EK-KA86D-UG-002

VAX 8600/8650 System Diagnostics User's Guide

For Internal Use Only

Prepared by Educational Services
of
Digital Equipment Corporation

1st Edition, October 1985
2nd Edition, March 1986

The material in this manual is for informational purposes and is subject to change without notice.

Digital Equipment Corporation assumes no reponsibility for any errors which may appear in this manual.

Printed in U.S.A.

This book was produced on a Digital Word Processing System. Book production was done by Educational Services Development and Publishing in Bedford, MA.

The following are trademarks of Digital Equipment Corporation:

digital™	PDP	RT
DEC	P/OS	UNIBUS
DECmate	Professional	VAX
DECUS	Q-Bus	VMS
DECwriter	Rainbow	VT
DIBOL	RSTS	Work Processor
MASSBUS	RSX	

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

FIGURES (Cont.)

FIGURES (Cont.)

TABLES

TABLES (Cont.)

TABLES (Cont.)

PREFACE

This manual describes the VAX 8600/8650 diagnostic software system. It discusses how to use the system's software maintenance tools to isolate hardware failures to a field replaceable unit (FRU). The reader must have had experience using VAX macrodiagnostics and the Diagnostic Supervisor running on other members of the VAX family. This manual emphasizes VAX 8600/8650-specific diagnostic information. For details on VAX generic diagnostics, the reader must refer to the VAX Diagnostic Supervisor User's Guide (EK-VXDSU-UG).

This manual is divided into seven chapters and several appendices. An outline of the manual follows.

Chapter 1 -- VAX 8600/8650 Diagnostic System -- Chapter 1 provides a functional description of the Diagnostic System that includes a discussion of field maintenance strategies. It lists and explains all the hardware and software components within the diagnostic system, and how they tie together to provide a set of troubleshooting tools for isolating hardware faults during corrective maintenance.

Chapter 2 -- PROM Tests -- Chapter 2 describes the operation of the PROM self-tests available on the system console module, and the use of these tests to isolate faults in the console subsystem hardcore logic.

Chapter 3 -- EDOBA Console Diagnostic -- Chapter 3 describes the purpose and operation of the T-11 based Console Diagnostic Program, EDOBA, and the use of this program to isolate faults on the console module and its immediate logic interfaces.

Chapter 4 -- Microhardcore (MHC) Tests -- Chapter 4 describes the purpose and operation of the MHC tests and the use of MHC to isolate faults in the hardcore logic.

Chapter 5 -- Microdiagnostics -- Chapter 5 describes the purpose and operation of all the microdiagnostic programs for the system, and the use of these programs to isolate logic faults in the CPU, internal memory, and ABus interfaces.

Chapter 6 -- Macrodiagnostics -- Chapter 6 describes the purpose and operation of the VAX 8600 macrodiagnostic programs and the use of these programs to isolate logic faults in the system.

Chapter 7 -- Console Command Sets -- Chapter 7 describes the creative use of the Diagnostic Console command sets to perform manual testing and fault analysis beyond what the diagnostics normally provide.

Appendices -- The Appendices provide easy reference to information and descriptions that support the theory and procedures described in Chapters 1 through 7. For example, Appendix A contains a summary of all the available console commands with illustrative examples of how to use the on-line HELP facility. Appendix D contains a summary description of the content and organization of the files on the RL02 disk pack. This organization permits the user to locate rapidly the most commonly used information without wading through a welter of descriptive text.

WARNING

Some of the command descriptions and sample program responses shown may differ depending upon software revision levels. In the early stages of a new product, functional enhancements and "bug" fixes mean unavoidable changes to software and hardware.

CHAPTER 1
VAX 8600/8650 DIAGNOSTIC SYSTEM OVERVIEW

1.1 INTRODUCTION

This first chapter provides an overview of the VAX 8600/8650 diagnostic system. It includes general discussions of maintenance strategy, basic software and hardware components, and modes of console operation in preparation for the detailed descriptions in Chapters 2 through 7.

> **NOTE**
>
> Unless otherwise specified, the information in this user guide applies to both the VAX 8600 and VAX 8650 systems.
>
> In addition, all illustrations of macro and microdiagnostic displays, listings, and reports are provided as representative examples, and may not reflect the current revision.

1.2 RELATED DOCUMENTATION

This manual describes how to use the console front-end software to test and troubleshoot the system. For detailed descriptions of the hardware itself, the reader may need to refer to one or more of the documents in Table 1-1.

1.3 DIAGNOSTIC STRATEGY OVERVIEW

The VAX 8600/8650 diagnostic testing and maintenance strategies are aimed at isolating hardware faults to a FRU, module, or subassembly as quickly and reliably as possible. To implement this strategy, the system hardware design includes redundant hardware logic circuits (parity and ECC) to detect, latch, and report hardware errors at strategic test points throughout the system. Special Error Handling Microcode (EHM), resident in the EBox control store, retrieves and formats the error information detected by the hardware and passes it to the operating system for "logging" in a system event file on the disk.

Table 1-1 Related Documentation

Title	Document Number
Technical Descriptions	
VAX 8600/8650 Console Technical Description	EK-KA86C-TD
VAX 8600/8650 EBox Technical Description	EK-KA86E-TD
VAX 8600/8650 System Power Technical Description	EK-KA86P-TD
VAX 8600/8650 FBox Technical Description	EK-FP86X-TD
VAX 8600/8650 IBox Technical Description	EK-KA86I-TD
VAX 8600/8650 MBox/Memory Technical Description	EK-KA86M-TD
VAX 8600/8650 SBIA Technical Description	EK-DB86X-TD
VAX 8600/8650 System Clocks Technical Description	EK-KA86K-TD
VAX 8600/8650 EMM Technical Description	EK-KA86V-TD
VAX 8600/8650 System Description and Processor Overview	EK-KA86S-TD
User's Guides	
VAX 8600/8650 System Hardware User's Guide	EK-8600H-UG
VAX 8600/8650 System Maintenance Guide*	EK-86XV1-MG
VAX 8600/8650 System Installation Manual	EK-8600I-IN
VAX 8600/8650 System Fault Isolation Manual*	EK-8600S-MM

* For Internal Use Only

In addition to the parity/ECC check circuits, the system contains a diagnostic console subsystem, driven by a T-11 microprocessor, that serves as the operator's console and diagnostic load device. A Serial Diagnostic Bus (SDB) connects the console to the VAX CPU to provide visibility to over 2000 key logic signals for diagnostic testing and fault isolation. The SDB is also used to load all the CPU control RAMs during system start-up and initialization. A second bus, called the CBus, connects the console subsystem to the EBox to provide the communications link between the console front end and the VAX CPU. Both the SDB and the CBus are exploited by the diagnostics during system test and fault isolation.

Finally, a complete set of console software programs, resident on the RL02 disk device, provide a set of software tools that do the following.

- Start and stop the system
- Control and monitor system operation
- Load and run diagnostics

The diagnostic software provides the facilities for both on-line and standalone testing. Which facility the service engineer uses depends on the nature of the hardware fault and the type of maintenance being performed.

1.3.1 On-Line Error Analysis

Intermittent hardware errors that the system can recover from are logged in a system event file on the system disk. This class of error may or may not cause an occasional system crash. Intermittent errors generally elude detection by standalone diagnostics. Several types of diagnostic tools are available for on-line testing of the system.

- System Program for Error Analysis and Reporting (SPEAR) -- A user-mode program that retrieves and analyzes the errors logged in the system event file.
- System Dump Analyzer (SDA) -- A VMS utility program that analyzes the core image of the operating system after a crash and successful restart.
- User Environmental Test Package (UETP) -- A field configurable set of program modules that can be run on-line to exercise the system interactively and to report failures within specific subsystems.
- Macrodiagnostics -- Functional macrodiagnostic programs that run on-line under control of the VMS operating system.

This manual focuses on standalone test procedures. For information on using SPEAR, SDA, and UETP, refer to the individual user guides for these programs. The VAX 8600/8650 System Fault Isolation Manual (EK-8600S-MM) also includes discussions of on-line testing. Chapter 6 includes some discussion of on-line macrodiagnostics for VAX 8600/8650 systems.

1.3.2 Standalone Testing

A complete set of diagnostic programs is available for standalone testing of the system. These programs range in complexity from simple T-11 PROM-based self-tests that validate the console module hardcore logic on power-up, to VAX internal memory-based macro-diagnostics that test the I/O subsystems. All of these programs, with the exception of the PROM tests, are loaded from the RL02 disk. Figure 1-1 summarizes the types of diagnostics that are available for testing and troubleshooting the system.

The diagnostic program set was designed in a building block format. As the user moves higher into each subsequent program level, it should be assumed that the hardware at the previous (lower) level is working. Moving from bottom to top in Figure 1-1, the size and complexity of the hardware/software test environment increases. The objective is to detect the fault at the simplest level, thus making it easier to identify the source of the problem.

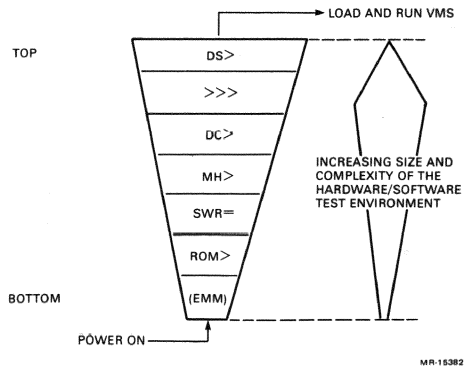


Figure 1-1 Standalone Testing -- Diagnostic Hierarchy

Each level includes one or more of these test programs.

- EMM -- During power-up, the 8085 microprocessor on the Environmental Monitoring Module (EMM) executes a set of PROM-based self-tests to check out the EMM. Refer to VAX 8600/8650 EMM Technical Description (EK-KA86V-TD) for details on the EMM tests.

- ROM> -- During power-up, the T-11 microprocessor executes the PROM self-tests to validate the console module hardcore logic. Chapter 2 describes the operation of the PROM tests.

- EDOBA -- A T-11 based console module diagnostic is invoked by the T command in the PROM tests. Chapter 3 describes the operation and use of EDOBA.

- MH> -- The T-11 based MHC diagnostic tests the CPU hardcore logic. Chapter 4 describes the operation and use of MHC.

- DC> -- The T-11 based Diagnostic Control Program (DCP) controls loading, running, and monitoring a complete set of CPU microdiagnostics. Chapter 5 describes the operation and use of the microdiagnostics.

- >>> -- The T-11 based Macro Control Program (MCP) controls loading, running, and monitoring VAX internal memory-based macrodiagnostics. These macrodiagnostics test the CPU hardcore instruction set used by the VAX diagnostic supervisor. Chapter 6 describes the operation and use of macrodiagnostics.

- DS> -- The VAX internal memory-based diagnostic supervisor controls loading, running, and monitoring of most of the VAX macrodiagnostics. Chapter 6 describes the operation and use of the supervisor and the VAX macrodiagnostics.

The exact sequencing of the diagnostics will depend on the type of maintenance being performed. During installation and preventive maintenance, a straight-line, bottom-up approach will most likely be used. In the case of demand corrective maintenance, the approach will probably differ depending upon the initial customer complaint and the individual troubleshooting style of the service engineer. For example, it would probably be fruitless to run microdiagnostics if the customer was complaining about occasional disk errors from drive 3. Each of the following chapters will include more detailed discussions of recommended test sequences.

1.4 DIAGNOSTIC FEATURES

To improve the maintainabilty of the VAX 8600/8650 computer systems, extensive hardware and software features were added so that the user would have a comprehensive set of maintenance tools for detecting and isolating hardware faults. For most failures, the user should be able to use these tools to localize the source of the problem and to identify the FRU that needs to be replaced.

1.4.1 Hardware Features

1. A console front-end system includes the following.

   - A T-11 microprocessor
   - An 8 Kbyte (KB) PROM
   - A 256 KB dynamic RAM
   - Three serial ASCII ports (local, remote, and EMM)
   - An RL02 disk subsystem
   - A Time-Of-Year clock (TOY)

   Refer to the VAX 8600/8650 Console Technical Description (EK-KA86C-TD) for a more detailed discussion of the console hardware.

2. The SDB provides the console with the ability to control logic facilities within the CPU and to record the state of thousands of key logic signals within the CPU modules.

3. An 8-bit parallel bus (CBus), connected to a dual-port register file, provides a communications link between the CPU and the console software.

4. Extensive hardware parity generation and checking circuits at strategically located test points within the CPU modules detect and report hardware errors.

1.4.2 Software Features

1. A complete set of console software programs uses the power of the PDP-11 instruction set to control the hardware features described above.

2. A set of console commands organized into logical sets permit the operator or service engineer to control the CPU during normal operation or diagnose the system when it fails.

3. A complete set of diagnostic programs and support files resident on the RL02 disk pack are used to test and troubleshoot the system.

The presence of a physically separate and independent subsystem overcomes the problems associated with systems where the diagnostics must be loaded and run using hardware that needs to be tested.

1.5 CONSOLE SOFTWARE OVERVIEW

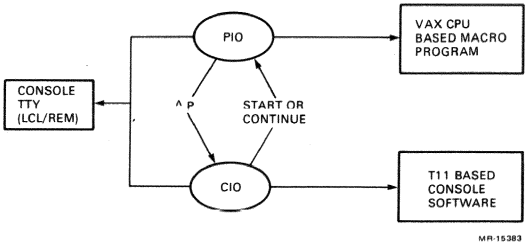
The console software is essentially a T-11 program, EDOAA.SAV, that is booted from the RL02 disk during system power-up and start-up. A comprehensive set of program overlays provide the means for controlling the console subsystem and the VAX CPU.

1.5.1 Console Operating Modes

From the user's perspective, there are two distinct modes of operation that the console software may operate in. These are summarized in Figure 1-2.

- Console I/O Mode (CIO) -- In this mode, the VAX CPU is normally halted and the console software accepts and processes commands from either the local or remote ports.

- Program I/O Mode (PIO) -- In this mode, the console software is "slave" to the macro program running in the VAX CPU, and does not accept or process commands from either the local or remote ports. Any user input is passed to the VAX CPU.



NOTE: CONTROL-P DOES NOT SWITCH MODES IF THE TERMINAL SELECT SWITCH IS IN ONE OF THE TWO DISABLE POSITIONS.

Figure 1-2 Console Software -- Modes of Operation

1.5.1.1 CIO Mode -- Three separate CIO mode contexts are provided to facilitate the handling of function-specific console tasks.

1. The MACRO context (default on power-up) is used for initializing the CPU and running VAX macrocode.

2. The MICROHARDCORE context is used to test the CPU hardcore logic before running microdiagnostics.

3. The DIAGNOSTIC context is used to run microdiagnostic test programs and to isolate machine faults.

Switching between contexts affects the state of the machine significantly because local initialization is performed by each context on entry.

Once in CIO mode, the user may select one of three contexts as summarized in Figure 1-3. Each context provides access to a unique set of commands designed to support specific console functions. A general set of commands is available in all three contexts. When in either the macro or diagnostic contexts, the user may use the DEBUG command to access commands in the HEX program overlay. Refer to Appendix A for a complete summary of all the command sets.

A PROM command is available from all three contexts that permits the user to access the PROM-based code.

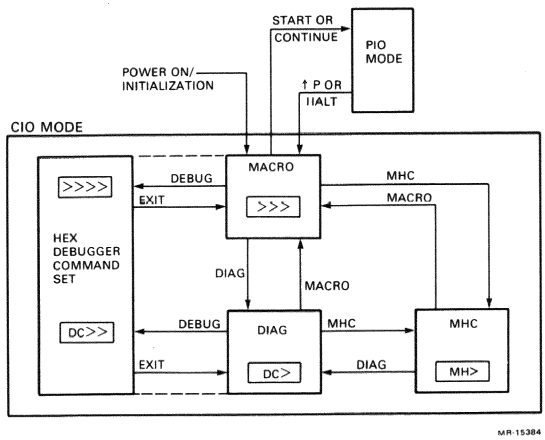


Figure 1-3 CIO Mode Contexts

1.5.1.1.1 Macro Context -- The macro context is the default on power-up initialization; it is used for normal control of the VAX CPU when loading and running macrodiagnostics. The macro context may also be entered from either the diagnostic or microhardcore contexts with the MACRO command. The command will cause the console software to invoke a reinitialization of the macro context. When in macro context, the VAX CPU control RAMs are loaded with system-level microcode to support running the operating system and macrodiagnostics. During the initialization process, Console Support Microcode (CSM), resident in the EBox control store, provides the communication link between the VAX CPU and the console software via CBus protocol. Appendix B provides a description of the CSM.

When in macro context, the console software displays the >>> prompt to indicate it is ready to accept and process commands. The following command sets are available in the macro context.

- General command set
- Macro command set
- HEX debugger command set

Once in macro context, the user may use either the START or CONTINUE commands to enter PIO mode assuming, of course, that some type of macroprogram has been loaded into VAX internal memory.

1.5.1.1.2 Microhardcore Context -- The microhardcore (MHC) context may be entered from either the macro or diagnostic contexts by typing the MHC command. No VAX CPU initialization is required on switching; the console software simply transfers control to a set of MHC program overlays and responds with the MH> prompt to indicate it is ready to accept commands. When in the MHC context, the user has access to the following command sets.

- MHC command set
- General command set

After running the MHC tests, the user may switch to either the macro or diagnostic contexts via the MACRO or DIAG commands, respectfully.

1.5.1.1.3 Diagnostic Context -- The diagnostic context may be entered from either the macro or microhardcore contexts when the user types the DIAG command. When the context is entered, the console software initializes the VAX CPU for running microdiagnostics. It switches control to the DCP overlay which responds with the DC> prompt, indicating it is ready to accept commands. Diagnostic Support Microcode (DSM) is loaded into the EBox control store to provide the communications link between the console software and the microdiagnostics. Appendix C contains a description of the DSM.

Once in diagnostic context, the user can load and run microdiagnostics to test the CPU, internal memory, and ABus interfaces. While in this context, the user has access to the following command sets.

- General command set
- Diagnostic command set
- HEX debugger command set

From here, the user may switch to either the macro or microhardcore contexts via the MACRO or MHC commands, respectively.

1.5.1.2 PIO Mode -- When operating in PIO mode, the console software services requests from the VAX CPU and provides special monitoring functions to the total system. PIO mode can only be entered from the MACRO context by typing either the START or CONTINUE command. PIO mode is entered automatically if an automatic warm-start or cold-start attempt is made either at the completion of a console reboot, or after console initialization.

PIO mode is exited upon receipt of a <CTRL/P> from either the local or remote port if the Terminal Select Switch on the SCP is not in one of the two DISABLE positions. If disabled, the <CTRL/P> simply passes to the VAX CPU with no change in console mode. It may also be exited if the VAX CPU halts (such as for a Keep Alive Failure [KAF]). PIO mode always exits to CIO mode in the macro context.

When PIO mode is exited via a <CTRL/P>, the VAX CPU continues to run. The user must type a HALT command to halt the VAX CPU. Most of the console commands that affect the state of the CPU are not allowed unless the CPU is halted.

While running in PIO mode, the console software does not process any operator commands from either the local or remote ports. Any data received from either port is simply passed to the macroprogram running in the VAX CPU as ASCII data. When in PIO mode, the console software services the following functions.

- Independent character transfers between the CPU and the local or remote ports
- Passing requests to the EMM port from the CPU, and passing responses and unsolicited warning messages from the EMM port to the VAX CPU
- Servicing "logical console" requests from the CPU
- Handling CPU control store parity error correction where possible
- Handling detection and recovery of CPU ERROR HALTS and CPU KEEP ALIVE FAILURES
- Handling CI reboot operations (ABus DEAD Interrupt)
- Handling detection of and recovery from power failures
- Handling CPU requests to access the RL02 disk
- Providing TOY information to the CPU with 10 ms resolution
- Handling updating of the LED indicators on the SCP

1.6 BOTTOM-UP TEST PROCEDURE

This section summarizes a straight-line procedure for testing a system from the time the power is turned on to the point where the user can log into VMS. First, make sure the diagnostic pack is mounted in the RL02 and the system disk contains the VMS operating system. Perform the following steps to invoke the PROM command parser and get to the ROM> prompt.

1. Install the RL02 disk pack.
2. Place the Terminal Control Switch in the LOCAL position to turn on the power.
3. Type any key on the console terminal when the "bell" sounds.

At this point, execute the command procedure summarized in Table 1-2. If any of the tests result in errors or abnormal responses, refer to the appropriate chapter for a detailed description of that step in the procedure.

Table 1-2 VAX 8600/8650 CPU Bottom-Up Test Procedure

Prompt	Command	Action	Time (minutes)
ROM>	T	Load and run EDOBA, the console diagnostic (2 passes).	3
ROM>	B	Boot the console software to load and start EDOAA.SAV. Press <CTRL/C> when the message "Initializing CPU" is displayed.	3
>>>	MHC	Switch to microhardcore context.	.2
MH>	START	Run all the microhardcore tests.	20
MH>	DIAGNOSE	Switch to diagnostic context.	.25
DC>>	@MICROS	Run all the VAX CPU micro-diagnostic programs.	30
DC>>	MACRO	Switch to macro context.	5
>>>	@EDKAA	Load and start EDKAA.EXE, the VAX macrohardcore diagnostic.	*
>>>	@EDSAA	Load and start the VAX 8600/8650 diagnostic supervisor.	3
DS>	RUN EVSBA	Load and run the VAX autosizer @CONFIG. †	*
DS>	SEL ALL	Select all available devices.	*
DS>	RUN xxxxx	Run CPU and adapter diagnostics. ‡ Press <CTRL/P> to return to the >>> prompt.	*
>>>	BOOT	Load and start the operating system using the DEFBOO.COM command file.	*

* Time will vary depending upon system configuration, number of passes run, number of programs run, etc.

† The file CONFIG.COM, if available, may be submitted instead of running the autosizer.

‡ "xxxxx" is the name of the diagnostic to be run.

1.7 SUMMARY

This chapter presents an overview of the system's hardware and software which are designed to improve the system maintainability and availability, two very important factors in any large computer system. The following chapters will describe how to use these features in more detail.

CHAPTER 2
PROM TESTS

2.1 OVERVIEW

An 8 Kbyte PROM on the console module contains a PDP-11 self-test program for checking the console module hardcore logic on power-up. Besides the console self-tests, the PROM code also provides a minimal command set so the user can manually test the console logic. It also includes code to load the root console software program into T-11 RAM from the RL02 during system start-up and initialization. The PROM code facilities can be accessed from either the local or remote port. This chapter will describe how the PROM code operates, how it displays any errors detected, and how the PROM command set is used. Figure 2-1 summarizes the PROM test environment.

2.2 PROGRAM DESCRIPTION

PROM code initialization is performed whenever the machine power is turned on or whenever a power failure recovery is being attempted. Only part of the program is performed when a console reboot is requested by the CPU interrupt (TSTRT) or by the REBOOT command. Initialization begins with the execution of a set of self-tests and ends with the booting of the console program from the RL02.

Optionally, the user may wish to use the limited PROM command set to test the console further or perform some I/O register manipulation.

Invisible to the user is the fact that the console PROM is composed of two 4 Kbyte sections, only one of which is enabled at a time. When a segment is enabled, it occupies the T-11 address space between 164000(8) and 173776(8). This location reduces the amount of console memory space needed to run the 8 Kbyte PROM to only 4 Kbyte. Switching between segments is handled mainly by the PROM code itself, with the exception of a power-up condition.

2.2.1 Power-Up Operation

On power-up, the T-11 microprocessor first issues a console INIT pulse to clear most of the console hardware registers (which disables all T-11 interrupts and enables the lower segment of the PROM). The T-11 microprocessor then vectors to the PROM address 172000(8). Immediately after, the PROM begins a series of self-tests designed to verify logic critical to operation. It is assumed, however, that the T-11 chip and its instruction set are fully operational. The following subsection describes the self-tests in detail.

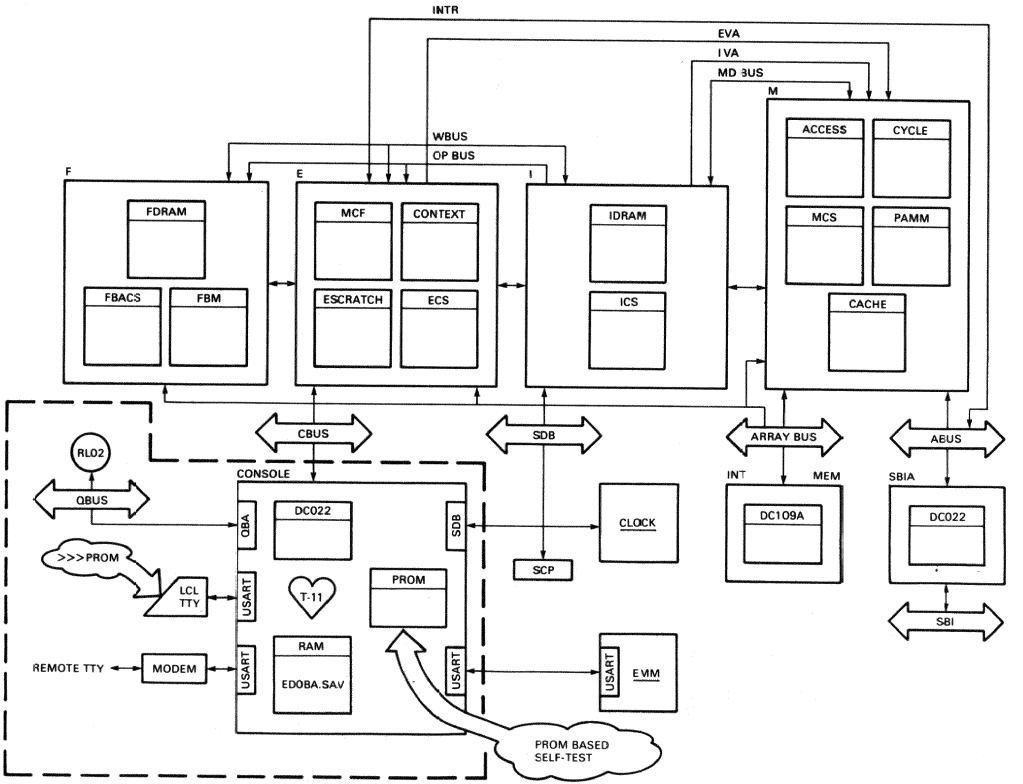


Figure 2-1 PROM Test Environment

When a self-test fails, the console loops indefinitely. In some cases, the front panel LEDs are used to indicate the failing test number. Most of the time, however, there will be an error message displayed on the CTY.

If none of the self-tests fail, the PROM code turns off the four front panel LEDs and sets up all T-11 interrupt vectors to point to the "unexpected interrupt handler" entry to the PROM (172010). It then transfers control to the upper 4 Kbyte segment. The upper segment begins by sending a <RETURN><LINE FEED> sequence (followed by a bell sound) to the CTY to signal the operator that the self-tests have completed and that the console program is about to be booted. A pause of three to four seconds occurs between the ringing of the CTY bell and the beginning of the RL02 boot sequence, during which any key typed on the CTY causes the PROM to enter its command input loop and display the ROM> prompt.

The RL02 boot sequence consists of a series of RL02-access tests prior to the actual boot attempt. These tests are explained below in the section describing the B command.

2.2.2 Self-Test Execution

The execution of the PROM self-tests occurs immediately after power is supplied to the T-11 processor and surrounding circuitry. It is assumed that the T-11 internal logic is operational.

Test loop controls similar to those found in diagnostic programs are used. A test will loop on error; respond to <CTRL/S>, <CTRL/Q>, and <CTRL/O> (if the test outputs anything); and will exit the loop in response to <CTRL/E> (only for some tests). Furthermore, a special trigger signal is asserted immediately after detection of the error and negated at the end of each test loop. This serves as a trigger input for oscilloscopes and analyzers by asserting the signal CL09 DIAG TRIGGER H on the CPU backplane.

The tests are arranged in a building block fashion so that logic is used only after it is tested. Because the CTY interface may be nonfunctional, the initial tests rely on the front panel LEDs to contain test numbers. When one of the primitive tests fails, the front panel LEDs will contain the test's number. As soon as the CTY interface is tested, the system banner is printed. The CTY is used to report errors beyond that point while the LEDs remain in their last set state (1001 [test 9]).

Errors reported to the CTY are displayed in the following format.

```
CONSOLE SELF-TEST DETECTED ERROR: TEST #n  SUBTEST #n
"descriptive message"
"data header"
"data ..."
```

The first couple of self-tests verify the integrity of the PROM code, the front panel LEDs, and much of the T-11 addressing and data paths.

Upon receiving power, the front panel LEDs are forced to the ON state. At this time, the PROM code begins its first test, which is to calculate the additive checksum of the entire 8 Kbyte PROM. If unsuccessful, the code falls into a "BR ." instruction and the LEDs remain in the "1111" state. Note that the "1111" state will also remain if some other critical T-11 logic is broken.

2.2.3 PROM Test Descriptions

Table 2-1 lists all of the PROM tests executed up to the point where the program displays the system banner.

At this point, the CTY interface, cable, and printer are assumed to be operational. Test failures from here on are reported on the CTY, and CTY input may also be recognized where indicated. The remaining tests in the PROM initialization are listed in Table 2-2. Tests that allow the <CTRL/E> input character can be exited if they fail, otherwise the test will loop indefinitely. The other input control characters can be used to stop (<CTRL/S>), resume (<CTRL/Q>) test execution, or suppress (<CTRL/O>) error report output.

Table 2-1 PROM Test Descriptions -- Part I

Test No.	Title/LED State/Description
0	PROM CHECKSUM TEST LED state = 1111 An additive checksum of both the lower and upper segments of the PROM is calculated. The result should be 377(8). On error, a "BR ." is executed.
1	SCP SDB CHANNEL TEST LED state = 0001 --> 0010 --> 0100 --> 1000 A floating 1 pattern is shifted through the LEDs at a fairly slow speed so the operator can tell if there is an LED failure. The test checks most of the SDB control logic and the SCP LED logic which is used by subsequent tests to report test numbers. The test is open-ended and cannot fail.
2	CTY INTERFACE MODE REGISTER 1 BIT TEST LED state = 0010 Alternating 01010101 and 10101010 patterns are written and read from the CTY's PCI_MODE_REGISTER_1.

Table 2-1 PROM Test Descriptions -- Part I (Cont.)

Test No.	Title/LED State/Description
3	CTY INTERFACE MODE REGISTER 2 BIT TEST LED state = 0011 Alternating 01010101 and 10101010 patterns are written and read from the CTY's PCI_MODE_REGISTER_2.
4	CTY INTERFACE COMMAND REGISTER BIT TEST LED state = 0100 Both a 01010101 and a 10101010 pattern are written and read from the CTY's PCI_COMMAND_REGISTER.
5	CTY INTERFACE RESET TEST LED state = 0101 A pattern is loaded into the PCI_COMMAND_REGISTER and a PCI RESET is performed. If the command register clears, then the PCI's RESET input is connected and working.
6	CTY INTERFACE TXRDY BIT TEST LED state = 0110 The local PCI is configured to run in its normal operating mode (9600 baud, 8-bit, no parity, 1 stop bit). The TXRDY bit in the PCI status register is then expected to set within 100 ms. If ok, the PCI transmit register is loaded and the TXRDY bit is expected to clear immediately.
7	CTY INTERFACE RXRDY BIT TEST LED state = 0111 The local PCI is reinitialized, but this time in local loop back mode. A character is transmitted and the RXRDY bit in the PCI status register is expected to set. When the RECEIVED_CHARACTER_REGISTER is read, the RXRDY bit is expected to clear.
10	CTY INTERFACE LOOPBACK TEST LED state = 1000 This is essentially the same as the previous test except that several loopback transmissions are performed to ensure the PCI and crystal can handle it.
11	CTY BANNER TEST LED state = 1001 This is an open-ended test that relies on the operator to observe the system banner.

Table 2-2 PROM Test Descriptions -- Part II

Test No.	Title/Control Characters/Description
12	PARITY ERROR LATCH TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. Tests whether the latch responsible for indicating parity errors can be set and cleared directly.
13	PARITY CIRCUIT TEST, PART 1 Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. Simultaneously tests that RAM location 0 will hold a 01010101 pattern. If all right, parity RAM location 0 is expected to contain 1.
14	PARITY CIRCUIT TEST, PART 2 Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. Simultaneously tests whether RAM location 0 will hold a 10101010 pattern. If all right, parity RAM location 0 is expected to contain 0 (since the "force parity error" bit in MCSR0 was set prior to depositing the pattern).
15	58 KB RAM DATA/ADDRESS TEST, BOTTOM-UP Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. A modified moving-inversions test is performed on the first 58 Kbytes of physical RAM. The test verifies all data faults that are likely to occur in the console RAM configuration, as well as verifying all addressing faults in the positive (incrementing) direction. On error, the address, expected data, and received data are displayed.
16	58 KB RAM DATA/ADDRESS TEST, TOP-DOWN Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. A modified moving-inversions test is performed on the first 58 Kbytes of physical RAM. The test verifies all data stuck-at faults likely to occur in the console RAM configuration, as well as all addressing faults in the negative (decrementing) direction. On error, the address, expected data, and received data are displayed.

Table 2-2 PROM Test Descriptions -- Part II (Cont.)

Test No.	Title/Control Characters/Description
17	MAP RAM LOCATION 0 QV TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. A loop is first performed that uses the first mapping RAM location (MAPR00) to initialize all of physical memory with 0s and good parity, and places each 4 Kbyte page number in the first location of its own 4 Kbyte page boundary. The process is repeated to check that the first byte of each page contains the correct value.
20	MAPPING RAM DATA TEST, BOTTOM-UP Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. This test uses the memory pattern verified by the previous test to check that all mapping RAM locations can access pages uniquely, in the positive direction.
21	MAPPING RAM DATA TEST, TOP-DOWN Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. This test uses the memory pattern verified by the previous test to check that all mapping RAM locations can access pages uniquely, in the negative direction.
22	MAPPING RAM ADDRESSING TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. This test uses the memory pattern verified by the previous tests to check that all mapping RAM locations are themselves uniquely addressable. The memory mapper is then turned off.
23	TOY CHIP ACCESS TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. This test checks that registers in the console's TOY chip can be accessed to verify that the TOY chip is receiving power from the +5 B signal input from the BBU.
24	RTY INTERFACE MODE REGISTER 1-BIT TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. Normal operation patterns are loaded into the RTY's PCI_MODE_REGISTER_1 and checked.

Table 2-2 PROM Test Descriptions -- Part II (Cont.)

Test No.	Title/Control Characters/Description
25	RTY INTERFACE MODE REGISTER 2-BIT TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. Normal operation patterns are loaded into the RTY's PCI_MODE_REGISTER_2 and checked.
26	RTY INTERFACE COMMAND REGISTER BIT TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. Both a 01010101 and a 10101010 pattern are written and read from the RTY's PCI_COMMAND_REGISTER.
27	RTY INTERFACE RESET TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. A pattern is loaded into the PCI_COMMAND_REGISTER and a PCI RESET is performed. If the command register clears, it means the PCI's RESET input is connected and working.
30	RTY INTERFACE TXRDY BIT TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. The remote PCI is configured to run in its normal operating mode (1200 baud, 8-bit, no parity, 1 stop bit). The TXRDY bit in the PCI status register is then expected to set within 100 ms. If ok, the PCI transmit register is loaded and the TXRDY bit is expected to clear immediately.
31	RTY INTERFACE RXRDY BIT TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. The remote PCI is reinitialized, but this time in local loopback mode. A character is transmitted and the RXRDY bit in the PCI status register is expected to set. When the RECEIVED_CHARACTER_REGISTER is read, the RXRDY bit is expected to clear.
32	RTY INTERFACE LOOPBACK TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. This test duplicates the function of the previous test in order to verify that the remote PCI can handle consecutive character transmissions.

Table 2-2 PROM Test Descriptions -- Part II (Cont.)

Test No.	Title/Control Characters/Description
33	SCP SDB LOGIC TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>, <CTRL/E>. This test checks the continuity of the SDB control channel on the SCP. This test will affect the state of the front panel LEDs, but is done so quickly it is unnoticeable.
34	"CL15 TSTRT" INTERRUPT TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>. This test checks that the TSTRT interrupt input to the console is not asserted. This interrupt is not maskable through the PSW, so it must be cleared before the console program will run.
35	UNEXPECTED INTERRUPT TEST Allowable control characters are <CTRL/S>, <CTRL/Q>, <CTRL/O>. This test checks that after a full console reset there are no pending (or stuck) interrupts to the T-11 microprocessor. This ensures the proper start-up of RT and the console kernel.

2.2.4 PROM Command Loop

When the PROM code signals the completion of the self-tests by ending the bell character to the CTY, the operator can type any key to signal the PROM to enter its command null loop. While in the null loop, the PROM code may service the RTY device, as well as the CTY, for possible input (provided the front panel switch is in the REMOTE position and a remote connection has been made). The PROM code indicates its readiness for input by displaying the ROM> prompt.

The command parser in the PROM is extremely simple. It accepts single-character commands and alphanumeric arguments. Table 2-3 lists the commands available to the user.

Table 2-3 PROM Command Set

Command	Description
B	Boot the console software from the RL02. This command begins by testing the interface between the console and the RL02 controller. The sequence of tests follows. 1. Checks for AC LOW and DC LOW conditions in the BA11. 2. Checks that console-generated BUS INIT reaches the RL02 controller and clears all RL02 controller registers. 3. Checks that RLCS, RLBA, and RLDA will retain a full complement of patterns. 4. Performs an RL02 MAINTENANCE TRANSFER to verify DMA logic between the console and RL02, and other logic on the RLV12 controller. 5. Checks that the first word read in from the RL02 boot block contains "240" (NOP). If any of the above checks fail, a message is reported to both the CTY and RTY, and the B command aborts. There is no test looping capability. If a drive or controller status error is detected during the actual booting of the device, a number of retries are made.
D addr data	Deposit the data word to the specified address. The address must be an even number.
E addr	Examine the data word at the specified address. Odd addresses are made even by dropping the low order bit.
S addr	Start the T-11 execution at the specified address. The address must be an even number. The command-argument delimiter is optional.
T [filename]	Without any argument, this command reruns the PROM self-tests and then loads and runs the console diagnostic program (ED0BA) for two passes. If a filename is specified, the self-tests are not run and the file is loaded and started at location 200. The default filename extension is .SAV.

Table 2-3 PROM Command Set (Cont.)

Command	Description
Q addr data	This command allows direct deposits to Q-Bus registers. The addresses 174400, 174402, 174404, and 174406 are the RLCS, RLBA, RLDA, and RLMPR registers, respectively. All other addresses are rejected.
R addr	This command allows direct examines of Q-Bus registers. The addresses 174400, 174402, 174404, and 174406 are the RLCS, RLBA, RLDA, and RLMPR registers, respectively. All other addresses are rejected.
V	With this command, the PROM code enters a loop which allows characters to pass directly between the CTY and RTY. The escape sequence "<CTRL/P><CTRL/X><CTRL/P>" can be entered from either input device to force the PROM code to exit this mode.
X	This command complies (less any switches) with the description of same in Chapter 11 of Digital Standard 032. It is intended to be used for binary data transfers between T-11 memory and a computer using the remote port input. It is not intended for human use and will work properly only when issued from the RTY device.

2.2.5 PROM RTY Support

The PROM code services the RTY port with a much simpler procedure than the console program's. This simplicity is necessary since the PROM code has neither a timer nor an interrupt capability.

After the PROM code completes its power-up self-tests, which include loopback tests on the RTY PCI, it configures the PCI to operate with default characteristics (same defaults used by the SET TERMINAL command). Once the PCI is set up, and if the front panel Terminal Control Switch is in the REMOTE position, the PROM asserts the DTR signal to the modem, thus allowing the modem to turn on-line. If the DSR and CD signals from the modem are sensed as TRUE, then all character output to the CTY will also be sent to the RTY port, and input will be accepted from either. Note that because the DTR signal to the modem is not asserted until the end of the self-tests, it will be virtually impossible to establish a modem connect to the RTY port during system power-up initialization while the PROM is still running. However, a terminal connected locally with a NULL MODEM cable should be able to get control of the PROM code; this is why this scheme becomes useful.

Once the PROM asserts DTR to the modem, the dial-in capability is activated. When the console program starts running, it begins to service the RTY port immediately, thus giving the remote user control throughout PROM-to-console program transitions, and vice versa. This means that if T-11 control transfers to the PROM because of some unexpected condition, such as for a T-11 HALT or unexpected interrupt, the PROM can continue to service the remote connection.

During the handling of connections and disconnections of the remote terminal by the PROM code, the state of the front panel indicators is updated in accordance with how those indicators are defined in this document.

NOTE

The console diagnostic EDOBA invoked with the PROM T command tests the RTY interface which causes the remote port to be disconnected. This prevents running EDOBA from the remote port.

2.2.6 Special Services

The console PROM code provides two special services to the console program. The first is to handle unexpected interrupts and the second is to handle console reboot requests. These services are described next.

2.2.6.1 Unexpected Interrupt Handling -- Whenever the PROM code receives control of the T-11, whether by first-time power-up, a console reboot, or an unexpected interrupt, it first disables T-11 interrupts by raising the Interrupt Priority Level (IPL) in the PSW to 7 (highest). It then loads all T-11 interrupt vectors with a pointer to the PROM's unexpected interrupt service routine (172010). The new PSW, in the location following each vector location, contains a 4-bit ID in the condition code field such that a single service routine entry can be used for all interrupt vectors.

There are 16 possible vectored interrupts on the console module, 2 of which are unused and can never occur. Each vector is dedicated for one specific device. Table 2-4 lists the vectors, priority levels, and ID codes for all possible console interrupts.

The same devices handled by vectors 24, 70, and 140 are always handled by the PROM code as unexpected interrupts.

Table 2-4 Console Interrupt Vectors

Vector	IPL	ID	I/O Device
24	*	00	PROM restart
60	4	01	Remote PCI transmit/receive/modem
64	4	02	Local PCI transmit/receive
70	4	03	Q-Bus reply timeout
100	6	04	Unused
104	6	05	TOY 1 ms interrupt
110	6	06	CPU control store parity error
114	6	07	STOR RDY
120	5	10	TXCS RDY
124	5	11	RXCS DNE
130	5	12	Q-Bus adapter
134	5	13	EMM PCI transmit/receive
140	7	14	Console RAM parity error
144	7	15	Unused
150	7	16	System ac low
154	7	17	ABus dead

* Unmaskable by PSW

When the PROM receives control at its unexpected interrupt service routine, it decodes the 4-bit ID number in the condition code field of the PSW and displays the vector through which the transfer was made. Then, the PROM code displays the contents of the T-11 registers (R0 through R7 and PSW) at the time of the interrupt. In the case of vector 140, console RAM parity error, the PROM code constructs the address of the parity error (from the PECAS and PERAS registers) and displays this, along with the contents of the location and whether or not the parity error is hard or soft.

Because most of the interrupts on the console are level-sensitive rather than edge-sensitive, there is no easy way for the PROM to dismiss the interrupt and return to the previous T-11 instruction stream. Thus, the PROM code enters its command loop and waits for operator intervention, or a CPU reboot request, to restart the console program.

2.2.6.2 Console Reboot Handling -- In addition to the 16 vectored interrupts described in the previous section, there is another which forces the console to begin execution at the PROM entry responsible for handling console reboots (172004). When the T-11 interrupt facility is enabled (MCSRQ<0>=1), it treats this interrupt as an unmaskable, edge-sensitive input which saves the PC and PSW, forces the IPL to priority 7, and vectors the T-11 directly to the reboot address. This same operation occurs when the T-11 microprocessor executes a HALT instruction.

If the entry into the reboot/halt routine was caused by a TSTRT interrupt (generated either by an instruction or the console's REBOOT command), the message "?Console Reboot Initiated" is printed and the console reboots the RL02. The CPU will cause a TSTRT interrupt if it feels the console has not updated the TOY register for a considerable period of time.

If the cause of the entry was due to a T-11 HALT instruction, the "?T-11 HLT" message is printed, followed by the T-11 registers saved prior to the halt. The PROM code then enters its command loop and waits for input. While the PROM code waits in its command loop, it is still sensitive to TSTRT interrupt requests from the CPU.

2.3 PROM ERROR REPORTING

If the PROM self-tests detect any errors, they are reported by two different methods depending upon which test detected the failure.

If the failure is detected before test 12(8), the failing test number is indicated by the 4-bit code displayed in the SCP indicators, and the failing test loops indefinitely. The only way to stop the program is to power down the system. You can refer to the test descriptions listed in Paragraph 2.2.3 to determine which logic function failed. In most all cases, excluding power-supply problems, the L0201 console module is bad and should be replaced.

Tests 12(8) through 35(8) provide error message displays on the console TTY and accept user input (<CTRL/E>, <CTRL/S>, <CTRL/Q>, and <CTRL/O>) to permit user control of the PROM code. Like the previous tests, the program will loop continuously on the failing test, but the user can press <CTRL/E> to cause an exit from the failing test and proceed with the next test in sequence. Figure 2-2 shows the general format of the error messages along with a couple of examples. In most cases, the message will call out the functional logic area that is failing. Like the previous tests, most of the error message displays indicate that you have a bad L0201 console module.

Finally, after successful completion of test 35(8), the PROM will perform several RL02 tests to verify the RL02 Q-Bus interface and then proceed to boot the console software from the RL02 disk. If any solid RL02 interface faults are detected, they will be reported on the console terminal. If all the other tests run successfully, and only the RL02 tests fail, you most likely have an RL02 disk subsystem problem.

```
General Format:

CONSOLE SELF_TEST DETECTED ERROR: TEST # nn, SUBTEST # sss
optional: may be followed by a descriptive error message

Sample Outputs:

CONSOLE SELF_TEST DETECTED ERROR: TEST # 000033, SUBTEST # 000001
SCP LOGIC ERROR
EXP'D   RCV'D
000016  000000


CONSOLE SELF_TEST DETECTED ERROR: TEST # 000015, SUBTEST # 000001
DATA RAM CELL ERROR
ADDR    XOR
000000  000010
```

Figure 2-2 PROM Tests -- Error Displays

2.4 TROUBLESHOOTING PROCEDURES

Let's summarize the operation and use of the PROM tests using the simple troubleshooting flow chart shown in Figure 2-3. The figure summarizes what should happen when you first turn on the power switch. Assume that the Restart Control Switch is in the HALT position and the standard RL02 disk pack is installed.

1. Turn the Terminal Control Switch to the LOCAL position.

2. The fans start up and the SCP indicators rapidly twinkle from "1111" to "0001 --> 0010 --> 0100 --> 1000" to "0010" through "1001."

3. The PROM banner is displayed on the CTY and the SCP indicators remain at "1001."

4. The RL02 READY light blinks as the boot block is read.

5. The console software is loaded. It self-starts and then displays its banner and initializing messages.

6. Finally, the macro context prompt >>> is displayed, indicating it's ready to accept console commands.

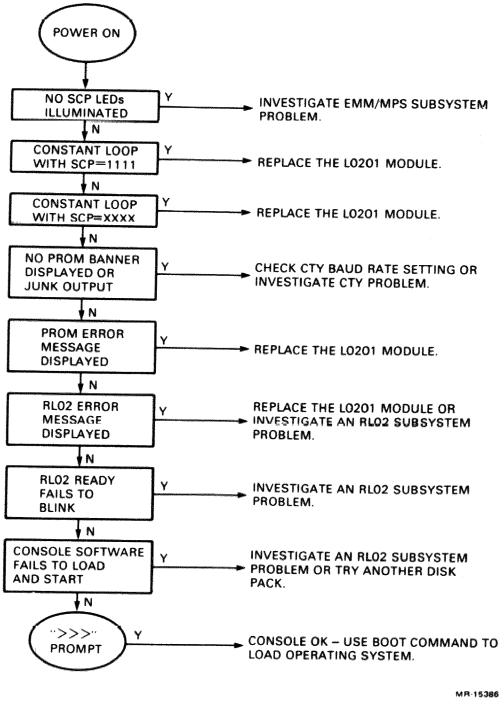


Figure 2-3 Simplified PROM Troubleshooting Flow Chart

If this sequence fails, the flow chart should help you decide how to correct the problem. The four major sources of problems follow.

1. Nothing happens -- investigate power system problems (EMM/MPS)

2. Tests 00(8) through 11(8) pass but no PROM banner is displayed -- investigate CTY problems (terminal, cables, baud rate setting)

3. PROM banner displayed followed by PROM error messages -- replace the L0201 console module

4. All PROM tests pass but console software fails to load and start -- investigate RL02 disk subsystem fault or a bad disk pack

2.5 SUMMARY

The PROM tests check out the T-11 processor, RAM, and hard core logic required for accessing the RL02 used to load T-11 programs. The next step in bottom-up testing is to load and run ED0BA, the console diagnostic, which performs more extensive testing of the console subsystem. That is the subject of the next chapter.

CHAPTER 3
ED0BA - CONSOLE DIAGNOSTIC

3.1 OVERVIEW

ED0BA is a T-11 based console module diagnostic designed to test extensively all the logic on the console module, including the clock, SDB, CBus, Q-Bus, and local and remote PCI interface logic. It assumes that the console hardcore logic, tested by the PROM tests, is working. Figure 3-1 shows what the test environment is when running ED0BA. Note that ED0BA is a T-11 based test program that does not test any of the VAX CPU logic itself. The hardware used and tested by the console diagnostic is outlined by the broken lines.

3.2 PROGRAM DESCRIPTION

The following sections describe the test sequences, operating procedure, and error message formats. Also included is a description of the switch register options used to control ED0BA operation.

3.2.1 Test Sequence

There are 162(10) major tests, T001(8) through T242(8). Table 3-1 groups tests by type and describes the general function and number of subtests in each. An asterisk indicates that a special test fixture is required; the test is ignored unless the fixture is installed.

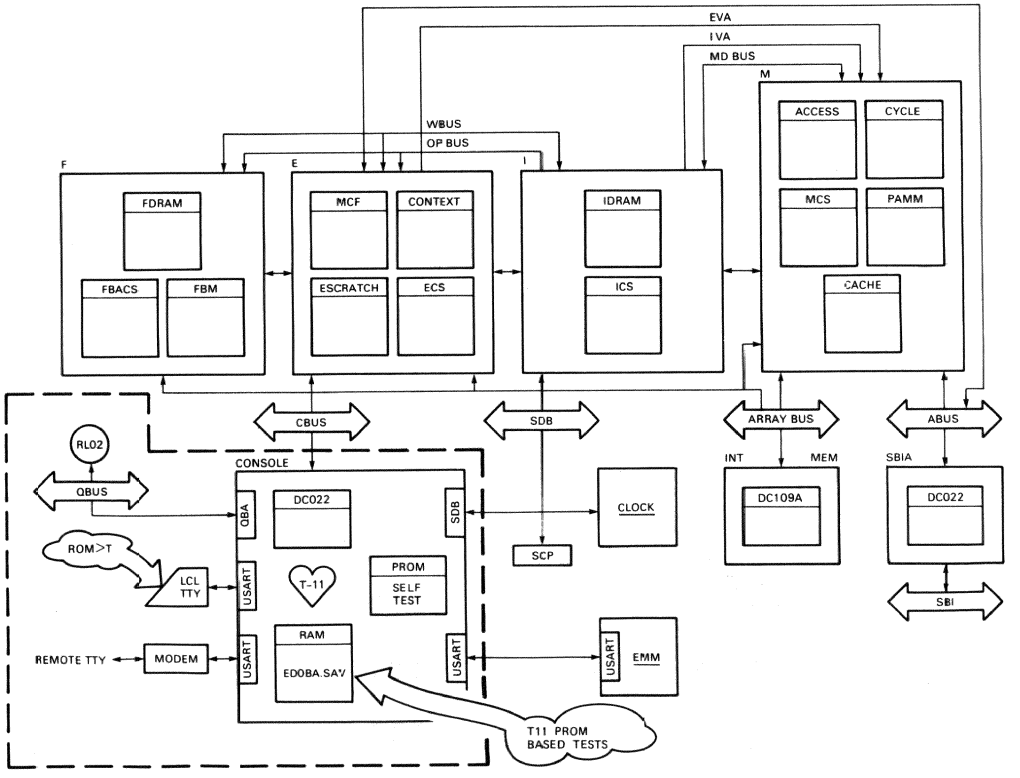


Figure 3-1 ED0BA Test Environment

Table 3-1 ED0BA Test Descriptions

Test No.	Description	No. of Subtests
SDB Control Logic Tests		
001	SDMS REGISTER TEST (CL20)	3
002	SDDB REGISTER TEST (CL20)	3
003	SDCS REGISTER TEST (CL20)	3
004	CL18 STOP CLOCK TEST (CL18)	2
005	DB SINGLE STEP MODE TEST (CL19)	3
006	SDB NORMAL MODE TEST (CL19)	2
007	SDB LOOPBACK TEST, NORMAL MODE (CL19)	2
Tester SDB Channel Continuity Tests		
010	SDB CHANNEL 0 CONTINUITY TEST (CL21)	1*
011	SDB CHANNEL 1 CONTINUITY TEST (CL21)	2*
012	SDB CHANNEL 2 CONTINUITY TEST (CL21)	2*
013	SDB CHANNEL 3 CONTINUITY TEST (CL21)	1*
014	SDB CHANNEL 4 CONTINUITY TEST (CL21)	2*
015	SDB CHANNEL 5 CONTINUITY TEST (CL21)	1*
016	SDB CHANNEL 6 CONTINUITY TEST (CL21)	2*
017	SDB CHANNEL 7 CONTINUITY TEST (CL21)	1*
020	SDB CHANNEL 8 CONTINUITY TEST (CL21)	1*
021	SDB CHANNEL 9 CONTINUITY TEST (CL21)	1*
022	SDB CHANNEL 10 CONTINUITY TEST (CL21)	1*
023	SDB CHANNEL 11 CONTINUITY TEST (CL21)	1*
024	SDB CHANNEL 12 CONTINUITY TEST (CL21)	1*
025	SDB CHANNEL 13 CONTINUITY TEST (CL21)	2*
026	SDB CHANNEL 14 CONTINUITY TEST (CL21)	2*
027	SDB CHANNEL 15 CONTINUITY TEST (CL21)	1*
030	SDB CHANNEL 16 CONTINUITY TEST (CL21)	1*
031	SDB CHANNEL 17 CONTINUITY TEST (CL21)	1*
032	SDB CHANNEL 18 CONTINUITY TEST (CL21)	1*
033	SDB CHANNEL 19 CONTINUITY TEST (CL21)	1*
034	SDB CHANNEL 20 CONTINUITY TEST (CL21)	1*
035	SDB CHANNEL 21 CONTINUITY TEST (CL21)	1*
036	SDB CHANNEL 22 CONTINUITY TEST (CL21)	1*
037	SDB CHANNEL 23 CONTINUITY TEST (CL21)	1*
Miscellaneous Register Tests		
040	MCSR0 REGISTER TEST (CL08)	3
041	MCSR1 REGISTER TEST (CL08)	2
042	MCSR2 REGISTER TEST, PART 1 (CL08)	3
043	MCSR2 REGISTER TEST, PART 2 (CL08)	3

* Indicates a special test fixture is required; the test is ignored unless the fixture is installed.

Table 3-1 EDOBA Test Descriptions (Cont.)

Test No.	Description	No. of Subtests
Miscellaneous Console Functions Tests		
044	CL09 DC LOW TEST (CL20)	2*
045	CL CSM REQ TEST (CL09)	2*
046	EMM3 CPU AC LO TEST (CL08)	2*
047	CL09 SYS AC FAULT TEST (CL08)	4*
050	CL CPU PF INTR TEST (CL09)	2*
051	CL UNHANG RESET TEST (CL18)	2*
052	CL MASTER RESET TEST (CL09)	2*
053	CL ABUS ENABLE TEST (CL09)	2*
054	CL ARRAY DC OK TEST (CL09)	3*
055	CL09 CPU ALIVE TEST (CL08)	3*
056	CL09 ERROR 0 TEST (CL08)	2*
057	CL09 ERROR 1 TEST (CL08)	2*
060	CL09 ERROR 2 TEST (CL08)	2*
061	CL09 CPU CS PE TEST (CL08)	6*
062	CL09 ABUS REQ0 TEST (CL08)	2*
063	CL09 ABUS REQ1 TEST (CL08)	2*
064	CL09 ABUS REQ2 TEST (CL08)	2*
065	CL09 ABUS REQ3 TEST (CL08)	2*
066	CL09 ABUS DEAD INTR TEST (CL08)	5*
067	SID0 REGISTER TEST (CL12)	2*
070	SID1 REGISTER TEST (CL12)	2*
071	SID2 REGISTER TEST (CL12)	2*
072	CL ID0 BIT TEST (CL08)	2*
073	CL ID1 BIT TEST (CL08)	2*
074	CL ID2 BIT TEST (CL08)	2*
EMM PCI Tests		
075	EMM PCI MODE REGISTER 1 TEST (CL10)	3
076	EMM PCI MODE REGISTER 2 TEST (CL10)	3
077	EMM PCI COMMAND AND STATUS REGISTER TESTS (CL10)	4
100	EMM PCI REGISTER ADDRESSING TESTS (CL10)	3
101	EMM PCI TxRDY BIT TEST (CL10)	1
102	EMM PCI LOCAL LOOPBACK TEST, 19.2K BAUD (CL10)	1
103	EMM PCI BUS LOOPBACK TEST, 19.2K BAUD (CL10)	1
104	EMM PCI BUS INHIBIT TEST (RTS DEASSERTED) (CL10)	1
Remote PCI Tests, Part 1		
105	REMOTE PCI MODE REGISTER 1 TEST (CL10)	3
106	REMOTE PCI MODE REGISTER 2 TEST (CL10)	3
107	REMOTE PCI COMMAND AND STATUS REGISTER TESTS (CL10)	4
110	REMOTE PCI REGISTER ADDRESSING TESTS (CL10)	3

* Indicates a special test fixture is required; the test is ignored unless the fixture is installed.

