:55:07.02
```

❶ The SET BOOT command stores a nickname for a set of parameters to the BOOT command. (The lower key switch on the control panel must be set to Update when this command is issued.) This BOOT command loads VAX/DS from disk. Alternatively, you can use the command *BOOT/R5:10 CSA1* if you have a TK70 as a load device, or you can boot over the Ethernet from a CD server. For more information on the BOOT and SET BOOT commands, see any of the VAX 6000 *Owner's Manuals*.

❷ The off-line autosizer program EVSBA identifies hardware on your system and builds a database for the VAX Diagnostic Supervisor. The autosizer eliminates the need for you to type in the name and characteristics of the hardware you intend to test under VAX/DS with level 3 diagnostic programs.

Example 2–13: Sample VAX/DS Session, Part 2 of 3

```
DS> SHOW DEV❸

_DWMBB1 DWMBB   HUB     61F00000  XMI node # (1,2,3,4,B,C,D,E) =0000000E(X)
BI Node Number (HEX)=00000001(X)
_DUA    KDB50   _DWMBB1 7C008000  BI Node Number (HEX)=00000004(X)
_DJA23  RA60    _DUA    7C500000
_KA0    <REFERENCE>(XRP)   HUB     61880000  XMI Node ID=00000001(X)
_KA1    <REFERENCE>(XRP)   HUB     61900000  XMI Node ID=00000002(X)
_DWMBB0 DWMBB   HUB     61E80000  XMI node # (1,2,3,4,B,C,D,E) =0000000D(X)
BI Node Number (HEX)=00000001(X)
_TXA    DMB32   _DWMBB0 7A00A000  BI Node Number (HEX)=00000005(X)
_ETA0   DEBNI   _DWMBB0 7A00A000  BI Node Number (HEX)=00000006(X)
_DUA2   RA82    _DUA    7C500000
_DUA61  RA82    _DUA    7C500000
_DUA    TBK70   _DWMBB1 7C00C000  BI Node Number (HEX)=00000006(X)
_MUA0   TK70    _DUA    7C580000

DS> SELECT ALL❹
DS> SET TRACE
DS> SET EVENT 2
DS> RUN ERKMP

.. Program: ERKMP - <REFERENCE>(XRP) MP Exerciser, revision 1.0, 10 tests,
   at 18:11:41.25.
Testing:  _KA0 _KA1

                Booting Secondary Processor #02

Test 1: Memory Interlock Test
Test 2: Interprocessor Interrupt Test
Test 3: Write Error Interrupt Test
Test 4: Cache Invalidate Test
Test 5: XMI Bus Arbitration Test
Test 6: XMI Bus Arbitration Collision Test
Test 7: XMI Lockout Test
Test 8: Cache Coherency Test
Test 9: XMI Suppress Assertion Test
Test 10: Multiprocessor Exerciser
.. End of run, 0 errors detected, pass count is 1,
   time is 31-DEC-1989 18:16:24.49

DS> ^P❺
?02 External halt (CTRL/P, break, or external halt)
    PC  = 00027056
    PSL = 00000200
    KSP = 0004D210
```

❸ The autosizer builds a table of devices for VAX/DS that can be printed by using the SHOW DEVICE/BRIEF command at the DS> prompt. The command lists system devices, similar to the SHOW CONFIGURATION command in console mode.

❹ When preparing to run a diagnostic, the SELECT ALL command selects all devices listed in ❸ and the SET TRACE command enables printing of test numbers and names when the diagnostic runs. If you run another diagnostic after this one and you want the tests traced, you will need to issue the SET TRACE command again. The SET EVENT 2 command disables some informational messages printed by this particular diagnostic.

❺ An external halt causes VAX/DS to print the contents of the program counter, the processor status longword, and the stack pointer. Since VAX/DS was called from console mode, the console prompt is returned.

In a <REFERENCE>(rigel) system an external halt can, in some cases, cause an unexpected trap or interrupt through SCB vector 60. The remainder of this example shows how to avoid this condition by using the CLEAR EXCEPTION command and what happens if the condition is not cleared by the console. The condition does **NOT** apply to the <REFERENCE>(hyperion).

Example 2–14: Sample VAX/DS Session (Model 400 Only), Part 3 of 3

```
>>> CLEAR EXCEPTION ❻
XBER  = 00000041     ! no error bits set
XFADR = 61880008     ! no error bits set
RCSR  = 01240001     ! no error bits set

>>> SHOW CONFIGURATION ❼
      Type           Rev
  1+  KA64A   (8082) 0007
  3+  KA64A   (8082) 0007
  A+  MS65A   (4001) 0002
  B+  MS65A   (4001) 0002
  E+  DWMBB/A (2002) 0001

  XBI E
  1+  DWMBB/B (210F) 000A
  2+  CIBCA   (0108) 41C1
  4+  KDB50   (010E) 0F1C
  5+  DMB32   (0109) 210B
  6+  TBK70   (410B) 0307
  8+  DEBNI   (0118) 0100

>>> CLEAR EXCEPTION ❽
XBER  = 8001B041     ! error bits set
XFADR = 61900000     ! error bits set
RCSR  = 012C0011     ! error bits set

>>> CONTINUE ❾              ! No errors seen DS> appears
DS>
DS> ^P
?02 External halt (CTRL/P, break, or external halt)
    PC  = 0002704A
    PSL = 00000204
    KSP = 0004D210

>>> SHOW CONFIGURATION
   .
   .
   .
>>> CONTINUE ❿              ! Errors seen before DS> appears

?? Unexpected trap or interrupt thru SCB vector 0060
PC at error:            0002704A(X)
PSL at error:           00000004(X)  ; CUR=KERNEL,PRV=KERNEL,IPL=00(X),Z
User return PC:         none found!
DS> CONTINUE
.. Continuing from 0002704A

DS>
```

❻ The CLEAR EXCEPTION command prints the contents of three registers (XBER, XFADR, and RCSR) and then clears their error bits, if set. No error bits have been set at this point.

❼ The SHOW CONFIGURATION command attempts to examine unused address space, creating errors.

❽ Issuing the CLEAR EXCEPTION command again shows the contents of the three registers with error bits set. These error bits are then cleared by the command. When VAX/DS is halted as it was in ❺ of part 2 of this example and a command is issued that causes errors, as in ❼, the CLEAR EXCEPTION command must be issued before issuing the CONTINUE command to resume the VAX/DS session.

❾ VAX/DS is halted and SHOW CONFIGURATION is issued, again creating errors. (The response to SHOW CONFIGURATION is the same as shown in ❼.)

❿ This time the CONTINUE command is issued ❿ without first issuing the CLEAR EXCEPTION command. Since error bits were not cleared, VAX/DS attempts to perform its error recovery procedures, and an interrupt occurs. This is normal behavior for a <REFERENCE>(rigel) system in these circumstances.

2.4.4 VAX/DS Diagnostics

Table 2–9 lists the VAX Diagnostic Supervisor tests available for both Model 400 and Model 300 systems.

Table 2–9: VAX Diagnostic Supervisor Programs

Diagnostic	Level	Diagnostic Title
ERSAA[1]		<REFERENCE>(rigel) Diagnostic Supervisor
ELSAA[2]		<REFERENCE>(hyperion) Diagnostic Supervisor
EVSBA	3	VAX Standalone Autosizer
EVSBB	1	VAX Diagnostic Online Autosizer
		VAX 6000 EEPROM Update Utility
EVUCA	3	VAX 6000 EEPROM Update Utility
		<REFERENCE>(xyp)-Specific Diagnostics
ELKAX[2]	3	Manual Tests (5–6 min)
ELKMP[2]	3	Multiprocessor Exerciser (2 min—quick) (4 min—default)
		<REFERENCE>(XRP)-Specific Diagnostics
ERKAX[1]	3	Manual Tests (5–6 min[2])
ERKMP[1]	3	Multiprocessor Exerciser (2 min—quick) (4 min—default)
		<REFERENCE>(xrv)-Specific Diagnostics

[1]Diagnostic software with file names beginning with ER are tests created specifically for the <REFERENCE>(rigel) system. This software is not transportable.

[2]Diagnostic software with file names beginning with EL are tests created specifically for the <REFERENCE>(Hyperion) system.

Table 2–9 (Cont.): VAX Diagnostic Supervisor Programs

Diagnostic	Level	Diagnostic Title
		<REFERENCE>(xrv)-Specific Diagnostics
EVKAG	2	VAX Vector Instruction Exerciser, Part 1 (1 1/2 min—quick) (16 min—default)
EVKAH	2	VAX Vector Instruction Exerciser, Part 2 (1 min—quick) (18 min—default)
		VAX CPU Cluster Exerciser
EVKAQ	2	VAX Basic Instructions Exerciser, Part 1
EVKAR	2	VAX Basic Instructions Exerciser, Part 2
EVKAS	2	VAX Floating-Point Instruction Exerciser, Part 1
EVKAT	2	VAX Floating-Point Instruction Exerciser, Part 2
EVKAU	3	VAX Privileged Architecture Instruction Test, Part 1
EVKAV	3	VAX Privileged Architecture Instruction Test, Part 2
		XMA Online Memory Diagnostic
EVKAM	2R	VAX Memory User Mode Test
		DEMNA Diagnostics
EVDYE	2R	DEMNA NI Functional Diagnostic
EVGDB	2	DEMNA EEPROM Update Utility
EVDWC	2R	VAX NI Exerciser
		KDM70 Diagnostics
EVRAE	2R	Generic MSCP Disk Exerciser
EVRLJ	3	VAX UDA50/KDB50/KDM70 Exerciser
EVRLM	3	KDM70 EEPROM Update Utility
EVRLN	3	DUP Control Program

Table 2–9 (Cont.): VAX Diagnostic Supervisor Programs

Diagnostic	Level	Diagnostic Title
		CIXCD Diagnostics
EVGAA	3	CI Functional Test Part I
EVGAB	3	CI Functional Test Part II
EVGAC	3	Standalone CI Exerciser
EVXCI	1	VAX CI Exerciser
EVGEA	3	CIXCD Functional Test
EVGEB	3	CIXCD EEPROM Update Utility
		KDB50 Diagnostics
EVRLF	3	UDA50/KDB50 Basic Subsystem Diagnostic
EVRLG	3	UDA50/KDB50 Disk Drive Exerciser
EVRLB	3	VAX RAxx Formatter
EVRLJ	3	VAX UDA50-A/KDB50 Subsystem Exerciser
EVRLK	3	VAX Bad Block Replace Utility
EVRLL	3	Disk Drive Internal Error Log Utility
EVRAE	2R	VAX Generic MSCP Disk Exerciser
		DEBNA Diagnostics
EVDYD	2R	DEBNA Online Functional Diagnostic
EVDWC	2R	VAX NI Exerciser
		DEBNI Diagnostics
EVDYE	2R	DEBNI Online Functional Diagnostic
EVDWC	2R	VAX NI Exerciser
		TBK Diagnostic
EVMDA	2R	TK Data Reliability Exerciser

Table 2–9 (Cont.): VAX Diagnostic Supervisor Programs

Diagnostic	Level	Diagnostic Title
		CIBCA-AA Diagnostics
EVGCA	3	T1015 Repair Level Diagnostic, Part 1
EVGCB	3	T1015 Repair Level Diagnostic, Part 2
EVGCC	3	T1015 Repair Level Diagnostic, Part 3
EVGCD	3	T1015 Repair Level Diagnostic, Part 4
EVGCE	3	T1025 Repair Level Diagnostic
EVGAA	3	CI Functional Diagnostic, Part 1
EVGAB	3	CI Functional Diagnostic, Part 2
EVGDA	3	CIBCA EEPROM Program and Update Utility
EVXCI	1	VAX CI Exerciser
		CIBCA-BA Diagnostics
EVGEE	3	CIBCA-BA Repair Level Diagnostic, Part 1
EVGEF	3	CIBCA-BA Repair Level Diagnostic, Part 2
EVGEG	3	CIBCA-BA Repair Level Diagnostic, Part 3
EVGAA	3	CI Functional Diagnostic, Part 1
EVGAB	3	CI Functional Diagnostic, Part 2
EVGDA	3	CIBCA EEPROM Program and Update Utility
EVXCI	1	VAX CI Exerciser
		KLESI-B/TU81 Diagnostics
EVMBA	2R	VAX TU81 Data Reliability Diagnostic
EVMBB	3	VAX Front-End/Host Functional Diagnostic
		DHB32 Diagnostics
EVDAR	3	DHB32 Diagnostics
EVDAS	2R	DHB32 Macrodiagnostics

Table 2–9 (Cont.): VAX Diagnostic Supervisor Programs

Diagnostic	Level	Diagnostic Title
		DMB32 Diagnostics
EVAAA	2R	VAX Line Printer Diagnostic
EVDAJ	2R	DMB32 Online Asynchronous Port Test
EVDAK	3	DMB32 standalone Functional Verification
EVDAL	2R	DMB32 Online Synchronous Port Test
EVDAN	2R	DMB32 Online Data Communications Link
		DRB32 Diagnostics
EVDRH	3	DRB32-M, -E Functional Diagnostic
EVDRI	3	DRB32-W Functional Diagnostic
		TM32 Diagnostics
EVMEA	2R	TM32 L2R Reliability Diagnostic
EVMEB	3	TM32 L3 Functional Diagnostic, Part 1
EVMEC	3	TM32 L3 Functional Diagnostic, Part 2
		UNIBUS Diagnostics
EVCBB	3	VAX DWBUA VAXBI to UNIBUS
EVDRB	2R	VAX DR11W Online Diagnostic
EVDRE	3	VAX DR11W Repair Level Diagnostic
EVDUP	3	DUP11 Repair Level, Part 1
EVDUQ	3	DUP11 Repair Level, Part 2
EVAAA	2R	VAX Line Printer Diagnostic (LP11)
		DSB32 Diagnostics
EVDAP	3	DSB32 Level 3 Diagnostic
EVDAQ	2R	DSB32 Level 2R Diagnostic

Table 2–9 (Cont.): VAX Diagnostic Supervisor Programs

Diagnostic	Level	Diagnostic Title
		RV20 Diagnostics
EVRVA	3	RV20 Level 3 Functional Diagnostic
EVRVB	2R	RV20 Level 2R Diagnostic

Chapter 3

<REFERENCE>(xyp) Scalar Processor

This chapter contains the following sections:

- KA62B Physical Description and Specifications
- Configuration Rules
- Functional Description
- Boot Processor
- Power-Up Sequence
- <REFERENCE>(xyp) Self-Test Results: Console Display
- <REFERENCE>(xyp) Self-Test Results: Module LEDs
- ROM-Based Diagnostics
  - <REFERENCE>(xyp) Self-Test RBD
  - CPU/Memory Test RBD
  - Second-Level Cache RBD
- VAX/DS Diagnostics
- Machine Checks
- Console Commands
- CPU Replacement in Single CPU System
  - Using the RESTORE Command
  - Using SET Commands
- CPU Replacement in Multiple CPU Systems
- How to Add a New Processor
- Patching the EEPROM with EVUCA
- <REFERENCE>(XYP) Registers

3.1 KA62B Physical Description and Specifications

The <REFERENCE>(xyp) is a single-module VAX processor based on a single CPU chip and a floating-point accelerator chip. The module designation is T2011-YA. Features of the module are shown in Figure 3–1.

Figure 3–1: <REFERENCE>(XYP) Module

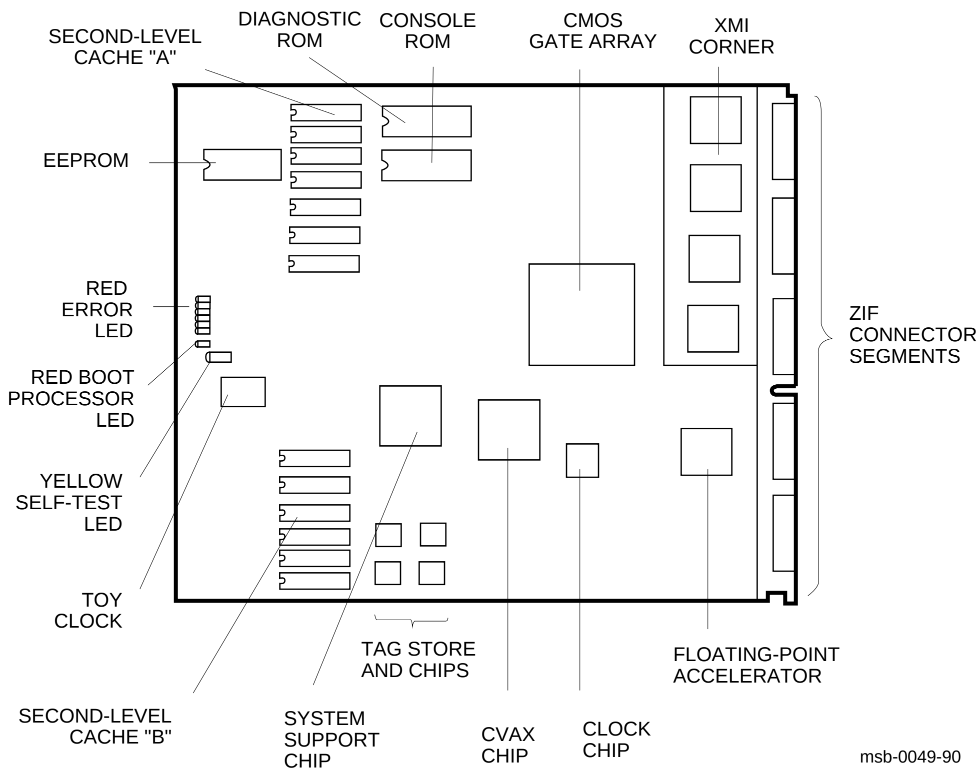


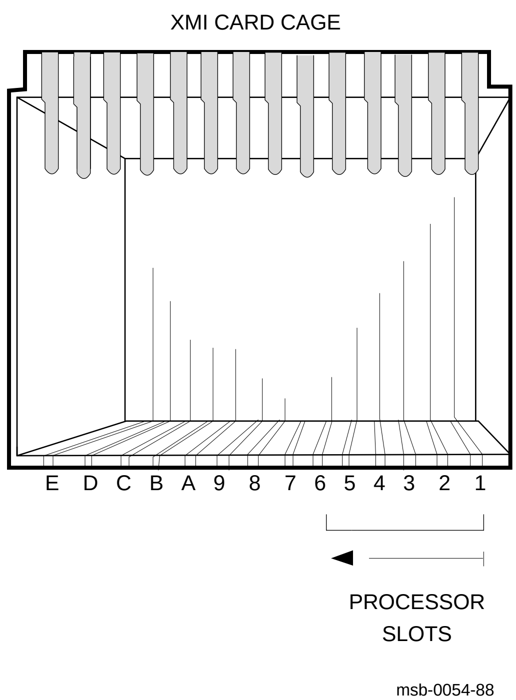
Table 3–1: KA62B Specifications

Parameter	Description
Module Number:	T2011-YA
Dimensions:	23.3 cm (9.2") H x 0.6 cm (0.25") W x 28.0 cm (11.0") D
Temperature:	
Storage Range	-40°C to 66°C (-40°F to 151°F)
Operating Range	5°C to 50°C (41°F to 122°F)
Relative Humidity:	
Storage	10 to 95% noncondensing
Operating	10 to 95% noncondensing
Altitude:	
Storage	Up to 4.8 km (16,000 ft)
Operating	Up to 2.4 km (8000 ft)
Current:	8.2A at +5V
Power:	41W
Cables:	None
Diagnostics:	ROM-based diagnostics (0, 1, and 4) VDS diagnostics, see Section 3.9.

3.2 <REFERENCE>(xyp) Configuration Rules

The <REFERENCE>(xyp) module will operate in any slot. Processors usually go on the right, beginning with slot 1. The maximum number of <REFERENCE>(xyp)s in a <REFERENCE>(hyperion) is six.

Figure 3–2: Typical <REFERENCE>(xyp) Configuration



By convention, processors are placed in the right <REFERENCE>(XMI) slots, beginning with slot 1. Memories are placed in the middle slots, from slot A to slot 5 and then slots B and C. A VAXBI adapter, if installed, occupies slot E. If a processor is failing intermittently and you are working on a system remotely, you may want to leave the module in the system temporarily and prevent the operating system from using that processor.

To disable a CPU:

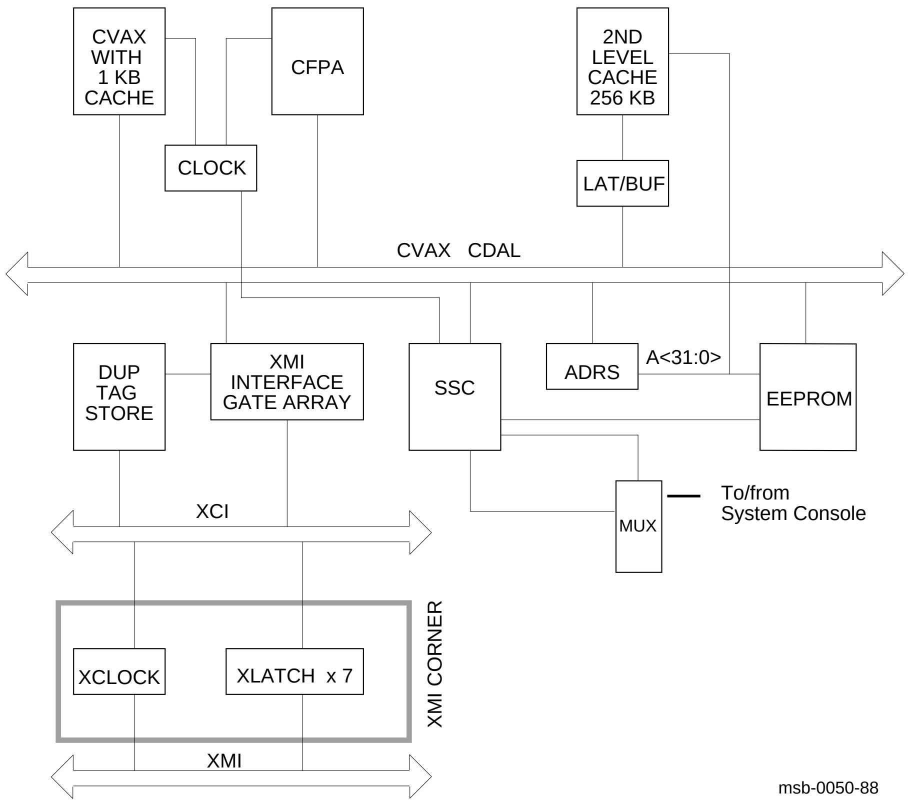
1. Enter console mode.
2. Use the command SET CPU/NOENABLE to remove the processor from the software configuration.
3. Reboot the operating system.

The console self-test display does not indicate that a CPU has been disabled. However, the SHOW CPU command reports which CPUs have been disabled on the /NOENABLED line of its display.

3.3 KA62B Functional Description

The <REFERENCE>(xyp) processor has four functional sections (see Figure 3–3): the CPU section, the second-level cache, the XMI interface, and the console and diagnostics sections.

Figure 3–3: <REFERENCE>(xyp) Block Diagram



The CPU section includes:

- The CVAX processor chip, which supports the VAX Base Instruction Group and data types. It has full VAX memory management including demand paging and 4 Gbytes of virtual memory. The CPU chip includes the first-level cache, for I-stream (instruction) storage only. First-level cache is 1 Kbyte, organized with 128 tags. Cache is write-through, two-way associative, and is filled eight bytes at a time.
- A floating-point accelerator chip, which supports the VAX Base Instruction Group floating-point instruction set. Data types supported by the hardware are D_floating, F_floating, and G_floating.
- The clock chip includes a VAX standard time-of-year (TOY) clock with access to battery backup, an interval timer with 10 ms interrupts, and two programmable timers.

The second-level cache is for both I-stream and D-stream (data) storage. Second-level cache is 256 Kbytes, organized with 4096 tags. Cache is write-through and direct-mapped. If a processor read misses an entry in the cache, or if the entry is invalid, the XMI gate array reads the data from main memory. Cache is filled 32 bytes at a time; the first longword read satisfies the processor's read request.

The <REFERENCE>(XMI) interface includes:

- An octaword write buffer that decreases bus and memory controller bandwidth needs by packing writes into larger, more efficient blocks prior to sending them to main memory.
- Hexword cache fill logic that loads the second-level cache with eight longwords of data on each cache miss.
- <REFERENCE>(XMI) write monitoring logic that uses a duplicate tag store to detect when another <REFERENCE>(XMI) node writes a memory location that is cached on this processor. Then the gate array invalidates the corresponding entry in the second-level cache.
- Full set of error recovery and logging capabilities.

Example 3–1: ROM and EEPROM Version Numbers

```
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   .   A   .   M   M   M   M   .   .   P   P   P   P       TYP
    o   .   +   .   +   +   +   +   .   .   +   +   +   +       STF
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD
    .   .   .   .   .   .   .   .   .   .   +   +   +   +       ETF
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD

.   .   .   .   .   .   .   .   .   +   .   +   .   +   +   .   XBI E +

    .   .   .   .  A4   A3  A2  A1  .   .   .   .   .   .       ILV
    .   .   .   .  64   64  64  64  .   .   .   .   .   .       256Mb

ROM = 6.0          EEPROM = 2.0/6.1      SN = SG01234567
       ❶                     ❷      ❸
```

The console and diagnostics sections include:

- A console read-only memory (ROM). This ROM contains code to initialize and boot the system and execute console commands. The last line of the self-test display shows the ROM version. In this example, the callout ❶ indicates that the console ROM is version 6.0.
- A diagnostic ROM, which contains the power-up self-test and extended diagnostics. The diagnostic ROM has the same version number as the console ROM ❶.
- An electrically-erasable, programmable ROM (EEPROM), which contains system parameters and boot code. You can modify the parameters with the console SET commands. Patches read into a special area of the EEPROM patch the console and diagnostic code in ROM. A program run under the VAX Diagnostic Supervisor is used to patch the EEPROM. See Section 3.15, Patching the EEPROM with EVUCA.

  The last line of the self-test display shows two EEPROM version numbers. The first number ❷ indicates the format version of the EEPROM. This version is changed only when the internal structure of the EEPROM is modified.

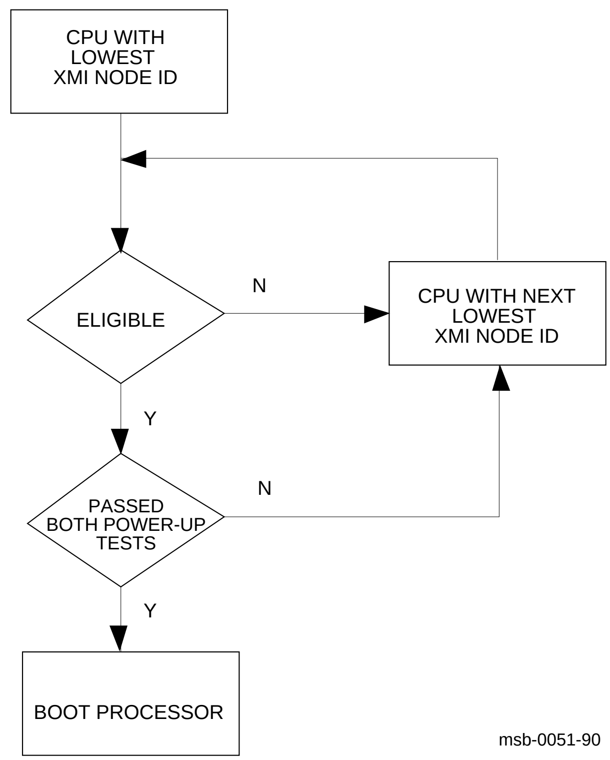
  The second number ❸ is the revision of ROM patches that have been applied to the EEPROM. The major number in this revision (before the decimal point) corresponds to the major number of the ROM revision ❶. The minor number indicates the actual patch revision. In this example, the EEPROM has been patched once for console ROM 6 and the patch revision is .1 .

  The location of the two ROMs and the EEPROM can be seen in Figure 3–1, Section 3.1.
- A system support chip (SSC) contains circuits for writing the EEPROM, controlling the console, and timer support. On the module, the red LED next to the yellow LED is controlled by the SSC. When the power-up tests have completed successfully, the SSC on the boot processor turns off this LED.

3.4 Boot Processor

The <REFERENCE>(hyperion) system is designed so that all processors share system resources equally. Because only one processor can boot the system or use the console at any given time, this processor is designated the primary or boot processor. The others are called secondary processors. The boot processor is selected during the power-up sequence.

Figure 3–4: Selection of Boot Processor



Using boot code stored in its EEPROM, the boot processor reads the boot block from a specified device. Booting may be triggered by a command issued to the boot processor from the console, or by a system reset with the bottom key switch in the Auto Start position.

The boot processor also communicates with the system console, using the common console lines on the backplane. When you change system parameters in the EEPROM using SET commands, the boot processor automatically copies the new values to the EEPROMs on the secondary processors. If you swap in a new <REFERENCE>(xyp) module, it should be configured as a secondary processor. Then you can either use the UPDATE command to copy the boot processor's entire EEPROM to the new secondary or use EVUCA and a series of SET commands to customize the EEPROM. Since UPDATE is slow, and can, in certain instances, render a processor unusable, the preferred method of updating a new processor placed in a system is to do it using SET commands. See Section 3.15 for information on running EVUCA and Section 3.12.2 on setting parameters.

CAUTION: *Using UPDATE can be dangerous because of revision mismatches. See Appendix C for information on what happens in mismatched cases.*

Usually the processor with the lowest <REFERENCE>(XMI) node number (which is also the lowest slot number) is selected as the boot processor. However, if this processor does not pass all its power-up tests, the next higher-numbered processor is selected. This is one way the boot processor can change.

The user also has control over boot processor selection with the SET CPU command. This command may declare a processor ineligible for selection. SET CPU can also select a boot processor explicitly.

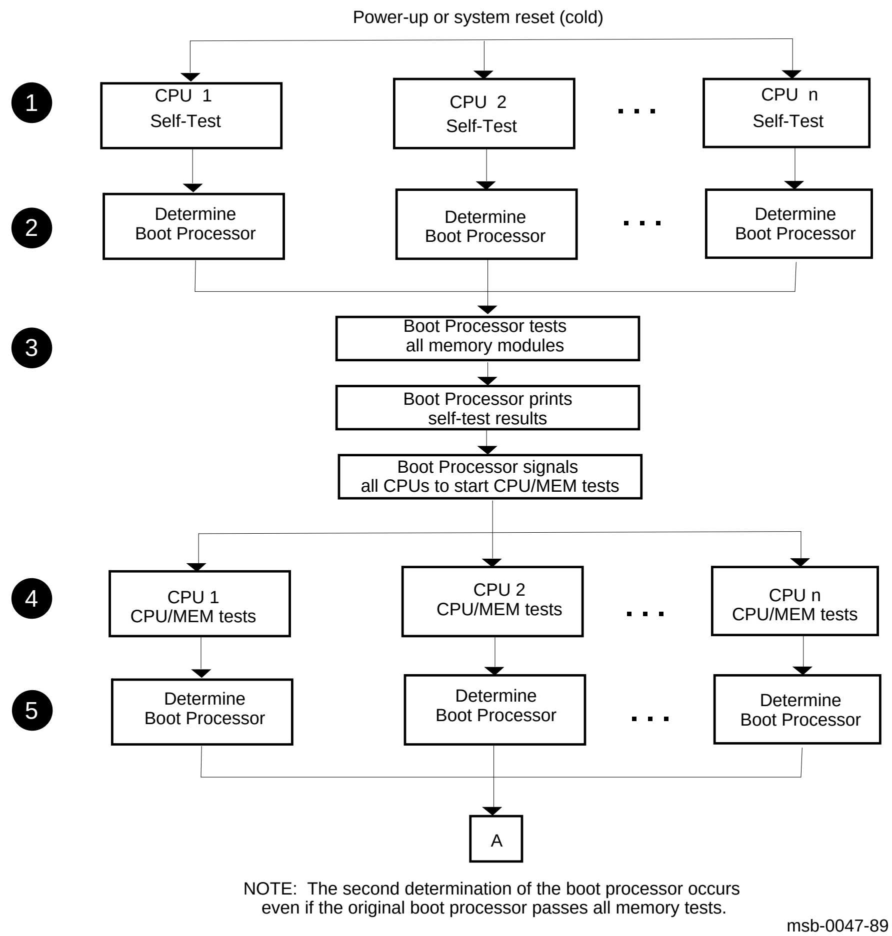
You can see the boot processor selection three ways:

- In the self-test display, the boot processor is indicated by a **B** on lines labeled **BPD**.
- In console mode, the command SHOW CPU displays the boot processor as "Current primary."
- In program mode, the bottom red LED (next to the larger yellow LED) is off on the boot processor module. It is lit on secondary processors.

3.5 Power-Up Sequence

Figure 3–5 shows the power-up sequence for <REFERENCE>(xyp) processors. All processors execute two phases of self-test, and a boot processor is selected. The boot processor tests VAXBI adapters, if present, and prints the self-test display.

Figure 3–5: <REFERENCE>(xyp) Power-Up Sequence, Part 1 of 2



❶ All CPUs execute their on-board self-tests at the beginning of the power-up tests. On the **STF** line of the self-test display, a plus sign (+) is shown for every module whose self-test passes (see Section 3.6).

❷ The boot processor is determined as described in Section 3.4. On the **first BPD line**, the letter **B** corresponds to the processor selected as boot processor. Because the processors have not yet completed their power-up tests, the designated processor may later be disqualified from being boot processor. For this reason, the BPD line appears twice in the self-test display.

❸ The boot processor tests all memory modules and then prints the results of self-test, lines **NODE, TYP, STF,** and **BPD** on the self-test display. The boot processor then signals all CPUs to start running the extended test.

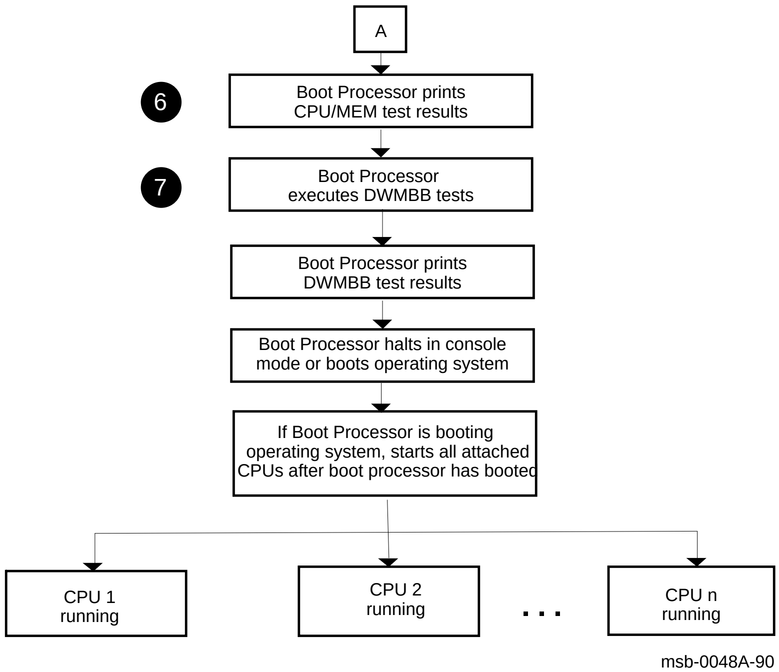
❹ All CPUs execute an extended test using the memories. On the **ETF** line of the self-test display, a plus sign (+) is shown for every module that passes extended test.

❺ If all CPUs pass the extended test, the original boot processor selection is still valid. Lines STF and ETF would be identical for all the processors.

The yellow LED is lit on all processor modules that pass both power-up tests. On the secondary processors, the red LED next to the yellow one is also lit. On the boot processor, this red LED is off (see Figure 3–7).

If the original boot processor fails the extended test (indicated by a minus sign (–) on line ETF), a new boot processor is selected. On the **second BPD line**, the letter **B** corresponds to the processor finally selected as boot processor.

Figure 3–6: <REFERENCE>(XYP) Power-Up Sequence, Part 2 of 2



❻ The boot processor prints the **ETF** line and the second **BPD** line of the self-test display. If none of the processors is successfully selected as the boot processor, no self-test results are displayed and the console hangs. You can identify this hung state by examining the LEDs on the processor modules (see Section 3.7). All yellow LEDs will be OFF. The group of six red LEDS will flash two alternate patterns, and in one pattern only the bottom light will be ON.

You can force a processor to become the boot processor so that it will display self-test results by typing the following:

```
>>n
```

where *n* is the CPU to become the boot processor.

❼ The boot processor tests the DWMBB if one is present. Test results are indicated on the lines labeled **XBI** on the self-test display. A plus sign (+) at the right means that the adapter test passed; a minus sign (–) means that the test failed.

3.6 <REFERENCE>(xyp) Self-Test Results: Console Display

You can check the <REFERENCE>(xyp) self-test results both in the self-test display and in the lights on the module. Pertinent information in the self-test display is shown in Example 3–2.

Example 3–2: Self-Test Results

```
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   .   A   A   M   M   M   M   .   .   P   P   P   P       TYP    ❶
    o   .   +   +   +   +   +   +   .   .   +   +   +   +       STF    ❷
    .   .   .   .   .   .   .   .   .   .   E   E   D   B       BPD    ❸
    .   .   .   .   .   .   .   .   .   .   +   +   +   -       ETF    ❹
    .   .   .   .   .   .   .   .   .   .   E   B   D   E       BPD    ❺