Table 3-1 EDOBA Test Descriptions (Cont.)

Test No.	Description	No. of Subtests
Remote PCI Tests, Part 2		
111	CDSRS TEST (CL11)	2*
112	CL11 RTERM CARRIER TEST (CL10)	2*
113	CL11 RTERM DSR TEST (CL10)	2*
114	CRTS TEST (CL10)	2*
115	ASSERT CDTR TEST (CL10)	2*
116	REMOTE PCI TxEMT/DSCHG TEST (CL10)	1*
117	CL RTERM DSC TEST (CL10)	1*
120	REMOTE PCI TxRDY TEST, PART 1 (CL10)	1*
Remote PCI Tests, Part 3		
121	REMOTE PCI LOCAL LOOPBACK TEST, 75 BAUD (CL10)	1
122	REMOTE PCI LOCAL LOOPBACK TEST, 110 BAUD (CL10)	1
123	REMOTE PCI LOCAL LOOPBACK TEST, 150 BAUD (CL10)	1
124	REMOTE PCI LOCAL LOOPBACK TEST, 1800 BAUD (CL10)	1
Remote PCI Test, Part 4		
125	REMOTE PCI SPLIT BAUD RATE TEST (CL10)	1*
SCP Interface Tests, Part 1		
126	SCP CHANNEL CONTINUITY TEST (SCP1)	1
127	SCP CHANNEL INIT TEST (SCP1)	1
130	SCP LED DRIVE/SENSE TEST (SCP1)	3
SCP Interface Tests, Part 2		
131	SCP SWITCH INPUT TEST (SCP1)	2*
CBus Tests, Part 1		
132	CL15 TSEL TEST (CL13)	3
133	CL15 TXCS RDY TEST, PART 1 (CL13)	3*
134	CL15 RXCS DNE TEST, PART 1 (CL13)	3*
135	CL15 STOR RDY TEST, PART 1 (CL13)	3*
136	CL15 TSTRT TEST (CL13)	3

* Indicates a special test fixture is required; the test is ignored unless the fixture is installed.

Table 3-1 EDOBA Test Descriptions (Cont.)

Test No.	Description	No. of Subtests
CBus Tests, Part 2		
137	CBUS RAM DATA TEST, TTL PORT, PART 1 (CL16)	2*
140	CBUS RAM ADDRESSING TEST, TTL PORT (CL16/CL17)	1*
141	CBUS ACCESS TEST, ECL PORT (CL16)	2*
142	CBUS RAM ADDRESSING TEST, ECL PORT (CL16/CL17)	1*
143	CL15 TXCS IE TEST (CL13)	5*
144	CL15 RXCS IE TEST (CL13)	5*
145	CL15 STOR TIE TEST (CL13)	5*
146	CL15 TSTRT TEST (CL13)	4*
147	CL15 TXCS RDY TEST, PART 2 (CL13)	2*
150	CL15 TXCS RDY TEST, PART 3 (CL13)	2*
151	CL15 RXCS DNE TEST, PART 2 (CL13)	2*
152	CL15 RXCS DNE TEST, PART 3 (CL13)	1*
153	CL15 STOR RDY TEST, PART 2 (CL13)	2*
154	CL15 STOR RDY TEST, PART 3 (CL13)	1*
155	CL15 TSEL SENSED BY ECL PORT TEST (CL13)	2*
156	CL TTX INTR TEST (CL15)	4*
157	CL TRX INTR TEST (CL15)	4*
160	CL RL02 INTR TEST (CL15)	4*
TOY Clock Tests		
161	TOY SOFTWARE-MASTER-RESET TEST (CL22)	16
162	TOY COUNTER GROUP 1 REGISTER TEST (CL22)	8
163	TOY COUNTER GROUP 2 REGISTER TEST (CL22)	8
164	TOY COUNTER GROUP 3 REGISTER TEST (CL22)	8
165	TOY COUNTER GROUP 4 REGISTER TEST (CL22)	8
167	TOY COUNTER GROUP 5 REGISTER TEST (CL22)	8
170	TOY MASTER MODE REGISTER TEST (CL22)	2
171	TOY COUNTER REGISTER COUNT TEST (CL22)	15
172	CL22 TOY 15 OUT TEST (CL22)	1
RAM Parity Tests		
173	RAM DATA PARITY TESTS (CL06)	3
Mapped RAM Data and Addressing Tests (64 Kbytes to 256 Kbytes)		
174	MAPPED RAM DATA/ADDRESSING TEST (CL04)	2

* Indicates a special test fixture is required; the test is ignored unless the fixture is installed.

Table 3-1 EDOBA Test Descriptions (Cont.)

Test No.	Description	No. of Subtests
T-11 Interrupt Tests		
175	CL19 ENA TXRDY TEST (CL20)	3
176	CL19 ENA STOR RDY TEST (CL20)	3
177	UNSOLICITED INTERRUPT TEST (CL02)	3
200	CL15 TSTRT INTERRUPT TEST (CL02)	3
201	CL04 CSL PE INTR INTERRUPT TEST (CL02)	6
202	CL09 SYS AC FAULT INTERRUPT TEST, PART 1 (CL02)	5*
203	CL09 ABUS DEAD INTR INTERRUPT TEST (CL02)	4
204	CL22 TOY 1MS INTR INTERRUPT TEST (CL02)	6
205	CL15 TXCS RDY INTERRUPT TEST (CL02)	5
206	CL15 STOR RDY INTERRUPT TEST (CL02)	5
207	CL31 QBA INTR INTERRUPT TEST (CL02)	6
210	CL10 ETERM RDY INTERRUPT TEST (CL02)	5
211	CL11 RTERM INTR INTERRUPT TEST (CL02)	6
212	CL10 LOCAL RDY INTERRUPT TEST (CL02)	5
213	CL30 RPLY TIMEOUT INTERRUPT TEST (CL02)	4
T-11 Interrupt Tests		
214	CL09 CPU CS PE INTERRUPT TEST (CL02)	4*
215	T-11 MANUAL RESTART INTERRUPT TEST (CL02)	3*
Q-Bus Adapter Tests		
216	QCSR0 REGISTER TEST (CL29)	4
217	QCSR1 REGISTER TEST (CL29)	3
220	QBA ADDRESS REGISTER TEST (CL27)	2
221	QBA DATA REGISTER TEST (CL27)	2
222	CL31 CSL MASTER PEND TEST (CL29)	5
223	QBA TIMEOUT COUNTER TEST (CL20)	6
224	CL30 RPLY TIMEOUT TEST, PART 1 (CL20)	6
225	QBA SIMULATED-READ-CYCLE TEST, PART 1 (CL23)	13
226	QBA SIMULATED-READ-CYCLE TEST, PART 2 (CL23)	2
227	QBA SIMULATED-WRITE-CYCLE TEST (CL23)	13
230	QBA SIMULATED-INTR-ACK-CYCLE TEST, PART 1 (CL23)	8
231	SIMULATED-INTR-ACK-CYCLE TEST, PART 2 (CL23)	3
232	QBA CL32 SREC SACK TEST (CL29)	3
233	QBA SIMULATED-DMA-READ TEST, PART 1 (CL23)	24
234	QBA SIMULATED-DMA-READ TEST, PART 2 (CL23)	3
235	QBA SIMULATED-DMA-WRITE TEST (CL23)	24
236	CL30 RPLY TIMEOUT TEST, PART 2 (CL29)	2

* Indicates a special test fixture is required; the test is ignored unless the fixture is installed.

Table 3-1 EDOBA Test Descriptions (Cont.)

Test No.	Description	No. of Subtests
RL02-Specific QBA Tests		
237	RL02 REGISTER READ TEST (CL23)	10
240	RL02 REGISTER WRITE TEST (CL23)	10
241	RL02 REGISTER WRITE/READ TEST (CL23)	2
242	RL02 DMA READ TEST (CL23)	3

* Indicates a special test fixture is required; the test is ignored unless the fixture is installed.

3.2.2 Loading and Running EDOBA

When at one of the three possible CIO mode contexts (macro [>>>], diag [DC>], or macrohardcore [MH>]), type the PROM command to transfer control to the PROM code, which responds with the ROM> prompt as shown in Figure 3-2. At this point, typing T will load EDOBA from the RL02 and self-start it at location 200(8), as shown in Figure 3-3.

```
>>>PROM
   PROM restart -- confirm (Y/N) Y

?T11 HLT
?REGS 140272 000102 000211 026464 033244 033702 000740 000162 000000

ROM>
```

Figure 3-2 Transferring Control to the PROM Code

```
ROM>T


EDOBA V35 (25-Jun-1985)

          CONFIDENTIAL DIAGNOSTIC SOFTWARE
               PROPERTY OF
          DIGITAL EQUIPMENT CORPORATION

   Use Authorized Only Pursuant to a Valid Right-to-Use License

    0 ERRORS DETECTED;      0 TOTAL ERRORS AFTER PASS # 1
    0 ERRORS DETECTED;      0 TOTAL ERRORS AFTER PASS # 2
ROM>
```

Figure 3-3 Starting EDOBA

EDOBA runs two complete passes, which takes approximately three minutes, and then returns to the ROM> prompt to await user input. EDOBA provides switch register option and control character recognition features so the user can control program operation. Pressing <CTRL/G> will signal EDOBA to pause at the end of the current test, display the SWR prompt, and wait for user input. At this time, the user may choose to type one of the following.

- <CTRL/P> -- to exit to the ROM> prompt
- <CTRL/C> -- to restart EDOBA at location 200(8)
- xxxxxx(8) -- to resume execution with the SWR=xxxxxx(8)

Note in Figure 3-2 that the software first displays the contents of the T-11 registers, then the ROM> prompt. The format of the dump is as follows.

```
?REGS (R0) (R1) (R2) (R3) (R4) (R5) (SP) (PS) (PC)
```

Table 3-2 lists the EDOBA switch register options.

EDOBA also responds to the control characters shown in Table 3-3.

Table 3-2 EDOBA Switch Register Options

Bit Number	SWR	Description
15	100000	Exit subtest loop and proceed
14	040000	Loop on failing subtest
13	020000	Inhibit error display
12	010000	Enable trace
11	004000	Inhibit test iterations
9	001000	Loop on test # in <07:00>
<07:00>	000xxx	Select test xxx for loop

Table 3-3 EDOBA Control Characters

Control Key	Function
<CTRL/S>	Suspend output
<CTRL/Q>	Resume output
<CTRL/G>	Enter SWR-CHANGE mode
<CTRL/U>	Erase command line
<CTRL/C>	Restart EDOBA
<CTRL/P>	Exit to the PROM CLI

Figure 3-4 shows how to use the TRACE option. Note that the priority software disclaimer message is not displayed when the program is restarted. It is important to remember that changes in the switch register are not recognized by the program until it is restarted. If the user presses <CTRL/G> to enter the SWR prompt and the switch settings are changed, the program will not respond to the new settings. To overcome this limitation, proceed as follows.

1. Press <CTRL/G> to get back to the SWR prompt.
2. Press <CTRL/P> to get back to the ROM> prompt.
3. Type S 200 as shown in Figure 3-4.

Note how certain tests are skipped because the special manufacturing tester was not connected. After successful completion of ED0BA, the user types the B command at the ROM> prompt to load and start the console software (ED0AA.SAV).

```
ROM>S 200

ED0BA V35 (25-Jun-1985)

SWR = 000000  NEW = 100000

BEGIN TEST #000001
BEGIN TEST #000002
BEGIN TEST #000003
BEGIN TEST #000004
BEGIN TEST #000005
BEGIN TEST #000006
BEGIN TEST #000007
IGNORE TEST #000010
IGNORE TEST #000011
   .
   .
   .
   .
IGNORE TEST #000036
IGNORE TEST #000037
BEGIN TEST #000040
BEGIN TEST #000041
IGNORE TEST #00042
BEGIN TEST #000043^G

SWR = 010000 NEW = ^P

ROM>
```

Figure 3-4 Running ED0BA with TRACE Option Set

3.2.3 Error Reporting

EDOBA's message format is shown below.

```
ERROR DETECTED BY TEST #xxx, SUBTEST #yyy
"descriptive error message"
"expected and received data"
```

The "descriptive error message" will, in most cases, be a console module signal name and print designation that point to the logic that was being tested when the fault was detected. The user must refer to the program listing to obtain further details about the test sequence. In most cases, a subtest failure will isolate the fault to a few components, ensuring quick repair of the console module.

The "expected and received data," (good/bad) information, is unique to the specific subtest failing. Each data item has a header that indicates what the data represents. In some cases, the data field will be blank if it has no meaning. Figure 3-5 shows two sample error displays.

```
General Format:

ERROR DETECTED BY TEST #xxx, SUBTEST #yyy
"descriptive error message"
"expected and received data"

EXAMPLE 1:

ERROR DETECTED BY TEST #4, SUBTEST #1
"CL18 STOP CLOCK" FAILURE (CL18)

EXAMPLE 2:

ERROR DETECTED BY TEST #40, SUBTEST #2
"MCSR0 REGISTER BIT FAILUR (CL08)
EXPD      RCVD
000242    000252
```

Figure 3-5 EDOBA Error Displays

3.3 TROUBLESHOOTING

Since the majority of logic testing performed by ED0BA is meant to verify logic circuits on the L0201 console module, replacing the L0201 module will correct most faults. If further analysis is required, it may be necessary to refer to the program listings and logic prints to determine the cause of the fault. It also may require using an oscilloscope to observe backplane signals, or having to extend the module to observe signals on the board. The following general procedure will outline the approach.

1. Restart ED0BA at location 200.

2. At the switch register prompt, type 40000 and press <RETURN> (loop on failing subtest).

3. Let several error messages be displayed and then press <CTRL/G> to get back to the SWR prompt.

4. Using the error message type out, refer to the listing to determine the details of the test sequence.

5. Relate the test sequencing information to the L0201 logic prints to determine the signals to be observed.

6. Now type 60000 and press <RETURN> to resume looping with the error type out inhibited.

7. Using signal tracing, perform the required waveform measurements to isolate the fault.

3.4 SUMMARY

ED0BA, the T-11 based console subsystem diagnostic, performs extensive testing of the console module and its immediate interfaces to the VAX CPU, local and remote ports, and the EMM. The next step in the bottom-up test procedure is to load and initialize the console software in preparation for running EDKAA, the MHC diagnostic. That's the subject of the next chapter.

CHAPTER 4
MICROHARDCORE (MHC) TESTS

4.1 OVERVIEW

EDKAA has been developed to verify proper operation of the VAX CPU hardcore logic required to successfully load and run microdiagnostics. Microdiagnostics require the following.

1. Loading microcode into the CPU's control RAMs
2. Sequencing this microcode to test the CPU logic circuits
3. Using the CBus to communicate between the console and the EBox
4. Using the SDB to retrieve signal state information

It's the task of EDKAA, a T-11 based diagnostic, to test the CPU logic that must be operational to perform these functions. Prior to running EDKAA, you should have successfully run ED0BA, the console diagnostic.

EDKAA also requires that the correct information be contained in the file ~~CONFIG~~.DAT on the load media. This file contains the names and revision levels of the SDB signal name files. Figure 4-1 shows a sample of CONFIG.DAT information.

CDF86.
CDF865.

Figure 4-2 summarizes the microhardcore context test environment. Note that the main test code is based in T-11 RAM, but some tests will require loading test microcode into the VAX CPU control stores.

CDF865.
CDF860.

```
>>>SHOW CONFIG.DAT/ASCII
! SDB Cad information files for M5 machines.
! Version: 003.000
! Released: December 21, 1985
CLKC03.CDF      !CLK C5
CSAB02.CDF      !CSA B2
CSBB02.CDF      !CSB B2
EBCC05.CDF      !EBC C5
EBDD02.CDF      !EBD D2
EBEB02.CDF      !EBE B2
EDPC02.CDF      !EDP C2
FBAB01.CDF      !FBA E7
FBMC01.CDF      !FBM C5
IBDF05.CDF      !IBD F5
ICAH02.CDF      !ICA H4
ICBF01.CDF      !ICB F5
IDPF02.CDF      !IDP F4
MAPD02.CDF      !MAP D2
MCCK01.CDF      !MCC K1
MCDD04.CDF      !MCD D4
MTMB01.CDF      !MTM B1
!VBAA01.CDF     !VBA A1 (SBI VISIBILITY MODULE #1)
!VBBA01.CDF     !VBB A1 (SBI VISIBILITY MODULE #2)
>>>
```

Figure 4-1 CONFIG.DAT File Format
CDF860.
CDF865.

4.2 OPERATING INSTRUCTIONS

To start EDKAA, type MHC at either the macro or diagnostic context prompts (>>> or DC>). Once MHC has been started, it will display the version name and number, followed by the MHC prompt (MH>), and then wait for the user to type a command. Figure 4-3 shows the typical response to the MHC command.

```
>>>MHC
                CONFIDENTIAL DIAGNOSTIC SOFTWARE
                        PROPERTY OF
                DIGITAL EQUIPMENT CORPORATION
   Use Authorized Only Pursuant to a Valid Right-to-Use License
Initializing MHC
MH>
```

Figure 4-3 MHC Command Response

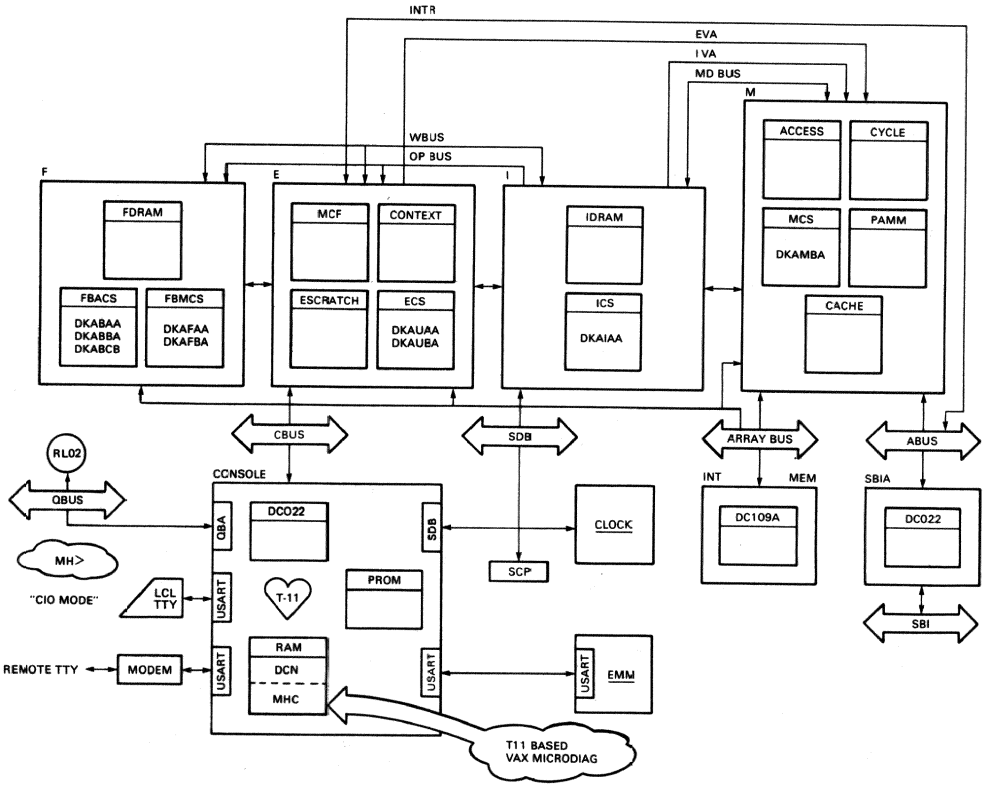


Figure 4-2 MHC Context Test Environment

4.2.1 MHC Command Summary

Once at the MH> prompt, type HELP MICRO CONTINUE to display a list of the available commands with brief descriptions, as shown in Figure 4-4.

```
MH>HELP MICRO CONTINUE
CONTINUE
    Command Syntax:
        CONTINUE
    Description:
        This command can be used to continue testing after execution was
        paused due to an error detected and the /FAULT:PAUSE switch selected.
MH>
```

Figure 4-4 HELP MICRO CONTINUE Display

The user may append one or more switches to any START or LOOP command.

To obtain a list of the switches with their descriptions, simply type the HELP MICRO START command. Figure 4-5 shows the output.

```
MH>HELP MICRO
MICRO help available:
CONTINUE                    START                       REPORT
LOOP
Select
 help available:
CONSOLE_COMMAND_SYNTAX      DIAGNOSTIC_COMMAND_SET      FRONT_PANEL
GENERAL_COMMAND_SET         HEX_DEBUG_COMMAND_SET       MACRO_COMMAND_SET
MICROHC_COMMAND_SET         REGISTERS_SUPPORTED
Select
MH>
MH>HELP MICRO START
START
    Command Syntax:
        S[TART][[Q, C, E, F, I, L, M, N, R, S, U]] [ /switches ]
        where '/switches' can be any combination of the following:
                /FAULT:[ ISOLATE, CONTINUE, LOOP, PAUSE ]
                /PRINT_MODE:[ VERBOSE, BRIEF, QUIET]
                /NUMBER:START_TEST[,END_TEST ]
                /PASS:dec_num
                where 'dec_num' specifies the number of iterations to make
                        through the entire test series.  The default of
                        one pass is made if this switch is not used.
    Description:
        The start command causes the selected test group to run for one pass,
        or for the number of passes specified by /PASS.  By default the entire
        test group is run, unless the /NUMBER switch is used to specify a range
        of tests within that group.  The test group is selected by appending
        a single letter character to the START command or to its abbreviated
        form of "S".
```

Figure 4-5 HELP MICRO START Display (Sheet 1 of 2)

```
        A brief description of what each START command does is listed below:
                S       Run all tests in all groups (approx. 20 minutes)
                SQ      Run most tests in all groups (quicker, approx. 10 min.)
                SC      Run all Clock tests
                SE      Run all Ebox tests (group R, S, and U)
                SF      Run all Fbox tests
                SI      Run all Ibox tests
                SL      Run all Last multi box access tests
                SM      Run all Mbox logic tests
                SN      Run all Mbox ucode tests
                SR      Run only the Ebox MCF-CTX RAM and basic ucode tests
                SS      Run only the Ebox SDB and control store tests
                SU      Run only the Ebox expanded ucode tests
        The switch options available on this command are listed below.
        /FAULT:[ ISOLATE, CONTINUE, LOOP, PAUSE ]               default = ISOLATE
                /FAULT:ISOLATE
                Continue after reporting a test-detected error.

                /FAULT:CONTINUE
                Continue after reporting a test-detected error.
                /FAULT:LOOP
                Loop indefinately on the next failing test.  Only ^C will
                stop this loop.
                /FAULT:PAUSE
                Pause at MH> prompt after reporting an error.  The
                CONTINUE command can be used to resume testing at the next test.
        /PRINT_MODE:[ VERBOSE, BRIEF, QUIET]                    default = VERBOSE
                /PRINT_MODE:VERBOSE
                Reports errors in long form, with RAM chip callouts.
                This cancels QUIET mode.
                /PRINT_MODE:BRIEF
                Reports errors in short form, with NO RAM chip callouts.
                This cancels QUIET mode.
                /PRINT_MODE:QUIET
                Suppresses all output except for error reports, which are
                displayed in VERBOSE form.
        /NUMBER:START_TEST[,END_TEST ]                          default = 1,LAST
                Specifies the series of test numbers to be run.  This range
                applies to the group or groups of tests being run.  Test
                numbers in MHC are in HEXadecimal format.
        /PASS:dec_num                                           default = 1
                This switch overrides the default pass count of 1.  If a range
                of tests is selected with /NUMBER then those tests will be
                repeated, in sequence, /PASS number of times.
MH>
```

Figure 4-5 HELP MICRO START Display (Sheet 2 of 2)

4.3 OPERATING PROCEDURES

The first step is to switch to microhardcore context by typing the command MHC at either the >>> or DC> prompts. The console software will load and start the required T-11 root overlay, perform the required initialization to switch to the MHC context, display program revision information, and wait at the MH> prompt for command input. While at the MH> prompt, you have access to any command in the general command set plus the additional set of MHC commands described in the previous section.

To run a complete pass of all 103 tests, type the following.

```
MH>START
```

All 103 tests will run in a predefined sequence and display performance status as each test completes. At the end of the last test, an END OF PASS message is displayed and EDKAA returns to the MH> prompt. Figure 4-6 summarizes the output to the START command. A complete pass of the START command will take approximately 20 minutes. If you want a quicker test run, the STARTQ command will run all the tests in less than 10 minutes, but perform less extensive RAM data testing. By pressing <CTRL/C>, each subtest may be exited. The program will abort when the current subtest completes, and will return to MH>.

```
MH>STARTQ
 CLK - (C-1) L0217 (CLK) SDB Shift [OK]
 CLK - (C-2) L0217 (CLK) SDB shift chain LOAD function [OK]
 CLK - (C-3) L0217 (CLK) Vis Mux Selects [OK]
 CLK - (C-4) L0217 (CLK) Vis data and LD_FUNC_REG_B [OK]
 CLK - (C-5) L0217 (CLK) Load Frequency Reg [OK]
 CLK - (C-6) L0217 (CLK) CLK2_START_H circuit [OK]
 CLK - (C-7) L0217 (CLK) CLK3 Mark Bit [OK]
 CLK - (C-8) L0217 (CLK) Burst Counter read and write [OK]
 CLK - (C-9) L0217 (CLK) CPU Clock Stop [OK]
 CLK - (C-A) L0217 (CLK) Mark Bit Stop Condition [OK]
 CLK - (C-B) L0217 (CLK) Burst Counter Ability to Count [OK]
 CLK - (C-C) L0217 (CLK) WBus and FBOX T3 to T3 shift loop [OK]
 CLK - (C-D) L0217 (CLK) WBus T2 Clk-EBE,EDP,FBA,IDP WBus Ena [OK]
 CLK - (C-E) L0217 (CLK) Stop FBOX in phase 0 and phase 1 [OK]
 E BOX - (S-1) Modules SDB Shift and Loopback [OK]
 E BOX - (S-2) L0215 (CSA) Module SDB Shift and Loopback [OK]
 E BOX - (S-3) SDB Select Line and Enable [OK]
 E BOX - (S-4) L0215 (CSA)/L0216 (CSB) SDB Control Channel [OK]
 E BOX - (S-5) L0209 (EDP) Module SDB Control Channel [OK]
 E BOX - (S-6) L0210 (EBC) Module SDB Control Channel [OK]
 E BOX - (S-7) L0219 (EBE) Module SDB Control Channel [OK]
 E BOX - (S-8) L0219 (EBE) Module ICR Time Base Counter [OK]
 E BOX - (S-9) L0210 (EBC) Module DIAG & EIS Reg CLK3 [OK]
 E BOX - (S-A) L0210 (EBC) Module MCF RAM CLK3 [OK]
 E BOX - (S-B) L0217 (CLK) CLK_141_Reset [OK]
 E BOX - (S-C) L0215 (CSA)/L0216 (CSB) ECS "55" data test [OK]
 E BOX - (S-D) L0215 (CSA)/L0216 (CSB) ECS "AA" data test [OK]
 E BOX - (R-1) L0216 (CSB) UPC+UPC SAVE Initilization [OK]
 E BOX - (R-2) L0215/L0216 (ECS) "55" 1 WORD (V$TERM) test [OK]
 E BOX - (R-3) L0215/L0216 (ECS) "AA" 1 WORD (V$TERM) test [OK]
 E BOX - (R-4) L0210 (EBC) MCF ram data test  [OK]
 E BOX - (R-5) L0210 (EBC) MCF ram address test  [OK]
 E BOX - (R-6) L0210 (EBC) Context ram data test  [OK]
 E BOX - (R-7) L0210 (EBC) Context ram address test  [OK]
 E BOX - (R-8) L0215/L0216 (ECS) RAM PARITY [OK]
 E BOX - (R-9) L0210 (EBC) MCF RAM PARITY TEST [OK]
 E BOX - (R-A) Quiescent State for UTRAP + STALL (DKAUBA) [OK]
 E BOX - (R-B) L0216 (CSB) BASIC UPC UPDATE TEST (DKAUAA)  [OK]
 E BOX - (R-C) L0216 (CSB) EXPANDED UPC UPDATE TEST (DKAUAA)  [OK]
 E BOX - (R-D) L0216 (CSB) UJUMP register TEST  (DKAUBA) [OK]
```

Figure 4-6 MHC START Command Display (Sheet 1 of 2)

```
  E BOX - (U-1) L0216 (CSB) MicroStack TEST (DKAUBA) [OK]
  E BOX - (U-2) L0209 (EDP)/L0210(EBC)WBUS REQ CTX (DKAUBA) [OK]
  E BOX - (U-3) L0209 (EDP) SHIFT COUNTER FUNCTION (DKAUBA) [OK]
  E BOX - (U-4) L0209/L0206/L0212/L0205 DATA PATH (DKAUBA) [OK]
  E BOX - (U-5) L0216 (CSB)/L0209 (EDP) Branch Cond. (DKAUBA) [OK]
  E BOX - (U-6) L0209 (EDP) ALU FUNCTION (DKAUBA) [OK]
  E BOX - (U-7) L0209 (EDP) SCRATCH PADS DATA (DKAUBA) [OK]
  E BOX - (U-8) L0219(EBE)/L0201(CSL) CBUS/CSL-INT (DKAUBA) [OK]
  E BOX - (U-9) L0219(EBE)/L0201(CSL) EBCS REGISTER (DKAUBA) [OK]
  I BOX - (I-1) SDB shift and loopback [OK]
  I BOX - (I-2) SDB Select Line and Enable [OK]
  I BOX - (I-3) L0208 (IBD) Module SDB Control Channel [OK]
  I BOX - (I-4) L0207 (ICA) Module SDB Control Channel [OK]
  I BOX - (I-5) L0217 (CLK) CLK_141_Reset [OK]
  I BOX - (I-6) L0207 (ICA) ICS UPC register test [OK]
  I BOX - (I-7) L0207 (ICA) ICS "55" data test [OK]
  I BOX - (I-8) L0207 (ICA) ICS "AA" data test [OK]
  I BOX - (I-9) L0207 (ICA) ICS address test  [OK]
  I BOX - (I-A) L0208 (IBD) IDRAM "55" data test [OK]
  I BOX - (I-B) L0208 (IBD) IDRAM "AA" data test [OK]
  I BOX - (I-C) L0208 (IBD) IDRAM address test [OK]
  I BOX - (I-D) L0207 (ICA) ICS Ram Parity test [OK]
  I BOX - (I-E) Quiescent State for I BOX control signals  [OK]
  I BOX - (I-F) L0207 (ICA) UJUMP Micro-code TEST (DKAIAA)  [OK]
  M BOX - (M-1) SDB shift and loopback [OK]
  M BOX - (M-2) L0222 (MTM) SDB shift and loopback [OK]
  M BOX - (M-3) SDB Select Line and Enable [OK]
  M BOX - (M-4) L0217 (CLK) CLK_141_Reset [OK]
  M BOX - (M-5) L0220 (MCC) MCS "55" data test [OK]
  M BOX - (M-6) L0220 (MCC) MCS "AA" data test [OK]
  M BOX - (M-7) L0220 (MCC) Cycle ram "55" data test [OK]
  M BOX - (M-8) L0220 (MCC) Cycle ram "AA" data test [OK]
  M BOX - (M-9) L0220 (MCC) ACCESS ram  data test   [OK]
  M BOX - (M-A) L0220 (MCC) Cycle ram address test  [OK]
  M BOX - (M-B) L0220 (MCC) MCS ram address test   [OK]
  M BOX - (M-C) L0220 (MCC) Access ram address test   [OK]
  M BOX - (N-1) L0220 (MCC) BASIC UCODE FUNCTION (DKAMAA)  [OK]
  M BOX - (N-2) L0220 (MCC) BASIC STACK UCODE (DKAMBA)  [OK]
  M BOX - (N-3) L0220 (MCC) STACK PUSH LEVEL (DKAMCA)  [OK]
  M BOX - (N-4) L0220 (MCC) STACK POP LEVEL (DKAMDA)  [OK]
  F BOX - (F-1) L0213 (FBM) F BOX present V$TERM [OK]
  F BOX - (F-2) L0212 (FBA) SDB shift and loopback [OK]
  F BOX - (F-3) L0213 (FBM) SDB shift and loopback [OK]
  F BOX - (F-4) F Box Modules SDB Select Line and Enable [OK]
  F BOX - (F-5) L0217 (CLK) CLK_141_Reset [OK]
  F BOX - (F-6) L0212 (FBA) UPC register tests  [OK]
  F BOX - (F-7) L0213 (FBM) UPC register tests  [OK]
  F BOX - (F-8) L0212 (FBA) Control Store "55" data test [OK]
  F BOX - (F-9) L0212 (FBA) Control Store "AA" data test [OK]
  F BOX - (F-A) L0213 (FBM) Control Store "55" data test [OK]
  F BOX - (F-B) L0213 (FBM) Control Store "AA" data test [OK]
  F BOX - (F-C) L0213 (FBM) Control Store address test  [OK]
  F BOX - (F-D) L0212 (FBA) Control Store address test  [OK]
  F BOX - (F-E) L0213 (FBM) Decode ram "55" data test  [OK]
  F BOX - (F-F) L0213 (FBM) Decode ram "AA" data test  [OK]
  F BOX - (F-10) L0213 (FBM) Decode address test    [OK]
  F BOX - (F-11) L0213 (FBM) Basic Micro-code test (DKAFAA) [OK]
  F BOX - (F-12) L0212 (FBA) Basic Micro-code test (DKABAA)  [OK]
  F BOX - (F-13) L0213 (FBM) Expanded UPC update (DKAFBA)  [OK]
  F BOX - (F-14) L0212 (FBA) Expanded UPC update (DKABBA)  [OK]
  F BOX - (F-15) L0212 (FBA) Micro-code test (DKABCB)  [OK]
 VAX-8600 - (L-1) CLK and E,I,M Boxes uWord Mark Bit [OK]
 VAX-8600 - (L-2) VTERM MASTER RESET TEST [OK]
PASS COUNTER = 00001, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
MH>
```

Figure 4-6 MHC START Command Display (Sheet 2 of 2)

To suppress all output except errors and END OF PASS, you can use the /MODE:QUIET switch appended to the START command. This switch is especially useful if you're running EDKAA via the RD port over a 1200 baud line, or only have a hard-copy local console device.

Table 4-1 illustrates how several commands and the available switches are used to modify program operation.

These examples should show how to modify program operation using commands and switches. Ideally, the user should find a VAX 8600/8650 system and experiment with other combinations. Remember, any time the program pauses on error, you have access to the general command set to retrieve additional information.

Table 4-1 Illustrative MCH Commands

Command	Description
SE	Run one pass of all the EBox tests
SC/PASS:5	Run five passes of all the clock tests
LM/NU:5 6/MODE:QUIET	Loop continuously on the MBox control store RAM data tests
SM/FAULT:PAUSE	Run one pass of all the MBox tests, but pause and return to the MH> prompt when an error is detected
CONTINUE	Resume testing after a pause on error
REPORT	Display current program statistics (passes, error count, etc.)

4.4 PROGRAM FUNCTIONAL DESCRIPTION

This section describes the structure and organization of EDKAA and includes examples of program output and brief descriptions of each test. If you need additional detail, refer to the document EDKAA.DOC in the microfiche library. Prepared by the designer, it is a comprehensive description of the program's purpose, structure, and operation.

4.4.1 Program Overview

EDKAA consists of 103 subtests that functionally test the VAX CPU. These tests have been subdivided into nine CPU box-related groups. The operator may select to execute one or all of these group tests. The groups reside on the RL02 load media and are only loaded into memory when selected by the operator. The program only reports messages to the operator console and the remote port. It does not create or modify any files on the RL02 load device. EDKAA does not require that any special initialization files (i.e., DCLOAD.COM, DSM.INI, or LOAD.COM) be executed before starting.

There is one root section and nine disk resident overlays. Three overlays are for the EBox testing. There is one overlay for each FBox, IBox, and clock box. The overlay for the MBox has been divided, the additional overlay providing additional microcode coverage for the MBox. The files for EDKAA are mapped into and called by the root console software segment. EDKAA uses 12 additional files that contain microcode information which needs to be loaded into the CPU control stores for testing the microsequencers. These files all have the .BPN extension and are listed in Table 4-2.

Table 4-2 MHC Microcode Files

File name	Size	Reason
DKABAA.BPN	1	FBA C/S file - load #1
DKABBA.BPN	1	FBA C/S file - load #2
DKABCA.BPN	1	FBA C/S file - load #3
DKAFAA.BPN	2	FBM C/S file - load #1
DKAFBA.BPN	1	FBM C/S file - load #2
DKAIAA.BPN	7	IBox C/S file - load #1
DKAMAA.BPN	9	MBox C/S file - load #1
DKAMBA.BPN	9	MBox C/S file - load #2
DKAMCA.BPN	9	MBox C/S file - load #3
DKAMDA.BPN	9	MBox C/S file - load #4
DKAUAA.BPN	4	EBox C/S file - load #1
DKAUBA.BPN	37	EBox C/S file - load #2

The root section (MHCMHC) of EDKAA is resident in low memory. The other nine sections are resident on the console disk (RL02) and are loaded into an overlay area in low memory when needed. For the current version of the console, this overlay area starts at location 100000(8). When the operator selects the IBox test, for example, the IBox overlay section is loaded by the root section and control is then transfered to the program code in the overlay. Upon completion of the IBox subtests, control is returned to the root section. Table 4-3 lists the different overlay listing names, section names, segment numbers, base addresses, and functions.

4.4.2 Fault Detection and Error Reporting

When an error is detected, there are three general types of error displays that may be output. The major difference between them is the amount of detail they provide. In early versions of the program, the user may only have the brief format to work with and will need to refer to the EDKAA.DOC to extract the additional fault isolation information required. As the program matures, there will be fewer and fewer cases where the user will have to do this.

Table 4-3 EDKAA T-11 Program Modules

Listing Names	Section Names	Segment Number	Base Address	Function
MHCMHC	MHCCOD	4	062450	Root controller and handlers
MHCEBA	EBACOD	14	100002	EBox MCF, CTX + logic tests
MHCEBB	EBBCOD	15	100002	EBox microcode tests
MHCEBC	EBCCOD	16	100002	EBox SDB and C/S tests
MHCFBX	FBXCOD	17	100002	FBox logic, RAM + Ucode tests
MHCIBX	IBXCOD	20	100002	IBox logic, RAM + Ucode tests
MHCMBA	MBACOD	21	100002	MBox logic, RAM tests
MHCMBB	MBBCOD	22	100002	MBox Ucode tests
MHCCLK	CLKCOD	23	100002	Clock logic tests
MHCLMT	LMTCOD	24	100002	Logical Multi-box Tests

Figure 4-7 shows an example of the brief error display. Note that it was the EBox UTRAP/STALL logic test that failed, the failure was detected in STAGE 01, and there is a 1-bit difference between the expected and actual data. What does this all mean? Figure 4-8 shows the information you would find in EDKAA.DOC that describes the failing test and how to interpret the expected/actual data. It looks like it detected a result parity error and most likely is the EBox module(s) containing the UTRAP/STALL logic.

```
MHC>STARTE
        .
        .
        .
        .
        .
        .

E BOX - CONTEXT RAM ADDRESS TEST [OK]

E BOX - CONTROL STORE RAM PARITY [OK]
        .
        .
E BOX - MCF RAM PARITY TEST [OK]
STAGE 01, EXPECTED 263F, ACTUAL 223F
STAGE 01, EXPECTED 263F, ACTUAL 223F
STAGE 01, EXPECTED 263F, ACTUAL 223F
STAGE 01, EXPECTED 263F, ACTUAL 223F
STAGE 01, EXPECTED 263F, ACTUAL 223F
STAGE 01, EXPECTED 263F, ACTUAL 223F
STAGE 01, EXPECTED 263F, ACTUAL 223F
ERROR **** E BOX - QUIESCENT STATE FOR UTRAP AND STALL TEST ****
```

Figure 4-7 MHC Brief Error Display

```
        E BOX MICRO CODE TEST PROCEDURE
        -------------------------------

        This defines the procedure that was followed to test the EBOX
        hardware that is covered by the Micro-Hard-Core diagnostic.


        MICRO SEQUENCER TESTS
        ---------------------

         E BOX - (R-A) QUIESCENT STATE FOR UTRAP + STALL (DKAUBA)

      SETUP:

        Requires Functional Microcode be loaded. (DKAUBA.BPN)

      TEST PROCEDURE:

        o Execute Master Reset Sequence.
        o Load starting address of "QUIESCENT STATE" microcode [0050]H.
        o Set sdb control bits as follows:

                1.Consol op1 = 0
                2.Consol op0 = 0

REPEAT> o Burst system clock 1 cycle to execute first Microinstruction.
          of UCODE routine.
        o Read the following 63. SDB signals in the sequence shown.

STAGE1>         EBD EN ETRAP H = 0                  <MSB>
EXPECT=263F     EBD EN ETRAP DLY L = 1
                EDP WBUS ERROR DLY H = 0

                EDP OPR PAR ERR H = 0
                -EDP RESULT PAR ERR H = 1
                -EBC MCF RAM PAR ERR H = 1
                CSB USTK PAR ERR H = 0

                CSB CSPE RESET A H = 0
                EBD EB PORT STAT 3 C H = 0
                EBC MCF-MEM REQ LTH L = 1
                -EBD9 MEM REQ IN PROG H =  1

                EBC MCF-OP WRT LTH L = 1
                -EBD9 OP WRT LST CYC H = 1
                -EBD9 OP WRT IN PROG H = 1
                EBC MCF-EN OWSTL LTH L = 1      <LSB>
```

Figure 4-8 MHC Test Description from EDKAA.DOC

Figure 4-9 shows an example of the second type of message. Note how the command uses the /NUMBER:C,C switch to select only test C and calls for verbose printout with /MODE:VERBOSE with abbreviations (/N:C,C/M:V). This display provides additional information that calls out the module (L0215) or even the RAM chip (E140) that's most likely the culprit. It also includes the RAM address along with the good/bad data patterns. Figure 4-10 shows a third example that includes signal callout information (Paragraph 4.4.4).

```
MH>SS/N:C,C/M:V

 E BOX - (S-C) L0215 (CSA)/L0216 (CSB) ECS "55" data test

SECTION   SLOT  COMPONENT    MODULE          TERM-MOD   SYMBOL    SIGNAL
CPU         3      E140    L0215 (CSA)                          CSA USRC2 5
CPU         3      E140    L0215 (CSA)                          CSA USRC2 3
ADDR =0004 GOOD = 055555555555D55 BAD = 050555555555D55
```

Figure 4-9 Sample Error Display 2

```
MH>REPORT

          SECTION, TOTAL # OF ERRORS
          --------------------------
          CLOCK        (SC) = 00000
          E SDB/CS(SS) = 00000
          E RAMS   (SR) = 00000
          E UCODE (SU) = 00009
          I BOX    (SI) = 00000
          M BOX    (SM) = 00000
          F BOX    (SF) = 00000
          VENUS    (SL) = 00000
 PASS COUNTER = 00001, # OF ERRORS = 00009, TOTAL # OF ERRORS = 00009
MH>
```

Figure 4-10 MHC REPORT Display

4.4.3 Test Descriptions

Tables 4-4 through 4-12 list all of the tests organized by the functions being tested. The string enclosed in parentheses in the table title is the command that invokes that series of tests. If the MODE: switch is set to turn QUIET off, the test descriptions shown in the tables will be displayed as test headers.

Table 4-4 Clock Box Subtests (SC)

```
CLK -- (C-1) L0217 (CLK) SDB SHIFT
CLK -- (C-2) L0217 (CLK) SDB SHIFT CHAIN LOAD FUNCTION
CLK -- (C-3) L0217 (CLK) VIS MUX SELECTS
CLK -- (C-4) L0217 (CLK) VIS DATA AND LD_FUNC_REG_B
CLK -- (C-5) L0217 (CLK) LOAD FREQUENCY REG
CLK -- (C-6) L0217 (CLK) CLK2_START_H CIRCUIT
CLK -- (C-7) L0217 (CLK) CLK3 MARK BIT
CLK -- (C-8) L0217 (CLK) BURST COUNTER READ AND WRITE
CLK -- (C-9) L0217 (CLK) CPU CLOCK STOP
CLK -- (C-A) L0217 (CLK) MARK BIT STOP CONDITION
CLK -- (C-B) L0217 (CLK) BURST COUNTER ABILITY TO COUNT
CLK -- (C-C) L0217 (CLK) WBUS AND FBOX T3 TO T3 SHIFT LOOP
CLK -- (C-D) L0217 (CLK) WBUS T2 CLK TO EBE,EDP,FBA,IDP WBUS
      ENABLE
CLK -- (C-E) L0217 (CLK) STOP FBOX IN PHASE 0 AND PHASE 1
```

Table 4-5 EBox SDB and C/S Subtests (SS)

```
E BOX -- (S-1) MODULES SDB SHIFT AND LOOPBACK
E BOX -- (S-2) L0215 (CSA) MODULE SDB SHIFT AND LOOPBACK
E BOX -- (S-3) E BOX MODULES SDB SELECT LINE AND ENABLE
E BOX -- (S-4) L0215 (CSA) and L0216 (CSB) SDB CONTROL CHANNEL
E BOX -- (S-5) L0209 (EDP) MODULE SDB CONTROL CHANNEL
E BOX -- (S-6) L0210 (EBC) MODULE SDB CONTROL CHANNEL
E BOX -- (S-7) L0219 (EBE) MODULE SDB CONTROL CHANNEL
E BOX -- (S-8) L0219 (EBE) MODULE ICR TIME BASE COUNTER
E BOX -- (S-9) L0210 (EBC) MODULE DIAG & EIS REG CLK3
E BOX -- (S-A) L0210 (EBC) MODULE MCF RAM CLK3
E BOX -- (S-B) L0217 (CLOCK) CLK 141 RESET
E BOX -- (S-C) L0215 (CSA) and L0216 (CSB) ECS "55" DATA TEST
E BOX -- (S-D) L0215 (CSA) and L0216 (CSB) ECS "AA" DATA TEST
```

Table 4-6 MCF+CTX+ Misc. EBox Subtests (SR)

```
E BOX -- (R-1) L0216 (CSB) UPC+UPC SAVE INITILIZATION
E BOX -- (R-2) L0215/L0216 (ECS) "55" 1 WORD (V$TERM) TEST
E BOX -- (R-3) L0215/L0216 (ECS) "AA" 1 WORD (V$TERM) TEST
E BOX -- (R-4) L0210 (EBC) MCF RAM DATA TEST
E BOX -- (R-5) L0210 (EBC) MCF RAM ADDRESS TEST
E BOX -- (R-6) L0210 (EBC) CONTEXT RAM DATA TEST
E BOX -- (R-7) L0210 (EBC) CONTEXT RAM ADDRESS TEST
E BOX -- (R-8) L0215/L0216 (ECS) RAM PARITY
E BOX -- (R-9) L0210 (EBC) MCF RAM PARITY TEST
E BOX -- (R-A) QUIESCENT STATE FOR UTRAP + STALL (DKAUBA)
E BOX -- (R-B) L0216 (CSB) BASIC UPC UPDATE TEST (DKAUAA)
E BOX -- (R-C) L0216 (CSB) EXPANDED UPC UPDATE TEST (DKAUAA)
E BOX -- (R-D) L0216 (CSB) UJUMP REGISTER TEST (DKAUBA)
```

Table 4-7 EBox Ucode Subtests (SU)

```
E BOX -- (U-1) L0216 (CSB) MICROSTACK TEST (DKAUBA)
E BOX -- (U-2) L0209 (EDP) +L210 (EBC) WBUS REQUEST AND CT (DKAUBA)
E BOX -- (U-3) L0209 (EDP) SHIFT COUNTER FUNCTION (DKAUBA)
E BOX -- (U-4) L0209 (EDP) BASIC DATA PATH FUNCTION (DKAUBA)
E BOX -- (U-5) L0216 (CSB) OR L0209 (EDP) BRANCH CONDITION (DKAUBA)
E BOX -- (U-6) L0209 (EDP) ALU FUNCTION (DKAUBA)
E BOX -- (U-7) L0209 (EDP) SCRATCH PADS DATA (DKAUBA)
E BOX -- (U-8) L0219 (EBE) CBUS AND CSL-INT (DKAUBA)
E BOX -- (U-9) L0219 (EBE) EBCS REGISTER <31:27> (DKAUBA)
```

Table 4-8 IBox Subtests (SI)

```
I BOX -- (I-1) SDB SHIFT AND LOOPBACK
I BOX -- (I-2) SDB SELECT LINE AND ENABLE
I BOX -- (I-3) L0208 (IBD) MODULE SDB CONTROL CHANNEL
I BOX -- (I-4) L0207 (ICA) MODULE SDB CONTROL CHANNEL
I BOX -- (I-5) L0217 (CLOCK) CLK 141 RESET
I BOX -- (I-6) L0207 (ICA) ICS UPC REGISTER TEST
I BOX -- (I-7) L0207 (ICA) ICS "55" DATA TEST
I BOX -- (I-8) L0207 (ICA) ICS "AA" DATA TEST
I BOX -- (I-9) L0207 (ICA) ICS ADDRESS TEST
I BOX -- (I-A) L0208 (IBD) IDRAM "55" DATA TEST
I BOX -- (I-B) L0208 (IBD) IDRAM "AA" DATA TEST
I BOX -- (I-C) L0208 (IBD) IDRAM ADDRESS TEST
I BOX -- (I-D) L0207 (ICA) ICS RAM PARITY TEST
I BOX -- (I-E) QUIESCENT STATE FOR I BOX CONTROL SIGNALS
I BOX -- (I-F) L0207 (ICA) UJUMP MICRO-CODE TEST (DKAIAA)
```

Table 4-9 MBox Logic Subtests (SM)

```
M BOX -- (M-1) SDB SHIFT AND LOOPBACK
M BOX -- (M-2) MTM SDB SHIFT AND LOOPBACK
M BOX -- (M-3) SDB SELECT LINE AND ENABLE
M BOX -- (M-4) L0217 (CLK) CLK 141 RESET
M BOX -- (M-5) L0220 (MCC) MCS "55" DATA TEST
M BOX -- (M-6) L0220 (MCC) MCS "AA" DATA TEST
M BOX -- (M-7) L0220 (MCC) CYCLE RAM "55" DATA TEST
M BOX -- (M-8) L0220 (MCC) CYCLE RAM "AA" DATA TEST
M BOX -- (M-9) L0220 (MCC) ACCESS RAM DATA TEST
M BOX -- (M-A) L0220 (MCC) CYCLE RAM ADDRESS TEST
M BOX -- (M-B) L0220 (MCC) MCS RAM ADDRESS TEST
M BOX -- (M-C) L0220 (MCC) ACCESS RAM ADDRESS TEST
```

Table 4-10 MBox Ucode Subtests (SN)

```
M BOX -- (N-1) L0220 (MCC) BASIC UCODE FUNCTION (DKAMAA)
M BOX -- (N-2) L0220 (MCC) BASIC STACK UCODE (DKAMBA)
M BOX -- (N-3) L0220 (MCC) STACK PUSH LEVEL (DKAMCA)
M BOX -- (N-4) L0220 (MCC) STACK POP LEVEL (DKAMDA)
```

Table 4-11 FBox Subtest (SF)

```
F BOX -- (F-1) Test F BOX Present V$TERM VALUES
F BOX -- (F-2) L0212 (FBA) SDB SHIFT AND LOOPBACK
F BOX -- (F-3) L0213 (FBM) SDB SHIFT AND LOOPBACK
F BOX -- (F-4) F BOX MODULES SDB SELECT LINE AND ENABLE
F BOX -- (F-5) L0217 (CLOCK) CLK 141 RESET
F BOX -- (F-6) L0212 (FBA) UPC REGISTER TESTS
F BOX -- (F-7) L0213 (FBM) UPC REGISTER TESTS
F BOX -- (F-8) L0212 (FBA) CONTROL STORE "55" DATA TEST
F BOX -- (F-9) L0212 (FBA) CONTROL STORE "AA" DATA TEST
F BOX -- (F-A) L0213 (FBM) CONTROL STORE "55" DATA TEST
F BOX -- (F-B) L0213 (FBM) CONTROL STORE "AA" DATA TEST
F BOX -- (F-C) L0213 (FBM) CONTROL STORE ADDRESS TEST
F BOX -- (F-D) L0212 (FBA) CONTROL STORE ADDRESS TEST
F BOX -- (F-E) L0213 (FBM) DECODE RAM "55" DATA TEST
F BOX -- (F-F) L0213 (FBM) DECODE RAM "AA" DATA TEST
F BOX -- (F-10) L0213 (FBM) DECODE ADDRESS TEST
F BOX -- (F-11) L0213 (FBM) BASIC MICRO-CODE TEST (DKAFAA)
F BOX -- (F-12) L0212 (FBA) BASIC MICRO-CODE TEST (DKABAA)
F BOX -- (F-13) L0213 (FBM) EXPANDED UPC UPDATE TEST (DKAFBA)
F BOX -- (F-14) L0212 (FBA) EXPANDED UPC UPDATE TEST (DKABBA)
F BOX -- (F-15) L0212 (FBA) MICRO-CODE TEST (DKABCB)
```

Table 4-12 Last Box Subtest (SL)

```
VENUS -- (L-1) CLK and E,I,M boxes Uword MARK BIT
VENUS -- (L-2) V$TERM MASTER RESET VISIBILITY TEST
```

4.4.4 End of Pass Report

Figure 4-10 shows an example of the End of Pass report when an error occurred. The user can display these statistics any time by typing the REPORT command.

Note that on this run, nine EBox errors were detected. The error messages accompanying each error most likely indicated which module(s) to replace. If no errors were detected, the REPORT command would have displayed the last line shown in Figure 4-10.

4.5 SUMMARY

EDKAA, a T-11 based diagnostic program, tests the logic needed to run VAX CPU microdiagnostics. During bottom-up testing, it is run after ED0BA and before EDKBA, the EBox microdiagnostic. EDKAA provides fault isolation messages that identify the module(s) or functional logic area that may be the cause of hardware problems in the CPU. The next step in the bottom-up test procedure is to run the VAX CPU microdiagnostics, as described in the next chapter.

CHAPTER 5
MICRODIAGNOSTICS

5.1 OVERVIEW

This chapter describes how to use the microdiagnostics to test and troubleshoot logic faults in the VAX 8600/8650 CPU, internal memory system, and the SBIA interfaces to the ABus. It will cover the following.

- The purpose and operation of microdiagnostics for the system
- What microdiagnostics are available and how they should be sequenced
- The procedures for loading, starting, and controlling the execution of the microdiagnostics
- The format of the fault isolation messages displayed when a fault is detected
- The use of software switches to modify the operation of the microdiagnostics
- The use of console commands to retrieve supplementary fault information to that provided by the microdiagnostics themselves

A more detailed description of the fault isolation process is provided in Appendix H.

5.2 MICRODIAGNOSTIC OPERATION

At this point, you should read Appendix H to get an overview of how fault isolation and microdiagnostics were designed and implemented for the VAX systems. This section will describe how microdiagnostics operate in the VAX 8600/8650 systems.

5.2.1 Purpose and Basic Operation

First, microdiagnostics are nonfunctional test microprograms that reside in the EBox control store. They are designed to provide stimulus/response testing of individual logic circuits within the VAX CPU. For each stimulus, the response is checked to determine if the hardware is operating properly. When a fault is detected, the microdiagnostic leaves the test results in predefined EBox scratchpad locations and signals the Diagnostic Control Program (DCP), a T-11 based program, that a fault was detected.

DCP retrieves the fault information (ERROR SYNDROME) from the EBox scratchpad and formats and displays it on the console terminal. If the fault is "solid," DCP invokes its fault isolation process to analyze the error information, and to append module and component call-out messages to the error display.

In short, the EBox-based microcode provides the actual testing and fault detection, while the T-11 based console software provides the error information retrieval, analysis, and reporting mechanism. The T-11 based software also provides the user interface for loading, starting, stopping, and modifying the operation of the microdiagnostics.

5.2.2 Diagnostic Context Test Environment

To begin running microdiagnostics, you must switch to the diagnostic context mode of operation (see Chapter 1) by typing the DIAGNOSE command. Figure 5-1 summarizes the system environment that exists when running the EBox microdiagnostic EDKBA.