.   .   .   .   .   +   +   .   .   +   .   +   .   +   +   .   XBI E +


    .   .   .   .  A4  A3  A2  A1   .   .   .   .   .   .       ILV
    .   .   .   .  64  64  64  64   .   .   .   .   .   .       256Mb

ROM = 6.0         EEPROM = 2.0/6.0      SN = SGO1234567    ❻
>>>
```

❶ The second line in the self-test display indicates the type (TYP) of module at each <REFERENCE>(XMI) node. Processors are type P. In this example, processors are at nodes 1, 2, 3, and 4.

❷ The third line shows self-test fail status (STF), which are the results of on-board self-test. Possible values for processors are:

+ (pass)
– (fail)

All processors passed self-test in this example.

❸ The BPD line indicates boot processor determination. When the system completes on-board self-test, the processor with the lowest ID number that passes self-test and is eligible (designated by an E on the BPD line) is selected as boot processor — in this example, the processor at node 1.

The results on the BPD line indicate:

- B — The boot processor
- E — Processors eligible to become the boot processor
- D — Processors ineligible to become the boot processor

❹ During extended testing (ETF) all processors run a test, which includes reading and writing memory and using the cache. On line ETF, results are reported for each processor in the same way as on line STF—a plus sign indicates that extended self-test passed and a minus sign that extended test failed. In this example, the processor at node 1 (originally selected boot processor) failed the CPU/memory extended test.

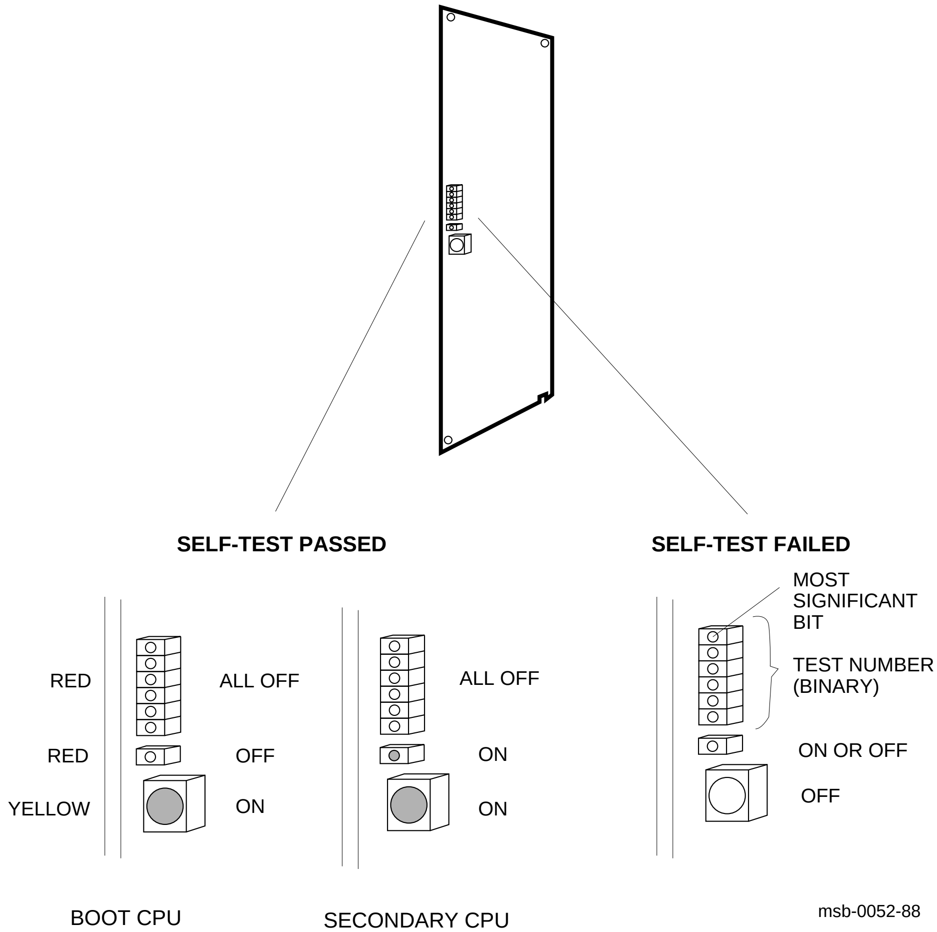
❺ Another BPD line is displayed, because it is possible for a different CPU to be designated boot processor before the system actually boots. This occurs in this example, because the processor at node 1 failed the extended test. The lowest-numbered processor that passed both tests is the processor at node 2. However, a previous SET CPU/NOPRIMARY command has made this processor ineligible to be boot processor (indicated by the designation D on the BPD line). Therefore, the processor at node 3 is designated boot processor.

❻ The bottom line of the self-test display shows the ROM and EEPROM version numbers and the system serial number.

3.7 <REFERENCE>(xyp) Self-Test Results: Module LEDs

You can check <REFERENCE>(XYP) self-test results both in the self-test display and in the lights on the module. As shown in Figure 3–7, if self-test passes, the large yellow LED is on.

Figure 3–7: <REFERENCE>(xyp) LEDs After Power-Up Self-Test



NOTE: *The two processors in this book have different LED patterns. Refer to Section 4.7 for interpreting <REFERENCE>(XRP) LEDs.*

The large yellow LED at the bottom of the LED array is ON if self-test passes and OFF if self-test fails. (Here self-test means both the on-board power-up test and the extended CPU/memory test.)

On the boot processor module, the red LED next to the yellow is OFF. This LED, which is controlled by the SSC chip, is ON on all the secondary processors.

The six red LEDs on top are all OFF if self-test passes. If any of the LEDs is ON, self-test failed and the LEDs contain a binary error code. The error code corresponds to the number of the test that failed in hexadecimal. In the six error LEDs, the most significant bit is at the top, but the lights have a reverse interpretation — a bit is ONE if the light is OFF.

For example, assume a processor fails self-test (yellow LED is OFF), has a minus sign (–) on its STF line, and shows the following pattern in its top six LEDs:

```
                TOP
(MSB)  on      0
       on      0  =  0

       off     1
       off     1  =  E
       off     1
       on      0
                BOTTOM
```

The failing test number decodes to 001110 (hex 0E, decimal 14). If you then ran the ROM-based diagnostic 0 with TRACE ON, the last test number you would see displayed is T0014.

When any of the six red error LEDs is lit, a failure has occurred during testing. There are three sets of power-up self-test: KA62B power-up tests, CPU/memory tests, and optionally the VAXBI adapter (DWMBB) tests. Interpretation of the processor board error LEDs depends on which set of tests was running at the time of failure. Table 3–2 decodes the LED conditions.

Processor error LEDs can also indicate failures of memories or VAXBI adapters.

Table 3–2: KA62B Error LEDs

Yellow LED	Error LEDs (hex)	Diagnostic and Test No. (decimal)	Device Failing	Self-Test Line
OFF	1–22	Power-up self-test (equivalent to RBD 0) T0001–T0034	KA62B	STF
OFF	25–38	CPU/memory test - Memory 1 (equivalent to RBD 1) T0001–T0020	MS65A 1 (module with lowest XMI node number)	ETF
OFF	39	CPU/memory test - Memory 2 T0002 (equivalent to ST 1/T=2)	MS65A 2	ETF
OFF	3A	CPU/memory test - Memory 3 T0002 (equivalent to ST 1/T=2)	MS65A 3	ETF
OFF	3B	CPU/memory test - Memory 4 T0002 (equivalent to ST 1/T=2)	MS65A 4	ETF
OFF	3C	CPU/memory test - Memory 5 T0002 (equivalent to ST 1/T=2)	MS65A 5	ETF
OFF	3D	CPU/memory test - Memory 6 T0002 (equivalent to ST 1/T=2)	MS65A 6	ETF
OFF	3E	CPU/memory test - Memory 7 T0002 (equivalent to ST 1/T=2)	MS65A 7	ETF
OFF	3F	CPU/memory test - Memory 8 T0002 (equivalent to ST 1/T=2)	MS65A 8	ETF
ON (boot processor)	1–1A	DWMBB test (equivalent to RBD 2) T0001–T0026 See Table 7–5.	DWMBB	XBI

If a processor's yellow LED is OFF and the six red LEDs show an error code in the range 1–22 (hex), the power-up self-test failed and the processor board is bad. On the self-test console display, the processor shows a minus sign (–) on the STF line.

After the power-up tests, each processor runs the CPU/memory tests. If a test fails, the processor shows a minus sign (–) on the ETF line of the self-test console display. With the first memory, LED error codes are numbered from 25 to 38 (hex), to distinguish them from the power-up tests. For example, assume that a processor fails self-test (yellow LED is OFF) and shows the following pattern in the error LEDs:

```
                TOP
(MSB)   off   1
        on    0   =   2

        on    0
        off   1   =   7
        off   1
        off   1
                BOTTOM
```

The failing test number decodes to 100111 (hex 27), which corresponds to the third CPU/memory test. If you then ran the ROM-based diagnostic 1 with TRACE ON, the last test number you would see displayed is T0003.

Each processor, after testing with the first memory, runs the CPU/memory test on every other good memory module. (However, only CPU/memory test T0002 is run.) If a failure occurs, the memory module is probably bad, although the processor's yellow light is OFF and the memory module's yellow light is ON. If several processors fail on the same memory, the memory is certainly bad. Try using SET MEMORY to configure the bad module out of the interleave set. For error codes higher than 38, consult Table 3–2 to determine the failing memory.

The last series is the DWMBB tests. If one fails, the top six LEDs contain an error code, although the processor's yellow self-test LED is ON (because the CPU itself has passed). The failing error code (converted to decimal) corresponds to a test number in Table 7–5. Note that only the boot processor performs the DWMBB tests, so the red LED next to the yellow LED is OFF.

3.8 ROM-Based Diagnostics

The <REFERENCE>(xyp) ROM contains five diagnostics, which you run using the boot processor's RBD monitor program described in Chapter 2. RBD 0, 1, and 4 test the boot processor. RBD 2 tests VAXBI adapters, and RBD 3 tests memories.

Table 3–3: KA62B ROM-Based Diagnostics

Diagnostic	Test
0	<REFERENCE>(xyp) power-up test
1	<REFERENCE>(xyp) extended CPU/memory test
2	<REFERENCE>(XBI_PLUS) tests
3	Memory test
4	Second-level cache test

RBD 0 is the same as the power-up self-test. It is useful for running several passes when a processor fails self-test intermittently. Section 3.8.1 shows an example and lists the tests.

RBD 1 is the same as the extended CPU/memory test. It is useful for running several passes when a processor fails self-test intermittently. Section 3.8.2 shows an example and lists the tests.

RBD 2 is the set of tests that the boot processor runs for each DWMBB VAXBI-to-XMI adapter when the system is powered on. (The DWMBB has no on-board self-test of its own.) The diagnostic reports whether the <REFERENCE>(XBI_PLUS) passed and whether each I/O device on that adapter's VAXBI bus passed its own self-test.

RBD 3 is a set of memory tests that sizes and runs extended tests on all of memory. Section 6.11 and Section 6.12 show an example of memory RBDs and list the tests.

RBD 4 is a set of exhaustive cache tests. It is not meant to be used frequently—you must explicitly request each test to run. The complete set of seven tests takes over one hour.

For a detailed explanation of the diagnostic printout, see Chapter 2.

3.8.1 <REFERENCE>(xyp) Self-Test — RBD 0

RBD 0 is equivalent to the <REFERENCE>(xyp) power-up self-tests.

Example 3–3: <REFERENCE>(xyp) Self-Test RBD 0

```
>>>                         ! Console program prompt.
>>> T/R                     ! Command to enter RBD monitor program.
RBD3>                       ! RBD monitor prompt, where 3 is the hexa-
                            ! decimal node number of the boot processor.
RBD3> START 0 /TRACE/HE     ! Runs the KA62B self-test on boot processor
                            ! Trace prints each test number; halt on error
                            ! Test results written to the console terminal:
;XCPST            6.0

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018  T0019  T0020
; T0021 ❶

;        F ❷        3     8001      1 ❸
;       HE       XDEV_ERR        1  T0021 ❶
;       21     00000000 00000001 00000000 00001420 1FFD074C 00040CD0 02
```

In the example above: ❶ test 21 failed, ❷ F indicates failure, and ❸ the diagnostic ran for one pass.

Table 3–4: <REFERENCE>(xyp) Power-Up Test — RBD 0

Test	Function
T0001	<REFERENCE>(xyp) ROM test
T0002	SSC Base Address Register test
T0003	SSC RAM test
T0004	SSC Configuration Register test
T0005	SSC Bus Timeout test
T0006	SSC Programmable Address Decode test
T0007	KA62B CSR1 test
T0008	SSC Output Port test

Table 3–4 (Cont.): <REFERENCE>(xyp) Power-Up Test — RBD 0

Test	Function
T0009	KA62B EEPROM test
T0010	SSC Interval Timer test
T0011	SSC Programmable Timers tests
T0012	SSC Console test
T0013	SSC TOY Clock test
T0014	CVAX Critical Path test
T0015	Cache Data RAM March test
T0016	Cache Mask Write test
T0017	Cache Data Parity RAM March test
T0018	CSR1:CPUD test
T0019	CFPA test
T0020	CFPA Critical Path test
T0021	XDEV Register test
T0022	XBER Register test
T0023	XFADR Register test
T0024	XGPR Register test
T0025	KA62B CSR2 Register test
T0026	XMI High Longword Data test
T0027	Interprocessor IVINTR test
T0028	Write Error IVINTR test
T0029	CNAK Read test
T0030	CNAK Write test
T0031	CNAK IP/WE IVINTR test
T0032	Multiple Interrupt test
T0033	Parity Error CNAK Read test

Table 3–4 (Cont.): <REFERENCE>(xyp) Power-Up Test — RBD 0

Test	Function
T0034	Parity Error CNAK Write test

3.8.2 CPU/Memory Test — RBD 1

RBD 1 is equivalent to the extended CPU/memory test.

Example 3–4: CPU/memory Test RBD 1

```
>>>Z 3                        ! This example uses the Z command
                              ! to connect processor 3 to the console
                              ! Note new console program prompt

3>> T/R                       ! Command to enter RBD monitor program

RBD3>                         ! RBD monitor prompt, where 3 is the hexa-
                              ! decimal node number of the processor
                              ! that is currently receiving your input.

RBD3> START 1 /TRACE/HE       ! Runs the CPU/memory RBD with trace; halt
                              ! on error.  Test results written to the
                              ! console terminal:

;CPUMEM       6.0

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018  T0019  T0020

         ❶                            ❷
;        P            3     8001        1
;000000000 00000000 00000000 00000000 00000000 00000000 00000000

RBD3>
```

In the example above:

❶ P means that the diagnostic ran successfully.

❷ One pass was completed.

Table 3–5: Extended CPU/Memory Test — RBD 1

Test	Function
T0001	Cache Disable test
T0002	Cache Invalidate test
T0003	Cache Read Fill test
T0004	Interlock Instruction Cache test
T0005	Longword Write WB0 test
T0006	Octaword Write WB0 test
T0007	WB Switch and Purge test
T0008	Octaword Write WB1 test
T0009	Write Buffer Read Tests
T0010	Hit WB0 test
T0011	Hit WB1 test
T0012	WB Mask Bit Byte Tests
T0013	WB Mask Bit Word Tests
T0014	Intlk Read/Unlck Write WB test
T0015	I/O Space Write WB test
T0016	Node Private Space WB test
T0017	WB Address test
T0018	WB Pending Purge test
T0019	Duplicate Tag Invalidate test
T0020	Duplicate Tag Address test

3.8.3 Second-Level Cache Test — RBD 4

RBD 4 tests the second-level cache. Note that no tests are run by default.

Example 3–5: Second-Level Cache Test RBD 4

```
>>> T/R                             ! Command to enter RBD monitor program

RBD1>                               ! RBD monitor prompt, where 1 is the
                                    ! hexadecimal node number of the processor
                                    ! that is currently receiving your input

RBD1> ST4/T=1:2/TR                  ! Specifically asks for tests 1 and 2 to
                                    ! be run with trace enabled

;SLCRBD       6.0

; T0001                                     ! Test 1 started
          ❶                          ❸
;       S        1     8001        1    ! Status message for test 1
;      XX SL_CACHE       XX    T0001

;00000000 00040000 00000000 00000000 00000000 00000000 00000000 00000000
                                        ! Print message showing current
                                        ! address and ending address
; T0002                                 ! Test 2 started
          ❶                       ❸
;       S        1     8001        1    ! Status message for test 2
;      XX SL_CACHE       XX    T0002

;00000000 01000000 00000000 00000000 00000000 00000000 00000000 00000000
                                        ! Print message showing current
                                        ! address and ending address.
;00000042 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                                        ! Print message showing the
                                        ! number of cache hits
          ❷                       ❸
;       P        1     8001        1
;00000000 00000000 00000000 00000000 00000000 00000000 00000000

RBD1> QUIT                              ! Exit the RBD monitor
```

❶ S denotes status message for individual test.

❷ P means that the entire diagnostic ran successfully.

❸ One pass was completed.

Table 3–6: Second-Level Cache Test — RBD4

Test	Function
T0001	Parity Error Test
T0002	Cache Coherency Test
T0003	Cache Invalidate Test
T0004	Self-Invalidate Test
T0005	Invalidate/Self-Invalidate Test
T0006	Self-Invalidate Scope Test
T0007	Cache Ram Parity Error Test
T0008	Passive Release Test

3.9 VAX/DS Diagnostics

The <REFERENCE>(xyp) software diagnostics that run under the VAX Diagnostic Supervisor (VAX/DS) are listed in Table 3–7. An example follows. See Section 2.4 for instructions on running the supervisor.

Table 3–7: <REFERENCE>(xyp) VAX/DS Diagnostics

Program	Description
EVSBA	VAX Standalone Autosizer
EVKAQ	VAX Basic Instructions Exerciser - Part 1
EVKAR	VAX Basic Instructions Exerciser - Part 2
EVKAS	VAX Floating Point Instruction Exerciser - Part 1
EVKAT	VAX Floating Point Instruction Exerciser - Part 2
EVKAU	VAX Privileged Architecture Instruction Test - Part 1
EVKAV	VAX Privileged Architecture Instruction Test - Part 2
ELKAX	Manual Tests
ELKMP	Multiprocessor Exerciser
EVUCA	VAX 6000 EEPROM Update Utility

Example 3–6: Running Standalone Processor Diagnostics

```
DS> RUN EVSBA   ❶
DS> SEL KA0     ❷
DS> RUN ELKAX   ❸
  [diagnostic messages]
DS> EXIT  ❹
```

❶ Run the standalone autosizer; then you do not need to attach devices to the supervisor explicitly. However, if you want to know the Attach command, enter HELP diagnostic_name ATTACH.

❷ The instruction and manual tests run on the boot processor. If the boot processor is the CPU with the lowest <REFERENCE>(XMI) node number (which is usually the case), issue the command to select KA0. The Diagnostic Supervisor numbers the processors consecutively. For example, if the <REFERENCE>(xyp) module with the second-lowest <REFERENCE>(XMI) node number were boot processor, you would select KA1.

❸ This example runs the manual tests (ELKAX), which include powerfail, machine check, restart, and EEPROM functions. The diagnostic prints messages, and you must manually intervene using console switches.

❹ Exit from VAX/DS.

3.10 Machine Checks

A machine check is an exception that indicates a processor-detected internal error. Figure 3–8 shows the parameters that are pushed on the stack in response to a machine check. Table 3–8 lists these parameters.

Figure 3–8: The Stack in Response to a Machine Check

BYTE COUNT (0010 hex)	SP:
MACHINE CHECK CODE	SP+4:
MOST RECENT VIRTUAL ADDRESS	SP+8:
INTERNAL STATE INFORMATION #1	SP+C:
INTERNAL STATE INFORMATION #2	SP+10:
PC	SP+14:
PSL	SP+18:

msb-0053-89

Table 3–8: Machine Check Parameters

Parameter	Value	Description
Machine check code (hex) (SP+4)	1	Floating-point protocol error
	2	Floating-point reserved instruction
	3	Floating-point unknown error
	4	Floating-point unknown error
	5	Process PTE in P0 space during TB miss flows
	6	Process PTE in P1 space during TB miss flows
	7	Process PTE in P0 space during M = 0 flows

Table 3–8 (Cont.): Machine Check Parameters

Parameter	Value	Description
	8	Process PTE in P1 space during M = 0 flows
	9	Undefined INT.ID value
	A	Undefined MOVCx state
	80	Memory read error
	81	SCB, PCB, or SPTE read error
	82	Memory write error
	83	SCB, PCB, or SPTE write error
Most recent virtual address (SP+8)	<31:0>	Current contents of VAP register
Internal state information #1 (SP+C)	<31:24>	Opcode
	<23:16>	1110, highest priority software interrupt <3:0>
	<15:8>	CADR<7:0>
	<7:0>	MSER<7:0>
Internal state information #2 (SP+10)	<31:24>	Most recent contents of SC register <7:0>
	<23:16>	11, state flags <5:0>
	<15:8>	Restart flag, 111, ALU CC flags <3:0>
	<7:0>	Offset from saved PC to PC at time of machine check
PC (SP+14)	<31:0>	PC at the start of the current instruction
PSL (SP+18)	<31:0>	Current contents of PSL

Machine checks are taken regardless of the current IPL. During a machine check the IPL is raised to 1F and the machine check frame is pushed onto the interrupt stack. If the machine check exception vector bits (<1:0>) are not both one, the operation of the processor is undefined.

3.11 Console Commands

Table 3–9 summarizes the console commands. The VAX 6000 Series Owner's Manual gives a full description of each command, its qualifiers, and examples.

Table 3–9: Console Commands

Command	Function
BOOT	Initializes the system, causing a self-test, and begins the boot program.
CONTINUE	Begins processing at the address where processing was interrupted by a CTRL/P console command.
DEPOSIT	Stores data in a specified address.
EXAMINE	Displays the contents of a specified address.
FIND	Searches main memory for a page-aligned 256-Kbyte block of good memory or for a restart parameter block.
HALT	Null command; no action is taken since the processor has already halted in order to enter console mode.
HELP	Prints explanation of console commands.
INITIALIZE	Performs a system reset, including self-test.
PATCH EEPROM	Patches console, diagnostic, and boot primitive code from a TK70.
REPEAT	Executes the command passed as its argument.
RESTORE EEPROM	Copies, if present, the TK tape's EEPROM contents to the EEPROM of the processor executing the command.
SAVE EEPROM	Copies to the TK tape, if present, the contents of the EEPROM of the processor executing the command.
SET BOOT	Stores a boot command by a nickname.
SET CPU	Specifies eligibility of processors to become the boot processor.
SET LANGUAGE	Changes the output of the console error messages between numeric code only (international mode) and code plus explanation (English mode).

Table 3–9 (Cont.): Console Commands

Command	Function
SET MEMORY	Designates the method of interleaving the memory modules; supersedes the console program's default interleaving.
SET TERMINAL	Sets console terminal characteristics in the EEPROM.
SHOW ALL	Displays the current value of parameters set.
SHOW BOOT	Displays all boot commands and nicknames that have been saved using SET CPU.
SHOW CONFIGURATION	Displays the hardware device type and revision level for each XMI and VAXBI node and indicates self-test status.
SHOW CPU	Displays the /ENABLED and /PRIMARY values for each node.
SHOW ETHERNET	Locates all Ethernet adapters on the system and displays their addresses.
SHOW MEMORY	Displays the memory lines from the system self-test, showing interleave and memory size.
SHOW TERMINAL	Displays the baud rate and terminal characteristics functioning on the console terminal.
START	Begins execution of an instruction at the address specified in the command string.
STOP	Halts the specified node.
TEST	Passes control to the self-test diagnostics; /RBD qualifier invokes ROM-based diagnostics.
UPDATE	Copies contents of the EEPROM on the processor executing the command to the EEPROM of the processor specified in the command string.
Z	Logically connects the console terminal to another processor on the <REFERENCE>(XMI) bus or to a VAXBI node.
!	Introduces a comment.

3.12 CPU Replacement in Single CPU Systems

There are two methods available to customize a processor in a single-processor system. The first uses the RESTORE command; the second sets the EEPROM by console commands.

3.12.1 Using the RESTORE Command

Use the RESTORE command to customize a replaced CPU in a single-processor system. Use this method only if the ROM revision of the spare is the same as the ROM revision of the CPU being replaced. Before you begin set the terminal baud rate to 1200.

Example 3–7: Replacing a Single Processor

```
F    E    D    C    B    A    9    8    7    6    5    4    3    2    1    0    NODE #

     A    .    A    .    M    M    .    .    .    .    .    .    .    P         TYP     ❻
     o    .    +    .    +    +    .    .    .    .    .    .    .    +         STF     ❼
     .    .    .    .    .    .    .    .    .    .    .    .    .    B         BPD
     .    .    .    .    .    .    .    .    .    .    .    .    .    +         ETF     ❼
     .    .    .    .    .    .    .    .    .    .    .    .    .    B         BPD

.    .    .    .    .    .    +    +    .    +    .    +    .    +    +    .    XBI E +

     .    .    .    .    A2   A1   .    .    .    .    .    .    .    .         ILV
     .    .    .    .    64   64   .    .    .    .    .    .    .    .         128Mb

ROM = 6.0 ❽        EEPROM = 2.0/6.0      SN = 0000000000 ❾

?4E System serial number has not been initialized ❾

>>> RESTORE EEPROM ⓬

?6D EEPROM Revision = 2.0/6.0
?6F Tape image Revision = 2.0/6.1
Proceed with update of EEPROM? (Y or N)

!optional - may need latest console/diag patches again  ⓭

>>> BOOT  ⓯
```

1. Turn the upper key switch straight up to the Off position (0).
2. Remove the defective CPU module.
3. Insert the new processor module.
4. Turn the lower key switch to Halt.
5. Turn the upper key switch to Enable.
6. Check the self-test display for the processor, indicated by a P on the TYP line (usually in slot 1). See Example 3–7 ❻.
7. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test. See Example 3–7 ❼.
8. Compare the ROM revision ❽ of the new processor with the ROM revision of the one you just replaced. If they are the same, continue with this procedure; otherwise use the second method described in Section 3.12.2.
9. You will see the following message: ❾

   ```
   ?4E System serial number has not been initialized
   ```
10. Turn the lower key switch to Update.
11. Mount the TK cartridge containing the most recent saved image of the old processor's EEPROM.[1]
12. Issue the console command RESTORE EEPROM ⓬ to copy all areas of the EEPROM except the module-specific area to the boot processor's EEPROM.
13. If any patches have been issued since the last save, use EVUCA to patch the EEPROM. See Section 3.15 for details.
14. Turn the lower key switch to the Auto Start position.
15. Boot the operating system ⓯. Booting will initialize the system and the EEPROM will be read. If the system console baud rate was not normally set at 1200, you will have to change the terminal baud rate back to its original value.

---

[1] When the system is installed or after maintenance, customer service engineers should save the EEPROM on a TK cartridge if available. The cartridge is left in the care of the customer. Subsequently, the EEPROM might have been changed and saved several times. This would normally happen following a PATCH operation.

3.12.2 Using SET Commands

When replacing the only processor module in a system that does not have a TK, you must enter console commands that customize the EEPROM. The Site Management Guide should contain this information. Before you begin set the terminal baud rate to 1200.

Example 3–8: Replacing a Single Processor

```
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   .   .   M   M   .   .   .   .   .   .   .   P       TYP    ❼
    +   +   .   .   +   +   .   .   .   .   .   .   .   +       STF    ❽
    .   .   .   .   .   .   .   .   .   .   .   .   .   B       BPD
    .   .   .   .   .   .   .   .   .   .   .   .   .   +       ETF    ❽
    .   .   .   .   .   .   .   .   .   .   .   .   .   B       BPD

    .   .   .   .   A2  A1  .   .   .   .   .   .   .   .       ILV
    .   .   .   .   64  64  .   .   .   .   .   .   .   .       128Mb

ROM = 6.0         EEPROM = 2.0/6.0      SN = 0000000000

?4E System serial number has not been initialized

>>> [ESC] (or [CTRL/3]) [DEL] SET SYSTEM SERIAL [RET] ❿

    ! Follow the prompts to set the serial number.

>>> SET BOOT DEFAULT /XMI:E DU0 ⓫

>>> SET LANGUAGE INTERNATIONAL ⓬
      OR
>>> SET LANGUAGE ENGLISH

>>> SET TERMINAL /[NO]SCOPE /SPEED:9600 /[NO]BREAK  ⓭

!optional - may need latest console/diag patches ⓮

>>> BOOT  ⓯
```

1. Turn the upper key switch straight up to the Off position (0).
2. Remove the defective CPU module.
3. Insert the new processor module.
4. The EEPROM on the new module has a default baud rate of 1200. Make sure your terminal is set to 1200.
5. Turn the lower key switch to Halt.
6. Turn the upper key switch to Enable.
7. Check the self-test display for the processor, indicated by a P on the TYP line. See ❼ in Example 3–8.
8. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test. See ❽.
9. Turn the lower key switch to Update.
10. To correct the error message, enter the SET SYSTEM SERIAL command (see ❿) and follow the command prompts. The serial number should be in the *Site Management Guide*. If you do not find it there, look on an old self-test display or on the tape located near the bottom of the left rear cabinet upright.
11. To set up the boot alternatives, enter the SET BOOT command supplying the parameters recorded in the *Site Management Guide*. You might want to check these with the system manager. ⓫
12. Use the SET LANGUAGE command to set up the language desired.⓬
13. Use the SET TERMINAL command to set the EEPROM terminal characteristics.⓭ Results of the SET TERMINAL command will take effect when the system is reset. Make sure the terminal itself is set to the same characteristics.
14. Use whatever console commands are necessary to completely customize the EEPROM to the customer's satisfaction.

    If any patches or boot primitives need to be selected, follow the procedure in Section 3.15. ⓮
15. Turn the lower key switch to the Auto Start position and boot the operating system. ⓯

3.13 CPU Replacement in Multiple CPU Systems

In a multiprocessing system replacing a processor requires customizing the new processor into the system. If the processor being replaced is the boot processor, then a few extra steps are required. Since different CPUs can have different ROM and EEPROM revisions in the same system, updating the EEPROM of any processor should be done using SET commands and EVUCA.

Example 3–9: Retrieving EEPROM Information

```
F    E    D    C    B    A    9    8    7    6    5    4    3    2    1    0    NODE #

     A    A    A    .    M    M    M    M    .    .    P    P    P    P         TYP     ❼
     +    +    +    .    +    +    +    +    .    .    +    +    +    +         STF     ❽
     .    .    .    .    .    .    .    .    .    .    E    E    E    B         BPD
     .    .    .    .    .    .    .    .    .    .    +    +    +    +         ETF     ❽
     .    .    .    .    .    .    .    .    .    .    E    E    E    B         BPD

     .    .    .    .   A4    A3   A2   A1   .    .    .    .    .    .         ILV
     .    .    .    .  128   128  128  128   .    .    .    .    .    .         512Mb

ROM = 6.0          EEPROM = 2.0/6.0      SN = 0000000000 or SGO1234567

?4F System serial number not initialized on primary processor. ❿
                           or
?2D For Secondary Processor 3
?59 System serial number mismatch.  Secondary processor has xxxxxxx

>>> SET CPU 3 ⓫

>>> [ESC] (or [CTRL/3]) [DEL] SHOW SYSTEM SERIAL ⓬

>>> SHOW ALL ⓭

     [System configuration and customization information prints]
```

1. Turn the upper key switch straight up to the Off position (0).
2. Remove the defective CPU module.
3. Insert the new processor module.
4. If the processor you are replacing is the first processor on the right (usually in slot one), the console will make it the boot processor. If this is the case, make sure that the console terminal baud rate is 1200, the default baud rate of the spare.
5. Turn the lower key switch to Halt.
6. Turn the upper key switch to Enable.
7. Check the self-test display for the processor, indicated by a P on the TYP line. See ❼ in Example 3–9.
8. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test. ❽
9. Turn the lower key switch to Update.
10. You will see either the ?4F message or the ?2D and ?59 messages.❿ If you see mismatch messages, you may have to patch the EEPROM using EVUCA. See Section 3.15. If you see the ?4F message, the boot processor has been replaced and you must execute the next step; otherwise, go to Step 12.
11. Issue the SET CPU *n* command so that the console is connected to a processor that has been in the system for some time. ⓫
12. Get the system serial number by entering the SHOW SYSTEM SERIAL command. ⓬
13. Issue the SHOW ALL command to get the customized boot parameters, interleave characteristics, terminal setup, and any other parameters. ⓭

If you collected the necessary information, you are now ready to customize the EEPROM.

Example 3–10: Customizing an EEPROM

```
>>> SET CPU 3  ⓮
>>> [ESC] (or [CTRL/3]) [DEL] SET SYSTEM SERIAL ⓯
    ! Follow the prompts to set the serial number.
>>> SET BOOT DEFAULT /XMI:E DU0 ⓰
>>> SET LANGUAGE INTERNATIONAL or ENGLISH ⓱
>>> SET TERMINAL /[NO]SCOPE /SPEED:9600 /[NO]BREAK  ⓲
>>> SET CPU/NOPRIMARY 3 ⓳
!optional - may need latest console/diag patches again  ⓴
>>> BOOT ㉑
```

14. Issue the SET CPU *n* command to connect the console to the primary or secondary processor you just replaced. ⓮
15. Issue the SET SYSTEM SERIAL ⓯ and follow the prompts to correct the error message ❿.
16. Use the SET BOOT command supplying the parameters displayed by the SHOW ALL command to set up the boot alternatives. ⓰
17. Use the SET LANGUAGE command to set the language desired. ⓱
18. Use the SET TERMINAL command to set the EEPROM terminal characteristics. These characteristics take effect when the system is initialized and the EEPROM is read again. Make sure the terminal is set to the same characteristics. ⓲
19. If the CPU you are working on has been designated as ineligible to be the boot processor, use the SET CPU/NOPRIMARY command to set its EEPROM correctly. ⓳
20. If any patches or boot primitives need to be selected, follow the procedure in Section 3.15. ⓴
21. Turn the lower key switch to the Auto Start position and boot the operating system. ㉑

3.14 How to Add a New Processor

Add a new processor in a slot to the left of the boot processor so it will be a secondary processor at power-up. This procedure is similar to that described in Section 3.13.

Example 3–11: Adding a Processor

```
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   A   .   M   M   M   M   .   .   P   P   P   P       TYP    ❺
    +   +   +   .   +   +   +   +   .   .   +   +   +   +       STF    ❻
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD
    .   .   .   .   .   .   .   .   .   .   +   +   +   +       ETF    ❻
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD

    .   .   .   .  A4   A3  A2  A1  .   .   .   .   .   .       ILV
    .   .   .   . 128  128 128 128  .   .   .   .   .   .       512Mb

ROM = 6.0         EEPROM = 2.0/6.0      SN = SGO1234567

?2D For Secondary Processor 3 ❽
?59 System serial number mismatch.  Secondary processor has xxxxxxx

>>> SHOW ALL ❾

     [System configuration and customization information prints]

>>> SET CPU 4  ❿    ! the node number of the CPU you added

>>> ESC (or CTRL/3) DEL SET SYSTEM SERIAL ⓫

>>> SET BOOT DEFAULT /XMI:E/ DU0 ⓬

>>> SET LANGUAGE INTERNATIONAL or ENGLISH ⓭

>>> SET TERMINAL /[NO]SCOPE /SPEED:9600 /[NO]BREAK  ⓮

>>> SET CPU/NOPRIMARY ⓯

!optional - may need latest console/diag patches again  ⓰

>>> BOOT ⓱
```

1. Turn the upper key switch straight up to the Off position (0).
2. Insert the new processor module to the left of the boot processor.
3. Turn the lower key switch to Halt.
4. Turn the upper key switch to Enable.
5. Check the self-test display for the processor, indicated by a P on the TYP line. See ❺ in Example 3–11.
6. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test. ❻
7. Turn the lower key switch to Update.
8. You will see the ?2D and ?59 messages. ❽ In addition, you may see messages indicating that you need to patch the EEPROM. Should you need to do so, see Section 3.15.
9. Type SHOW ALL to get the customized boot parameters, interleave characteristics, terminal setup, and any other parameters. ❾
10. Connect the console to the CPU you just added with the SET CPU *n* command. ❿
11. Issue the SET SYSTEM SERIAL command ⓫ and follow the prompts to correct the error message.
12. Issue the SET BOOT command supplying the parameters displayed by the SHOW ALL command to set up the boot alternatives. ⓬
13. Issue the SET LANGUAGE command to set the language desired.⓭
14. Issue the SET TERMINAL command to set the EEPROM terminal characteristics. ⓮
15. Use the SET CPU/NOPRIMARY command if the customer does not want this particular CPU to become the primary for some reason. ⓯
16. If any patches or boot primitives need to be selected, follow the procedure in Section 3.15. ⓰
17. Turn the lower key switch to the Auto Start position and boot the operating system. ⓱

3.15 Patching the EEPROM with EVUCA

To update the console and diagnostic ROMs on all VAX 6000 systems, use EVUCA under the Diagnostic Supervisor in standalone mode.

Example 3–12: Patching the EEPROM with EVUCA — Part 1

```
>>> BOOT /XMI:A /R5:110 EX0 ❸
       [self-test display prints]
Filename: ISL_LVAX
Follow Prompts
       [Diagnostic Supervisor banner]
DS> LOAD EVUCA  ❹
DS> ATTACH KA62B HUB KA0 1  ❺
DS> ATTACH KA62B HUB KA1 2
DS> ATTACH KA62B HUB KA2 5
DS> ATTACH KA62B HUB KA3 6
DS> SELECT ALL  ❻
DS> SET TRACE
DS> START

.. Program: EVUCA - VAX 6000 EEPROM Update Utility, revision 1.0, 5 tests,
Testing: _KA0 _KA1 _KA2 _KA3
   Booting secondary CPU 02.
   Booting secondary CPU 05.
   Booting secondary CPU 06.
Test 2: Load data from media
   Data file? <ELUCB.BIN>   ❼
Searching for data file...
   Data file loaded.
Looking for patch for CPU 01 - ROM 06.00 EEPROM 06.00
No patch image was found for CPU 01 - ROM 06.00 EEPROM 06.00  ❽

Looking for patch for CPU 02 - ROM 06.00 EEPROM 06.00
No patch image was found for CPU 02 - ROM 06.00 EEPROM 06.00

Looking for patch for CPU 05 - ROM 04.10 EEPROM 04.10
Patch image is revision 04.60
Do you really want to apply this patch [(No), Yes] YES ❾

Looking for patch for CPU 06 - ROM 04.10 EEPROM 04.60
Patch image is revision 04.60
Do you really want to apply this patch [(No), Yes]

Test 3: Determine Typecode Updated  ❿
```

1. Turn the lower key switch to Update.
2. Load the latest diagnostic CD in the CD drive. The CD is labeled 6000_ DIAG_x where x is a letter revision.
3. Boot the VAX Diagnostic Supervisor from the CD server using a command similar to that shown in Example 3–12 (see ❸).

   Alternatively, you could boot the supervisor from some other disk, perform the appropriate ATTACH commands, and then use the VAX/DS command SET LOAD to load EVUCA from the CD server. If you have already copied the latest diagnostics to a local disk, run EVUCA from there.
4. At the DS> prompt, type LOAD EVUCA (see ❹ in Example 3–12).
5. Issue the ATTACH command similar to that shown by ❺ to enable VAX/DS access to the CPUs on the system.
6. The CPUs are selected, TRACE is set, and EVUCA is started (see ❻).
7. Test 2 of EVUCA selects the correct patch file for the system being patched ❼. A carriage return is the appropriate response. (Test 1 is for Manufacturing Automated Verification System use.)
8. After loading the file, EVUCA identifies the ROM and EEPROM revisions of each CPU in the system and reports whether there is a patch for that revision. CPUs at nodes 1 and 2 ❽ have no patches.
9. When a patch is found, EVUCA asks the operator whether the patch should be applied. For CPU 05 the patch image found is 4.60 and the EEPROM rev. is 4.10 (see ❾). A "yes" response is given. For CPU 06 the patch revision is 4.60 and the EEPROM revision is 4.60. Therefore, the choice is made not to patch this EEPROM.
10. Test 3 determines what sections of the EEPROM need updating (see ❿).

Example 3–13: Patching the EEPROM with EVUCA — Part 2

```
Test 4: Update EEPROM data    ⓭
Getting selectable boot primitives for CPU 05, ROM 04.10

    [I/O device types in system identified]

    [Boot primitives available identified]

Available boot primitive space is 27F4
Please enter what boot primitive to delete by number. [1-3(D)] 2
Boot primitives fit into allotted EEPROM area.