Diagnostic context initialization is performed whenever the DIAGNOSE command is entered, whether or not diagnostic context is already running. This initialization creates a state where microdiagnostics can be loaded, executed, and monitored. The steps taken by this initialization procedure are as follows.

1. Takes control of the TXCS RDY interrupt vector.

2. Submits the DCP initialization file, DCLOAD.COM, internally in order to perform the following steps.

   - Stops the CPU clock and perform a master reset.
   - Loads the Diagnostic Support Microcode (DSM) into the EBox control store.
   - Loads other control stores, as necessary, to initialize parity logic, etc.
   - Enables the console's external logic interrupt, using the SET EXTI ON command.
   - Forces the EBox to begin executing at the DSM start address and start the CPU clocks.

3. Passes the default diagnostic switch settings to DSM (via the CBus).

4. Displays the diagnostic context command prompt (DC>) and await commands.

The following examples illustrate the console software's response when the user types the DIAGNOSE command. Figure 5-2 shows the default response with the QUIET switch on. Figure 5-3 shows the response with the QUIET switch off to illustrate the commands executed out of the command file DCLOAD.COM.

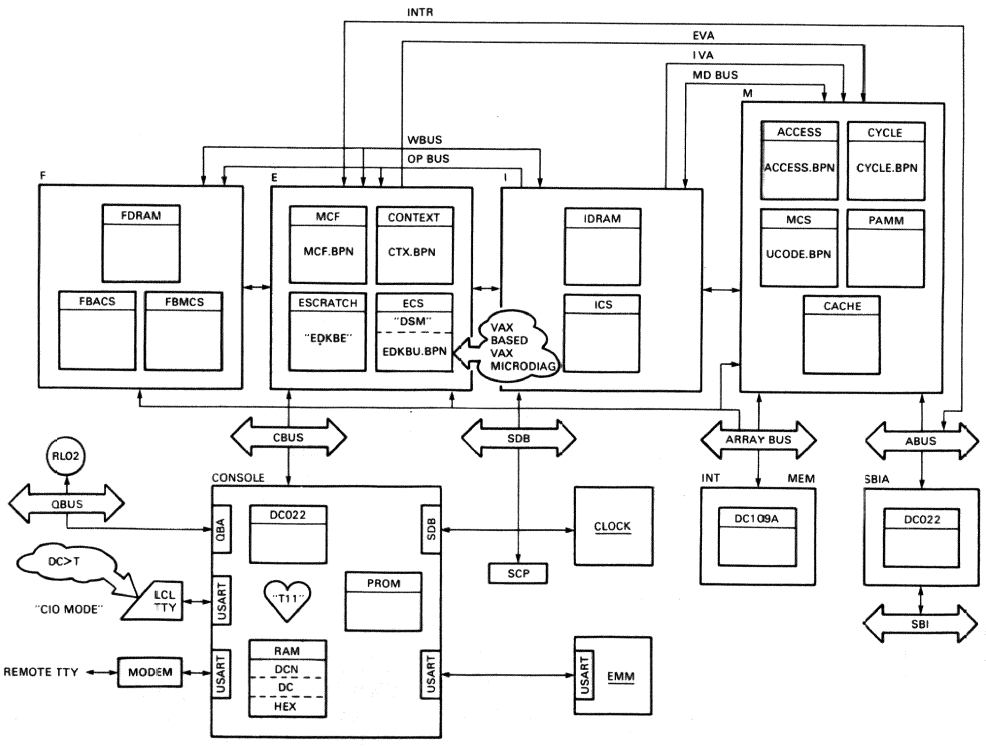


Figure 5-1 Diagnostic Context Test Environment

Note that DCLOAD.COM contains a DEBUG command that leaves the HEX command set enabled when it terminates, as evidenced by the DC>> prompt.

```
MH>DIAGNOSE
                CONFIDENTIAL DIAGNOSTIC SOFTWARE
                        PROPERTY OF
              DIGITAL EQUIPMENT CORPORATION
   Use Authorized Only Pursuant to a Valid Right-to-Use License
Initializing DC
DC>>
```

Figure 5-2 Diagnostic Context Initialization (QUIET:ON)

```
DC>>SET QUIET OFF
DC>>DIAGNOSE
                CONFIDENTIAL DIAGNOSTIC SOFTWARE
                        PROPERTY OF
              DIGITAL EQUIPMENT CORPORATION
   Use Authorized Only Pursuant to a Valid Right-to-Use License
Initializing DC
DC>! DCLOAD.COM - Initialization file for DIAG context
DC>! Version: 001.000
DC>! Released: 14-Dec-1984
DC>!
DC>SET DEFAULT/BELL:OFF          !Suppress the bell upon fault detection
DC>SET DEFAULT/TRACE:OFF         !Suppress single test trace of udiags
DC>SET DEFAULT/FAULT:ISOLATE     !Isolate when fault is solid
DC>SET DEFAULT/MODE:VERBOSE      !Print a complete fault report
DC>SET DEFAULT/LINE:OFF          !Suppress trace in isolation files
DC>SET DEFAULT/PASS:100          !Each test is executes 100 times
DC>SET DEFAULT/NUM:1,FF          !First_test = 1 , Last_test = 256
DC>DEBUG                         !Enable full set of commands
DC>>SET MEMENA OFF               !Keep Array OFF
DC>>SET ABUS ON                  !ABUS INIT True**
DC>>SET ABUS OFF                 !ABUS INIT False**
DC>>INIT/CLOCK                   !Clock circuitry init
DC>>SET CLOCK FREQ QUIET         !TEMPORARILY NEEDED TO HANDLE OLD (C04)
                                 !CLK MODULE
DC>>RESET                        !Reset CPU..Does little without any Ucode
DC>>LOAD/MCF                     !MCF rams must Always be loaded
DC>>DEP/ECS 0 0                  !Good Parity into EBOX Reset location
DC>>LOAD/CONTEXT                 !Context Rams
DC>>LOAD/ICS                     !Nop Micro Word into IBOX control store
DC>>LOAD/MCS                     !MBOX control store to avoid fatal parity
DC>>LOAD/ACCESS                  !Access control RAM
DC>>LOAD/CYCLE                   !Cycle Parameter Rams
DC>>INIT/SDB                     !All SDB channels to nominal state
DC>>SET MEMENA ON                !Enable Array as part of the system
DC>>RESET
DC>>LOAD/ECS DSM                 !Diagnostic Support Microcode
DC>>RESET
DC>>START CPU                    !Need CPU clock to do anything useful
DC>>DEPOSIT/WBUS 12 0            !Disable Fbox, preserve wbus integrity
DC>>
```

Figure 5-3 Diagnostic Context Initialization (QUIET:OFF)

5.3 MICRODIAGNOSTIC SET

With the introduction of the VAX 8650 system, several microdiagnostic load filenames (i.e., .BPN files) were revised to reflect the new system requirements. 8650-specific load files can be identified by an E as the second character in the diagnostic's name. For example, the MBox diagnostic (EDKDA) loads the EDKCM.BPN file for the VAX 8600 system and loads the EEKCM.BPN file for the VAX 8650 system.

In addition, all files shared by the VAX 8600 and VAX 8650 systems will retain their original names. Only 8650-specific diagnostics use the E character. There are three file types.

1. Shared 8600/8650 files with the D character

2. 8600-secific files with the D character

3. 8650-specific files with the E character

Note that the 8600 and 8650 command scripts have the same names.

Table 5-1 lists the set of 24 microdiagnostic programs resident on the RL02 disk, grouped into 6 functional series.

Table 5-1 VAX 8600/8650 Microdiagnostic Sets

Description	Name	No. of Programs
EBox Series	EDKBA	1
MBox Series	EDKCA -- EDKHA	6
IBox Series	EDKOA -- EDKWA	9
FBox Series	EDK1A -- EDK4A,EDKZA	5
ARRAY Series	EDK5A	1
SBIA Series	EDK6A -- EDK7A	2

The programs were designed using a building block approach and should normally be run in the sequence shown in Figure 5-4. Each successive program in the series uses logic tested by the previous programs. For example, the IBox series uses the data cache which is tested by the preceding MBox series, while the MBox series uses the CPU logic tested by the EBox series.

All 24 programs may be loaded and run automatically in the proper sequence via an indirect command file named MICROS.COM. Figure 5-5 shows an example of this process.

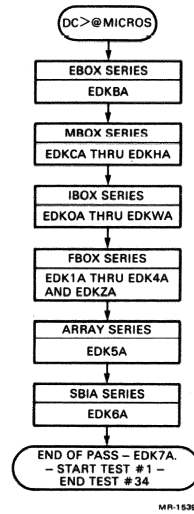


Figure 5-4 Microdiagnostic Sequencing

Before moving on to the actual operating procedures, it is important to understand the purpose of certain control and data files on the RL02 that are used to support each microdiagnostic program. In general, there are three types of files used.

1. xxxxx.COM -- Command files used for setup and initialization

2. xxxxx.BPN -- Specially formatted binary files that contain the data loaded into the CPU's control RAMs

3. xxxxx.DCI, DCB -- Specially formatted files that contain the coarse and fine isolation algorithms used by DCP to analyze fault data

```
DC>>SHOW MICROS.COM/ASCII
!General Command file for Running ALL microddiagnostics for VAX 8600 cpu
! Version: 004.000
! Released: 17-JUN-1985
!Command File for executing the entire string of Microdiagnostics delivered
! at Engineering Venus release:
!*****************************************************************************
!                M 5
!*****************************************************************************
!       *Revised December 11, 1984
!       *Revised February 20, 1985
!       *Revised March 22, 1985
!       *Revised June 12, 1985
!
!**Note: this command file does Not run the MicroHardCore, which is run from
! within the console software subsystem.
!
!Ebox Microdiagnostic:  Includes Coarse Component Isolation
Run/pass:10 EDKBA
!Mbox Microdiagnostic - MCF field decoding
Run/pass:100 EDKCA
!Mbox Microdiagnostic
Run/pass:3 EDKDA
!Mbox Microdiagnostic
Run/pass:100 EDKEA
!Mbox Microdiagnostic
Run/pass:100 EDKFA
!Mbox Microdiagnostic
Run/pass:100 EDKGA
!Mbox Microdiagnostic
Run/pass:100 EDKHA
!Ibox Microdiagnostic
Run/pass:100 EDKOA
!Ibox Microdiagnostic
Run/pass:100 EDKPA
!Ibox Microdiagnostic
Run/pass:100 EDKQA
!Ibox Microdiagnostic
Run/pass:100 EDKRA
!Ibox Microdiagnostic
Run/pass:100 EDKSA
!Ibox Microdiagnostic
Run/pass:100 EDKTA
!Ibox Microdiagnostic
Run/pass:100 EDKUA
!Ibox Microdiagnostic
Run/pass:100 EDKVA
!Ibox Microdiagnostic
Run/pass:100 EDKWA
!Includes component Isolation
!Fbox Microdiagnostic - Adder Module part I
Run/pass:100 EDK1A
!Includes component Isolation
!Fbox Microdiagnostic - Adder Module part II
Run/pass:100 EDK2A
!Includes component Isolation
!Fbox Microdiagnostic - Multiplier Module
Run/pass:100 EDK3A
!Includes component Isolation
!Fbox Microdiagnostic - System Interaction Test - part 1
Run/pass:100 EDK4A
!Includes component Isolation
!Fbox Microdiagnostic - System Interaction Test - part 2
```

Figure 5-5 MICROS.COM Command File (Sheet 1 of 2)

```
Run/pass:100 EDKZA
!Array Microdiagnostic
Run/pass:3 EDK5A
!SBIA Microdiagnostic - Part 1.  Tests either or both SBIAs
Run/pass:50 EDK6A
!*** Note  ***
!SBIA Microdiagnostic - Part 2.  Tests either or both SBIAs
Run/pass:50 EDK7A
!The SBIA microdiagnostic has been changed.  EDK6A is now part 1, and has
!standard tests that it applies against either SBIA 0, SBIA 1, depending upon
!the contents of the selection register "ESC 34".  See GUIDE.MEM, or
!"show/ascii edk6a.com"
!** EDK7A is now part 2, and also has standard tests that it applies against
! either SBIA 0 or SBIA 1, depending upon the contents of the selection
!register "ESC 34".
!*** End Special Note ***
DC>>
```

Figure 5-5 MICROS.COM Command File (Sheet 2 of 2)

Use "xxxxx" to assign a unique filename for each microdiagnostic. Appendix F summarizes the diagnostics naming conventions. Table 5-2 lists all 24 microdiagnostics, indicating which RAMs are used and the filename of the xxxxx.BPN file loaded.

The following discussion describes the environment for running the EBox microdiagnostic EDKBA (refer to Figure 5-1). Note that the IBox and FBox control RAMs are not used during execution of the EBox microdiagnostic.

- EDKBA.COM -- Top-level command file used to control loading of all control RAMs in the VAX CPU and initializing the CPU for running the EBox microdiagnostic.
- EDKBE.COM -- Lower-level command file invoked from EDKBA.COM to deposit test data and initialize the EBox's scratchpad RAMs.
- DSM.BPN -- Diagnostic Support Microcode loaded into the low 2 Kbyte segment of the EBox control store during diagnostic context initialization.
- EDKBU.BPN -- EBox microcoded tests loaded into the upper 6 Kbyte segment of the EBox control store. This is the actual microdiagnostic.
- MCF.BPN -- System microcode loaded into the EBox's MCF RAMs.
- CTX.BPN -- System microcode loaded into the EBox's CONTEXT RAMs.
- ACCESS.BPN -- System microcode loaded into the MBox's Access Violation RAMs.

- CYCLE.BPN -- System microcode loaded into the MBox's Cycle Condition Code RAMs.
- UCODE.BPN -- System microcode loaded into the MBox's Control Store.
- EDKBA.DCI -- Coarse isolation algorithms read by DCP when a solid fault is detected by the microdiagnostic.
- EDKBA.DCB -- Fine isolation algorithms read by DCP when a solid fault is detected.

Table 5-2 Microdiagnostics Microcode Files

NAME	ECS	MCF	ICS	IDRAM	MCS	CTX	CYC	ACS	CACHE	FBA	FBM	FDRAM
code*=>	(U)	(Y)	(I)	(D)	(M)	(X)	(N)	(H)	(C)	(B)	(F)	(G)
DSM	DDD†	SYS†	...	...	SYS	SYS	SYS	SYS	...	...	...	...
EDKBA	DDD	SYS	...	...	SYS	SYS	SYS	SYS	...	...	...	...
EDKCA	DDD	DDD	...	...	DDD	SYS	DDD	SYS	...	...	...	...
EDKDA	DDD	DDD	...	...	DDD	SYS	SYS	SYS	...	...	...	...
EDKEA	DDD	DDD	...	...	SYS	SYS	SYS	SYS	...	...	...	...
EDKFA	DDD	DDD	...	...	SYS	SYS	SYS	SYS	...	...	...	...
EDKGA	DDD	DDD	...	...	SYS	SYS	SYS	SYS	...	...	...	...
EDKHA	DDD	DDD	...	...	DDD	SYS	SYS	SYS	DDD	...	...	...
EDKOA	DDD	SYS	DDD	...	SYS	SYS	SYS	SYS	...	...	...	...
EDKPA	DDD	SYS	DDD	...	SYS	SYS	SYS	SYS	DDD	...	...	...
EDKQA	DDD	SYS	DDD	...	SYS	SYS	SYS	SYS	...	...	...	...
EDKRA	DDD	SYS	DDD	DDD	SYS	DDD	SYS	SYS	DDD	...	...	...
EDKSA	DDD	SYS	DDD	DDD	SYS	SYS	SYS	SYS	DDD	...	...	...
EDKTA	DDD	SYS	DDD	DDD	SYS	SYS	SYS	SYS	DDD	...	...	...
EDKUA	DDD	SYS	DDD	DDD	SYS	SYS	SYS	SYS	DDD	...	...	...
EDKVA	DDD	SYS	DDD	DDD	SYS	SYS	SYS	SYS	DDD	...	...	...
EDKWA	DDD	SYS	DDD	DDD	SYS	DDD	SYS	SYS	DDD	...	...	...
EDK1A	DDD	SYS	...	...	SYS	SYS	SYS	SYS	...	DDD	DDD	...
EDK2A	DDD	SYS	...	...	SYS	SYS	SYS	SYS	...	DDD	DDD	...
EDK3A	DDD	SYS	...	...	SYS	SYS	SYS	SYS	...	DDD	DDD	...
EDK4A	DDD	SYS	SYS	SYS	SYS	SYS	SYS	SYS	...	DDD	DDD	SYS
EDKZA	DDD	SYS	SYS	SYS	SYS	SYS	SYS	SYS	...	DDD	DDD	SYS
EDK5A	DDD	SYS	...	...	SYS	SYS	SYS	SYS	...	...	...	...
EDK6A	DDD	DDD	...	...	DDD	SYS	SYS	SYS	DDD	...	...	...
EDK7A	DDD	DDD	...	...	DDD	SYS	SYS	SYS	DDD	...	...	...

* The "code" indicates the last letter in the filename.

† A "DDD" means that the diagnostic loads diagnostic microcode into that control store. An "SYS" means that the diagnostic uses the standard system microcode in that control store. A "..." means that the diagnostic does not require the particular control store to be loaded with anything.

To complete our discussion of the test environment, we need to look at the following programs resident in the T-11 RAM.

- DCN -- The root console software overlay.
- DCP -- The program overlay that provides the communication interface between the console terminal (local or remote) and the microdiagnostic EDKBU via DSM and the CBus.
- HEX -- The HEX command set program overlay that may be enabled. HEX uses the DEBUG command to extend the commands available to the user.

With this understanding of all the pieces and how they integrate to form the test environment, we can move on to the detailed procedures for running microdiagnostics.

5.4 OPERATING PROCEDURES

This section describes how to actually run microdiagnostics. Several examples will be used to show both the command string you type in on the console terminal and the normal responses displayed by DCP. We'll start with the EBox. To run the EBox microdiagnostic EDKBA, type the @ command to invoke the required indirect command file. Refer to Figure 5-6 for a summary of the command files invoked.

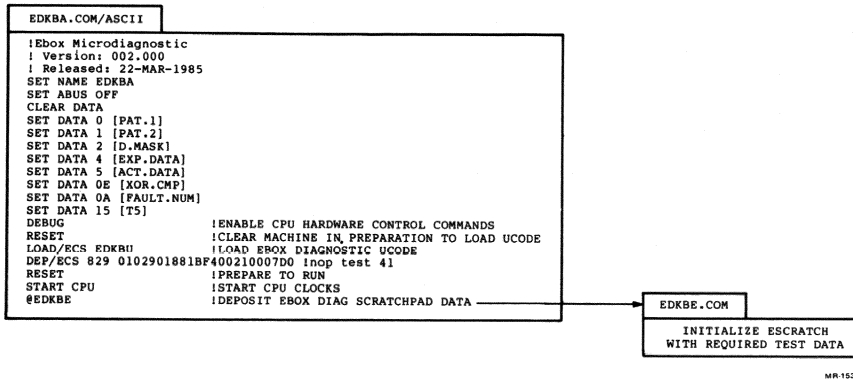


Figure 5-6 EBox Microdiagnostic Initialization

The first step is to type the following command.

```
DC>@EDKBA
```

This causes DCP to read the file EDKBA.COM from the RL02 and execute the commands it contains. When DCP is finished executing the top-level command file, it returns the DC> prompt and waits for the user to start the microdiagnostic by typing the following command.

```
DC>>START
```

DCP sends the START command to DSM over the CBus, DSM starts the microdiagnostic at test 1, and the diagnostic is "off and running" out of the EBox control store. All tests contained in EDKBU.BPN will be executed 100 times each (Default PASS COUNT). If no errors are detected, it will report END OF PASS to DCP following the last test. At this time, DCP will display the following message and return to the DC>> prompt. Note again that the command file enabled the HEX command set.

```
END OF PASS - EDKBA - START TEST#1 - END TEST #43
DC>>
```

Figure 5-7 summarizes the complete run and shows how <CTRL/T> is used to determine how the test is progressing. <CTRL/T> is useful if the user suspects the microdiagnostic might be hung up.

```
DC>>@EDKBA
DC>>START
^T
Running diagnostic - EDKBA
Test # = 2 & alive byte = 26
^T
Running diagnostic - EDKBA
Test # = 4 & alive byte = 43
^T
Running diagnostic - EDKBA
Test # = 7 & alive byte = 32
^T
Running diagnostic - EDKBA
Test # = 12 & alive byte = 55
^T
Running diagnostic - EDKBA
Test # = 30 & alive byte = 146
End of pass - EDKBA - start test #1 - end test #43
DC>>
```

Figure 5-7 Sample Run of EDKBA (EBox Microdiagnostic)

The same general procedure applies for all 24 microdiagnostic programs; only the names of the diagnostics change. The following three examples will illustrate the procedure.

1. The MBox diagnostic, EDKCA
2. The IBox diagnostic, EDKRA
3. The Array diagnostic, EDK5A

Figure 5-8 shows the command procedure for running EDKCA, one of the MBox microdiagnostics, while Figure 5-9 summarizes the RL02 files the procedure uses.

```
DC>>@EDKCA
DC>>START
End of pass - EDKCA - start test #1 - end test #9
DC>>
```

Figure 5-8 EDKCA Command Procedure

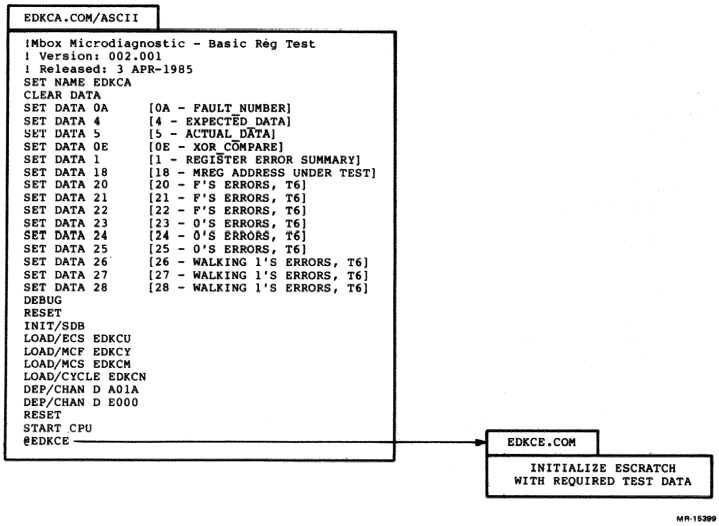


Figure 5-9 MBox Microdiagnostic Initialization

Figure 5-10 shows the command procedure for running EDKRA, one of the IBox microdiagnostics, while Figure 5-11 summarizes the RL02 files the procedure uses.

```
DC>>@EDKRA
DC>>START
End of pass - EDKRA - start test #1 - end test #D
DC>>
```

Figure 5-10 EDKRA Command Procedure

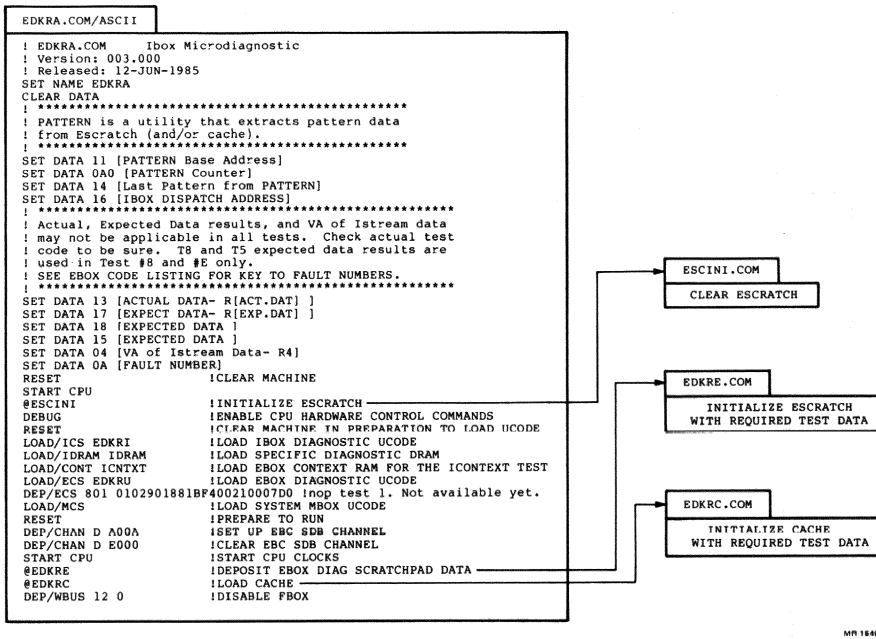


Figure 5-11 IBox Microdiagnostic Initialization

Figure 5-12 shows the command procedure for running EDK5A, the array microdiagnostic, while Figure 5-13 summarizes the RL02 files used by the procedure.

```
DC>>@EDK5A
DC>>START/PASSES:1
End of pass - EDK5A - start test #1 - end test #5
DC>>
```

Figure 5-12 EDK5A Command Procedure

EDK5A.COM/ASCII

```
! EDK5A.COM     Array Microdiagnostic
! Version: V003.001
! Released: 18-JUN-1985
SET NAME EDK5A
CLEAR DATA
SET DATA 11     [11 - ARRAY SELECT MASK]
SET DATA 12     [12 - STARTING ADDRESS]
SET DATA 13     [13 - HIGH ADDRESS]
SET DATA 14     [14 - CURRENT ADDRESS]
SET DATA 15     [15 - ERROR COUNT]
SET DATA 25     [25- MBOX 18/14/10/C]
SET DATA 26     [26- MBOX 4]
SET DATA 0A     [A - FAULT.NUM]
SET DATA 20     [20 - MBOX 2C/28/24/20 MSTAT1]
SET DATA 21     [21 - MBOX 54/5C/58 MSTAT2]
SET DATA 22     [22- MBOX 70/60/50 MDECC]
SET DATA 24     [24- MBOX 7C MEAR]
SET DATA 27     [27- EBCS]
SET DATA 01     [01- R1]
DEBUG
RESET
INIT/SDB
LOAD/ECS EDK5U
LOAD/MCS
LOAD/CYCLE
LOAD/MCF
START CPU
dep/cache 0 0  !Init mbox
@EDK5E
DEP/ESC A0 FFFFFFFF
DEP/ESC A1 FFFFFFFF
DEP/ESC A2 FFFFFFFF
DEP/ESC A3 FFFFFFFE
DEP/ESC A4 FFFFFFFF
DEP/ESC A5 FFFFFFFF
DEP/ESC A6 FFFFFFFF
DEP/ESC A7 7FFFFFFF
DEP/ESC A8 FFFFFFFF
DEP/ESC A9 FFFFFFFF
DEP/ESC AA FFFFFFFF
DEP/ESC AB 555555555
DEP/ESC AC FFFFFFFF
DEP/ESC AD FFFFFFFF
DEP/ESC AE FFFFFFFF
DEP/ESC AF AAAAAAAA
DEP/ESC 11 FEFF ! SELECT ALL ARRAYS, AUTOCONFIG ALL BUT SLOT 0
DEP/ESC 25 10000400 !CONTROL REGISTER SETUP
DEP/ESC 26 3    !ENABLE BOTH CACHES
```

EDK5A.COM

INITIALIZE ESCRATCH WITH REQUIRED TEST DATA

MR-15401

Figure 5-13 Array Microdiagnostic Initialization

There is one command file on the RL02 named MICROS.COM, that permits the user to automatically sequence all available microdiagnostics in the proper order. The user can do this with the following single command.

```
DC>@MICROS
```

This command file both loads and starts each microdiagnostic in each of the six microdiagnostics series, and sets the PASS counter to provide a quick verify mode of the CPU hardware. The commands in this file are shown in Figure 5-5. Figure 5-14 shows the normal response to the @MICROS command.

```
DC>>@MICROS
Running diag EDKBA
End of pass - EDKBA - start test #1 - end test #43
Running diag EDKCA
End of pass - EDKCA - start test #1 - end test #9
Running diag EDKDA
End of pass - EDKDA - start test #1 - end test #2D
Running diag EDKEA
End of pass - EDKEA - start test #1 - end test #45
Running diag EDKFA
End of pass - EDKFA - start test #1 - end test #1E
Running diag EDKGA
End of pass - EDKGA - start test #1 - end test #1
Running diag EDKHA
End of pass - EDKHA - start test #1 - end test #D
Running diag EDKOA
End of pass - EDKOA - start test #1 - end test #1
Running diag EDKPA
End of pass - EDKPA - start test #1 - end test #6
Running diag EDKQA
End of pass - EDKQA - start test #1 - end test #4
Running diag EDKRA
End of pass - EDKRA - start test #1 - end test #D
Running diag EDKSA
End of pass - EDKSA - start test #1 - end test #6
Running diag EDKTA
End of pass - EDKTA - start test #1 - end test #6
Running diag EDKUA
End of pass - EDKUA - start test #1 - end test #9
Running diag EDKVA
End of pass - EDKVA - start test #1 - end test #9
Running diag EDKWA
End of pass - EDKWA - start test #1 - end test #4
Running diag EDK1A
End of pass - EDK1A - start test #1 - end test #17
Running diag EDK2A
End of pass - EDK2A - start test #1 - end test #B
Running diag EDK3A
End of pass - EDK3A - start test #1 - end test #D
MBOX=0000 EBOX=04EC FBOXA=000F FBOXM=0004 IBOX=002D
Running diag EDK4A
End of pass - EDK4A - start test #1 - end test #6
Running diag EDKZA
End of pass - EDKZA - start test #1 - end test #3
Running diag EDK5A
End of pass - EDK5A - start test #1 - end test #5
Running diag EDK6A
End of pass - EDK6A - start test #1 - end test #70
Running diag EDK7A
End of pass - EDK7A - start test #1 - end test #3C
DC>>
```

Figure 5-14 Console Display When Running MICROS.COM

Note that the command file, MICROS.COM, does not include running the microhardcore diagnostics before running all the available microdiagnostics in the proper sequence. For a complete bottom-up test of the VAX CPU, there is another command file that runs MHC first, and then invokes MICROS.COM to run the microdiagnostics. This command file is invoked by typing @TSTCPU and causes the console to read and execute commands from the file TSTCPU.COM. Figure 5-15 demonstrates this.

```
DC>>SHOW TSTCPU.COM/ASCII
!General Command file for running MHC and all Micros for VAX 8600 cpu
! Version: 001.000
! Released: 30-jan-1985
!Command File for executing the entire string of Microdiagnostics delivered
! at Engineering Venus release M 5.0 ****PRELIMINARY****
!       * Revised Aug 21, 1984
!       * Revised October 3, 1984
!       * Revised Dec 5, 1984
!
MHC             ! Enter MH context
Start           ! Run all MH tests
DIAG            ! Enter DIAG context, which will run MICROS.COM automatically
@MICROS         ! Run diagnostic script file
DC>>
```

Figure 5-15 TSTCPU.COM Command File

Other miscellaneous command files are available on the RL02 that permit running fixed diagnostic sequences required for testing specific CPU modules. Tables 5-3 through 5-5 list all of the currently available module test command files. Figure 5-16 lists the commands contained in QVL205.COM that test the MBox's MAP module.

The following topics have been covered up to this point.

1. What types of microdiagnostics are provided
2. How to run them in the proper sequence
3. How to use the special command files to automate the process

The next step is to discuss the error messages that DCP displays when the microdiagnostic detects a fault.

Table 5-3 Microdiagnostic Module Test Command Files

Filename	Module
QVL202.COM	L0202 (SBS)
QVL203.COM	L0203 (SBA)
QVL204.COM	L0204 (MCD)
QVL205.COM	L0205 (MAP)
QVL206.COM	L0206 (IDP)
QVL207.COM	L0207 (ICA)
QVL208.COM	L0208 (IBD)
QVL209.COM	L0209 (EDP)
QVL210.COM	L0210 (EBC)
QVL211.COM	L0211 (EBD)
QVL212.COM	L0212 (FBA)
QVL213.COM	L0213 (FBM)
QVL214.COM	L0214 (ICB)
QVL215.COM	L0215 (CSA)
QVL216.COM	L0216 (CSB)
QVL217.COM	L0217 (CLK)
QVL218.COM	L0218 (FJM)
QVL219.COM	L0219 (EBE)
QVL220.COM	L0220 (MCC)
QVL222.COM	L0222 (MTM)

Table 5-4 MHC/Micro Module Test Command Files

Filename	Module
QVX202.COM	L0202 (SBS)
QVX203.COM	L0203 (SBA)
QVX204.COM	L0204 (MCD)
QVX205.COM	L0205 (MAP)
QVX206.COM	L0206 (IDP)
QVX207.COM	L0207 (ICA)
QVX208.COM	L0208 (IBD)
QVX209.COM	L0209 (EDP)
QVX210.COM	L0210 (EBC)
QVX211.COM	L0211 (EBD)
QVX212.COM	L0212 (FBA)
QVX213.COM	L0213 (FBM)
QVX214.COM	L0214 (ICB)
QVX215.COM	L0215 (CSA)
QVX216.COM	L0216 (CSB)
QVX217.COM	L0217 (CLK)
QVX218.COM	L0218 (FJM)
QVX219.COM	L0219 (EBE)
QVX220.COM	L0220 (MCC)
QVX222.COM	L0220 (MTM)
QVX223.COM	L0223 (FTM)

Table 5-5 MHC Module Test Command Files

Filename	Module
MHC209.COM	L0209 (EDP)
MHC222.COM	L0222 (MTM)
MHC223.COM	L0223 (FTM)

```
DC>>SHOW QVL205.COM/ASCII
!QVL205.COM - Mfg STAGE1 script file to Run Micros
! Version: 002.000
! Released: 12-JUN-1985
! Command File for QV testing of the L0205 module. (MAP)
!. Microdiagnostics delivered at Engineering Venus release M 5
!       * Revised December 11, 1984
!       * Revised June 12, 1985
!
!**Note: this command file does Not run the MicroHardCore, which is run from
! within the console software subsystem.
!
!Mbox Microdiagnostic
Run/pass:100 EDKCA
!Mbox Microdiagnostic
Run/pass:3 EDKDA
!Mbox Microdiagnostic
Run/pass:100 EDKEA
!Mbox Microdiagnostic
Run/pass:100 EDKFA
!Mbox Microdiagnostic
Run/pass:100 EDKGA
!Mbox Microdiagnostic
Run/pass:100 EDKHA
!Ibox Microdiagnostic
Run/pass:100 EDKRA
!Ibox Microdiagnostic
Run/pass:100 EDKTA
!Array Microdiagnostic - 4Meg Version
!Default is to test Array Module in slot 0 only
Run/pass:3 EDK5A
!SBIA Microdiagnostic
Run/pass:50 EDK6A
!SBIA Microdiagnostic
Run/pass:50 EDK7A
DC>>
```

Figure 5-16 L0205 Module Test Command File

5.5 FAULT ISOLATION MESSAGES

This section will use three examples to illustrate the format of the error messages that DCP displays when the microdiagnostics detect a fault. The first example includes a detailed description of each component of the message displayed.

Figure 5-17 shows a sample /PRINT MODE:VERBOSE form of printout from a microdiagnostic. The numbers in the left column do not appear in the actual output, but are used as a way to refer to the components in the detailed description which follows.

```
 1.  DC>>START/NU:1 11
 2.  //////////////////////////////////////////////////////
 3.  Diagnostic test name:  EDK3A.
 4.  BPM version:  EDK3A V0002.000 03/22/1985
 5.  Syndrome #:  1
 6.  Start test #:  1
 7.  End test #:  11
 8.  Loop goal:  100
 9.  Pass count:  1
10.  Ebox scratchpad data:
11.  [PAT.1] = 00000000
12.  [PAT.2] = 00000000
13.  [ACT.DAT] = 00000000
14.  [ACT.DAT2] = 00000000
15.  [FAULT.NUM] = 00000001
16.  [TST.LABEL] = 00000002
17.  [XOR.CMP] = 00000000
18.  [F.REG] = 00000000
19.  Fault detected in test #2 - EDK3A
20.  Isolation data:
21.  The Fbox FBM Module made an incorrect branch on exponent zero,
22.  data ready, or was unable to return from a call subroutine.
23.  SECTION SLOT COMPONENT MODULE
24.  CPU 7 E-42 L0213/L0218
25.  CPU 7 E-37 L0213/L0218
26.  CPU 8 E-3 L0213/L0218
27.  //////////////////////////////////////////////////////

     NOTE: The numbers in the first column do not appear
           in the actual display.
```

Figure 5-17 Isolation Message -- Sample 1

The following list describes each component part of the isolation message display shown in Figure 5-17.

1. DC>>START/NU:1 11 -- The user typed command to run tests 1 through 11.

2. //////...../////// -- Indicates start of message.

3. Diagnostic test name: EDK3A -- Official name of this diagnostic. All problem reports and questions should include this name. This name is also the key to finding listings for this test. Every diagnostic test, no matter what box it is testing, is controlled by special EBox microcode. Furthermore, the descriptions and documentation for any test will be found in the EBox microcode listing for that diagnostic. The name of the listing for the EBox microcode in the diagnostic is determined by dropping the last letter of the official diagnostic name (this last letter shall always be an "A"), and putting the letter "U" in its place. Therefore, the EBox microcode listing for diagnostic EDK3A is found in the EDK3U.MCR file. All microcode listing files have a .MCR extension.

4. BPN version: EDK3A V0002.000 03/22/1985 -- This is the version number of this diagnostic, as read out of Ucode location 800, along with the date of that version's release. This number will match the contents of location 800 as found in the EDK3U.MCR file.

5. Syndrome 1 -- The 1 means that this is the first error syndrome encountered by this diagnostic. The error syndrome is all of the scratchpad data and the values in those scratchpads. If even a single bit in any of the printed scratchpads should change from one fault report to another, DCP will treat it as an entirely new error syndrome. If a second syndrome were encountered, it would be number 2, and so on, up to a maximum of 10 syndromes. Additional syndromes after 10 are not printed. If more than one syndrome is encountered by the microdiagnostics, there will be no attempt to isolate the fault. Multiple syndromes mean either that the fault is not solid, or that the diagnostic is suffering from an initialization problem. In both cases, the isolation could be misleading.

6. Start test: 1 -- The number of the starting test specified in the START command.

7. End test: 11 -- The number of the ending test number specified in the START command.

8. Loop goal: 100 -- Ignore for current release.

9. Pass count: 1 -- Ignore for current release.

10. EBox scratchpad data: -- The registers that are printed following this header are specific to each diagnostic and may change from diagnostic to diagnostic. Also, depending on what is happening in each specific test of a diagnostic, some of these registers may have no meaning. The listing for each test should show which registers are used during any particular test. Some tests are checking branch conditions in the micro sequencers, or perhaps expecting microtraps, and do not really have any expected data, or actual data.

11. [PAT.1] = 00000000 -- Usually a stimulus pattern used to set up a test condition.

12. [PAT.2] = 00000000 -- Some tests require more than one stimulus condition. PAT.2 would be the second pattern used to set up the test condition.

13. [ACT.DAT] = 00000000 -- Actual data received.

14. [ACT.DAT2] = 00000000 -- Actual data received.

15. [FAULT.NUM] = 00000001 -- This is a key register and should be valid for each and every microdiagnostic test. The FAULT.NUM is just like a subtest number and should point you into the listing to the particular error call to DSM, or the particular test setup that detected the fault. FAULT.NUM is the second thing you should look at after the failing test number, in order to find the spot in the microcode listing that detected the fault.

16. [TST.LABEL] = 00000002 -- If implemented, the number in this register should match the failing test number.

17. [XOR.CMP] = 00000000 -- If the test had any expected/actual data, this register should contain the XOR of that data.

18. [F.REG] = 00000000 -- This particular register is only found in the FBox microdiagnostics and contains the contents of one of the FBox control status registers.

19. Fault detected in test 2 -- EDK3A -- Indicates the number of the test that detected this failure.

20. Isolation data: -- Header that marks the beginning of this test's component isolation printout. A brief description of the failure is in 21 and 22.

23. SECTION SLOT COMPONENT MODULE -- Chip/module callout header.

24. CPU 7 E-42 L0213/L0218 -- Component callout. Includes the backplane section and backplane slot number of the module that may contain the failure. This is the module that should be replaced. Also contains the list of potentially faulty components. These are the components that should be replaced.

25. CPU 7 E-37 L0213/L0218 -- Chip/module callout message.

26. CPU 8 E-3 L0213/L0218 -- Chip/module callout message.

27. ////////...../////////// -- Indicates end of isolation message.

The isolation data in lines 20 through 26 is only displayed for solid faults where the test fails every pass with the same error syndrome. For non-solid faults, no isolation data is displayed, and only error messages for up to 10 different error syndromes are displayed. These messages will include the information contained in lines 2 through 19, and replace lines 20 through 26 with the following two lines.

1. Test Failure Rate: "n" failure(s) from "p" passes
2. There were "x" syndrome(s) encountered

Here "n" indicates the number of failures detected, "p" indicates the current pass count setting, and "x" indicates the number of different error syndromes detected.

Two more sample message displays are shown in Figures 5-18 and 5-19.

```
DC>>START/PASS:1 EDKGA
////////////////////////////////////////
DIAGNOSTIC TEST NAME: EDKGA.
START TEST NO: 1
END TEST NO: FF
LOOP GOAL: 1
PASS COUNT: 1
EBOX SCRATCHPAD DATA:
[R0] = 00000000
[T0] = 00000000
[T1] = 00000000
[T10] = 00000000
[T13] = 00000000
[T15] = 00000001
[A - FAULT.NUM] = 00000001
[D - ACT.DATA] = 00000026
[E = XOR.CMP] = 00000030
[1F - D.MASK] = 00000000
[22 - EXP.DATA] = 00000016
[34 - TRAP.MASK] = 00000000
[35 - TRAP.OCCURRED] = 00000000
[36 - VMQ] = 00000000
[37 - VMQ.SAV] = 00000000
[38 - DATA.SAV] = 00000000
FAULT DETECTED IN TEST #1 - EDKGA.
ISOLATION DATA:
VERIFY ERROR ON 'WRITE ARRAY 0 WITH BIT 0 BAD' TEST

SECTION    SLOT    COMPONENT    MODULE
CPU        16       E-0         L0204
////////////////////////////////////////
```

Figure 5-18 Isolation Message -- Sample 2

```
DC>>START EDKBA
////////////////////////////////////////
DIAGNOSTIC TEST NAME: EDKBA.
START TEST NO: 1
END TEST NO: FF
LOOP GOAL: 100
PASS COUNT: 1
EBOX SCRATCHPAD DATA:
[PAT.1] = 00000000
[PAT.2] = 00000000
[D.MASK] = FFFF9FFE
[EXP.DATA] = 00000000
[ACT.DATA] = 00000002
[XOR.CMP] = 00000002
[FAULT.NUM] = 00000002
FAULT DETECTED IN TEST #28 - EDKBA.
ISOLATION DATA:
SECTION    SLOT    COMPONENT    MODULE
CPU        6       E-42         L0211
CPU        6       E-41         L0211
CPU        6       E-18         L0211
CPU        6       E-15         L0211
CPU        6       E-136        L0211
CPU        6       E-69         L0211
CPU        6       E-66         L0211
CPU        6       E-19         L0211
CPU        6       E-4          L0211
CPU        6       E-55         L0211
CPU        6       E-7          L0211
CPU        6       E-53         L0211
CPU        6       E-156        L0211
CPU        6       E-68         L0211
CPU        9       E-143        L0219
CPU        9       E-6          L0219
CPU        9       E-5          L0219
////////////////////////////////////////
```

Figure 5-19 Isolation Message -- Sample 3

5.6 UNUSUAL ERROR MESSAGES

There will be times when DCP will respond with a nonstandard error message to indicate something has gone awry; the microdiagnostic is either hung or has sensed some unexpected response that it is unable to handle. These messages are generally caused by the nature of the fault, a bug in the microdiagnostic, or a simple operator error. Operator errors can generally be cleared up by simply reloading and restarting the microdiagnostic that caused the problem. The other two will require software design changes to correct the problem.

This section describes the three most common types of responses. It also outlines procedures to capture the state of the microdiagnostic at the time of the fault so that diagnostic engineering can analyze the problem and design modifications to correct it.

5.6.1 DSM/DCP Communication Failures

When DCP, running in the T-11 microprocessor, is unable to communicate with DSM, running in the EBox control store, the following message is displayed and DCP returns to the DC prompt.

"?DCP-E-NOANSD, DSM-DC communication failure"

This could be caused by programming or hardware faults and can occur any time DCP is running in the diagnostic context. Whatever the cause, the state of the machine must be captured for analysis by diagnostic engineering. This information can then be analyzed to either fix the bug or write an additional microtest to detect and properly report the hardware fault earlier in the diagnostic testing sequence. Here's the "snapshot" procedure.

1. Enable hardcopy by pressing **<CTRL/PRINT>**, if a hardcopy terminal is available.

2. Type **DEBUG** to enable the HEX command set.

3. Type **STOP CPU** to stop the clocks.

4. Type **MIC** to snapshot the contents of the EBox, IBox, MBox, and FBox micro-PC.

5. Press the space bar 10 times to snapshot the next 10 sets of micro-PCs.

6. Press **<RETURN>** to exit the space-bar mode.

7. Type **RESET** to reinitialize the CPU.

8. Type **START CPU** to restart the CPU clocks.

9. Type **EXAMINE/ESC 70** to snapshot the status of DSM.

10. Type **EXAMINE/ESC 73** to snapshot the test number that failed.

11. Type **SHOW DATA** to snapshot the fault syndrome.

12. Type **@EDxxA.COM** to reload the microdiagnostic that failed.

13. Type **START** to restart the microdiagnostic to determine if the fault symptoms are consistent.

14. If the fault occurs again, repeat the previous steps to capture a second snapshot.

15. Finally, save all this data and include it with a Software Problem Report (SPR) to Diagnostic Engineering in Marlboro, Mass.

5.6.2 Unexpected EBox Microtraps

After starting any microdiagnostic, the following message may occur.

"?DCP-E-UMICTP, unexpected microtrap at vector xx"

This indicates that a hardware fault caused an EBox microtrap that the current microdiagnostic test neither forced nor was prepared to handle. The "xx" indicates the actual trap vector address. Two things could have caused this error. Either the microdiagnostic has an initialization problem or the logic fault that caused the trap should have been detected by an earlier microdiagnostic. Both require program changes by Diagnostic Engineering. To make the required changes, the designer needs to analyze the state of the machine at the time the trap occurred. To provide this feedback, you need to perform the following procedure.

1. Press <CTRL/PRINT> to enable hardcopy.
2. Type **DEBUG** to enable the HEX command set.
3. Type **SHOW SWITCHES** to snapshot the state of the DCP program control switches.
4. Type **SHOW DATA** to snapshot the contents of the currently defined data tables for the microdiagnostic that failed.
5. Type **EXAMINE/WBUS 6** to snapshot the WBus registers.
6. Type **EXAMINE/WBUS 7.**
7. Type **EXAMINE/WBUS 9.**
8. Type **EXAMINE/WBUS 11.**
9. Type **EXAMINE/WBUS 12.**
10. Type **EXAMINE/WBUS 13.**

Now type **START** to restart the tests. If the problem reoccurs, repeat the procedure to take another snapshot of the machine state. Finally, save all this data and include it with a Software Problem Report (SPR) to Diagnostic Engineering in Marlboro, Mass.

5.6.3 Keep-Alive Failures

While a microdiagnostic is running, DCP polls a location in the DC022 register file called DSM$ALIVE to determine if the microdiagnostic is still running. DSM increments this location each time it is called to perform some service to the microdiagnostic, which signals DCP that the diagnostic is still running. DCP uses a 5-second timer to check the DSM$ALIVE location, and if it expires before a test completes, an error message is displayed and the current command is aborted.

When a microdiagnostic gets hung in some infinite loop, due to either a programming error or the nature of the hardware fault, DCP senses that the test failed to finish during the 5-second interval and displays the following error message.

"?DCP-E-ALIVEE. invalid dsm keepalive byte"

Again, you must save key information about the state of the machine so that Diagnostic Engineering can analyze the fault and make the required changes to correct it. Use the following procedure to capture this information.

1. Enable hardcopy by pressing <CTRL/PRINT> if a hardcopy terminal is available.

2. Type **DEBUG** to enable the HEX command set.

3. Type **STOP CPU** to stop the clocks.

4. Type **MIC** to snapshot the contents of the EBox, IBox, MBox, and FBox micro-PC.

5. Press the space bar 10 times to snapshot the next 10 sets of micro-PCs.

6. Press <RETURN> to exit space-bar mode.

7. Type **RESET** to reinitialize the CPU.

8. Type **START CPU** to restart the CPU clocks.

9. Type **EXAMINE/ESC 70** to snapshot the status of DSM.

10. Type **EXAMINE/ESC 73** to snapshot the test number that failed.

11. Type **SHOW DATA** to snapshot the fault syndrome.

12. Type **@EDxxA.COM** to reload the microdiagnostic that failed.

13. Type **START** to restart the microdiagnostic to determine if the fault symptoms are consistent.

14. If the fault occurs again, repeat the previous steps to capture a second snapshot.

15. Finally, save all this data and include it with a Software Problem Report (SPR) to Diagnostic Engineering in Marlboro, Mass.

5.7 MICRODIAGNOSTIC CONTROL

This section describes the interaction between DCP and DSM to control running the tests in a microdiagnostic. Figure 5-20 summarizes the functional flow and Appendix C describes the detailed operation of DSM. You should take a few minutes to read Appendix C before proceeding.

The user can control the flow either by setting switches, or typing commands or special control characters. First, we'll examine the use of the three allowable control characters and then, using a few examples, describe some typical control operations using commands and switches.

DCP handles three control characters while a microdiagnostic is running.

1. <CTRL/P> -- Interrupts the currently running microdiagnostic and returns control to the user. This places the microdiagnostic in the "pause" state, allowing the user to examine or modify the state of the diagnostic test environment, and then resume execution of the microdiagnostic.

2. <CTRL/C> -- Aborts the current command, which may include the need to abort a currently running microdiagnostic, and returns control to the user.

3. <CTRL/T> -- Commands DCP to display information about the state of the currently running microdiagnostic without disturbing its execution. Figure 5-21 shows an example of the information displayed.

Using the flow chart in Figure 5-20, the following discussion will use a few examples to describe the operation of microdiagnostics running under DCP's control. Each example assumes that the user typed the @EDKBA command to load and initialize the CPU for running the EBox microdiagnostic.

Example 1

```
START/NUMBER:  1 5  (No error)
```

This command instructs DCP to run tests 1 through 5 in the currently loaded microdiagnostic (EDKBA). Each test will be run as follows.

```
TEST 1          100 times
TEST 2          100 times
TEST 3          100 times
TEST 4          100 times
TEST 5          100 times
```

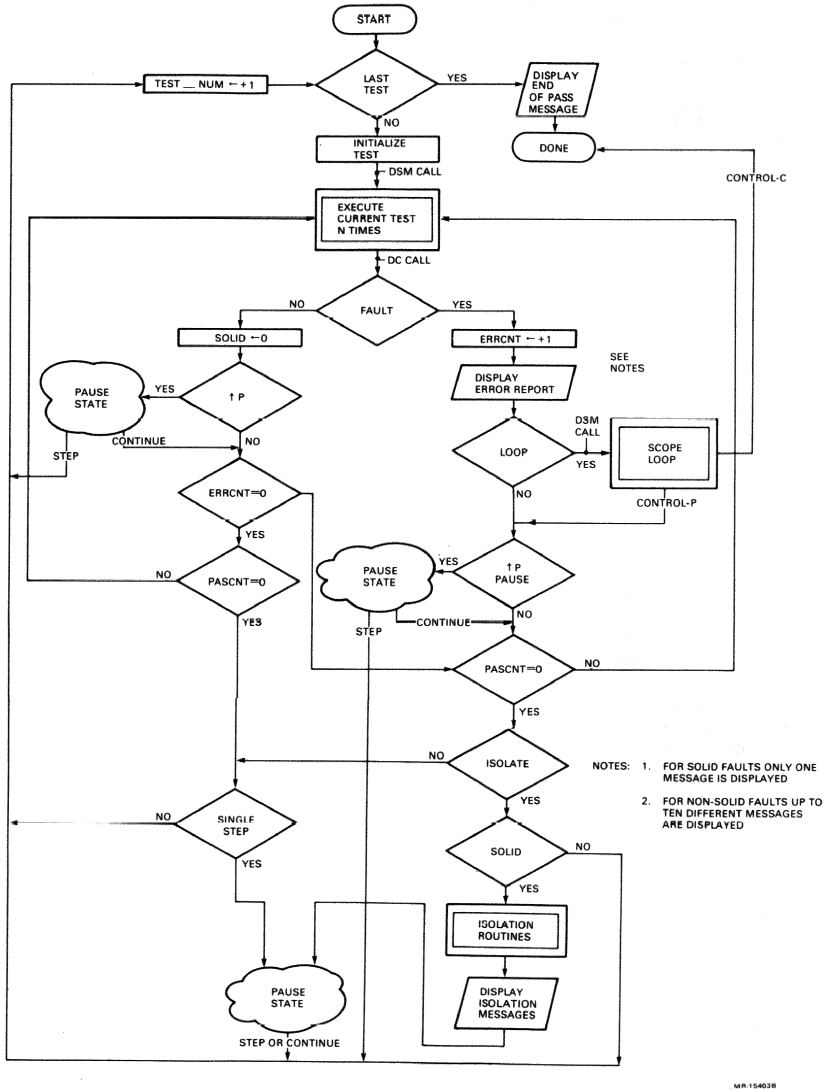


Figure 5-20 DCP Control Flow Summary

```
DC>>@EDK6A
DC>>START
^T
Running diagnostic - EDK6A
Test # = 2 & alive byte = 34
^T
Running diagnostic - EDK6A
Test # = 4 & alive byte = 32
^T
Running diagnostic - EDK6A
Test # = 1B & alive byte = 116
End of pass - EDK6A - start test #1 - end test #70
DC>>
```

Figure 5-21 <CTRL/T> Status Display

DCP will then display the following message and terminate.

```
END OF PASS - EDKBA - START TEST#1 - END TEST #5
DC>>
```

Here's the sequence of events.

1. DCP selects the test number, performs the required initialization, and calls DSM.

2. DSM transfers control to the microdiagnostic to execute the selected test 100 times.

3. DSM signals DCP that the test is done.

4. DCP increments the test number and repeats the sequence until tests 1--5 have been run.

5. Finally, after test 5, DCP displays the END OF PASS message and returns to the DC>> prompt to await another command.

Example 2

```
START/NUMBER:  1 5  (Solid error in Text 1)
```

On the first test, the microdiagnostic detects a fault and signals DSM to send a FAULT DETECTED response packet to DCP. This causes DCP to respond as follows.

1. Increments ERRCNT to count the errors.

2. Retrieves, formats, and displays the error syndrome data from the EBox scratchpad RAM. For a solid error, this data will be the same for all 100 passes of the failing test, so it is only displayed once.

3. DCP calls DSM again to repeat the sequence until all 100 passes have been completed.

4. When DCP sees the PASCNT=0, it calls its isolation routines to analyze the error syndrome data as directed by the isolation algorithms contained in EDKBA.DCI(DCB). This results in DCP appending fault isolation messages to the error display.

5. After displaying the fault isolation information, DCP enters the "pause" state to wait for user commands.

6. The user can now type either STEP or CONTINUE to resume testing with the next test.

Example 3

```
START/NUMBER:  1 5  (Non-solid error in Test 2)
```

On the second test, the microdiagnostic detects a fault and signals DSM to send a FAULT DETECTED response packet to DCP. This causes DCP to respond as follows.

1. Increments ERRCNT to count the errors.

2. Retrieves, formats, and displays the error syndrome data from the EBox scratchpad RAM. For a non-solid error, the error data will be displayed for each different syndrome up to a maximum of 10 messages.

3. DCP calls DSM again to repeat the sequence until all 100 passes have been completed.

4. When DCP sees the PASCNT=0, it exits to the next test without calling its isolation routines because the fault was not solid.

Example 4

```
START/SINGLE_STEP:ON (Single stepping mode)
```

Figure 5-22 illustrates DCP's response to this command.

This example assumes EDKBA runs error-free. The DCP/DSM sequence is as follows.

1. DCP selects the test number, performs the required initialization, and calls DSM.

2. DSM transfers control to the microdiagnostic to execute the selected test 100 times.

3. DSM signals DCP that the test is done.

```
DC>>@EDKBA
DC>>START/SINGLE_STEP
DC>>?DCP-I-PAUSI, pausing...
PAUSING AT END OF TEST #1
DC>>STEP
DC>>?DCP-I-PAUSI, pausing...
PAUSING AT END OF TEST #2
DC>>STEP
DC>>?DCP-I-PAUSI, pausing...
PAUSING AT END OF TEST #3
DC>>STEP
DC>>?DCP-I-PAUSI, pausing...
PAUSING AT END OF TEST #4
DC>>CONTINUE
DIAGS CONTINUING
END OF PASS - edkba - start test#1 - end test #43
DC>>
```

Figure 5-22 Single Step Mode Display

4. DCP, finding the SINGLE_STEP switch set, displays a message and enters the pause state.

5. The user types STEP which signals DCP to increment the test number and repeat the above sequence.

6. When DCP pauses after test 4, the user types CONTINUE which causes DCP to resume executing the remaining tests until all have been run 100 times. CONTINUE resets the SINGLE_STEP switch.

7. Finally, after the last test, DCP displays the END OF PASS message and returns to wait for another command.

Note how the user examined the contents of the EBox scratchpad location 22 after DCP entered the pause state. This illustrates the primary use of the pause state. It allows the user to use any available console command to retrieve additional information about the state of the machine. It is also possible to use the HEX command set while in the pause state to single-clock the microsequencer, and to examine key SDB signals after each clock state to obtain additional detail about the nature of the fault. Chapter 7 will describe more detailed examples of these procedures.

5.8 CONSOLE COMMANDS

While operating in the diagnostic context, the user has access to a set of 13 commands in addition to those provided by the general command set. Typing the HELP DIAG command will list the commands, as shown in Figure 5-23.

```
DC>>HELP DIAG
DIAG help available:
CLEAR_DATA              CONFIGURE               CONTINUE
DEPOSIT                 DESELECT                EXAMINE
GENERATE                RUN                     SELECT
START                   STEP                    SET_DATA
SET_DEFAULT             SET_ISOLATION           SET_NAME
SET_SWITCH              SHOW_CONFIGURATION      SHOW_DATA
SHOW_SWITCHES

Select "xxxxxx"
```

Figure 5-23 Diagnostic Context Commands

Typing the name of any command listed in response to the Select prompt (xxxxxx) will list additional descriptions for each of the 19 commands. This section will briefly describe how each command is used. Many of the commands are intended for inclusion in the top-level command file (used to load and initialize the machine for running microdiagnostics) and are not generally used by the service engineer during system maintenance.

NOTE

The following commands reflect Version 6.0 of the RL02 diagnostic pack. Earlier versions will have a slightly different set of commands.

5.8.1 CLEAR DATA

Syntax: CLEAR DATA

Description:

This command clears the table of pointers defined by the SET DATA command and should be used prior to redefining a new set of EBox scratchpad locations for error reporting purposes.

5.8.2 CONFIGURE (Arrays, SBIAs, FBox)

Syntax: CONFIGURE

Description:

This command causes the diagnostic operating system to determine which arrays and SBIAs are physically present and tests for the presence of an FBox. The command establishes internal configuration tables that identify:

- Which units are physically present
- Sizes and slots for arrays

In addition, this command sets software status bits for each of the available units, which makes them automatically selected_for_test when the appropriate microdiagnostics are run.

Current configuration and selected_for_test status can be displayed by the SHOW CONFIGURATION command. Selected_for_test status can be altered with the SELECT and DESELECT commands.

5.8.3 CONTINUE (a microdiagnostic)

Syntax: CONTINUE

Description:

This command allows the currently loaded microdiagnostic to resume test execution after being paused by either a switch setting (/FAULT:PAUSE, /FAULT:ISOLATE) or <CTRL/P>. The microdiagnostic resumes at the test following the one that was being executed when the pause occurred. If /SINGLE_STEP is in effect and CONTINUE command is issued, single stepping is stopped and the remaining tests are run. A CONTINUE command, issued prior to giving either a RUN or START command, will result in an error message.

5.8.4 DEPOSIT (Cache, Escratch, WBus)

Syntax: DEPOSIT { /switch } hex_addr hex_data

"/switch" can be any one of the following items.

/CACHE /ESCRATCH /WBUS

Description:

This command allows the user to modify the EBox scratchpad RAM, the system cache, or the WBus RAM. The 32-bit hexdata is deposited into the hexaddr location of the specified RAM. The CPU clock must be running.

5.8.5 DESELECT (Arrays, SBIAs, FBox)

Syntax: DESELECT ARRAY/SLOT:n (n = 1 thru 8, default = 1)
DESELECT SBIA0
DESELECT SBIA1
DESELECT FBOX

Description:

This command allows you to alter DCP configuration tables so as to exclude specific units for testing when the appropriate microdiagnostics for the options are run. The scheme supports the following.

ARRAYS in slots 1 thru 8
SBIA0 SBIA zero
SBIA1 SBIA one
FBOX System FBox

The command DESELECT FBOX is unrelated to the DCON command SET FBOX:{on,off}.

5.8.6 EXAMINE (Cache, Escratch, WBus)

Syntax: EXAMINE { /switch } hex_addr

"/switch" can be any one of the following items.

/CACHE /ESCRATCH /WBUS

Description:

This command allows the user to examine the contents of specific EBox scratchpad locations, cache locations, or WBus locations. The CPU clock must be running.

5.8.7 GENERATE

Syntax: GENERATE { filename }

Description:

The GENERATE command is used to invoke the DCP generate/verify process for validation and generation of some of the isolation data. This command is not for field use. It is used to support the in-house development of isolation algorithms. There are no switches associated with this command. The only option is the name of the command file responsible for loading the microdiagnostic to be generated.

5.8.8 RUN

Syntax: RUN [/switch] filename

"/switch" can be any combination of the following switches, provided that the command line does not exceed 80 characters. Switches appended to the RUN command remain in effect until the command completes. If a RUN command is given without switches, their default values are used.

```
/BELL:{ ON, OFF }
/FAULT:{ ISOLATE, NOABORT, LOOP, PAUSE, CONTINUE, IGNORE}
/LINES:{ ON, OFF }
/NUMBER:FIRST_TEST [ {<space>,<comma>} LAST_TEST ]
/PASSES:dec_num
/PRINT_MODE:{ BRIEF, VERBOSE }
/SINGLE_STEP
```

Description:

This command initiates the execution of a command file whose purpose is to load and start a microdiagnostic program. Execution begins with the first number specified in the /NUMBER switch and ends with either the last test in the group of microdiagnostic tests or with the last test specified in the /NUMBER switch. Note that the CPU clock must be running for the RUN command to work.

Refer to the description of the SET SWITCH command for information on switches not described here.

/NUMBER:FIRST_TEST [{<space>,<comma>}LAST_TEST] default = 01,FF

This switch is used to specify the starting and ending test numbers in hex. If the switch is omitted, default values are used.

/PASSES:'dec-num' default = 100

The argument "dec_num" is the number of test passes to execute before continuing to the next test or attempting isolation. If the /PASSES switch is omitted, the default value will be used.

A special case of /PASSES:0 modifies the sequencing of the tests so that a particular group of tests can be run indefinitely. If the user specifies a 0 pass count, DCP will run each of the tests indefinitely (from FIRST to LAST), until <CTRL/C> or <CTRL/P> interrupts or until a fault is detected. When a fault is detected, the action will be governed by the current setting of the /FAULT switch.

/SINGLE_STEP

This switch enables the single_step feature. When specified, diagnostic execution is interrupted after completing the proper number of passes of an individual test. A message, along with the test number just completed, is displayed and the operator is prompted for input. The STEP command can be used to run the next test and is followed by another pause at completion. The CONTINUE command can be used to clear this mode and resume full speed test execution.

5.8.9 SELECT (Arrays, SBIAs, FBox)

```
Syntax:  SELECT ARRAY/SLOT:n/TYPE:m      (n = 1 thru 8, default = 1)
         SELECT SBIA0                    (m = 4 or 16, default = 4)
         SELECT SBIA1
         SELECT FBOX
```

Description:

This command allows you alter DCP configuration tables so as to include specific units for testing when the appropriate microdiagnostics for the options are run. The units are in addition to currently selected devices. The scheme supports the following.

```
ARRAYS (4 and 16 meg) in slots 1 thru 8
SBIA0  SBIA zero
SBIA1  SBIA one
FBOX   System FBox
```

It is legal to select a unit for test, even if the hardware does not appear (via SHOW CONFIGURATION) to be physically present.

The command SELECT FBOX is unrelated to the DCON command SET FBOX:{on,off}.

5.8.10 SET DATA

Syntax: SET DATA escratch_address data_name

Where:

- "escratch_address" is a hexadecimal address within the EBox scratchpad RAM (0 to FF)
- "data_name" is a 1-30 character string to be associated with the specified Escratch data

Description:

This command allows a symbolic name to be assigned to a location in the EBox scratchpad RAM. When a microdiagnostic test detects a fault and DCP is in verbose printing mode, the "data_name" and data taken from the associated "escratch_address" are included in the error report.

SET DATA commands are normally used in command files responsible for loading microdiagnostics and setting diagnostic control switches. A maximum of 16 SET DATA commands can be used at one time. The CLEAR DATA command should be used to initialize the SET DATA memory prior to defining set data information. The SHOW DATA command can be used to display the current settings of SET DATA, along with the current state of the Escratch variables.

5.8.11 SET ISOLATION

Syntax: SET ISOLATION [/switch]

/PASSES:'dec_num'
/ERROR_DUMPS:'dec_num'

Description:

This command provides a way to modify default parameters associated with isolation. Once the defaults are modified, they remain in effect until DCP is reloaded (i.e., DIAG command).

/PASSES:'dec_num' default = 100

The argument "dec_num" indicates the number of times each test will be executed before continuing to the next test or attempting isolation. It is not recommended that this number be altered in the field, as it affects the confidence level of any isolation data that may follow. Factory settings are preferred. This number should always be greater than zero (0).

/ERROR_DUMPS:'dec_num' default = 10

The argument "dec_num" indicates the maximum number of error printouts that will occur as faults with different syndromes are detected within a given test. Lowering this number has the effect of reducing the amount of error printouts at the expense of throwing away what may be useful data for tracking down an intermittent. This number has a maximum value of 10.

5.8.12 SET NAME

Syntax: SET NAME microdiagnostic_filename

Description:

This command specifies the name of the microdiagnostic currently being loaded. This information is included in the fault report and is used to locate other associated files with the same name (different extensions).

5.8.13 SET SWITCH

Syntax: SET SWITCH [/switch]

"/switch" can be any combination of the following switches, provided that the line does not exceed 80 characters. Switches altered with this command retain the new setting until changed again with the SET SWITCH command, or until DCP is reloaded.

```
/BELL:{ ON, OFF }
/FAULT:{ ISOLATE, NOABORT, LOOP, PAUSE, CONTINUE, IGNORE }
/LINES:{ ON, OFF }
/PRINT_MODE: { BRIEF, VERBOSE }
```

Description:

This command redefines the default switch settings that govern the behavior of DCP and DSM in the control and running of microdiagnostics. A description of each switch is provided below, including the default setting of the switch when DIAGNOSTIC context is entered.

/BELL:{ ON, OFF } default = OFF

When "ON," this switch causes the terminal bell to ring each time a new fault is detected and reported.

/FAULT:{ ISOLATE, NOABORT, LOOP, PAUSE, CONTINUE, IGNORE }

default = ISOLATE

This switch controls the action DCP takes after a microdiagnostic test detects a fault. Only one of the above arguments may be specified; they are mutually exclusive.

/FAULT:ISOLATE

This setting, which is also the default, directs DCP to attempt isolation on all solid faults encountered. After the isolation attempt is completed, DCP returns to its command prompt. The CONTINUE or STEP commands can be used to resume test execution.

/FAULT:NOABORT

This setting directs DCP to attempt isolation on all solid faults encountered. After the isolation attempt is completed, DCP will continue onto the next test, thus providing a way to continue with the diagnostic execution after isolating.

/FAULT:LOOP

Tests within a diagnostic are targeted to run /PASSES times. When an error is detected, the standard error report is displayed and then DSM is instructed to loop forever on the failing test. Only <CTRL/C> or <CTRL/P> can terminate the loop.

/FAULT:PAUSE

This setting causes DCP to return to its command prompt after reporting a test failure. The CONTINUE or STEP command can be used to resume test execution.

/FAULT:CONTINUE

With this switch in effect, DCP will perform all of its normal error reporting. After each error report, DCP will continue its normal test sequencing as if no error had occurred. Isolation will not take place.

/FAULT:IGNORE

This is a special switch intended only for microcoder use. It has no useful application in the field. It directs DSM never to signal DCP that an error has occurred. No hardware errors can ever be detected or reported.

/LINES:{ ON, OFF } default = OFF

This switch is used by microcoders to debug isolation code. If the switch has been set to ON and DCP is processing isolation statements, a source line number will be displayed as the statement is processed. This line number matches the line number of the isolation source statement that is currently being processed.

/PRINT_MODE:{ BRIEF, VERBOSE } default = VERBOSE

This switch controls the level of detail included in an error report.

5.8.14 SHOW CONFIGURATION (Arrays, SBIAs, FBox)

Syntax: SHOW CONFIGURATION

Description:

This command will display, in tabular form, information regarding the hardware configuration and selected for test status for arrays, SBIAs, and the FBox. The table will indicate:

- Which slots currently contain arrays, and their sizes and current status (selected or excluded from testing)
- Which SBIAs are physically present and their current status (selected or excluded from testing)
- If an FBox is present and its current status (selected or excluded from testing)

The physical presence of these devices is determined with the CONFIGURE command. The choice of whether a particular option is selected or excluded from testing is made via the SELECT and DESELECT commands.

5.8.15 SHOW DATA

Syntax: SHOW DATA

Description:

This command displays all symbolic names and associated data currently defined by the SET DATA command. The range of diagnostic test numbers, with the number of passes, is also displayed under this command for the user's information.

5.8.16 SHOW SWITCHES

Syntax: SHOW SWITCHES

Description:

This command displays the current switch settings and the default switch settings. The default settings are either those settings that were in place when DCP was loaded or those that have been modified via the SET SWITCH command.

The current switch settings are normally the same as the default settings since local settings (i.e., appending switches to RUN or START) return to defaults when the command completes.

The current switches will differ from the defaults when they are examined from within another command. This could occur by examining the switches from the paused state of a RUN or START command which uses the /SINGLE_STEP switch. When proceeding from that point with the CONTINUE or STEP command, the current switch settings will remain in effect until the initial RUN or START command completes.

5.8.17 START

Syntax: START [/switch]

"/switch" can be any combination of the following switches, provided that the command line does not exceed 80 characters. Switches appended to the START command remain in effect until the command completes. If a START command is given without switches, their default values are used.

```
/BELL:{ ON, OFF }
/FAULT:{ ISOLATE, NOABORT, LOOP, PAUSE, CONTINUE, IGNORE }
/LINES:{ ON, OFF }
/NUMBER:FIRST_TEST [ {<space>,<comma>} LAST_TEST ]
/PASSES:dec_num
/PRINT_MODE:{ BRIEF, VERBOSE }
/SINGLE_STEP
```

Description:

This command initiates the execution of a test or group of tests within an already loaded microdiagnostic program. Execution begins with the first number specified in the /NUMBER switch. It ends with either the last test in the group of microdiagnostic tests or with the last test specified in the /NUMBER switch. Note that the CPU clock must be running for the START command to work.

Refer to the description of the SET SWITCH command for information on the switches not described here.

/NUMBER:FIRST_TEST,[{<space>,<comma>}LAST_TEST] default = 01,FF

> This switch is used to specify the starting and ending test numbers in hex. If the switch is omitted, the default values are used.

/PASSES:'dec_num' default = 100

The argument "dec_num" indicates the number of test passes to execute before continuing to the next test or attempting isolation. If the /PASSES switch is not specified, the default value will be used. A special case of /PASSES:0 modifies the sequencing of the tests so that a particular group of tests can be run indefinitely. If the user specifies a 0 pass count, DCP will run each of the tests indefinitely (from FIRST to LAST) until <CTRL/C> or <CTRL/P> interrupts or a fault is detected. When a fault is detected, the action will be governed by the current setting of the /FAULT switch.

/SINGLE_STEP

This switch enables the single_step feature. When specified, diagnostic execution is interrupted after completing the proper number of passes of an individual test. A message, along with the test number just completed, is displayed and the operator is prompted for input. The STEP command can be used to run the next test and is followed by another pause when it completes. The CONTINUE command can be used to clear this mode and resume full speed test execution.

5.8.18 STEP

Syntax: STEP

Description:

This command causes the next test in the currently loaded microdiagnostic to be executed. The STEP command is invalid unless the microdiagnostics have been started. Once they have been started and there is a pause in the diagnostic execution (/FAULT:PAUSE, /FAULT:ISOLATE, or <CTRL/P>), the STEP command may be used. A STEP command issued prior to giving either a RUN or START command will result in an error message.

CHAPTER 6
MACRODIAGNOSTICS

6.1 OVERVIEW

Macrodiagnostics are functional macroprograms based in the system's internal memory subsystem. They may be loaded from either the RL02 via the console front-end or from the system disk via the ABus. A limited set of macrodiagnostic programs resides on the RL02 and is used to test the load paths required for loading macroprograms from the system disk and magnetic tape devices. Part of the system installation procedure requires building the system disk from magnetic tape; the RL02 macrodiagnostics permit testing and verifying the operation of the hardware used during the disk building process.

Macrodiagnostics differ from microdiagnostics in several ways. Table 6-1 outlines the differences between the two types of test strategies.

Table 6-1 Micro/Macro Diagnostic Differences

Microdiagnostics	Macrodiagnostics
Nonfunctional tests	Functional tests
Reports failing modules and components	Reports failing function only
Resides in the EBox CS	Resides in VAX internal memory
Operates in CIO mode	Operates in PIO Mode
Accesses SDB visibility channel	No access to SDB visibility channel
No access to I/O devices	Accesses and tests I/O devices
Only runs standalone	Can be run on-line
Runs under control of the Diagnostic Control Program (DCP)	Runs under control of EDSAA, the VAX diagnostic supervisor

6.2 MACRO CONTEXT TEST ENVIRONMENT

To run macrodiagnostics, the user must switch to the MACRO context by typing the MACRO command. This is the default context during normal power-up initialization, as indicated by the >>> prompt on the console terminal. Figure 6-1 shows the state of the system when operating in the macro context. Note that system microcode must be loaded into all the CPU control store RAMs and that the actual macrodiagnostics reside in internal memory.

6.2.1 Macro Context Initialization

Macro context initialization may be aborted at any time by the <CTRL/C> or <CTRL/P> characters. Aborting macro context initialization is useful if the user wants to switch immediately to diagnostic or microhardcore context, where context-specific initialization is done automatically. The console software executes the following sequence of events.

1. Prints "Initializing CPU" message to indicate the start of this section.

2. Submits for execution the command file LOAD.COM, which performs a complete initialization of the CPU for use as a macrocode processor. The steps taken by the LOAD.COM file are described in "CPU Initialization" (Paragraph 6.2.2).

3. If this is the first initialization after a system power-up and the battery backup unit was inactive on entry, or if this is a context switch coming from diagnostic or microhardcore context, then automatic UNJAM and CLEAR MEMORY commands are performed.

4. If the Terminal Control Switch is in the LOCAL DISABLE position, or if the Restart Control Switch is in the RESTART BOOT or RESTART HALT position, then attempt a warm-start. Otherwise, go to the next step. The steps involved in a warm-start attempt follow.

   - Prints "Attempting System Restart" message.

   - Tests that the BBU was valid at the time power was restored (flag saved during console program initialization). If not, the warm-start attempt has failed (go to the next step).

   - Tests if the WARM flag is set. If set, the warm-start attempt has failed (go to the next step).

   - Commands CSM to search for a valid Restart Parameter Block. If not found, the warm-start attempt has failed (go to the next step).

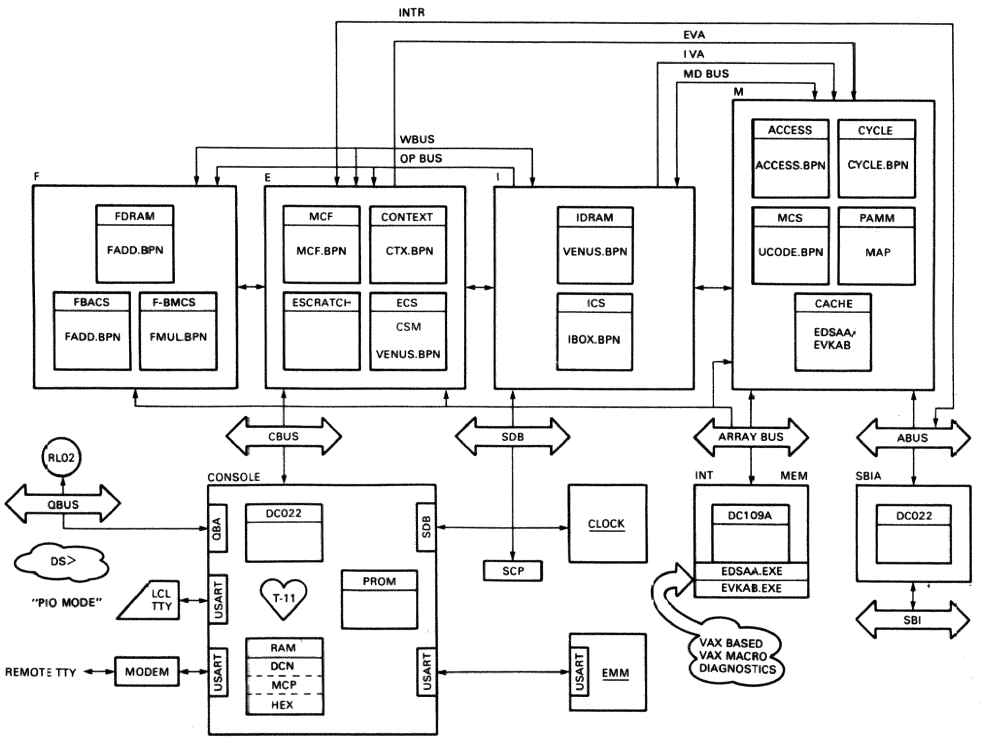


Figure 6-1 MACRO Context Test Environment

- Loads VAX processor GPRs with applicable data, as per Digital Standard 032, and sets the WARM flag.
- Starts the VAX processor running at its restart address (located in the RPB).
- Enters PIO mode.

5. If the restart attempt failed and the Terminal Control Switch is in the LOCAL DISABLE position, or if the Restart Control Switch is in the RESTART BOOT or BOOT position, then attempt a cold-start. Otherwise, go to the next step.
   - Prints "Attempting System Bootstrap" message.
   - Tests if the COLD flag is set. If set, the cold-start attempt has failed (go to the next step).
   - Loads VAX processor GPRs with applicable data, as per Digital Standard 032, and set the COLD flag.
   - Submits the BOOT command, which in turn submits the file DEFBOO.COM for execution. A sample of the BOOT command file is shown in Figure 6-12.
6. If the boot attempt failed or if the Restart Control Switch is in the HALT position, then enter the CIO null loop, macro context, and prompt for operator commands.

6.2.2 CPU Initialization

CPU initialization is performed by the console as part of the macro context initialization or whenever the command file LOAD.COM is invoked. The steps the LOAD.COM file performs are described next.

1. The RESET command is entered to perform a master reset of the machine.
2. The INIT/POWER command is used to initialize and/or ensure that the power system is fully operational.
3. The INIT/SDB command is used to force all SDB control channels to a known state to allow microcode loading.
4. The INIT/MICRO command is used to load all control stores with the default system microcode. INIT/MICRO submits the file ULOAD.COM for execution.
5. The INIT/SDB command is again used to force all SDB control channels to a NOP or RUN state.

6. The INIT/CPU command is used to initialize CSM and many of the CPU's registers and IPRs.

7. Lastly, the PAMM is initialized using the INIT/PAMM command.

At this point, the CPU can execute VAX macrocode.

Figure 6-2 shows the typical console response to the MACRO command. The example was executed with the QUIET flag off so that the user sees the commands invoked from within the LOAD.COM and ULOAD.COM files.

6.3 MACRODIAGNOSTIC SEQUENCING

Most all of the VAX macrodiagnostics are designed to run under control of the VAX diagnostic supervisor (EDSAA), which can run either standalone or on-line under VMS. This section will only discuss standalone testing. The recommended sequence is as follows.

1. Load and run EVKAA, the VAX family macrohardcore diagnostic, to verify the macro instructions used by the diagnostic supervisor.

2. Load and run EDSAA to allow loading and running of the rest of the diagnostics.

3. Load and run EVSBA, the VAX generic autosizer, to size the system and set up the system configuration tables that EDSAA requires.

4. Load and run the CPU cluster diagnostics.

5. Load and run the I/O adapter diagnostics.

6. Load and run the system-specific I/O device diagnostics in order to test the load path to the system disk.

6.3.1 EVKAA -- Macrohardcore Diagnostic

Figure 6-3 shows the typical procedure for running EVKAA. It uses the LOAD EVKAA command to load the program into VAX internal memory from the RL02, followed by the START 200 command to start the program at location 200(X). It will run one quick pass, display an END OF PASS message, and then continue to run complete passes until halted with a <CTRL/P>. Note that it will signal that it's still alive every A0(X) passes.

EVKAA may be loaded at some address other than location 0 by typing "LOAD/START:xxxxxxxx EVKAA." Remember that if this is done, the program must start at location "xxxxxxxx+200." This feature is useful if the user suspects memory module 0 is causing a problem.

```
DC>>SET QUIET OFF
DC>>MACRO
>>>! CLOCK.COM - init clock and define clock parameters
>>>! Version: 001.000
>>>! Released: 21-May-1985
>>>!
>>>SET CLOCK X1 40                ! Assign mnemonics for rev E01/F01 clock modules
>>>SET CLOCK X2 50/NORMAL
>>>SET CLOCK X3 53/HIGH
>>>SET CLOCK X5 56
>>>SET CLOCK FREQ NORMAL          ! Force normal clock operation
>>>SET CLOCK FULL                 ! Full speed
>>>SET CLOCK DEFAULT     !...and make it stick
>>>INIT/CLOCK            ! Initialize clock logic and do 141 reset
Initializing CPU
>>>! LOAD.COM - Full machine init file for macro context
>>>! Version: 001.002
>>>! Released: 14-Mar-1985
>>>!
>>>RESET                          ! Stop the CPU clock.
>>>INIT/POWER            ! Check for solid power system.
>>>INIT/SDB              ! Init control channels before microcode load.
>>>INIT/MICRO            ! Load all control stores.
>>>! ULOAD.COM - Called by INIT/MICRO command to load control stores
>>>! Version: 002.000
>>>! Released: 30-May-1985
>>>!
>>>SET EXTI OFF          ! Turn off CPU-To-Console interrupts
>>>LOAD/ECS              ! Load KA86AA.BPN
>>>LOAD/MCF              ! Load MCF.BPN
>>>LOAD/CONTEXT          ! Load CTX.BPN
>>>LOAD/ICS              ! Load IBOX.BPN
>>>LOAD/IDRAM            ! Load KA86AA.BPN
>>>LOAD/MCS              ! Load UCODE.BPN
>>>LOAD/ACCESS           ! Load ACCESS.BPN
>>>LOAD/CYCLE            ! Load CYCLE.BPN
>>>LOAD/FBA              ! Load FADD.BPN
>>>LOAD/FBM              ! Load FMUL.BPN
>>>LOAD/FDRAM            ! Load FADD.BPN
>>>INIT/SDB              ! Init control channels again.
>>>INIT/CPU              ! Init CPU/CSM/Escratch.
>>>INIT/PAMM             ! Init PAMM.
  Total memory available is 24. Megabytes
        Array Slot 1 contains 4 Megabytes
        Array Slot 2 contains 4 Megabytes
        Array Slot 3 contains 4 Megabytes
        Array Slot 4 contains 4 Megabytes
        Array Slot 5 contains 4 Megabytes
        Array Slot 6 contains 4 Megabytes

        IOA0 is an SBIA (Rev level = 0)
        IOA1 not present
        IOA2 not present
        IOA3 not present

>>>
```

Figure 6-2 Macro Context Initialization

```
>>>LOAD EVKAA
>>>
>>>START 200
                VAX DIAGNOSTIC SOFTWARE
                        PROPERTY OF
             DIGITAL EQUIPMENT CORPORATION
           ***CONFIDENTIAL AND PROPRIETARY***
Use Authorized Only Pursuant to a Valid Right-to-Use License
EVKAA V8.0 Hardcore Instruction Test
Hit any key to continue
EVKAA V8.0 pass # 1(X) done!
EVKAA V8.0 pass # A0(X) done!
?MCP-I-CPSRUN, CPU is still running
>>>HALT
        CPU stopped, INVOKED BY CONSOLE (CSM code 11)
        PC 0000B7CD
>>>
```

Figure 6-3 EVKAA Display

If any errors are detected, the failing test number and good/bad data will be displayed. Two examples are shown below.

Example 1

```
??? ERROR Test #nn, Subtest #nn (instr) failed
brief description of failing function
Expected data - xxxxxxxx
Received data - xxxxxxxx
```

Example 2

```
??? ERROR Test #nn, Subtest #nn, Subsubtest #nn (instr) failed
brief description of failing function
Expected data - xxxxxxxx
Received data - xxxxxxxx
```

Where:

"nn"	is the failing test, subtest, and subsubtest numbers
instr	"ADDL (R), R" for example
xxxxxxxx	Hex data (good/bad)
description	"not incrementing properly," for example

After displaying the error, EVKAA will halt. At this point, consult the listing to locate the failing test from which you can determine the address of the HALT instruction. Each HALT instruction is followed by an unconditional branch back to the start of the failing test. Simply deposit a NOP (01) in place of the HALT (00) and type START 200 to restart the program. Now, after the failure is detected again, the program enters a tight loop to permit manual analysis. Refer to Chapter 7 for detailed procedures on single stepping and single clocking a failing macro routine. Obviously, this requires knowing the EBox microsequencer in detail, and knowing how to read the microcode listings. Such information is beyond the scope of this manual.

If any errors are detected by EVKAA, the fault should be corrected before attempting to continue. Trying to load and run EDSAA may result in errors that could be misleading, since EDSAA assumes that EVKAA ran error-free.

6.3.2 EDSAA -- VAX 8600/8650 Diagnostic Supervisor

Once EVKAA has run successfully, the next step is to load and start the supervisor. Typing the command @EDSAA will read the command file from the RL02, which will load and start EDSAA. Figure 6-4 shows the typical console response to this command. To see the commands within EDSAA.COM, simply set the QUIET flag off before typing the command. Once started, EDSAA displays the standard disclaimer, program title, and revision information. It then returns the DS> prompt to indicate it is ready to accept a command. Figure 6-5 shows the commands contained within EDSAA.COM.

At this point, the console is operating in PIO mode and any input typed will be passed to EDSAA for processing. By the same token, any output from EDSAA will be passed on to the user's terminal by the console software in the T-11.

This manual assumes the reader knows what to do from here. In fact, most of the operation from this point on is generic and works just like any other VAX system. Refer to the VAX Diagnostic Supervisor User's Guide (EK-VXDSU-UG) for additional information about operating the diagnostic supervisor. There is an extensive on-line HELP facility in the diagnostic supervisor. Simply type HELP at the DS> prompt and follow the directions. The HELP DEV command is very useful for obtaining information about the VAX macrodiagnostics available. Figures 6-6 through 6-8 show three examples of how to use this command. Note how the information displayed describes how to attach the device for testing.

Now let's continue with the job of actually running macrodiagnostics.

```
>>>@EDSAA

                        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,  1985.  All Rights Reserved.
DIAGNOSTIC SUPERVISOR.  ZZ-EDSAA-Y8.2-214  20-AUG-1985 11:00:40
DS>
```

Figure 6-4 EDSAA Start-Up Display

```
>>>SHOW EDSAA.COM/ASCII
!
! Version: 001.000
! Load and start the diagnostic supervisor for Venus
!
INIT                              ! Do basic initialization
SET SNAP OFF                      ! Don't take snapshots
DEP CSWP 8                        ! Turn off cache
CLEAR MEMORY                      ! Clear blue sky
LOAD/START:0FE00 EDSAA  ! Load supervisor
DEP CSWP B                        ! Turn on the cache
UNJAM                       ! Unjam all existant sbia's
INIT/CPU                          ! Init the CPU
START 10000                       ! Start supervisor
>>>
```

Figure 6-5 EDSAA.COM Command File

```
DS> HELP DEV KA86
  Devices
    KA86
       Description:   KA86 8600 central processor
       link:      HUB
       Generic name:  KAn
       Additional information:
          G-floating instructions? [Yes/No]
          H-floating instructions? [Yes/No]
          WCS last address? [hex 0-FFFF]
          Accelerator type? [decimal 0-255]
       Tested by: EVKAA, EVKAB, EVKAC, EVKAD, EVKAE,
             EVXBB
DS>
```

Figure 6-6 HELP DEV Command -- Display 1

```
DS> HELP DEV
  Devices
     As well as the "device-independent" parameters required by the
     ATTACH command for any device, most devices require device-
     dependent information.  Type "HELP DEVICE device-designator" for
     information on a specific device, or "HELP DEVICE..." for a list
     of all devices' requirements.  If the device is not listed here,
     it is known only within the diagnostic program(s) that use it;
     type "HELP program DEVICE" for information on it (e.g., "HELP
     EVRAA DEVICE").

     Device parameters include the legal range of values in square
     brackets (e.g., "[decimal 4-7]" for BR level), and device
     standard values in angle brackets (e.g., "<776750>" for the AA11
     CSR).
  Additional information available:
  AA11K           AD11K           CI750           CI780           CI_NODE
  CR11            DHU11           DISK            DMC11           DMF32A
  DMF32P          DMF32S          DMP11           DMR11           DMZ32
  DN11            DR11E           DR11C           DR11K           DR11W
  DR750           DR780           DUP11           DW730           DW750
  DW780           DZ11            DZ32            IEU11A          KA730
  KA750           KA780           KA86            KMC11           KMS11
  KMSYG           KW11K           LA34            LA36            LA38
  LA120           LA180           LN01            LP04            LP05
  LP06            LP11            LP14            LP25            LPA11K
  MA780           MBE             ML11            MS750           MS780
  PCL11           R80             RA60            RA80            RA81
  RB730           LESI            RC25            RCF25           RH750
  RH780           RK06            RK07            RK611           RL01
  RL02            RL11            RM03            RM05            RM80
  RP04            RP05            RP06            RP07            RUX50
  RX02            RX180           RX31            RX32            RX50
  RX211           SBIA            TE16            TM03            TM78
  TS11            TU45            TU58            TU77            TU78
  TU80            TU81            UBE             UDA50           UNA11
  VT50            VT52            VT55            VT100           VT101
  VT102           VT125           VT220           VT240
DS>
```

Figure 6-7 HELP DEV Command -- Display 2

```
DS> HELP DEV SBIA
  Devices
    SBIA
       Description:   SBIA Synchronous Backplane Interconnect Adapter
       link:      HUB
       Generic name:  SIn
       Additional information:
          The VAX 8600 supports a dual-sbi configuration. Each
          SBI that exists must be attached to HUB as follows:

               for SBIA0 "ATTACH SBIA HUB SI0"
               for SBIA1 "ATTACH SBIA HUB SI1"

          Next, any adapter such as DW780, DR780, RH780, or CI780,
          is attached to either SI0, or SI1.

       Special cases: DW780s should be attached to an 8600 as follows:

               to SI0: DW0, 1, 2, or 3
               to SI1: DW4, 5, 6, or 7.

       WARNING: DW0 and DW4 should not be attached to the same SBIA since
          they translate to the same Unibus address space. Likewise for
          DW1 and 5, DW2 and 6, and DW3 and 7.
       Tested by: EDCLA
DS>
```

Figure 6-8 HELP DEV Command -- Display 3

6.3.3 Macrodiagnostics

Before EDSAA can load and run any VAX macrodiagnostics, it needs to know something about the specific configuration. At this point, all we have is a very smart control program, EDSAA, running in the VAX CPU, but it knows nothing about the system configuration. To illustrate this, try typing the SHOW DEVICE command. Probably nothing much was displayed.

It's the job of EVSBA, a special VAX macroprogram running under the diagnostic supervisor, to "size" the system. EVSBA finds out what's there and sets up a configuration table within EDSAA that establishes device-specific information (names, addresses, TRs, etc.) that are required to run macrodiagnostics.

Figure 6-9 shows a typical response to the command that loads and runs EVSBA. Setting the QUICK flag prior to invocation shortens run time by skipping code that sizes all terminal lines. After completion, EVSBA returns to the DS> prompt, and the SEL ALL and SHOW SEL commands will instruct the supervisor to display a list of the devices found. Only devices shown in this list can be tested by the appropriate diagnostics. Let's look at a few examples.

```
DS> RUN EVSBA
.. Program: EVSBA - AUTOSIZER  LEVEL 3, revision 2.1, 3 tests,
   at 11:13:09.74.
.. End of run, 0 errors detected, pass count is 1,
   time is 20-AUG-1985 11:13:23.38
DS>
```

Figure 6-9 EVSBA Autosizer Display

Figure 6-10 shows the normal response to a command to run two passes of the basic instruction exerciser macrodiagnostic, EVKAB. Note how it was first run with the FBox OFF and then with the FBox ON.

```
DS> SELECT ALL
DS> RUN/PASS:1 EVKAB
.. Program: ZZ-EVKAB, VAX Basic Inst's Exrczr, revision 3.4, 161 tests,
   at 11:15:00.68.
Testing: _KA0
Accelerator type 1 is disabled.
Accelerator type 1 is enabled.
.. End of run, 0 errors detected, pass count is 1,
   time is 20-AUG-1985 11:16:43.46
DS>
```

Figure 6-10 EVKAB Display

Figure 6-11 shows the normal response to the command to run the DW780 repair diagnostic (EVCBA).

```
DS> RUN/PASS:1 EVCBA
.. Program: ZZ - EVCBA - DW 780 REPAIR DIAGNOSTIC, revision 1.1, 54 tests,
   at 11:17:48.74.
Testing: _DW0    DW2
                 UBA UNIT UNDER TEST IS _DW0
 NOTE -  "UBE" IS NOT IN THE SYSTEM   HENCE NOT TESTING IT
                 UBA UNIT UNDER TEST IS _DW2
 NOTE -  "UBE" IS NOT IN THE SYSTEM   HENCE NOT TESTING IT
.. End of run, 0 errors detected, pass count is 1,
   time is 20-AUG-1985 11:19:10.24
DS>
```

Figure 6-11 EVCBA Display

6.3.4 RL02 Resident Macrodiagnostics

There is a minimum set of VAX macrodiagnostics resident on the RL02 disk pack for testing the VAX 8600 and VAX 8650 CPU and system load paths. Table 6-2 lists these macrodiagnostics.

These macrodiagnostics, along with the rest of the VAX system macrodiagnostics, will be installed in the [SYSMAINT] account on the system disk to permit running on-line macrodiagnostics. Part of the system installation involves creating this account and copying the programs from magnetic tape to the system disk.

Table 6-2 RL02 Resident VAX Macrodiagnostics

Name	Description
EDCLA.EXE	DW780, CI780, DR780, RH780 SBI Exerciser
EDSAA.EXE	VAX 8600 Diagnostic Supervisor
EDKAB.EXE	VAX Basic Instruction Exerciser
EVCAA.EXE	RH780 Diagnostic
EVCBA.EXE	DW780 Repair Diagnostic
EVCGA.EXE	CI780 Repair Level Diagnostic Part 1
EVCGB.EXE	CI780 Repair Level Diagnostic Part 2
EVCGC.EXE	CI780 Repair Level Diagnostic Part 3
EVCGD.EXE	CI780 Repair Level Diagnostic Part 4
EVGAA.EXE	CI780 Functional Diagnostic Part 1
EVGAB.EXE	CI780 Functional Diagnostic Part 2
EVKAA.EXE	VAX Hardcore Instruction Test
EVKAB.EXE	VAX Basic Instruction Exerciser
EVKAC.EXE	VAX Floating-Point Exerciser
EVKAD.EXE	VAX Compatibility Mode Instructions Exerciser
EVKAE.EXE	VAX Privileged Architecture Exerciser
EVMAA.EXE	VAX Generic Tape Exerciser
EVMAB.EXE	TM03-TE16/TU45/TU77 Function Timing Tests
EVMAC.EXE	TM03-TE16/TU45/TU77 Control Logic Tests
EVMAE.EXE	VAX TM78/TU78 Control Logic Diagnostic
EVMBB.EXE	TU81 Front End/Host Functional Diagnostic
EVMBD.EXE	VAX TU80 Functional Diagnostic Part 1
EVMBE.EXE	VAX TU80 Functional Diagnostic Part 2
EVQTF.EXE	TM78 Loadable Driver
EVQTM.EXE	TM03 TE16/TU77 Loadable Driver
EVQTS.EXE	VAX TS11 Standalone Driver
EVQUE.EXE	VAX UBE Driver/UBE QIO Driver
EVRLA.EXE	VAX UDA and RA Drive Diagnostic
EVRLB.EXE	VAX RA60/RA80/RA81 Formatter
EVSBA.EXE	Autosizer Level 3
EDKAX.EXE	VAX 8600 CPU Cluster Exerciser

6.3.5 Booting the Operating System

Assuming that all the macrodiagnostics ran error-free, it should be possible to bring up the operating system. The BOOT command executes the commands found in the file DEFBOO.COM on the RL02. Refer to the example in Figure 6-12.

The command file is system-specific and performs the following VMS loading sequence.

1. Deposit information into the VAX CPU's GPRs to specify the system load device.

2. Find the first 64 Kbytes of good memory.

3. Load VMB.EXE from the RL02 into VAX internal memory using the address of the good memory.

4. Start VMB, which will find, load, and start SYSBOOT.EXE from the system disk.

5. Once started, SYSBOOT will load and start the VMS kernel.

6. VMS then executes its start-up command procedures and eventually returns the Username: prompt to signal it's "on the air" and ready to accept logins.

Figure 6-13 shows a typical example of booting VMS and logging in to the field service account, [SYSMAINT]. After logging in, the diagnostic supervisor may be run on-line to load and run on-line diagnostics. Figure 6-14 shows an example of running macrodiagnostics on-line.

6.4 SHUTTING DOWN VMS

The results from running on-line diagnostics may indicate a need to bring the system down and run standalone diagnostics. To ensure an orderly shutdown of the system, the system-specific procedure should be run as shown in Figure 6-15. After the procedure terminates, the user presses **<CTRL/P>** to return to CIO mode, where the console responds with the macro context prompt, >>>. The user must type the HALT command to halt the CPU.

Once back at the >>> prompt, the user can either run standalone macrodiagnostics, or switch to diagnostic or microhardcore context to run either microdiagnostics (Chapter 5) or microhardcore (Chapter 4).

6.5 SUMMARY

VAX macrodiagnostics provide functional testing of the VAX CPU, and I/O controllers and devices. They may be run standalone or on-line, generally under the control of EDSAA, the VAX 8600 diagnostic supervisor. Running the macrodiagnostics is the final step in the bottom-up test procedure. Chapter 7 will conclude with a discussion of how to use the console command sets to perform manual testing of the CPU.

```
>>>!    CI PORT BOOT COMMAND FILE - CIBOO.COM
>>>!
>>>!    This CI port boot command file is set up to boot from a CI
>>>!    device; for example, a HSC based disk.
>>>!
>>>!    It assumes the CI780 is on SBIA #0, the TR level of the CI780
>>>!    is set to 14, the HSC node number is set to 15 and the disk's
>>>!    unit number is 0.
>>>!
>>>!    If any of these assumptions are not true for your configuration,
>>>!    you may still use this command file by entering the BOOT/NOSTART
>>>!    console command and then altering the appropriate register values
>>>!    when the console command prompt reappears. Use the console
>>>!    command SHOW BOOT.HLP/ASCII to get more information on how to
>>>!    use the BOOT/NOSTART command and R5 boot options.
>>>!
>>>!
>>>! Operating System Disk:     CI DEVICE
>>>!
>>>!
>>>SET SNAP ON           ! Enable ERROR_HALT snapshots
>>>SET FBOX OFF          ! VMS will turn on Fbox
>>>INIT                  ! SRM processor init
>>>UNJAM                       ! UNJAM SBIAs, Enable Master SBI interrupts
>>>DEPOSIT CSWP 8              ! Turn off the cache (VMS turns the cache on)
>>>!
>>>DEPOSIT R0 20               ! Device Type is CI780
>>>DEPOSIT R1 E          ! SBIA #0; TR number of the CI780 is 14
>>>DEPOSIT R2 F          ! HSC port number 15
>>>DEPOSIT R3 0          ! Unit number to boot from (in HEX)
>>>DEPOSIT R4 0          ! Logical block number to boot from if R5 bit 3 is set
>>>                      ! Use R5 for optional boot control flags
>>>FIND/MEMORY           ! Locate a 64KB chunk of good memory
>>>EXAMINE SP            ! Display load address
>>>LOAD/START:@ VMB      ! Load VMB 200 bytes above the start of the good block
>>>START @               ! Start VMB at the load address
```

Figure 6-12 Sample DEFBOO.COM Command File

```
>>>BOOT

        G 0E 00000200

   VAX/VMS Version X4.1 27-NOV-1984 22:29
PLEASE ENTER DATE AND TIME (DD-MMM-YYYY  HH:MM)  29-AUG-1985
Username: FIELD
Password:
        Welcome to VAX/VMS version X4.1 on node SLIMY
    Last interactive login on Tuesday, 27-AUG-1985 13:32
    Last non-interactive login on Wednesday,  5-JUN-1985 16:34

$
```

Figure 6-13 VMS Bootstrap Display

```
$ RUN EDSAA
                VAX DIAGNOSTIC SOFTWARE
                      PROPERTY OF
              DIGITAL EQUIPMENT CORPORATION
            ***CONFIDENTIAL AND PROPRIETARY***
Use Authorized Only Pursuant to a Valid Right-to-Use License
DIAGNOSTIC SUPERVISOR.  ZZ-EDSAA- 8.0-203  29-AUG-1985 09:59:00
DS> @CONFIG
DS> SEL KA0
DS> RUN/PASS:2 EVKAB
.. Program: ZZ-EVKAB, VAX Basic Inst's Exrczr, revision 3.3, 161 tests,
   at 10:00:11.16.
Testing: _KA0
.. First pass done, 0 errors detected, time is 29-AUG-1985 10:01:21.68
.. End of run, 0 errors detected, pass count is 2,
   time is 29-AUG-1985 10:02:31.80
DS>^C
$
```

Figure 6-14 On-Line Diagnostics Display

```
$ SHOW USERS
          VAX/VMS Interactive Users
           29-AUG-1985 10:06:57.04
    Total number of interactive users = 1
 Username .   Process Name       PID     Terminal
 FIELD        FIELD            0000010D  OPA1:
$
$ @SYS$SYSTEM:SHUTDOWN
        SHUTDOWN -- Perform an Orderly System Shutdown
How many minutes until final shutdown [0]:
Reason for shutdown [Standalone]:
Do you want to spin down the disk volumes [NO]?
Do you want to invoke the site-specific shutdown procedure [YES]?
Should an automatic system reboot be performed [NO]?
When will the system be rebooted [later]:
Shutdown options (enter as a comma-separated list):
 REBOOT_CHECK        Check existence of basic system files
Shutdown options [NONE]:
        .
        .
        .
        System shutdown messages
                .
                .
                .
         SYSTEM SHUTDOWN COMPLETE - USE CONSOLE TO HALT SYSTEM
^P
?MCP-I-CPSRUN, CPU is still running
>>>HALT

         CPU Stopped, INVOKED BY CONSOLE (CSM code 11)
         PC 80008D7E
>>>
```

Figure 6-15 VMS Shutdown Display

CHAPTER 7
CONSOLE COMMAND SETS

7.1 INTRODUCTION

This chapter discusses the creative use of the console commands during corrective maintenance. Certainly, there are going to be situations where the user must go beyond the level of running diagnostics as described in the last five chapters. The user must know how to select and sequence specific sets of commands to modify the test environment, and retrieve additional isolation information not provided by the diagnostics.

There are over 100 commands divided into seven different groups, shown below.

1. The PROM Commands
2. The EDOBA Commands
3. The General Command Set
4. The Microhardcore Command Set
5. The Macro Command Set
6. The Diagnostic Context Command Set
7. The HEX Debugger Command Set

How does the user pick and choose the commands needed to perform a particular function? The following sections demonstrate typical examples of how to select the right command set for the maintenance function required. This material is organized in the same sequence as the preceding five chapters, beginning with the PROM commands and working up to the macrodiagnostics.

7.2 PROM COMMAND SET

Once at the ROM> prompt, the user has access to a limited command set that permits some manual testing of the console module. The following commands may prove useful in gathering additional troubleshooting information related to solid console hardware problems.

1. D -- deposit data into T-11 address space
2. E -- examine data in T-11 address space
3. Q -- deposit data into the Q-Bus registers for the RL02
4. R -- examine data in the Q-Bus RL02 registers
5. S -- start a T-11 routine deposited into T-11 RAM

Figure 7-1 shows several examples of how to use deposit/examine sequences to test the console. Note that locations 000 through 156 in T-11 RAM cannot be altered by the DEPOSIT command. These locations contain system vectors that are protected by the PROM code. It is important to remember that depositing short T-11 test routines in the RAM and starting them with the S command is of limited value, since the user has no control of the routine after it starts running. The only way to get out is to power down the system and power back up again to get back to the PROM command interpreter.

In most cases, replacing the console module will resolve console hardware problems. If this does not work, the user may need to use the console commands to perform further isolation.

```
>>>PROM
   PROM restart -- confirm (Y/N) Y

?T11 HLT
?REGS 140210 000102 000211 040346 045126 045706 000740 000162 000000
ROM>E 150
000150 /172010

ROM>D 150 0

ROM>E 150
000150 /172010

ROM>E 300
000300 /012345

ROM>D 300 0

ROM>E 300
003000 /000000

ROM>R 174400
174400 /000205

ROM>E 174402
174402 /040764

ROM>Q 174402 2

ROM>E 174402
174402 /000002

ROM>
```

Figure 7-1 PROM Command Examples

7.3 ED0BA DIAGNOSTIC

ED0BA provides a set of program switches that permit the user to modify the execution of the program during troubleshooting. These switches, along with the control keys <CTRL/G>, <CTRL/P>, <CTRL/C>, <CTRL/S>, and <CTRL/Q>, can be used to perform the following diagnostic functions.

1. Exit to the ROM> prompt
2. Restart ED0BA at location 200(8)
3. Stop and start the output display
4. Exit subtest error loops
5. Loop on a failing subtest
6. Inhibit error output
7. Trace program flow by displaying test headers
8. Inhibit test iterations to speed up execution
9. Loop on any selected test number
10. Exit to the switch register prompt at the end of the current test

Figure 7-2 illustrates how to use the TRACE option and Figure 7-3 shows how to loop on a selected test.

```
ROM>T

ED0BA V35 (25-Jun-1985)

          CONFIDENTIAL DIAGNOSTIC SOFTWARE
                PROPERTY OF
          DIGITAL EQUIPMENT CORPORATION

   Use Authorized Only Pursuant to a Valid Right-to-Use License

    0 ERRORS DETECTED;      0 TOTAL ERRORS AFTER PASS # 1
    0 ERRORS DETECTED;      0 TOTAL ERRORS AFTER PASS # 2
ROM>S 200


ED0BA V35 (25-Jun-1985)

SWR = 000000  NEW = 100000

BEGIN TEST #000001
BEGIN TEST #000002
BEGIN TEST #000003
BEGIN TEST #000004
BEGIN TEST #000005
BEGIN TEST #000006
BEGIN TEST #000007
IGNORE TEST #000010
IGNORE TEST #000011
   .
   .
   .
   .
IGNORE TEST #000036
IGNORE TEST #000037
BEGIN TEST #000040
BEGIN TEST #000041
IGNORE TEST #00042
BEGIN TEST #000043^G

SWR = 010000 NEW = ^P

ROM>
```

Figure 7-2 ED0BA Trace Switch Option

```
ROM>S 200

EDOBA V35 (25-Jun-1985)

SWR = 000000  NEW = 11043

BEGIN TEST #000001
BEGIN TEST #000002
BEGIN TEST #000003
BEGIN TEST #000004
BEGIN TEST #000005
BEGIN TEST #000006
BEGIN TEST #000007
IGNORE TEST #000010
IGNORE TEST #000011
   .
   .
   .
   .
IGNORE TEST #000036
IGNORE TEST #000037
BEGIN TEST #000040
BEGIN TEST #000041
IGNORE TEST #00042
BEGIN TEST #000043
```

Figure 7-3 EDOBA Test Select Switch Option

7.4 MICROHARDCORE CONTEXT

EDKAA provides a set of program switches that allow the user to modify execution during troubleshooting. These switches provide the following options.

1. Modify the action of the program when a fault is detected
2. Control the volume of output when a fault is detected
3. Select a specific range of tests to be run
4. Control the number of passes made by the diagnostic

Figure 7-4 shows how to use the /MODE:QUIET switch to inhibit all output except ERROR and END OF PASS messages. Figure 7-5 shows how to use a combination of the /MODE:QUIET, /NUMBER:, and /PASS: switches.

```
MH>STARTC/PRINT_MODE:QUIET
PASS COUNTER = 00001, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
MH>STARTC/PASS:3/PRINT_MODE:QUIET
PASS COUNTER = 00001, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
PASS COUNTER = 00002, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
PASS COUNTER = 00003, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
MH>
```

Figure 7-4 MHC Quiet Mode Switch Option

```
MH>STARTC/PASS:2/NUMBER:2 3
PASS COUNTER = 00001, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
PASS COUNTER = 00002, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
MH>
MH>STARTC/PRINT_MODE:VERBOSE/PASS:3/NUMBER 2 5
 CLK - (C-2) L0217 (CLK) SDB shift chain LOAD function [OK]
 CLK - (C-3) L0217 (CLK) Vis Mux Selects [OK]
 CLK - (C-4) L0217 (CLK) Vis data and LD_FUNC_REG_B [OK]
 CLK - (C-5) L0217 (CLK) Load Frequency Reg [OK]
PASS COUNTER = 00001, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
 CLK - (C-2) L0217 (CLK) SDB shift chain LOAD function [OK]
 CLK - (C-3) L0217 (CLK) Vis Mux Selects [OK]
 CLK - (C-4) L0217 (CLK) Vis data and LD_FUNC_REG_B [OK]
 CLK - (C-5) L0217 (CLK) Load Frequency Reg [OK]
PASS COUNTER = 00002, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
 CLK - (C-2) L0217 (CLK) SDB shift chain LOAD function [OK]
 CLK - (C-3) L0217 (CLK) Vis Mux Selects [OK]
 CLK - (C-4) L0217 (CLK) Vis data and LD_FUNC_REG_B [OK]
 CLK - (C-5) L0217 (CLK) Load Frequency Reg [OK]
PASS COUNTER = 00003, # OF ERRORS = 00000, TOTAL # OF ERRORS = 00000
MH>
```

Figure 7-5 MHC Test Number Selection Switch Option

7.5 DIAGNOSTIC CONTEXT

This section discusses several ways to use the console command sets to modify operation of the microdiagnostics. There will be occasions when the nature of the fault causes the microdiagnostic to misbehave or fail to adequately identify the source of the problem. What does the user do then? The following paragraphs describe how to use the console command sets' power to retrieve additional troubleshooting information that will aid in fault isolation.

7.5.1 Defining and Adding Trace Items

Before discussing how to actually single step a failing microtest, the user needs to learn how to define additional SDB registers and how to add these registers and other SDB signals to the trace list used to retrieve data when the user microsteps the CPU.

The following examples show how to use the TRACE commands. Figure 7-6 illustrates how to use the SHOW DEFINE command to determine how an existing register is defined. Note that the display provides the following information.

1. A one-liner that gives the register ID (8035), the name (PSL), and the number of bits

2. One or more lines of visibility ID numbers (E117, C113,...., xxxx,...9167) where the E117 is the MSB and the E9167 is the LSB

> NOTE
> The "xxxx" signifies an unused bit position.

```
DC>>SHOW DEFINE
DC>>SHOW DEFINE PSL
    8035  PSL  21.
    E117 C113 E112 E152 A144 A162 XXXX XXXX D125 D127 D126 D131
    D130 XXXX XXXX C109 XXXX 9171 9172 9168 9167
DC>>
```

Figure 7-6 SHOW DEFINE Command Display

To use the visibility ID numbers in other commands, the characters "V$" must precede them (i.e., V$E117 for the MSB). When we want to define a register, the reverse procedure is used (remove the "V$" and use only the number).