Secondary CPUs are being updated, please wait a maximum of 20 seconds. ⓮
Updating CPU 05

Test 5: Show Boot primitives

ROM boot primitives for CPU 01, revision 06.00 are:
1       This boot primitive supports the following:
        - boot primitive designation DI
        - boot primitive designation MI
        - boot primitive designation CI
          Device CIBCA, device type 0108   ⓯
          Device KFMSA, device type 0810
          Device CIXCD, device type 0C05
   .
   .
   .
     [Messages about retrieving secondary CPU boot primitives]

CPU 02 has the same console revision as node 01.  ⓰
Boot primitives are the same for these CPUs.

ROM boot primitives for CPU 05 are:
1       This boot primitive supports the following:
        - boot primitive designation CI            ⓱
2       This boot primitive supports the following:
        - boot primitive designation DU
   .
   .
   .
EEPROM boot primitives for CPU 05 are:

1       This boot primitive supports the following:
        - boot primitive designation ET
   .
   .
   .

CPU 06 has the same console revision as node 05.
Boot primitives are the same for these CPUs.
```

Example 3–13 Cont’d on next page

Example 3–13 (Cont.): Patching the EEPROM with EVUCA — Part 2

```
The primary CPU was not updated.
Secondary CPU 05, was successfully updated.

Current ROM  and EEPROM revisions for each CPU are:

     [ROM revisions and EEPROM revisions summarized for each CPU] ⓲

.. End of run, 0 errors detected, pass count is 1,
   time is 1-JAN-1991 17:06:57.88
DS>
```

13. Test 4 creates a new updated EEPROM image in memory (see ⓭). Several boot primitives are available. Those that are permanent reside in ROM; those that are selectable or are patched reside in the EEPROM. If all boot primitives fit, EVUCA will not go through a selection process. However, if all primitives do not fit, the user is shown all the boot primitives available and prompted to choose which primitive is **not** wanted. If enough space is available, EVUCA will continue. If enough space is not available, the user must choose another **unwanted** primitive. This process continues until space is available for the remaining primitives.
14. Once the primitives fit, the EEPROMs are updated and the message noted by ⓮ is printed.
15. Later ROM revisions identify both what boot primitives are available and what devices they support (see ⓯).
16. CPU 2 and CPU 1 have the same ROM revision and have the same boot primitives ⓰.
17. CPU 5 has been patched, and test 5 shows the primitives in ROM followed by those in EEPROM (See ⓱ in Example 3–13). Since the ROM is an older revision, the devices the primitives support are not printed.
18. A final list of current ROM and EEPROM revisions is printed ⓲.

3.16 <REFERENCE>(XYP) Registers

The <REFERENCE>(xyp) registers consist of the processor status longword, internal processor registers, <REFERENCE>(xyp) registers in <REFERENCE>(XMI) private space, <REFERENCE>(XMI) required registers, and 16 general purpose registers.

Table 3–10: <REFERENCE>(xyp) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
Kernel Stack Pointer	KSP	IPR0	R/W	1
Executive Stack Pointer	ESP	IPR1	R/W	1
Supervisor Stack Pointer	SSP	IPR2	R/W	1
User Stack Pointer	USP	IPR3	R/W	1
Interrupt Stack Pointer	ISP	IPR4	R/W	1
Reserved		IPR5–IPR7		3
P0 Base Register	P0BR	IPR8	R/W	1
P0 Length Register	P0LR	IPR9	R/W	1
P1 Base Register	P1BR	IPR10	R/W	1
P1 Length Register	P1LR	IPR11	R/W	1

Key to Types:

R–Read
W–Write
R/W–Read/write

Key to Classes:

1–Implemented by the <REFERENCE>(xyp), specified in the *VAX Architecture Reference Manual*.
2–Implemented uniquely by the <REFERENCE>(xyp).
3–Not implemented. Read as zero; NOP on write.
4–Access not allowed; accesses result in a reserved operand fault.
5–Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
n I–The register is initialized on <REFERENCE>(xyp) reset (power-up, system reset, and node reset).

Table 3–10 (Cont.): <REFERENCE>(xyp) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
System Base Register	SBR	IPR12	R/W	1
System Length Register	SLR	IPR13	R/W	1
Reserved		IPR14–IPR15		3
Process Control Block Base	PCBB	IPR16	R/W	1
System Control Block Base	SCBB	IPR17	R/W	1
Interrupt Priority Level	IPL	IPR18	R/W	1 I
AST Level	ASTLVL	IPR19	R/W	1 I
Software Interrupt Request	SIRR	IPR20	W	1
Software Interrupt Summary	SISR	IPR21	R/W	1 I
Reserved		IPR22–IPR23		3
Interval Clock Control/Status	ICCS	IPR24	R/W	2 I
Next Interval Count	NICR	IPR25	W	3
Interval Count	ICR	IPR26	R	3
Time-of-Year Clock	TODR	IPR27	R/W	1
Console Storage Receiver Status	CSRS	IPR28	R/W	5 I
Console Storage Receiver Data	CSRD	IPR29	R	5 I
Console Storage Transmitter Status	CSTS	IPR30	R/W	5 I
Console Storage Transmitter Data	CSTD	IPR31	W	5 I
Console Receiver Control/Status	RXCS	IPR32	R/W	2 I
Console Receiver Data Buffer	RXDB	IPR33	R	2 I
Console Transmit Control/Status	TXCS	IPR34	W	2 I
Console Transmit Data Buffer	TXDB	IPR35	W	2 I
Translation Buffer Disable	TBDR	IPR36	R/W	3
Cache Disable	CADR	IPR37	R/W	2 I
Machine Check Error Summary	MCESR	IPR38	R/W	3
Memory System Error	MSER	IPR39	R/W	2 I
Reserved		IPR40–IPR41		3

Table 3–10 (Cont.): <REFERENCE>(xyp) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
Console Saved PC	SAVPC	IPR42	R	2
Console Saved PSL	SAVPSL	IPR43	R	2
Reserved		IPR44–IPR47		3
SBI Fault/Status	SBIFS	IPR48	R/W	3
SBI Silo	SBIS	IPR49	R	3
SBI Silo Comparator	SBISC	IPR50	R/W	3
SBI Maintenance	SBIMT	IPR51	R/W	3
SBI Error	SBIER	IPR52	R/W	3
SBI Timeout Address	SBITA	IPR53	R	3
SBI Quadword Clear	SBIQC	IPR54	W	3
I/O Bus Reset	IORESET	IPR55	W	2
Memory Management Enable	MAPEN	IPR56	R/W	1
Translation Buffer Invalidate All	TBIA	IPR57	W	1
Translation Buffer Invalidate Single	TBIS	IPR58	W	1
Translation Buffer Data	TBDATA	IPR59	R/W	3
Microprogam Break	MBRK	IPR60	R/W	3
Performance Monitor Enable	PMR	IPR61	R/W	3
System Identification	SID	IPR62	R	1
Translation Buffer Check	TBCHK	IPR63	W	1
Reserved		IPR64–IPR127		4

The IPRs are explicitly accessible to software only by the Move To Processor Register (MTPR) and Move From Processor Register (MFPR) instructions, which require kernel mode privileges. From the console, EXAMINE/I and DEPOSIT/I commands read and write the IPRs.

Table 3–11: <REFERENCE>(XMI) Registers for the the <REFERENCE>(xyp)

Register	Mnemonic	Address
XMI Device	XDEV	BB + 00
XMI Bus Error	XBER	BB + 04
XMI Failing Address	XFADR	BB + 08
XMI XGPR	XGPR	BB + 0C
KA62B Control/Status #2	CSR2	BB + 10

Note: "BB" = base address of a node, which is the address of the first location in nodespace.

Table 3–12: <REFERENCE>(xyp) Registers in <REFERENCE>(XMI) Private Space

Register	Mnemonic	Address
KA62B Control/Status #1	CSR1	2000 0000
KA62B ROM	ROM	2004 0000 to 2007 FFFF
KA62B EEPROM	EEPROM	2008 0000 to 2008 7FFF
SSC Base Address	SSCBR	2014 0000
SSC Configuration	SSCCR	2014 0010
CDAL Bus Timeout	CBTCR	2014 0020
Console Select	CONSEL	2014 0030
Timer Control Register 0	TCR0	2014 0100
Timer Interval Register 0	TIR0	2014 0104
Timer Next Interval Register 0	TNIR0	2014 0108
Timer Interrupt Vector Register 0	TIVR0	2014 010C
Timer Control Register 1	TCR1	2014 0110
Timer Interval Register 1	TIR1	2014 0114
Timer Next Interval Register 1	TNIR1	2014 0118
Timer Interrupt Vector Register 1	TIVR1	2014 011C
CSR1 Base Address	CSR1BADR	2014 0130
CSR1 Address Decode Mask	CSR1ADMR	2014 0134
EEPROM Base Address	EEBADR	2014 0140
EEPROM Address Decode Mask	EEADMR	2014 0144
SSC BBU RAM	BBURAM	2014 0400 to 2014 07FF
IP IVINTR Generation	IPIVINTRGEN	2101 0000 to 2101 FFFF
WE IVINTR Generation	WEIVINTRGEN	2102 0000 to 2102 FFFF

Chapter 4

<REFERENCE>(XRP) Scalar Processor

This chapter contains the following sections:

4.1 <REFERENCE>(XRP) Physical Description and Specifications

The <REFERENCE>(XRP) is a single-module VAX processor. The module designation is T2015. <REFERENCE>(rigel) systems can include up to six <REFERENCE>(XRP) processors, which use the 100 Mbyte/second <REFERENCE>(XMI) system bus to communicate with memory. Features of the module are shown in Figure 4–1.

Figure 4–1: <REFERENCE>(XRP) Module

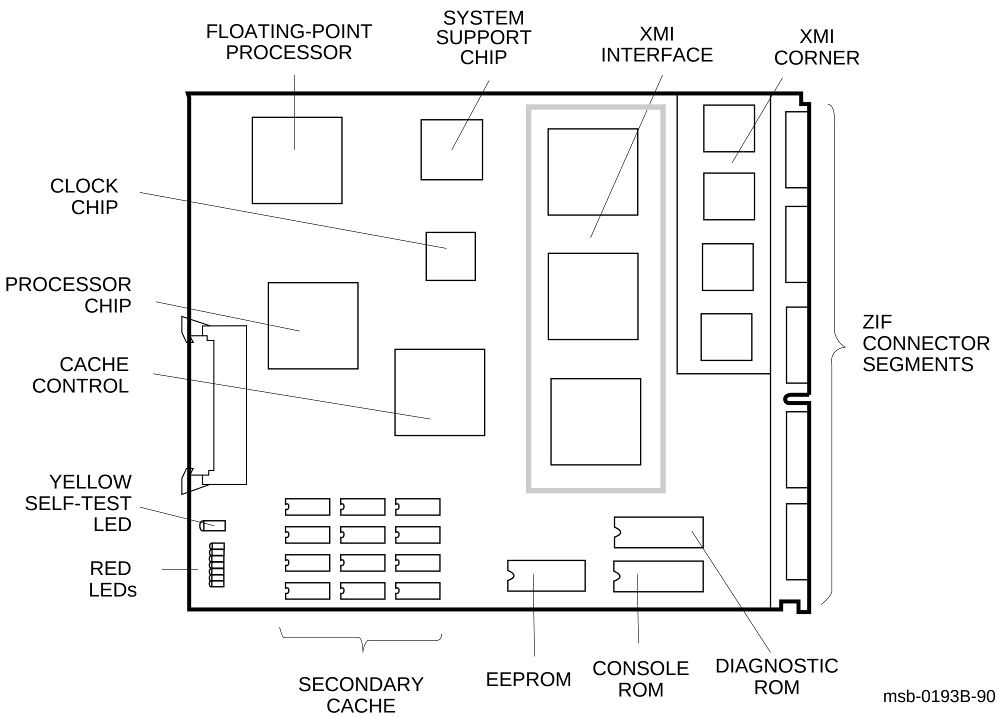


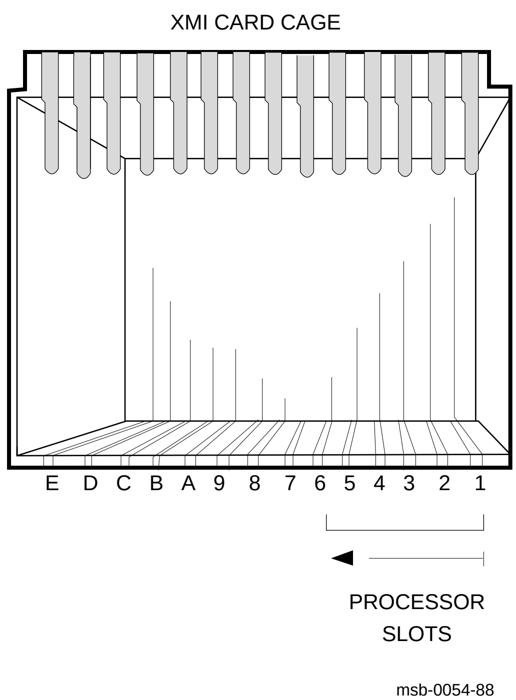
Table 4–1: <REFERENCE>(XRP) Specifications

Parameter	Description
Module Number:	T2015
Dimensions:	23.3 cm (9.2") H x 0.6 cm (0.25") W x 28.0 cm (11.0") D
Temperature:	
Storage Range	-40^{o}C to 66^{o}C (-40^{o}F to 151^{o}F)
Operating Range	5^{o}C to 50^{o}C (41^{o}F to 122^{o}F)
Relative Humidity:	
Storage	10% to 95% noncondensing
Operating	10% to 95% noncondensing
Altitude:	
Storage	Up to 4.8 km (16,000 ft)
Operating	Up to 2.4 km (8000 ft)
Current:	8.2A at +5V
Power:	41W
Cables:	Optional VIB for vector, 17-02240-03
Diagnostics:	ROM-based diagnostics (0 and 1) VAX/DS diagnostics, see Section 4.11.

4.2 <REFERENCE>(XRP) Configuration Rules

<REFERENCE>(XRP) modules will operate in any slot of the XMI card cage; however, processors usually go on the right, beginning with slot 1. Special rules apply if the KA64A has an attached vector processor; see Section 5.3.

Figure 4–2: Typical <REFERENCE>(XRP) Configuration



By convention, processors are placed in the right <REFERENCE>(XMI) slots, beginning with slot 1 and extending to slot 6. Memories are placed in the middle slots, from slot A to slot 5 and then slots B and C, and VAXBI adapters are installed in the left side of the card cage, beginning with slot E.

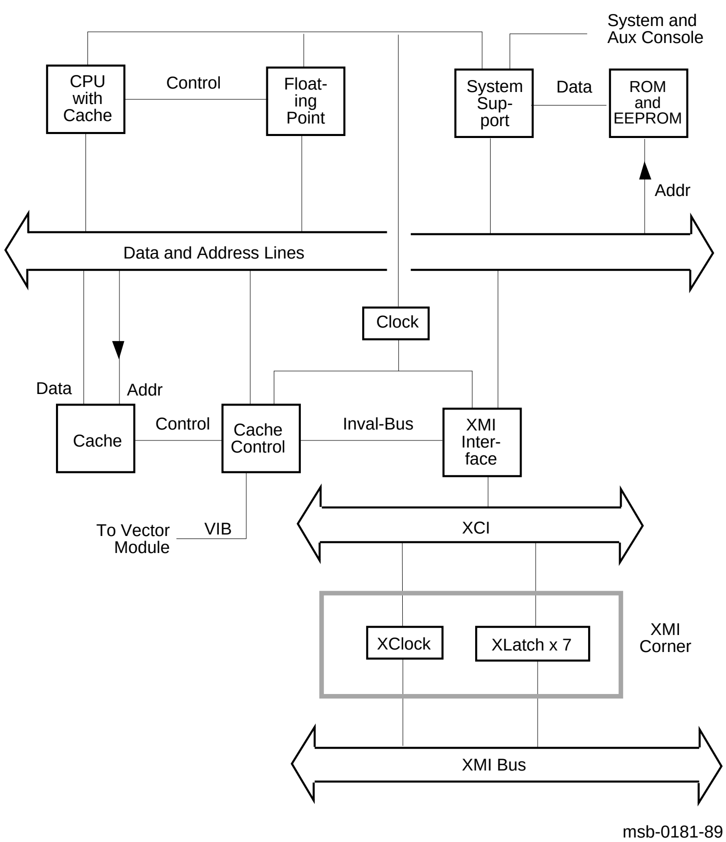
An attached vector processor must be in the slot to the left of the <REFERENCE>(XRP) module. The slot to the left of the vector processor can be used *only* for a memory module. Installing another kind of module can damage the vector module. A second scalar, vector, memory trio may be placed next to the first. See Section 5.3 for a discussion of configuration rules related to scalar/vector pairs.

For performance reasons, the scalar processor of a scalar/vector pair should not be made the primary processor when other scalar processors are in the system.

4.3 KA64A Functional Description

The <REFERENCE>(XRP) processor has four functional sections (see Figure 4–3): the CPU section, the backup cache, the XMI interface, and the console and diagnostics sections.

Figure 4–3: <REFERENCE>(XRP) Block Diagram



The CPU section includes:

- The processor chip, which supports the VAX Base Instruction Group and data types. It contains a 64-entry, fully associative translation buffer for both process and system-space mappings. The processor chip includes a 2-Kbyte, direct-mapped, write-through instruction and data cache with a quadword block and fill size.
- A floating-point accelerator chip that enhances the computation phase of floating-point and some integer instructions. This chip receives operands from the processor chip, computes the result, and passes the result and status back to the processor chip to complete the instruction.

The backup cache is a 128-Kbyte, direct-mapped, write-through instruction and data cache. It is implemented in 24 16-Kbyte x 4 data RAMs. The backup cache contains 2-Kbyte tags, organized to provide an octaword fill size and a 4-octaword block size.

The <REFERENCE>(XMI) interface includes:

- An octaword write buffer that decreases bus and memory controller bandwidth needs by packing writes into larger, more efficient blocks prior to sending them to main memory.
- Cache fill logic that loads the backup cache with four octawords of data on each cache miss.
- <REFERENCE>(XMI) write monitoring logic that uses a duplicate tag store to detect when another <REFERENCE>(XMI) node writes a memory location that is cached on this processor. Then the XMI interface chips invalidate the corresponding entry in the backup cache.
- Full set of error recovery and logging capabilities.

Example 4–1: ROM and EEPROM Version Numbers

```
#123456789 0123456789 0123456789 01234567#

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   .   .   M   M   M   M   .   .   P   P   P   P       TYP
    o   +   .   .   +   +   +   +   .   .   +   +   +   +       STF
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD
    .   .   .   .   .   .   .   .   .   .   +   +   +   +       ETF
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD

.   .   .   .   .   .   +   +   .   +   .   +   .   +   +   .   XBI E +


    .   .   .   .  A4   A3  A2  A1  .   .   .   .   .   .       ILV
    .   .   .   .  64   64  64  64  .   .   .   .   .   .       256Mb

ROM0 = V3.00❶  ROM1 = V3.00❷  EEPROM = 2.03/3.00❸  SN = SG01234567❹
```

The console and diagnostics sections include:

- A console read-only memory (ROM), which contains the code for initialization, executing console commands, and bootstrapping the system. The last line of the self-test display shows the ROM version. In this example callout ❶ indicates that the console ROM, ROM0, is version V3.00.
- A diagnostic ROM, ROM1, which contains the power-up self-test and extended diagnostics. The diagnostic ROM has the same version number as the console ROM. In this example callout ❷ indicates that the diagnostic ROM is version V3.00.
- An electrically-erasable, programmable ROM (EEPROM), which contains system parameters and boot code. You can modify the parameters with the console SET commands. Patching console and diagnostic code in the ROMs is done by reading the patches into a special area of the EEPROM. The recommended procedure is described in Section 4.18.

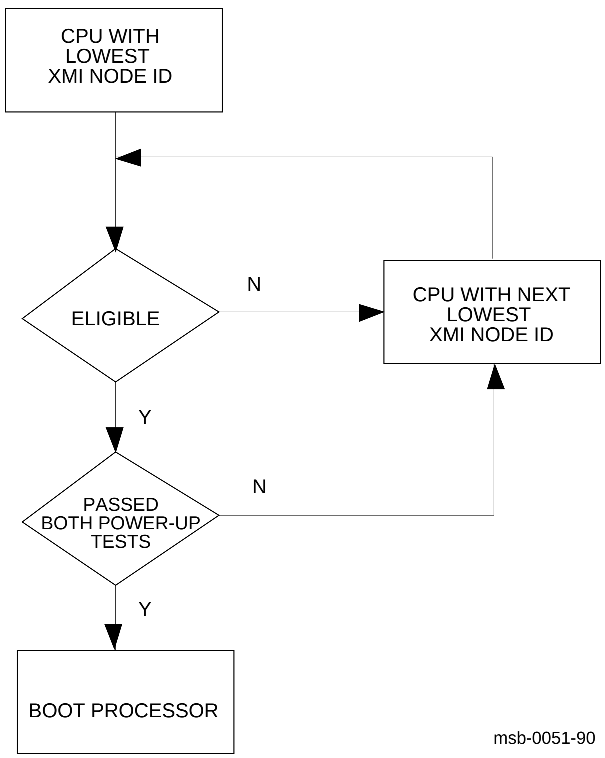
  The last line of the self-test display shows two EEPROM version numbers. The first number ❸ indicates the format version of the EEPROM. This version is changed only when the internal structure of the EEPROM is modified.

  The second number ❹ is the revision of ROM patches that have been applied to the EEPROM. The major number in this revision (before the decimal point) corresponds to the major number of the ROM revision ❶. The minor number indicates the actual patch revision. In this example, the EEPROM has not been patched for console ROM V3.00.
- A system support chip (RSSC) that includes support for external ROM/EEPROM, 1 Kbyte of battery-backed-up RAM, console terminal UARTs, bus reset logic, interval timer, programmable timers, time-of-year (TOY) clock, bus timeout, and halt arbitration logic.

4.4 Boot Processor

All <REFERENCE>(XRP) processors share system resources equally. The processor controlling the console at any given time is designated as the primary or boot processor. The others are called secondary processors. The boot processor is selected during the power-up sequence.

Figure 4–4: Selection of Boot Processor



Using boot code stored in its EEPROM, the boot processor reads the boot block from a specified device. Booting may be triggered by a command issued to the boot processor from the console, or by a system reset with the bottom key switch in the Auto Start position.

The boot processor also communicates with the system console, using the common console lines on the backplane. When you change system parameters in the EEPROM using SET commands, the boot processor automatically copies the new values to the EEPROMs on the secondary processors. If you swap in a new <REFERENCE>(XRP) module, it should be configured as a secondary processor. Then you can either use the UPDATE command to copy the boot processor's entire EEPROM to the new secondary or use EVUCA and a series of set commands to customize the EEPROM. Since UPDATE is slow, and can, in certain instances, render a processor unusable, the preferred method of updating a new processor placed in a system is to do it using SET commands. See Section 4.18 for information on running EVUCA and Section 4.15.2 on setting parameters.

CAUTION: *Using UPDATE can be dangerous because of revision mismatches. See Appendix C for information on what happens in mismatch cases.*

Usually the processor with the lowest <REFERENCE>(XMI) node number (which is also the lowest slot number) is selected as the boot processor. However, if this processor does not pass all its power-up tests, the next higher-numbered processor is selected. This is one way the boot processor can change.

The user also has control over boot processor selection with the SET CPU command. This command may declare a processor ineligible for selection. SET CPU can also select a boot processor explicitly.

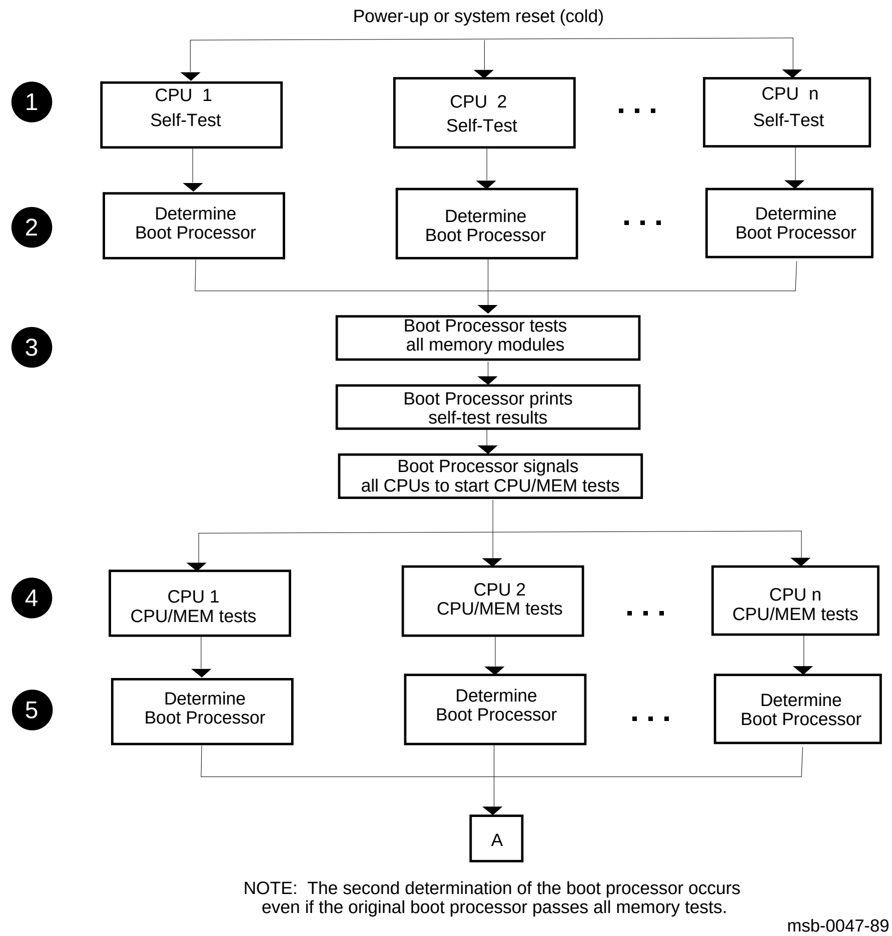
You can see the boot processor selection three ways:

- In the self-test display, the boot processor is indicated by a **B** on the second line labeled **BPD**.
- In console mode, the command SHOW CPU displays the boot processor as "Current primary." See Section 4.7.
- The bottom red LED is off on the boot processor module. It is lit on secondary processors.

4.5 Power-Up Sequence

Figure 4–5 shows the power-up sequence for <REFERENCE>(XRP) processors. All processors execute two phases of self-test, and a boot processor is selected. The boot processor tests the VAXBI adapters, if present, and prints the self-test display.

Figure 4–5: <REFERENCE>(XRP) Power-Up Sequence, Part 1 of 2



❶ All CPUs execute their on-board self-tests at the beginning of the power-up tests. On the **STF** line of the self-test display, a plus sign (+) is shown for every module whose self-test passes (see Section 4.6).

❷ The boot processor is determined as described in Section 4.4. On the **first BPD line**, the letter **B** corresponds to the processor selected as boot processor. Because the processors have not yet completed their power-up tests, the designated processor may later be disqualified from being boot processor. For this reason, the BPD line appears twice in the self-test display.

❸ The boot processor tests all memory modules, and then prints the results of self-test, lines **NODE, TYP, STF,** and **BPD** on the self-test display. The boot processor then signals all CPUs to start running the extended test.

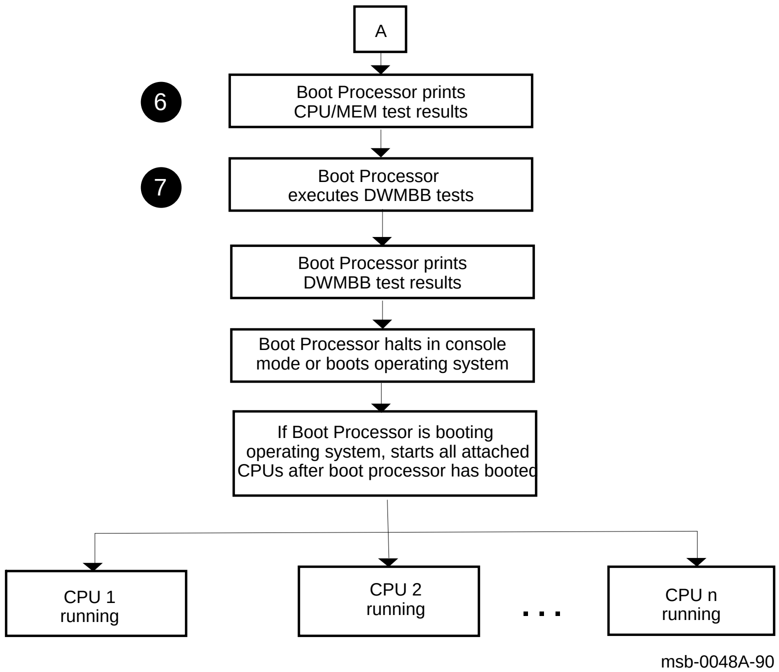
❹ All CPUs execute an extended test using the memories. On the **ETF** line of the self-test display, a plus sign (+) is shown for every module that passes extended test.

❺ If all CPUs pass the extended test, the original boot processor selection is still valid. Lines STF and ETF would be identical for all the processors.

The yellow LED and the top red LED are lit on all processor modules that pass both power-up tests. On the secondary processors, the bottom red LED is also lit. On the boot processor, this red LED is off (see Figure 4–7).

If the original boot processor fails the extended test (indicated by a minus sign (–) on line ETF), a new boot processor is selected. On the **second BPD line**, the letter **B** corresponds to the processor finally selected as boot processor.

Figure 4–6: <REFERENCE>(XRP) Power-Up Sequence, Part 2 of 2



❻ The boot processor prints the **ETF** line and the second **BPD** line of the self-test display. If none of the processors is successfully selected as the boot processor, no self-test results are displayed and the console hangs. You can identify this hung state by examining the LEDs on the processor modules (see Section 4.7). All yellow LEDs will be OFF. The group of seven red LEDS indicate the failing test number in binary-coded decimal.

You can force a processor to become the boot processor so that it will display self-test results by typing the following:

```
>>n
```

where *n* is the CPU to become the boot processor.

❼ The boot processor tests the DWMBB if one is present. Test results are indicated on the line labeled **XBI** on the self-test display. A plus sign (+) at the right means that the adapter test passed; a minus sign (–) means that the test failed.

4.6 <REFERENCE>(XRP) Self-Test Results: Console Display

You can check <REFERENCE>(XRP) self-test results in three ways: the self-test display, the lights on the module, and the contents of the XGPR register. Pertinent information in the self-test display is shown in Example 4–2. See Chapter 6 for information on vector processors.

Example 4–2: Self-Test Results

```
#123456789 0123456789 0123456789 01234567#❶

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   .   .   M   M   M   M   .   .   P   P   P   P       TYP❷
    o   +   .   .   +   +   +   +   .   .   +   +   +   +       STF❸
    .   .   .   .   .   .   .   .   .   .   E   E   D   B       BPD❹
    .   .   .   .   .   .   .   .   .   .   +   +   +   -       ETF❺
    .   .   .   .   .   .   .   .   .   .   E   B   D   E       BPD❻

.   .   .   .   .   .   +   +   .   +   .   +   .   +   +   .   XBI E +


    .   .   .   .  A4   A3  A2  A1  .   .   .   .   .   .       ILV
    .   .   .   . 128  128 128 128  .   .   .   .   .   .       512Mb

ROM0 = V3.00  ROM1 = V3.00  EEPROM = 2.03/3.00  SN = SG01234567❼
>>>
```

❶ The first line of the self-test printout is the progress trace. This line prints if a <REFERENCE>(XRP) module is in slot 1. The progress trace has two purposes: to give a visual indication that the system is functioning during self-test, and, if self-test fails, to display the failing test number. The numbers correspond to the 37 tests in the <REFERENCE>(XRP) self-test. When self-test passes, the line prints as in Example 4–2. If a test fails, the failing test number is the last one printed. For example, if test 14 fails, the line is printed as follows:

```
#123456789 01234
```

❷ This line indicates the type (TYP) of module at each <REFERENCE>(XMI) node. Processors are type P found at nodes 1, 2, 3, and 4 in this example.

❸ This line shows self-test fail status (STF), which are the results of on-board self-test. Possible values for processors are:

+ (pass)
– (fail)

All processors passed self-test in this example.

❹ The BPD line indicates boot processor designation. When the system completes on-board self-test, the processor with the lowest XMI ID number that passes self-test and is eligible is selected as boot processor — in this example, the processor at node 1.

The results on the BPD line indicate:

- B — The boot processor
- E — Processors eligible to become the boot processor
- D — Processors ineligible to become the boot processor

❺ During extended test (ETF) all processors run additional tests, which include reading and writing memory and using the cache. On line ETF, results are reported for each processor in the same way as on line STF — a plus sign indicates that extended test passed and a minus sign that extended test failed. In this example, the processor at node 1 (originally selected boot processor) failed the CPU/memory interaction tests.

❻ Another BPD line is displayed, because it is possible for a different CPU to be designated boot processor before the system actually boots. This occurs in this example, because the processor at node 1 failed the extended test. The lowest-numbered processor that passed both tests is the processor at node 2. However, a previous SET CPU/NOPRIMARY command has made this processor ineligible to be boot processor (indicated by the designation D on the BPD line). Therefore, the processor at node 3 is designated boot processor.

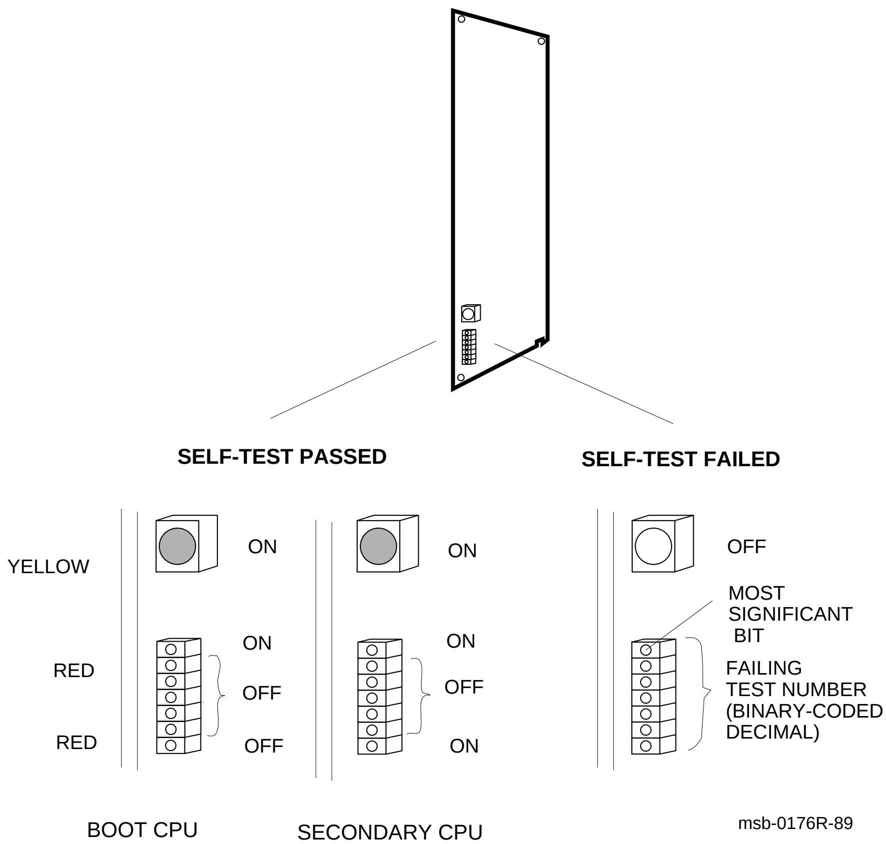
❼ The bottom line of the self-test display shows the ROM and EEPROM version numbers and the system serial number.

A <REFERENCE>(XRP) performs additional tests on an attached FV64A vector module (see Section 5.5).

4.7 <REFERENCE>(XRP) Self-Test Results: Module LEDs

You can check <REFERENCE>(XRP) self-test results in the self-test display, in the lights on the module, or in the XGPR register. If self-test passes, the large yellow LED is on.

Figure 4–7: <REFERENCE>(XRP) LEDs After Power-Up Self-Test



If self-test passes, the large yellow LED at the top of the LEDs is ON. (Here self-test means both the on-board power-up tests, RBD 0, and the CPU/memory interaction tests, RBD 1.) The top red LED (next to the yellow one) is also ON, and the next five red LEDs are OFF. The bottom LED is OFF if the processor is the boot processor, and ON if it is a secondary processor.

If self-test fails, the yellow LED is OFF, and the red LEDs contain an error code that corresponds to the number of the failing test. The test number is represented in binary-coded decimal. In the seven red error LEDs, the most significant bit is at the top, but the lights have a reverse interpretation — a bit is ONE if the light is OFF.

For example, assume a processor fails self-test (yellow LED is OFF) and shows the following pattern in its seven red LEDs:

```
TOP
    (MSB) on       0
          off      1  =  3
          off      1

          on       0
          on       0  =  2
          off      1
          on       0
BOTTOM
```

The failing test number decodes to 011 0010 (binary-coded decimal 32). If you then ran RBD 0 with the /TR and /HE qualifiers, the last test number you would see displayed is T0032.

When any of the red error LEDs is lit, a failure has occurred during the self-test sequence. But system power-up self-test actually comprises three sets of tests: <REFERENCE>(XRP) power-up tests (RBD 0), CPU/memory interaction tests (RBD 1), and VAXBI adapter (DWMBB) tests (RBD 2). Interpretation of the processor board error lights depends on which set of tests was running, as explained below and in Table 4–2.

Processor error LEDs can also indicate failures of memories or VAXBI adapters.

Table 4–2: <REFERENCE>(XRP) Error LEDs

Yellow LED	Red LEDs (Binary-coded decimal)	Diagnostic and Test Number	Device Failing	Self-Test Line
OFF	1–37*	Power-up self-test (RBD 0) T0001–T0037 See Table 4–6.	<REFERENCE>(XRP)	STF
OFF	50–62*	CPU/memory test - Memory 1 (RBD 1) T0001–T0013 See Table 4–7.	KA64A or MS65A 1 (module with lowest XMI node number)	ETF
OFF	67	CPU/memory test - Memory 2 T0003 (equivalent to ST1/T=3)	MS65A 2	ETF
OFF	68	CPU/memory test - Memory 3 T0003 (equivalent to ST1/T=3)	MS65A 3	ETF
OFF	69	CPU/memory test - Memory 4 T0003 (equivalent to ST1/T=3)	MS65A 4	ETF
OFF	70	CPU/memory test - Memory 5 T0003 (equivalent to ST1/T=3)	MS65A 5	ETF
OFF	71	CPU/memory test - Memory 6 T0003 (equivalent to ST1/T=3)	MS65A 6	ETF
OFF	72	CPU/memory test - Memory 7 T0003 (equivalent to ST1/T=3)	MS65A 7	ETF
OFF	73	CPU/memory test - Memory 8 T0003 (equivalent to ST1/T=3)	MS65A 8	ETF
ON	1–26	DWMBB test (RBD 2) T0001–T0026 See Table 7–5.	DWMBB	XBI

*Applies to scalar-only configuration. When a vector module is attached to a KA64A, the power-up tests are 1–49, and the CPU/memory tests are 50–66. Table 4–6 lists these tests.

If a processor's yellow LED is OFF and the red LEDs show an error code in the range 1–37, the power-up self-test failed and the processor board is bad. On the self-test console display, the processor shows a minus sign (–) on the STF line.

After the power-up tests, each processor runs the CPU/memory interaction tests. If a test fails, the processor shows a minus sign (–) on the ETF line of the self-test console display. The LED error codes are numbered from 50 to 62, which is the failing test number (1 through 13) plus 49, the number of tests in RBD 0. For example, assume that a processor fails self-test (yellow LED is OFF) and shows the following pattern in the error LEDs:

```
TOP
    (MSB) off      1
          on       0  =  5
          off      1

          on       0
          off      1  =  6
          off      1
          on       0
BOTTOM
```

The failing test number decodes to 101 0110 (binary-coded decimal 56), which corresponds to the seventh CPU/memory interaction test ((56–49 = 7).) If you then run RBD 1 with the /TR and /HE qualifiers, the last test number you see displayed is T0007.

Each processor, after testing with the first memory, runs the CPU/memory interaction tests on every other good memory module. (However, only CPU/memory interaction test T0003 is run.) If a failure occurs, the memory module is probably bad, although the processor's yellow light is OFF and the memory module's yellow light is ON. If several processors fail on the same memory, the memory is certainly bad. Try using SET MEMORY to configure the bad module out of the interleave set. For error codes higher than 66, consult Table 4–2 to determine the failing memory.

The last series is the DWMBB tests. If one fails, the red LEDs contain an error code, although the processor's yellow self-test LED is ON (because the CPU itself has passed). The failing test numbers are listed in Table 7–5. Note that only the boot processor performs the DWMBB tests.

4.8 <REFERENCE>(XRP) Self-Test Results: XGPR Register

When a <REFERENCE>(XRP) failure occurs during power-up and the failing test number cannot be found in the module LEDs and RBDs cannot be run, you can examine the XGPR register. The failing test number is left in the upper byte of the XGPR register of the failing <REFERENCE>(XRP) processor or of the boot processor if a memory or DWMBB module fails.

Example 4–3: XGPR Register After Power-Up Test Failure

```
>>> E/P/L 2190000C           ! Examine the longword at physical address
                             ! 2190000C, the address of the XGPR
2190000C  30xxxxxx           ! register of the <REFERENCE>(XRP) processor in slot 2.
                             ! The result indicates that test 30 of the
                             ! <REFERENCE>(XRP) self-test failed.  See Table 4-4
                             ! to interpret the data returned.

>>> E/P/L 2188000C           ! Examine the XGPR register of the <REFERENCE>(XRP)
                             ! processor in slot 1.  Derivation of the
                             ! address is explained below.
2188000C  49xxxxxx           ! CPU/memory interaction test 12 failed.

>>> E/P/L 2188000C           ! Examine the XGPR register of the boot
                             ! processor, which is in slot 1.
2188000C  A0xxxxxx           ! Disregard bit <31> (which indicates a
                             ! DWMBB test failure); the failing test
                             ! number is 20.
```

When a failure occurs in power-up test, you can examine the XGPR register to determine the failing test number. The XGPR register of the <REFERENCE>(XRP) processor that failed self-test or CPU/memory interaction test, or of the boot processor if DWMBB test failed, contains the failing test number. If all power-up tests pass, the XGPR register contains other data and should be ignored.

To examine the XGPR register, first see Table 4–3 to determine the base address (BB) of the <REFERENCE>(XRP) processor's node. Then calculate the address of the XGPR register by adding 0C (hex) to the base address.

The failing test number is derived from the upper byte (bits <31:24>) of the longword returned. For self-test, the upper byte contains the failing test number. If CPU/memory interaction test fails, this byte contains the failing test number plus 49. If DWMBB test fails, bit <31> is set (making the first digit 8 through A), and bits <30:24> contain the failing test number. All numbers are expressed in binary-coded decimal (BCD). See Table 4–4.

Table 4–3: XMI Base Addresses

Slot	Node	Base Address (BB)
1	1	2188 0000
2	2	2190 0000
3	3	2198 0000
4	4	21A0 0000
5	5	21A8 0000
6	6	21B0 0000
7	7	21B8 0000
8	8	21C0 0000
9	9	21C8 0000
10	A	21D0 0000
11	B	21D8 0000
12	C	21E0 0000
13	D	21E8 0000
14	E	21F0 0000

Table 4–4: Interpreting XGPR Failing Test Numbers

Failing Diagnostic	XGPR <31>	XGPR <30:24> (BCD)	Test Numbers
Self-test [1]	Clear	1–49	1–49
CPU/memory interaction test [2]	Clear	50–66	1–17
Additional memory [3]	Clear	67–73	3
DWMBB test [4]	Set	1–26	1–26

[1] See Table 4–6, KA64A Self-Test — RBD 0.

[2] See Table 4–7, CPU/Memory Interaction Tests — RBD 1.

[3] See Table 4–2, <REFERENCE>(XRP) Error LEDs.

[4] See Table 7–5, <REFERENCE>(xbi_plus_title) RBD Tests.

4.9 ROM-Based Diagnostics

The <REFERENCE>(XRP) ROMs contain diagnostics, which you run using the boot processor's RBD monitor program described in Chapter 2. RBD 0 and RBD 1 test the CPU, memory, and their interaction. RBD 2 tests the DWMBB/A and DWMBB/B adapters, and RBD 3 tests XMI memories.

Table 4–5: ROM-Based Diagnostics

Diagnostic	Test
0	<REFERENCE>(XRP) self-test
1	<REFERENCE>(XRP) CPU/memory interaction tests
2	<REFERENCE>(XBI_plus) tests
3	XMI memory tests

RBD 0 is the same as the <REFERENCE>(XRP) self-test. It is useful for running several passes when a processor fails self-test intermittently. Section 4.10.1 shows examples of running RBD 0 on both the boot processor and a secondary processor, and lists the tests in RBD 0.

RBD 1 is the same as the CPU/memory interaction tests. It is useful for running several passes when a processor fails CPU/memory interaction tests intermittently. Section 4.10.2 shows an example and lists the tests.

RBD 2 is the set of tests that the boot processor runs for each DWMBB VAXBI-to-XMI adapter when the system is powered on. (The DWMBB has no on-board self-test of its own.) Section 7.4 shows an example and lists the tests.

RBD 3 is a set of XMI memory tests that sizes and runs extended tests on all of memory. Sections 6.11 and 6.12 list the tests and show examples.

For a detailed explanation of the diagnostic printout, see Chapter 2.

4.10 <REFERENCE>(XRP) ROM-Based Diagnostics

4.10.1 Self-Test — RBD 0

RBD 0 is equivalent to the <REFERENCE>(XRP) self-tests. The first 37 tests test scalar CPU modules; tests 38–49 test vector modules.

Example 4–4: Self-Test — RBD 0

```
>>> T/R                    ! Command to enter RBD monitor program.  Short
                           ! for TEST/RBD.
RBD1>                      ! RBD monitor prompt, where 1 is the hexa-
                           ! decimal node number of the boot processor.
RBD1> ST0/TR/HE            ! Runs the <REFERENCE>(XRP) self-test on boot processor
                           ! Trace prints each test number; halt on error
                           ! Test results written to the console terminal:
;XRP/V_ST           3.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018❶

;       F❷       1     8082        1❸
;      HE   REX520       XX    T0018❶
;      10 AAAAAAAA A8AAAAAA 00000000 000004AC 2006451F 01

;       F        1     8082        1
;00000000 00000001 00000000 00000000 00000000 00000000 00000000

RBD1>
```

In Example 4–4:

❶ Test 18 failed. The /HE switch causes execution to stop when the error is encountered.

❷ F indicates failure.

❸ The diagnostic ran for one pass.

Example 4–5: Running <REFERENCE>(XRP) Self-Test (RBD 0) on a Secondary Processor

```
>>> SET CPU 2❶
>>> T/R
RBD2> ST0/TR❷        ! Note that only 37 tests ran.  If a vector
                      ! processor was present 49 tests would run

;XRP/V_ST       1.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018  T0019  T0020
; T0021  T0022  T0023  T0024  T0025  T0026  T0027  T0028  T0029  T0030
; T0031  T0032  T0033  T0034  T0035  T0036  T0037

;       P        2     8082        1
;00000000 00000000 00000000 00000000 000000000 00000000 00000000
```

In Example 4–5:

❶ This command causes the <REFERENCE>(XRP) module at node 2 to become the primary processor.

❷ The prompt indicates that the CPU at node 2 is the primary processor. RBD 0 is run on this processor.

Table 4–6: <REFERENCE>(XRP) Self-Test — RBD 0

Test	Function
	KA64A Tests
T0001	<REFERENCE>(XRP) ROM Checksum Test
T0002	IPL Step-Down Test
T0003	RSSC Configuration Register Test
T0004	RSSC RAM Test
T0005	RSSC Output Port Test
T0006	RSSC Address Decode Register Access Test
T0007	RSSC Console UART External Loopback and Baud Rate Test
T0008	RSSC Console UART Internal Loopback and Interrupt Test
T0009	<REFERENCE>(XRP) EEPROM Test
T0010	RSSC Input Port Test
T0011	RSSC Bus Timeout Test
T0012	RSSC Programmable Timers Test
T0013	RCSR Register Test
T0014	RSSC TOY Clock Test
T0015	RSSC Interval Timer Test
T0016	Interrupts at IPL 14 to 17 Test
T0017	Primary Cache Tag Store Test
T0018	Primary Cache Data RAM March Test
T0019	Backup Tag Store Test
T0020	Flush Cache Test
T0021	Backup Tag Store Parity Error Test
T0022	C-Chip Primary Tag Store Test
T0023	C-Chip Refresh Register Test
T0024	Backup Cache Data Line Test
T0025	Backup Cache Data RAM March Test
T0026	Backup Cache Data Parity RAM March Test

Table 4–6 (Cont.): <REFERENCE>(XRP) Self-Test — RBD 0

Test	Function
	KA64A Tests
T0027	Cache Mask Write Test
T0028	Data Parity Logic Test
T0029	Cache Disable Test
T0030	XGPR Register Test
T0031	CNAK Test
T0032	IVINTR Test
T0033	Multiple Interrupt Test
T0034	DC520 Critical Path Test
T0035	F-Chip Test
T0036	Disable F-Chip Test
T0037	F-Chip Critical Path Test
	FV64A Tests
T0038	VECTL Registers Test
T0039	Verse Registers Test
T0040	Load/Store Registers Test
T0041	VIB Error Logic Test
T0042	Other VECTL Chip Logic Test
T0043	Verse and Favor Test
T0044	Load/Store Translation Buffer and CAM Test
T0045	Load/Store Cache Test
T0046	Load/Store Instruction Test
T0047	Load/Store Tag and Duplicate Tag Test
T0048	Load/Store Error Cases Test
T0049	Module Critical Path Test

4.10.2 CPU/Memory Interaction Tests — RBD 1

RBD 1 is equivalent to the CPU/memory interaction tests. The first 13 tests test scalar CPU modules; tests 14–17 test vector modules.

Example 4–6: CPU/Memory Interaction Tests — RBD 1

```
>>> T/R                         ! Command to enter RBD monitor program

RBD3>                           ! RBD monitor prompt, where 3 is the hexa-
                                ! decimal node number of the processor
                                ! that is currently receiving your input.

RBD3> ST1/TR/HE                 ! Runs the CPU/memory interaction RBD with
                                ! trace; halt on error.  Test results
                                ! written to the console terminal:

;CPUMEM       3.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013

;        P❶      3     8082        1❷
;000000000 00000000 00000000 00000000 00000000 00000000 00000000

RBD3>
```

In the example above:

❶ P means that the diagnostic ran successfully.

❷ One pass was completed.

Table 4–7: CPU/Memory Interaction Tests — RBD 1

Test	Function
KA64A/Memory Interaction Tests	
T0001	Parity Error CNAK Read/Write Test
T0002	Miss When Invalid Test
T0003	Cache Read Fill Test
T0004	Interlock Instruction Cache Test
T0005	Octaword Write Buffer Test
T0006	Quadword-Boundary Byte Write Buffer Test
T0007	Two-Byte Write in Different Quadword Write Buffer Test
T0008	Write Buffer Switch and Purge Test
T0009	Statistical Write Buffer Test
T0010	Hit Write Buffer Test
T0011	Write Buffer Address Test
T0012	Invalidate Test
T0013	Hit on Disabled Tag Store Test
FV64A/Memory Interaction	
T0014	Cache Test
T0015	Write Buffer Test
T0016	Duplicate Tag Test
T0017	Miscellaneous Error Test

4.11 VAX/DS Diagnostics

The <REFERENCE>(XRP) software diagnostics that run under the VAX Diagnostic Supervisor (VAX/DS) are listed in Table 4–8. An example follows. See Section 2.4 for instructions on running the supervisor. See Section 5.8 for vector-specific tests.

Table 4–8: <REFERENCE>(XRP) VAX/DS Diagnostics

Program	Description
EVSBA	VAX Standalone Autosizer
EVSBB	VAX Diagnostic Online Autosizer
EVKAQ	VAX Basic Instructions Exerciser, Part 1
EVKAR	VAX Basic Instructions Exerciser, Part 2
EVKAS	VAX Floating Point Instruction Exerciser, Part 1
EVKAT	VAX Floating Point Instruction Exerciser, Part 2
EVKAU	VAX Privileged Architecture Instruction Test, Part 1
EVKAV	VAX Privileged Architecture Instruction Test, Part 2
ERKAX	Manual Tests
ERKMP	Multiprocessor Exerciser
EVUCA	VAX 6000 EEPROM Update Utility

Example 4–7: VAX/DS Commands for Running Standalone Processor Diagnostics

```
DS> RUN EVSBA❶
DS> SEL KA0❷
DS> RUN ERKAX❸
DS> EXIT❹
```

The callouts in Example 4–7 are explained below:

❶ Run the standalone autosizer; then you do not need to attach devices to the supervisor explicitly. However, if you want to know how to use the Attach command for a specific diagnostic, enter:

```
DS> HELP diagnostic_name ATTACH
```

❷ The instruction and manual tests run on the boot processor. If the boot processor is the CPU with the lowest <REFERENCE>(XMI) node number (which is usually the case), issue the command to select KA0. The Diagnostic Supervisor numbers the processors consecutively. For example, if the <REFERENCE>(XRP) module with the second-lowest <REFERENCE>(XMI) node number were boot processor, you would select KA1.

❸ This example runs the manual tests (ERKAX), which include powerfail, machine check, restart, and EEPROM functions. The diagnostic prints messages, and you must manually intervene using console switches.

❹ Exit from VAX/DS.

4.12 Machine Checks

Figure 4–8 and Table 4–9 show parameters for machine checks. The machine check frame is pushed onto the interrupt stack.

Figure 4–8: The Stack in Response to a Machine Check

Table 4–9: Machine Check Parameters

Parameter	Value (hex) or Bit	Description
Byte Count (SP)	18	Size of stack frame in bytes, not including PSL, PC, or byte count longword
Machine check code (SP+4)	01	Floating-point operand or result transfer error
	02	Floating-point reserved instruction
	03	Floating-point operand parity error
	04	Floating-point unknown status error
	05	Floating-point returned result parity error

Table 4–9 (Cont.): Machine Check Parameters

Parameter	Value (hex) or Bit	Description
	08	Translation buffer miss in ACV/TNV microflow
	09	Translation buffer hit in ACV/TNV microflow
	0A	Undefined INT.ID value
	0B	Undefined MOVCx state
	0C	Undefined instruction trap code
	0D	Undefined control store address
	10	Cache read tag/data parity error
	11	DAL bus or data parity read error
	12	DAL bus error on write or clear write buffer
	13	Undefined bus error microtrap
	14	Vector module error
Virtual address (SP+8)	<31:0>	Current contents of VAP register
Virtual instruction buffer address (SP+C)	<31:0>	Current virtual instruction buffer address
Interrupt state (SP+10)	<22>	ICCS bit <6>
	<15:1>	SISR bits <15:1>
Internal state (SP+14)	<31:24>	Difference between current PC and opcode PC
	<20:18>	Address of last memory reference
	<17:16>	Data length of last memory reference
	<15:8>	Opcode
	<3:0>	Last GPR referenced by E-box
Internal register (SP+18)	<31:0>	
Program counter (SP+1C)	<31:0>	PC at the start of the current instruction
Processor status longword (SP+20)	<31:0>	Current contents of PSL

4.13 Console Commands

Table 4–10 summarizes the console commands. The VAX 6000 Series Owner's Manual gives a full description of each command, its qualifiers, and examples.

Table 4–10: Console Commands

Command	Function
BOOT	Initializes the system, causing a self-test, and begins the boot program.
CLEAR EXCEPTION	Cleans up error state in XBER and RCSR registers.
CONTINUE	Begins processing at the address where processing was interrupted by a CTRL/P console command.
DEPOSIT	Stores data in a specified address.
EXAMINE	Displays the contents of a specified address.
FIND	Searches main memory for a page-aligned 256-Kbyte block of good memory or for a restart parameter block.
HALT	Null command; no action is taken since the processor has already halted in order to enter console mode.
HELP	Prints explanation of console commands.
INITIALIZE	Performs a system reset, including self-test.
PATCH EEPROM	Patches console, diagnostic, and boot primitive code from a TK70
REPEAT	Executes the command passed as its argument.
RESTORE EEPROM	Copies the TK tape's EEPROM contents to the EEPROM of the processor executing the command.
SAVE EEPROM	Copies to the TK tape the contents of the EEPROM of the processor executing the command.
SET BOOT	Stores a boot command by a nickname.
SET CPU	Specifies eligibility of processors to become the boot processor or whether the vector processor is to be included in the system configuration.

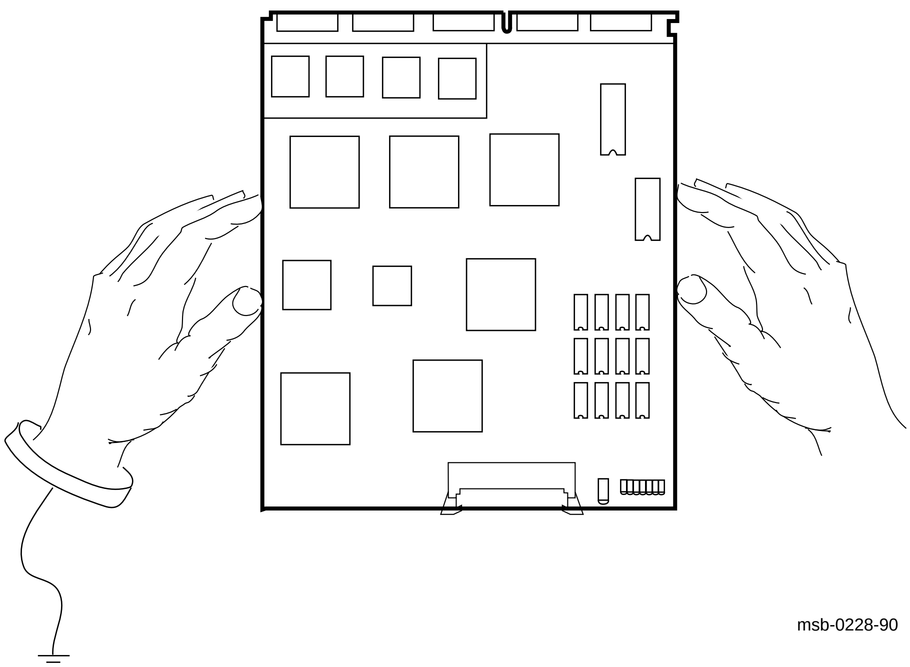
Table 4–10 (Cont.): Console Commands

Command	Function
SET LANGUAGE	Changes the output of the console error messages between numeric code only (international mode) and code plus explanation (English mode).
SET MEMORY	Designates the method of interleaving the memory modules; supersedes the console program's default interleaving.
SET TERMINAL	Sets console terminal characteristics.
SHOW ALL	Displays the current value of parameters set.
SHOW BOOT	Displays all boot commands and nicknames that have been saved using SET BOOT.
SHOW CONFIGURATION	Displays the hardware device type and revision level for each XMI and VAXBI node and indicates self-test status.
SHOW CPU	Identifies the primary processor and the status of other processors.
SHOW ETHERNET	Locates all Ethernet adapters on the system and displays their addresses.
SHOW LANGUAGE	Displays the mode currently set for console error messages, international or English.
SHOW MEMORY	Displays the memory lines from the system self-test, showing interleave and memory size.
SHOW TERMINAL	Displays the baud rate and terminal characteristics functioning on the console terminal.
START	Begins execution of an instruction at the address specified in the command string.
STOP	Halts the specified node.
TEST	Passes control to the self-test diagnostics; /RBD qualifier invokes ROM-based diagnostics.
UPDATE	Copies contents of the EEPROM on the processor executing the command to the EEPROM of another processor.
Z	Logically connects the console terminal to another processor on the <REFERENCE>(XMI) bus or to a VAXBI node.
!	Introduces a comment.

4.14 <REFERENCE>(XRP) Handling Procedures

The <REFERENCE>(XRP) module is static sensitive and fragile. The CMOS2 technology used on this module is more vulnerable to static than past technology. The 25 mil leads used to attach chips to the module are very small, close together, and easily bent. Be careful with the module.

Figure 4–9: Holding the <REFERENCE>(XRP) Module



The <REFERENCE>(XRP) module **must** be handled carefully. Figure 4–9 shows the proper way to hold the module. Be sure your hands do not touch any components, leads, or XMI fingers. When inserting it in or removing it from the XMI card cage, grasp the module only at the spot shown in Figure 4–10, avoiding any contact with the 25 mil leads. Do not use any component as a handle.

To avoid damaging the <REFERENCE>(XRP) module, follow these handling procedures:

1. Always wear an antistatic wrist strap.
2. Before removing the module from its ESD box, place the box on a clean, stable surface.

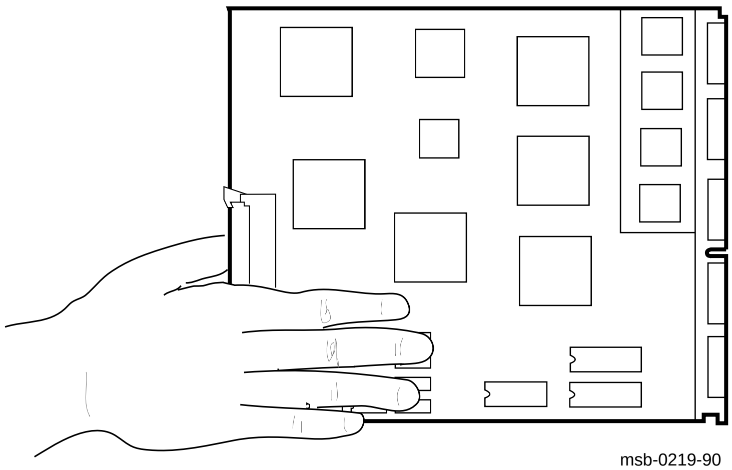
   Be sure the box will not slide or fall. **Never** place the box on the floor. And be sure no tools, papers, manuals, or anything else that might damage the module are near it. Some components on this module can be damaged by a 600-volt static charge; paper, for example, can carry a charge of 1000 volts.
3. Hold the module only by the edges, as shown in Figure 4–9.

   Do not hold the module so that your fingers touch any components or leads. Be sure you do not bend the module as you are holding it.
4. Be sure nothing touches the module surface or any of its components.

   If anything touches the module, components or leads can be damaged. This includes the antistatic wrist strap, clothing, jewelry, cables, components on other modules, and anything in the work area (such as tools, manuals, or loose papers).

   Remove your jacket and roll up your sleeves before handling the module. Also remove any jewelry.

Figure 4–10: Inserting the <REFERENCE>(XRP) Module in an XMI Card Cage



You must take special precautions when inserting the <REFERENCE>(XRP) module in or removing it from the XMI card cage.

1. Be sure, when inserting the module in or removing it from the XMI card cage, that no part of the module comes in contact with another module or a cable.

2. When swapping out a module, place it in an unused XMI slot, if one is available, or set the module on an ESD mat while you install the new module.

   An unused XMI slot is the best place to leave a module that is being swapped out until it can be placed in the ESD box. If there are no extra slots, place the module you removed on an ESD mat on a stable, uncluttered surface, with side 1 (the side with the heat sinks) up. Do not put it on the top of the system cabinet. And never slide the module across any surface. The leads on the components are fragile and can be damaged by contact with fingers or any surface.

3. Hold the XMI card cage handle while removing or inserting the module.

   If it is not held in place, the handle can spring down and damage the module.

4. When inserting the module in the card cage, grasp it as shown in Figure 4–10, and slide it slowly and gently into the slot.

5. **Do not attach the repair tag to the module.**

   Place the repair tag in the plastic bag attached to the bottom of the ESD box. Allowing the repair tag to come in contact with the module can cause damage to a component.

4.15 CPU Replacement in Single CPU Systems

There are two methods available to customize a processor in a single-processor system. The first uses the RESTORE command; the second sets the EEPROM by console command.

4.15.1 Using the RESTORE Command

Use the RESTORE command to customize a replaced CPU in a single-processor system. Use this method only if the ROM revision of the spare is the same as the ROM revision of the CPU being replaced. Before you begin set the terminal baud rate to 1200.

Example 4–8: Replacing a Single Processor

```
#123456789 0123456789 0123456789 01234567#

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   .   A   .   M   M   .   .   .   .   .   .   .   P       TYP    ❻
    o   .   +   .   +   +   .   .   .   .   .   .   .   +       STF    ❼
    .   .   .   .   .   .   .   .   .   .   .   .   .   B       BPD
    .   .   .   .   .   .   .   .   .   .   .   .   .   +       ETF    ❼
    .   .   .   .   .   .   .   .   .   .   .   .   .   B       BPD

.   .   .   .   .   .   +   +   .   +   .   +   .   +   +   .   XBI E +

    .   .   .   .  A2  A1   .   .   .   .   .   .   .   .       ILV
    .   .   .   .  64  64   .   .   .   .   .   .   .   .       128Mb

ROM0 = V3.00  ROM1 = V3.00 ❽  EEPROM = 2.03/3.00   SN = 0000000000 ❾

?4F System serial number has not been initialized ❾

>>> RESTORE EEPROM ⓬

?6E EEPROM Revision = 2.03/3.00
?70 Tape image Revision = 2.03/3.01
Proceed with update of EEPROM? (Y or N)

!optional - may need latest console/diag patches again  ⓭

>>> BOOT  ⓯
```

1. Turn the upper key switch straight up to the Off position (0).
2. Remove the defective CPU module.
3. Insert the new processor module.
4. Turn the lower key switch to Halt.
5. Turn the upper key switch to Enable.
6. Check the self-test display for the processor, indicated by a P on the TYP line (usually in slot 1). See ❻.
7. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test. See ❼.
8. Compare the ROM revision ❽ of the new processor with the ROM revision of the one you just replaced. If they are the same, continue with this procedure; otherwise use the second method described in Section 4.15.2.
9. You will see the following message: ❾

   ```
   ?4F System serial number has not been initialized.
   ```

10. Turn the lower key switch to Update.
11. Mount the TK cartridge containing the most recent saved image of the old processor's EEPROM.[1]
12. Issue the console command RESTORE EEPROM ⓬ to copy all areas of the EEPROM except the module-specific area to the boot processor's EEPROM.
13. If any patches have been issued since the last save, use EVUCA to patch the EEPROM. See Section 4.18 for details.
14. Turn the lower key switch to the Auto Start position.
15. Boot the operating system ⓯. Booting will initialize the system and the EEPROM will be read. If the system console baud rate was not normally set at 1200, you will have to change the terminal baud rate back to its original value.

[1] When the system is installed or after maintenance, customer service engineers should save the EEPROM on a TK cartridge if available. The cartridge is left in the care of the customer. Subsequently, the EEPROM might have been changed and saved several times. This would normally happen following a PATCH operation.

4.15.2 Using SET Commands

When replacing the only processor module in a system that does not have a TK, you must enter console commands that customize the EEPROM. The Site Management Guide should contain this information. Before you begin set the terminal baud rate to 1200.

Example 4–9: Replacing a Single Processor

```
#123456789 0123456789 0123456789 01234567#

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   .   .   M   M   .   .   .   .   .   .   .   P       TYP    ❼
    +   +   .   .   +   +   .   .   .   .   .   .   .   +       STF    ❽
    .   .   .   .   .   .   .   .   .   .   .   .   .   B       BPD
    .   .   .   .   .   .   .   .   .   .   .   .   .   +       ETF    ❽
    .   .   .   .   .   .   .   .   .   .   .   .   .   B       BPD

    .   .   .   .  A2   A1  .   .   .   .   .   .   .   .       ILV
    .   .   .   .  64   64  .   .   .   .   .   .   .   .       128Mb

ROM0 = V3.00  ROM1 = V3.00     EEPROM = 2.03/3.00   SN = 0000000000

?4F System serial number has not been initialized

>>> ESC (or CTRL/3) DEL SET SYSTEM SERIAL RET ❿

    ! Follow the prompts to set the Serial Number.