Figure 7-7 shows another example (using the EXAM/SDB command) to examine a signal using the visibility ID. Note how the "V$" was added, as previously explained. Note, also, how the user must stop the CPU clock before examining the SDB. If the user forgets, the console software will display a polite reminder.

```
DC>>SHOW NAME V$E117
    1CA1  V$E117 EBE PSL CM TO CSB H
DC>>
```

Figure 7-7 SHOW NAME Command Display

Figure 7-8 shows the REPORT command's typical response which displays the items in the currently defined trace list. The example happens to show the default list, but that can be amended, as the next paragraph will show.

```
DC>>REPORT
Registers:
      OPMCF_:000000000001      OPCODE:000000000011      OPBUS_:0000FFFFFF82
      EVABUS:000000000000      EDPPE_:00008000002F      EBFLSH:00000000001C
    * WBUS__:0000C10258B1      REGBUS:000000000000      PAMM__:000000000005
      PAMD__:000000841000      MEMREQ:00000010B100      ARBUS_:000000000000
      ARADR_:00000147E200      ABUS__:000F00000000      STALL_:000000FF0000
      NATRAM:000000000592      MDBUSM:00000621F000      MDBUSI:00000621F000
      IOPSEL:000000000000      INCR__:000000000010      IBUF__:00000000F000
      IBDBUF:0000016EFF06      IVABUS:000000000003      DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H
DC>>
```

Figure 7-8 REPORT Command Display

Figure 7-9 shows how to add additional registers to the trace list with the TRACE ADD command and how to use the REPORT command to verify that it took place. By the way, there are many more predefined registers in the console software (PSL IUPC, EUPC, etc.) which can be added when needed. For a list of these predefined registers, simply use the SHOW REGISTER command. Appendix E contains a complete description of all the predefined SDB registers.

```
DC>>TRACE ADD PSL IUPC EUPC
DC>>REPORT
Registers:
      OPMCF_:000000000001        OPCODE:000000000011        OPBUS_:0000FFFFFF82
      EVABUS:000000000000        EDPPE_:000080000020        EBFLSH:00000000001C
    * WBUS__:000060125A23        REGBUS:000000000000        PAMM__:000000000005
      PAMD__:000000841000        MEMREQ:00000010B100        ARBUS_:000000000000
      ARADR_:00000147E200        ABUS__:000F00000000        STALL_:000000FF0000
      NATRAM:000000000592        MDBUSM:00000621F000        MDBUSI:00000621F000
      IOPSEL:000000000000        INCR__:000000000010        IBUF__:00000000F000
      IBDBUF:0000016EFF06        IVABUS:000000000003        DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF        PSL___:000000080024        IUPC__:00000000002D
    * EUPC__:000000000432
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H
DC>>
```

Figure 7-9 TRACE ADD Command Display

Figure 7-10 shows the quickest way to restore the trace list to its default values using the TRACE RESTORE command. Note how the REPORT command shows how PSL, EUPC, and IUPC are taken out.

```
DC>>TRACE RESTORE
DC>>REPORT
        No previous trace data available
Registers:
    * OPMCF_:000000000001    * OPCODE:000000000011    * OPBUS_:0000FFFFFF82
      EVABUS:000000000000    * EDPPE_:000080000020    * EBFLSH:00000000001C
    * WBUS__:0000915099A3      REGBUS:000000000000    * PAMM__:000000000005
    * PAMD__:000000841000    * MEMREQ:00000010B100      ARBUS_:000000000000
    * ARADR_:00000147E200    * ABUS__:000F00000000    * STALL_:000000FF0000
    * NATRAM:000000000592    * MDBUSM:00000621F000    * MDBUSI:00000621F000
      IOPSEL:000000000000    * INCR__:000000000010    * IBUF__:00000000F000
    * IBDBUF:0000016EFF06    * IVABUS:000000000003    * DBUS__:00007FFFD797
    * FABUS_:0000FFFFFFFF
Hi sigs:
    * V$C216 -MCC MD RESP A H          * V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H
DC>>
```

Figure 7-10 TRACE RESTORE Command Display

Now, suppose the user needs to define a specific set of SDB signals to watch while tracing a failing microdiagnostic. How is it done? First, the user jots down the names of the signals to be retrieved, probably from the logic drawings. The next step could be to use the EXAMINE/SDB command with signal name argument, which will display the state of the signal along with its visibility ID. Another alternative could be to use the files on the RL02. Figure 7-11 shows the procedure. The SHOW file command is first used to get the names of the signal files in CONFIG.DAT. It's used again to display the signal file(s) for the module(s) that contain the signals to be retrieved. The next example will use the circled signals shown in Figure 7-12 as an example for defining registers and adding them to the trace list.

```
DC>>SHOW CONFIG.DAT/ASCII
! SDB Cad information files for M5 machines.
! Version: 003.000
! Released: December 21, 1985
CLKC03.CDF      !CLK C5
CSAB02.CDF      !CSA B2
CSBB02.CDF      !CSB B2
EBCC05.CDF      !EBC C5
EBDD02.CDF      !EBD D2
EBEB02.CDF      !EBE B2
EDPC02.CDF      !EDP C2
FBAB01.CDF      !FBA E7
FBMC01.CDF      !FBM C5
IBDF05.CDF      !IBD F5
ICAH02.CDF      !ICA H4
ICBF01.CDF      !ICB F5
IDPF02.CDF      !IDP F4
MAPD02.CDF      !MAP D2
MCCK01.CDF      !MCC K1
MCDD04.CDF      !MCD D4
MTMB01.CDF      !MTM B1
!VBAA01.CDF     !VBA A1 (SBI VISIBILITY MODULE #1)
!VBBA01.CDF     !VBB A1 (SBI VISIBILITY MODULE #2)
DC>>
```

Figure 7-11 Displaying the CONFIG.COM File

```
DC>>SHOW MAPD02.CDF/ASCII
;CHASER Version 1(31)-1, 21 January 1984. Sources in LSCAD:<SDB>
/CADIF-VERSION/ 3(5)
/SUDS-SDB/ 1(2)
$SUDS-CHANNEL-TO-SIGNAL
13000    V$B197  ABUS DATA ADDRS 26 H
13001    V$B198  ABUS DATA ADDRS 25 H
13002    V$B199  ABUS DATA ADDRS 27 H
13003    V$B200  ABUS DATA ADDRS 23 H
13004    V$B201  EDP EVA A25 H
13005    V$B202  IVA BUS 29 H
13006    V$B203  ABUS DATA ADDRS 30 H
13007    V$B204  ABUS DATA ADDRS 21 H
13010    V$B233  -MCCB M ABUS LAT LD H
13011    V$B151  -MCC6 PHYS REF H
13012    V$B227  -MCC7 PA LAT LD H
13013    V$B142  -MCC7 WRITE LRU H
13014    V$B156  -MCC7 WRITE CACHE EN A H
13015    V$B154  MCCB M PA MUX SEL 1 H
13016    V$B155  MCCB M PA MUX SEL 0 H
13017    V$B234  -MCCA RFL LAT LD H
13020    V$B146  EDP EVA A13 H
13021    V$B143  EDP EVA A14 H
13022    V$B144  EDP EVA A11 H
13023    V$B145  EDP EVA A12 H
13024    V$B147  IVA BUS 14 H
13025    V$B162  EDP EVA A00 A H
13026    V$B148  IVA BUS 12 H
13027    V$B149  IVA BUS 13 H
13040    V$B122  EDP EVA A17 H
13041    V$B123  EDP EVA A20 H
13042    V$B124  IVA BUS 17 H
13043    V$B138  ABUS DATA ADDRS 17 H
13044    V$B126  EDP EVA A19 H
13045    V$B136  ABUS DATA ADDRS 16 H
13046    V$B127  ABUS DATA ADDRS 11 H
13047    V$B128  ABUS DATA ADDRS 28 H
13050    V$B100  MCDM ECC CORR ERROR H
13051    V$B232  MCCE U CLR CSH WV H
13052    V$B112  EDP EVA A30 H
13053    V$B223  IVA BUS 30 H
13054    V$B105  MCC5 VAL 2 REQ H
13055    V$B104  MCC5 VAL 1 REQ H
13056    V$B107  MCC5 VAL 3 REQ H
13057    V$B106  MCC5 VAL 0 REQ H
13060    V$B160  IVA BUS 31 H
13061    V$B226  MCCM HLD ERR ADR REG H
13062    V$B109  MCC6 VIRT REF H
13063    V$B222  EDP EVA A31 H
13064    V$B229  -MCC6 EBOX 1ST CYC H
13065    V$B230  MCC5 CP CYC H
13066    V$B157  MCCB M PA MUX SEL 2 H
13067    V$B231  MAPC PA 04 BUF C H
13100    V$B239  MCCA M ABUS DR EN H
13101    V$B240  -MAPN LD CLK3 T1C B H
13102    V$B110  -MAPN CLK6 PHASE T1D H
13103    V$B111  MCCJ CYC TYP MAP 1 H
13104    V$B102  MCCJ CYC TYP MAP 0 H
13105    V$B101  MCCJ CYC TYP MAP 2 H
13106    V$B224  MCC8 HLD START LAT H
13107    V$B113  MAPN LD CLK3 T1A C H
^C
?DCN-I-CCABRT, ^C abort
DC>>
```

Figure 7-12 Displaying an SDB Signal Name File

Figure 7-13 shows how to define a new register called "PAMUX" using the TRACE DEFINE command and how to verify it with the SHOW DEFINE command.

```
DC>>TRACE DEFINE PAMUX
    Enter V$ terms separated with <sp> or <,>
    B157 B154 B155

    B9A8  PAMUX  3.
DC>>SHOW DEFINE PAMUX
    B9A8  PAMUX  3.
    B157 B154 B155
DC>>
```

Figure 7-13 Defining a New SDB Register

Figure 7-14 shows the use of REPORT to verify that PAMUX was automatically added to the current trace list, while Figure 7-15 shows how to remove PAMUX when done. Remember, though, it's still defined and can be added back later.

```
DC>>REPORT
Registers:
      OPMCF_:000000000001      OPCODE:000000000011      OPBUS_:0000FFFFFF82
      EVABUS:000000000000    * EDPPE_:00008000002F      EBFLSH:00000000001C
    * WBUS__:00008218E082      REGBUS:000000000000      PAMM__:000000000005
      PAMD__:000000841000      MEMREQ:00000010B100      ARBUS_:000000000000
      ARADR_:00000147E200      ABUS__:000F00000000      STALL_:000000FF0000
      NATRAM:000000000592      MDBUSM:00000621F000      MDBUSI:00000621F000
      IOPSEL:000000000000      INCR__:000000000010      IBUF__:00000000F000
      IBDBUF:0000016EFF06      IVABUS:000000000003      DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF      PAMUX_:000000000004
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H
DC>>
```

Figure 7-14 REPORT Display to Show Defined Register

```
DC>>TRACE REMOVE PAMUX
DC>>REPORT
Registers:
      OPMCF_:000000000001      OPCODE:000000000011      OPBUS_:0000FFFFFF82
      EVABUS:000000000000      EDPPE_:00008000002F      EBFLSH:00000000001C
    * WBUS__:00009D619966      REGBUS:000000000000      PAMM__:000000000005
      PAMD__:000000841000      MEMREQ:00000010B100      ARBUS_:000000000000
      ARADR_:00000147E200      ABUS__:000F00000000      STALL_:000000FF0000
      NATRAM:000000000592      MDBUSM:00000621F000      MDBUSI:00000621F000
      IOPSEL:000000000000      INCR__:000000000010      IBUF__:00000000F000
      IBDBUF:0000016EFF06      IVABUS:000000000003      DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H
DC>>
```

Figure 7-15 TRACE REMOVE Command Display

The following examples show how to add SDB signals to the trace list. Figure 7-16 shows adding PAMUX back. Figure 7-17 shows how to add the SDB signal known by the SDB visibility ID as V$B227 (-MCC7 PA LAT LD H). How is this done? Look back at Figure 7-12. Note how the REPORT command in Figure 7-17 shows that both PAMUX and -MCC7 PA LAT LD H have been added.

```
DC>>TRACE ADD PAMUX
DC>>REPORT
Registers:
      OPMCF_:000000000001      OPCODE:000000000011    * OPBUS_:0000FFFFFF82
      EVABUS:000000000000      EDPPE_:000080000020      EBFLSH:00000000001C
    * WBUS__:00000C6123D4      REGBUS:000000000000      PAMM__:000000000005
      PAMD__:000000841000      MEMREQ:00000010B100      ARBUS_:000000000000
      ARADR_:00000147E200      ABUS__:000F00000000      STALL_:000000FF0000
      NATRAM:000000000592      MDBUSM:00000621F000      MDBUSI:00000621F000
      IOPSEL:000000000000      INCR__:000000000010      IBUF__:00000000F000
      IBDBUF:0000016EFF06      IVABUS:000000000003      DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF      PAMUX_:000000000004
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H
DC>>
```

Figure 7-16 Adding a Defined SDB Register

```
DC>>TRACE ADD V$B227
DC>>REPORT
Registers:
      OPMCF_:000000000001      OPCODE:000000000011      OPBUS_:0000FFFFFF82
      EVABUS:000000000000      EDPPE_:000080000020      EBFLSH:00000000001C
    * WBUS__:000043849E3A      REGBUS:000000000000      PAMM__:000000000005
      PAMD__:000000841000      MEMREQ:00000010B100      ARBUS_:000000000000
      ARADR_:00000147E200      ABUS__:000F00000000      STALL_:000000FF0000
      NATRAM:000000000592      MDBUSM:00000621F000      MDBUSI:00000621F000
      IOPSEL:000000000000      INCR__:000000000010      IBUF__:00000000F000
      IBDBUF:0000016EFF06      IVABUS:000000000003      DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF      PAMUX_:000000000004
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H             V$B227 -MCC7 PA LAT LD H
DC>>
```

Figure 7-17 Adding an SDB Signal Name

7.5.2 Microstepping a Microdiagnostic

Microdiagnostics have the general structure shown in Figure 7-18. They consist of a set of microtests that run out of the EBox control store. Each test consists of a set of microinstructions in which each microinstruction is executed in four time states. The console software allows the user to control the sequence at each level with commands that do the following.

1. Single step a diagnostic one test at a time
2. Single microstep a test one microinstruction at a time
3. Single clock a microinstruction one time state at a time

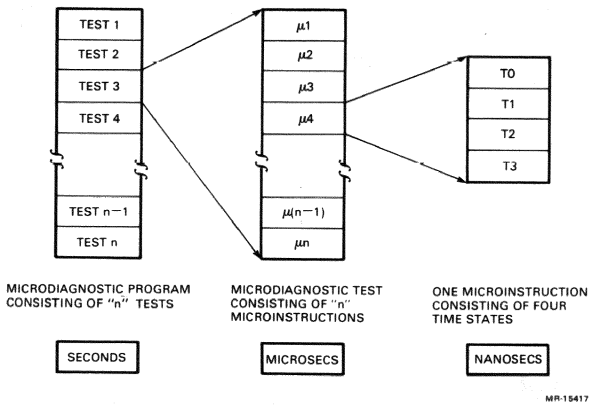


Figure 7-18 Microdiagnostic Program Structure

The first step in microstepping a failing microdiagnostic test to trace the operation of each microinstruction is to stop the CPU at the beginning of the microtest. This section explains the procedure, using the FBox microdiagnostic EDK1A (test 1) as an example.

To stop on an EBox micromark, type the following three commands.

1. Use SET SOMM ON to enable stopping.
2. Use DEP/MARK/ECS adr ON to deposit the mark bit in the EBox control store.
3. Use START to start the microdiagnostic.

Figure 7-19 shows how to use @EDK1A to load the microdiagnostic and SHOW FLAGS to verify that SOMM is ON. It also shows how to deposit the mark bit in location 801 to stop at the start of test 1. Location 801 in the EBox control store contains a GO TO microinstruction that transfers control to the start of test 1. Refer to Appendix C for a description of the microdiagnostic dispatch table. Location 8nn contains the dispatch address for test nn. After setting the breakpoint, the START command gets things rolling until the micromark is sensed and the CPU stops, as shown at the end of Figure 7-19. Note that if the SHOW FLAGS command indicated that the SOMM flag was OFF, the user needs to use the SET SOMM ON command to set the flag.

```
DC>>@EDK1A
DC>>SHOW FLAGS
        ABOrt    on
        ABUs     off
        Base     off
        COld     off
        Exti     on
        Fbox     off
        Iosafe   off
        Local    off
        Memena   on
        Quiet    on
        SOmm     off
        SNap     on
        STxalt   off
        Warm     off
DC>>STOP CPU
DC>>DEP/MARK/ECS 801 ON
DC>>START CPU
DC>>SET SOMM ON
DC>>SHOW FLAGS
        ABOrt    on
        ABUs     off
        Base     off
        COld     off
        Exti     on
        Fbox     off
        Iosafe   off
        Local    off
        Memena   on
        Quiet    on
        SOmm     on      Ecs
        SNap     on
        STxalt   off
        Warm     off
DC>>START
?DCP-E-BADMWD, stop on umark bit
DC>>
```

Figure 7-19 Setting a Micromark Breakpoint

NOTE

At full clock speed, the microsequencer stops two microinstructions beyond the mark. If it's necessary to stop on the spot of the mark, use the SET CLOCK ONE-FIFTH command to slow things down. This is a design limitation of the machine hardware.

Figure 7-20 shows how to use the MIC command and space-bar mode to step the microdiagnostics through two microsteps. Note how the micromark break stopped the clock with the EBox micro-PC at location 081B, not 0801. After executing one microinstruction, DCP enters space-bar mode with the DC>> prompt at the end of the line. Successive microinstructions may be executed from here by pressing the space bar. To exit space-bar mode, simply press **<RETURN>** to get to the normal DC>> prompt. Note that the only information displayed by the MIC command is the micro-PCs in all the boxes.

```
DC>>MIC
MBOX=0018 EBOX=081B FBOXA=005F FBOXM=001D IBOX=002D DC>>
MBOX=0018 EBOX=081C FBOXA=005F FBOXM=001D IBOX=002D DC>>
DC>>
```

Figure 7-20 MIC Command Display

Figures 7-21 and 7-22 illustrate use of the TMIC command and space-bar mode to microstep the program and display a trace list at the end of each step. Like the MIC command, simply press the space bar to proceed to the next microinstruction. TMIC is essentially the same as a MIC command followed by a REPORT command. It provides much more information than the current micro-PCs, including most of the key SDB registers and three SDB signals. Note that the "*" in the display indicates either a change in signal state or the value of an SDB register.

```
DC>>TMIC
        Tracing from count: 0 (0.)
         Micro-cycle count: 1 (1.)  Phase: T3
Micro PC's:
            E:081E       M:0018       I:002D       FA:005F       FM:001D
Registers:
      OPMCF_:000000000001       OPCODE:000000000011       OPBUS_:0000FFFFFF82
      EVABUS:000000000000       EDPPE_:00018000002F       EBFLSH:00000000001C
      WBUS__:000000000000       REGBUS:000000000000       PAMM__:000000000005
      PAMD__:000000841000       MEMREQ:00000010B100       ARBUS_:000000000000
      ARADR_:00000147E200       ABUS__:000F00000000       STALL_:000000FF0000
      NATRAM:000000000592       MDBUSM:00000621F000       MDBUST:00000621F000
      IOPSEL:000000000000       INCR__:000000000010       IBUF__:00000000F000
      IBDBUF:0000016EFF06       IVABUS:000000000003       DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF       PAMUX_:000000000004
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H             V$B227 -MCC7 PA LAT LD H
SBSMDC>>
        Tracing from count: 1 (1.)
         Micro-cycle count: 2 (2.)  Phase: T3
Micro PC's:
            E:083A       M:0018       I:002D       FA:005F       FM:001D
Registers:
      OPMCF_:000000000001       OPCODE:000000000011       OPBUS_:0000FFFFFF82
      EVABUS:000000000000       EDPPE_:00018000002F       EBFLSH:00000000001C
      WBUS__:000000000000       REGBUS:000000000000       PAMM__:000000000005
      PAMD__:000000841000       MEMREQ:00000010B100       ARBUS_:000000000000
      ARADR_:00000147E200       ABUS__:000F00000000       STALL_:000000FF0000
      NATRAM:000000000592       MDBUSM:00000621F000       MDBUST:00000621F000
      IOPSEL:000000000000       INCR__:000000000010       IBUF__:00000000F000
      IBDBUF:0000016EFF06       IVABUS:000000000003       DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF       PAMUX_:000000000004
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H             V$B227 -MCC7 PA LAT LD H
SBSMDC>>
```

Figure 7-21 TMIC Command -- Display 1

```
        Tracing from count: 2 (2.)
         Micro-cycle count: 3 (3.)  Phase: T3
Micro PC's:
          E:083B       M:0018       I:002D       FA:005F       FM:001D
Registers:
      OPMCF_:000000000001       OPCODE:000000000011       OPBUS_:0000FFFFFF82
      EVABUS:000000000000       EDPPE_:00018000002F       EBFLSH:00000000001C
      WBUS__:000000000000       REGBUS:000000000000       PAMM__:000000000005
      PAMD__:000000841000       MEMREQ:00000010B100       ARBUS_:000000000000
      ARADR_:00000147E200       ABUS__:000F00000000       STALL_:000000FF0000
      NATRAM:000000000592       MDBUSM:00000621F000       MDBUST:00000621F000
      IOPSEL:000000000000       INCR__:000000000010       IBUF__:00000000F000
      IBDBUF:0000016EFF06       IVABUS:000000000003       DBUS__:00007FFFD797
      FABUS_:0000FFFFFFFF       PAMUX_:000000000004
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H             V$B227 -MCC7 PA LAT LD H
SBSMDC>>
DC>>LUPC/ECS 0
DC>>START CPU
DC>>SET SOMM OFF
DC>>START
^T
Running diagnostic - EDK1A
Test # = 7 & alive byte = 27
End of pass - EDK1A - start test #1 - end test #17
DC>>
```

Figure 7-22 TMIC Command -- Display 2

Figure 7-23 summarizes the entire operation from the time the user types DIAG to enter diagnostic context, until using TMIC with the first microinstruction after the stop on micromark. Let's assume that the next thing the user wants to do is to single clock time states for the microinstruction in location 081C, which is the next microinstruction. How is this done? Look at the E:081C in the trace display in Figure 7-23.

Finally, Figures 7-24 through 7-27 show how to use TSTATE in space-bar mode to trace the next four time states. Figure 7-28 shows how to use MIC to display four more microinstructions and exit space-bar mode. Figure 7-28 also shows the procedure for restarting DSM and the CPU clock after a microstepping sequence. It then shows how to turn off the SOMM feature.

It is important to remember the following sequence of commands if the user wishes to exit space-bar mode and continue a microdiagnostic. Jot them down.

1. LUPC/ECS 0 -- restarts DSM
2. START CPU -- turns on the CPU CLOCK
3. SET SOMM OFF -- disables Stop On Micromark (SOMM)
4. START -- restarts the diagnostic

```
MH>DIAG
                CONFIDENTIAL DIAGNOSTIC SOFTWARE
                        PROPERTY OF
                DIGITAL EQUIPMENT CORPORATION
   Use Authorized Only Pursuant to a Valid Right-to-Use License
Initializing DC
DC>>@EDK1A
DC>>SET SOMM ON
DC>>STOP CPU
DC>>DEP/MARK/ECS 801
?DCN-W-PARSER, invalid command
DC>>DEP/MARK/ECS 801 ON
DC>>START CPU
DC>>START
?DCP-E-BADMWD, stop on umark bit
DC>>MIC
MBOX=0018 EBOX=081B FBOXA=005F FBOXM=001D IBOX=002D DC>>
DC>>TMIC
        Tracing from count: 0 (0.)
         Micro-cycle count: 1 (1.)  Phase: T3
Micro PC's:
            E:081C      M:0018      I:002D      FA:005F      FM:001D
Registers:
      OPMCF_:000000000001      OPCODE:000000000011      OPBUS_:0000FFFFFF82
      EVABUS:000000000000      EDPPE_:00018000002F      EBFLSH:00000000001C
      WBUS__:000000000000      REGBUS:000000000000      PAMM__:000000000005
      PAMD__:000000841000      MEMREQ:00000010B100      ARBUS_:000000000000
      ARADR_:00000147E200      ABUS__:000F00000000      STALL_:000000FF0000
      NATRAM:000000000592      MDBUSM:00000621F000      MDBUSI:00000621F000
      IOPSEL:000000000000      INCR__:000000000010      IBUF__:00000000F000
      IBDBUF:0000016EFF06      IVABUS:000000000003      DBUS__:00007FFFD797
    * FABUS_:0000FFFFF7FF      PAMUX_:000000000004
Hi sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H             V$B227 -MCC7 PA LAT LD H
SBSMDC>>
DC>>LUPC/ECS 0
DC>>START CPU
DC>>SET SOMM OFF
DC>>START
End of pass - EDK1A - start test #1 - end test #17
DC>>
```

Figure 7-23 Microstepping Summary

One final note on MIC and TMIC. Both of these commands accept a numeric argument that permits executing "n" microinstructions or time states before stopping. This is useful for skipping over several microinstructions to get to the point of interest. For example, the command MIC 10 would execute 10 microinstructions before stopping.

7.6 MACRO CONTEXT

Error reports from macrodiagnostics may not provide enough data to isolate the fault. The user may need to modify the program and use the failing test, along with the descriptions in the listing, to perform further analysis. The procedure is generally as follows (refer to Figure 7-24).

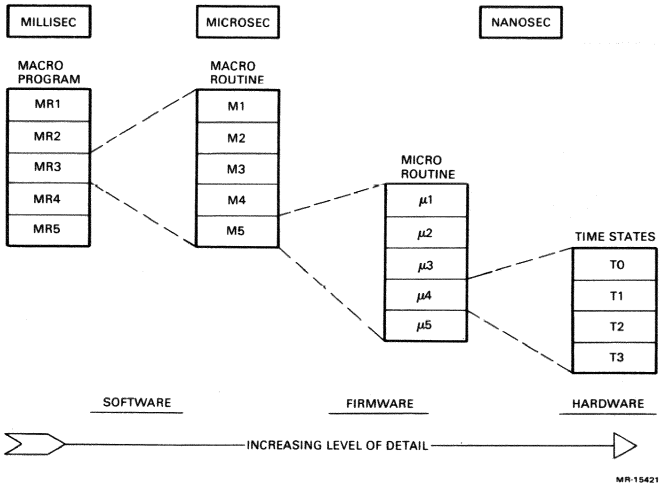


Figure 7-24 Macrodiagnostic Program Structure

1. Use the macrodiagnostic to identify the failing sequence of macroinstructions (MR3).

2. Single step each macroinstruction in the loop to verify that the proper sequence of PCs is being generated.

3. Check the results of each instruction, one at a time, to find the failing instruction (M4).

4. Single step the microsequencers to verify that the proper sequence of microinstructions is being generated.

5. Check the data within the VAX CPU after each microinstruction to determine which one is failing (u4).

6. Perhaps, step the VAX CPU, one time state at a time, to determine the actual point of failure (T0--T3).

7. Retrieve and analyze SDB signal information to isolate the fault to a module or component.

The next three sections will describe this procedure using a simple macro routine deposited into VAX internal memory.

7.6.1 Single Stepping Macroinstructions

Figure 7-25 shows a simple routine that will be used to illustrate the process. It contains CLRL, MOVL, INCL, ADDL, and JMP commands that operate as shown below.

1. Start at location 1000
2. Clear R3
3. Move a longword (0000FFFF) from location 2000 into R2
4. Increment the contents of R3
5. Add the longword in R2 (0000FFFF) to the longword in R2 (00000001) and write the result (00010000) back into R3
6. Jump back to location 1000 and repeat the sequence

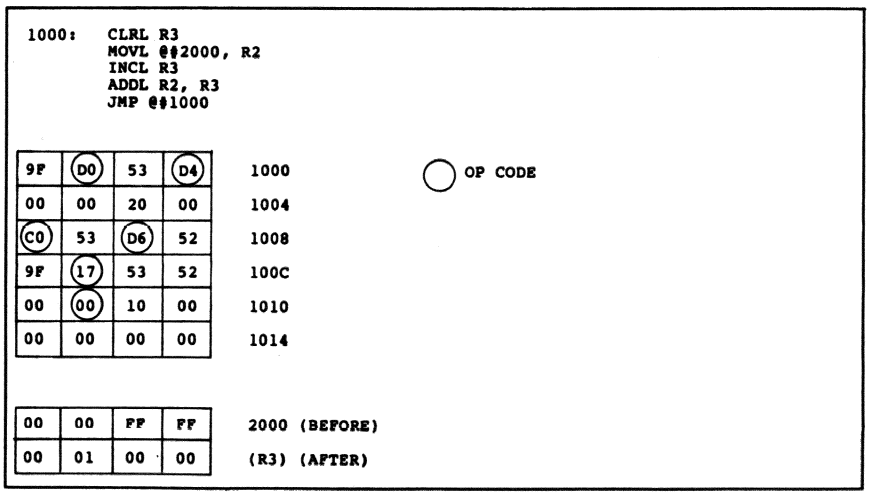


Figure 7-25 Macro Test Routine Overview

Figure 7-26 shows how to deposit the information into memory starting at location 1000, and includes how to deposit the FFFF into location 2000. Figure 7-27 shows how to use the EXAMINE command with the /NEXT switch to verify that it was deposited correctly. Figure 7-28 shows how to to single instruct the routine using the NEXT (N) command in space-bar mode. Note how the PCs match the "circled" addresses in Figure 7-25.

Figure 7-29 shows how to enable cache, start the routine looping, and halt with <CTRL/P> followed by a HALT command. After the machine halts, the instructions and data should be in cache for the following procedures so the routine won't have to go to the array bus for refills. Finally, Figure 7-30 shows how to single instruct using the NEXT command and check the results of each instruction using the EXAMINE command.

```
>>>D 1000 9FD053D4
>>>D + 2000
>>>D + C053D652
>>>D + 9F175352
>>>D + 1000
>>>D + 0
>>>
>>>D 2000 FFFF
>>>
```

Figure 7-26 Depositing a Macro Test Routine

```
>>>E/N:5 1000
        P 00001000 9FD053D4
        P 00001004 00002000
        P 00001008 C053D652
        P 0000100C 9F175352
        P 00001010 00001000
        P 00001014 00000000
>>>E 2000
        P 00002000 0000FFFF
>>>
```

Figure 7-27 Examining a Macro Test Routine

```
>>>DEP PC 1000
>>>N
        PC 00001002>>>
        PC 00001009>>>
        PC 0000100B>>>
        PC 0000100E>>>
        PC 00001000>>>
        PC 00001002>>>
        PC 00001009>>>
        PC 0000100B>>>
        PC 0000100E>>>
        PC 00001000>>>
>>>
```

Figure 7-28 Single Instructing a Macro Test Routine

```
>>>E CSWP
        I 42 00000000
>>>D CSWP B
>>>E CSWP
        I 42 00000003
>>>START 1000
?MCP-I-CPSRUN, CPU is still running
>>>HALT
        CPU stopped, INVOKED BY CONSOLE (CSM code 11)
        PC 00001000
>>>
```

Figure 7-29 Enabling Cache and Starting the Routine

```
>>>DEP PC 1000
>>>N
        PC 00001002>>>
>>>E R3
        G 03 00000000
>>>N
        PC 00001009>>>
>>>E R2
        G 02 0000FFFF
>>>N
        PC 0000100B>>>
>>>E R3
        G 03 00000001
>>>N
        PC 0000100E>>>
>>>E R3
        G 03 00010000
>>>N
        PC 00001000>>>
>>>
```

Figure 7-30 Single Instructing and Checking Results

7.6.2 Single Clocking Microinstructions

After identifying the failing macroinstruction, the next step is to microstep that instruction to determine the failing microinstruction. Refer to Figure 7-31 for the following discussion. The START/STEP 1000 command prepares the routine for microstepping. The DEBUG command enables the HEX command set and the MIC command begins the microstepping in space-bar mode. Each time the user presses the space bar, the EBox executes a single microinstruction and displays the five "box" micro-PCs (MBOX, EBOX, FBOXA, FBOXM, and IBOX). Obviously, the user will need the microcode listings to verify the correct sequence before determining any sequencing faults. That's beyond the scope of this manual. Note the EBox micro-PC sequence in Figure 7-31. Notice that the sequence begins to repeat after EBox=0200. That's the start of a second pass through the loop. The first nine microinstructions are executing the CSM overlay for the START command and do not repeat again. At the end, press <RETURN> and exit space-bar mode to execute additional commands.

It is important to remember that when exiting space-bar mode, the EBox micro-PC is left so that it has lost communication with CSM and the clock is OFF. To restart the clock and CSM, the user must execute an UNHANG command. Depending upon the state of the machine, this might not always work. It may be necessary to execute an INIT/CPU command to reload and restart CSM. If this fails, type MACRO to reinitialize macro context.

7.6.3 Microinstruction Tracing

After the previous procedure, the user may or may not have identified one or more failing microinstructions. In any case, more data may be needed to isolate the fault. That's the purpose of the trace. Figure 7-31 shows the use of START/STEP and MIC to get the EBox microsequencer to where tracing is to begin (EBOX=1B00 [AFORK]). At this time, use TMIC in space-bar mode to begin the trace. Figures 7-32 through 7-36 show the trace display at the end of each microstep. In each of these figures, note the following important information and relate it to the actual macroroutine in Figure 7-25.

1. IVABUS -- the virtual address sent to the MBox
2. MDBUSI,M -- the cache data coming back
3. IBUF -- the contents of the instruction buffer
4. OPCODE -- the current OP code being executed

Do things look correct? They must be because the routine is known to be working. After analyzing all the data in the trace at each microstep, the user should be able to identify the point of failure and how it is failing. If more information is needed, use the STATESTEP command to step the failing microinstruction as described in the next section.

```
>>>START/STEP 1000
>>>DEBUG
>>>>MIC
MBOX=0018 EBOX=1081 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1082 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1083 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1085 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1086 FBOXA=0004 FBOXM=0006 IBOX=000A >>>>
MBOX=0018 EBOX=1088 FBOXA=0007 FBOXM=0007 IBOX=002A >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00CC >>>>
```

Figure 7-31 TMIC Setup Display

```
>>>>TMIC
        Tracing from count: 0 (0.)
         Micro-cycle count: 1 (1.)  Phase: T3
Micro PC's:
            E:1B00      M:0019      I:00CA      FA:0004      FM:0006
Registers:
      OPMCF_:000000000001     * OPCODE:0000000000D4     * OPBUS_:000000001004
      EVABUS:000000000000     * EDPPE_:000B8020012C       EBFLSH:00000000001C
    * WBUS__:000000002000       REGBUS:000000000000     * PAMM__:000000000000
    * PAMD__:000000941040     * MEMREQ:00000090C100       ARBUS_:000000000000
    * ARADR_:000000001000       ABUS__:000F00000000     * STALL_:000018FF0020
    * NATRAM:000000043826     * MDBUSM:000000002000     * MDBUSI:000000002000
      IOPSEL:000000000000     * INCR__:000000000026     * IBUF__:000000009FD0
    * IBDBUF:022509EA33C5     * IVABUS:000000001008     * DBUS__:0000AC2B602F
    * FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
    * V$B227 -MCC7 PA LAT LD H
Lo sigs:
    * V$C216 -MCC MD RESP A H           * V$6208 ICA LID ISTALL H
      V$C169 MCC DEST CODE 0 H
SBSM>>>>
```

Figure 7-32 TMIC Display -- Step 1

```
        Tracing from count: 1 (1.)
         Micro-cycle count: 2 (2.)  Phase: T3
Micro PC's:
            E:1B00      M:0019      I:00D6      FA:0004      FM:0006
Registers:
      OPMCF_:000000000001       OPCODE:0000000000D4       OPBUS_:000000001004
      EVABUS:000000000000     * EDPPE_:000B0000012D       EBFLSH:00000000001C
    * WBUS__:000000010000       REGBUS:000000000000       PAMM__:000000000000
    * PAMD__:000000801000     * MEMREQ:000000100100       ARBUS_:000000000000
    * ARADR_:000000001008     * ABUS__:000100000000     * STALL_:00000A020000
    * NATRAM:000000044901     * MDBUSM:0000C053D652     * MDBUSI:0000C053D652
      IOPSEL:000000000000     * INCR__:000000000075     * IBUF__:0000000052D0
    * IBDBUF:02100D5EFF06     * IVABUS:00000000100C     * DBUS__:0000FFFFDFFF
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
      V$B227 -MCC7 PA LAT LD H
Lo sigs:
      V$C216 -MCC MD RESP A H             V$6208 ICA LID ISTALL H
      V$C169 MCC DEST CODE 0 H
SBSM>>>>
```

Figure 7-33 TMIC Display -- Step 2

```
        Tracing from count: 2 (2.)
         Micro-cycle count: 3 (3.)  Phase: T3
Micro PC's:
            E:1BC5       M:0018       I:00CA       FA:0004       FM:0006
Registers:
    * OPMCF_:00000000000A     * OPCODE:0000000000D0     * OPBUS_:000000010000
      EVABUS:000000000000     * EDPPE_:000B0040012A       EBFLSH:00000000001C
    * WBUS__:000000002000       REGBUS:000000000000       PAMM__:000000000000
    * PAMD__:00000090A080     * MEMREQ:000001106A00       ARBUS_:000000000000
      ARADR_:000000001008       ABUS__:000100000000     * STALL_:0000080200C0
    * NATRAM:000000046981     * MDBUSM:000000000000     * MDBUSI:000000000000
      IOPSEL:000000000000     * INCR__:000000000023     * IBUF__:0000000053D6
    * IBDBUF:022508E922C1     * IVABUS:000000002000     * DBUS__:0000AD3FAC29
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
    * V$C216 -MCC MD RESP A H
Lo sigs:
      V$6208 ICA LID ISTALL H              V$C169 MCC DEST CODE 0 H
    * V$B227 -MCC7 PA LAT LD H
SBSM>>>>
```

Figure 7-34 TMIC Display -- Step 3

```
        Tracing from count: 3 (3.)
         Micro-cycle count: 4 (4.)  Phase: T3
Micro PC's:
            E:1B06       M:0019       I:00CA       FA:0004       FM:0006
Registers:
    * OPMCF_:000000000001       OPCODE:0000000000D0     * OPBUS_:00000000FFFF
      EVABUS:000000000000     * EDPPE_:000B0020012D       EBFLSH:00000000001C
    * WBUS__:000000000000       REGBUS:000000000000       PAMM__:000000000000
    * PAMD__:000000901040     * MEMREQ:000000904100       ARBUS_:000000000000
    * ARADR_:000000002000     * ABUS__:000F00000000     * STALL_:00000C020020
    * NATRAM:000000043030     * MDBUSM:00000000FFFF     * MDBUSI:00000000FFFF
    * IOPSEL:000000000030     * INCR__:000000000021     * IBUF__:00000000FFC0
    * IBDBUF:0235095A33C1     * IVABUS:00000000100C     * DBUS__:0000FFFFFF3F
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
    * V$C169 MCC DEST CODE 0 H          * V$B227 -MCC7 PA LAT LD H
Lo sigs:
    * V$C216 -MCC MD RESP A H            V$6208 ICA LID ISTALL H
SBSM>>>>
```

Figure 7-35 TMIC Display -- Step 4

```
        Tracing from count: 4 (4.)
         Micro-cycle count: 5 (5.)  Phase: T3
Micro PC's:
            E:0EC1       M:0019       I:00D0       FA:0004       FM:0006
Registers:
      OPMCF_:000000000001     * OPCODE:0000000000D6       OPBUS_:00000000FFFF
    * EVABUS:000000000FFFF     * EDPPE_:000B00200012F       EBFLSH:00000000001C
    * WBUS__:00000000FFFF       REGBUS:000000000000       PAMM__:000000000000
      PAMD__:000000901040     * MEMREQ:000000905100       ARBUS_:000000000000
    * ARADR_:000000002008     * ABUS__:000100000000     * STALL_:000009020020
      NATRAM:000000043030     * MDBUSM:00009F175352     * MDBUSI:00009F175352
    * IOPSEL:000000000000     * INCR__:000000000075     * IBUF__:0000000052C0
    * IBDBUF:021001EEFF82     * IVABUS:000000001010     * DBUS__:0000FFFFFFFF
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
      V$B227 -MCC7 PA LAT LD H
Lo sigs:
      V$C216 -MCC MD RESP A H              V$6208 ICA LID ISTALL H
    * V$C169 MCC DEST CODE 0 H
SBSM>>>>
```

Figure 7-36 TMIC Display -- Step 5

7.6.4 STATESTEP Trace

The final step in the process is to single step the CPU clock one tick at a time and trace the machine state after each tick (20 ns). To arrive at the display shown in Figures 7-37 through 7-40, execute the following procedure.

1. Use START/STEP to prepare for single stepping.
2. Use MIC to step to the failing microinstruction.
3. Use TSTATE to single step the clock in space-bar mode.

An exercise for the reader will be to use the STATESTEP display in Figures 7-37 through 7-40 to determine the actual microinstruction being displayed.

```
>>>>TSTATE
        Tracing from count: 0 (0.)
          State-step count: 1 (1.)  Phase: T0
Registers:
      OPMCF_:000000000001       OPCODE:0000000000D6       OPBUS_:00000000FFFF
      EVABUS:00000000FFFF     * EDPPE_:000B0020012E       EBFLSH:00000000001C
      WBUS__:00000000FFFF       REGBUS:000000000000       PAMM__:000000000000
      PAMD__:000000901040       MEMREQ:000000905100       ARBUS_:000000000000
    * ARADR_:000000001010       ABUS__:000100000000       STALL_:000009020020
      NATRAM:000000043030       MDBUSM:00009F175352       MDBUST:00009F175352
      IOPSEL:000000000000       INCR__:000000000075       IBUF__:0000000052C0
      IBDBUF:021001EEFF82       IVABUS:000000001010     * DBUS__:0000FFFF0000
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
      V$B227 -MCC7 PA LAT LD H
Lo sigs:
      V$C216 -MCC MD RESP A H             V$6208 ICA LID ISTALL H
      V$C169 MCC DEST CODE 0 H
SBSM>>>>
```

Figure 7-37 STATESTEP Display -- Phase 0

```
        Tracing from count: 1 (1.)
          State-step count: 2 (2.)  Phase: T1
Registers:
      OPMCF_:000000000001       OPCODE:0000000000D6       OPBUS_:00000000FFFF
      EVABUS:00000000FFFF     * EDPPE_:000B8020012E       EBFLSH:00000000001C
      WBUS__:00000000FFFF       REGBUS:000000000000       PAMM__:000000000000
    * PAMD__:000000941040     * MEMREQ:00000090D100       ARBUS_:000000000000
      ARADR_:000000001010       ABUS__:000100000000     * STALL_:000019FF0020
      NATRAM:000000043030       MDBUSM:00009F175352       MDBUST:00009F175352
      IOPSEL:000000000000     * INCR__:000000000035       IBUF__:0000000052C0
    * IBDBUF:021001DEFF82       IVABUS:000000001010       DBUS__:0000FFFF0000
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
    * V$6208 ICA LID ISTALL H             V$B227 -MCC7 PA LAT LD H
Lo sigs:
      V$C216 -MCC MD RESP A H             V$C169 MCC DEST CODE 0 H
SBSM>>>>
```

Figure 7-38 STATESTEP Display -- Phase 1

```
        Tracing from count: 2 (2.)
          State-step count: 3 (3.)  Phase: T2
Registers:
    * OPMCF_:00000000000A       OPCODE:000000000D6      OPBUS_:00000000FFFF
      EVABUS:00000000FFFF       EDPPE_:000B80200012E    EBFLSH:00000000001C
    * WBUS__:000000000000       REGBUS:000000000000     PAMM__:000000000000
    * PAMD__:00000084A040     * MEMREQ:000000908A00     ARBUS_:000000000000
      ARADR_:000000001010       ABUS__:000100000000     STALL_:000019FF0020
      NATRAM:000000043030     * MDBUSM:000000001000   * MDBUSI:000000001000
    * IOPSEL:00000000000C       INCR__:000000000035     IBUF__:0000000052C0
      IBDBUF:021001DEFF82     * IVABUS:000000001014     DBUS__:0000FFFF0000
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
      V$6208 ICA LID ISTALL H               V$B227 -MCC7 PA LAT LD H
Lo sigs:
      V$C216 -MCC MD RESP A H               V$C169 MCC DEST CODE 0 H
SBSM>>>>
```

Figure 7-39 STATESTEP Display -- Phase 2

```
        Tracing from count: 3 (3.)
          State-step count: 4 (4.)  Phase: T3
Registers:
      OPMCF_:00000000000A       OPCODE:000000000D6    * OPBUS_:000000000000
      EVABUS:00000000FFFF     * EDPPE_:000B8100012E     EBFLSH:00000000001C
    * WBUS__:00000000FFFF       REGBUS:000000000000     PAMM__:000000000000
    * PAMD__:00000084A000     * MEMREQ:000000108A00     ARBUS_:000000000000
      ARADR_:000000001010       ABUS__:000100000000   * STALL_:000018FF0200
    * NATRAM:00000000083F       MDBUSM:000000001000     MDBUSI:000000001000
      IOPSEL:00000000000C     * INCR__:000000000036   * IBUF__:000000009F17
    * IBDBUF:021009523290       IVABUS:000000001014   * DBUS__:0000AD60E8AC
      FABUS_:0000FFFFFFFE       PAMUX_:000000000004
Hi sigs:
      V$6208 ICA LID ISTALL H               V$B227 -MCC7 PA LAT LD H
Lo sigs:
      V$C216 -MCC MD RESP A H               V$C169 MCC DEST CODE 0 H
SBSM>>>>
>>>>UNHANG
>>>>START 1000
?MCP-I-CPSRUN, CPU is still running
>>>>HALT
        CPU stopped, INVOKED BY CONSOLE (CSM code 11)
        PC 00001002
>>>>N
        PC 00001009>>>>
        PC 0000100B>>>>
        PC 0000100E>>>>
        PC 00001000>>>>
>>>>
```

Figure 7-40 STATESTEP Display -- Phase 3

Figure 7-41 summarizes the microstepping process using the same test routine. It uses the START/STEP command followed by the MIC command to step the routine through one complete iteration. Note how the EBox micro-PC sequence begins to repeat at location 1BC5. After exiting space-bar mode, the UNHANG command is used to reinitialize the EBox micro-PC before executing the START command to restart the entire test routine.

```
>>>>START/STEP 1000
>>>>MIC
MBOX=0018 EBOX=1081 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1082 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1083 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1085 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1086 FBOXA=0004 FBOXM=0006 IBOX=000A >>>>
MBOX=0018 EBOX=1088 FBOXA=0007 FBOXM=0007 IBOX=002A >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00CC >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00D6 >>>>
MBOX=0018 EBOX=1BC5 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0019 EBOX=1B06 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0019 EBOX=0EC1 FBOXA=0004 FBOXM=0006 IBOX=00D0 >>>>
MBOX=0019 EBOX=0FC0 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0018 EBOX=1BC1 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00D6 >>>>
MBOX=0019 EBOX=1B90 FBOXA=0004 FBOXM=0006 IBOX=00CC >>>>
MBOX=0018 EBOX=1BFE FBOXA=0004 FBOXM=0006 IBOX=00CC >>>>
MBOX=0018 EBOX=1A71 FBOXA=0004 FBOXM=0006 IBOX=00CC >>>>
MBOX=0019 EBOX=15F6 FBOXA=0004 FBOXM=0006 IBOX=000A >>>>
MBOX=0018 EBOX=0200 FBOXA=0007 FBOXM=0007 IBOX=002A >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00C8 >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0019 EBOX=1B00 FBOXA=0004 FBOXM=0006 IBOX=00D6 >>>>
MBOX=0018 EBOX=1BC5 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
MBOX=0019 EBOX=1B06 FBOXA=0004 FBOXM=0006 IBOX=00CA >>>>
>>>>UNHANG
>>>>START 1000
?MCP-I-CPSRUN, CPU is still running
>>>>HALT
        CPU stopped, INVOKED BY CONSOLE (CSM code 11)
        PC 00001002
>>>>N
        PC 00001009>>>>
        PC 0000100B>>>>
        PC 0000100E>>>>
        PC 00001000>>>>
>>>>EXIT
>>>
```

Figure 7-41 MACRO Routine Microstepping Summary

7.6.5 Breakpoints

The HEX command set has three commands that permit additional control of the sequences when microstepping in trace mode. These commands enable stopping the machine when the following conditions occur.

1. An SDB register contains a specific value or changes value

2. An SDB signal assumes a certain value (1 or 0) or changes state

There are two types of breakpoints, OR breaks and AND breaks (OBREAK/ABREAK), supported by two separate tables within the HEX debugger software. Up to four breakpoints may be set in each table. The following three commands are used to maintain these tables.

1. SET BREAK
2. CLEAR BREAK
3. SHOW BREAK

Use the HELP command to display detailed descriptions for each of the above commands. For example, typing **HELP HEX SET BREAK** will display the instructions for using the SET BREAK command.

When tracing with breakpoints set, the TMICRO and TSTATE commandS will stop and report the current machine state only when a break condition is met. This occurs when all breakpoints in the AND table are true or when any condition in the OR table is true.

The following discussion assumes that the user is microstepping the same macro routine discussed in the previous sections. Figure 7-42 illustrates how to use the SET BREAK command to stop the machine whenever a microword is encountered (where the MDBUSI register contains a 2000). Note how the SHOW TRACE command indicates the presence of an entry in the OR break table.

```
>>>>SET OBREAK MDBUSI VALUE:2000
>>>>SHOW BREAK
A_breaks set: none
O_breaks set:
    8029  MDBUSI  Value:2000
>>>>
```

Figure 7-42 Setting an SDB Register Value Breakpoint

After setting the breakpoint, the routine is started at location 1000 and the TMIC command is used to initiate the trace, as shown in Figure 7-43. Note how the machine only stops and displays the trace data when MDBUSI=2000.

```
>>>>START/STEP 1000
>>>>TMIC
        Tracing from count: 0 (0.)
         Micro-cycle count: 8 (8.)  Phase: T3
Micro PC's:
            E:1B00      M:0019      I:00CA      FA:0004      FM:0006
Registers:
    * OPMCF  :000000000000A      OPCODE:0000000000C0    * OPBUS  :000000001004
      EVABUS:000000000000      * EDPPE  :000B8020012C      EBFLSH:00000000001C
    * WBUS   :00000000FFFB       REGBUS:000000000000      PAMM   :000000000000
    * PAMD   :00000094A040     * MEMREQ:00000090CA00      ARBUS  :000000000000
    * ARADR  :000000001000     * ABUS   :000F00000000    * STALL  :00001CFF0020
    * NATRAM:000000043826     * MDBUSM:000000002000    * MDBUSI:000000002000
      IOPSEL:000000000000      * INCR   :000000000026    * IBUF   :000000009FD0
    * IBDBUF:022509EA33C5     * IVABUS:000000001008    * DBUS   :0000AC2B602F
      FABUS_:0000FFFFFFFE
Hi sigs:
Lo sigs:
    * V$C216 -MCC MD RESP A H          * V$6208 ICA LID ISTALL H
      V$C169 MCC DEST CODE 0 H
SBSM>>>>
        Tracing from count: 8 (8.)
         Micro-cycle count: 16 (22.)  Phase: T3
Micro PC's:
            E:1B00      M:0019      I:00CA      FA:0004      FM:0006
Registers:
    * OPMCF  :000000000000      * OPCODE:000000000017      OPBUS  :000000001004
    * EVABUS:000000001000        EDPPE  :000B8020012C      EBFLSH:00000000001C
    * WBUS   :0000FFFFFFFF       REGBUS:000000000000      PAMM   :000000000000
    * PAMD   :000000940040     * MEMREQ:00000090C000      ARBUS  :000000000000
      ARADR  :000000001000       ABUS   :000F00000000    * STALL  :000018FF0020
      NATRAM:000000043826       MDBUSM:000000002000      MDBUSI:000000002000
      IOPSEL:000000000000       INCR   :000000000026      IBUF   :000000009FD0
      IBDBUF:022509EA33C5       IVABUS:000000001008      DBUS   :0000AC2B602F
      FABUS_:0000FFFFFFFE
Hi sigs:
Lo sigs:
      V$C216 -MCC MD RESP A H          V$6208 ICA LID ISTALL H
      V$C169 MCC DEST CODE 0 H
SBSM>>>>
>>>>
```

Figure 7-43 Stop On Register Value Breakpoint Display

The next example shows how to set an ABREAK entry using an SDB signal name. The display shown in Figure 7-44 shows how to set an ABREAK so that the machine will stop each time the signal -MCC7 PA LAT LD H changes state. Note also that this display shows two OBREAKS are to be included in the trace, namely MDBUSI=2000 and MDBUSM=9F175352.

```
>>>>SET ABREAK V$B227
>>>>SHOW BREAK
A_breaks set:
    160A  V$B227 -MCC7 PA LAT LD H  Change
O_breaks set:
    8029  MDBUSI  Value:2000
>>>>
```

Figure 7-44 Setting an SDB Signal Change Breakpoint

Figure 7-45 shows the display that results when tracing the routine with the ABREAK to stop whenever -MCC7 PA LAT LD H changes state (refer to the asterisk adjacent to the signal name). Note, however, that in the third microstep the machine stopped because it found MDBUSI=2000 rather than a change in state of -MCC PA LAT LD H.

Finally, Figure 7-46 shows an example of how to use the CLEAR BREAK command to clear breakpoints. Note the use of the ALL argument to clear all the breakpoints with a single command.

In summary, the breakpoint facilities provide a powerful tool for more detailed microsequencer tracing during troubleshooting. They permit the user to trace through lengthy microsequences, stopping only when the specific conditions occur.

```
>>>>START/STEP 1000
>>>>TMIC
        Tracing from count: 16 (22.)
         Micro-cycle count: 17 (23.)  Phase: T3
Micro PC's:
            E:1081       M:0018       I:00CA       FA:0004       FM:0006
Registers:
    * OPMCF_:000000000001     * OPCODE:0000000000C0     * OPBUS_:00000000100F
    * EVABUS:000000000000     * EDPPE_:000B8000012E       EBFLSH:00000000001C
    * WBUS__:0000041F0000       REGBUS:000000000000       PAMM__:000000000000
    * PAMD__:000000841000     * MEMREQ:00000010B100       ARBUS_:000000000000
    * ARADR_:000000000010     * ABUS__:000800000000     * STALL_:00000CFF0000
    * NATRAM:00000000083F     * MDBUSM:000070000004     * MDBUSI:000070000004
      IOPSEL:000000000000     * INCR__:000000000052     * IBUF__:000000009F17
    * IBDBUF:021009D23290     * IVABUS:000000001013     * DBUS__:0000AD60E8AC
      FABUS_:0000FFFFFFFE
Hi sigs:
    * V$C216 -MCC MD RESP A H           * V$6208 ICA LID ISTALL H
Lo sigs:
      V$C169 MCC DEST CODE 0 H
SBSM>>>>
        Tracing from count: 17 (23.)
         Micro-cycle count: 1D (29.)  Phase: T3
Micro PC's:
            E:1B00       M:0019       I:00C8       FA:0004       FM:0006
Registers:
    * OPMCF_:00000000000F       OPCODE:0000000000C0     * OPBUS_:000000001000
      EVABUS:000000000000     * EDPPE_:000B0040012D       EBFLSH:00000000001C
    * WBUS__:000000001004       REGBUS:000000000000       PAMM__:000000000000
    * PAMD__:00000080F080     * MEMREQ:000001300F00       ARBUS_:000000000000
    * ARADR_:000000001000     * ABUS__:000F00000000     * STALL_:00000A0200C0
    * NATRAM:000000044985     * MDBUSM:00009FD053D4     * MDBUSI:00009FD053D4
      IOPSEL:000000000000     * INCR__:000000000094     * IBUF__:0000000053D4
    * IBDBUF:0000016EFFFE     * IVABUS:000000001004     * DBUS__:00008FFFFFFB
      FABUS_:0000FFFFFFFE
Hi sigs:
    * V$C169 MCC DEST CODE 0 H
Lo sigs:
    * V$C216 -MCC MD RESP A H           * V$6208 ICA LID ISTALL H
SBSM>>>>
        Tracing from count: 1D (29.)
         Micro-cycle count: 1E (30.)  Phase: T3
Micro PC's:
            E:1B00       M:0019       I:00CA       FA:0004       FM:0006
Registers:
    * OPMCF_:000000000000       OPCODE:0000000000C0     * OPBUS_:000000001004
      EVABUS:000000000000     * EDPPE_:000B8020012C       EBFLSH:00000000001C
    * WBUS__:0000FFFFFFFF       REGBUS:000000000000       PAMM__:000000000000
    * PAMD__:000000940040     * MEMREQ:00000090C000       ARBUS_:000000000000
      ARADR_:000000001000       ABUS__:000F00000000     * STALL_:000018FF0020
    * NATRAM:000000043826     * MDBUSM:000000002000     * MDBUSI:000000002000
      IOPSEL:000000000000     * INCR__:000000000026     * IBUF__:000000009FD0
    * IBDBUF:022509EA33C5     * IVABUS:000000001008     * DBUS__:0000AC2B602F
      FABUS_:0000FFFFFFFE
Hi sigs:
Lo sigs:
      V$C216 -MCC MD RESP A H           V$6208 ICA LID ISTALL H
    * V$C169 MCC DEST CODE 0 H
SBSM>>>>
>>>>
```

Figure 7-45 Stop on SDB Signal Change Breakpoint Display

```
>>>>SHOW BREAK
A_breaks set:
    160A  V$B227 -MCC7 PA LAT LD H  Change
O_breaks set:
    8029  MDBUSI  Value:2000
>>>>CLEAR OBREAK MDBUSI
>>>>SHOW BREAK
A_breaks set:
    160A  V$B227 -MCC7 PA LAT LD H  Change
O_breaks set: none
>>>>
>>>>
>>>>CLEAR ABREAK ALL
>>>>SHOW BREAK
A_breaks set: none
O_breaks set: none
>>>>
```

Figure 7-46 Clearing Breakpoints

APPENDIX A
CONSOLE COMMAND SETS

A.1 OVERVIEW

This appendix provides an overview of all the commands available to the service engineer when using the console to test and troubleshoot VAX 8600 and VAX 8650 systems. It organizes the commands into functional sets according to the level and type of testing being performed.

A.1.1 Command Summary

Figure A-1 summarizes each of the seven, distinct command sets available for testing the systems. This chart also shows how to switch from one command set to another. Depending upon the current context of the console software, you may have access to more than one command set, as shown in Table A-1.

The following sections will describe command syntax and how to use the on-line HELP facility to display detailed command descriptions for the general, macro, diagnostic, and microhardcore command sets. For descriptions of the PROM and EDOBA commands, refer to the appropriate sections in Chapters 2 and 3, respectively.

A.1.2 Console Command Syntax

The information in this section applies to all command sets in the console except for the PROM and EDOBA. Command syntax is illustrated using the following conventions.

[ ]	Brackets are used to indicate an optional switch or keyword.
{ }	Braces are used to show a list of choices from which one item must be selected.

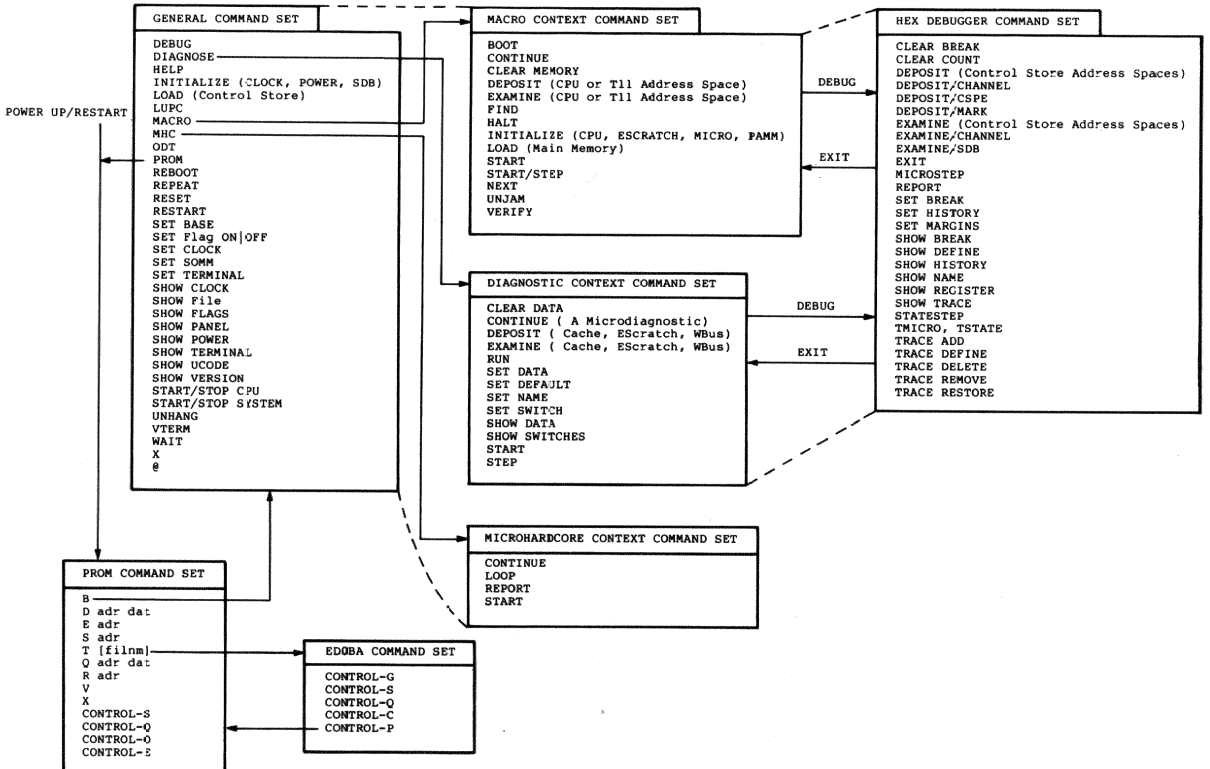


Figure A-1 VAX 8600 Console Command Set Summary

Table A-1 Command Set/Prompt Relationship

Context	Debug	Prompt	Command Sets Available
MACRO	OFF	>>>	GENERAL, MACRO
MACRO	ON	>>>>	GENERAL, MACRO, HEX
DIAGNOSTIC	OFF	DC>	GENERAL, DIAGNOSTIC
DIAGNOSTIC	ON	DC>>	GENERAL, DIAGNOSTIC, HEX
MHC	OFF	MH>	GENERAL, MHC
MHC	ON	MH>	GENERAL, MHC

The console parses commands made up of one or more of the following parts.

- Command -- Commands are the first item on any command line input.
- Switch -- Switches begin with the slash (/) character and select some option associated with the command. Switches must appear in proper sequence with other parts of a command line, as shown by the syntax rules describing a command. Switches may be separated from commands, keywords, or other switches by space or tab characters.
- Switch argument -- Some switches require associated numerical arguments (specified as "/switch:number").
- Keyword -- Keywords appear to the right of the command part and are arguments to the command. Keywords must be separated from commands, switches, or other keywords with one or more spaces or tabs. Keywords can be numeric values or symbolic names.
- Keyword argument -- Some keywords require associated numeric or symbolic arguments (such as "keyword:argument").

The following notes describe the main characteristics of the console with respect to command input.

- A single console command line cannot exceed 80 characters, counting the terminating <RETURN> and <LINEFEED> characters. Truncation of excessive input is done and the command line is parsed normally.
- The DELETE key will delete the previously typed character.
- The <CTRL/U> character will flush the console input buffer and begin a fresh line for command input.

- The <CTRL/R> character is ignored from the local terminal. From the remote terminal, it causes the current command line to be retyped on the following line.

- The <CTRL/O> character acts as a toggle switch that, when set, suppresses all output to the issuing terminal (the code continues to run). The state of the <CTRL/O> flag is reset at the completion of any command.

- Upper- and lowercase input is interchangeable.

- A parser error message results when (1) an unrecognized command is entered, (2) an invalid switch or keyword is used with a command, or (3) when an invalid switch argument or keyword argument is used. A parser error also occurs when invalid hexadecimal numbers are entered, or, in some cases, when the allowable range of numeric input is exceeded. All numeric input is taken as hexadecimal, except for the SET CLOCK FREQUENCY command (decimal). Numeric output is also hexadecimal unless indicated as decimal by either a decimal point (.) or unit name (e.g., megacycles, volts, etc.).

- An optional radix specifier prefix (%O, %D, %X) may be used to override the default radix specified above (i.e., %D100 implies decimal 100).

- Abbreviations are formed by dropping characters from the end of a command, switch, or keyword. In general, commands may be abbreviated to the point where they are no longer unique among other commands of the current context. The exception to this is that some commands, like EXAMINE and DEPOSIT, are defined architecturally as having single (or double) character equivalents. Thus, although the command input E may not be unique in the current context, it is treated as an equivalent to EXAMINE. Table A-2 shows a complete list of such command equivalencies.

- Keywords and switches may also be abbreviated to the point where they become no longer unique among other keywords and switches for that command.

Table A-2 lists architecturally defined command and switch equivalencies.

Table A-2 Architecturally Defined Commands

Short Form	Long Form
B	BOOT
C	CONTINUE
D	DEPOSIT
E	EXAMINE
F	FIND
H	HALT
I	INITIALIZE
L	LOAD
N	NEXT
S	START
SE	SET
SH	SHOW
U	UNJAM
V	VERIFY
W	WAIT
/P	/PHYSICAL

A.1.3 The HELP Facility

There is an extensive, on-line HELP facility that displays detailed information about command format and available switches. It also describes completely the purpose and use of the commands and switches. For the VAX 8600/8650 novice, the HELP command provides detailed information about each command. This section describes how to use the HELP facility and includes a couple of examples of how it is typically used. First, look over the syntax and description, as shown below.

```
Syntax: HELP
        HELP category
        HELP category topic [ subtopic ]
```

Description:

This command provides on-line help on the various console software commands (and other topics) and their proper usage.

The HELP command will display all available HELP categories and wait for the user to select one. After the category is selected, all available HELP topics in the selected category are displayed, and once again the user is asked to make a selection. The HELP text on the topic is then displayed.

The "subtopic" argument is used to differentiate certain generic command topics such as SHOW, SET, EXAM, etc., which use optional switch or keyword arguments.

Figure A-2 shows how the system responds when HELP is typed. Note that there are eight categories of HELP information available, five for the command sets plus three for information on the syntax, front panel, and supported register names. Note, also, that the program prompts a second time for the category you wish help on. It displays "Select" and waits for the user to provide the input. In this example, the program displayed all the commands when HELP was typed, prompted with Select, and waited for the user to enter input. The user typed MACRO to get the list of the macro context commands. When the list was displayed, the program again prompted with "Select"; the user typed CSM to get a description of the CSM command.

```
>>>HELP
 help available:
CONSOLE_COMMAND_SYNTAX    DIAGNOSTIC_COMMAND_SET    FRONT_PANEL
GENERAL_COMMAND_SET       HEX_DEBUG_COMMAND_SET     MACRO_COMMAND_SET
MICROHC_COMMAND_SET       REGISTERS_SUPPORTED
Select MACRO
MACRO help available:
BOOT                      CLEAR                     CONTINUE
CSM                       DEPOSIT                   EXAMINE
FIND                      HALT                      INITIALIZE
LOAD                      NEXT                      START
START_STEP                UNJAM                     VERIFY
Select CSM
CSM
                Console support microcode (CSM)
        Many of the functions of the MACRO context depend on Console
        Support Microcode (CSM).  CSM is a microcode package which allows
        the console program to communicate with the CPU by means of data
        packets passed through the CBUS.
        CSM is divided into microcode overlays which are managed by the console
        program. Depending on the type of packet function being used the console
        loads and starts the correct microcode overlay.  To improve performance
        of the overlay loading all CSM overlays are buffered in T-11 memory
        after the first load from the console pack.
>>>
```

Figure A-2 HELP Command -- Format 1

Figure A-3 shows how to use the second format by providing the HELP command with an argument, MACRO. At the "Select" prompt, the user typed LOAD to display information about the LOAD command in the MACRO context.

Figure A-4 shows how, with a single command, the third format can be used to obtain information about the START command within the macro context.

Figure A-5 shows an example of how to obtain information about the SHOW UCODE command in the GENERAL_COMMAND_SET. Note the use of abbreviations. As long as the user types enough characters to make the item unique, most abbreviations are allowed.

```
>>>HELP MACRO
MACRO help available:
BOOT                    CLEAR                   CONTINUE
CSM                     DEPOSIT                 EXAMINE
FIND                    HALT                    INITIALIZE
LOAD                    NEXT                    START
START_STEP              UNJAM                   VERIFY
Select LOAD
LOAD
    Command Syntax:
        LOAD [ /START:hex_adr ] filename[.exe]
    Description:
        This command is used to load main memory with binary data taken
        from the specified file.  If the /START switch is used then the
        data is loaded starting at the specified 'hex_adr', otherwise the
        data is loaded starting at location zero.
>>>
```

Figure A-3 HELP Command -- Format 2

```
>>>HELP MACRO START
START
    Command Syntax:
        START [ hex_address ]
    Description:
        This command is essentially the same as the CONTINUE command except
        that if a 'hex_address' is specified it is used as the current PC
        at which to start the VAX processor.  If the CPU is already running
        then this command simply re-enters PIO mode without affecting the
        running CPU, even if a 'hex_address' is included.
>>>
```

Figure A-4 HELP Command -- Format 3, Example 1

```
>>>HELP GEN SHOW UCODE
SHOW UCODE
    Command Syntax:
        SHOW UCODE
    Description:
        This command displays the state of all control_stores and rams in
        terms of:
        o  the current (last loaded) .BPN file name.
        o  the default .BPN file name for that control store.
        o  the ucode revision of the currently loaded microcode.
        o  the ram status:   bit 15 -- 1 = modified
                             bit 14 -- 1 = failed verification
                             bit 13:00 --  parity error count
        The "modified" bit, when set, indicates that one or more locations
        have been deposited or otherwise altered after that last load of that
        control store.  Note that this bit is always set for the E_SCratch ram.
        The "failed verification" bit, when set, indicates that the VERIFY
        command detected one or more discrepancies in the control store data.
        This could be the result of a multiple (undetected) control store
        parity error.
        The "Parity error count" field contains the total number of detected
        parity errors for the control store, whether they were corrected or
        not.  MCF, CTX, CYC, and ACC rams always show 0's in this field.
>>>
```

Figure A-5 HELP Command -- Format 3, Example 2

A few more examples will illustrate how to use Figure A-1 with the HELP command to display information about the console command sets.

Let's assume the user wants to examine cache. Looking over the chart in Figure A-1, it's clear that the EXAMINE command in the DIAG context is the one that's needed. The following command will display the needed information.

```
DC> HELP DIAG EXAM
```

Now suppose the user wants to restart the CPU clock. It's in the GENERAL command set, so the following command will suffice.

```
>>>HELP GEN STA CPU
```

Finally, suppose the user wants to set a microcode breakpoint. From Figure A-1, it appears to be a HEX command, so typing the following command should provide the help needed.

```
>>>HELP HEX SET BRE
```

Obviously, the user should now find a VAX 8600 or 8650 system and spend several minutes practicing with the HELP command. Try to find out information about the commands using Figure A-1 and the system itself. Let the machine do the look up and page turning. Isn't that what computers are for?

APPENDIX B
CONSOLE SUPPORT MICROCODE

B.1 INTRODUCTION

Part of the system microcode loaded into the EBox control store during CPU initialization contains a set of microroutines designed to support the CBus communication interface between the VAX CPU and the console software. These microroutines, the Console Support Microcode (CSM), consist of a permanently resident root segment and a set of overlays that are loaded from the console when a command is to be executed. This appendix provides a general description of the CSM.

B.2 CSM OVERLAYS

The following overlay files reside in the RL02 and are read in and loaded into the EBox control store overlay segment when a command is first invoked. The command overlays are cached in T-11 RAM and do not require an RL02 access each time they are invoked. Table B-1 summarizes the CSM overlay files that reside on the RL02 pack.

Figure B-1 summarizes the functional relationship between the console and the EBox during console/CSM communication. The Macro Control Program (MCP) runs in the T-11 while CSM runs in the EBox control store. Information packets (command/response) are transported over the CBus using the DC022 register file, while the CSM overlays are loaded into the EBox control store via the SDB. Note also how CSM uses reserved locations in the EBox scratchpad to execute the command. A simple example will suffice to summarize how the overlays work.

Take, for example, the case where the user types the following command.

```
>>>FIND/RPB
```

The console software reads and parses the command to determine the action required. Based on the command typed, it loads the EBox control store overlay segment with the contents of the CSM023.BPN overlay using the SDB, and then sends a command to the root segment of CSM via the CBus. The root segment of CSM then transfers control to the overlay segment to execute the FIND/RPB command and returns the result back to the console software via the CBus. From here, the result is formatted and displayed on the console terminal. After executing the command, the CSM returns to the root segment console loop to await the next command.

Table B-1 CSM Overlay Files

File Name	Overlay Function
CSM001.BPN	Read_IPR_Group_0
CSM002.BPN	IRD
CSM003.BPN	Continue_Microflow
CSM004.BPN	Examine_EBox_Scratchpad_Register
CSM005.BPN	Examine_EBox_Misc._Register
CSM006.BPN	Examine_IBox_Misc._Register
CSM007.BPN	Deposit_EBox_Scratchpad_Register
CSM008.BPN	Deposit_EBox_Misc._Register
CSM009.BPN	Examine_Physical_CPU_Memory
CSM010.BPN	Flush_IBox_and_Loop
CSM011.BPN	Deposit_Physical_CPU_Memory
CSM012.BPN	Translate_and_Test_Virtual_Read
CSM013.BPN	Find_64kb
CSM014.BPN	Read_Internal_MBox_Register
CSM015.BPN	Read_Internal_EBox_Register
CSM016.BPN	Write_Internal_MBox_Register
CSM017.BPN	Write_Internal_EBox_Register
CSM018.BPN	Clear_Internal_MBox_Register
CSM019.BPN	Translate_and_Test_Virtual_Write
CSM020.BPN	RLOG_Unwind_and_VA
CSM021.BPN	Flush_IBox
CSM022.BPN	Access_IOA
CSM023.BPN	Find_RPB
CSM024.BPN	Examine_PC
CSM025.BPN	Read_IPR_Group_1
CSM026.BPN	Read_IPR_Group_2
CSM027.BPN	Read_IPR_Group_3
CSM028.BPN	Write_IPR_Group_0
CSM029.BPN	Write_IPR_Group_1
CSM030.BPN	Write_IPR_Group_2
CSM031.BPN	Write_IPR_Group_3
CSM032.BPN	Write_IPR_Group_4
CSM033.BPN	Read_IPR_Group_4
CSM034.BPN	Write_IPR_Group_5
CSM035.BPN	Write_IPR_Group_6
CSM036.BPN	Clear memory command
CSM040.BPN	CSM CPU power-up sequence
CSM041.BPN	CSM CPU power-up sequence
CSM042.BPN	Makes MARK bit breakpoints easier for the user

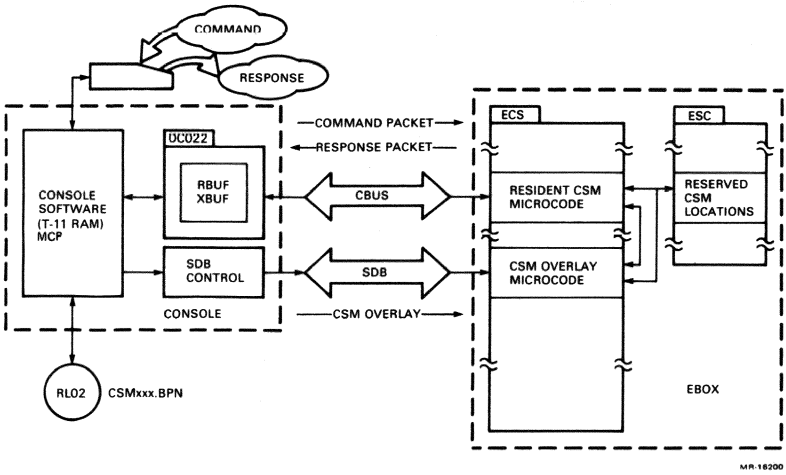


Figure B-1 Console/CSM Communication

B.3 DC022 REGISTERS

The DC022 dual-port RAM file on the console module is the key link in the communication path between the console and the EBox. When the console sends commands to the EBox, it writes specified locations and sets a flag to signal CSM (which is running in the EBox) to read the information. CSM then reads the same locations to retrieve the command and its associated arguments, and then clears the flag when done.

When CSM needs to send a response back to the console, it loads the response into specified locations and clears a flag to signal the console to read and display the response on the console terminal. After reading the information, the console sets the flag to indicate it is ready to accept another response.

Ten locations in the DC022 RAM file are reserved for this console/CSM communication. Five are for sending command packets to the EBox from the console and five are for sending responses from the EBox to the console.

CONSOLE --> CSM: Locations 06 through 09 hold four bytes of data (32 bits)
Location 0A holds 8 bits of control information

CONSOLE <-- CSM: Locations 16 through 19 hold four bytes of data (32 bits)
Location 1A holds 8 bits of control information

The format of the information in these locations is as follows.

1. (0A) RBUFC -- Contains encoded control information, as shown below.

   - BIT<7> -- The DONE bit is set by the console and cleared by CSM.
   - BIT<6> -- The PKT CTRL bit is set if a second packet is to follow and cleared if it is a checksum packet.
   - BIT<5:0> -- Contains the command code.

2. (06) RBUF0 -- Contains byte 0 of the data.
3. (07) RBUF1 -- Contains byte 1 of the data.
4. (08) RBUF2 -- Contains byte 2 of the data.
5. (09) RBUF3 -- Contains byte 3 of the data.
6. (1A) XBUFC -- Contains control and status information, as encoded below.

   - BIT<7> -- The READY bit is set by the console and cleared by CSM.
   - BIT<6> -- The PKT CTRL bit is set if a second packet is to follow and cleared if it is a checksum packet.
   - BIT<5:0> -- Contains the response code, as summarized in Table B-2.

7. (16) XBUF0 -- Contains byte 0 of the data.
8. (17) XBUF1 -- Contains byte 1 of the data.
9. (18) XBUF2 -- Contains byte 2 of the data.
10. (19) XBUF3 -- Contains byte 3 of the data.

A running checksum is calculated on each location loaded by the sender and then checked by the receiver. Checksum errors that CSM detects simply cause an error response to be returned to the console. Checksum errors detected by the console are displayed on the console terminal and the command is aborted. Table B-2 lists the response codes that CSM returns.

Table B-2 CSM Response Codes

Encoding	Error	Packets	Description
0	Y	1	Bad Checksum -- The computed checksum of the RBUF packets does not match the console checksum.
1	Y	1	Bad Virtual Address -- A good translation of the virtual address is unobtainable.
2	Y	1	ACV Condition -- The console does not have access to part or all of the data specified to perform the address translation. This error condition may be remedied if the PSL<CUR_MOD> field is raised to kernel mode.
4	Y	1	Hardware Error Abort -- A hardware error occurred before or during the requested CSM command. Error information, possible a machine check image, has been created in the EBox scratchpad RAM for console inspection. The requested CSM command is aborted.
5	N	1-2	Successful Completion with Data -- The command has been successfully completed. The first packet contains the zero-extended data. The second packet contains a byte checksum in the low order byte position.
6	N	1	Successful Completion -- The command has been successfully completed.
7	Y	1	Can't Find 64kb -- The FIND_64Kb command was unable to find 64 Kbytes of good CPU memory. This is due to MBox hardware errors.
8	Y	1	Can't Find RPB -- The FIND_RPB procedure cannot find an RPB in memory.
9	Y	1	No Macro PC -- The EXAMINE_PC or RLOG_UNWIND_AND_VA commands found that ESA, ISA, and CPC were invalid. There is no macro-PC defined for the ISP.

B.4 CSM REGISTERS

When information is transferred between MCP and CSM, it is done in units of 8-bit bytes over the CBus. CSM formats these bytes into 32-bit longwords and stores them in dedicated locations in the EBox scratchpad. The CSM overlay then executes the commands using the contents of these locations and also stores the results of the commands in these locations. Table B-3 summarizes the scratchpad locations used.

Table B-3 CSM Registers

Name	ESC LOC	Purpose
CSM.STATUS	C0	Holds CSM status longword
CSM.VMQ	C1	Saves VMQ
CSM.SPADRSC	C2	Saves SPADR, SC register
CSM.PSLCC	C3	Saves PSL condition codes
CSM.EMD	C4	Saves EMD
CSM.CC1	C5	Holds first console packet data
CSM.CC2	C6	Holds second console packet data
CSM.CC3	C7	Holds third console packet data
CSM.T2	C8	Third temporary storage location

Location C0 in the EBox scratchpad holds the CSM.STATUS longword which is encoded as follows.

1. Bits <31:16> are set to indicate a non-EBox double error. A nonzero value in this field indicates that a double error was detected.

2. Bits <15:08> are set to indicate the general CPU status. A zero value in this field indicates that CSM is executing in the EBox, while a nonzero value indicates that ISP microcode is being executed.

3. Bits <07:00> contain an 8-bit entry reason, as summarized in Table B-4.

Table B-4 CSM Status Reason Codes

Code	Reason
0	CSM is not executing now. VAX-11 ISP microcode is executing.
4	Software error (Interrupt Stack Not Valid).
5	A non-EBox double error has occurred.
6	HALT instruction in kernel mode.
7	Software error (SCB vector<1:0> = 3).
8	Software error (SCB vector<1:0> = 2, no WCS microcode).
9	Software error (Pending Error on HALT).
A	Software error (CHMx with IS = 1).
B	Software error (CHMx vector<1:0> # 0).
10	A microbreak occurred and the console started CSM at CSM.ENTRY.MICRO.
11	The console set the CONSOLE HALT flag, which set the CONSOLE HALT PENDING flag, which caused CSM to start running in the microtrap vector at CSM.ENTRY.MACRO:.
15	The CPU has been powered up and CSM was started by the console at CSM.ENTRY.PO:.
16	The FIND_64KB and FIND_RPB procedures use this as a link to the CSM.ERROR: microcode. When a hardware error occurs, EHM will jump to CSM.ERROR: which then sees if a 16 is in CSM.STATUS. If so, CSM.ERROR jumps to the FIND_64KB/FIND_RPB restart address. If not, CSM.ERROR: will issue a hardware error abort response.

B.5 CONSOLE/CSM PROTOCOL

Figure B-2 shows the detailed sequencing of the command/response protocol that provides communication between the console and CSM. A simple example will summarize the handshaking that occurs. In the example, it is assumed the user has typed the EXAMINE PC command. The console software executes the following sequence.

1. Reads and parses the command
2. Stops the CPU clock and loads the required CSM overlay
3. Writes the command packet into RBUF
4. Starts the CPU clock and sets RBUFC<DONE>

Now, CSM responds by executing the following sequence.

1. Reads the command packet from RBUF
2. Clears RBUFC<DONE>
3. Transfers control to the CSM overlay to execute the command
4. Writes the response into XBUF
5. Clears XBUF<READY>

Finally, to complete the command, the console software executes the following sequence.

1. Reads the response packet from XBUF
2. Sets XBUF<READY>
3. Formats and displays the response on the console terminal
4. Returns to wait for the next command

This summary was based on no reported errors. Refer to Figure B-2 for a more detailed description, which includes how errors are handled.

There are errors, other than those shown in Figure B-2, that may occur during MCP-to-CSM communication. The most common error occurs when the EBox microcode gets hung in a loop and is unable to respond to the MCP request. This will cause MCP to time out. As a result, the current command will abort and display an appropriate error message. In general, there are two solutions to this problem.

1. Try the UNHANG command which will reinitialize the CPU clock and restart CSM at its restart address.
2. Reload and restart CSM with the INIT/CPU command.

Refer to Appendix K, Table K-15, for a more complete description of the errors detected by MCP during the communication protocol.

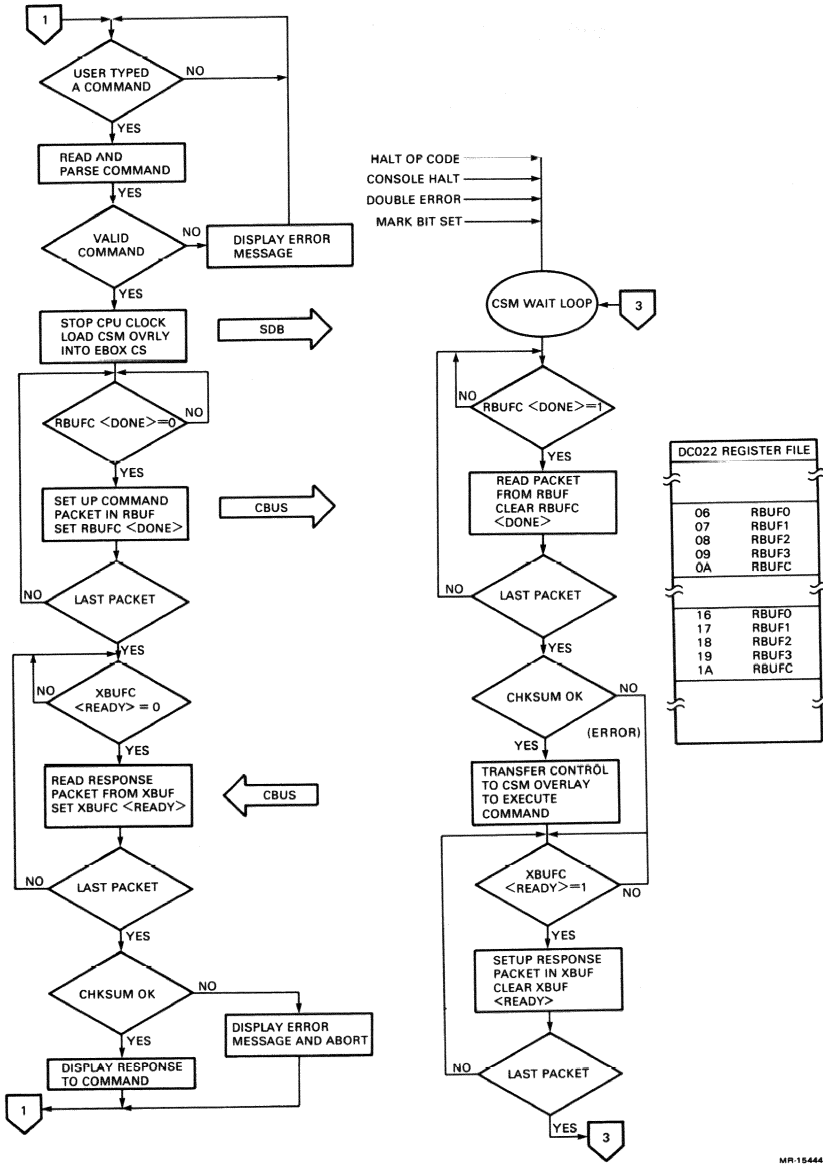


Figure B-2 Console/CSM Protocol Summary

APPENDIX C
DIAGNOSTIC SUPPORT MICROCODE

C.1 INTRODUCTION

This appendix provides an overview of the Diagnostic Support Microcode (DSM). DSM gets loaded into the EBox control store when switching to the diagnostic context with the DIAG command.

Refer to Figure C-1. The purpose of DSM is to provide a standard firmware interface between the microdiagnostic programs running in the EBox control store and the DCP running in T-11 RAM. DSM occupies the first 2048 locations in the EBox control store, from location 000(16) through location 7FF(16).

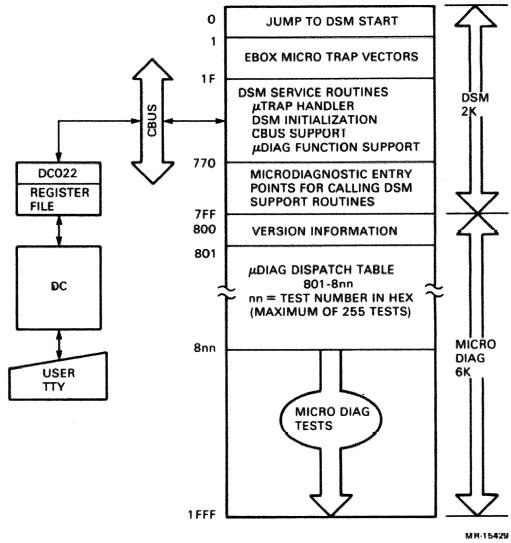


Figure C-1 EBox Control Store Utilization (DSM/Microdiagnostic)

Besides providing CBus service routines, DSM also includes a standard set of utilities that support common microdiagnostic functions. Locations 770(16) through 7FF(16) contain linkages between the microdiagnostic program and the specific support function requested.

Locations 001(16) through 1F(16) are set up to provide linkage to the DSM microtrap handlers in the event of expected or unexpected microtraps. The bulk of the DSM microcode resides between location 1F(16) and location 770(16), and includes the following routines.

- Microtrap handler
- DSM initialization microcode
- CBus support functions
- Microdiagnostic support functions

The main control module for each microdiagnostic is loaded into the upper 8 Kbyte segment of the EBox control store.

When a microdiagnostic is loaded into this segment, location 800(16) will contain program revision information. The data contained in this microword has the following format.

0000 NNnn MMDD YYYY RRRR rrr (hex representation of 92 bits)

This format has the following specifications.

- 0000 (2 bytes) -- The leftmost 4 hex digits are currently set to zero.

- NNnn (2 bytes) -- An encoding of the microdiagnostics EDK series name. The single ASCII character (let's call it "c") represented by the first 8 bits ("NN") corresponds to the diagnostic name, EDK"c"A. The other 8 bits of the field ("nn") are reserved for future name encoding. "00" is the current setting.

- MM (1 byte) -- The standard decimal number indicating MONTH of the year.

- DD (1 byte) -- The decimal day of the month.

- YYYY (2 bytes) -- The decimal year.

- RRRR (2 bytes) -- The major revision number. The major revision will be "1" for the initial diagnostic release.

- rrr (2 bytes) -- The minor revision number. The minor revision will be "0" for the initial diagnostic release.

- The rightmost 4 bits are not used and will always be zero.

Locations 801(16) through 8nn(16) contain a table of test dispatch microwords that provide the linkage between DSM and the actual microtests in the diagnostic.

The remaining locations provide the test linkages as follows:

801 contains the linkage to test 1
802 contains the linkage to test 2
.
.
.
8nn contains the linkage to test nn (the last test)

A maximum number of 255 separate test linkages are possible, depending upon the size of each test. Obviously, the number of possible tests is limited by the available control store space.

C.2 DCP/DSM COMMUNICATION

DCP, running in T-11 memory, communicates with DSM, running in the EBox control store, via a special CBus protocol defined for running microdiagnostics.

This protocol consists of six (DSM to DCP) or seven (DCP to DSM) byte packets and one "keep alive" byte, as shown in Table C-1.

Communication is achieved by passing packets between DCP and DSM using the protocol described above. Each byte in a packet is assigned a unique location in the DC022 register file on the console module which corresponds to the CBus address listed in Table C-1. (For example, the "keep alive" byte is in location 174020[8].)

Table C-1 DSM Packet Format

Symbol	CBus	Description
DSM$CONTROL	174000	DSM to DCP: Function code
DSM$CHECK	174003	packet checksum
DSM$D0	174004	data byte 0 (LSB)
DSM$D1	174005	data byte 1
DSM$D2	174006	data byte 2
DSM$D3	174007	data byte 3
DSM$CONTROL	174010	DSM to DCP: Function code
DSM$TARGET	174012	Extended Function code
DSM$CHECK	174013	packet checksum
DSM$D0	174014	data byte 0 (LSB)
DSM$D1	174015	data byte 1
DSM$D2	174016	data byte 2
DSM$D3	174017	data byte 3 (MSB)
DSM$ALIVE	174020	DSM alive status

C.2.1 DCP-to-DSM Protocol

Refer to Figure C-2. When DCP is ready to send a packet to DSM, it writes a nonzero function code into the DC$CONTROL byte in the DC022 register file, which is sensed by DSM (by polling). The nonzero byte in DC$CONTROL signals DSM to retrieve the packet, perform a checksum, and execute the command if no errors were detected. After executing the command, DSM clears DC$CONTROL to signal that the job is done. DCP, sensing zeroes in DC$CONTROL, then goes back and waits for further commands.

If a checksum error was detected, DSM writes an error code into DC$CONTROL to signal DSM that something is awry. When DCP senses the error code, it re-transmits the packet to try again if the retry count was not exceeded and the whole process is repeated. DCP will attempt six retries before it displays a checksum error message to inform the user, and aborts the current command.

Function codes sent to DSM by DCP include the following.

- Examine or deposit ESC
- Examine or deposit cache
- Examine or deposit a WBus register
- Start a microdiagnostic
- Continue a paused microdiagnostic
- Pause at the end of the current microtest

C.2.2 DSM-to-DCP Protocol

Refer to Figure C-3. The DSM to DCP communications is similar, except that it is interrupt driven. When DSM is ready to send a packet to DCP, it sets up the packet in the DC022 register file and loads DSM$CONTROL with the function code. It then enters a loop and waits for DCP to respond by reading the packet and clearing DSM$CONTROL. Meanwhile, back in the console, a CBus interrupt signals DCP to examine DSM$CONTROL. When DCP finds a nonzero DSM$CONTROL, it reads and checks the packet. If no errors are detected, DCP processes the DSM request and displays the response or executes the command. If a checksum error is found, DCP writes an error code into DSM$CONTROL to signal DSM to retransmit the packet. If unsuccessful after six retries, DCP displays a checksum error message and aborts the current command.

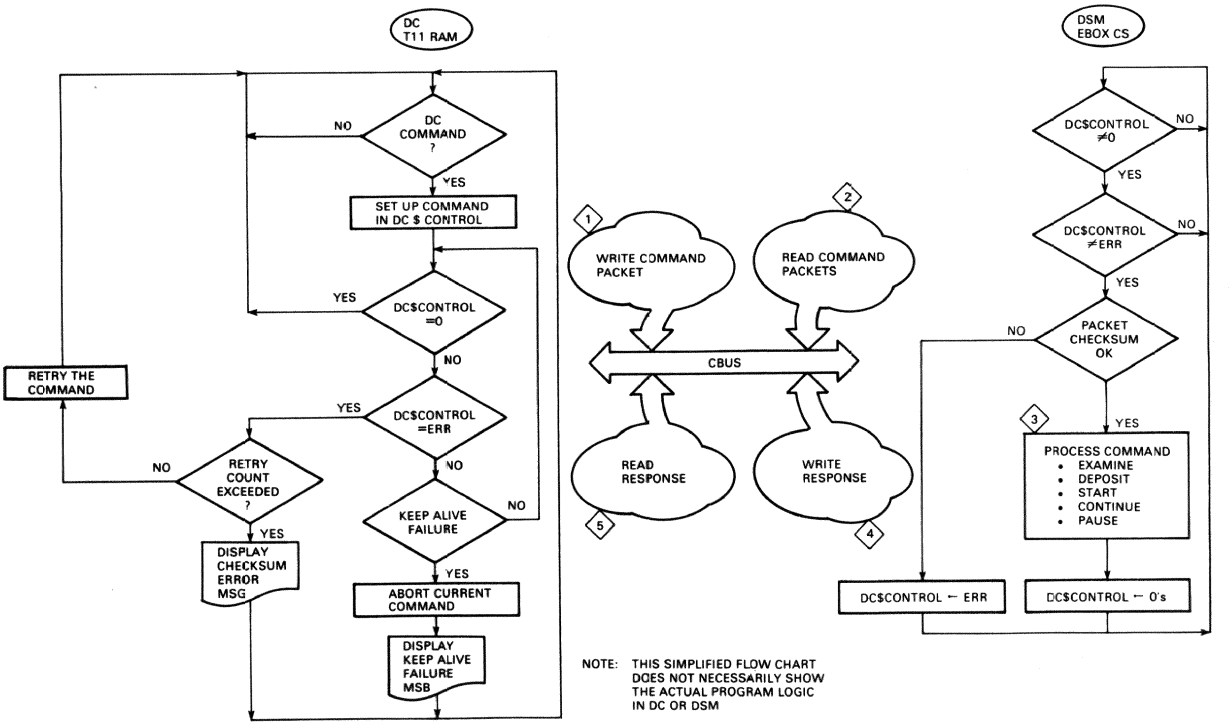


Figure C-2 DCP to DSM Protocol Overview

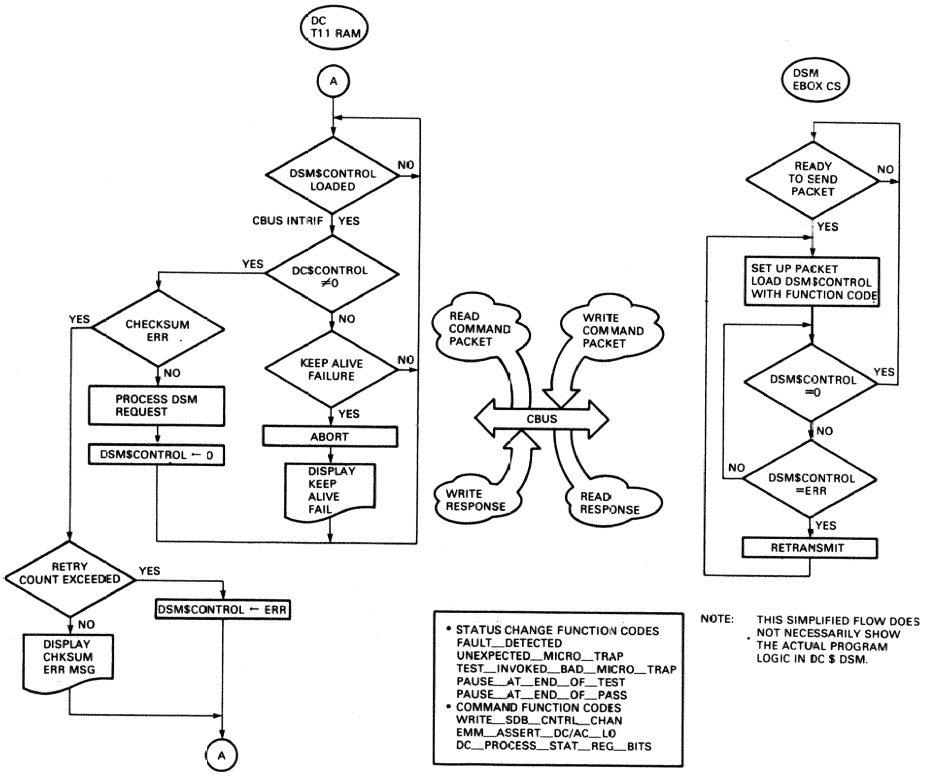


Figure C-3 DSM to DCP Protocol Overview

There are two categories of function codes possible when DSM sends packets to DCP, as described below.

C.2.2.1 Status Change Codes

1. FAULT_DETECTED -- A microdiagnostic has detected a fault and DSM has stopped executing tests in order to report the fault.

2. UNEXPECTED_MICRO_TRAP -- DSM has sensed an unexpected, fatal EBox microtrap.

3. TEST_INVOKED_BAD_MICRO_TRAP -- A logic fault caused an unexpected EBox microtrap.

4. PAUSE_AT_END_OF_TEST -- DSM has completed running a single test and has entered the "pause" state.

5. PAUSE_AT_END_OF_PASS -- DSM has completed all tests up through END_TEST and has stopped executing tests.

C.2.2.2 Command Function Codes

1. WRITE_SDB_CNTRL_CHAN -- Requests that DCP write the SDB control channels.

2. EMM_ASSERT_DC/AC_LO -- Requests that DCP have the EMM assert or deassert AC LO or DC LO.

3. DC_PROCESS_STAT_REG_BITS -- Requests that DCP manipulate bits in the console's miscellaneous status registers.

C.3 SUMMARY

DSM provides a firmware interface between the microdiagnostic test code running in the EBox and the DCP running in the console. DSM is loaded and started during diagnostic context initialization when the user types the DIAGNOSE command. Note this final word of advice before we end this discussion of DSM. There will be conditions, caused either by the nature of the hardware fault or operator errors that inadvertently disturb the test environment. The solution is to reinitialize diagnostic context with another DIAGNOSE command. This reloads and restarts DSM and should put the user back in business.

APPENDIX D
RL02 DISK ORGANIZATION

D.1 INTRODUCTION

The RL02 disk pack supplied with the system contains all the files necessary to support the operation of the diagnostic console subsystem. The files resident on this disk are important to the operation of the console front-end system. This appendix describes the general content and organization of the disk pack and the purpose of each major file group. In general, the files may be classified into 14 major groups, as summarized in Table D-1. The file name extension indicates the major group.

The following sections describe the file types classified according to system function and include examples of key text and command files. No attempt is made to include a complete catalogue of all the files, since it will change with each subsequent release of the RL02 pack. If needed, the user can retrieve a current listing of the RL02 directory three different ways.

1. Use the PROM/RT command from the general command set to gain access to the DIR and TYPE utilities within RT-11.

2. Use the VAX diagnostic supervisor's DIRECTORY command with the RL02 selected as the load device.

3. Use the operating system utilities.

One example will suffice to illustrate how this would be done using RT-11.

```
>>>PROM/RT          ;invoke RT-11

.DIR *.CDF          ;list all the .CDF files

    {RT-11 displays all the SDB signal name filenames}

.TYPE xxxxxx.CDF    ;display contents of xxxxxx.CDF

    {RT-11 displays file's contents}

.RUN ED0AA          ;reload and restart console software

    {Console software initialization messages}

>>>                 ;back in CIO mode MACRO context
```

Table D-1 RL02 File Types

Extension	General Purpose
.BPN	Binary coded files that contain microcode information loaded into the VAX CPU control RAMs, including both system and diagnostic microcode
.BIN	Binary coded file that is down-line loaded into the CI780
.CDF	ASCII files that contain SDB signal name information that is loaded into T-11 RAM during console software initialization
.COM	ASCII files that contain console command information used to automate test and control procedures
.DAT	ASCII files that contain system configuration information or snapshot data after a KAF
.DCI, .DCB	Binary coded files that contain isolation algorithm procedures used by the DCP
.EXE	Binary image files for macrodiagnostic programs loaded into VAX internal memory
.HLP	ASCII files that contain HELP information for console commands and diagnostics
.MEM	ASCII text files that contain miscellaneous memos that describe the use of the diagnostic system
.REV	ASCII file that contains revision information
.SAV	Binary coded T-11 program images
.SYS	Binary coded T-11 console software modules
.TXT	ASCII text files

D.2 T-11 BASED PROGRAMS

T-11 program files are identified by the .SAV extension. The two most important are ED0AA.SAV and ED0BA.SAV. ED0AA.SAV is the console software image that includes several program overlays to support all the console command sets (MACRO, DIAG, MHC, and HEX). ED0BA.SAV is the standalone T-11 console module diagnostic. In addition to these two, there are several .SAV files that support a minimum subset of RT-11 utilities. They are primarily used for console software maintenance, not normally used in the field.

D.3 MICRODIAGNOSTICS

There are over 200 files in this group that support loading and running of microdiagnostics and isolating faults when the microdiagnostics detect errors. The file types in this group are as follows.

- .COM -- Command files that automate loading and initializing the VAX CPU for running microdiagnostics.
- .BPN -- Binary files containing the microcode to be loaded for a microdiagnostic.
- .DCI -- Machine readable files containing the isolation algorithms used by DCP during fault isolation.
- .DCB -- Machine readable files containing additional clock-bursting information used by DCP during fault isolation.

For example, these are the files that support the IBox diagnostic, EDKRA.

```
EDKRA.COM          ;top level command file
EDKRE.COM          ;command file to set up ESCRATCH patterns
ESCINI.COM         ;command file to initialize ESCRATCH
EDKRC.COM          ;command file to set up CACHE patterns
EDKRI.BPN          ;test microcode loaded in IBox CS
EDKRU.BPN          ;diagnostic microcode loaded in EBox CS
EDKRA.DCI          ;isolation algorithm file
EDKRA.DCB          ;isolation algorithm file
```

D.4 MACRODIAGNOSTICS

Over 50 files reside on the disk to support loading and running VAX macrodiagnostics from the RL02. These programs are a subset of the VAX macrodiagnostic library. They were selected to permit macro testing of the VAX CPU and the load paths to the system load devices.

The macrodiagnostic files all have the .EXE extension and are loaded into VAX internal memory via the console front-end. During system installation, the system disk must be built from the system mag-tape device. The system magtape and disk load paths must be operational for this process. If there are load path hardware problems, the user will use the RL02-based macrodiagnostics to test and troubleshoot the load path devices.

Some of the more important programs follow.

- EVKAA.EXE -- Standalone VAX CPU hardcore diagnostic
- EDSAA.EXE -- VAX 8600/8650 diagnostic supervisor
- EVSBA.EXE -- VAX system autosizer

EVKAA tests the instructions used by the supervisor. The supervisor is then loaded and used to run EVSBA to size the system and set up the required device configuration tables. The supervisor can then be used to load and run the remaining CPU and I/O diagnostics to test the system in standalone mode.

Another important macroprogram included in this group is VMB.EXE, the boot loader program used to boot the operating system. During normal system start-up, VMB is loaded from the RL02 and used to load a secondary bootstrap loader from the system disk, which in turn loads and starts the operating system.

There is one file in this group, CI780.BIN, that contains microcode information to be down-line loaded into the CI780 during system start-up and when running the CI780 macrodiagnostics. It is critical to use the current revision level of the file when operating CI-based systems.

D.5 MISCELLANEOUS FILES

There are many other files on the RL02 disk pack that are required for normal system operation. This section summarizes the purpose of most of these files.

Over 160 files have the .COM extension, and contain console command information used to automate most of the system start-up and diagnostic control procedures. These are all ASCII files that can be displayed using the SHOW FILE/ASCII command. The user can learn a lot about how the system works by examining these command files.

NOTE

It is possible, but not recommended, for the user to edit and modify these command files because any unauthorized changes may cause abnormal system operations. If system-specific command files are required, they should be created using filenames that are different from those currently on the pack.

Over 100 files have the .BPN extension and contain microcode information that is loaded into the EBox control RAMs during system operation. In general, there are four classes of microcode files.

1. The first class of files contains the information loaded into the CSM overlay region of the EBox control store when executing console commands. All 40 files in this group have the filename CSMnnn.BPN, where "nnn" is the number of the overlay. Refer to Appendix B for a list of each of these files. During console software initialization, the contents of these files are read from the RL02 and cached in T-11 RAM so the files can be accessed quickly when console commands are executed.

2. The second class of files contains system microcode information that is loaded into the CPU control RAMs during macro context initialization. Certain system microcode files are also used during diagnostic context initialization. Table D-2 lists the nine files in this group.

3. The third class of files contains special microcode information used by the microdiagnostics. The 54 files in this group have filenames that begin with EDK for shared VAX 8600/8650 files, and EEK for VAX 8650-specific files. In this group is the file DSM.BPN which contains the diagnostic support microcode loaded into the EBox control store during diagnostc context initialization.

4. The fourth class of files contains special diagnostic microcode used by EDKAA, the microhardcore diagnostic.

Table D-2 System Microcode Files

File Name	Unit	RAM
ACCESS.BPN	MBox	Access Violation RAM (ACV)
CTX.BPN	EBox	Context RAMs
CYCLE.BPN	MBox	Cycle Condition Code RAMs (CCC)
FADD.BPN	FBox	Control Store RAMs on the FBA module
FMUL.BPN	FBox	Control Store RAMs on the FBM module
IBOX.BPN	IBox	Control Store RAMs in the IBox
MCF.BPN	EBox	MCF RAMs in the EBox
UCODE.BPN	MBox	Control Store RAMs in the MBox
KA86AA.BPN	EBox	Control Store RAMs in the EBox and the IDRAM RAMs in the IBox

There are 28 files with the .HLP extension that contain ASCII text information used by the HELP command. These are read and displayed when the user requests information about how a command works.

Three files currently have the .DAT extension. These files contain system configuration information and snapshot dump information created when a KAF is detected. CONFIG.DAT contains a list of SDB signal filenames, that is used during console software initialization, to access and load the SDB signal name tables in T-11 RAM. The two files, SNAP1.DAT and SNAP2.DAT, contain system dump information and are created during a KAF. These snapshot files are copied from the RL02 to the system disk during system restart for subsequent analysis on-line.

There are six text files with the extension .MEM that contain user documentation. The user should display and read these memos for each new release of the RL02 disk pack to obtain up-to-date information about the diagnostic system. GUIDE.MEM, the largest of the files, contains diagnostic release notes for the microdiagnostics.

Nineteen files have the .CDF extension. These are ASCII files that contain all the SDB signal name information and are read and loaded into tables in the T-11 RAM during console software initialization. A unique file exists for each module in the system with SDB visibility hardware.

Last, but not least, are the two files called KA86.REV and NOTICE.TXT. KA86.REV is an ASCII text file that contains hardware and microcode revision information. NOTICE.TXT, also an ASCII file, contains console software information displayed during system start-up.

D.6 SUMMARY

The RL02 disk pack contains all the information necessary for normal system operation and diagnostic fault isolation. It is important for the user to understand the role of the files on the RL02 and how they affect system operation. Appendix J contains information that describes how to build and update the RL02 system disk pack.

APPENDIX E
SDB OVERVIEW

E.1 SDB OVERVIEW

This appendix reviews the Serial Diagnostic Bus (SDB) in the context of how it is used by the console software to snapshot the state of key hardware logic signals during system testing and troubleshooting. It begins with a summary of how the SDB visibility terminators work in conjunction with the console software to select, retrieve, and display the state of internal CPU logic signals. An example from the MBox is used to illustrate this operation. Next, there is a list of the Default SDB Registers permanently defined by the console software with a discussion of when and how they are displayed. This appendix concludes with procedures for defining additional SDB registers and signals that can be temporarily added to the default list during troubleshooting. Actual examples are used to illustrate the process.

E.2 SDB OPERATION

This section describes the operation of the SDB visibility channels. Basically, the SDB visibility channels provide the mechanism by which the console software displays the state of over 2000 internal CPU logic signals. Most all of the backplane signals between CPU modules are accessible via the SDB. The three major elements that comprise this mechanism are as follows.

- Signal name files on the RL02 disk drive
- Console software
- SDB logic on the console and CPU modules

Let's take a look at how these three elements tie together to permit displaying a CPU logic signal state on the console TTY. In order to display a signal state, the console software needs to know two things: the name of the signal and control information to access the signal via the SDB hardware. The signals visible on the SDB are segmented into channels; each module is assigned a unique channel number. The hardware design fixes the maximum number of channels at 24. Currently, only channels 00(x) through 14(x) are active. Channels 15(x), 16(x), and 17(x) are reserved and currently unused. The SDB channel numbers are assigned as follows.

00 = FBA	01 = FBM	02 = MCD
03 = IBD	04 = IDP	05 = ICA
06 = ICB	07 = CLK	08 = EDP
09 = EBE	0A = MCC	0B = MAP
0C = EBD	0D = EBC	0E = CSB
0F = CSA	10 = IOA0	11 = IOA1
12 = IOA2	13 = IOA3	14 = MTM
15 = Reserved	16 = Reserved	17 = Reserved

E.2.1 Signal Name Files

There are a set of ASCII files on the RL02 used by the console software to obtain the SDB signal name and control information. The first file, named CONFIG.DAT, contains a list of all the signal name files on the RL02. Figure E-1 shows an example of what this file looks like. Note that it contains a list of filenames, all with the .CDF filename extension which indicates an SDB signal name file. The .CDF filename is formatted as follows.

xxxyyy.CDF Where: xxx = module name MAP, MCC., etc.
yyy = module revision C04, E06, etc.

Example: MAPC04.CDF L0205 MBox MAP module

Figures E-2, E-3, and E-4 summarize the three file formats for the three MBox modules: MCD, MCC, and MAP. Each .CDF file is organized into three columns.

Column 1 SDB signal address information (in octal)
Column 2 SDB visibility ID
Column 3 Signal name

CDF860.DAT CDF865.DAT

```
>>>SHOW CONFIG.DAT/ASCII
! SDB Cad information files for M5 machines.
! Version: 003.000
! Released: December 21, 1985
CLKC03.CDF      !CLK C5
CSAB02.CDF      !CSA B2
CSBB02.CDF      !CSB B2
EBCC05.CDF      !EBC C5
EBDD02.CDF      !EBD D2
EBEB02.CDF      !EBE B2
EDPC02.CDF      !EDP C2
FBAB01.CDF      !FBA E7
FBMC01.CDF      !FBM C5
IBDF05.CDF      !IBD F5
ICAH02.CDF      !ICA H4
ICBF01.CDF      !ICB F5
IDPF02.CDF      !IDP F4
MAPD02.CDF      !MAP D2
MCCK01.CDF      !MCC K1
MCDD04.CDF      !MCD D4
MTMB01.CDF      !MTM B1
!VBAA01.CDF     !VBA A1 (SBI VISIBILITY MODULE #1)
!VBBA01.CDF     !VBB A1 (SBI VISIBILITY MODULE #2)
>>>
```

Figure E-1 CONFIG.DAT File Format

```
>>>SHOW MCDD04.CDF/ASCII
;CHASER Version 1(31)-1, 21 January 1984. Sources in LSCAD:<SDB>
/CADIF-VERSION/ 3(5)
/SUDS-SDB/ 1(2)
$SUDS-CHANNEL-TO-SIGNAL
2000    V$2191  MAP9 CACHE DAT PA 02 H
2001    V$2192  MAP9 CACHE DAT PA 03 H
2002    V$2193  MCC5 BUF WD CNT 2 H
2003    V$2224  REG BUS 3 H
2004    V$2194  MCC5 BUF WD CNT 3 H
2005    V$2225  REG BUS 2 H
2006    V$2226  REG BUS 4 H
2007    V$2227  REG BUS 5 H
2010    V$2215  MCCA M ECC CHECK H
2011    V$2216  MCCD U DS MUX SEL 1 H
2012    V$2169  MCC6 CLR ERR REGS H
2013    V$2174  MAPL CACHE 1 DAT H
2014    V$2171  MCCB M CB DR EN H
2015    V$2217  MCCA M SEL WD CNT H
2016    V$2173  MCCJ WRITE REG H
2017    V$2218  MCCA M DSM VALID H
2040    V$2123  MAPL C0 MUX SEL B H
2041    V$2124  MCCJ READ REG H
2042    V$2206  MCCD U DS MUX SEL 0 H
2043    V$2207  MCCJ CP WRITE H
2044    V$2125  -EBE WBUS OPAR B1 H
^C
?DCN-I-CCABRT, ^C abort
>>>
```

Figure E-2 MCDD04.CDF File

```
>>>SHOW MCCK01.CDF/ASCII
;CHASER Version 1(31)-1, 21 January 1984. Sources in LSCAD:<SDB>
/CADIF-VERSION/ 3(5)
/SUDS-SDB/ 1(2)
$SUDS-CHANNEL-TO-SIGNAL
12000   V$A234  MCC6 OP CURR CTX 2 H
12001   V$A235  MCC8 REFL IN PROG H
12002   V$A236  -MCCM BYTWR CACH PERR H
12003   V$A237  MCC5 CP CYC ACTIVE H
12004   V$A189  ABUS MSKED CMD H
12005   V$A188  ABUS WR CMD H
12006   V$A238  -MCCK CLK3 T1A A H
12007   V$A239  -MCCK CLK3 T1D C H
12010   V$A272  MCC6 DIS IOA REQ H
12011   V$A286  MCC5 ABUS XFR IN PROG H
12012   V$A229  -MCCC U CPR IO AC EN H
12013   V$A287  MCCB WAITING ON ARY H
12014   V$A288  MCC1 UADR B 1 H
^C
?DCN-I-CCABRT, ^C abort
>>>
```

Figure E-3 MCCK01.CDF File

```
>>>SHOW MAPD04.CDF/ASCII
?DCN-W-FILNAM, MAPD04.CDF file not found
>>>SHOW MAPD02.CDF/ASCII
;CHASER Version 1(31)-1, 21 January 1984. Sources in LSCAD:<SDB>
/CADIF-VERSION/ 3(5)
/SUDS-SDB/ 1(2)
$SUDS-CHANNEL=TO=SIGNAL
13000    V$B197  ABUS DATA ADDRS 26 H
13001    V$B198  ABUS DATA ADDRS 25 H
13002    V$B199  ABUS DATA ADDRS 27 H
13003    V$B200  ABUS DATA ADDRS 23 H
13004    V$B201  EDP EVA A25 H
13005    V$B202  IVA BUS 29 H
13006    V$B203  ABUS DATA ADDRS 30 H
13007    V$B204  ABUS DATA ADDRS 21 H
13010    V$B233  -MCCB M ABUS LAT LD H
13011    V$B151  -MCC6 PHYS REF H
13012    V$B227  -MCC7 PA LAT LD H
13013    V$B142  -MCC7 WRITE LRU H
13014    V$B156  -MCC7 WRITE CACHE EN A H
13015    V$B154  MCCB M PA MUX SEL 1 H
13016    V$B155  MCCB M PA MUX SEL 0 H
^C
?DCN-I-CCABRT, ^C abort
>>>
```

Figure E-4 MAPD02.CDF File

During system start-up and console software initialization, CONFIG.DAT is read first to determine which signal name files to load. The console software then loads the information contained in all of the .CDF files into signal name tables in T-11 memory. Once the tables are loaded, the HEX command set can be used to examine and display the state of any SDB signal using the information stored in the T-11 tables.