>>> SET BOOT DEFAULT /XMI:E DU0 ⓫

>>> SET LANGUAGE INTERNATIONAL ⓬
      OR
>>> SET LANGUAGE ENGLISH

>>> SET TERMINAL /[NO]SCOPE /SPEED:9600 /[NO]BREAK  ⓭

!optional - may need latest console/diag patches ⓮

>>> BOOT  ⓯
```

1. Turn the upper key switch straight up to the Off position (0).
2. Remove the defective CPU module.
3. Insert the new processor module.
4. The EEPROM on the new module has a default baud rate of 1200. Make sure your terminal is set to 1200.
5. Turn the lower key switch to Halt.
6. Turn the upper key switch to Enable.
7. Check the self-test display for the processor, indicated by a P on the TYP line. See ❼ in Example 4–9.
8. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test (see ❽).
9. Turn the lower key switch to Update.
10. To correct the error message, enter the SET SYSTEM SERIAL command (see ❿) and follow the prompts. The serial number should be in the *Site Management Guide*. If you do not find it there, look on an old self-test display or on the tape located near the bottom of the left rear cabinet upright.
11. To set up the boot alternatives, enter the SET BOOT command supplying the parameters recorded in the *Site Management Guide*. You might want to check these with the system manager. ⓫
12. Use the SET LANGUAGE command to set up the language desired.⓬
13. Use the SET TERMINAL command to set the EEPROM terminal characteristics.⓭ Results of the SET TERMINAL command will take effect when the system is reset. Make sure the terminal itself is set to the same characteristics.
14. Use whatever console commands are necessary to completely customize the EEPROM to the customer's satisfaction.

    If any patches or boot primitives need to be selected, follow the procedure in Section 4.18. ⓮
15. Turn the lower key switch to the Auto Start position and Boot the operating system. ⓯

4.16 CPU Replacement in Multiple CPU Systems

In a multiprocessing system replacing a processor requires customizing the new processor into the system. If the processor being replaced is the boot processor, then a few extra steps are required. Since different CPUs can have different ROM and EEPROM revisions in the same system, updating the EEPROM of any processor should be done using SET commands and EVUCA.

Example 4–10: Retrieving EEPROM Information

```
#123456789 0123456789 0123456789 01234567#

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   A   .   M   M   M   M   .   .   P   P   P   P       TYP    ❼
    +   +   +   .   +   +   +   +   .   .   +   +   +   +       STF    ❽
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD
    .   .   .   .   .   .   .   .   .   .   +   +   +   +       ETF    ❽
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD

    .   .   .   .  A4   A3  A2  A1  .   .   .   .   .   .       ILV
    .   .   .   . 128  128 128 128  .   .   .   .   .   .       512Mb

ROM0 = V3.00 ROM1 = V3.00 EEPROM = 2.03/3.00 SN = 0000000000

?50 System serial number not initialized on primary processor.
                           or                               ❿
?2D For Secondary Processor 3
?5A System serial number mismatch.  Secondary processor has xxxxxxx

>>> SET CPU 3 ⓫

>>> [ESC] (or [CTRL/3]) [DEL] SHOW SYSTEM SERIAL ⓬

>>> SHOW ALL ⓭

     [System configuration and customization information prints]
```

1. Turn the upper key switch straight up to the Off position (0).
2. Remove the defective CPU module.
3. Insert the new processor module.
4. If the processor you are replacing is the first processor on the right (usually in slot one), the console will make it the boot processor. If this is the case, make sure that the console terminal baud rate is 1200, the default baud rate of the spare.
5. Turn the lower key switch to Halt.
6. Turn the upper key switch to Enable.
7. Check the self-test display for the processor, indicated by a P on the TYP line. See ➐ in Example 4–10.
8. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test. ➑
9. Turn the lower key switch to Update.
10. You will see either the ?50 message or the ?2D and ?5A messages.➓ If you see mismatch messages, you may have to patch the EEPROM using EVUCA. See Section 4.18. If you see the ?50 message, the boot processor has been replaced and you must execute the next step; otherwise, go to Step 12.
11. Issue the SET CPU *n* command so that the console is connected to a processor that has been in the system for some time. ⓫
12. Get the system serial number by entering the SHOW SYSTEM SERIAL command. ⓬
13. Issue the SHOW ALL command to get the customized boot parameters, interleave characteristics, terminal setup, and any other parameters. ⓭

If you collected the necessary information, you are now ready to customize the EEPROM.

Example 4–11: Customizing an EEPROM

```
>>> SET CPU 3  ⓮
>>> [ESC] (or [CTRL/3]) [DEL] SET SYSTEM SERIAL ⓯
    ! Follow the prompts to set the serial number.
>>> SET BOOT DEFAULT /XMI:E DU0 ⓰
>>> SET LANGUAGE INTERNATIONAL or ENGLISH ⓱
>>> SET TERMINAL /[NO]SCOPE /SPEED:9600 /[NO]BREAK  ⓲
>>> SET CPU/NOPRIMARY 3 ⓳
!optional - may need latest console/diag patches again  ⓴
>>> BOOT ㉑
```

14. Issue the SET CPU *n* command to connect the console to the primary or secondary processor you just replaced. ⓮
15. Issue the SET SYSTEM SERIAL command ⓯ and follow the command prompts to correct the error message ❿.
16. Use the SET BOOT command supplying the parameters displayed by the SHOW ALL command to set up the boot alternatives. ⓰
17. Use the SET LANGUAGE command to set the language desired. ⓱
18. Use the SET TERMINAL command to set the EEPROM terminal characteristics. These characteristics take effect when the system is initialized and the EEPROM is read again. Make sure the terminal is set to the same characteristics. ⓲
19. If the CPU you are working on has been designated as ineligible to be the boot processor, use the SET CPU/NOPRIMARY command to set its EEPROM correctly. ⓳
20. If any patches or boot primitives need to be selected, follow the procedure in Section 4.18. ⓴
21. Turn the lower key switch to the Auto Start position and boot the operating system. ㉑

4.17 How to Add a New Processor

Add a new processor in a slot to the left of the boot processor so it will be a secondary processor at power-up. This procedure is similar to that described in Section 4.16.

Example 4–12: Adding a Processor

```
#123456789 0123456789 0123456789 01234567#

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   A   .   M   M   M   M   .   .   P   P   P   P       TYP    ❺
    +   +   +   .   +   +   +   +   .   .   +   +   +   +       STF    ❻
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD
    .   .   .   .   .   .   .   .   .   .   +   +   +   +       ETF    ❻
    .   .   .   .   .   .   .   .   .   .   E   E   E   B       BPD

    .   .   .   .  A4   A3  A2  A1  .   .   .   .   .   .       ILV
    .   .   .   . 128  128 128 128  .   .   .   .   .   .       512Mb

ROM0 = V3.00  ROM1 = V3.00     EEPROM = 2.03/3.00      SN = SGO1234567

?2D For Secondary Processor 3 ❽
?5A System serial number mismatch.  Secondary processor has xxxxxxx

>>> SHOW ALL ❾

     [System configuration and customization information prints]

>>> SET CPU 3  ❿     ! the node number of the CPU you added

>>> [ESC] (or [CTRL/3]) [DEL] SET SYSTEM SERIAL ⓫

>>> SET BOOT DEFAULT /XMI:E/ DU0 ⓬

>>> SET LANGUAGE INTERNATIONAL or ENGLISH ⓭

>>> SET TERMINAL /[NO]SCOPE /SPEED:9600 /[NO]BREAK  ⓮

>>> SET CPU/NOPRIMARY ⓯

!optional - may need latest console/diag patches again  ⓰

>>> BOOT ⓱
```

1. Turn the upper key switch straight up to the Off position (0).
2. Insert the new processor module to the left of the boot processor.
3. Turn the lower key switch to Halt.
4. Turn the upper key switch to Enable.
5. Check the self-test display for the processor, indicated by a P on the TYP line. See ❺ in Example 4–12.
6. If the processor shows a plus sign (+) on both the STF and ETF lines, it passed self-test. ❻
7. Turn the lower key switch to Update.
8. You will see the ?2D and ?5A messages.❽ In addition, you may see messages indicating that you need to patch the EEPROM. Should you need to do so, see Section 4.18.
9. Type SHOW ALL to get the customized boot parameters, interleave characteristics, terminal setup, and any other parameters. ❾
10. Connect the console to the CPU you just added with the SET CPU *n* command. ❿
11. Issue the SET SYSTEM SERIAL command ⓫ and follow the command prompts to correct the error message.
12. Issue the SET BOOT command supplying the parameters displayed by the SHOW ALL command to set up the boot alternatives. ⓬
13. Issue the SET LANGUAGE command to set the language desired.⓭
14. Issue the SET TERMINAL command to set the EEPROM terminal characteristics. ⓮
15. Issue the SET CPU/NOPRIMARY command if the customer does not want this particular CPU to become the primary for some reason. ⓯
16. If any patches or boot primitives need to be selected, follow the procedure in Section 4.18. ⓰
17. Turn the lower key switch to the Auto Start position and boot the operating system. ⓲

4.18 Patching the EEPROM with EVUCA

To update the console and diagnostic ROMs on all VAX 6000 systems, use EVUCA under the Diagnostic Supervisor in standalone mode.

Example 4–13: Patching the EEPROM with EVUCA — Part 1

```
>>> BOOT /XMI:A /R5:110 EX0 ❸
       [Self-test display prints]
Filename: ISL_LVAX
Follow Prompts
       [Diagnostic Supervisor Banner]
DS> LOAD EVUCA  ❹
DS> ATTACH KA64A HUB KA0 1  ❺
DS> ATTACH KA64A HUB KA1 2
DS> ATTACH KA64A HUB KA2 5
DS> ATTACH KA64A HUB KA3 6
DS> SELECT ALL  ❻
DS> SET TRACE
DS> START

.. Program: EVUCA - VAX 6000 EEPROM Update Utility, revision 0.5, 5 tests,
Testing: _KA0 _KA1 _KA2 _KA3
Booting secondary CPU 02.
Booting secondary CPU 05.
Booting secondary CPU 06.
Test 2: Load data from media
Data file? <ERUCA.BIN>   ❼
Searching for data file...
Data file loaded.
Looking for patch for CPU 01 - ROM 03.00 EEPROM 03.00
No patch image was found for CPU 01 - ROM 03.00 EEPROM 03.00  ❽

Looking for patch for CPU 02 - ROM 03.00 EEPROM 03.00
No patch image was found for CPU 02 - ROM 03.00 EEPROM 03.00

Looking for patch for CPU 05 - ROM 02.00 EEPROM 02.00
Patch image is revision 02.02
Do you really want to apply this patch [(No), Yes] YES ❾

Looking for patch for CPU 06 - ROM 02.00 EEPROM 02.02
Patch image is revision 02.02
Do you really want to apply this patch [(No), Yes]

Test 3: Determine Typecodes Updated ❿
```

1. Turn the lower key switch to Update.
2. Load the latest diagnostic CD in the CD drive. The CD is labeled 6000_ DIAG_x where x is a letter revision.
3. Boot the VAX Diagnostic Supervisor from a CD server using a command similar to that shown in Example 4–13 (see ❸).

   Alternatively, you could boot the supervisor from some other disk, perform the appropriate ATTACH commands, and then use the VAX/DS command SET LOAD to load EVUCA from the CD server. If you have already copied the latest diagnostics to a local disk, run EVUCA from there.
4. At the DS> prompt, type LOAD EVUCA (see ❹ in Example 4–13).
5. Issue the ATTACH command similar to that shown by ❺ to enable the VAX/DS access to the CPUs on the system.
6. The CPUs are selected, TRACE is set, and EVUCA is started (see ❻).
7. Test 2 of EVUCA selects the correct patch file for the system being patched. ❼. A carriage return is the appropriate response. (Test 1 is for Manufacturing Automated Verification System use.)
8. After loading the file, EVUCA identifies the ROM and EEPROM revisions of each CPU in the system and reports whether that revision has a patch. CPUs at nodes 1 and 2 ❽ have no patches.
9. When a patch is found, EVUCA asks the operator whether the patch should be applied. For CPU 05 the patch image found is 2.02 and the EEPROM revision is 2.00. A "yes" response is given (see ❾). For CPU 06 the patch revision is 2.02 but the EEPROM revision is 2.02. Therefore, the choice is made not to patch this EEPROM.
10. Test 3 determines what sections of the EEPROM need updating.

Example 4–14: Patching the EEPROM with EVUCA — Part 2

```
Test 4: Update EEPROM data    ⓭
Getting selectable boot primitives for CPU 05, ROM 02.00

    [I/O device types in system identified]

    [Boot primitives available identified]

Available boot primitive space is 27F4
Please enter what boot primitive to delete by number. [1-3(D)] 2
Boot primitives fit into allotted EEPROM area.

Secondary cpus are being updated, please wait a maximum of 20 seconds. ⓮
Updating CPU 05

Test 5: Show Boot primitives

ROM boot primitives for CPU 01, revision 3.00 are:
1       This boot primitive supports the following:
        - boot primitive designation DI
        - boot primitive designation MI
        - boot primitive designation CI
          Device CIBCA, device type 0108   ⓯
          Device CIXCD, device type 0C05
   .
   .
   .
     [Messages about retrieving secondary CPU boot primitives]

CPU 02 has the same console revision as node 01.  ⓰
Boot primitives are the same for these cpus.

ROM boot primitives for CPU 05, are:
1       This boot primitive supports the following:
        - boot primitive designation CI            ⓱
2       This boot primitive supports the following:
        - boot primitive designation DU
   .
   .
   .
EEPROM boot primitives for CPU 05 are:

1       This boot primitive supports the following:
        - boot primitive designation ET
   .
   .
   .

CPU 06 has the same console revision as node 05.
Boot primitives are the same for these cpus.

The primary CPU was not updated.
Secondary CPU 05, was successfully updated.
```

Example 4–14 Cont’d on next page

Example 4–14 (Cont.): Patching the EEPROM with EVUCA — Part 2

```
Current ROM and EEPROM revisions for each CPU are:

  [ROM revisions and EEPROM revisions summarized for each CPU] ⓲

.. End of run, 0 errors detected, pass count is 1,
   time is 24-SEP-1990 17:06:57.88
DS>
```

13. Test 4 creates a new updated EEPROM image in memory (see ⓭). Several boot primitives are available. Those that are permanent reside in ROM; those that are selectable or are patched reside in the EEPROM. If all boot primitives fit, EVUCA will not go through a selection process. However, if all primitives do not fit, the user is shown all the boot primitives available and prompted to choose which primitive is **not** wanted. If enough space is available, EVUCA continues. If enough space is not available, the user must choose another **unwanted** primitive. This process continues until space is available for the remaining primitives.
14. Once the primitives fit, the EEPROMs are updated and the message noted by ⓮ is printed.
15. Later ROM revisions identify both what boot primitives are availiable and what devices they support (see ⓯).
16. CPU 2 and CPU 1 have same revision and have the same boot primitives ⓰.
17. CPU 5 has been patched, and test 5 shows the primitives in ROM followed by those in EEPROM ⓱. Since the ROM is an older revision, the devices the primitives support are not printed.
18. CPU 5 and 6 are now the same ⓲.

4.19 <REFERENCE>(XRP) Registers

The <REFERENCE>(XRP) registers consist of the processor status longword, internal processor registers, <REFERENCE>(XRP) registers in <REFERENCE>(XMI) private space, <REFERENCE>(XMI) required registers, and 16 general purpose registers.

Table 4–11: <REFERENCE>(XRP) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
Kernel Stack Pointer	KSP	IPR0	R/W	1
Executive Stack Pointer	ESP	IPR1	R/W	1
Supervisor Stack Pointer	SSP	IPR2	R/W	1
User Stack Pointer	USP	IPR3	R/W	1
Interrupt Stack Pointer	ISP	IPR4	R/W	1
Reserved		IPR5–IPR7		3
P0 Base	P0BR	IPR8	R/W	1
P0 Length	P0LR	IPR9	R/W	1
P1 Base	P1BR	IPR10	R/W	1
P1 Length	P1LR	IPR11	R/W	1

Key to Types:

R–Read
W–Write
R/W–Read/write

Key to Classes:

1–Implemented by the <REFERENCE>(XRP) (as specified in the *VAX Architecture Reference Manual*).
2–Implemented uniquely by the <REFERENCE>(XRP).
3–Not implemented. Read as zero; NOP on write.
4–Access not allowed; accesses result in a reserved operand fault.
5–Accessible, but not fully implemented; accesses yield UNPREDICTABLE results.
6–Implemented by the FV64A vector module.
I–The register is initialized on <REFERENCE>(XRP) reset (power-up, system reset, and node reset).

Table 4–11 (Cont.): <REFERENCE>(XRP) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
System Base	SBR	IPR12	R/W	1
System Length	SLR	IPR13	R/W	1
Reserved		IPR14–IPR15		3
Process Control Block Base	PCBB	IPR16	R/W	1
System Control Block Base	SCBB	IPR17	R/W	1
Interrupt Priority Level	IPL	IPR18	R/W	1 I
AST Level	ASTLVL	IPR19	R/W	1 I
Software Interrupt Request	SIRR	IPR20	W	1
Software Interrupt Summary	SISR	IPR21	R/W	1 I
Reserved		IPR22–IPR23		3
Interval Counter Control and Status	ICCS	IPR24	R/W	2 I
Reserved		IPR25–IPR26		3
Time-of-Year Clock	TODR	IPR27	R/W	1
Console Storage Receiver Status	CSRS	IPR28	R/W	5 I
Console Storage Receiver Data	CSRD	IPR29	R	5 I
Console Storage Transmitter Status	CSTS	IPR30	R/W	5 I
Console Storage Transmitter Data	CSTD	IPR31	W	5 I
Console Receiver Control/Status	RXCS	IPR32	R/W	2 I
Console Receiver Data Buffer	RXDB	IPR33	R	2 I
Console Transmitter Control/Status	TXCS	IPR34	R/W	2 I
Console Transmitter Data Buffer	TXDB	IPR35	W	2 I
Reserved		IPR36–IPR37		3
Machine Check Error Summary	MCESR	IPR38	W	2
Reserved		IPR39		3
Accelerator Control and Status	ACCS	IPR40	R/W	2 I
Reserved		IPR41		3

Table 4–11 (Cont.): <REFERENCE>(XRP) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
Console Saved PC	SAVPC	IPR42	R	2
Console Saved PSL	SAVPSL	IPR43	R	2
Reserved		IPR44–IPR46		3
Translation Buffer Tag	TBTAG	IPR47	W	2
Reserved		IPR48–IPR54		3
I/O Reset	IORESET	IPR55	W	2
Memory Management Enable	MAPEN	IPR56	R/W	1 I
Translation Buffer Invalidate All	TBIA	IPR57	W	1
Translation Buffer Invalidate Single	TBIS	IPR58	W	1
Translation Buffer Data	TBDATA	IPR59	W	2
Reserved		IPR60–IPR61		3
System Identification	SID	IPR62	R	1
Translation Buffer Check	TBCHK	IPR63	W	1
Reserved		IPR64–IPR111		3
Backup Cache Reserved	BC112	IPR112	R/W	5
Backup Cache Tag Store	BCBTS	IPR113	R/W	2
Backup Cache P1 Tag Store	BCP1TS	IPR114	R/W	2
Backup Cache P2 Tag Store	BCP2TS	IPR115	R/W	2
Backup Cache Refresh	BCRFR	IPR116	R/W	2
Backup Cache Index	BCIDX	IPR117	R/W	2
Backup Cache Status	BCSTS	IPR118	R/W	2 I
Backup Cache Control	BCCTL	IPR119	R/W	2 I
Backup Cache Error	BCERR	IPR120	R	2
Backup Cache Flush Backup Tag Store	BCFBTS	IPR121	W	2
Backup Cache Flush Primary Tag Store	BCFPTS	IPR122	W	2

Table 4–11 (Cont.): <REFERENCE>(XRP) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
Reserved		IPR123		2
Vector Interface Error Status	VINTSR	IPR123	R/W	2
Primary Cache Tag Store	PCTAG	IPR124	R/W	2
Primary Cache Index	PCIDX	IPR125	R/W	2
Primary Cache Error Address	PCERR	IPR126	R/W	2
Primary Cache Status	PCSTS	IPR127	R/W	2 I
Reserved		IPR128–IPR143		3
Vector Processor Status	VPSR	IPR144	R/W	6
Vector Arithmetic Exception	VAER	IPR145	R	6
Vector Memory Activity Check	VMAC	IPR146	R	6
Vector Translation Buffer Invalidate All	VTBIA	IPR147	W	6
Reserved		IPR148–IPR156		5
Vector Indirect Register Address	VIADR	IPR157	R/W	6
Vector Indirect Data Low	VIDLO	IPR158	R/W	6
Vector Indirect Data High	VIDHI	IPR159	R/W	6

The IPRs are explicitly accessible to software only by the Move To Processor Register (MTPR) and Move From Processor Register (MFPR) instructions, which require kernel mode privileges. From the console, EXAMINE/I and DEPOSIT/I commands read and write the IPRs.

Table 4–12: <REFERENCE>(XMI) Registers for the <REFERENCE>(XRP)

Register	Mnemonic	Address
XMI Device	XDEV	BB + 00
XMI Bus Error	XBER	BB + 04
XMI Failing Address	XFADR	BB + 08
XMI GPR	XGPR	BB + 0C
<REFERENCE>(XRP) Control and Status	RCSR	BB + 10

Note: "BB" = base address of an XMI node, which is the address of the first location in nodespace.

Table 4–13: <REFERENCE>(XRP) Registers in <REFERENCE>(XMI) Private Space

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

Chapter 5

<REFERENCE>(XRV) Vector Processor

Of the two CPUs discussed in this book only the <REFERENCE>(rigel) system supports the vector processor.

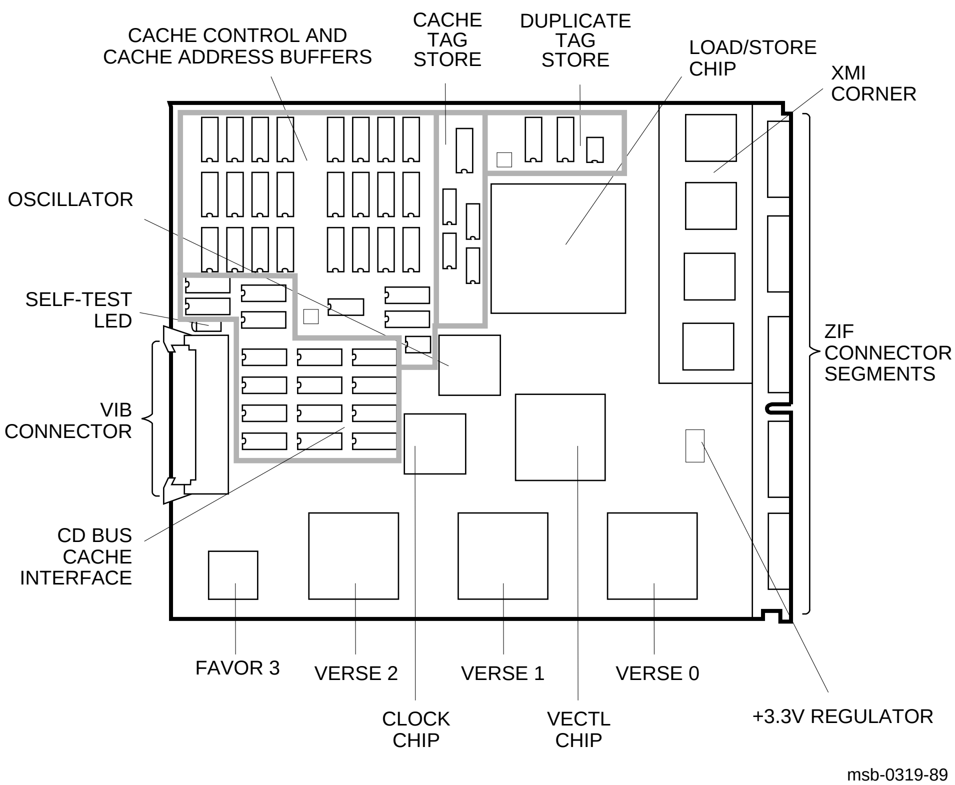
This chapter contains the following sections:

- <REFERENCE>(xrv) Physical Description and Specifications
- KA64A/FV64A Coprocessors
- <REFERENCE>(xrv) Configuration Rules
- Functional Description
- Self-Test Results: Console Display and Self-Test LED
- Self-Test Results: Scalar XGPR Register
- Vector Processor Tests — RBD 0 and RBD 1
- VAX/DS Diagnostics
- Machine Checks
- Vector Console Commands
- FV64A Handling Procedures
- How to Replace a Vector Module
- Vector Processor Registers

5.1 <REFERENCE>(xrv) Physical Description and Specifications

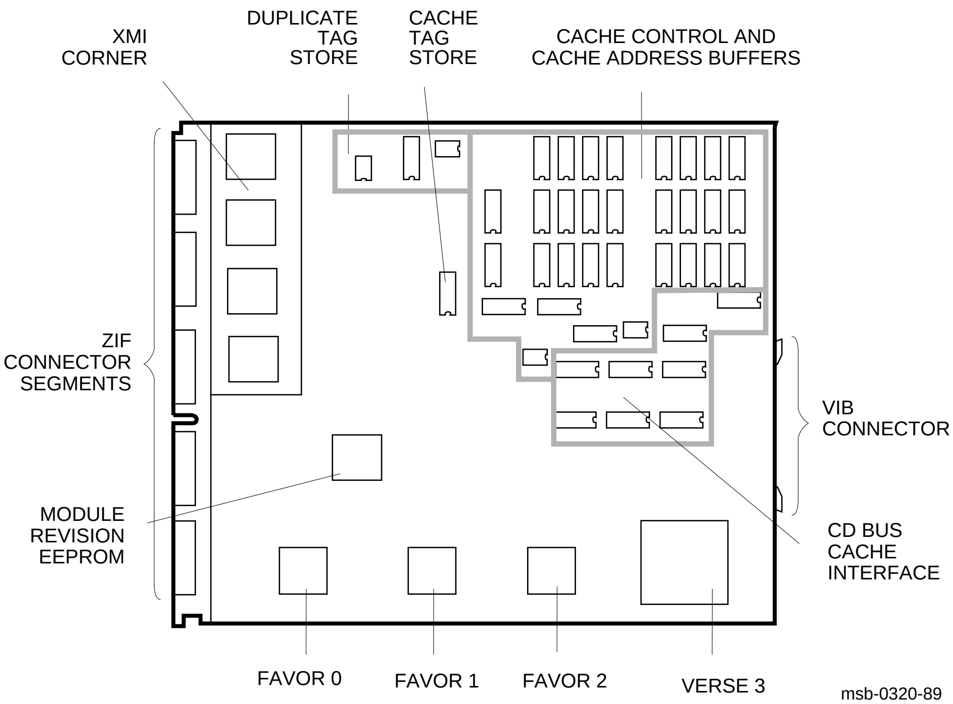
The <REFERENCE>(XRV) is a vector processor used with the <REFERENCE>(XRP) scalar processor. The module designation is T2017. The two processor modules are connected with a VIB cable. Figure 5–1 shows side 1 of the module, and Figure 5–2 shows side 2.

Figure 5–1: <REFERENCE>(XRV) Module (Side 1)



Because the vector module has components on side 2, only memory modules can be installed next to side 2 (see Figure 5–2).

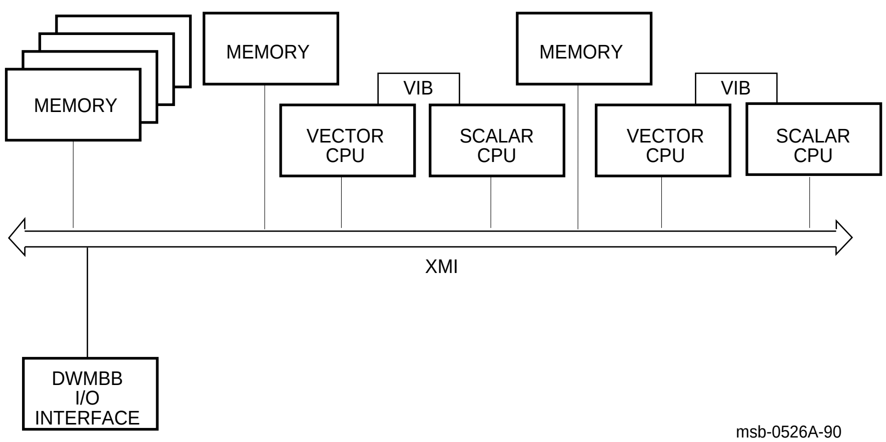
Figure 5–2: <REFERENCE>(XRV) Module (Side 2)



5.2 KA64A/FV64A Coprocessors

The <REFERENCE>(rigel) uses a high-speed system bus, called the <REFERENCE>(XMI) bus, to interconnect its processors and its memory modules. In Figure 5–3 all I/O devices connect to the VAXBI bus. The <REFERENCE>(rigel) supports multiprocessing with up to six scalar processors or one or two scalar/vector pairs.

Figure 5–3: VAX 6000 Model 400 Vector Processing System



NOTE: *Installation of an <REFERENCE>(xrv) vector processor requires that the* ***attached*** *<REFERENCE>(XRP) module (T2015) be at a minimum revision of K. In addition, the ROMs on any additional <REFERENCE>(XRP) modules must be at a minimum revision of V2.00 (ROM 0 and ROM 1).*

Table 5–1: <REFERENCE>(XRV) Specifications

Parameter	Description
Module Number:	T2017
Dimensions:	23.3 cm (9.2") H x 2.4 cm (0.94") W x 28.0 cm (11.0") D
Temperature:	
Storage Range	-40°C to 66°C (-40°F to 151°F)
Operating Range	5°C to 50°C (41°F to 122°F)
Relative Humidity:	
Storage	10% to 95% noncondensing
Operating	10% to 95% noncondensing
Altitude:	
Storage	Up to 4.8 km (16,000 ft)
Operating	Up to 2.4 km (8000 ft)
Current:	14.2A at +5V
Power:	70W
Cables:	VIB cable, 17-02240-03
Diagnostics:	ROM-based diagnostics 0 and 1. VAX/DS diagnostics, see Section 5.8.

The FV64A vector processor is an integrated vector processor; that is, the vector processor module performs as a coprocessor that is tightly coupled with a host scalar processor. The two processors are physically connected by an intermodule cable, the VIB. The scalar processor is specifically designed to support its vector coprocessor, and the vector instruction set is implemented as part of the host native instruction set. Both the scalar and vector processors are on the XMI bus, and they share a common memory.

A VAX 6000 Model 400 system can have one or two scalar/vector pairs. If the system has only one pair, it can also have additional scalar processors. See Table 6–2 for memory configuration rules related to vector processing.

5.3 <REFERENCE>(xrv) Configuration Rules

A vector processor must be installed to the left of its companion scalar processor. An intermodule cable connects the two modules. A memory module or an empty slot must be to the left of the vector processor. Any other configuration may damage the vector module.

Figure 5–4: Scalar/Vector Configurations

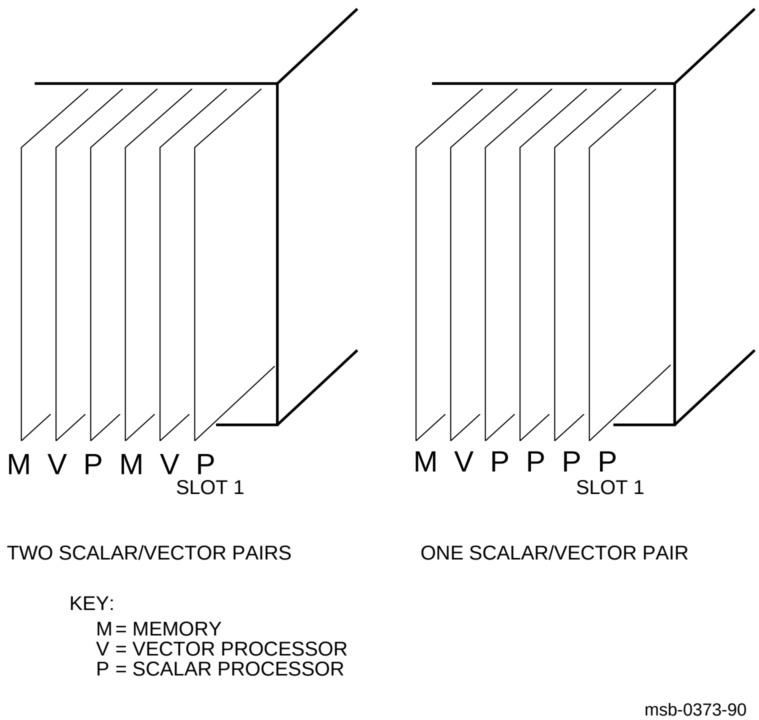


Table 5–2 shows the maximum number of scalar and vector processors supported in a VAX 6000 Model 400 system.

Table 5–2: Processor Module Combinations

Maximum CPUs	Maximum Vectors	Configuration (Slot 1 at Right)
6	0	P P P P P P
4	1	M V P P P P
2	2	M V P M V P

Figure 5–4 shows system configurations for a VAX 6000 Model 400 system with one or two vector processors. The diagram on the left indicates the configuration for two scalar/vector pairs (V- -P) with a memory module in the slot to the left of the vector processor. The diagram on the right shows a single scalar/vector pair with additional scalar processors.

Typically, processors are placed in the right <REFERENCE>(XMI) slots, beginning with slot 1 and extending to slot 6. Memories are placed in the middle slots, from slot A to slot 5 and then slots B and C, and VAXBI adapters are installed in the left side of the card cage, beginning with slot E. However, in a system with a vector processor, the modules should be installed as shown in Figure 5–4. These configurations must be followed to avoid damage to the modules and for performance reasons:

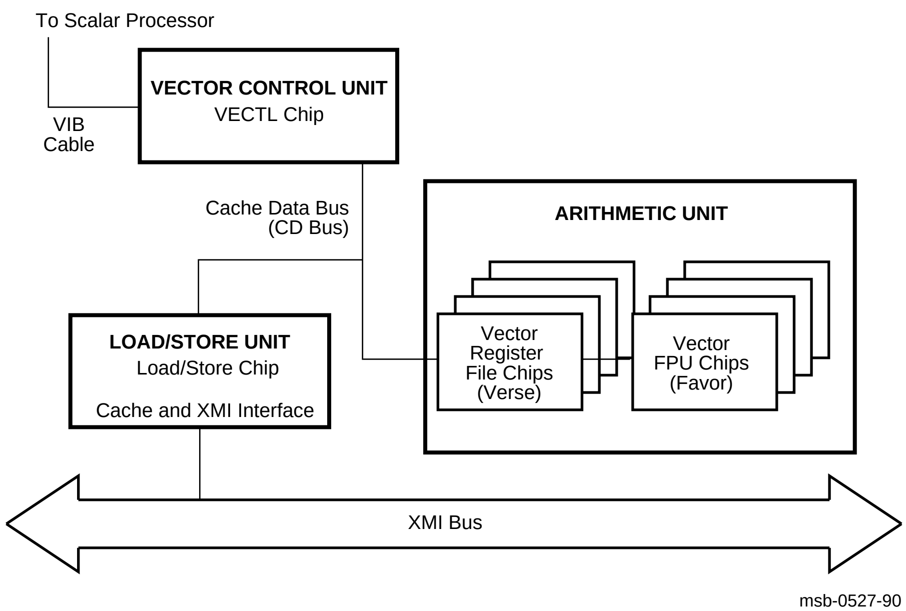
- Because the <REFERENCE>(xrv) module has VLSI components with heat sinks protruding from both sides, only a memory module, with its low components, can be placed next to side 2 of the <REFERENCE>(xrv).
- In a system with one scalar/vector pair and one or more additional scalar processors, the scalar processor of the pair should be prevented from being the boot processor for performance reasons.

  If the scalar/vector pair is to the left of other scalar processors, then the processor of the scalar/vector pair will not become the boot processor unless other processors fail self-test or have been disabled with the SET CPU console command. Alternatively, you can issue the SET CPU/NOPRIMARY command and give the node number of the attached scalar processor that you do not want to be the boot processor.

5.4 Functional Description

Figure 5–5 shows the three main functional units of the <REFERENCE>(XRV) processor: the vector control unit, the arithmetic unit, and the load/store unit, which includes the XMI interface and cache control.

Figure 5–5: <REFERENCE>(XRV) Block Diagram



The <REFERENCE>(xrv) is an integrated vector processor, tightly coupled to the <REFERENCE>(XRP) scalar processor. The vector instructions are issued from the scalar processor, and the vector processor then dispatches them internally. All communication between the scalar and vector modules takes place across the intermodule VIB cable. All communication with memory is over the XMI bus.

The vector processor has 16 vector data registers, each 64 quadwords long. There is a 1-megabyte direct-mapped cache and a 136-entry translation buffer.

The <REFERENCE>(xrv) is an XMI module with the standard XMI Corner. The module has a cable connector at the rear edge of the module that connects to the rear edge of a <REFERENCE>(XRP) module. The instructions are issued over the VIB bus and pass to the VECTL chip, which then controls the operations on the module. It passes instructions to the load/store unit over the CD bus. The load/store unit then issues XMI memory transactions. The VECTL chip also issues instructions to the four pairs of Verse and Favor chips that make up the arithmetic unit. The 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 module uses the standard XMI Corner interface, but it functions only as an XMI commander. The vector processor does not issue transactions to I/O space, nor does it respond to XMI transactions directed to it. All error reporting is done by the scalar processor.

5.5 Self-Test Results: Console Display and Self-Test LED

You can check the vector processor self-test results in three ways: the self-test display if the vector is attached to the processor in node 1, the yellow self-test LED on the <REFERENCE>(xrv) module, and the contents of the XGPR register of the attached <REFERENCE>(XRP) module. If self-test passes, the large yellow LED on the vector module lights. If the vector module fails self-test, the light remains unlit.

Example 5–1: Self-Test Results

```
#123456789 0123456789 0123456789 0123456789 0123456789 # ❶

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #

    A   A   A   .   M   M   .   .   M   V-  -P  M   V-  -P      TYP❷
    o   +   +   .   +   +   .   .   +   +   +   +   +   +       STF❸
    .   .   .   .   .   .   .   .   .   D   E   .   E   B       BPD❹
    .   .   .   .   .   .   .   .   .   +   +   +   +   +       ETF❺
    .   .   .   .   .   .   .   .   .   D   E   .   E   B       BPD❻

.   .   .   .   .   .   .   .   .   +   .   +   .   +   +   .   XBI E +

    .   .   .   .  A2   A1  .   .  B2   .   .  B1   .   .       ILV
    .   .   .   .  64   64  .   .  32   .   .  32   .   .       192Mb

ROM0 = V3.00  ROM1 = V3.00  EEPROM = 2.03/3.00  SN = SG01234567❼
>>>
```

Example 5–1 shows the self-test results for a system with two scalar/vector pairs. Each <REFERENCE>(XRP) runs its self-test and then tests any attached vector processor.

❶ The first line of the self-test printout is the progress trace. This line shows the self-test progress of the <REFERENCE>(XRP) in node 1. The numbers correspond to tests in the system self-test. If there is an attached vector processor module and self-test passes, the line prints as in Example 5–1 ending with #. If there is no attached vector processor, testing stops after the first 37 tests. If a test fails, the failing test number is the last one printed. For example, if test 14 fails, the line is printed as follows:

```
#123456789 01234
```

❷ This line indicates the type (TYP) of module at each <REFERENCE>(XMI) node. Scalar processors are type P, and vector processors are type V. The dashes indicate that the scalar processors are attached to the adjacent vector processors.

❸ This line shows self-test fail status (STF), which are the results of on-board self-test. Possible values for processors are:

+ (pass)
– (fail)

All processors passed self-test in this example.

❹ The BPD line indicates boot processor designation and whether vector processors are enabled or disabled.[1] When the system completes on-board self-test, the scalar processor with the lowest XMI ID number that passes self-test and is eligible is selected as boot processor — in this example, the processor at node 1.

The results on the BPD line indicate:

- The boot processor (B)
- Scalar processors eligible (E) or ineligible (D) to become the boot processor
- Vector processors enabled (E) or disabled (D)

In this example the vector processor attached to the scalar processor at node 4 has been disabled. A vector processor can be disabled by the SET CPU/NOVECTOR_ENABLED command.

❺ During extended test (ETF) all processors run additional tests, which include reading and writing memory and using the cache. On line ETF, results are reported for each processor in the same way as on line STF—a plus sign indicates that extended test passed and a minus sign that extended test failed.

❻ Another BPD line is displayed, because it is possible for a different CPU to be designated boot processor if the processor first designated as the boot processor fails the extended testing.

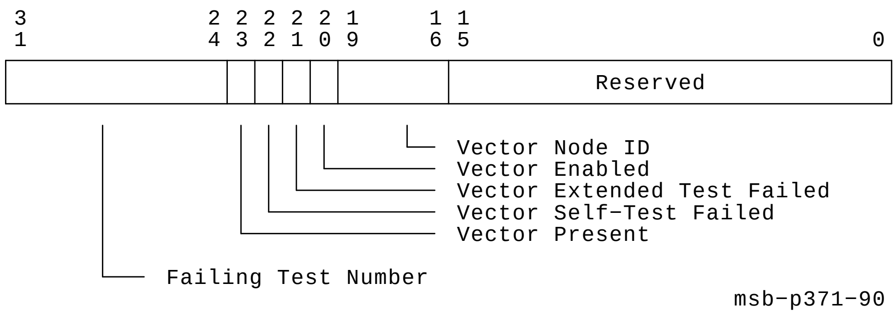
❼ The last line of the self-test display shows the ROM and EEPROM version numbers and the system serial number. Version 2 or greater ROMs and EEPROMs are required to support vector processors.

[1] If a revision J scalar processor has an attached vector module, the vector will be disabled, and this error message is displayed: ?7D Vector module is disabled–check <REFERENCE>(XRP) revision at node *n*. The *attached* scalar module (T2015) must be at a minimum revision of K. In addition, the ROMs on any other <REFERENCE>(XRP) modules must be at a minimum revision of V2.0.

5.6 Self-Test Results: Scalar XGPR Register

You can check self-test results in the self-test display or in the XGPR register. The failing test number is left in the upper byte of the XGPR register of the failing <REFERENCE>(XRP) processor.

Figure 5–6: XGPR Register



Example 5–2: XGPR Register After Power-Up Test Failure

```
>>> E/P/L 2188000C          ! Examine the longword at physical address
                            ! 2188000C, the address of the XGPR
2188000C  45F0xxxx          ! register of the processor in slot 1.
                            ! The result indicates that test 45 of
                            ! self-test failed (Load/Store Cache test).
```

Figure 5–6 shows the XGPR register of the scalar processor. Bit <23>, when set, indicates that there is a vector processor attached to this processor. Bits <22:16> give status on an attached vector processor.

The failing test number is derived from the upper byte (bits <31:24>) of the longword returned. For self-test, the upper byte contains the failing test number. If CPU/memory interaction test fails, this byte contains the failing test number plus 49. If DWMBB test fails, bit <31> is set (making the first digit 8 through A), and bits <30:24> contain the failing test number. All numbers are expressed in binary-coded decimal (BCD). See Table 5–3.

As shown in Example 5–2, you can examine the XGPR register of the failing node to determine the failing test number. See Table 4–3 to determine the base address (BB) of the <REFERENCE>(XRP) processor's node. Then calculate the address of the XGPR register by adding 0C (hex) to the base address.

Table 5–3: Interpreting XGPR Failing Test Numbers

Failing Diagnostic	XGPR <31>	XGPR <30:24> (BCD)	Test Numbers
Self-test	Clear	1–49	1–49
CPU/memory interaction test	Clear	50–66	1–17
Additional memory	Clear	67–73	3
DWMBB test	Set	1–26	1–26

5.7 Vector Processor Tests — RBD 0 and RBD 1

T0038 through T0049 of RBD 0 test the vector processor during self-test. Tests 14–17 of RBD 1 test the vector processor during CPU/memory testing.

Example 5–3: Running RBD 0 on a Secondary Processor with an Attached Vector Processor

```
>>> SET CPU 4 ❶
>>> T/R
RBD4> ST0/TR ❷

;XRP/V_ST     2.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018  T0019  T0020
; T0021  T0022  T0023  T0024  T0025  T0026  T0027  T0028  T0029  T0030
; T0031  T0032  T0033  T0034  T0035  T0036  T0037  T0038  T0039  T0040
; T0041  T0042  T0043  T0044  T0045  T0046  T0047  T0048  T0049

;       P        2     8082        1
;00000000 00000000 00000000 00000000 000000000 00000000 00000000
```

In Example 5–3:

❶ This command causes the <REFERENCE>(XRP) module at node 4 to become the primary processor.

❷ The prompt indicates that the CPU at node 4 is the primary processor. RBD 0 is run on the scalar/vector processor pair at node 4.

Table 5–4: Vector Processor Tests in Self-Test — RBD 0

Test	Function
T0038	VECTL Registers Test
T0039	Verse Registers Test
T0040	Load/Store Registers Test
T0041	VIB Error Logic Test
T0042	Other VECTL Chip Logic Test
T0043	Verse and Favor Test
T0044	Load/Store Translation Buffer and CAM Test
T0045	Load/Store Cache Test
T0046	Load/Store Instruction Test
T0047	Load/Store Tag and Duplicate Tag Test
T0048	Load/Store Error Cases Test
T0049	Module Critical Path Test

Table 5–5: Vector Tests in CPU/Memory Interaction Tests — RBD 1

Test	Function
T0014	Cache Test
T0015	Write Buffer Test
T0016	Duplicate Tag Test
T0017	Miscellaneous Error Test

5.8 VAX/DS Diagnostics

The <REFERENCE>(xrv) software diagnostics that run under the VAX Diagnostic Supervisor (VAX/DS) are listed in Table 5–6. Example 5–4 lists VAX/DS commands used in testing vector processors. See Section 2.4 for instructions on running the supervisor.

Table 5–6: <REFERENCE>(XRV) VAX/DS Diagnostics

Program	Description
ERKMP	Multiprocessor Exerciser (2 min—quick) (4 min—default)
EVKAG	VAX Vector Instruction Exerciser, Part 1 (1 1/2 min—quick) (16 min—default)
EVKAH	VAX Vector Instruction Exerciser, Part 2 (1 min—quick) (18 min—default)

Example 5–4: VAX/DS Commands for Testing Vector Processors

```
DS> RUN ERKMP        ! Multiprocessor Exerciser also tests
                     ! vector processors.
DS> SET QUICK        ! Abbreviated version of the VAX Vector
                     ! Instruction Exerciser will be run.
DS> DESELECT KA1     ! Removes the second scalar/vector pair
                     ! from testing.
DS> RUN EVKAG        ! Part 1 of VAX Vector Instruction Exerciser.
DS> RUN EVKAH        ! Part 2.

DS> BOOT n           ! If more than one vector, make the other
DS> DESELECT KA0     ! scalar of the second scalar/vector pair
DS> SELECT KA1       ! the boot processor.  Run EVKAG and EVKAH
                     ! on the second vector.
DS> BOOT n           ! Restore original boot processor.
DS> EXIT
```

5.9 Machine Checks

A machine check is an exception that indicates a processor-detected internal error. Figure 5–7 and Table 5–7 show these parameters.

Figure 5–7: The Stack in Response to a Machine Check

Table 5–7: <REFERENCE>(xrv) Machine Check Parameters

Parameter	Value (hex)	Description
Machine check code (SP+4)	14	Vector module error

Machine checks are taken regardless of the current IPL. If the machine check exception vector bits (<1:0>) are not both one, the operation of the processor is undefined. The exception is taken on the interrupt stack and the IPL is raised to 1F (hex). See Table 4–9 for the complete list of machine check codes.

5.10 Vector Console Commands

Table 4–10 gives the console commands specific to the vector processor.

Table 5–8: Vector Console Commands

Command	**Function**
DEPOSIT	Stores data in a specified address. Additional addresses can be VMR, VCR, and VLR (for Vector Mask Register, Vector Count Register, and Vector Length Register).
/M	Defines the address space as a vector indirect register; accesses addresses 400 and higher.
/Q	Quadword is the default data size for vector registers (except for VCR and VLR).
/VE	Defines the address space as the vector register set.
EXAMINE	Displays the contents of a specified address. Additional addresses can be VMR, VCR, and VLR (for Vector Mask Register, Vector Count Register, and Vector Length Register).
/M	Defines the address space as a vector indirect register; accesses addresses 400 and higher.
/Q	Quadword is the default data size for vector registers (except for VCR and VLR).
/VE	Defines the address space as the vector register set.
SET CPU	Specifies attributes of processors, such as eligibility to become the boot processor or whether a vector processor is enabled.
/NOVECTOR_ENABLED	Prevents a vector module from being recognized in the system configuration.
/VECTOR_ENABLED	Specifies that a vector module will be recognized in the system configuration; the default.

DEPOSIT Examples

1.
```
>>> DEPOSIT/VE V12 0        !  Deposits zero into all 64 elements
                            !  of vector register V12.
```

2.
```
>>> DEPOSIT V6:2C/n:2  0    !  Deposits zero into V6 beginning at
                            !  element 2C (hex) and also in the next
                            !  two elements.
```

3.
```
>>> DEPOSIT VLR 1           !  Deposits one in the Vector Length
                            !  Register.
```

4.
```
>>> DEPOSIT/Q/P 200 FFFFFFFF45370201
                            !  Deposits FFFFFFFF45370201, a quadword
                            !  of data into physical memory at address
                            !  200.
```

5.
```
>>> DEPOSIT/M  440 0        !  Deposits zeros to vector indirect
                            !  register with address 440 (hex).
```

EXAMINE Examples

1.
```
>>> EXAMINE VLR              !  Examines the Vector Length
                             !  Register.
  M 00000001 0E
```

2.
```
>>> EXAMINE/Q/P 200          !  Examines the quadword in
                             !  physical memory at address 200.
```

3.
```
>>> EXAMINE/VE V12:2E        !  Examines element 2E (hex)
                             !  (which is 41 decimal) of vector
                             !  data register V12.
```

4.
```
>>> EXAMINE/M  440           !  Examines the vector indirect
                             !  register at hex address 440.
  M 440 FFFFFFFF 00000000    !  /M is used to access vector
                             !  indirect registers.
```

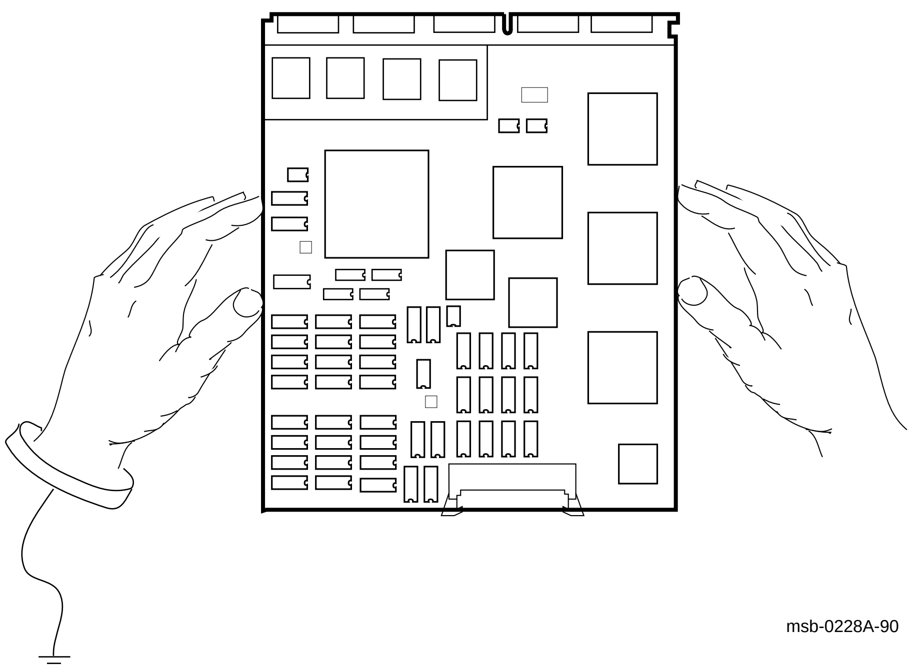
5.
```
>>> EXAMINE/VE V0          !  Examines vector register V0; system
                           !  displays all 64 elements of register V0.

  VE V00:00  00000000 00000002      VE V00:01  00000000 00000002
  VE V00:02  00000000 00000002      VE V00:03  00000000 00000002
  VE V00:04  00000000 00000002      VE V00:05  00000000 00000002
  VE V00:06  00000000 00000002      VE V00:07  00000000 00000002
  VE V00:08  00000000 00000002      VE V00:09  00000000 00000002
  VE V00:0A  00000000 00000002      VE V00:0B  00000000 00000002
  VE V00:0C  00000000 00000002      VE V00:0D  00000000 00000002
  VE V00:0E  00000000 00000002      VE V00:0F  00000000 00000002
  VE V00:10  00000000 00000002      VE V00:11  00000000 00000002
  VE V00:12  00000000 00000002      VE V00:13  00000000 00000002
  VE V00:14  00000000 00000002      VE V00:15  00000000 00000002
  VE V00:16  00000000 00000002      VE V00:17  00000000 00000002
  VE V00:18  00000000 00000002      VE V00:19  00000000 00000002
  VE V00:1A  00000000 00000002      VE V00:1B  00000000 00000002
  VE V00:1C  00000000 00000002      VE V00:1D  00000000 00000002
  VE V00:1E  00000000 00000002      VE V00:1F  00000000 00000002
  VE V00:20  00000000 00000002      VE V00:21  00000000 00000002
  VE V00:22  00000000 00000002      VE V00:23  00000000 00000002
  VE V00:24  00000000 00000002      VE V00:25  00000000 00000002
  VE V00:26  00000000 00000002      VE V00:27  00000000 00000002
  VE V00:28  00000000 00000002      VE V00:29  00000000 00000002
  VE V00:2A  00000000 00000002      VE V00:2B  00000000 00000002
  VE V00:2C  00000000 00000002      VE V00:2D  00000000 00000002
  VE V00:2E  00000000 00000002      VE V00:2F  00000000 00000002
  VE V00:30  00000000 00000002      VE V00:31  00000000 00000002
  VE V00:32  00000000 00000002      VE V00:33  00000000 00000002
  VE V00:34  00000000 00000002      VE V00:35  00000000 00000002
  VE V00:36  00000000 00000002      VE V00:37  00000000 00000002
  VE V00:38  00000000 00000002      VE V00:39  00000000 00000002
  VE V00:3A  00000000 00000002      VE V00:3B  00000000 00000002
  VE V00:3C  00000000 00000002      VE V00:3D  00000000 00000002
  VE V00:3E  00000000 00000002      VE V00:3F  00000000 00000002
```

5.11 <REFERENCE>(xrv) Handling Procedures

Handle the processor modules with care. The technology used on the later 6000 series modules is more vulnerable to static than earlier technology. Also, these modules have 25 mil leads to the chips; these leads are small, close together, and easily bent.

Figure 5–8: Holding the <REFERENCE>(xrv) Module



The later 6000 series modules require careful handling. Prepare yourself and the work area before handling these modules. Roll up your sleeves and remove any jewelry. Figure 5–8 shows the proper way to hold these modules.

Follow these handling procedures to avoid damaging the processor modules:

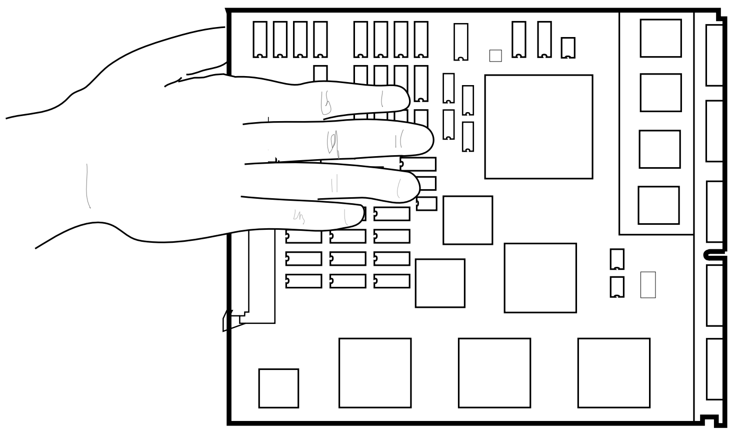
1. Always wear an antistatic wrist strap.
2. Before removing the module from its ESD box, place the box on a clean, stable surface.

   Be sure the box will not slide or fall. **Never** place the box on the floor. And be sure no tools, papers, manuals, or anything else that might damage the module is near it. Some components on this module can be damaged by a 600-volt static charge; paper, for example, can carry a charge of 1000 volts.
3. Hold the module only by the edges, as shown in Figure 5–8.

   Do not hold the module so that your fingers touch any 25 mil devices, leads, or XMI fingers. Be sure you do not bend the module as you are holding it.
4. Be sure nothing touches the module surface or any of its components.

   If anything touches the module, components or leads can be damaged. This includes the antistatic wrist strap, clothing, jewelry, cables, components on other modules, and anything in the work area (such as tools, manuals, or loose papers).

Figure 5–9: Inserting the <REFERENCE>(xrv) Module in an XMI Card Cage



You must take special precautions when moving the processor modules in or out of the XMI card cage.

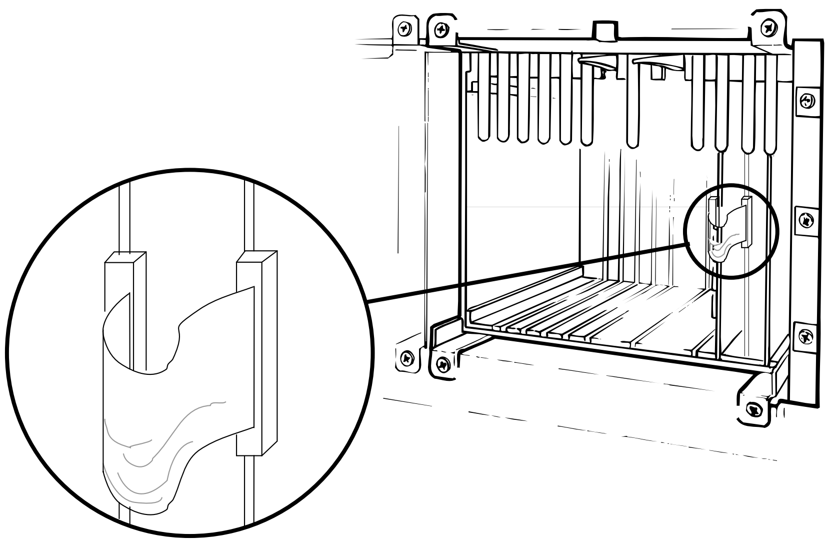
1. Be sure, when inserting the module in or removing it from the XMI card cage, that no part of the module comes in contact with another module or a cable. The leads on the components are fragile and can be damaged by contact with fingers or any surface.
2. When you swap out a module, place it in the correct ESD box before you install the new module.
3. Hold the XMI card cage handle while removing or inserting the module. If it is not held in place, the handle can spring down and damage the module.
4. When inserting the module in the card cage, grasp it as shown in Figure 5–9, being careful not to touch any 25 mil devices, and slide it slowly and gently into the slot.
5. **Do not attach the repair tag to the module.**

   Place the repair tag in the plastic bag attached to the bottom of the ESD box. Allowing the repair tag to come in contact with the module can cause damage to a component.

5.12 How to Replace a Vector Module

If a vector module is defective, you can replace it with a new one. If you install an additional one, see the complete installation instructions in the VAX 6000 Series Upgrade Manual.

Figure 5–10: Replacing a Vector Module in an XMI Card Cage



msb-0407-90

CAUTION: *Special care must be taken when handling processor modules. See Section 5.11 before replacing this module. Also review the configuration rules in Section 5.3.*

While removing or inserting a module in the XMI card cage, you must hold the XMI card cage lever. Failure to do so may result in damage to the module.

1. Turn the upper key switch straight up to the Off position (0).
2. Open the cabinet door and remove the plastic door in front of the XMI card cage.

   CAUTION: *You must wear an antistatic wrist strap attached to the cabinet when you handle any modules.*
3. Disconnect the VIB cable (17-02240-03) from the vector module.
4. Remove the defective vector processor module.
5. Take the new vector processor module from the ESD box and insert it in the XMI card cage. Place the defective module in the ESD box.
6. Attach the connecting VIB (vector interface bus) cable. The keyed end of the cable attaches to the vector module.
7. Press the lever down to close the connector.
8. Replace the plastic door and shut the cabinet door.
9. Turn the lower key switch to Halt and the upper key switch to Enable.
10. Check the self-test display for the new vector processor, indicated by a V on the TYP line.
11. If the processor shows a plus sign (+) on both lines STF and ETF, it passed self-test.

NOTE: *Installation of an <REFERENCE>(xrv) vector processor requires that the* ***attached*** *<REFERENCE>(XRP) module (T2015) be at a minimum revision of K. In addition, the ROMs on any additional <REFERENCE>(XRP) modules must be at a minimum revision of V2.0 (ROM 0 and ROM 1).*

5.13 Vector Processor Registers

The <REFERENCE>(XRV) internal processor registers are listed in Table 5–9. See Chapter 4 for the complete list of IPR registers. The console program allows you to access the vector registers. Software accesses the vector registers with MTPR/MFPR and MTVP/MFVP instructions.

Table 5–9: <REFERENCE>(XRV) Internal Processor Registers

Register	Mnemonic	Address	Type	Class
Vector Interface Error Status	VINTSR	IPR123	R/W	1
Vector Processor Status	VPSR	IPR144	R/W	2
Vector Arithmetic Exception	VAER	IPR145	R	2
Vector Memory Activity Check	VMAC	IPR146	R	2
Vector Translation Buffer Invalidate All	VTBIA	IPR147	W	2
Vector Indirect Register Address	VIADR	IPR157	R/W	2
Vector Indirect Data Low	VIDLO	IPR158	R/W	2
Vector Indirect Data High	VIDHI	IPR159	R/W	2

Key to Types:

R–Read
W–Write
R/W–Read/write

Key to Classes:

1–Implemented by the <REFERENCE>(XRP) CPU module.
2–Implemented by the <REFERENCE>(XRV) vector module.

The IPRs listed in Table 5–9 are explicitly accessible to software only by the Move To Processor Register (MTPR) and Move From Processor Register (MFPR) instructions, which require kernel mode privileges. (The vector indirect registers are also accessed with MTPR and MFPR instructions. These registers are described in the *System Technical User's Guide*.)

From the console, EXAMINE/I and DEPOSIT/I commands read and write the IPRs. EXAMINE/M and DEPOSIT/M commands provide access to the vector indirect registers above hex address 400. EXAMINE/VE and DEPOSIT/VE provide access to the vector data registers.

Other instructions, the Move To/From Vector Processor (MTVP/MFVP) instructions, are used by software to access the Vector Length, Vector Count, and Vector Mask control registers. From the console, these registers are specified as VLR, VCR, and VMR after DEPOSIT and EXAMINE commands, with no qualifiers.

For more information on accessing the vector module registers, see the *VAX 6000 Series Vector Processor Owner's Manual*.

Chapter 6
MS65A Memory

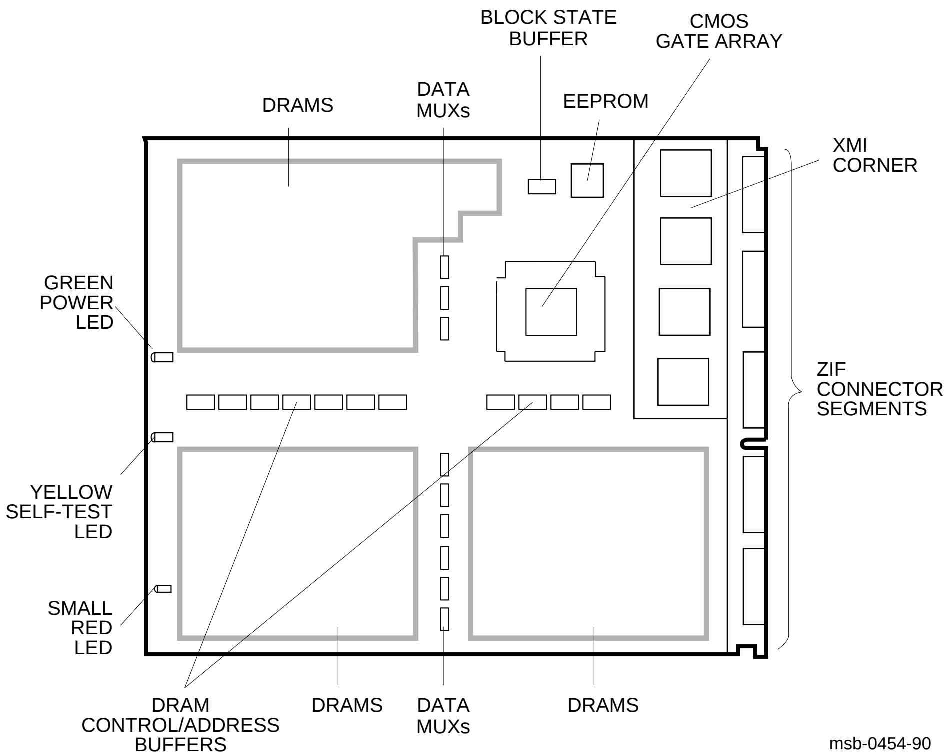
This chapter discusses the <REFERENCE>(xma2) memory module. Sections include:

- Physical Description
- Configuration Rules
- Specifications
- Functional Description
- System Interleaving Requirements
- MS65A Interleaving
- Console Commands for Interleaving
- Addressing
- Memory Self-Test
- Memory Self-Test Errors
- Memory RBD
- Memory RBD Test Examples
- Control and Status Registers
- Mixing MS65A and MS62A Memory Modules

6.1 MS65A Physical Description

The <REFERENCE>(xma2) memory module is a metal-oxide semiconductor (MOS), dynamic random access memory (DRAM), which provides up to 128 Mbytes of data storage. The memory module is designed for use with the <REFERENCE>(6000) system through the XMI bus.

Figure 6–1: MS65A Module



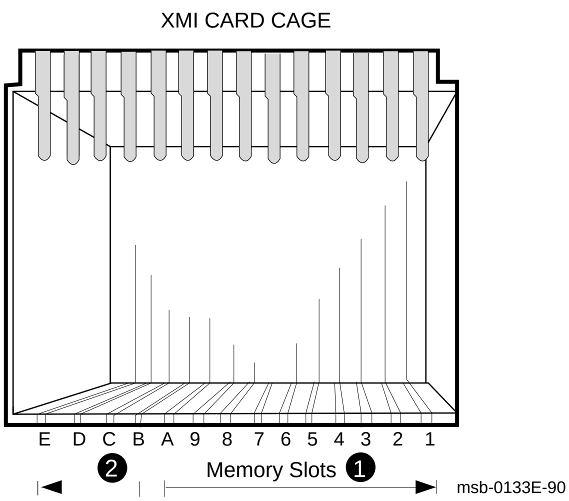
The <REFERENCE>(xma2) memory module has the following features:

- 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 known as the XMI Corner.
- Storage arrays are made up of two or four banks, either 155 or 299 DRAMs.
- ECC logic detects single-bit and double-bit errors and corrects single-bit errors on 64-bit words.
- Memory self-test checks all RAMs, the data path, and control logic on power-up.
- Quadwords, octawords, and hexwords can be read from or written to memory.
- Memory is configured by the console program for 2-, 4-, 8-way or no interleaving.

6.2 MS65A Configuration Rules

Figure 6–2 shows the order of placement of MS65A modules in the XMI backplane.

Figure 6–2: MS65A Configuration



Memory modules are configured after I/O adapter and processor modules. Install memory modules next to vector processors first, then install additional memories as follows:

❶ Install the first memory module in slot A. Fill all available slots left to right from slot A to slot 1.

❷ Install any additional memory modules right to left in available slots from slot B to slot E.

6.3 MS65A Specifications

Table 6–1 gives the <REFERENCE>(xma2) module specifications.

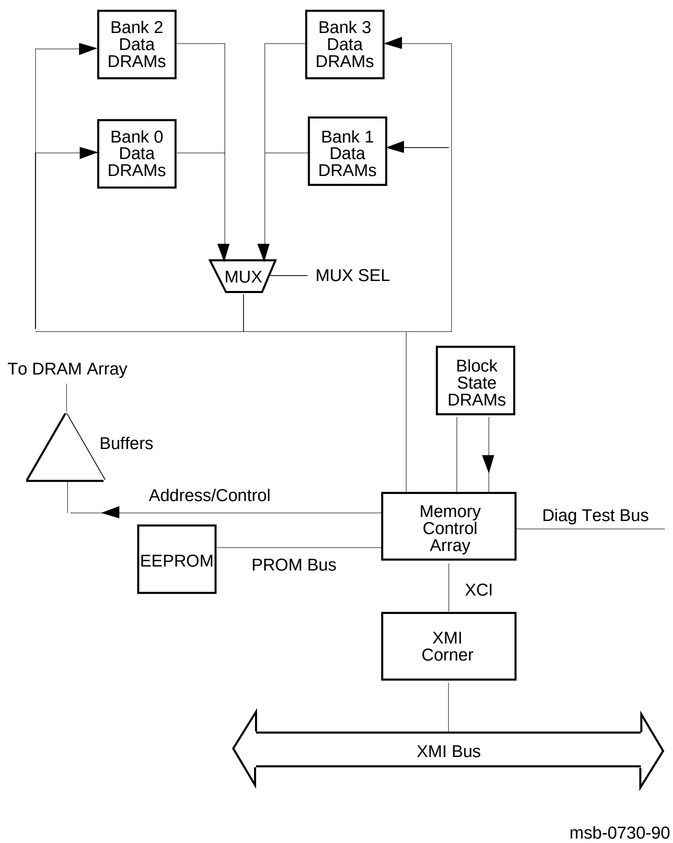
Table 6–1: <REFERENCE>(xma2_title) Specifications

Parameter	Description
Module Number:	T2053
Dimensions:	23.3 cm (9.2") H and 28.0 cm (11.0") D
Memory size:	MS65A-BA 32 Mbytes MS65A-CA 64 Mbytes MS65A-DA 128 Mbytes
Addresses:	16-Mbyte boundaries
Starting Address	0 Mbytes to 512 Mbytes
Ending Address	0 to 512 Mbytes
Technology:	
DRAMS	1 or 4 Mbit dynamic RAMs
Gate Arrays	CMOS memory control gate array
Interleave:	2-, 4-, 8-way or none
Error Correction Code:	Detects single- and double-bit errors and corrects single-bit errors
Temperature:	
Storage Range	–40°C to 66°C (–40°F to 151°F)
Operating Range	5°C to 50°C (41°F to 122°F)
Relative Humidity:	
Storage and Operating	10 to 95% noncondensing
Altitude:	
Storage	Up to 4.8 km (16,000 ft)
Operating	Up to 2.4 km (8000 ft)
Current:	10A active, 3.8A standby, max. at +5V
Power:	50W active, 19W standby, max. at +5V

6.4 MS65A Functional Description

The <REFERENCE>(xma2) consists of an XMI Corner, a memory control gate array, address and control drivers, block state DRAMs, DRAM arrays, and an EEPROM.

Figure 6–3: MS65A Block Diagram



The XMI Corner is located on the <REFERENCE>(xma2) module and contains interface logic.

The memory control gate array transfers data between the XMI Corner and the DRAMs. The memory control gate array also controls address multiplexing, command decoding, arbitration, and CSR logic functions.

Address and control logic modifies address bits received from the XMI Corner. These modified address bits are used to control the selection of the DRAMs during reading and writing.

Memory is arranged in two or four banks of DRAMs. Each bank contains either 155 or 299 DRAMs on each memory module.

The data in the EEPROM is used to initialize the memory control gate array. After a power-up or system reset, the data in the EEPROM is loaded into the memory control gate array.

6.5 System Interleaving Requirements

System performance is affected by the speed with which memory responds to requests made of its resources. For this reason, <REFERENCE>(HYPERION) and <REFERENCE>(rigel) systems have memory interleaving requirements. Those requirements are given in Table 6–2.

Table 6–2: VAX 6000 Interleaving Requirements

Interleave Factor	Model 300	Model 400	Model 400 with Vectors
One way	310 320 330	410 420	
Two way	340 350 360	430 440	410 or 420 with 1 vector
Four way		450 460	420 with 2 vectors 430 or 440 with 1 vector

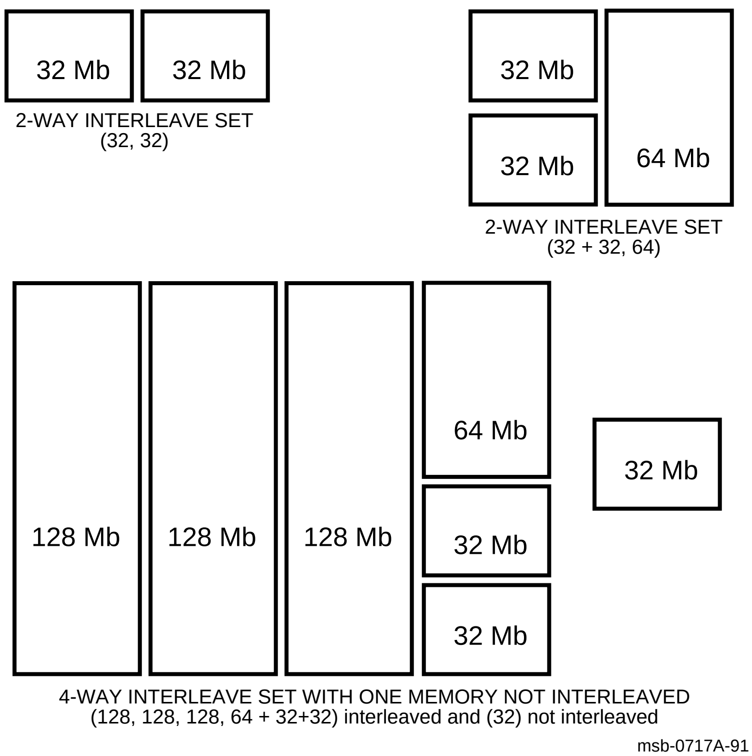
The interleave configuration requirements shown in Table 6–2 are necessary for system performance. Differences between the two systems are primarily due to cache differences on the two CPUs.

Note that memory size is not the issue. However, to achieve 4-way interleaving the minimum memory capacity is 128 Mbytes (4 x 32-Mbyte arrays).

6.6 MS65A Interleaving

Interleaving optimizes memory access time and increases the effective memory transfer rate by operating memory modules in parallel.

Figure 6–4: Examples of Interleaving



Memory supports 2-, 4-, 8-way or no interleaving. Up to eight memory modules of the same size can be interleaved. Memory modules of different sizes can also be interleaved. Figure 6–4 shows three examples of interleaving.

- A 2-way interleave set is made from two, same sized, arrays.
- A 2-way interleave set is made from three memory arrays of different sizes.
- The final example shows how the console builds a 4-way interleave set from several different array sizes.

Interleaving is done on hexword boundaries. Interleaving addresses are set in the Starting and Ending Address Register by the console program (see Section 6.8). The <REFERENCE>(xma2) does not check for valid or invalid interleaving configurations.

NOTE: *Memory modules that fail self-test due to multiple-bit errors are not included in the interleave set.*

When different sizes of memory modules are installed in a <REFERENCE>(6000) system, the console interleaves the memory modules according to size and sets as follows:

- Sorts memory modules into groups by size.
- Interleaves the largest size memory modules first.
- Stacks remaining sets of modules into sets that equal the largest size memory modules and interleaves them with the largest size memory modules.
- Stacks remaining modules into sets of the next largest size memory modules and interleave them.
- Continues stacking and interleaving memory modules until all memory modules have been interleaved (including noninterleaved modules).

Unless the system requires a specific, dedicated memory use, let the console set the default interleave set rather than setting interleaving manually. In default, the console program chooses the optimal configuration for the system.

6.7 Console Commands for Interleaving

The SET MEMORY command is used to set memory interleaving in a configuration other than the default. This is not usually advisable, but occasional customer use will warrant overriding the original console setting of the interleave. The INITIALIZE command causes both <REFERENCE>(6000) systems to execute <REFERENCE>(xma2) self-tests and the SET MEMORY commands to take effect.

Example 6–1: SET MEMORY and INITIALIZE Commands

```
>>> SET MEMORY  /INTERLEAVE:DEFAULT ❶
                           ! For a system with one 64-Mbyte and two
                           ! 32-Mbyte memory modules, it creates a 2-way
                           ! interleave of 64-Mbyte memory modules
                           ! (1x64-Mbyte and 2x32-Mbyte memory modules)
                           ! located at XMI nodes A, 9, and 8.