Let's suppose the user wanted to know the state of the signal MCCM HLD ERR DAT REG, which is terminated on the MCD module. Refer to Figure E-5. The user would type a command something like the following.

```
>>>> EXAMINE/SDB V$2152
```

The console software would use the V$2152 argument to access the table entry with the required information. It would use the octal ID information, 02117, as follows.

1. Selects Channel 2, which is the SDB channel assignment for the MCD module in the MBox.

2. Enables a visibility multiplexer on MCDV by asserting MCDV VIS MUX ENA 1.

3. Selects input D7 (pin 15) on the enabled multiplexer, which samples the state of MCCM HLD ERR DAT REG H.

4. Specifies that the selected signal will be returned in bit position 2 MCDV VIS BUS DATA 02 H.

Finally, it is important to understand the relationship between this file and console software information and the SDB hardware logic in the console and the MBox.

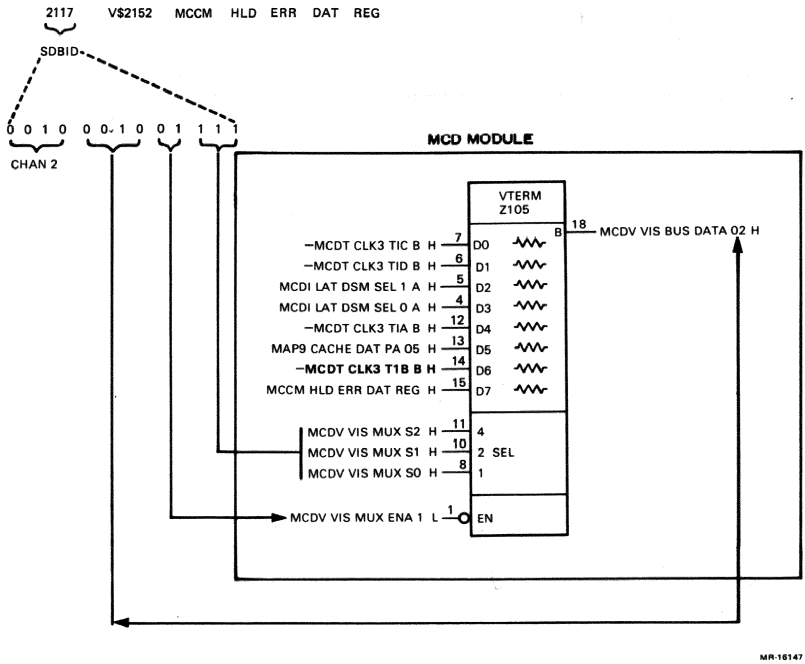


Figure E-5 VTERM Selection

E.2.2 Visibility Channel Operation

Refer to Figure E-6. Each module assigned an SDB visibility channel contains a set of visibility multiplexers (VTERMs) that are used to both terminate the SDB signals and provide the means of selecting any signal to be retrieved. The number of VTERM circuits varies from channel to channel depending upon the total number of logic test points made visible. Each VTERM can handle up to eight signals.

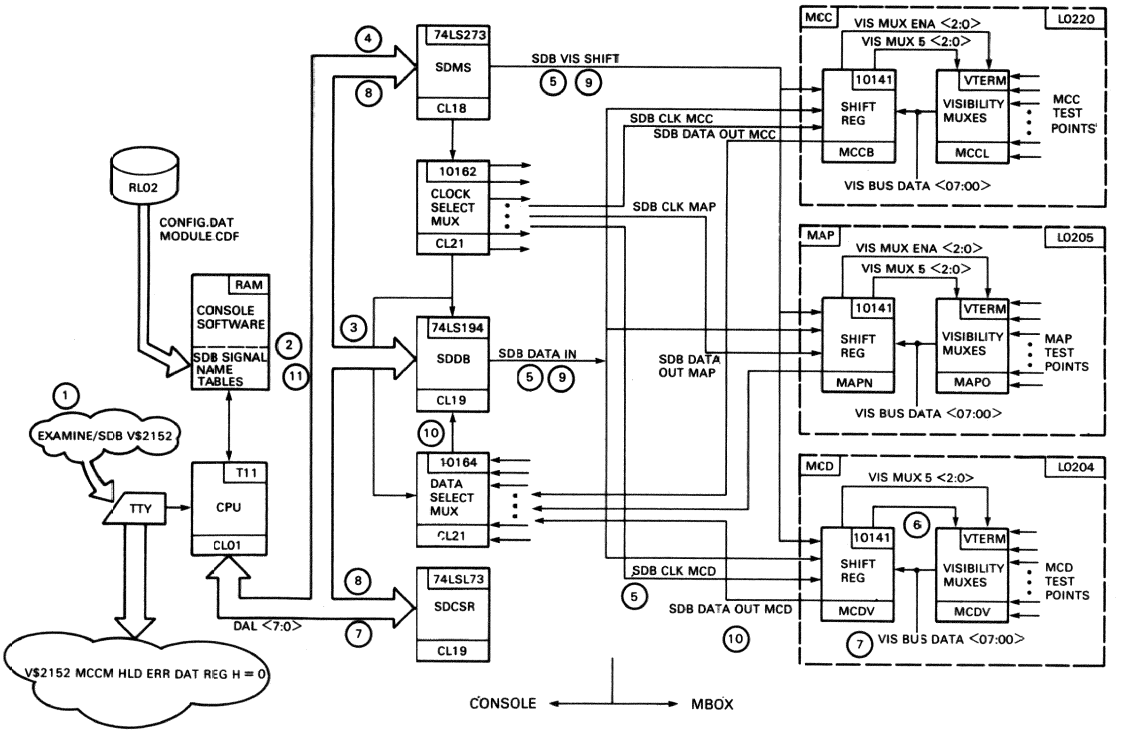


Figure E-6 SDB Visibility Channel Block Diagram

Along with the VTERM multiplexers, each module (channel) contains two 10141 shift register devices that provide an 8-bit, serial shift/parallel load interface between the CPU module and the console module. The following steps describe the process of retrieving signal states from the MCD module.

1. The user types the EXAMINE/SDB V$2152 command to display the state of MCCM HLD ERR DAT REG.

2. The console software parses the command and uses the V$2152 argument to access the SDB table entry in T-11 RAM.

3. The console software loads the SDDB register with the multiplexer enable/select information (01001111).

4. The console software then loads the SDMS register with the channel select information and sets the shift bit.

5. The console hardware then asserts SDB VIS SHIFT, followed by eight SDB CLK MCD signals, in order to shift the contents of the SDDB register out via SDB DATA IN. From SBD DATA IN, the SDDB register contents are shifted into the 10141 shift register on MCDV.

6. The data shifted into the 10141 asserts the required VIS MUX ENA<2:0> and VIS MUX S<2:0> signals on MCDV to enable and select the VTERM input.

7. The console software loads the SDCSR register with a command to parallel load the output from the selected VTERMs into the 10141 shift register devices on MCDV.

8. The console software then loads the SDMS and SDCSR registers with CHANNEL SELECT, SHIFT, and READ command information to retrieve the data.

9. The console hardware then generates SDB VIS SHIFT and eight SDB CLK MCD clocks in order to shift the signal information out of the 10141 shift register and back to the console via SDB DATA OUT MCD.

10. The information on SDB DATA OUT MCD is shifted into the SDDB register via the 10164 data select multiplexer.

11. The console software reads the SDDB into T-11 RAM and decodes the state of the requested bit, <2>.

12. Finally, the console software displays the signal name retrieved from the SDB tables, column 3, with the appropriate state indication (=1 or =0 based on the actual signal state found in step [11]).

Note that even though the command specified examining a single signal, eight bits are still selected and shifted back via SDB DATA OUT MCD. Any SDB visibility signal read involves first, shifting eight bits of control information from the console, and then shifting eight bits of SDB signal information back to the console.

E.3 DEFAULT VISIBILITY REGISTERS

The HEX debugger command set provides a number of predefined visibility registers useful for tracing the machine state. Most of these registers are part of the default trace list used by the TMICRO, TSTATE, and REPORT commands. The TRACE ADD command can be used to add a register to the trace list that is not already there. The TRACE DEFINE command can be used to define a new visibility register. The following list summarizes the various visibility registers present in the console program.

*ABUS	*ARADR	*ARBUS	*DBUS	*EBFLSH
*EDPPE	EFORK	EMCF	ESTALL	EUPC
*EVABUS	*FABUS	FAUPC	FINFO	FMUPC
*IBDBUF	*IBUF	IBXERR	IDIAG	*INCR
*IOPSEL	IUPC	*IVABUS	*MDBUSI	*MDBUSM
*MEMREQ	MISC1	MISC2	MISC3	MUPC
*NATRAM	OPAR	*OPBUS	*OPCODE	*OPMCF
OPPORT	PAACK	*PAMD	*PAMM	PARITY
PSL	RANDM1	RANDM2	*REGBUS	SBIBUS
SBITR	*STALL	UPCSAV	*WBUS	

*These registers are included in the default trace list; the contents of these registers are displayed by the REPORT, TMIC, and TSTATE commands. Any other register must be explicitly added to the trace list with the TRACE ADD command.

The above information is not complete without a breakdown of what each register contains. Tables E-1 through E-41 provide a list of all visibility registers and the signal names that compose them. Note that bits are listed in order of the most significant bit first. The special filler symbol, "XXXX," is used to pad a visibility register in places where signals are no longer defined or where nibble alignment is desired. If the accuracy of a visibility register definition is in question, the actual definition can be found in the source listing for the HEXBOX program module (HEXBOXT11.LST).

E.4 DEFINING ADDITIONAL VISIBILITY REGISTERS

In addition to the default visibility registers defined in the previous section, the user can define and display additional registers. This section describes using the TRACE DEFINE and TRACE ADD commands in the HEX command set to define and display these personalized registers and signals. It also includes how to use the TRACE DELETE and TRACE REMOVE commands. Figure E-7 shows the default registers and SDB signals included in the trace list. Any other registers and/or signals to be displayed must be defined and added to the trace list.

```
>>>DEB
>>>>SHOW TRACE
Registers traced:
    8020  OPMCF              801F  OPCODE             801E  OPBUS
    801D  EVABUS             801C  EDPPE              801B  EBFLSH
    801A  WBUS               8019  REGBUS             8018  PAMM
    8017  PAMD               8016  MEMREQ             8014  ARBUS
    8013  ARADR              8012  ABUS               802C  STALL
    802B  NATRAM             802A  MDBUSM             8029  MDBUSI
    8028  IOPSEL             8027  INCR               8025  IBUF
    8024  IBDBUF             8023  IVABUS             8022  DBUS
    8021  FABUS
Signals traced:
    18A5  V$C216 -MCC MD RESP A H       0CE8  V$6208 ICA LID ISTALL H
    1880  V$C169 MCC DEST CODE 0 H
A_breaks set: none
O_breaks set: none
>>>>
>>>>EXI
>>>
```

Figure E-7 Default Trace List

E.4.1 TRACE DEFINE

This command allows the user to define a problem-specific register containing a set of signals to be monitored while tracing. The first step for the user is to determine which signals to trace and their associated visibility ID numbers (V$xxxx). Proceed as follows.

1. Determine the modules where the signals originate.

2. Type SHOW CONFIG.DAT/ASCII to display a list of the module ".CDF" filenames that contain the SDB signal names and IDs.

3. Type SHOW xxxxxx.CDF/ASCII to display the signal names in each file and note the "V$xxxx" ID for each signal in the list.

At this point, the user should have a list of "V$xxxx" IDs, one for each signal to be traced. Arrange these in the order they're to be displayed, from Most Significant Bit (MSB) to Least Significant Bit (LSB). Everything's now ready to do the TRACE DEFINE command. Pick a short, meaningful name for the new register (for example, MYREG1).

Now, type the following.

```
DC>>TRACE DEFINE MYREG1
```

The program will prompt the user for a list of SDB IDs. Respond by typing the "xxxx" numbers from the list. Each number should be separated by a space and typed in descending order from MSB to LSB. After typing the LSB, simply press <RETURN> twice to signal the end. The example shown in Figure E-8 illustrates the process.

Refer to Chapter 7 for examples of how the following commands are used.

```
DC>>TRACE DEFINE PAMUX
    Enter V$ terms separated with <sp> or <,>
    B157 B154 B155

    B9A8  PAMUX  3.
DC>>SHOW DEFINE PAMUX
    B9A8  PAMUX  3.
    B157 B154 B155
DC>>
```

Figure E-8 TRACE DEFINE Example

E.4.2 TRACE ADD

Once the user's defined the "tailor-made" register, it is automatically added to the trace list. Thus, the user doesn't need to use the TRACE ADD command initially.

If either the TRACE RESTORE or TRACE REMOVE command were used to remove the register from the list, this command may be used to add it back. Individual SDB signal names can be added to the trace list, as well as registers.

E.4.3 TRACE DELETE

Any register you define can be deleted at any time using the TRACE DELETE command. It cannot be used to delete any one of the default registers described earlier. Once deleted, the register will have to be redefined if you need it later. To prevent this aggravation, the TRACE REMOVE command described next should be used to remove items from the trace list while still retaining their definition.

E.4.4 TRACE REMOVE

This command permits you to remove items from the trace list and then add them back later using TRACE ADD. You simply type "TRACE REMOVE regname" to remove any item from the trace list. Unlike the TRACE DELETE command, this command allows you to remove default registers and signals, also. Like TRACE ADD, individual SDB signal names may be removed from the trace list using this command.

E.4.5 TRACE RESTORE

This command simply restores the trace list to its default state in one, swift step. If you want to get back those registers that were added, you will have to repeat the TRACE ADD command.

E.4.6 SHOW REGISTER

This command will display a complete list, including name, ID, and size, of all the currently defined registers.

E.4.7 SHOW TRACE

This command will display a list of all registers and SDB signals currently in the trace list. It also includes any current breakpoints that may be set.

E.5 SUMMARY

The system hardware contains a sophisticated Serial Diagnostic Bus (SDB) that provides visibility to thousands of hardware logic signal states within the CPU. This hardware is enhanced by a comprehensive set of user commands in the HEX command set that allow the user to tailor the response to individual needs when troubleshooting problems.

Table E-1 ABus Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
ABUS	V$A223	43	11	SB ABUS IOA REQUEST 3 H
	V$A225	42		SB ABUS IOA REQUEST 2 H
	V$A215	41		SB ABUS IOA REQUEST 1 H
	V$A217	40		SB ABUS IOA REQUEST 0 H
	V$A130	39	10	ABUS CPU BUF DONE H
	V$A226	38		ABUS CPU BUF ERROR H
	V$A220	37		ABUS LEN STAT 1 H
	V$A222	36		ABUS LEN STAT 0 H
	V$A211	35	9	ABUS CMD MASK 3 H
	V$A214	34		ABUS CMD MASK 2 H
	V$A219	33		ABUS CMD MASK 1 H
	V$A210	32		ABUS CMD MASK 0 H
	V$B195	31	8	ABUS DATA ADDRS 31 H
	V$B203	30		ABUS DATA ADDRS 30 H
	V$B139	29		ABUS DATA ADDRS 29 H
	V$B128	28		ABUS DATA ADDRS 28 H
	V$B199	27	7	ABUS DATA ADDRS 27 H
	V$B197	26		ABUS DATA ADDRS 26 H
	V$B198	25		ABUS DATA ADDRS 25 H
	V$B183	24		ABUS DATA ADDRS 24 H
	V$B200	23	6	ABUS DATA ADDRS 23 H
	V$B130	22		ABUS DATA ADDRS 22 H
	V$B204	21		ABUS DATA ADDRS 21 H
	V$B131	20		ABUS DATA ADDRS 20 H
	V$B189	19	5	ABUS DATA ADDRS 19 H
	V$B135	18		ABUS DATA ADDRS 18 H
	V$B138	17		ABUS DATA ADDRS 17 H
	V$B136	16		ABUS DATA ADDRS 16 H
	V$B140	15	4	ABUS DATA ADDRS 15 H
	V$B190	14		ABUS DATA ADDRS 14 H
	V$B141	13		ABUS DATA ADDRS 13 H
	V$B191	12		ABUS DATA ADDRS 12 H
	V$B127	11	3	ABUS DATA ADDRS 11 H
	V$B196	10		ABUS DATA ADDRS 10 H
	V$B208	09		ABUS DATA ADDRS 09 H
	V$B193	08		ABUS DATA ADDRS 08 H

Table E-1 ABus Register Format (Cont.)

Name	V$ Symbol	Bit	Nibble	Signal Name
ABUS	V$B194	07	2	ABUS DATA ADDRS 07 H
	V$B169	06		ABUS DATA ADDRS 06 H
	V$B209	05		ABUS DATA ADDRS 05 H
	V$B206	04		ABUS DATA ADDRS 04 H
	V$B205	03	1	ABUS DATA ADDRS 03 H
	V$B168	02		ABUS DATA ADDRS 02 H
	V$B176	01		ABUS DATA ADDRS 01 H
	V$B177	00		ABUS DATA ADDRS 00 H

Table E-2 ARADR Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
ARADR	V$K103	25	7	MAP1 ARRAY ADR 28 H
	V$K105	24		MAP1 ARRAY ADR 27 H
	V$K101	23	6	MAP1 ARRAY ADR 26 H
	V$K179	22		MAP1 ARRAY ADR 25 H
	V$K174	21		MAP1 ARRAY ADR 24 H
	V$K177	20		MAP1 ARRAY ADR 23 H
	V$K173	19	5	MAP1 ARRAY ADR 22 H
	V$K171	18		MAP1 ARRAY ADR 21 H
	V$K159	17		MAP1 ARRAY ADR 20 H
	V$K162	16		MAP1 ARRAY ADR 19 H
	V$K161	15	4	MAP1 ARRAY ADR 18 H
	V$K157	14		MAP1 ARRAY ADR 17 H
	V$K156	13		MAP2 ARRAY ADR 16 H
	V$K155	12		MAP2 ARRAY ADR 15 H
	V$K151	11	3	MAP2 ARRAY ADR 14 H
	V$K154	10		MAP2 ARRAY ADR 13 H
	V$K148	09		MAP2 ARRAY ADR 12 H
	V$K147	08		MAP2 ARRAY ADR 11 H
	V$K143	07	2	MAP2 ARRAY ADR 10 H
	V$K146	06		MAP2 ARRAY ADR 09 H
	V$K142	05		MAP2 ARRAY ADR 08 H
	V$K145	04		MAP2 ARRAY ADR 07 H
	V$K141	03	1	MAP2 ARRAY ADR 06 H
	V$K140	02		MAP2 ARRAY ADR 05 H
	V$K139	01		MAP2 ARRAY ADR 04 H
	V$K135	00		MAP2 ARRAY ADR 03 H

Table E-3 ARBus Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
ARBUS	V$K113	31	8	ARRAY BUS D31 H
	V$K112	30		ARRAY BUS D30 H
	V$K106	29		ARRAY BUS D29 H
	V$K107	28		ARRAY BUS D28 H
	V$K108	27	7	ARRAY BUS D27 H
	V$K102	26		ARRAY BUS D26 H
	V$K175	25		ARRAY BUS D25 H
	V$K178	24		ARRAY BUS D24 H
	V$K104	23	6	ARRAY BUS D23 H
	V$K100	22		ARRAY BUS D22 H
	V$K176	21		ARRAY BUS D21 H
	V$K172	20		ARRAY BUS D20 H
	V$K167	19	5	ARRAY BUS D19 H
	V$K169	18		ARRAY BUS D18 H
	V$K166	17		ARRAY BUS D17 H
	V$K170	16		ARRAY BUS D16 H
	V$K131	15	4	ARRAY BUS D15 H
	V$K127	14		ARRAY BUS D14 H
	V$K129	13		ARRAY BUS D13 H
	V$K130	12		ARRAY BUS D12 H
	V$K126	11	3	ARRAY BUS D11 H
	V$K125	10		ARRAY BUS D10 H
	V$K128	09		ARRAY BUS D09 H
	V$K119	08		ARRAY BUS D08 H
	V$K123	07	2	ARRAY BUS D07 H
	V$K124	06		ARRAY BUS D06 H
	V$K122	05		ARRAY BUS D05 H
	V$K118	04		ARRAY BUS D04 H
	V$K121	03	1	ARRAY BUS D03 H
	V$K117	02		ARRAY BUS D02 H
	V$K120	01		ARRAY BUS D01 H
	V$K116	00		ARRAY BUS D00 H

Table E-4 DBus Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
DBUS	V$3271	31	8	-DBUS D31 H
	V$3262	30		-DBUS D30 H
	V$3272	29		-DBUS D29 H
	V$3266	28		-DBUS D28 H
	V$3170	27	7	-DBUS D27 H
	V$3225	26		-DBUS D26 H
	V$3241	25		-DBUS D25 H
	V$3240	24		-DBUS D24 H
	V$3270	23	6	-DBUS D23 H
	V$3263	22		-DBUS D22 H
	V$3273	21		-DBUS D21 H
	V$3275	20		-DBUS D20 H
	V$3171	19	5	-DBUS D19 H
	V$3187	18		-DBUS D18 H
	V$3237	17		-DBUS D17 H
	V$3236	16		-DBUS D16 H
	V$3269	15	4	-DBUS D15 H
	V$3268	14		-DBUS D14 H
	V$3274	13		-DBUS D13 H
	V$3267	12		-DBUS D12 H
	V$3220	11	3	-DBUS D11 H
	V$3222	10		-DBUS D10 H
	V$3239	09		-DBUS D09 H
	V$3242	08		-DBUS D08 H
	V$3176	07	2	-DBUS D07 H
	V$3178	06		-DBUS D06 H
	V$3105	05		-DBUS D05 H
	V$3104	04		-DBUS D04 H
	V$3145	03	1	-DBUS D03 H
	V$3144	02		-DBUS D02 H
	V$3243	01		-DBUS D01 H
	V$3126	00		-DBUS D00 H

Table E-5 EBFLSH Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
EBFLSH	V$3198	04	2	-ICA BUF FLUSH MRES3 H
	V$5230	03	1	-ICAB EFLSH FR CPC LAT H
	V$5218	02		-CSB UMCF 2 A H
	V$5242	01		ICA6 IFORK CTL 2 H
	V$5194	00		-CSB UMCF 0 A H

Table E-6 EDPPE Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
EDPPE	V$9104	43	11	EDP EDPE D3 H
	V$D157	42		EDP EDPE D2 H
	V$9102	41		EDP EDPE D1 H
	V$D156	40		EDP EDPE D0 H
	V$E138	39	10	EDP STATE 7 H
	V$E109	38		EDP STATE 6 H
	V$E177	37		EDP STATE 5 H
	V$E120	36		EDP STATE 4 H
	V$E184	35	9	EDP STATE 3 H
	V$E136	34		EDP STATE 2 H
	V$E110	33		EDP STATE 1 H
	V$E178	32		EDP STATE 0 H
	V$A180	31	8	ICA ISTALL A H
	V$9103	30		EBD RSV MODE H
	V$9100	29		ICB RLOG PE H
	V$9181	28		ICB IBUF PE H
	V$5123	27	7	ICA7 ICS PAR ERR H
	V$9101	26		ICB IDRAM PE H
	V$9176	25		IDP IAMUX PE H
	V$9182	24		IDP IBMUX PE H
	V$XXXX	23	6	Filler
	V$6127	22		MCC OP PA ACK B H
	V$5167	21		MCC IBF PA ACK H
	V$9162	20		MCC MBOX CS PE H
	V$5221	19	5	EBD ESTALL TO ICA H
	V$5186	18		EBE IBOX ERR LTH A H
	V$9170	17		EBD ECS PE FLAG H
	V$9110	16		EBD ECS PE LST CYC H

Table E-6 EDPPE Register Format (Cont.)

Name	V$ Symbol	Bit	Nibble	Signal Name
EDPPE	V$D166	15	4	EBD EBOX ERR LST CYC H
	V$4167	14		EBD EBOX ERR TO IDP H
	V$9170	13		EBD ECS PE FLAG H
	V$9110	12		EBD ECS PE LST CYC H
	V$D158	11	3	EBD EDP PE FLAG A H
	V$9169	10		EBD EDP PE FLAG H
	V$9173	09		EBD EMCR PE FLAG H
	V$9161	08		EBD EN ETRAP H
	V$5221	19	5	EBD ESTALL TO ICA H
	V$5186	18		EBE IBOX ERR LTH A H
	V$9170	17		EBD ECS PE FLAG H
	V$9110	16		EBD ECS PE LST CYC H
	V$D166	15	4	EBD EBOX ERR LST CYC H
	V$4167	14		EBD EBOX ERR TO IDP H
	V$9170	13		EBD ECS PE FLAG H
	V$9110	12		EBD ECS PE LST CYC H
	V$D158	11	3	EBD EDP PE FLAG A H
	V$9169	10		EBD EDP PE FLAG H
	V$9173	09		EBD EMCR PE FLAG H
	V$9161	08		EBD EN ETRAP H
	V$9174	07	2	EBD USTK PE FLAG H
	V$9143	06		EBD WBUS PE FLAG H
	V$C159	05		-EBC MCF RAM PAR ERR H
	V$C161	04		EBD6 EVC D09 H
	V$4139	03	1	-ICA BMUX CHK WITH CTX H
	V$0131	02		EBE WBUS OPAR B2 C H
	V$0129	01		EBE WBUS OPAR B1 C H
	V$4281	00		-DBUS D08 H

Table E-7 EFORK Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
EFORK	V$6146	08	3	-CSB EBOX FORK E H
	V$0100	07	2	-CSB EBOX FORK D H
	V$8124	06		-CSB EBOX FORK C H
	V$5159	05		-CSB EBOX FORK B H
	V$C226	04		-CSB EBOX FORK A H
	V$XXXX	03	1	Filler
	V$C102	02		-CSB EBOX IRD A H
	V$6101	01		-CSB EBOX IRD B H
	V$1231	00		-CSB EBOX IRD C H

Table E-8 EMCF Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
EMCF	V$D184	37	10	CSA UMCF 5 A H
	V$D186	36		CSA UMCF 4 A H
	V$D181	35	9	CSA UMCF 3 A H
	V$D183	34		CSB UMCF 2 A H
	V$D180	33		CSB UMCF 1 A H
	V$D182	32		CSB UMCF 0 A H
	V$XXXX	31	8	Filler
	V$XXXX	30		Filler
	V$A153	29		EBC EBOX MCF 5 H
	V$A182	28		EBC EBOX MCF 4 H
	V$A174	27	7	EBC EBOX MCF 3 H
	V$A165	26		EBC EBOX MCF 2 H
	V$A164	25		EBC EBOX MCF 1 H
	V$A154	24		EBC EBOX MCF 0 H
	V$XXXX	23	6	Filler
	V$D154	22		-EBC9 MCF-LOAD N H
	V$D153	21		-EBC9 MCF-LOAD T H
	V$D155	20		-EBC9 MCF-REQUE N H
	V$D151	19	5	-EBC9 MCF-REQUE T H
	V$D146	18		EBCA MCF-EN MBOX H
	V$D140	17		EBCA MCF-WCHK H
	V$D150	16		EBC8 MCF-E DISP I H
	V$C179	15	4	-EBC MCF-ACK REQ LTH H
	V$C187	14		-EBC MCF-CLR EBPS LTH H
	V$C184	13		-EBC MCF-CLR IBPS LTH H
	V$C186	12		-EBC MCF-CLR OPPS LTH H
	V$C180	11	3	-EBC MCF-EN OWSTL LTH H
	V$C178	10		-EBC MCF-MEM REQ LTH H
	V$C127	09		-EBC MCF-MEM WRT LTH H
	V$C199	08		-EBC MCF-OP WRT LTH H
	V$D143	07	2	-EBCA MCF-ACK REQ H
	V$D138	06		-EBC8 MCF-CLR EBPS H
	V$D149	05		-EBC8 MCF-CLR IBPS H
	V$D137	04		-EBC8 MCF-CLR OPPS H
	V$D134	03	1	-EBC7 MCF-EN OWSTL H
	V$D139	02		-EBC7 MCF-MEM REQ H
	V$D104	01		-EBC7 MCF-MEM WRT H
	V$D105	00		-EBC7 MCF-OP WRT H

Table E-9 ESTALL Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
ESTALL	V$E183	11	3	-EBD ESTALL TO CSB H
	V$D171	10		EBD ESTALL TO EBC H
	V$9166	09		EBD ESTALL TO EBE H
	V$8160	08		EBD ESTALL TO EDP H
	V$0190	07	2	EBD ESTALL TO FBA H
	V$1123	06		EBD ESTALL TO FBM H
	V$3157	05		-EBD ESTALL TO IBD H
	V$5221	04		EBD ESTALL TO ICA H
	V$6138	03	1	EBD ESTALL TO ICB H
	V$4126	02		EBD ESTALL TO IDP H
	V$A184	01		EBD ESTALL TO MCC H
	V$2114	00		EBD ESTALL TO MCD H

Table E-10 EUPC Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
EUPC	V$F122	12	4	CSB UPC 12 H
	V$F123	11	3	CSB UPC 11 H
	V$F111	10		CSB UPC 10 H
	V$F113	09		CSB UPC 09 H
	V$F107	08		CSB UPC 08 H
	V$F112	07	2	CSB UPC 07 H
	V$F115	06		CSB UPC 06 H
	V$F109	05		CSB UPC 05 H
	V$F110	04		CSB UPC 04 H
	V$F114	03	1	CSB UPC 03 H
	V$F126	02		CSB UPC 02 A H
	V$F118	01		CSB UPC 01 A H
	V$F127	00		CSB UPC 00 A H

Table E-11 EVABUS Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
EVABUS	V$B222	31	8	EDP EVA A31 H
	V$B112	30		EDP EVA A30 H
	V$B187	29		EDP EVA A29 H
	V$B182	28		EDP EVA A28 H
	V$B188	27	7	EDP EVA A27 H
	V$B132	26		EDP EVA A26 H
	V$B201	25		EDP EVA A25 H
	V$B119	24		EDP EVA A24 H
	V$B118	23	6	EDP EVA A23 H
	V$B133	22		EDP EVA A22 H
	V$B134	21		EDP EVA A21 H
	V$B123	20		EDP EVA A20 H
	V$B126	19	5	EDP EVA A19 H
	V$B192	18		EDP EVA A18 H
	V$B122	17		EDP EVA A17 H
	V$B207	16		EDP EVA A16 H
	V$B215	15	4	EDP EVA A15 H
	V$B143	14		EDP EVA A14 H
	V$B146	13		EDP EVA A13 H
	V$B145	12		EDP EVA A12 H
	V$B144	11	3	EDP EVA A11 H
	V$B216	10		EDP EVA A10 H
	V$B213	09		EDP EVA A09 H
	V$B175	08		EDP EVA A08 H
	V$B171	07	2	EDP EVA A07 H
	V$B179	06		EDP EVA A06 H
	V$B173	05		EDP EVA A05 H
	V$B174	04		EDP EVA A04 H
	V$B167	03	1	EDP EVA A03 H
	V$B170	02		EDP EVA A02 H
	V$B159	01		EDP EVA A01 A H
	V$B162	00		EDP EVA A00 A H

Table E-12 FABUS Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
FABUS	V$1197	31	8	-FBA FA BUS D31 H
	V$1269	30		-FBA FA BUS D30 H
	V$1138	29		-FBA FA BUS D29 H
	V$1180	28		-FBA FA BUS D28 H
	V$1171	27	7	-FBA FA BUS D27 H
	V$1167	26		-FBA FA BUS D26 H
	V$1250	25		-FBA FA BUS D25 H
	V$1170	24		-FBA FA BUS D24 H
	V$1198	23	6	-FBA FA BUS D23 H
	V$1260	22		-FBA FA BUS D22 H
	V$1139	21		-FBA FA BUS D21 H
	V$1165	20		-FBA FA BUS D20 H
	V$1261	19	5	-FBA FA BUS D19 H
	V$1248	18		-FBA FA BUS D18 H
	V$1137	17		-FBA FA BUS D17 H
	V$1169	16		-FBA FA BUS D16 H
	V$1199	15	4	-FBA FA BUS D15 H
	V$1262	14		-FBA FA BUS D14 H
	V$1134	13		-FBA FA BUS D13 H
	V$1183	12		-FBA FA BUS D12 H
	V$1263	11	3	-FBA FA BUS D11 H
	V$1244	10		-FBA FA BUS D10 H
	V$1249	09		-FBA FA BUS D09 H
	V$1164	08		-FBA FA BUS D08 H
	V$1200	07	2	-FBA FA BUS D07 H
	V$1265	06		-FBA FA BUS D06 H
	V$1133	05		-FBA FA BUS D05 H
	V$1196	04		-FBA FA BUS D04 H
	V$1264	03	1	-FBA FA BUS D03 H
	V$1245	02		-FBA FA BUS D02 H
	V$1251	01		-FBA FA BUS D01 H
	V$1168	00		-FBA FA BUS D00 H

Table E-13 FAUPC Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
FAUPC	V$0243	08	3	FA11 UPCA A8 H
	V$0321	07	2	FA11 UPCA A7 H
	V$0320	06		FA11 UPCA A6 H
	V$0319	05		FA11 UPCA A5 H
	V$0325	04		FA11 UPCA A4 H
	V$0324	03	1	FA11 UPCA A3 H
	V$0322	02		FA11 UPCA A2 H
	V$0242	01		FA11 UPCA A1 H
	V$0335	00		FA11 UPCA A0 H

Table E-14 FMUPC Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
FMUPC	V$1152	08	3	FM11 UPC A8 H
	V$1149	07	2	FM11 UPC A7 H
	V$1179	06		FM11 UPC A6 H
	V$1178	05		FM11 UPC A5 H
	V$1155	04		FM11 UPC A4 H
	V$1154	03	1	FM11 UPC A3 H
	V$1148	02		FM11 UPC A2 H
	V$1153	01		FM11 UPC A1 H
	V$1150	00		FM11 UPC A0 H

Table E-15 IBDBUF Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
IBDBUF	V$4289	42	11	IBD DRAM CTX 2 H
	V$4288	41		IBD DRAM CTX 1 H
	V$4287	40		IBD DRAM CTX 0 H
	V$5117	39	10	IBD DRAM TYPE 1 H
	V$5211	38		IBD DRAM TYPE 0 H
	V$5116	37		IBD DRAM REF 1 H
	V$5214	36		IBD DRAM REF 0 H
	V$5192	35	9	IBD BDEST NEXT H
	V$5163	34		IBD DRAM LAST H
	V$5171	33		IBD DRAM SUSP H
	V$5236	32		IBD DISABLE SB HIT H
	V$5245	31	8	IBD BUF FD INST H
	V$5249	30		IBD EXT OPC H
	V$6129	29		IBD BUF IBUF PE H
	V$6128	28		IBD BUF DRAM PE H
	V$5270	27	7	IBD IFORK VALID H
	V$5175	26		IBD DMUX VALID H
	V$5172	25		IBD EXC ONLY H
	V$5213	24		IBD EPC 0 OR 7 H
	V$5111	23	6	IBD ISEL B2 H
	V$3193	22		ICA LD IFORK DISP H
	V$5101	21		IBD BUF LD CTL 1 H
	V$5103	20		IBD BUF LD CTL 0 H
	V$4286	19	5	-IBD OPTIMIZED H
	V$4192	18		IBD BUF DELTA PC 2 H
	V$4187	17		IBD BUF DELTA PC 1 H
	V$4193	16		IBD BUF DELTA PC 0 H
	V$6132	15	4	IBD IBGPR 3 H
	V$6179	14		IBD IBGPR 2 H
	V$6135	13		IBD IBGPR 1 H
	V$6133	12		IBD IBGPR 0 H
	V$4285	11	3	IBD IGPR 3 H
	V$4284	10		IBD IGPR 2 H
	V$4283	09		IBD IGPR 1 H
	V$4282	08		IBD IGPR 0 H

Table E-15 IBDBUF Register Format (Cont.)

Name	V$ Symbol	Bit	Nibble	Signal Name
IBDBUF	V$5161	07	2	IBD BUF FA 7 H
	V$5136	06		IBD BUF FA 6 H
	V$5253	05		IBD BUF FA 5 H
	V$5158	04		IBD BUF FA 4 H
	V$5179	03	1	IBD BUF FA 3 H
	V$6123	02		IBD BUF FA 2 H
	V$6119	01		IBD BUF FA 1 H
	V$6121	00		IBD BUF FA 0 H

Table E-16 IBUF Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
IBUF	V$5114	15	4	IBD IBUF DATA B1 7 H
	V$5208	14		IBD IBUF DATA B1 6 H
	V$5209	13		IBD IBUF DATA B1 5 H
	V$5210	12		IBD IBUF DATA B1 4 H
	V$5237	11	3	IBD IBUF DATA B1 3 H
	V$5233	10		IBD IBUF DATA B1 2 H
	V$5246	09		IBD IBUF DATA B1 1 H
	V$5243	08		IBD IBUF DATA B1 0 H
	V$5104	07	2	IBD IBUF DATA B0 7 H
	V$5106	06		IBD IBUF DATA B0 6 H
	V$5108	05		IBD IBUF DATA B0 5 H
	V$5105	04		IBD IBUF DATA B0 4 H
	V$5250	03	1	IBD IBUF DATA B0 3 H
	V$5248	02		IBD IBUF DATA B0 2 H
	V$5247	01		IBD IBUF DATA B0 1 H
	V$5252	00		IBD IBUF DATA B0 0 H

Table E-17 IBXERR Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
IBXERR	V$C197	04	2	-EBE IBOX ERR LTH E H
	V$8163	03	1	EBE IBOX ERR LTH D H
	V$4166	02		EBE IBOX ERR LTH C H
	V$6204	01		EBE IBOX ERR LTH B H
	V$5186	00		EBE IBOX ERR LTH A H

Table E-18 IDIAG Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
IDIAG	V$D102	05	2	ICA IBOX DIAG DONE H
	V$5244	04		EBC E DISP I DLY H
	V$D150	03	1	EBC8 MCF-E DISP I H
	V$5196	02		-ICB ID FULL STALL H
	V$6116	01		ICA PC ISTALL H
	V$4130	00		IDP5 ISTALL A H

Table E-19 INCR Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
INCR	V$4175	07	2	-IBD INCR VIBA BY 4 H
	V$3131	06		IBD9 SHIFT COUNT 2 B H
	V$3150	05		IBDD LAT CTX 1 H
	V$3253	04		IBD9 SHIFT COUNT 0 B H
	V$6171	03	1	IBD UNLAT CUR VAL 3 H
	V$6170	02		IBD UNLAT CUR VAL 2 H
	V$6174	01		IBD UNLAT CUR VAL 1 H
	V$6206	00		IBD UNLAT CUR VAL 0 H

Table E-20 IOPSEL Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
IOPSEL	V$3163	05	2	CSB UOPSEL 1 B H
	V$3206	04		CSB UOPSEL 0 B H
	V$8146	03	1	CSA UMISC 3 H
	V$8168	02		CSA UMISC 2 H
	V$8143	01		CSA UMISC 1 H
	V$8150	00		CSA UMISC 0 H

Table E-21 IUPC Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
IUPC	V$5120	07	2	ICA8 LD OR SAV 7 H
	V$5119	06		ICA8 LD OR SAV 6 H
	V$5157	05		ICA8 LD OR SAV 5 H
	V$5121	04		ICA8 LD OR SAV 4 H
	V$5155	03	1	ICA8 LD OR SAV 3 H
	V$5154	02		ICA8 LD OR SAV 2 H
	V$5150	01		ICA8 LD OR SAV 1 H
	V$5156	00		ICA8 LD OR SAV 0 H

Table E-22 IVABUS Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
IVABUS	V$B160	31	8	IVA BUS 31 H
	V$B223	30		IVA BUS 30 H
	V$B202	29		IVA BUS 29 H
	V$B185	28		IVA BUS 28 H
	V$B117	27	7	IVA BUS 27 H
	V$B186	26		IVA BUS 26 H
	V$B184	25		IVA BUS 25 H
	V$B115	24		IVA BUS 24 H
	V$B114	23	6	IVA BUS 23 H
	V$B129	22		IVA BUS 22 H
	V$B116	21		IVA BUS 21 H
	V$B125	20		IVA BUS 20 H
	V$B121	19	5	IVA BUS 19 H
	V$B120	18		IVA BUS 18 H
	V$B124	17		IVA BUS 17 H
	V$B217	16		IVA BUS 16 H
	V$B214	15	4	IVA BUS 15 H
	V$B147	14		IVA BUS 14 H
	V$B149	13		IVA BUS 13 H
	V$B148	12		IVA BUS 12 H
	V$B218	11	3	IVA BUS 11 H
	V$B219	10		IVA BUS 10 H
	V$B220	09		IVA BUS 09 H
	V$B211	08		IVA BUS 08 H
	V$B172	07	2	IVA BUS 07 H
	V$B210	06		IVA BUS 06 H
	V$B180	05		IVA BUS 05 H
	V$B166	04		IVA BUS 04 H
	V$B178	03	1	IVA BUS 03 H
	V$B181	02		IVA BUS 02 H
	V$B161	01		IVA BUS 01 H
	V$B158	00		IVA BUS 00 H

Table E-23 MDBUSI Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
MDBUSI	V$3281	31	8	MD BUS D31 H
	V$3232	30		MD BUS D30 H
	V$3244	29		MD BUS D29 H
	V$3251	28		MD BUS D28 H
	V$3252	27	7	MD BUS D27 H
	V$3282	26		MD BUS D26 H
	V$3116	25		MD BUS D25 H
	V$3165	24		MD BUS D24 H
	V$3280	23	6	MD BUS D23 H
	V$3234	22		MD BUS D22 H
	V$3229	21		MD BUS D21 H
	V$3235	20		MD BUS D20 H
	V$3120	19	5	MD BUS D19 H
	V$3279	18		MD BUS D18 H
	V$3118	17		MD BUS D17 H
	V$3283	16		MD BUS D16 H
	V$3277	15	4	MD BUS D15 H
	V$3230	14		MD BUS D14 H
	V$3248	13		MD BUS D13 H
	V$3233	12		MD BUS D12 H
	V$3121	11	3	MD BUS D11 H
	V$3256	10		MD BUS D10 H
	V$3119	09		MD BUS D09 H
	V$3168	08		MD BUS D08 H
	V$3276	07	2	MD BUS D07 H
	V$3228	06		MD BUS D06 H
	V$3249	05		MD BUS D05 H
	V$3250	04		MD BUS D04 H
	V$3254	03	1	MD BUS D03 H
	V$3257	02		MD BUS D02 H
	V$3135	01		MD BUS D01 H
	V$3164	00		MD BUS D00 H

Table E-24 MDBUSM Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
MDBUSM	V$2130	31	8	MD BUS D31 H
	V$2131	30		MD BUS D30 H
	V$2129	29		MD BUS D29 H
	V$2132	28		MD BUS D28 H
	V$2140	27	7	MD BUS D27 H
	V$2139	26		MD BUS D26 H
	V$2135	25		MD BUS D25 H
	V$2136	24		MD BUS D24 H
	V$2144	23	6	MD BUS D23 H
	V$2134	22		MD BUS D22 H
	V$2155	21		MD BUS D21 H
	V$2161	20		MD BUS D20 H
	V$2154	19	5	MD BUS D19 H
	V$2153	18		MD BUS D18 H
	V$2162	17		MD BUS D17 H
	V$2160	16		MD BUS D16 H
	V$2166	15	4	MD BUS D15 H
	V$2156	14		MD BUS D14 H
	V$2167	13		MD BUS D13 H
	V$2157	12		MD BUS D12 H
	V$2158	11	3	MD BUS D11 H
	V$2101	10		MD BUS D10 H
	V$2195	09		MD BUS D09 H
	V$2100	08		MD BUS D08 H
	V$2103	07	2	MD BUS D07 H
	V$2102	06		MD BUS D06 H
	V$2106	05		MD BUS D05 H
	V$2198	04		MD BUS D04 H
	V$2107	03	1	MD BUS D03 H
	V$2199	02		MD BUS D02 H
	V$2200	01		MD BUS D01 H
	V$2105	00		MD BUS D00 H

Table E-25 MEMREQ Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
MEMREQ	V$C171	25	7	MCC EBOX PA ACK A H
	V$C172	24		MCC OP PA ACK A H
	V$5167	23	6	MCC IBF PA ACK H
	V$C215	22		EBC EBD MAST RST DLY H
	V$C173	21		MCC DEST CODE 1 H
	V$C174	20		-EBD9 MEM REQ LST CYC H
	V$C193	19	5	MCC PORT STAT CODE 3 H
	V$C191	18		MCC PORT STAT CODE 2 H
	V$C190	17		MCC PORT STAT CODE 1 H
	V$C192	16		MCC PORT STAT CODE 0 H
	V$A180	15	4	ICA ISTALL A H
	V$A167	14		ICA IBF REQUEST H
	V$C231	13		MCC STAT CODE OUT 1 H
	V$C226	12		-CSB EBOX FORK A H
	V$A168	11	3	ICB OP MCF 3 H
	V$A179	10		ICB OP MCF 2 H
	V$A190	09		ICB OP MCF 1 H
	V$A186	08		ICB OP MCF 0 H
	V$A221	07	2	ICA OP ABORT A H
	V$A184	06		EBD ESTALL TO MCC H
	V$A153	05		EBC EBOX MCF 5 H
	V$A182	04		EBC EBOX MCF 4 H
	V$A174	03	1	EBC EBOX MCF 3 H
	V$A165	02		EBC EBOX MCF 2 H
	V$A164	01		EBC EBOX MCF 1 H
	V$A154	00		EBC EBOX MCF 0 H

Table E-26 MUPC Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
MUPC	V$A192	07	2	MCC1 UADR B 7 H
	V$A231	06		MCC1 UADR B 6 H
	V$A273	05		MCC1 UADR B 5 H
	V$A274	04		MCC1 UADR B 4 H
	V$A109	03	1	MCC1 UADR A 3 H
	V$A110	02		MCC1 UADR A 2 H
	V$A288	01		MCC1 UADR A 1 H
	V$A279	00		MCC1 UADR A 0 H

Table E-27 NATRAM Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
NATRAM	V$3213	19	5	IBDC NAT CTX 2 H
	V$3217	18		IBDC NAT CTX 1 H
	V$3209	17		IBDC NAT CTX 0 H
	V$3112	16		IBDC NAT TYPE 1 H
	V$3183	15	4	IBDC NAT TYPE 0 H
	V$3167	14		IBDC NAT REF 1 H
	V$3182	13		IBDC NAT REF 0 H
	V$3210	12		IBDC NAT CTL 1 H
	V$3177	11	3	IBDC NAT CTL 0 H
	V$3208	10		IBDC NAT SUSPEND H
	V$3109	09		IBDC NAT BDEST NXT H
	V$3211	08		IBDC NAT LAST H
	V$3184	07	2	IBDB NAT OPAR H
	V$3101	06		IBDB NAT FPA H
	V$3180	05		IBDB NAT ADRS 05 H
	V$3103	04		IBDB NAT ADRS 04 H
	V$3278	03	1	IBDB NAT ADRS 03 H
	V$3102	02		IBDB NAT ADRS 02 H
	V$3108	01		IBDB NAT ADRS 01 H
	V$3100	00		IBDB NAT ADRS 00 H

Table E-28 OPAR Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
OPAR	V$4278	37	10	ICA FORCE AMUX OPAR H
	V$4102	36		ICA FORCE BMUX OPAR H
	V$C144	35	9	EDP OPR PAR ERR H
	V$3184	34		IBDB NAT OPAR H
	V$3221	33		IBDD LAT OPAR H
	V$5170	32		ICA5 ICS OPAR H
	V$XXXX	31	8	Filler
	V$4206	30		-ICA EN OP OPAR VAL H
	V$8184	29		-IDP OP OPAR VALID H
	V$8191	28		IDP OP LWD OPAR H
	V$8154	27	7	EBE WBUS OPAR B3 A H
	V$8155	26		EBE WBUS OPAR B2 A H
	V$8153	25		EBE WBUS OPAR B1 A H
	V$8156	24		EBE WBUS OPAR B0 A H
	V$0186	23	6	EBE WBUS OPAR B3 C H
	V$0131	22		EBE WBUS OPAR B2 C H
	V$0129	21		EBE WBUS OPAR B1 C H
	V$0180	20		EBE WBUS OPAR B0 C H
	V$4205	19	5	EBE WBUS OPAR B3 B H
	V$4123	18		EBE WBUS OPAR B2 B H
	V$4210	17		EBE WBUS OPAR B1 B H
	V$4209	16		EBE WBUS OPAR B0 B H
	V$2126	15	4	-EBE WBUS OPAR B3 H
	V$2127	14		-EBE WBUS OPAR B2 H
	V$2125	13		-EBE WBUS OPAR B1 H
	V$2113	12		-EBE WBUS OPAR B0 H
	V$3153	11	3	MCD MD BUS OPAR B3 H
	V$3149	10		MCD MD BUS OPAR B2 H
	V$3152	09		MCD MD BUS OPAR B1 H
	V$3148	08		MCD MD BUS OPAR B0 H
	V$XXXX	07	2	formally MCD MD BUS OPAR B3 H
	V$XXXX	06		formally MCD MD BUS OPAR B2 H
	V$XXXX	05		formally MCD MD BUS OPAR B1 H
	V$XXXX	04		formally MCD MD BUS OPAR B0 H
	V$4101	03	1	IBD DBUS OPAR B3 H
	V$4104	02		IBD DBUS OPAR B2 H
	V$4140	01		IBD DBUS OPAR B1 H
	V$4100	00		IBD DBUS OPAR B0 H

Table E-29 OPBUS Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
OPBUS	V$0178	31	8	OP BUS D31 H
	V$0174	30		OP BUS D30 H
	V$0148	29		OP BUS D29 H
	V$0106	28		OP BUS D28 H
	V$0203	27	7	OP BUS D27 H
	V$0260	26		OP BUS D26 H
	V$0153	25		OP BUS D25 H
	V$0232	24		OP BUS D24 H
	V$0202	23	6	OP BUS D23 H
	V$0261	22		OP BUS D22 H
	V$0149	21		OP BUS D21 H
	V$0228	20		OP BUS D20 H
	V$0285	19	5	OP BUS D19 H
	V$0172	18		OP BUS D18 H
	V$0154	17		OP BUS D17 H
	V$0229	16		OP BUS D16 H
	V$0295	15	4	OP BUS D15 H
	V$0198	14		OP BUS D14 H
	V$0344	13		OP BUS D13 H
	V$0289	12		OP BUS D12 H
	V$0301	11	3	OP BUS D11 H
	V$0300	10		OP BUS D10 H
	V$0293	09		OP BUS D09 H
	V$0271	08		OP BUS D08 H
	V$0286	07	2	OP BUS D07 H
	V$0274	06		OP BUS D06 H
	V$0103	05		OP BUS D05 H
	V$0275	04		OP BUS D04 H
	V$0179	03	1	OP BUS D03 H
	V$0264	02		OP BUS D02 H
	V$0152	01		OP BUS D01 H
	V$0233	00		OP BUS D00 H

Table E-30 OPCODE Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
OPCODE	V$1101	07	2	ICA OPC BIT 7 A H
	V$1103	06		ICA OPC BIT 6 A H
	V$1238	05		ICA OPC BIT 5 A H
	V$1125	04		ICA OPC BIT 4 A H
	V$1240	03	1	ICA OPC BIT 3 A H
	V$1237	02		ICA OPC BIT 2 A H
	V$1141	01		ICA OPC BIT 1 A H
	V$1241	00		ICA OPC BIT 0 A H

Table E-31 OPMCF Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
OPMCF	V$A168	03	1	ICB OP MCF 3 H
	V$A179	02		ICB OP MCF 2 H
	V$A190	01		ICB OP MCF 1 H
	V$A186	00		ICB OP MCF 0 H

Table E-32 OPPORT Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
OPPORT	V$6186	63	16	ICA OPV CTL 0 H
	V$6190	62		ICA OPV CTL 1 H
	V$5149	61		-ICAG SET OPV FB H
	V$5141	60	15	-ICB SET OPV H
	V$6158	59		-ICB9 IMD OPV LAT H
	V$6154	58		-ICB9 PEND IMD OPV H
	V$XXXX	57		Filler
	V$XXXX	56	14	Filler
	V$C172	55		MCC OP PA ACK A H
	V$6127	54		MCC OP PA ACK B H
	V$XXXX	53		Filler
	V$XXXX	52	13	Filler
	V$0191	51		EBD OPBUS VALID H
	V$1206	50		-FBA OPBU VAL TO FBM H
	V$XXXX	49		Filler
	V$C212	48	12	-ICB ALWAYS OP VALID H
	V$C211	47		-ICB ALMOST OP VALID H
	V$6155	46		-ICB8 ALMOST SET OPV H
	V$XXXX	45		Filler
	V$A221	44	11	ICA OP ABORT A H
	V$C183	43		ICA OP ABORT B LAT H
	V$1120	42		ICA IBF OR OP FLUSH H
	V$A264	41		ICA MBOX FLUSH A H
	V$C188	40	10	-ICA OP FLUSH B H
	V$4206	39		-ICA EN OP OPAR VAL H
	V$8184	38		-IDP OP OPAR VALID H
	V$XXXX	37		Filler
	V$XXXX	36	9	Filler
	V$C201	35		ICB KILL OP WRT H
	V$E102	34		MCC NEXT OP WRT H
	V$E186	32		MCC TRAP OP WCHK H
	V$E135	31	8	MCC TRAP OP WRT H
	V$C176	30		-EBD9 EN OP WRT STALL H
	V$5254	29		-ICAG OP REQ CIP LAT H
	V$5148	28		ICB SET OP WRT CIP H

Table E-32 OPPORT Register Format (Cont.)