>>> SHOW MEMORY ❷          ! Displays the memory lines from self-test.

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #
    .   .   .   .   A2  A2  A1  .   .   .   .   .   .   .       ILV
    .   .   .   .   32  32  64  .   .   .   .   .   .   .       128Mb

    /INTERLEAVE:DEFAULT

>>> SET MEMORY  /INTERLEAVE:(8, 9+A) ❸
                           ! Explicitly specifies what is created
                           ! as requested by the user (two interleave
                           ! sets with modules in nodes 8, 9, and A).

>>> INITIALIZE ❹           ! Enables the SET MEMORY command to take effect

         [self-test display]

>>> SHOW MEMORY ❺          ! Displays the memory lines from self-test.

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #
    .   .   .   .   B2  B1  A1  .   .   .   .   .   .   .       ILV
    .   .   .   .   32  32  64  .   .   .   .   .   .   .       128Mb

    /INTERLEAVE:(8, 9+A)

>>>
```

The callouts in Example 6–1 are explained below.

❶ Shows the SET MEMORY command that configures interleaving with the console program. This command invokes the default interleaving configuration. It is recommended that this default be used, rather than trying to interleave memory manually.

❷ The SHOW MEMORY command displays the node number (node #), interleave (ILV), and total usable memory (xxMb) lines from the self-test results.

❸ Shows the SET MEMORY command that creates a 2-way interleave as requested by the user. In this example the user explicitly specified how to interleave the memory modules. Each interleaving set must contain the node number of the memory module. If there is more than one memory module in a set, they are joined by a + sign. Each set of interleaved memory modules must be separated by a comma.

❹ The system is initialized and the new memory interleave configuration takes effect. You **cannot** initialize a memory node by itself after a SET MEMORY command because memory addressing would be changed and data structures lost.

❺ The SHOW MEMORY command displays the configuration set in ❸.

NOTE: *Refer to Chapter 5 of the VAX 6000 Series Owner's Manual for detailed information on the SET MEMORY and SHOW MEMORY commands.*

The SET MEMORY command does not change memory interleaving immediately; it just modifies the memory configuration in the EEPROM. The memory configuration specified by the SET MEMORY command takes place when the system is initialized (by a power-up or INITIALIZE command).

6.8 MS65A Addressing

Memory addressing is set on hexword boundaries and depends on the interleaving sets organized by the console. Starting and ending addresses are determined by the console regardless of how interleaving is done (by the user or by the console).

Figure 6–5: Memory Addressing

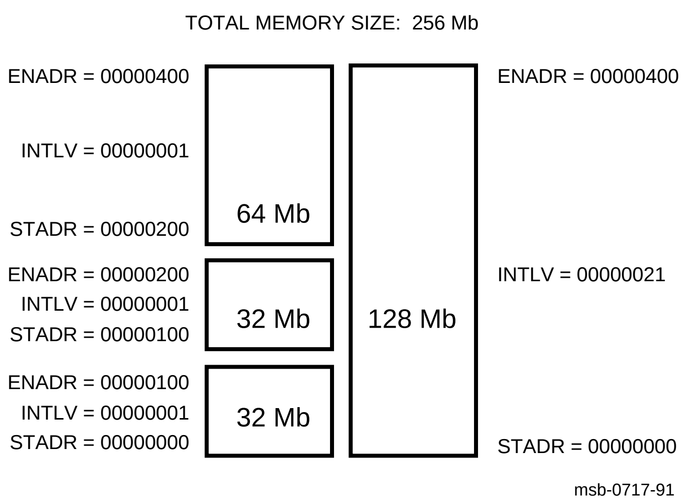


Figure 6–5 shows the starting address (STADR), ending address (ENADR), and interleave (INTLV) registers of a sample interleave set. The contents of these registers are set by the console.

The memory shown in Figure 6–5 is divided into two interleaving sets and totals 256 Mbytes. Set 0 consists of two 32-Mbyte arrays and one 64-Mbyte array. Set 1 consists of one 128-Mbyte array.

The starting address of the first array is 0. The ending address is determined by multiplying the density of the array by the interleave factor (number of sets). For example, the starting address of the first array in set 0 is 0, and the ending address is 100 hex (64 decimal, which is equal to 32 multiplied by 2). The starting address of the second array is the same as the ending address of the first.

Each array's interleave register indicates the set it belongs to (bits <7:5>) and the total number of interleave sets (bits <1:0>). The interleave register for the 128-Mbyte array indicates that the array is set 1 (bits <7:5>=001) of two interleave sets (bits <1:0>=01).

6.9 Memory Self-Test

The <REFERENCE>(xma2) performs an initialization and self-test sequence on power-up or when the sequence is requested by a console command. During self-test the array chip is initialized, all memory locations are tested, and the control and status registers are initialized.

Example 6–2: MS65A Memory Module Results in Self-Test

```
#123456789 0123456789 0123456789 01234567#

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #
    A   A   .   .   M   M   M   M   .   .   .   P   P   P       TYP    ❶
    +   +   .   .   -   +   +   +   .   .   .   +   +   +       STF    ❷
    .   .   .   .   .   .   .   .   .   .   .   E   E   B       BPD
    .   .   .   .   .   .   .   .   .   .   .   +   +   +       ETF
    .   .   .   .   .   .   .   .   .   .   .   E   E   B       BPD

    .   .   .   .   C1  B1  A2  A1  .   .   .   .   .   .       ILV    ❸
    .   .   .   .   64  64  64  64  .   .   .   .   .   .       256Mb  ❹

ROM0 = V3.00  ROM1 = V3.00  EEPROM = 2.03/3.00   SN = SG01234567

>>>
```

The callouts in Example 6–2 are explained below.

❶ The **TYP** line shows that memory modules are installed in XMI slots 7 through A as indicated by the M in this row.

❷ The **STF** line shows if memory modules pass self-test, as indicated by the + in this row. If a module fails self-test, a – is indicated, but the console still tests all pages within the module. The failing module is included in the configuration, and the addresses that fail self-test are not used by the system.

❸ The **ILV** line indicates that two memory array modules are 2-way interleaved and the other two modules are interleaved by themselves. That is, memory modules in slots 7 and 8 are two-way interleaved into one interleave set (indicated by all modules beginning with the letter A). The module at node A failed its self-test and although used by the system it is not included in an interleave set.

❹ This system contains a total usable memory of 256 Mbytes (four 64-Mbyte memory modules).

If all <REFERENCE>(xma2) nodes pass self-test, the CPU/memory test is performed on the <REFERENCE>(xma2) by the CPU. The console executes a simple read/write test to a small portion of memory. Since there are no errors from the self-test, the memory bitmap is set with all pages as good.

6.10 Memory Self-Test Errors

If an <REFERENCE>(xma2) node fails self-test, an explicit memory test is run on the failing module and console error messages are displayed. The failing module is still included in the memory configuration.

Example 6–3: MS65A Memory Module Node Exclusion

```
>>>  SET MEMORY  /INTERLEAVE:(8+9, A)
>>>  INITIALIZE
>>>  SHOW MEMORY

F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #
    .   .   .   .   B1  A2  A1  -   .   .   .   .   .   .       ILV
    .   .   .   .   64  64  64  .   .   .   .   .   .   .       192Mb
    /INTERLEAVE:(8+9, A)
```

If an <REFERENCE>(xma2) node fails self-test, then the console executes an explicit memory test during the building of the bitmap. Failing memory modules are included in the configuration, although they are interleaved by themselves. The only way to exclude a memory module from interleaving is to use the SET MEMORY command without designating the node you want to exclude. Example 6–3 shows how to exclude the memory module at node 7.

During the explicit memory test, any number of the following console messages might be displayed to aid the customer service engineer in diagnosing the problem.

```
?37  Explicit interleave list is bad. Configuring
     all arrays uninterleaved.
```

This means that the explicit set of memory arrays for the explicit interleave includes no nodes that contain memory arrays. All memory arrays found in the system are unconfigured (the SET MEMORY command may have specified nodes that did not contain memory modules).

```
?45  Memory interleave set is inconsistent: n n ...(<REFERENCE>(xyp) msg.)
?46  Memory interleave set is inconsistent: n n ...(<REFERENCE>(XRP) msg.)
```

This means that the listed nodes *(n n)* do not form a valid memory interleave set. One or more of the nodes might not be a memory array or the set contains an invalid number of memory arrays. Each listed memory array that is valid will be configured uninterleaved; any memory array that is not included in the set will not be interleaved.

```
?46  Insufficient working memory for normal operation. (<REFERENCE>(xyp) msg.)
?47  Insufficient working memory for normal operation. (<REFERENCE>(XRP) msg.)
```

This means that less than 256 Kbytes per processor of working memory were found. There may be insufficient memory for the console to function or for the operating system to boot.

```
?47  Uncorrectable memory errors -- long memory test
     must be performed. (<REFERENCE>(xyp) message)
?48  Uncorrectable memory errors -- long memory test
     must be performed. (<REFERENCE>(XRP) message)
```

This means that a memory array contains an unrecoverable error. The console must perform a slow test to locate all the failing locations. (Slow = approximately 3 minutes per 32 Mbytes.)

```
?48  Memory not interleaved due to Uncorrectable errors. (<REFERENCE>(xyp))
?4A  Memory not interleaved due to Uncorrectable errors. (<REFERENCE>(XRP))
```

This means that the listed arrays would normally have been interleaved (by default or an explicit request). Because one or more arrays contained unrecoverable errors, this interleave set will not be constructed.

NOTE: *Refer to Appendix A for a list of <REFERENCE>(xyp) console error messages and Appendix B for a list of <REFERENCE>(XRP) console error messages.*

When self-test has finished running on the module, the yellow LED on the module lights, indicating that the module has completed self-test. After self-test, starting and ending addresses are set by the boot processor.

Note that lighting the yellow LED does **not** mean that memory has passed self-test but only that self-test completed.

6.11 Memory RBD

RBD 3 of the ROM-based diagnostics sizes memory, runs extended memory tests, and indicates any failing tests.

Table 6–3: Memory Tests — RBD 3

Test	Function
T0001[1]	Memory Self-Test (13 sec[2,3])
T0002[4]	CSR Addressability Test
T0003[4]	CSR Bit Toggling Test
T0004[4]	Parity Error Detection Test
T0005[4]	Error Detection and Correction Logic Test
T0006[4]	Data Path Test
T0007[4]	Quadword and Octaword Masked Write Logic Test
T0008[4]	Interlock Lock Logic Test
T0009[1]	Interleaving and Address Boundary Test (20 sec[2])
T0010[1]	ECC RAM March Test (20 min[2])
T0011[1]	RAM March Test (9 min–RAM; 17 min–ROM[2])
T0012[1]	RAM Moving Inversions Test (2.5 hrs–RAM; 4.5 hrs–ROM[2])

[1]The /C qualifier is required for these tests.
[2]Run times are approximate for one 32-Mbyte module.
[3]If self-test fails, there is a 60 second timeout.
[4]Tests T0002 through T0008 are run by default.

Tests T0002 through T0008 are run by default. Tests T0001 and T0009 through T0012 must be selected by the user. Tests are performed on all <REFERENCE>(XMA2)s unless the user specifies a single <REFERENCE>(XMA2). Parameters specified in the command line (refer to Table 6–4) allow one or all memory modules to be tested. These parameters also allow RBD tests to be run from main memory or ROM for RBD tests T0011 and T0012. The /C (confirm destructive memory test) switch is required with RBD tests T0001, T0009, T0010, T0011, and T0012. Parameters are ignored by tests T0001 through T0010.

Table 6–4: RBD 3 Parameters

KA64A Parameter[1]	KA62B Parameter	Function
00 (or none)	1n	Run tests T0011 and T0012 from main memory (RAM) and test all memory modules
0n	1n	Run tests T0011 and T0012 from main memory (RAM) and test memory module n only
10	2n	Run tests T0011 and T0012 from ROM and test all memory modules
1n	3n	Run tests T0011 and T0012 from ROM and test memory module n only

[1]The first character indicates if the tests are run from RAM (0) or ROM (1). The second character indicates whether to test all modules (0) or a specific module (n), where *n* is the memory module backplane slot (must be one of the following: A, 9, 8, 7, 6, 5, B, or C).

NOTE: *If you suspect that all of memory is bad, run tests T0011 and T0012 from ROM.*

The CPU/memory interaction diagnostic also runs tests that exercise memory. See Chapter 3 for information on this CPU/memory interaction diagnostic. See Chapter 2 for more information on running the RBDs.

6.12 Memory RBD Test Examples

RBD 3 sizes memory, runs extended memory tests, and indicates any failing tests. Example 6–4 through Example 6–6 show the use of various qualifiers.

Example 6–4: RBD 3 — Test on All Modules with Halt on Error

```
>>> T/R                        ! Command to enter RBD monitor program

RBD3>                          ! RBD monitor prompt, where 3 is the hexa-
                               ! decimal node number of the processor
                               ! that is currently receiving your input

RBD3> ST3/TR/HE                ! Runs the default <REFERENCE>(XMA2) RBD
                               ! test; test results written to the
                               ! console terminal; tests will halt on
                               ! any hard error found (/HE)
;XMA_RBD        3.00

; T0002  T0003  T0004  T0005  T0006  T0007  T0008

;       S        3     8082        1  ❶
;      XX  NO_XMA2       09    T0008

;       P        3     8082        1
;00000000 00000000 00000000 00000000 00000000 00000000 00000000

RBD3>
```

Example 6–5: RBD 3 — Test with Confirm Switch

```
RBD3> ST3/TR/T=9:10/C          ! Runs the <REFERENCE>(XMA2) RBD tests
                               ! T0009 and T0010 only.  These are
                               ! destructive tests, so the confirm
                               ! switch is needed.  Confirm destructive
                               ! memory test switch (/C) is required
                               ! on tests T0001 and T0009 through T0012.
;XMA_RBD        3.00

; T0009  T0010

;       P        3     8082        1
;00000000 00000000 00000000 00000000 00000000 00000000 00000000

RBD3>
```

Example 6–6: RBD 3 — Parameter

```
RBD3> ST3/TR/T=11:12/C 0A     ! Runs <REFERENCE>(XMA2) RBD tests
                              ! T0011 and T0012 from RAM on the
                              ! the memory module in slot A. Confirm
                              ! destructive memory test switch (/C)
                              ! is required on these tests.
;XMA_RBD        3.00

; T0011  T0012

       [RBD status messages are printed every two minutes;
        use the /DS qualifier in the command string to inhibit
        these messages.]

;       P        3     8082        1
;00000000 00000000 00000000 00000000 00000000 00000000 00000000

RBD3>  QUIT                   ! Exit from RBD monitor program
?06 Halt instruction executed in kernel mode.
PC  = 200601D8
PSL = 041F0604
ISP = 201405B4

>>>                           ! Console prompt returns
```

In Example 6–4 note the status message in the display (❶). The message fields are as follows:

- S = Status message
- 3 = Node from which the RBD is running
- 8082 = Device type of node from which RBD is running
- 1 = Pass 1
- XX = Undefined
- NO_XMA2 = RBD does not perform parts of test 8 on MS65A memories
- 09 = Node where MS65A resides
- T0008 = Test 8 produced the status message

The RBD completes with the normal pass/fail message. Only test 8, the Interlock Lock Logic Test, behaves this way; all other tests are run on both MS62As and MS65As.

6.13 MS65A Control and Status Registers

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 to the CSRs. If a parity error occurs during a write operation, the operation is aborted and the contents of the CSRs are unchanged.

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.

Table 6–5: <REFERENCE>(xma2) Control and Status Registers

CSR Name	Mnemonic	Address
Device Type Register	XDEV	BB[1] + 00
Bus Error Register	XBER	BB + 04
Starting and Ending Address Register	SEADR	BB + 10
Memory Control Register 1	MCTL1	BB + 14
Memory ECC Error Register	MECER	BB + 18
Memory ECC Address Register	MECEA	BB + 1C
Memory Control Register 2	MCTL2	BB + 30
TCY Tester Register	TCY	BB + 34
Block State ECC Error Register	BECER	BB + 38
Block State ECC Address Register	BECEA	BB + 3C
Starting Address Register	STADR	BB + 50
Ending Address Register	ENADR	BB + 54
Segment/Interleave Control Register	INTLV	BB + 58
Memory Control Register 3	MCTL3	BB + 5C
Memory Control Register 4	MCTL4	BB + 60

[1]"BB" refers to the base address of an <REFERENCE>(XMI) node (2180 0000 + (node ID x 8000))

Table 6–5 (Cont.): <REFERENCE>(xma2) Control and Status Registers

CSR Name	Mnemonic	Address
Block State Control Register	BSCTL	BB + 68
Block State Address Register	BSADR	BB + 6C
EEPROM Control Register	EECTL	BB + 70
Timeout Control/Status Register	TMOER	BB + 74

<

6.14 Mixing MS65A and MS62A Memory Modules

This section discusses the interleaving of MS62A and <REFERENCE>(xma2) memory modules in Model 300 and Model 400 systems. For completeness the VAX 6000 Model 200 and the VAX 6000 Model 500 are included.

Table 6–6: Memory Module Configurations

System Type	MS65A	MS62A	Mixed
Model 200[1]	Yes	Yes	Yes
Model 300[2]	Yes	Yes	Yes
Model 400[3]	Yes	Yes	Yes
Model 500	Yes	No	No

[1]The minimum CPU ROM revision required is 5.00.
[2]The minimum CPU ROM revision required is 6.00.
[3]The minimum CPU ROM revision required is 3.00.

If <REFERENCE>(xma2)s or a mix of memory modules are used in a Model 200, 300, or 400 system, a ROM upgrade kit is required. The console and diagnostic ROMs are replaced, so that <REFERENCE>(xma2)s will be supported. See the footnotes in Table 6–6 for the minimum revisions of the console and diagnostic ROMs for each type of system.

There is no difference between MS62A memory modules and <REFERENCE>(xma2)s as far as interleaving is concerned. The console software that does the interleaving only recognizes the size difference (number of Mbytes) of the memory module.

Memory ROM-based diagnostics function with both types of memory modules. However, since the <REFERENCE>(xma2) has an EEPROM and the MS62A memory module does not, the RBD does not test the EEPROM on the <REFERENCE>(xma2). The following <REFERENCE>(xma2) RBDs will not run on <REFERENCE>(xma2)s installed in Model 200, 300, or 400 systems.

- SEADR Register Test
- Block State Test
- EEPROM Update Test

Status messages will print on the console terminal stating that the diagnostic test does not run.

Chapter 7
DWMBB I/O Adapter

This chapter discusses the DWMBB adapter, the interface to an optional VAXBI I/O channel. Sections include:

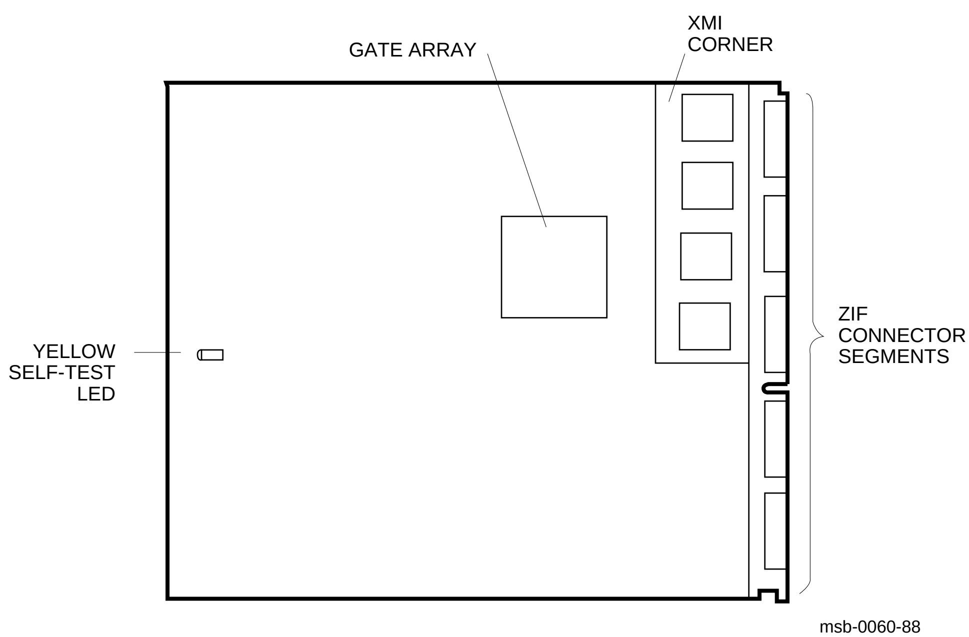
- Physical Description
    - Physical Layout
    - Specifications
- Configuration Rules
- Functional Description
- Registers

7.1 DWMBB Physical Description

7.1.1 Physical Layout

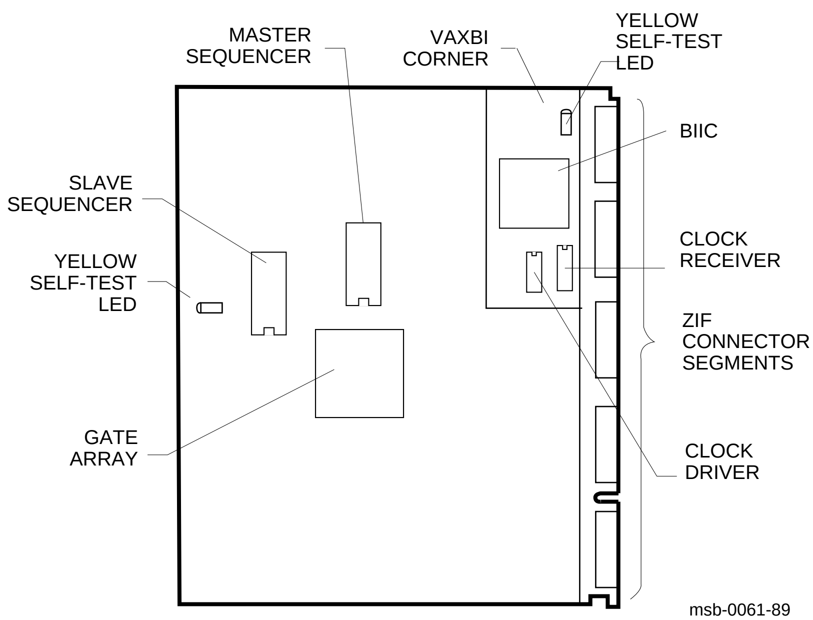
The <REFERENCE>(xbia) is an XMI module (T2018) with the standard XMI Corner, an XMI self-test OK LED indicator, IBUS drivers/receivers and transceivers, timeout logic, and a gate array that controls the <REFERENCE>(xbia). Most of the components on the <REFERENCE>(xbia) are surface-mounted.

Figure 7–1: <REFERENCE>(XBIA_TITLE)



The <REFERENCE>(XBIB) is a standard VAXBI (T1043) module with a VAXBI Corner, including a BIIC interface chip, the primary interface between the VAXBI bus and the <REFERENCE>(XBIB) node logic, a clock driver, and a clock receiver. The <REFERENCE>(XBIB) gate array is used mostly for data path logic. The VAXBI self-test OK LED is on the VAXBI Corner, and the module self-test OK LED is at the module edge opposite the connector edge.

Figure 7–2: <REFERENCE>(XBIB_TITLE)



7.1.2 Specifications

The following specifications apply to the DWMBB modules.

Table 7–1: DWMBB/A Specifications

Parameter	Description
Module Number:	T2018
Dimensions:	23.3 cm (9.2") H x 0.23 cm (0.093") W x 28.0 cm (11.0") D
Temperature:	
Storage Range	-40°C to 66°C (-40°F to 151°F)
Operating Range	5°C to 50°C (41°F to 122°F)
Relative Humidity:	
Storage and operating	10% to 95% noncondensing
Altitude:	
Storage	Up to 4.8 km (16,000 ft)
Operating	Up to 2.4 km (8000 ft)
Current:	6A at +5V
Power:	16W

Table 7–2: DWMBB/B Specifications

Parameter	Description
Module Number:	T1043
Dimensions:	20.3 cm (8") H x 0.23 cm (0.093") W x 23.3 cm (9.2") D
Temperature:	
Storage Range	-40°C to 66°C (-40°F to 151°F)
Operating Range	5°C to 50°C (41°F to 122°F)
Relative Humidity:	
Storage and operating	10% to 95% noncondensing
Altitude:	
Storage	Up to 4.8 km (16,000 ft)
Operating	Up to 2.4 km (8000 ft)
Current:	6A at +5V
	10mA at -12V
Power:	30W

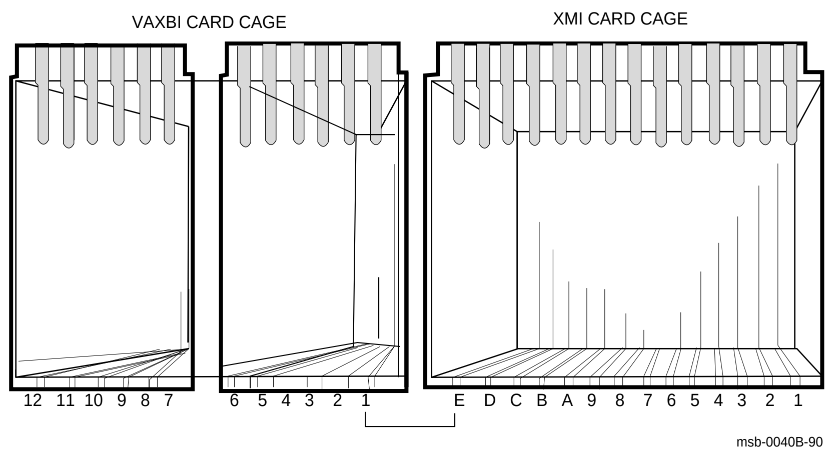
Table 7–3: <REFERENCE>(xbi_plus) Cables

Part Number	Description
17-01569-01	DWMBB to H7206-B power OK cable
17-01897-01	15' DWMBB cables for expander cabinet, from XMI slots 1, 2, 3, and 4 as needed (segments D and E) to VAXBI cages 2, 3, 4, and 5 (segments D and E). Two per DWMBB.
17-01897-02	7" DWMBB cables, from XMI slot E (segments D and E) to VAXBI cage 1 slot 1 (segments D and E). Two per DWMBB.

7.2 <REFERENCE>(xbi_plus) Configuration Rules

This section describes the configuration rules for the DWMBB/A module in the XMI card cage and for the DWMBB/B module in the VAXBI card cage.

Figure 7–3: VAX 6000 Slot Numbers



<REFERENCE>(XBIA) modules are placed in the order shown in Table 7–4.

Table 7–4: <REFERENCE>(xbi_plus) Configuration

<REFERENCE>(XMI) Node No.	VAXBI Channel	Location
E	1	System cabinet
1	2	Expander cabinet
2	3	Expander cabinet
3	4	Expander cabinet
4	5	Expander cabinet

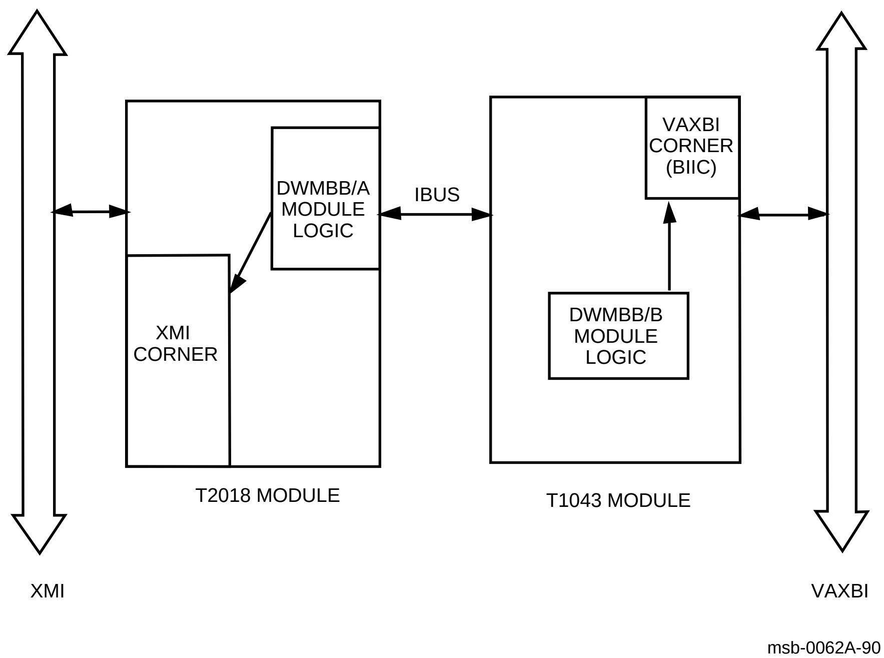
Configuration rules are as follows:

- The first VAXBI channel is the 12-slot channel in the system cabinet. The DWMBB/A module is placed in XMI slot E; the corresponding DWMBB/B module is placed in the system VAXBI cage, slot 1 (the rightmost slot). See Figure 7–3.
- Any additional VAXBI channels are 6-slot channels in the expander cabinet. The DWMBB/B module is placed in slot 1 of each. The corresponding DWMBB/A module is placed in the XMI slot listed in Table 7–4.

7.3 DWMBB Functional Description

The <REFERENCE>(xbi_plus) adapter provides an information path between the <REFERENCE>(XMI) bus and I/O devices on the VAXBI bus. The <REFERENCE>(xbi_plus) consists of two modules: the <REFERENCE>(XBIA) and the <REFERENCE>(XBIB). The <REFERENCE>(XBIA) resides on the <REFERENCE>(XMI) bus, and the <REFERENCE>(XBIB) resides on the VAXBI bus. Four 30-pin cables, which make up the IBUS, connect the two modules.

Figure 7–4: <REFERENCE>(XBI_TITLE) Block Diagram



The <REFERENCE>(XBIA) contains the <REFERENCE>(XMI) Corner, the register files, <REFERENCE>(XMI) required registers, <REFERENCE>(XBIA)-specific registers, page map registers, and the control sequencers for the <REFERENCE>(XMI) interface.

The <REFERENCE>(XBIB) contains the BIIC interface chip, interconnect drivers, control sequencers to handle the control of the data transfer, status bits to and from the <REFERENCE>(XBIA) module's register files and the BIIC, <REFERENCE>(XBIB)-specific registers, decode logic for direct memory access (DMA) operation, and VAXBI clock-generation circuitry.

The <REFERENCE>(XBIA) and <REFERENCE>(XBIB) modules are connected by four cables of 30 wires each. These 120 wires make up the IBUS, which transfers data and control information between the two modules.

The <REFERENCE>(xbi_plus) uses I/O and DMA transactions to exchange information. I/O transactions originate from the <REFERENCE>(XYP) or <REFERENCE>(XRP) module(s) and are presented to the <REFERENCE>(xbi_plus) from the <REFERENCE>(XMI) bus with the processor as the <REFERENCE>(XMI) commander and the <REFERENCE>(xbi_plus) as the <REFERENCE>(XMI) responder.

DMA transactions originate from VAXBI nodes that select the <REFERENCE>(xbi_plus) as the VAXBI slave. These are read or write transactions targeted to <REFERENCE>(XMI) memory space or are VAXBI-generated interrupt transactions that target a <REFERENCE>(xyp) or <REFERENCE>(XRP) module. For DMA transactions, the <REFERENCE>(xbi_plus) is the <REFERENCE>(XMI) commander, and the <REFERENCE>(XMA) module is the <REFERENCE>(XMI) responder.

The <REFERENCE>(xbi_plus) can be both a master and a slave on the VAXBI. As a master, it carries out transactions requested by its <REFERENCE>(XMI) devices. As a slave, it responds to VAXBI transactions that select its node.

7.4 <REFERENCE>(xbi_plus) Tests — RBD 2

The <REFERENCE>(xbi_plus) ROM-based diagnostic, RBD 2, checks functions of both <REFERENCE>(xbi_plus) modules. RBD 2 tests the <REFERENCE>(xbi_plus) modules and can trace the subtests, pinpointing errors.

Example 7–1: <REFERENCE>(xbi_plus) Tests — RBD 2

```
>>> T/R                        ! Command to enter RBD monitor program

RBD1>                          ! RBD monitor prompt, where 1 is the
                               ! hexadecimal node number of the
                               ! processor that is currently receiving
                               ! your input.

RBD1> ST2/TR E ❶                ! Runs the DWMBB RBD, testing
                               ! the DWMBB at XMI node number E. Test
                               ! results written to the console
                               ! terminal:

;XBI_TEST      3.00

; T0001  T0005  T0006  T0007  T0008  T0009  T0012  T0013  T0014  T0015
; T0016  T0017  T0018  T0019  T0020  T0021  T0022  T0023  T0024  T0025

;        P ❷        1      8082         1 ❸
;000000000  00000000  00000000  00000000  00000000  00000000  00000000

RBD1> ST2/TR/T=1:26 E ❹        ! Runs the DWMBB RBD, testing
                               ! the DWMBB at XMI node number E. Test
                               ! results written to the console
                               ! terminal:

;XBI_TEST      3.00

; T0001  T0002  T0003  T0004  T0005  T0006  T0007  T0008  T0009  T0010
; T0011  T0012  T0013  T0014  T0015  T0016  T0017  T0018  T0019  T0020
; T0021  T0022  T0023  T0024  T0025  T0026

;        P          1      8082         1
;000000000  00000000  00000000  00000000  00000000  00000000  00000000
```

The <REFERENCE>(xbi_plus) has no on-board self-test. The boot processor ROM code tests <REFERENCE>(xbi_plus)s during additional power-up tests. At power-up, the boot processor first sizes all <REFERENCE>(xbi_plus)s and then serially tests each one.

❶ When invoking RBD 2 from the monitor, the START command requires a parameter. This parameter is the XMI node number (hexadecimal) of the <REFERENCE>(xbia_plus) module of the <REFERENCE>(xbi_plus) to be tested.

❷ This diagnostic ran successfully.

❸ One pass was completed.

❹ This command runs all the tests in RBD 2.

7.5 DWMBB Tests — RBD 2 Subtests

RBD 2 consists of 26 tests, as shown in Table 7–5.

Table 7–5: <REFERENCE>(xbi_plus_title) RBD Tests

Test	Function	Default
T0001	DWMBB/A CSR Test	Yes
T0002	<REFERENCE>(XMI) Low Longword Parity Error Test	No
T0003	<REFERENCE>(XMI) High Longword Parity Error Test	No
T0004	<REFERENCE>(XMI) Function and ID Parity Error Test	No
T0005	<REFERENCE>(xbib_plus) CSR Test	Yes
T0006	BIIC VAXBI Loopback Transaction Test	Yes
T0007	BIIC VAXBI Transaction Test	Yes
T0008	DMA Test	Yes
T0009	DMA Buffer Test	Yes
T0010	XMI Parity Error Interrupt Test	No
T0011	Write Sequence Error Interrupt Test	No
T0012	CPU Buffer C/A Fetch Parity Error (Interrupt) Test	Yes
T0013	CPU Buffer Data Fetch Parity Error (Interrupt) Test	Yes
T0014	DMA Buffer Data Fetch Parity Error (Interrupt) Test	Yes
T0015	VAXBI Interlock Read Error (Interrupt) Test	Yes
T0016	DMA-A Buffer C/A Load Parity Error (Interrupt) Test	Yes
T0017	DMA-A Buffer Data Load Parity Error (IVINTR) Test	Yes
T0018	DMA-B Buffer C/A Load Parity Error (Interrupt) Test	Yes
T0019	DMA-B Buffer Data Load Parity Error (IVINTR) Test	Yes
T0020	CPU Buffer Data Load Parity Error (Interrupt) Test	Yes
T0021	BCI Parity Error Test	Yes
T0022	Nonexistent Memory (Interrupt) Test	Yes
T0023	CRD Error (Interrupt) Test	Yes

Table 7–5 (Cont.): <REFERENCE>(xbi_plus_title) RBD Tests

Test	Function	Default
T0024	VAXBI Interrupt Test	Yes
T0025	VAXBI IP Interrupt Test	Yes
T0026	No Stall Timeout Test	Yes

7.6 <REFERENCE>(xbi_plus) Registers

Two sets of registers are used by the <REFERENCE>(xbi_plus) adapter: VAXBI registers (residing in the BIIC) and <REFERENCE>(xbi_plus) registers (residing on both modules of the <REFERENCE>(xbi_plus)). The <REFERENCE>(xbi_plus) registers include the <REFERENCE>(XMI) required registers and <REFERENCE>(xbi_plus)-specific registers addressed in <REFERENCE>(xbi_plus) private space.

Table 7–6: VAXBI Registers

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

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

Table 7–6 lists the VAXBI registers. The VAXBI registers are described in Chapter 5 of the *VAXBI Options Handbook*. Table 7–7 lists the <REFERENCE>(xbi_plus) registers.

Table 7–7: <REFERENCE>(xbi_plus) XMI Registers

Name	Mnemonic[1]	Address[2]
Device Register	XDEV	BB+00
Bus Error Register	XBER	BB+04
Failing Address Register	XFADR	BB+08
Responder Error Address Register	AREAR	BB+0C
Error Summary Register	AESR	BB+10
Interrupt Mask Register	AIMR	BB+14
Implied Vector Interrupt Destination/Diagnostic Register	AIVINTR	BB+18
Diagnostic 1 Register	ADG1	BB+1C
Utility Register	AUTLR	BB+20
Control and Status Register	ACSR	BB+24
Return Vector Register	ARVR	BB+28
XMI Failing Address Extension Register	XFAER	BB+2C
VAXBI Error Address Register	ABEAR	BB+30
Control and Status Register	BCSR	BB+40
Error Summary Register	BESR	BB+44
Interrupt Destination Register	BIDR	BB+48
Timeout Address Register	BTIM	BB+4C
Vector Offset Register	BVOR	BB+50
Vector Register	BVR	BB+54
Diagnostic Control Register 1	BDCR1	BB+58
Reserved Register	BRSVD	BB+5C

[1]If the first letter of the mnemonic is "X" or "A," it indicates that the register resides on the <REFERENCE>(XBIA) module; a first letter of "B" indicates that the register resides on the <REFERENCE>(XBIB) module.

[2]The abbreviation "BB" refers to the base address of an <REFERENCE>(XMI) node (the address of the first location of nodespace).

Table 7–7 (Cont.): <REFERENCE>(xbi_plus) XMI Registers

Name	Mnemonic[1]	Address[2]
Page Map Register (first location)	PMR	BB+200
.		
.		
.		
Page Map Register (last location)	PMR	BB+401FC

Appendix A

Console Error Messages for Model 300

Table A–1 lists messages ?02 through ?1F which appear when the processor halts and the console gains control. Each message is followed by a "PC = xxxxxxxx" message giving the address where the processor was executing when it halted; these messages designate the reasons for the halt.

Table A–2 lists the standard console error messages ?20 through ?7C.

Table A–1: Model 300 Console Error Messages Indicating Halt

Error Message	Meaning
?02 External halt.	CTRL-P or STOP command.
?04 Interrupt stack not valid.	Interrupt stack pointer contained an invalid address.
?05 Machine check during exception.	A machine check occurred while handling another error condition.
?06 Halt instruction executed.	The CPU executed a Halt instruction.
?07 SCB vector bits <1:0> = 11.	An interrupt or exception vector in the System Control Block contained an invalid address.
?08 SCB vector bits <1:0> = 10.	An interrupt or exception vector in the System Control Block contained an invalid address.
?0A CHMx executed while on interrupt stack.	A change-mode instruction was issued while executing on the interrupt stack.
?0B CHMx to interrupt stack.	The System Control Block vector for a change-mode instruction indicated service on the interrupt stack.
?0C SCB read error.	The System Control Block was not accessible in memory.
?10 ACV or TNV during machine check.	An access violation or translation-not-valid error occurred while handling another error condition.

Table A–1 (Cont.): Model 300 Console Error Messages Indicating Halt

Error Message	Meaning
?11 ACV or TNV during KSP not valid.	An access violation or translation-not-valid error occurred while handling another error condition.
?12 Machine check during machine check.	A machine check occurred while handling another error condition.
?13 Machine check during KSP not valid.	A machine check occurred while handling another error condition.
?19 PSL <26:24>= 101 during interrupt or exception.	An exception or interrupt occurred while on the interrupt stack but not in kernel mode.
?1A PSL <26:24>= 110 during interrupt or exception.	An exception or interrupt occurred while on the interrupt stack but not in kernel mode.
?1B PSL <26:24>= 111 during interrupt or exception.	An exception or interrupt occurred while on the interrupt stack but not in kernel mode.
?1D PSL <26:24> = 101 during REI.	An REI instruction attempted to restore a PSL with an invalid combination of access mode and interrupt stack bits.
?1E PSL <26:24> = 110 during REI.	An REI instruction attempted to restore a PSL with an invalid combination of access mode and interrupt stack bits.
?1F PSL <26:24> = 111 during REI.	An REI instruction attempted to restore a PSL with an invalid combination of access mode and interrupt stack bits.

Table A–2: Model 300 Standard Console Error Messages

Error Message	Meaning
?20 Illegal memory reference.	An attempt was made to reference a virtual address (/V) that is either unmapped or is protected against access under the current PSL.
?21 Illegal command.	The command was not recognized, contained the wrong number of parameters, or contained unrecognized or inappropriate qualifiers.
?22 Illegal address.	The specified address was recognized as being invalid, for example, a general purpose register number greater than 15.
?23 Value is too large.	A parameter or qualifier value contained too many digits.
?24 Conflicting qualifiers.	A command specified recognized qualifiers that are illegal in combination.
?25 Checksum did not match.	The checksum calculated for a block of X command data did not match the checksum received.
?26 Halted.	The processor is currently halted.
?27 Item was not found.	The item requested in a FIND command could not be found.
?28 Timeout while waiting for characters.	The X command failed to receive a full block of data within the timeout period.
?29 Machine check accessing memory.	Either the specified address is not implemented by any hardware in the system, or an attempt was made to write a read-only address, for example, the address of the 33rd Mbyte of memory on a 32-Mbyte system.
?2A Unexpected machine check or interrupt.	A valid operation within the console caused a machine check or interrupt.
?2B Command is not implemented.	The command is not implemented by this console.
?2C Unexpected exception.	A valid operation within the console caused an exception.

Table A–2 (Cont.): Model 300 Standard Console Error Messages

Error Message	Meaning
?2D For Secondary Processor *n*	This message is a preface to second message describing some error related to a secondary processor. This message indicates which secondary processor is involved.
?2E Specified node is not an I/O adapter.	The referenced node is incapable of performing I/O or did not pass its self-test.
?30 Write to Z command target has timed out.	The target node of the Z command is not responding.
?31 Z connection terminated by ^P.	A CTRL/P was typed on the keyboard to terminate a Z command.
?32 Your node is already part of a Z connection.	You cannot issue a Z command while executing a Z command.
?33 Z connection successfully started.	You have requested a Z connection to a valid node.
?34 Specified target already has a Z connection.	The target node was the target of a previous Z connection that was improperly terminated. Reset the system to clear this condition.
?36 Command too long.	The command length exceeds 80 characters.
?37 Explicit interleave list is bad. Configuring all arrays uninterleaved.	The list of memory arrays for explicit interleave includes no nodes that are actually memory arrays. All arrays found in the system are configured.
?38 Waiting for a CR to terminate the command.	The command has not yet been issued by a carriage return.
?39 Console patches are not useable.	The console patch area in EEPROM is corrupted or contains a patch revision that is incompatible with the console ROM.
?3B Error encountered during I/O operation.	An I/O adapter returned an error status while the console boot primitive was performing I/O.

Table A–2 (Cont.): Model 300 Standard Console Error Messages

Error Message	Meaning
?3C Secondary processor not in console mode.	The primary processor console needed to communicate with a secondary processor, but the secondary processor was not in console mode. STOP the node or reset the system to clear this condition.
?3D Error initializing I/O device.	A console boot primitive needed to perform I/O, but could not initialize the I/O adapter.
?3E Timeout while sending message to secondary.	A secondary processor failed to respond to a message sent from the primary. The primary sends such messages to perform console functions on secondary processors.
?3F Key switch must be at "Update" to update EEPROM.	A SET command needs to update the EEPROM, but the key switch is not set to allow updates.
?40 Specified node is not a bus adapter.	A command that accesses a VAXBI node specified an XMI node that was not a bus adapter.
?41 Invalid terminal speed.	The SET TERMINAL command specified an unsupported baud rate.
?42 Unable to initialize node.	The INITIALIZE command failed to reset the specified node.
?43 Processor is not enabled to BOOT or START.	As a result of a SET CPU/NOENABLE command, the processor is disabled from leaving console mode.
?44 Unable to stop node.	The STOP command failed to halt the specified node.
?45 Memory interleave set is inconsistent: *n n ...*	The listed nodes do not form a valid memory interleave set. One or more of the nodes might not be a memory array, or the set could contain an invalid number of members. Each listed array that is a valid memory will be configured uninterleaved.

Table A–2 (Cont.): Model 300 Standard Console Error Messages

Error Message	Meaning
?46 Insufficient working memory for normal operation.	Less than 256 Kbytes per processor of working memory were found. There is insufficient memory for the console to function normally or for the operating system to boot.
?47 Uncorrectable memory errors—long memory test must be performed.	A memory array contains an unrecoverable error. The console must perform a slow test to locate all the failing locations.
?49 Memories not interleaved due to uncorrectable errors:	The listed arrays would normally have been interleaved (by default or explicit request). Because one or more of them contained unrecoverable errors, this interleave set will not be constructed.
?4A Internal logic error in console.	The console encountered a theoretically impossible condition.
?4B Invalid node for Z command.	The target of a Z command must be a CPU or an I/O adapter and must not be the primary processor.
?4C Invalid node for new primary.	The SET CPU command failed when attempting to make the specified node the primary processor.
?4D Specified node is not a processor.	The specified node is not a processor, as required by the command.
?4E System serial number has not been initialized.	No CPU in the system contains a valid system serial number.
?4F System serial number not initialized on primary processor.	The primary processor has an uninitialized system serial number. All other processors in the system contain a valid serial number.
?50 Secondary processor returned bad response message.	A secondary processor returned an unintelligible response to a request made by the console on the primary processor.
?51 ROM revision mismatch. Secondary processor has revision *x.y*.	The revision of console ROM of a secondary processor does not match the primary.

Table A–2 (Cont.): Model 300 Standard Console Error Messages

Error Message	Meaning
?52 EEPROM header is corrupted.	The EEPROM header has been corrupted. The EEPROM must be restored from the TK tape drive.
?53 EEPROM revision mismatch. Secondary processor has revision *x.y/x.y*.	A secondary processor has a different revision of EEPROM or has a different set of EEPROM patches installed.
?54 Failed to locate EEPROM area.	The EEPROM did not contain a set of data required by the console. The EEPROM may be corrupted.
?55 Console parameters on secondary processor do not match primary.	Console parameters do not match on the primary and secondary processors.
?56 EEPROM area checksum error.	A portion of the EEPROM is corrupted. It may be necessary to reload the EEPROM from the TK tape drive.
?57 Saved boot specifications on secondary processor do not match primary.	Saved boot specifications do not match on the primary and secondary processors.
?58 Invalid unit number.	A BOOT or SET BOOT command specifies a unit number that is not a valid hexadecimal number between 0 and FF.
?59 System serial number mismatch. Secondary processor has xxxxxxxx.	The indicated serial number of a secondary processor does not match the primary.
?5A Unknown type of boot device.	The console program does not have a boot primitive to support the specified type of device or the device could not be accessed to determine its type.
?5B No HELP is available.	The HELP command is not supported when the console language is set to International.
?5C No such boot spec found.	The specified saved boot specification was not found in the EEPROM.
?5D Saved boot spec table full.	The maximum number of saved boot specifications has already been stored.

Table A–2 (Cont.): Model 300 Standard Console Error Messages

Error Message	Meaning
?5E EEPROM header version mismatch.	The primary and a secondary processor have different versions of the EEPROM. The requested operation cannot be performed.
?5F Bad transfer length.	The primary processor attempted to send EEPROM data to a secondary processor using an invalid length.
?60 EEPROM header or area has bad format.	All or part of the EEPROM contains inconsistent data and is probably corrupted. Reload the EEPROM from the TK tape.
?61 Illegal node number.	The specified node number is invalid.
?62 Unable to locate console tape device.	The console could not locate the I/O adapter that controls the TK tape.
?63 Operation only applies to secondary processors.	The command can only be directed at a secondary processor.
?64 Insufficient memory to buffer EEPROM image.	The SAVE, RESTORE, and PATCH EEPROM commands require working memory, but not enough was found.
?65 Validation of EEPROM tape image failed.	The image on tape is corrupted or is not the result of a SAVE EEPROM command. The image cannot be restored.
?66 Read of EEPROM image from tape failed.	The EEPROM image was not successfully read from tape.
?67 Validation of local EEPROM failed.	For a PATCH EEPROM operation, the EEPROM must first contain a valid image before it can be patched. For a RESTORE EEPROM operation, the image was written back to EEPROM but could not be read back successfully.
?68 EEPROM not changed.	The EEPROM contents were not changed.
?69 EEPROM changed successfully.	The EEPROM contents were successfully patched or restored.

Table A–2 (Cont.): Model 300 Standard Console Error Messages

Error Message	Meaning
?6A Error changing EEPROM.	An error occurred in writing to the EEPROM. The EEPROM contents may be corrupted.
?6B EEPROM saved to tape successfully.	The EEPROM contents were successfully written to the TK tape.
?6C EEPROM not saved to tape.	The EEPROM contents were not completely written to the TK tape.
?6D EEPROM Revision = *x.x/y.y*.	The EEPROM contents are at revision *x.x* with revision *y.y* patches.
?6E Major revision mismatch between tape image and EEPROM.	The TK tape contains an EEPROM image with a major revision different from that found in the EEPROM. The image cannot be restored.
?6F Tape image Revision = *x.x/y.y*.	The EEPROM image on the TK tape is at revision *x.x* with revision *y.y* patches.
?74	CONSOLE_LIMIT value too small for proper operation. Value ignored.
?75	Error writing to tape. Tape may be write-locked.
?83	See *Loading system software* below.
?84	See *Failure* below.
?85	See *Restarting system software* below.
?B0	See *Initializing system* below.
Failure.	The console failed in a restart or boot operation. Shows as ?84 in SET LANGUAGE INTERNATIONAL mode.
Initializing system.	The console is resetting the system in response to a BOOT command. Shows as ?B0 in SET LANGUAGE INTERNATIONAL mode.

Table A–2 (Cont.): Model 300 Standard Console Error Messages

Error Message	Meaning
Loading system software.	The console is attempting to load the operating system in response to a BOOT command, power-up, or restart failure. Shows as ?83 in SET LANGUAGE INTERNATIONAL mode.
Node: *n ?xx*	Error message *?xx* was generated on secondary processor *n* and was passed to the primary processor to be displayed.
Restarting system software.	The console is attempting to restart the in-memory copy of the operating system following a power-up or serious error. Shows as ?85 in SET LANGUAGE INTERNATIONAL mode.

Appendix B

Console Error Messages for Model 400

Table B–1 lists Model 400 messages that appear when the processor halts and the console gains control. Most messages are followed by:

- PC = xxxxxxxx — program counter = address at which the processor halted or the exception occurred
- PSL = xxxxxxxx — processor status longword = contents of the register
- –SP = xxxxxxxx — –SP is one of the following:
  ESP executive stack pointer
  ISP interrupt stack pointer
  KSP kernel stack pointer
  SSP supervisor stack pointer
  USP user stack pointer

Table B–2 lists other console error messages.

Table B–1: Model 400 Console Error Messages Indicating Halt

Error Message	Meaning
?02 External halt (CTRL/P, break, or external halt).	CTRL/P or STOP command.
?03 Power-up halt.	System has powered up, had a system reset, or an XMI node reset.
?04 Interrupt stack not valid during exception processing.	Interrupt stack pointer contained an invalid address.
?05 Machine check occurred during exception processing.	A machine check occurred while handling another error condition.
?06 Halt instruction executed in kernel mode.	The CPU executed a Halt instruction.
?07 SCB vector bits <1:0> = 11.	An interrupt or exception vector in the System Control Block contained an invalid address.

Table B–1 (Cont.): Model 400 Console Error Messages Indicating Halt

Error Message	Meaning
?08 SCB vector bits <1:0> = 10.	An interrupt or exception vector in the System Control Block contained an invalid address.
?0A CHMx executed while on interrupt stack.	A change-mode instruction was issued while executing on the interrupt stack.
?10 ACV/TNV occurred during machine check processing.	An access violation or translation-not-valid error occurred while handling another error condition.
?11 ACV/TNV occurred during kernel-stack-not-valid processing.	An access violation or translation-not-valid error occurred while handling another error condition.
?12 Machine check occurred during machine check processing.	A machine check occurred while processing a machine check.
?13 Machine check occurred during kernel-stack-not-valid processing.	A machine check occurred while handling another error condition.
?19 PSL <26:24>= 101 during interrupt or exception.	An exception or interrupt occurred while on the interrupt stack but not in kernel mode.
?1A PSL <26:24>= 110 during interrupt or exception.	An exception or interrupt occurred while on the interrupt stack but not in kernel mode.
?1B PSL <26:24>= 111 during interrupt or exception.	An exception or interrupt occurred while on the interrupt stack but not in kernel mode.
?1D PSL <26:24> = 101 during REI.	An REI instruction attempted to restore a PSL with an invalid combination of access mode and interrupt stack bits.
?1E PSL <26:24> = 110 during REI.	An REI instruction attempted to restore a PSL with an invalid combination of access mode and interrupt stack bits.
?1F PSL <26:24> = 111 during REI.	An REI instruction attempted to restore a PSL with an invalid combination of access mode and interrupt stack bits.

Table B–2: Model 400 Standard Console Error Messages

Error Message	Meaning
?20 Illegal memory reference.	An attempt was made to reference a virtual address (/V) that is either unmapped or is protected against access under the current PSL.
?21 Illegal command.	The command was not recognized, contained the wrong number of parameters, or contained unrecognized or inappropriate qualifiers.
?22 Illegal address.	The specified address was recognized as being invalid, for example, a general purpose register number greater than 15.
?23 Value is too large.	A parameter or qualifier value contained too many digits.
?24 Conflicting qualifiers.	A command specified recognized qualifiers that are illegal in combination.
?25 Checksum did not match.	The checksum calculated for a block of X command data did not match the checksum received.
?26 Halted.	The processor is currently halted.
?27 Item was not found.	The item requested in a FIND command could not be found.
?28 Timeout while waiting for characters.	The X command failed to receive a full block of data within the timeout period.
?29 Machine check accessing memory.	Either the specified address is not implemented by any hardware in the system, or an attempt was made to write a read-only address, for example, the address of the 33rd Mbyte of memory on a 32-Mbyte system.
?2A Unexpected machine check or interrupt.	A valid operation within the console caused a machine check or interrupt.
?2B Command is not implemented.	The command is not implemented by this console.
?2C Unexpected exception.	An attempt was made to examine either a nonexistent IPR or an unimplemented register in RSSC address range (20140000—20140800).

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning
?2D For Secondary Processor *n*.	This message is a preface to second message describing some error related to a secondary processor. This message indicates which secondary processor is involved.
?2E Specified node is not an I/O adapter.	The referenced node is incapable of performing I/O or did not pass its self-test.
?30 Write to Z command target has timed out.	The target node of the Z command is not responding.
?31 Z connection terminated by ^P.	A CTRL/P was typed on the keyboard to terminate a Z command.
?32 Your node is already part of a Z connection.	You cannot issue a Z command while executing a Z command.
?33 Z connection successfully started.	You have requested a Z connection to a valid node.
?34 Specified target already has a Z connection.	The target node was the target of a previous Z connection that was improperly terminated. Reset the system to clear this condition.
?36 Command too long.	The command length exceeds 80 characters.
?37 Explicit interleave list is bad. Configuring all arrays uninterleaved.	The list of memory arrays for explicit interleave includes no nodes that are actually memory arrays. All arrays found in the system are configured.
?39 Console patches are not usable.	The console patch area in EEPROM is corrupted or contains a patch revision that is incompatible with the console ROM.
?3B Error encountered during I/O operation.	An I/O adapter returned an error status while the console boot primitive was performing I/O.

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning
?3C Secondary processor not in console mode.	The primary processor console needed to communicate with a secondary processor, but the secondary processor was not in console mode. STOP the node or reset the system to clear this condition.
?3D Error initializing I/O device.	A console boot primitive needed to perform I/O, but could not initialize the I/O adapter.
?3E Timeout while sending message to secondary processor.	A secondary processor failed to respond to a message sent from the primary. The primary sends such messages to perform console functions on secondary processors.
?3F Microcode power-up self-test failed in REX520.	Model 400 CPU chip failed its microcoded self-test.
?40 Key switch must be at "Update" to update EEPROM.	A SET command was issued, but the key switch was not set to allow updates to the EEPROM.
?41 Specified node is not a bus adapter.	A command to access a VAXBI node specified an XMI node that was not a bus adapter.
?42 Invalid terminal speed.	The SET TERMINAL command specified an unsupported baud rate.
?43 Unable to initialize node.	The INITIALIZE command failed to reset the specified node.
?44 Processor is not enabled to BOOT or START.	As a result of a SET CPU/NOENABLE command, the processor is disabled from leaving console mode.
?45 Unable to stop node.	The STOP command failed to halt the specified node.
?46 Memory interleave set is inconsistent: *n n ...*	The listed nodes do not form a valid memory interleave set. One or more of the nodes might not be a memory array or might be of a different size, or the set could contain an invalid number of members. Each listed array that is a valid memory will be configured uninterleaved.

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning
?47 Insufficient working memory for normal operation.	Less than 256 Kbytes per processor of working memory were found. There is insufficient memory for the console to function normally or for the operating system to boot.
?48 Uncorrectable memory errors—long memory test must be performed.	A Model 400 memory array contains an unrecoverable error. The console must perform a slow test to locate all the failing locations.
?49 Memory cannot be initialized.	The specified operation was attempted and prevented.
?4A Memories not interleaved due to uncorrectable errors:	The listed arrays would normally have been interleaved (by default or explicit request). Because one or more of them contained unrecoverable errors, this interleave set will not be constructed.
?4B Internal logic error in console.	The console encountered a theoretically impossible condition.
?4C Invalid node for Z command.	The target of a Z command must be a CPU or an I/O adapter and must not be the primary processor.
?4D Invalid node for new primary.	The SET CPU command failed when attempting to make the specified node the primary processor.
?4E Specified node is not a processor.	The specified node is not a processor, as required by the command.
?4F System serial number has not been initialized.	No CPU in the system contains a valid system serial number.
?50 System serial number not initialized on primary processor.	The primary processor has an uninitialized system serial number. All other processors in the system contain a valid serial number.
?51 Secondary processor returned bad response message.	A secondary processor returned an unintelligible response to a request made by the console on the primary processor.

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning
?52 ROM revision mismatch. Secondary processor has revision *x.xx*.	The revision of console ROM of a secondary processor does not match that of the primary.
?53 EEPROM header is corrupted.	The EEPROM header has been corrupted. The EEPROM must be restored from the TK tape drive.
?54 EEPROM revision mismatch. Secondary processor has revision *x.xx/y.yy*.	A secondary processor has a different revision of EEPROM or has a different set of EEPROM patches installed.
?55 Failed to locate EEPROM area.	The EEPROM did not contain a set of data required by the console. The EEPROM may be corrupted.
?56 Console parameters on secondary processor do not match primary.	The console parameters are not the same for all processors .
?57 EEPROM area checksum error.	A portion of the EEPROM is corrupted. It may be necessary to reload the EEPROM from the TK tape drive.
?58 Saved boot specifications on secondary processor do not match primary.	The saved boot specifications are not the same for all processors.
?59 Invalid unit number.	A BOOT or SET BOOT command specified a unit number that is not a valid hexadecimal number between 0 and FF.
?5A System serial number mismatch. Secondary processor has xxxxxxxx.	The indicated serial number of a secondary processor does not match that of the primary.
?5B Unknown type of boot device.	The console program does not have a boot primitive to support the specified type of device or the device could not be accessed to determine its type.
?5C No HELP is available.	The HELP command is not supported when the console language is set to International.
?5D No such boot spec found.	The specified boot specification was not found in the EEPROM.
?5E Saved boot spec table full.	The maximum number of saved boot specifications has already been stored.

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning
?5F EEPROM header version mismatch.	Processors have different versions of EEPROMs.
?61 EEPROM header or area has bad format.	All or part of the EEPROM contains inconsistent data and is probably corrupted. Reload the EEPROM from the TK tape.
?62 Illegal node number.	The specified node number is invalid.
?63 Unable to locate console tape device.	The console could not locate the I/O adapter that controls the TK tape.
?64 Operation only applies to secondary processors.	The command can only be directed at a secondary processor.
?65 Operation not allowed from secondary processor.	A secondary processor cannot perform this operation.
?66 Validation of EEPROM tape image failed.	The image on tape is corrupted or is not the result of a SAVE EEPROM command. The image cannot be restored.
?67 Read of EEPROM image from tape failed.	The EEPROM image was not successfully read from tape.
?68 Validation of local EEPROM failed.	For a PATCH EEPROM operation, the EEPROM must first contain a valid image before it can be patched. For a RESTORE EEPROM operation, the image was written back to EEPROM but could not be read back successfully.
?69 EEPROM not changed.	The EEPROM contents were not changed.
?6A EEPROM changed successfully.	The EEPROM contents were successfully patched or restored.
?6B Error changing EEPROM.	An error occurred in writing to the EEPROM. The EEPROM contents may be corrupted.
?6C EEPROM saved to tape successfully.	The EEPROM contents were successfully written to the TK tape.
?6D EEPROM not saved to tape.	The EEPROM contents were not completely written to the TK tape.

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning
?6E EEPROM Revision = *x.xx/y.yy*.	The EEPROM contents are at revision *x.xx* with revision *y.yy* patches.
?6F Major revision mismatch between tape image and EEPROM.	The major revision of tape and EEPROM do not match. The requested operation cannot be performed.
?70 Tape image Revision = *x.xx/y.yy*.	The EEPROM image on the TK tape is at revision *x.xx* with revision *y.yy* patches.
?73 System serial number updated.	The EEPROM has been updated with the correct system serial number.
?74 System serial number not updated.	The EEPROM has not been changed.
?75 /CONSOLE_LIMIT value too small for proper operation. Value ignored.	No change has been made.
?76 Error writing to tape. Tape may be write-locked.	Tape has not been written. Check to see if tape is write-locked.
?77 CCA not accessible or corrupted.	Attempt to find the console communications area (CCA) failed. The console then builds a local CCA, which does not allow for interprocessor communication.
?78 Vector module configuration error at node *n*	The console detected a vector module configuration error. Problem can be that the vector node number is not one greater than the scalar CPU or that the module to the left of a vector processor is not a memory module.
?79 Vector synchronization error.	The console could not synchronize with the vector processor on a console entry. The Busy bit in the Vector Processor Status Register remained set after a timeout, or a vector processor error occurred.
?7A No vector module associated with CPU at specified node.	No vector module is in the slot to the left of the specified CPU, or the VIB cable either is not attached or is bad.
?7B An error occurred while accessing the vector module.	Attempt to access VCR, VLR, or VMR registers failed.

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning
?7C I/O adapter configuration error at node *n*	The I/O adapter at node *n* is configured improperly.
?7D Vector module is disabled—check KA64A revision at XMI node *n*	The vector module is attached to a KA64A module that is not at the revision level required.
?83 Loading system software.[1]	The console is attempting to load the operating system in response to a BOOT command, power-up, or restart failure.
?84 Failure.[1]	An operation did not complete successfully. Should be issued with another message to clarify failure.
?85 Restarting system software.[1]	The console is attempting to restart the in-memory copy of the operating system following a power-up or serious error.
?A0 Initializing system.[1]	The console is resetting the system in response to a BOOT command.
?A6 Console halting after unexpected machine check or exception.[1]	The console executed a Halt instruction to reset the console state after processing an unexpected machine check.
?A7 RCSR <WD> is set. Local CCA must be built.	When the <WD> bit is set, writes to memory are disabled. The Model 400 processor must then build a CCA in local memory. Main memory cannot be written to or accessed with interlocked instructions.
?A8 Bootstrap failed due to previous error.[1]	The previous attempt to bootstrap the system failed.
?A9 Restart failed due to previous error.[1]	The previous attempt to restart the system failed.
Node *n: ?xx*	Error message *?xx* was generated on secondary processor *n* and was passed to the primary processor to be displayed.

[1]No numbered prefix appears with these messages in English language mode. These numbers are used for these messages in International mode.

Table B–2 (Cont.): Model 400 Standard Console Error Messages

Error Message	Meaning

Appendix C

EEPROM Mismatches and the UPDATE Command

Table C–1 shows the results of trying to update processor modules from primary processors that have different ROM revisions from the secondary being updated.

Table C–1: UPDATE Results on EEPROM Mismatches

Primary ROM Revision	Secondary ROM Revision	Results after UPDATE issued
<REFERENCE>(xrp) ROM Revisions		
1.0	2.0	Processor EEPROM corrupted, but still powers up.
1.0	3.0	Processor EEPROM corrupted, but still powers up.
2.0	1.0	Use EVUCA for updates to secondary EEPROM. Update not allowed.
2.0	3.0	Use EVUCA for updates to secondary EEPROM. Update not allowed.
3.0	1.0	Use EVUCA for updates to secondary EEPROM. Update not allowed.
3.0	2.0	Use EVUCA for updates to secondary EEPROM. Update not allowed.
<REFERENCE>(xyp) ROM Revisions		
4.1	6.0	Processor updated. Power-up tests broken. Processor will not be recognized.
6.0	4.1	Use EVUCA for updates to secondary EEPROM. Update not allowed.

Table C–1 (Cont.): UPDATE Results on EEPROM Mismatches

Primary ROM Revision	Secondary ROM Revision	Results after UPDATE issued
KA62A ROM Revisions		
3.0	5.0	Processor updated. Power-up tests broken. Processor will not be recognized.
5.0	3.0	Use EVUCA for updates to secondary EEPROM. Update not allowed.

Appendix D

Control Flags for Booting

With the console BOOT command, you can control various phases of booting by setting bits in General Purpose Register R5:

```
BOOT /R5:n
```

where *n* is in hexadecimal notation. For example, to set bit 4 in R5 when booting, you would enter:

```
BOOT /R5:10
```

The R5 bit functions are defined by VMB and by the operating system. The value –1 in R5 is reserved for Digital.

Table D–1: R5 Bit Functions for VMS

Bit	Function
0	Conversational boot. The secondary bootstrap program, SYSBOOT, prompts you for system parameters at the console terminal.
1	Debug. If this flag bit is set, the operating system maps the code for the XDELTA debugger into the system page tables of the running operating system.
2	Initial breakpoint. If this flag bit is set, VMS executes a breakpoint (BPT) instruction early in the bootstrapping process.
3	Secondary boot from boot block. The secondary boot is a single 512-byte block whose logical block number is specified in General Purpose Register R4.
4	Boots the VAX Diagnostic Supervisor. The secondary loader is an image called DIAGBOOT.EXE.
5	Boot breakpoint. This stops the primary and secondary loaders with a breakpoint (BPT) instruction before testing memory.

Table D–1 (Cont.): R5 Bit Functions for VMS

Bit	Function
6	Image header. The transfer address of the secondary loader image comes from the image header for that file. If this flag is not set, control shifts to the first byte of the secondary loader.
8	File name. VMB prompts for the name of a secondary loader.
9	Halt before transfer. VMB executes a HALT instruction before transferring control to the secondary loader.
13	No effect, since console program tests memory.
15	Reserved for the VAX Diagnostic Supervisor.
16	Do not discard CRD pages.
31:28	Specifies the top-level directory number for system disks.

Table D–2: R5 Bit Functions for ULTRIX

Bit	Function
0	Forces ULTRIXBOOT to prompt the user for an image name (the default is VMUNIX).
1	Boots the ULTRIX kernel image in single-user mode.
3	Must be set, and R4 must be zero.
16	Must be set.

Glossary

Adapter

A node that interfaces other buses, communication lines, or peripheral devices to the <REFERENCE>(XMI) bus or the VAXBI bus.

Address space

The 1 terabyte of physical address space that the XMI bus is capable of supporting; currently the XMI bus supports 1 gigabyte of physical memory.

Asymmetric multiprocessing

A multiprocessing configuration in which the processors are not equal in their ability to execute operating system code. In general, a single processor is designated as the primary, or master, processor; other processors are the slaves. The slave processors are limited to performing certain tasks, whereas the master processor can perform all system tasks. Contrast with *Symmetric multiprocessing*.

Bandwidth

The data transfer rate measured in information units transferred per unit of time (for example, Mbytes per second).

Boot device

Contains the bootblock and typically also contains the virtual memory boot program (VMB). A VAX 6000 series system can be booted from one of four boot devices: the console load device, a local system disk, a disk connected to the system through a CI adapter, or a disk connected to the system through the Ethernet.

Boot primitives

Small programs stored in ROM on each processor with the console program. Boot primitives read the bootblock from boot devices. There is a boot primitive for each type of boot device.

Boot processor

The CPU module that boots the operating system and communicates with the console.

Bootblock

Block zero on the system disk; it contains the block number where the virtual memory boot (VMB) program is located on the system disk and contains a program that, with the boot primitive, reads VMB from the system load device into memory.

CIBCA

VAXBI CI port interface; connects a system to a Star Coupler.

CIXCD

XMI CI port interface; connects a system to a Star Coupler.

Cold start

An attempt by the primary processor to boot a new copy of the operating system.

Compact disk server

Ethernet-based CD server; provides access to CDROMs for software installation, diagnostics, and on-line documentation.

Console communications area (CCA)

Segment of system main memory reserved by the console program.

Console mode

A mode of operation allowing a console terminal operator to communicate with nodes on the XMI bus.

DEBNI

VAXBI adapter; Ethernet port interface.

DEMNA

XMI adapter; Ethernet port interface.

DHB32

VAXBI adapter communication device; supports up to 16 terminals.

DMB32

VAXBI adapter interface for 8-channel asynchronous communications for terminals, one synchronous channel, and a parallel port for a line printer.

DRB32

VAXBI adapter; parallel port.

DSB32

VAXBI adapter communication device; provides two synchronous lines.

DWMBB

The XMI-to-VAXBI adapter; a 2-module adapter that allows data transfer from the XMI to the VAXBI; DWMBB/A is the module in the XMI card cage, and DWMBB/B is the VAXBI module. Every VAXBI on a VAX 6000 series system must have a <REFERENCE>(XBI) adapter.

Ethernet-based compact disk server

The RRD40 compact disk drive, a console load device, functions as a server on the Ethernet.

FV64A

Vector processor; works in a scalar/vector processor pair.

Interleaving memory

See Memory interleaving.

KDB50

VAXBI adapter for RAxx disks; enables connection to disk drives.

KDM70

XMI adapter for RAxx disks; enables connection to disk drives.

Memory interleaving

Method to optimize memory access time; the VAX 6000 series console program automatically interleaves the memories in the system unless the SET MEMORY command is used to set a specific interleave or no interleave (which would result in serial access to each memory module). Interleaving causes a number of memories to operate in parallel.

Memory node

Also called the <REFERENCE>(XMA2). Memory is a global resource equally accessible by all processors on the <REFERENCE>(XMI). See also *<REFERENCE>(XMA2)*.

Module

A single <REFERENCE>(XMI) or VAXBI card that is housed in a single slot in its respective card cage. XMI modules (11.02" x 9.18") are larger than VAXBI modules (8.0" x 9.18").

<REFERENCE>(XMA2)

XMI memory array; a memory subsystem of the XMI. Memory is a global resource equally accessible by all processors on the <REFERENCE>(XMI). A memory module can have 32, 64, or 128 Mbytes of memory, consisting of MOS 1–Mbit or MOS 4–Mbit dynamic RAMs, ECC logic, and control logic.

Node

An <REFERENCE>(XMI) node is a single module that occupies one of the 14 logical and physical slots on the <REFERENCE>(XMI) bus. A VAXBI node consists of one or more VAXBI modules that form a single functional unit.

Node ID

A hexadecimal number that identifies the node location. On the <REFERENCE>(XMI) bus, the node ID is the same as the physical location. On the VAXBI, the source of the node ID is an ID plug attached to the backplane.

Pended bus

A bus protocol in which the transfer of command/address and the transfer of data are separate operations. The <REFERENCE>(XMI) bus is a pended bus.

Primary processor

See *Boot processor*.

Processor node

A VAX processor that contains a central processor unit (CPU), executes instructions, and manipulates data contained in memory.

RBD

ROM-based diagnostics.

RBV20/RBV64

VAXBI adapter for write-once-read-many (WORM) optical disk drive. The RBV20 and RBV64 controllers use the KLESI–B adapter.

Scalar/vector processor pair

The FV64A vector processor functions as a coprocessor with a host scalar processor. The scalar/vector processor pair appear as one processor to an executing program.

Secured terminal

Console terminal in program mode while the machine is processing.

Shadow set

Two disks functioning as one disk, each shadowing the information contained on the other, controlled by an HSC controller under the VMS operating system.

Symmetric multiprocessing

A multiprocessing system configuration in which all processors have equal access to operating system code residing in shared memory and can perform all, or almost all, system tasks.

System root

In a BOOT commmand, the argument to the /R5 qualifier.

TBK70

VAXBI adapter connecting the TK tape drive to the system.

TU81E

VAXBI adapter for a local (nonclustered) tape subsystem. The TU81E controller uses the KLESI–B adapter.

VAX Diagnostic Supervisor (VAX/DS)

Software that loads and runs diagnostic and utility programs.

VAXBI bus

The 32-bit bus used for I/O.

VAXBI Corner

The portion of a VAXBI module that connects to the backplane and provides an electrically identical interface for every VAXBI node.

VMB

The virtual memory boot program (VMB.EXE) that boots the operating system. VMB is the primary bootstrap program and is stored on the boot device. The goal of booting is to read VMB from the boot device and load the operating system.

XBI

Lines in the self-test display that show the status of <REFERENCE>(XBI_plus) adapters and of VAXBI nodes. See also *<REFERENCE>(xbi_plus)*.

XMI

The 64-bit, high-speed system bus.

<REFERENCE>(XMI) Corner

The portion of an <REFERENCE>(XMI) module that connects to the backplane and provides an electrically identical interface for every XMI node.

Index

S

T