Name	V$ Symbol	Bit	Nibble	Signal Name
OPPORT	V$6109	27	7	-ICBA OP WRT CIP H
	V$C213	26		-EBD9 OP WRT IN PROG H
	V$C148	25		-EBD9 OP WRT LST CYC H
	V$6134	24		-ICA OP WRT LAT H
	V$C170	23	6	-ICA EN OP WRT ACK H
	V$6165	22		-ICA ENA OP MD RESP H
	V$XXXX	21		Filler
	V$4147	20		-ICA IVA SEL OP VA H
	V$4176	19	5	-ICA ENA OP VA LD H
	V$5131	18		IDP OP VA BIT 00 H
	V$5241	17		IDP OP VA BIT 01 H
	V$A186	16		ICB OP MCF 0 H
	V$A190	15	4	ICB OP MCF 1 H
	V$A179	14		ICB OP MCF 2 H
	V$A168	13		ICB OP MCF 3 H
	V$XXXX	12		Filler
	V$A185	11	3	ICB OP MEM CTX 0 H
	V$A170	10		ICB OP MEM CTX 1 H
	V$A183	09		ICB OP MEM CTX 2 H
	V$3143	08		ICB OP MEM REQ H
	V$4148	07	2	ICB OP MEM REQ A H
	V$XXXX	06		Filler
	V$6182	05		-ICA SET FA OPMEM REQ H
	V$C117	04		ICB FA OP MEM REQ H
	V$XXXX	03	1	Filler
	V$4172	02		IDP5 IVA SEL OP VA A H
	V$4221	01		IDP5 IVA SEL OP VA B H
	V$4108	00		IDP5 IVA SEL OP VA C H

Table E-33 PAACK Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
PAACK	V$D167	06	2	-MCC EBOX PA ACK B H
	V$C171	05		MCC EBOX PA ACK A H
	V$6142	04		EBD EB PA ACK LTH H
	V$XXXX	03	1	Filler
	V$5167	02		MCC IBF PA ACK H
	V$C172	01		MCC OP PA ACK A H
	V$6127	00		MCC OP PA ACK B H

Table E-34 PAMD Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
PAMD	V$D167	23	6	-MCC EBOX PA ACK B H
	V$A184	22		EBD ESTALL TO MCC H
	V$C215	21		EBC EBD MAST RST DLY H
	V$A167	20		ICA IBF REQUEST H
	V$A264	19	5	ICA MBOX FLUSH A H
	V$A180	18		ICA ISTALL A H
	V$A221	17		ICA OP ABORT A H
	V$A277	16		ICA MBOX FLUSH B H
	V$A168	15	4	ICB OP MCF 3 H
	V$A179	14		ICB OP MCF 2 H
	V$A190	13		ICB OP MCF 1 H
	V$A186	12		ICB OP MCF 0 H
	V$C193	11	3	MCC PORT STAT CODE 3 H
	V$C191	10		MCC PORT STAT CODE 2 H
	V$C190	09		MCC PORT STAT CODE 1 H
	V$C192	08		MCC PORT STAT CODE 0 H
	V$C172	07	2	MCC OP PA ACK A H
	V$5167	06		MCC IBF PA ACK H
	V$A153	05		EBC EBOX MCF 5 H
	V$A182	04		EBC EBOX MCF 4 H
	V$A174	03	1	EBC EBOX MCF 3 H
	V$A165	02		EBC EBOX MCF 2 H
	V$A164	01		EBC EBOX MCF 1 H
	V$A154	00		EBC EBOX MCF 0 H

Table E-35 PAMM Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
PAMM	V$A212	04	2	MAP9 PAMM CONF A H
	V$A173	03	1	MAP9 PAMM CONF 8 H
	V$A172	02		MAP9 PAMM CONF 4 H
	V$A224	01		MAP9 PAMM CONF 2 H
	V$A216	00		MAP9 PAMM CONF 1 H

Table E-36 PARITY Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
PARITY	V$9170	65	17	EBD ECS PE FLAG H
	V$D158	64		EBD EDP PE FLAG A H
	V$9169	63	16	EBD EDP PE FLAG H
	V$9173	62		EBD EMCR PE FLAG H
	V$9174	61		EBD USTK PE FLAG H
	V$9143	60		EBD WBUS PE FLAG H
	V$XXXX	59	15	Filler
	V$XXXX	58		Filler
	V$XXXX	57		Filler
	V$9110	56		EBD ECS PE LST CYC H
	V$C219	55	14	-CSB ECS PAR ERR H
	V$E164	54		CSA PAR ERR H
	V$F104	53		CSAS CSA PAR H
	V$F124	52		-CSB CS PAR OK A H
	V$C142	51	13	CSB USTK PAR ERR H
	V$E140	50		-CSBR FLIP USTK PAR H
	V$E151	49		CSBS DATA PAR H
	V$C144	48		EDP OPR PAR ERR H
	V$C160	47	12	-EDP RESULT PAR ERR H
	V$8134	46		DISA BYTE 10 PAR H
	V$8165	45		EDPI DISA BYTE 32 PAR H
	V$8126	44		EDPI FLIP WREG PAR H
	V$9150	43	11	EBC FLIP WBUS PAR B0 H
	V$9149	42		EBC FLIP WBUS PAR B1 H
	V$9145	41		EBC FLIP WBUS PAR B2 H
	V$9144	40		EBC FLIP WBUS PAR B3 H

Table E-36 PARITY Register Format (Cont.)

Name	V$ Symbol	Bit	Nibble	Signal Name
PARITY	V$XXXX	39	10	Filler
	V$C159	38		-EBC MCF RAM PAR ERR H
	V$D142	37		EBCA MCF PAR H
	V$D133	36		EBCH FLIP MCF RAM PAR H
	V$XXXX	35	9	Filler
	V$0240	34		FA17 GPR PPAR 00 H
	V$0241	33		FA17 GPR PPAR 01 H
	V$0122	32		FA17 GPR PPAR 02 H
	V$0121	31	8	FA19 UWD PARITY H
	V$0169	30		FBM CS PAR ERROR H
	V$0165	29		FBM FDRAM PAR ERROR H
	V$1275	28		FM16 UWD PARITY H
	V$XXXX	27	7	Filler
	V$6128	26		IBD BUF DRAM PE H
	V$6129	25		IBD BUF IBUF PE H
	V$4201	24		ICA FORCE GPR PE H
	V$6115	23	6	ICA FORCE RLOG PE H
	V$9179	22		ICA ICS PE H
	V$9181	21		ICB IBUF PE H
	V$9101	20		ICB IDRAM PE H
	V$9100	19	5	ICB RLOG PE H
	V$9176	18		IDP IAMUX PE H
	V$9182	17		IDP IBMUX PE H
	V$XXXX	16		Filler
	V$XXXX	15	4	Filler
	V$XXXX	14		Filler
	V$2165	13		MAP2 <29:4> PAR H
	V$XXXX	12		formally MCC1 MMS PAR ERR H
	V$XXXX	11	3	formally MCC2 ACCESS PAR H
	V$A118	10		MCCC U CPR PAR A H
	V$A126	09		MCCC U CPR PAR B H
	V$2109	08		MCCM INV CACH BYT PAR H
	V$A201	07	2	MCD3 ABUS DAT PERR H
	V$A251	06		-MCDU CACH DAT PERR H
	V$A250	05		-MCDU WR DAT PERR H
	V$A200	04		MAP2 ABUS ADR PERR H
	V$A204	03	1	MAPL TAG PERR H
	V$A202	02		MAPL TAG W PERR H
	V$A148	01		-MAPR TB PERR H
	V$9162	00		MCC MBOX CS PE H

Table E-37 PSL Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
PSL	V$E117	20	6	EBE PSL CM TO CSB H
	V$C113	19	5	-EBE PSL TP H
	V$E112	18		EBD UTRAP VECTOR 4 H
	V$E152	17		EBE PSL IS TO CSB H
	V$A144	16		EBE CURMOD 1 TO MCC H
	V$A162	15	4	EBE CURMOD 0 TO MCC H
	V$XXXX	14		Filler PREVMODE
	V$XXXX	13		Filler PREVMODE
	V$D125	12		EBE PSL IPL 4 H
	V$D127	11	3	EBE PSL IPL 3 H
	V$D126	10		EBE PSL IPL 2 H
	V$D131	09		EBE PSL IPL 1 H
	V$D130	08		EBE PSL IPL 0 H
	V$XXXX	07	2	Filler DECIMAL OVERFLOW
	V$XXXX	06		Filler FLOATING UNDERFLOW
	V$C109	05		-EBE PSL IV TO EBD H
	V$XXXX	04		Filler TRACE FAULT PENDING
	V$9171	03	1	EDP PSL N BIT A H
	V$9172	02		EDP PSL Z BIT A H
	V$9168	01		EDP PSL V BIT A H
	V$9167	00		EDP PSL C BIT A H

Table E-38 REGBUS Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
REGBUS	V$2205	07	2	REG BUS 7 H
	V$2203	06		REG BUS 6 H
	V$2227	05		REG BUS 5 H
	V$2226	04		REG BUS 4 H
	V$2224	03	1	REG BUS 3 H
	V$2225	02		REG BUS 2 H
	V$2204	01		REG BUS 1 H
	V$2223	00		REG BUS 0 H

Table E-39 STALL Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
STALL	V$6202	28	8	ICA IFORK NOP H
	V$3189	27	7	ICA IFORK CYCLE H
	V$6185	26		ICA CTX UNALIGNED H
	V$5133	25		ICA6 UTRAP CTL 1 H
	V$5132	24		ICA6 UTRAP CTL 0 H
	V$A180	23	6	ICA ISTALL A H
	V$9107	22		ICA ISTALL B H
	V$C205	21		ICA ISTALL BUF A H
	V$6137	20		ICA ISTALL C H
	V$3197	19	5	ICA ISTALL D H
	V$4156	18		ICA ISTALL E H
	V$5196	17		-ICB ID FULL STALL H
	V$3190	16		ICA PC ISTALL A H
	V$9103	15	4	EBD RSV MODE H
	V$9100	14		ICB RLOG PE H
	V$5123	13		ICA7 ICS PAR ERR H
	V$9181	12		ICB IBUF PE H
	V$9101	11	3	ICB IDRAM PE H
	V$9176	10		IDP IAMUX PE H.
	V$9182	09		IDP IBMUX PE H
	V$XXXX	08		Filler
	V$C172	07	2	MCC OP PA ACK A H
	V$6127	06		MCC OP PA ACK B H
	V$5167	05		MCC IBF PA ACK H
	V$9162	04		MCC MBOX CS PE H
	V$5221	03	1	EBD ESTALL TO ICA H
	V$5186	02		EBE IBOX ERR LTH A H
	V$9170	01		EBD ECS PE FLAG H
	V$9110	00		EBD ECS PE LST CYC H

Table E-40 UPCSAV Register Format

Name	V$ Symbol	Bit	Nibble	Signal Name
UPCSAV	V$E173	12	4	CSBQ UPCSAVE 12 H
	V$E172	11	3	CSBQ UPCSAVE 11 H
	V$E143	10		CSBQ UPCSAVE 10 H
	V$E157	09		CSBQ UPCSAVE 09 H
	V$E156	08		CSBQ UPCSAVE 08 H
	V$E150	07	2	CSBQ UPCSAVE 07 H
	V$E171	06		CSBQ UPCSAVE 06 H
	V$E174	05		CSBQ UPCSAVE 05 H
	V$E170	04		CSBQ UPCSAVE 04 H
	V$E149	03	1	CSBQ UPCSAVE 03 H
	V$E141	02		CSBP UPCSAVE 02 H
	V$E142	01		CSBP UPCSAVE 01 H
	V$E148	00		CSBP UPCSAVE 00 H

Table E-41 WBus Register

Name	V$ Symbol	Bit	Nibble	Signal Name
WBUS	V$4232	31	8	WBUS D31 H
	V$4231	30		WBUS D30 H
	V$4227	29		WBUS D29 H
	V$4223	28		WBUS D28 H
	V$4107	27	7	WBUS D27 H
	V$4257	26		WBUS D26 H
	V$4271	25		WBUS D25 H
	V$4256	24		WBUS D24 H
	V$4251	23	6	WBUS D23 H
	V$4252	22		WBUS D22 H
	V$4266	21		WBUS D21 H
	V$4253	20		WBUS D20 H
	V$4157	19	5	WBUS D19 H
	V$4131	18		WBUS D18 H
	V$4127	17		WBUS D17 H
	V$4128	16		WBUS D16 H
	V$4247	15	4	WBUS D15 H
	V$4243	14		WBUS D14 H
	V$4244	13		WBUS D13 H
	V$4245	12		WBUS D12 H
	V$4122	11	3	WBUS D11 H
	V$4274	10		WBUS D10 H
	V$4276	09		WBUS D09 H
	V$4202	08		WBUS D08 H
	V$4184	07	2	WBUS D07 H
	V$4182	06		WBUS D06 H
	V$4158	05		WBUS D05 H
	V$4177	04		WBUS D04 H
	V$4194	03	1	WBUS D03 H
	V$4191	02		WBUS D02 H
	V$4178	01		WBUS D01 H
	V$4188	00		WBUS D00 H

APPENDIX F
DIAGNOSTIC NAMING CONVENTIONS

F.1 MICRODIAGNOSTICS

Each microdiagnostic name uses the five-letter code EXXXX, where each letter indicates the following.

a. First letter letter: E = VAX family

b. Second letter: D = 8600-specific and shared 8600/8500, or E = 8650

c. Third letter: K = CPU

d. Fourth letter: identifies the specific diagnostic

e. Fifth letter: identifies the file type

Figure F-1 illustrates how this convention is used to name one of the EBox diagnostic files. Table F-1 summarizes the meaning of the fifth letter in the file name.

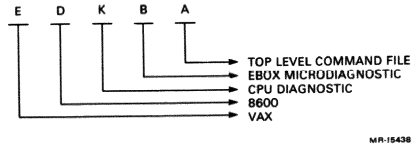


Figure F-1 Diagnostic Naming Convention

Table F-1 summarizes the meaning of the fifth letter in the filename.

Table F-1 Diagnostic Naming Convention (Fifth Letter)

Fifth Letter	File Type	Extension List	Description
A	.COM	.COM	Top-level command file for loading and initializing a microdiagnostic for execution
B	.BPN	.MCR	FBox adder microcode
C	.COM	.COM	Command file to deposit test patterns in cache
D	.BPN	.MCR	IDRAM microcode
E	.COM	.COM	Command file to deposit test patterns in the EBox scratchpad RAMs
F	.BPN	.MCR	FBox multiplier microcode
G	.BPN	.MCR	FDRAM microcode
H	.BPN	.MCR	MBox access RAM information
I	.BPN	.MCR	IBox microcode
M	.BPN	.MCR	MBox microcode
N	.BPN	.MCR	MBox cycle condition code RAM information
U	.BPN	.MCR	EBox microcode
X	.BPN	.MCR	EBox context RAM information
Y	.BPN	.MCR	EBox MCF RAM information

The following example describes one of the IBox microdiagnostics to illustrate how these file-naming conventions work.

- EDKRA.COM -- This top-level command file is invoked to load the microcode and initialize the CPU for running EDKRA, which is also the official name for this diagnostic.

- EDKRU.BPN -- This file contains the main diagnostic test microcode loaded into the EBox control store.

- EDKRU.MCR -- This is the listing for the EBox microcode in the microfiche library.

- EDKRI.BPN -- This file contains the diagnostic microcode loaded into the IBox control store.

- EDKRI.MCR -- This is the listing for the IBox diagnostic microcode.

- EDKRE.COM -- This command file is invoked from within EDKRA.COM and contains a set of DEPOSIT/ESC commands used to load test data into the EBox scratchpad RAMs.

- EDKRC.COM -- This command file is invoked from within EDKRA.COM and contains a set of DEPOSIT/CACHE commands used to load test instructions and data into cache.

F.2 MICROHARDCORE TESTS

The microhardcore tests contained within EDKAA are described in three different types of listings with either a .DOC, .MCR, or .LST extension. EDKAA.DOC is a single document file that contains operating instructions and program descriptions, including a discussion of error reporting. Table F-2 lists the names of all the microcode listings used by MHC, while Table F-3 lists all MACRO-11 program module listings.

Normally, the user should be able to use the error information displayed by the program to resolve most hardware faults and should seldom need to refer to the program listings. If it is necessary to go to the listings for additional information, the user may need to reference the listings. This involves the following procedure.

1. Refer to EDKAA.DOC for general test information.

2. Refer to the appropriate .LST listing shown in Table F-3 based on which test failed.

3. Refer to the appropriate .MCR listing shown in Table F-2 if the failing test uses test microcode loaded into the CPU control stores.

F.3 MACRODIAGNOSTICS

For additional information on the naming conventions and organization of the VAX family diagnostics, refer to the EVNDX document in the microfiche library. This document provides a complete index of all the VAX diagnostics, not only those specific to the VAX 8600 and VAX 8650 systems. Like previous members of the VAX family, certain macrodiagnostics are system-specific and use a unique letter in the diagnostic name to indicate the system type. The letter chosen for the VAX 8600 was D. Note that in the VAX 8600-specific diagnostics shown in Table F-4, D is the second letter in the diagnostic name.

Table F-2 MHC Microcode Listings

Name	Description
DKABAA.MCR	FBA control store file for load 1
DKABBA.MCR	FBA control store file for load 2
DKABCB.MCR	FBA control store file for load 3
DKAFAA.MCR	FBM control store file for load 1
DKAFBA.MCR	FBM control store file for load 2
DKAIAA.MCR	IBox control store file for load 1
DKAMAA.MCR	MBox control store file for load 1
DKAMBA.MCR	MBox control store file for load 2
DKAMCA.MCR	MBox control store file for load 3
DKAMDA.MCR	MBox control store file for load 4
DKAUAA.MCR	EBox control store file for load 1
DKAUBA.MCR	EBox control store file for load 2

Table F-3 MHC MACRO-11 Program Listings

Name	Description
MHCCLK.LST	Clock Logic Tests
MHCEBA.LST	EBox MCF RAM, CTX RAM, and logic tests
MHCEBB.LST	EBox microcode tests
MHCEBC.LST	EBox SDB and control store tests
MHCFBX.LST	FBox logic, RAM, and microcode tests
MHCIBX.LST	IBox logic, RAM, and microcode tests
MHCLMT.LST	Logical multi-box tests
MHCMBA.LST	MBox logic and RAM tests
MHCMBB.LST	MBox microcode tests
MHCMHC.LST	Root control section and handlers

Table F-4 VAX 8600-Specific Macrodiagnostics

Name	Description
EDSAA	VAX Diagnostic Supervisor
EDKAB	VAX Basic Instruction Exerciser
EDCLA	VAX DW780, CI780, DR780, RH780 SBI Exerciser
EDKAX	VAX CPU Kernel Exerciser

APPENDIX G
DIAGNOSTIC LISTINGS

G.1 OVERVIEW

This appendix provides an overview of the various types of program listings that the user must deal with when performing corrective maintenance on VAX 8600 and VAX 8650 systems. In general, there are three major types of listings supplied with the system microfiche library.

1. Listings for the T-11 programs that run in the console front-end subsystem.

2. Listings for the microcode that reside in the VAX CPU control RAMs when running diagnostics.

3. Listings for the macrocode that reside in the VAX internal memory subsystem.

Depending upon the nature of the hardware fault, the user may find it necessary to refer to several different listings while analyzing the failure symptoms. Hopefully, the diagnostic will display sufficient fault isolation messages so the need to refer to the actual program listings will be minimized.

The following sections summarize the major types of listings available with the system microfiche library. No attempt is made to describe how to read and interpret the program listings. It is assumed that the reader knows how to interpret the output listings of the macro and micro assemblers used to develop the system firmware and software.

G.2 T-11 MACROCODE LISTINGS

There are three major sets of program listings that describe the T-11 macroprograms used in the system.

1. The VAX 8600-8650 console software (ED0AA)
2. The console diagnostic (ED0BA)
3. The PROM code

The console software program consists of a set of program modules, each with its own listing. The program consists of six major modules.

1. DCN -- Console control code
2. DCP -- Diagnostic Control Program
3. MCP -- Macro Control Program
4. EMM -- EMM control code
5. HEX -- HEX debugger code
6. MHC -- Microhardcore control code

The listings for each of the six major modules are divided according to specific function into two or more separate sublistings with unique identifying names. They all have a .LST extension to identify them as MACRO-11 listings. For example, the listing DCNERR.LST describes the error handling code within the DCN main console control module. MCPMCP.LST describes the main control code within the MCP macro control module.

In addition to the .LST listings, there are three macro definition listings with the filename extension .MAC and one map listing with the .MAP extension. For example, the listing MCPMLB.MAC defines the macros used by the MCP module. The file ED0AA.MAP describes how all the program modules are linked to absolute T-11 addresses. It is used to locate specific segments of program code in T-11 memory.

The listing ED0BA.LST describes all the code in the console diagnostic and uses macros defined in the file SYSMAC.MAC.

Finally, the console PROM code is described in the PROMV36.LST listing.

G.3 MICROCODE LISTINGS

There are three major classes of microcode listings supplied with the system microfiche library.

1. Test microcode used by MHC

2. Test microcode used by the microdiagnostics

3. System microcode used by the operating system and the microdiagnostics

All the microcode listings are identified by the .MCR filename extension.

G.4 MICROHARDCORE LISTINGS

EDKAA, the MHC diagnostic, is described in three sets of listings.

1. EDKAA.DOC provides operating instructions and a general description of the operation.

2. The xxxxxx.LST files describe the T-11 based MACRO-11 program modules where "xxxxxx" is the name of the module. Appendix F, Table F-3, lists the individual MHC program module names.

3. The xxxxxx.MCR files describe the test microcode EDKAA uses. Table F-3 lists the microcode load module names.

G.5 VAX MACROCODE LISTINGS

The two major classes of VAX macrocode versions are the operating system code and the VAX macrodiagnostics. Refer to the VMS operating system microfiche library for information on the operating system. Refer to the EVNDX document within the VAX macrodiagnostic microfiche library for detailed information on how the VAX macrodiagnostics are named, indexed, and described. It is beyond the scope of this manual to describe VAX generic software documentation.

APPENDIX H
FAULT ISOLATION OVERVIEW

H.1 INTRODUCTION

This appendix summarizes the fault isolation design process implemented by the VAX 8600/8650 microdiagnostics. It provides background information on how all the parts of the diagnostic software interplay to display fault isolation messages on the operator's console. Understanding this process will enable the user to interpret the output more intelligently and to deal with unusual problems that affect the process itself.

H.2 MAJOR COMPONENTS

Fault isolation in the VAX 8600/8650 system involves a complex interaction between several major components, as summarized below.

1. A set of microdiagnostic tests, resident in the EBox control store, that provide stimulus/response testing of the individual hardware logic elements

2. A set of predefined EBox scratchpad locations that contain fault syndrome information when a microtest detects a failure

3. Diagnostic Support Microcode (DSM), also resident in the EBox control store, that provides the interface between the microdiagnostic tests and the Diagnostic Control Program (DCP) running in the T-11 console processor

4. A set of files stored on the RL02 and used by DCP to control the process

Since the whole process hinges on the contents of the files resident on the RL02, let's examine them first and then proceed to describe the isolation sequence. Each microdiagnostic program is associated with the following set of files that are read by DCP and used to control the process.

NOTE

xxxxx = name of the microdiagnostic.

1. xxxxx.COM -- This file contains a set of commands executed by DCP to do the following.

   - Initialize the required data structures within DCP itself.
   - Define the name of the microdiagnostic being invoked.
   - Define the EBox scratchpad locations used to convey fault information.
   - Load the EBox control store with the actual microtests from the RL02.
   - Load any other CPU control store (MBox, IBox, and FBox) with any microcode required to support running the microdiagnostic.
   - Load the EBox scratchpad to initialize test data used by the microtests.

2. xxxxE.COM -- This file, invoked from within xxxxx.COM, contains a set of DEPOSIT/ESC commands to initialize the EBox scratchpad.

3. xxxxC.COM -- This file, also invoked from within xxxxx.COM contains a set of DEPOSIT/CACHE commands used to load test data into the data cache for some microdiagnostics.

4. xxxxU.BPN - This file contains the actual microcode loaded into the EBox control store.

5. xxxxx.DCI -- This file contains a set of isolation algorithms read by DCP during actual fault isolation (basic isolation algorithms).

6. xxxxx.DCB -- This file contains a set of additional isolation statements read by DCP during fault isolation (timing information for clock bursting).

H.3 MICRODIAGNOSTIC TEST SEQUENCES

This section describes a typical sequence of events from the time the user types the @xxxxx.COM command, until a fault isolation message is displayed on the console terminal. The EBox diagnostic, EDKBA, will be used to illustrate the sequence. Refer to Figure H-1.

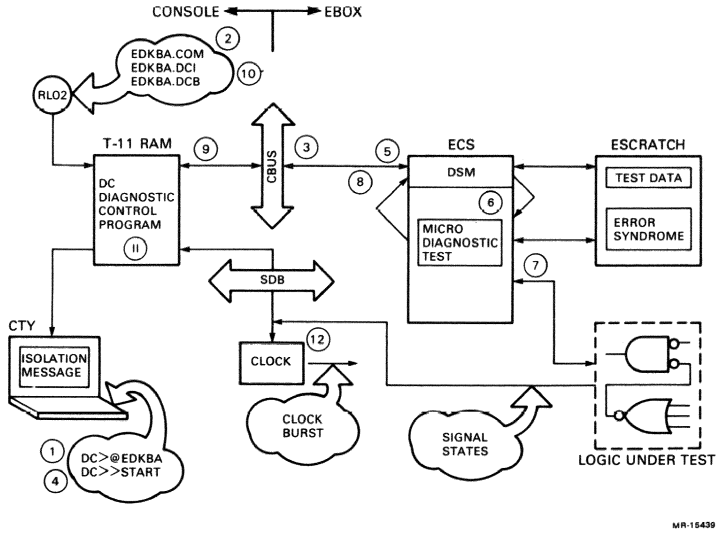


Figure H-1 Fault Isolation Summary

1. At the DC> prompt, the user types @EDKBA to command DCP to initialize the system for running the EBox microdiagnostic.

2. DCP reads the EDKBA.COM file from the RL02 system into T-11 RAM and executes the commands needed to initialize DCP, load the EBox control store with the microtests from EDKBU.BPN, and load the EBox scratchpad with the test data specified by EDKBE.COM.

3. Once loaded, DCP responds with the DC>> prompt to wait for user input.

4. The user now types START to begin execution.

5. DCP sends the START command to DSM, which in turn starts the microdiagnostic at test 1.

6. The microdiagnostic executes test 1 a specified number of times, as determined by the /PASSES:n switch, and if no errors are detected, signals completion to DCP via a call to DSM.

NOTE

The default setting of the /PASSES switch is n=100.

7. DCP then signals DSM to run test 2. It will run the test 100 times and repeat this sequence until all tests have been run.

8. When the microdiagnostic detects a solid fault in one of the tests, it leaves the fault data in the previously defined EBox scratchpad locations and signals DCP that a fault was detected via a call to DSM.

9. DCP retrieves the fault data from the EBox scratchpad, formats it, and displays it on the terminal.

10. Using the test number of the failing microtest, DCP now reads the EDKBA.DCI file to retrieve the isolation algorithm information for that test.

11. DCP executes the isolation algorithm to finally display the fault isolation information that indicates possible failing module(s) and or component(s).

    An isolation algorithm is nothing more than DCP executing a hierarchy of IF statements that analyze test data in one of the following ways.

    a. Direct callout -- The cause of the failure is directly known since the microcode has left a specific error code or fault number in a predefined scratchpad location.

    b. Scratchpad analysis -- All pertinent test data is in the scratchpad, but the isolation algorithm must perform bit-level analysis to isolate the fault properly.

    c. SDB analysis -- The isolation algorithm requires that DCP burst the failing test to key test points, and make isolation analysis decisions based on specific SDB data at those points.

12. When necessary, the EDKBA.DCB file may also be read by DCP to obtain additional isolation instructions. These instructions call for restarting the failing test using clock bursting techniques to retrieve additional fault symptom information.

13. After completing its isolation process, DCP returns to the DC>> prompt to await user input.

14. At this point, the user can use the isolation information to decide which modules, or which components on a module, to replace.

In summary, the actual logic testing is done by the microcoded tests executed out of the EBox control store. The results of the tests are left in designated locations in the EBox scratchpad. The fault isolation process is carried out by a T-11 based program, DCP, under the direction of isolation algorithms from files stored on the RL02 disk pack. As a result of this process, final isolation information is displayed on the console terminal.

The actual design process used to implement system fault isolation will be discussed next.

H.4 FAULT ISOLATION STRATEGY

This section will explain the fault isolation strategy and criteria used to define solid faults that are amenable to isolation call-out. First, let's define how we classify a solid fault. A solid fault must meet two requirements.

1. It must cause a microtest to fail **every** pass as specified by the pass count at run time. The default is that each microtest will be executed 100 times.

2. Each time the failure occurs, it must yield the same error syndrome. That is, the data error patterns must be identical for each pass of the failing test.

Only faults meeting these two requirements will invoke DCP to execute the isolation algorithms specified by the contents of the .DCI and .DCB files. Non-solid faults result in displaying the error data which the user must manually analyze. The output for non-isolatable faults includes the following information.

1. Error reports for up to a maximum of 10 unique error syndromes

2. Counts of the number of times the test passed and failed

3. Count of the number of syndromes encountered, if more than 10 (a message is displayed indicating this condition)

DCP will display a default maximum of 10 different error syndromes.

The user can force the invocation of the isolation process for non-solid faults if the test fails on the first and subsequent passes with different error syndromes. The user simply sets the /PASSES: switch to 1 before starting the test. This forces it to become solid since it now fails every pass (1) with the same syndrome (1). When using this technique, the isolation data may be invalid since the algorithm producing it assumes that the error syndrome was constant.

This discussion will now focus on the mechanics of coarse and fine isolation.

H.4.1 Isolation in General

Refer to Figure H-2. Assume that the user is running a microdiagnostic consisting of 14 tests, T1 through TE. When started, each test is run 100 times (default) and a failure is detected by test 7. Each time test 7 fails, it leaves error information in the EBox scratchpad that conveys the state of the hardware at the time of the failure. If this error information is the same for all 100 passes, the fault is classified as solid. DCP will then invoke the isolation process, as directed by the contents of the .DCI file, to generate the isolation output. The output displayed consists of a dump of the error data itself, plus an isolation report that may include a brief description of what the failure may be and a list of suspected module(s) and components.

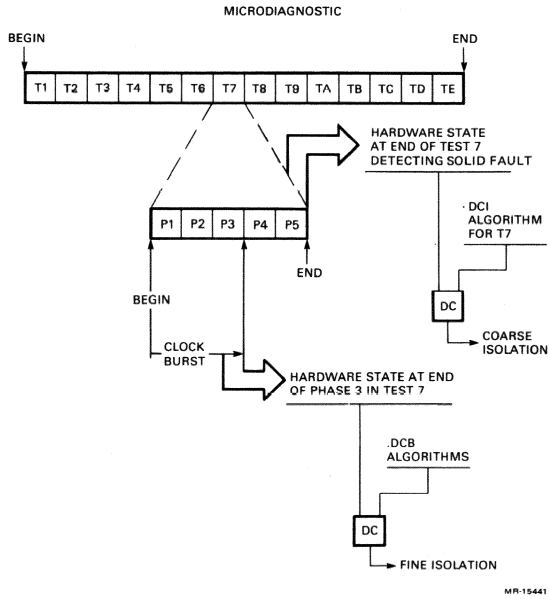


Figure H-2 Fault Isolation Overview

H.4.2 Isolation with Bursting

Burst isolation involves repeating the failing test using a clock bursting technique to capture the state of the hardware logic at intermediate check points within the failing test. If available, the isolation algorithms within the .DCI file will instruct DCP to read additional information from the .DCB file. DCP will be asked to restart test 7 at the beginning, "burst" the clock the required number of ticks to capture the state at the end of P3, for example, and examine the state of specific SDB signals. This information provides more detailed localization of the fault.

H.4.3 Summary of Fault Isolation

Briefly, the fault isolation strategy involves the following steps.

1. The microdiagnostic detects the failure and saves the error information.

2. DCP simply retrieves, formats, and displays this error information if the fault is non-solid (no isolation).

3. DCP uses the .DCI and .DCB files to analyze the error information and append a list of possible failing module(s) and components.

As VAX 8600 and 8650 systems mature, users will learn more about how they fail, how they display the failure symptoms, and these symptoms can be more accurately associated to the Field Replaceable Unit (FRU). The microdiagnostics and isolation algorithms can then be modified and expanded to provide more accurate and concise fault isolation displays.

H.5 ISOLATION DESIGN PROCESS

Refer to Figure H-3. The design process involves three programming groups. They jointly produce all the system components needed for VAX CPU fault isolation. The process begins when the diagnostic engineer designs the microdiagnostic tests and isolation algorithms. It ends when the service engineer runs the microdiagnostic on the failing system, using the isolation call-out messages to indicate the possible failing modules.

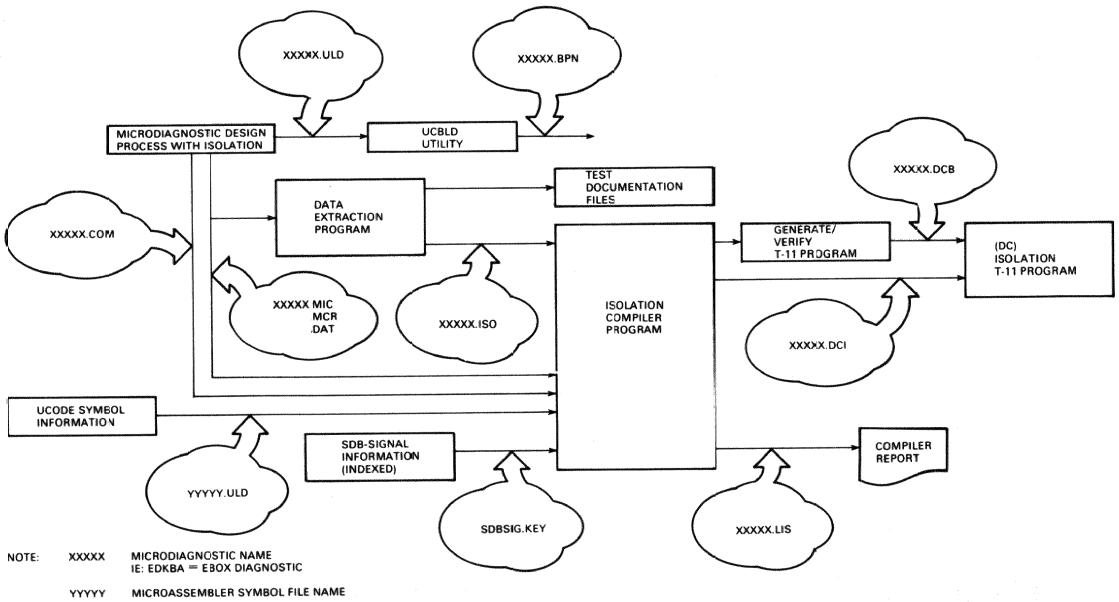


Figure H-3 Fault Isolation Design Process

H.5.1 Microdiagnostic Design Process

The diagnostic engineer analyzes the hardware logic and designs a logical sequence of microcoded tests to stimulate specific logic circuits and check for the correct responses. For each incorrect response, the engineer optionally assigns a fault identification number and saves the results of the test as an error syndrome in one or more (up to 16) EBox scratchpad locations. This error syndrome is usually the following information.

- Test number that failed
- Fault identification number
- Data patterns used to stimulate the logic under test
- Expected response data patterns
- Actual response data patterns
- Difference data patterns between expected and actual

For each fault defined, the diagnostic engineer must provide a procedure for analyzing the error syndrome information so as to associate it with a fault isolation message that clearly indicates the suspected module(s) and components that could be causing the failure. These analysis procedures are then coded as algorithms using a specialized VAX 8600/8650 fault isolation language and embedded into the actual microdiagnostic program along with the test microcode.

When the diagnostic engineer assembles the program, the microassembler produces three files that are used as inputs to the isolation compiler process: xxxxx.ULD(binary), xxxxx.MCR(listing), and xxxxx.MIC (source). An indirect command file (xxxxx.COM) specifies the procedure for loading and initializing the microdiagnostic at run time.

In summary, the diagnostic engineer designs the actual test microcode and the algorithms for interpreting and reporting the results to the service engineer. Next will be a discussion on how this information is used by the rest of the process.

H.5.2 UCBLD Utility

The microassembler produces a binary loadable file, xxxxx.ULD, that must be loaded eventually into the EBox control store from the RL02 system via the console subsystem. This file is processed by the UCBLD (Microcode Build) utility to generate an encoded binary file, xxxxx.BPN, that can be stored efficiently on the RL02 system and read by the console software at load time.

H.5.3 Data Extraction Utility

This utility uses the microassembler-generated files and extracts the embedded isolation language statements to produce a source file, xxxxx.ISO, that can be read and compiled by the isolation compiler program. A by-product of this program is a set of test documentation files that can be used to review the actual test algorithms implemented in the microdiagnostic, and to understand the actual test sequencing and fault isolation strategy for any program.

H.5.4 Isolation Compiler Program

This program was designed by the Diagnostic Tools Group to support the fault isolation process. It accepts input from the following four sources.

1. The .ISO file from the Data Extraction Program and the .MIC, .MCR, and .DAT files from the microdiagnostic design process

2. The .COM file from the microdiagnostic design process

3. The .ULD file containing microcode symbol information from the microassembler

4. The SDB signal name information from the files specified by SDBSIG.KEY

It compiles all of these inputs to produce three files as output.

1. The xxxxx.LIS file contains a listing and report of the compilation.

2. The xxxxx.DCI file contains the machine-readable coarse isolation algorithms read by DCP at run time.

3. The xxxxx.DCB file contains the machine-readable burst data read by DCP at run time to extend and supplement the algorithms contained in the xxxxx.DCI file. This file is actually generated to verify proper clock-bursting information.

Figure H-4 illustrates the relationship between the primary input and output from the isolation compiler for the EDKBA. EDKBA.ISO is the ASCII source input that contains the isolation algorithms specified in the specialized language, while EDKBA.DCI contains the machine-readable hexadecimal code read by DCP from the RL02 disk. Each source statement in the source file is assembled as a line or more in the binary file that is terminated by the 2-byte sequence 0D 0A (<RETURN> <LINEFEED>) in hexadecimal code. Each line in EDKBA begins with a single byte that contains a character count of the line, including the byte count and <RETURN> <LINEFEED> characters. The next two bytes represent the line number of the statement in the source code. The fourth byte specifies a line type identifier indicating how DCP is to interpret the remaining COUNT-6 bytes in the line. The next example will show how it works.

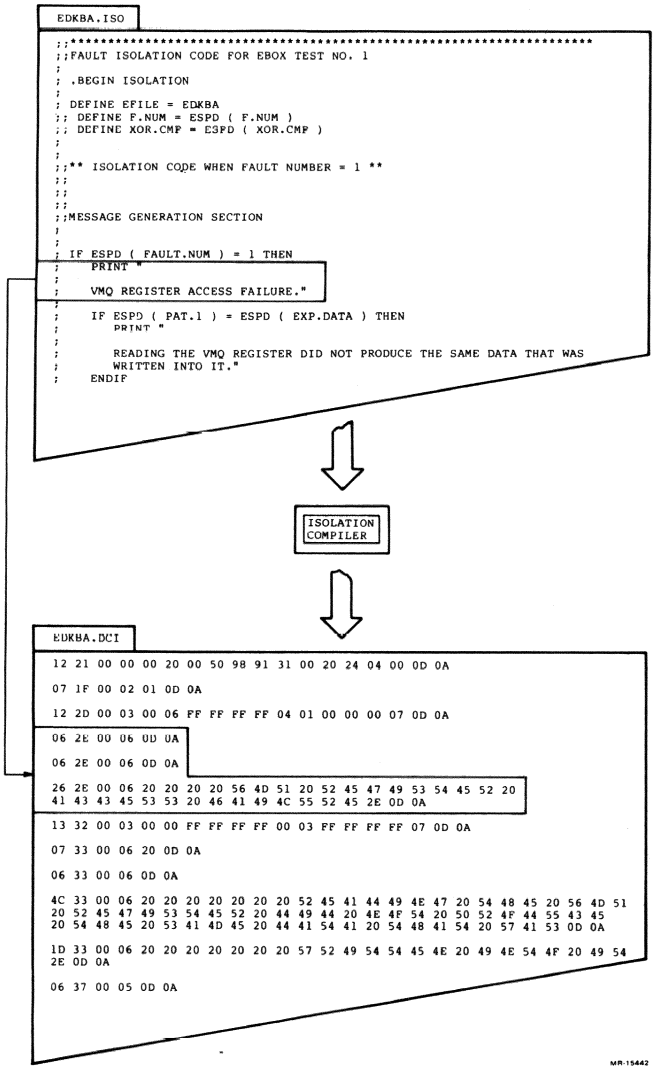


Figure H-4 Isolation Files Format

Consider the following source statement.

```
PRINT "

VMQ REGISTER ACCESS FAILURE."
```

When written into EDKBA.DCI, it becomes the following.

```
06 2F 00 06 0D 0A

06 2F 00 06 0D 0A

26 2F 00 06 20 20 20 20 56 4D 51 20 52 45 47 49 53 54 45 52 20
41 43 43 45 53 53 20 46 41 49 4C 55 52 45 2E 0D 0A
```

The encoding of this line follows.

- 26 -- Indicates 38 bytes in the line
- 2F00 -- Indicates the line number for the PRINT statement in the microdiagnostic listing
- 06 -- Specifies the line ID for the PRINT statement
- 0D 0A -- Specifies the line terminator, <RETURN> <LINEFEED>

The remaining 32 bytes between the 06 and the 0D 0A contain the ASCII message actually displayed and include four leading spaces. It is left as an exercise for the reader to verify that these bytes result in printing the specified message.

This brief discussion was meant to give the reader a glimpse of the magic performed by the isolation compiler and DCP and is not required when using the microdiagnostics.

Now we should be ready to examine the final step of the process.

H.5.5 DCP Isolation Program

DCP executes the process at run time which produces the isolation messages the user needs to determine the possible failing components. When a fault is detected by the microdiagnostic tests running in the EBox control store, DCP retrieves the error syndrome information specified by the xxxxx.COM command file, reads the xxxxx.DCI and xxxxx.DCB files to retrieve the required algorithms, and executes the analysis procedure to display the fault isolation messages to the service engineer.

H.6 SUMMARY

System fault isolation is a complex process involving interaction between three separate design groups.

- The Microdiagnostic Engineering Group
- The Fault Isolation Tools Group
- The Console Software Design Group

These groups produce a set of files, resident on the RL02 disk, that are used at run time to test the VAX CPU and generate fault isolation call-out messages that tell the service engineer which module(s) and or components are the possible cause of the failure. A list of the files required to support EBox microdiagnostic execution is shown next.

- EDKBA.COM -- Top-level, indirect command file executed by DCP
- EDKBU.BPN -- Test microcode loaded into the EBox control store
- EDKBE.COM -- Lower-level command file used to set up the EBox scratch pad
- EDKBA.DCI -- File containing the coarse isolation algorithms
- EDKBA.DCB -- File containing the fine isolation information

This completes the summary of the fault isolation process.

APPENDIX I
REMOTE DIAGNOSIS

I.1 INTRODUCTION

Remote diagnosis is a key ingredient in the overall maintenance strategy of VAX 8600/8650 system. Special hardware and software features have been designed into the system to permit establishing a remote connection to the console subsystem via a telephone line for the delivery of a wide range of remote services. This appendix provides a general description of the types of remote services available, the general operation of the hardware and software features designed into the system, and also includes detailed procedures for establishing the remote connection.

I.2 REMOTE SERVICES

Figure I-1 provides an overview of remote services delivery. For purposes of this discussion, the system under test will be referred to as the Unit Under Test (UUT). The service delivery may occur either via the remote port on the console module or via a standard timesharing dial-up line. The major differences between the two methods are as follows.

1. The standard dial-up line requires the operating system be up and running and is limited to delivering only those services supported by the on-line operating system. It does not permit access to the system diagnostic console software.

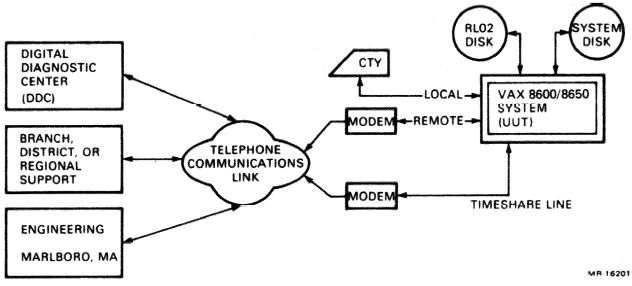


Figure I-1 Remote Diagnosis Overview

2. The remote port permits access to the system diagnostic console software. This allows testing a system that is unable to run the operating system due to the nature of the hardware fault. In addition to standalone testing using micro and macro diagnostics, this method also permits access to the operating system for on-line testing.

The remote service required may be delivered from any location with a terminal, modem, and access to a telephone communications link to the UUT. In general, the service is provided from one of three possible sources.

1. The Digital Diagnostic Center (DDC)
2. The branch, district, or regional support offices
3. The Engineering group in Marlboro, Ma.

For U.S. and Canadian customers, the primary source of delivery will be from the DDC in Colorado Springs. Basingstoke and Valbonne are delivery sources for Europe. Hardware and software support specialists at the branch, district, or regional offices can also dial-in to provide additional assistance. When necessary, the hardware and software design engineers in Marlboro can connect to provide technical expertise on the more difficult problems.

I.2.1 Types of Remote Services

The Remote Services Delivery System (RSDS) at the DDC uses a host computer, manned by VAX 8600/8650 support specialists, to provide the following types of services.

1. Remote Diagnosis -- A hardware support specialist can connect to the remote port and run diagnostics to isolate hardware faults to a Field Replaceable Unit (FRU). All test activity during the session is logged and saved on the DDC host computer and can be recalled for review during any future service calls.

2. Hardware Monitoring -- The RSDS host can connect to the UUT and monitor system operation for some fixed interval to attempt to capture the cause of intermittent faults. During the monitoring operation, all system responses from the UUT are saved in a session log and subsequently analyzed by a hardware support specialist.

3. Error File Analysis -- The RSDS host system supports Remote File Transfers (RFT) that permit transferring system event files from the UUT's system disk to the host computer for analysis. This transfer can occur either via the remote port or via a dial-up timesharing line.

   In addition to the system event file, the system also saves snapshot files created by the console software and saved on the RL02 disk. If the system can be successfully restarted, the snapshot files are copied to the system disk. The RSDS host is able to transfer these files from either the UUT's system disk or from its RL02 disk. The host computer can also analyze these files to determine the cause of hardware faults.

4. Software Updates -- The RSDS host computer can down-line load software updates to the UUT.

I.2.2 Remote Services Features

The system remote connection facilities provide the following features required by the RSDS host.

1. Local copy -- This feature is turned on by a SET LOCAL-COPY ON command from the local CTY and can not be turned off from the remote port. It provides a means of maintaining an audit trail of all the activity occuring over the remote port during any session. It is necessary to satisfy both the legal and customer requirements for remote service sessions.

2. Talk mode -- This feature allows the remote user and the local user to talk using the remote electronic connection. All messages typed on the local CTY are echoed on the remote RTY and vice versa. It is implemented on the console by starting each message line with the "!" character which is interpreted by the console software as a comment and not executed as a command line.

3. Parallel control/monitor mode -- This feature allows either the remote user or the local user to enter commands to the system. It allows the local user to override any command entered remotely at any time. Its inclusion requires cooperation on the part of both the local and remote user because input from one terminal can disrupt the input from the other.

4. Remote file transfer -- This feature requires that special software modules be installed on the system under test to communicate with the RSDS host computer so that files can be transferred between the two systems.

In summary, the remote diagnosis features provide an efficient and timely method of delivering service to the customer. The next section will describe the actual hardware and software features implemented.

I.3 HARDWARE FEATURES

This section summarizes the hardware features provided with the system. Three major components comprise the remote hardware in a system.

1. A Programmable Communications Interface (PCI) built into the console module provides the communications link between the console software and the remote port.

2. One switch and two LED indicators on the System Control Panel (SCP) are used for operator control of the remote port.

   a. Terminal Control Switch -- This switch has two positions to control the remote port's mode of operation.

      - REMOTE DISABLE -- In this position, remote access is enabled in only PIO mode, limiting the level of control given the remote user. This position is interpreted as "REMOTE terminal activated, but <CTRL/P> disabled."

      - REMOTE -- In this position, remote access is allowed in both CIO and PIO mode, giving the remote user full control of the system. This position is interpreted as "REMOTE access activated and <CTRL/P> enabled."

   b. LED indicators -- Two green LEDs indicate the state of the remote port.

      - REMOTE ACTIVE -- This green LED is set to the ON state when the carrier from the modem is asserted and set to the OFF state when the carrier is lost.

      - REMOTE ENABLE -- This green LED is set to the ON state when access to the RD port is allowed. When REMOTE ENABLE is ON, it indicates to the operator that the system is not secure from remote access.

3. A modem connects the remote port on the console module to a direct-dial phone line in the computer room.

I.4 SOFTWARE FEATURES

The hardware features just described are controlled by the T-11 based console software. This section provides a description of the console software's operation during remote access.

I.4.1 PROM Remote Support

The PROM code services the remote port with a much simpler procedure than the one used by the console program, since the PROM code has neither timer nor interrupt capability.

After the PROM code completes its power-up self-tests, which include loopback tests on the remote PCI, it configures the PCI to operate with default characteristics (the same defaults used by the SET TERMINAL command). Once the PCI is set up, and if the front panel Terminal Control Switch is in the REMOTE position, the PROM asserts the DTR signal to the modem, allowing the modem to turn on-line. If the DSR and CD signals from the modem are sensed as TRUE, then all character output to the CTY will also be sent to the remote port and input will be accepted from either. Note that because the DTR signal to the modem is not asserted until the end of the self-tests, it will be virtually impossible to establish a modem connect to the remote port during system power-up initialization while the PROM is still running. However, a terminal connected locally with a NULL MODEM cable should not have any problem getting control of the PROM code, and it is for this reason that this scheme becomes useful.

Once the PROM asserts DTR to the modem, the dial-in capability is activated. When the console program starts running, it begins to service the remote port immediately, thus giving the remote user control throughout PROM-to-console program transitions, and vice versa. This means that if T-11 control transfers to the PROM because of some unexpected condition, such as for a T-11 halt or unexpected interrupt, the PROM can continue to service the remote connection.

During the handling of connections and disconnections of the remote terminal by the PROM code, the state of the front panel indicators is updated in accordance with how this document defines those indicators.

I.4.2 Console Software Remote Support

The descriptions of remote handling in this section apply to the console program (EDOAA) only; the PROM code has the simplified connect and disconnect procedure discussed in the previous section. The modem connect and disconnect protocols described in Digital Standard 052 have been used in the design of the remote terminal support. This implementation allows console modem support to work properly in all areas of the world.

I.4.2.1 Establishing a Remote Connection -- The following conditions must exist before the console program can make a remote port connection.

- If the console program is in CIO mode, the Terminal Control Switch must be in the REMOTE position.
- If the console program is in PIO mode, the Terminal Control Switch must be in either the REMOTE or REMOTE DISABLE position.
- A modem device must be connected to the rear async-panel port for the remote terminal, or a nearby terminal may be connected using a null modem cable (the null modem must assert DSR and CD to the console).

With either of the first two items satisfied, the console program asserts the Data Terminal Ready (DTR) and Ready To Send (RTS) signals to the modem, turns on the REMOTE ENABLE light, and waits indefinitely for DSR from the modem. Note that the console deasserts the Data Set Rate Select (DSRS) signal during its initialization.

Once DSR asserts, the console program begins a 30-second timer while waiting for Carrier Detect (CD) to assert from the modem. The console program does not look at the state of CD until approximately 520 ms after the assertion of DSR to avoid data transfers during this turn-on period.

If the 30-second timer expires before CD asserts, then the console initiates a disconnect sequence (see next section). Otherwise, the assertion of CD triggers the console program to turn the front panel REMOTE ACTIVE light on, and to send the "<CR><LF>Console Password?" message to the remote terminal if the password feature is enabled (see SET TERMINAL command). If the password prompt is sent, the console begins a two-minute timeout and waits for the correct password. If the timeout occurs, a disconnect sequence is initiated.

Once the password step passes successfully and the console program is running in PIO mode, an RX register request is sent to the CPU. If not in PIO mode, the RX register request is bypassed and the console program enters a low-activity state when it processes CTY and remote characters and monitors different disconnect indicators. At this point, the remote connection has been made (REMOTE ACTIVE).

I.4.2.2 Remote Disconnection -- A disconnect sequence will occur at any stage of the remote port connection and steady-state operation if one of the following events occurs.

- CD is not asserted within 30 seconds after the assertion of DSR.
- The password is not entered within two minutes.
- DSR from modem is negated.
- CD from the modem is negated for more than two seconds.
- The Terminal Control Switch is changed to a non-REMOTE position.
- If running in PIO mode and the VAX operating system (e.g., VMS) deasserts the logical DTR bit in the RXCS register (connection dropped). Note that this will cause a disconnect only if the logical DTR bit changes from a set to a clear state.

The disconnect sequence involves several steps. First, the DTR signal to the modem is deasserted and a two-second timer is started. The remote port transmitter and receiver are turned off and the logical DTR flag to the CPU is deasserted. The disconnect sequence ends when either the DSR signal from the modem deasserts or the two-second timeout occurs.

I.4.2.3 Operation -- Once a remote port connection has been established and the console password prompt has been satisfied, the console processes input and output characters with the remote terminal. Note, however, that if the CPU program failed to respond to the RX register request made by the console during the final steps of the connection process, then remote input is not passed to the CPU (it is ignored). The console program will only disconnect the remote connection if the CPU first accepts the line, and then drops it. This allows the remote user to remain connected while running a nonoperating-system type program in the CPU. The <CTRL/P> break character is honored by the remote user if the front panel switch is in the REMOTE position.

In PIO mode, remote port input characters are sent to the VAX CPU through the RX register interface with no control character checking or masking (providing, of course, that the CPU has accepted the remote connection). Remote characters are echoed by the VAX operating system through the TX console. While in PIO mode, input from the remote and local ports and from the CTY are sent to the VAX CPU independent of each other. In this way, the two terminal ports behave as separate devices, even though the console is controlling character flow to and from both of them.

In CIO mode, input characters from the remote port are echoed (to remote port only) and placed in a command input buffer. The standard command line control characters (<CTRL/U>, <CTRL/R>, <CTRL/O>, and <DELETE>) can be used by the remote user. When the command line terminator is entered, the contents of the remote command buffer are displayed on the CTY device and the command is parsed and executed. Any output resulting from the command is sent to both terminal ports.

The following operational characteristics of the remote handler should be noted.

- The console password is not effective until the SET TERMINAL/PASSWORD command is encountered (optionally) in the LOAD.COM file during macro context initialization. If the front panel Terminal Control Switch is in the REMOTE position and a remote connection occurs before this command is encountered, then the remote connection will be made without a password. To safeguard against this, the Terminal Control Switch can be placed in any non-REMOTE position during system initialization.

- When prompted, the password must be entered within two minutes. Upper and lower case characters are interchangeable. Press <RETURN> to terminate input.

- The remote user's input characters are echoed only while the console program is executing in its null loop. If the console program is executing a command (whether it was entered by the remote or local user) then input character echoing will be suspended until the executing command completes. The remote port input buffer in the console program can easily be filled using the type-ahead process while the console program is executing a command.

- The SET TERMINAL command cannot be used from the remote port, with the exception of the /SCOPE and /NOSCOPE switches.

- In both CIO and PIO mode, the console program will send the XOFF control character to the remote port when the console's input ring buffer is three-quarters full. This is indicated by the appearance of the KBD LOCKED LED on a VT100 device. The XON control character is sent once when the input buffer is one-quarter full.

- When in CIO mode, the <CTRL/S> control character (XOFF) entered from either terminal will suspend console program operation for both terminals. A <CTRL/Q> (XON) entered at the same terminal will resume console program output. A <CTRL/C> or <CTRL/P> entered from either terminal forces an XON sequence which frees terminal output if locked from a previous XOFF.

I.5 OPERATING PROCEDURES

This section describes the actual procedures required to set up the system for the delivery of remote service.

I.5.1 Remote Services Delivery Via Timeshare Line

When the connection is made via a dial-up timesharing line, the procedure is simple. The customer or Digital representative makes a phone call to the location providing the service and supplies the following information.

1. The customer name, address, and system ID
2. A description of the problem to be solved
3. The telephone number of the dial-in line
4. The baud rate of the dial-in line
5. The name of the account to be used and its password

With this information, the call can be logged and a remote connection established to deliver the required service. The services that can be provided are limited to running on-line diagnostics, system dump analysis, error file transfer and analysis, and software updates.

I.5.2 Remote Services Delivery Via Remote Port

If the remote service requires on-line testing under VMS, the user may need to execute the following commands to connect the RD port.

```
$ RUN SYSGEN
SYSGEN>CONNECT CONSOLE/REMOTE
```

Normally, these commands should be included in the SYSTARTUP.COM file to automate the process. Note that the SCP switch still controls remote access.

When the connection is made via the remote port, the setup procedure is more involved. The customer must perform the following additional steps.

1. Use the SET LOCAL-COPY ON command to ensure that an audit trail is left during the remote session. The default setting for this switch is ON and can only be turned OFF from the local CTY.

2. Use the following commands to set the receive/transmit baud rate for the remote port.

   ```
   SET TERMINAL/RECEIVE:nnnn

   SET TERMINAL/TRANSMIT:nnnn
   ```

   where nnnn=baud rate (1200 or 300)

3. Use the following command to set a login password for the remote port. If no password is set, remote access is allowed as soon as the connection is established.

    SET PASSWORD[<password>]

    where <password> is a string of up to six alphanumeric characters (0--9, A--Z)

    NOTE
    The SET commands should be included in the ULOAD.COM command file to eliminate the need to reset them if the system is reinitialized during the remote session.

4. Place the Terminal Control Switch to either the REMOTE or REMOTE DISABLE position.

5. Set the modem baud rate select switch to the baud rate specified in the SET TERMINAL commands.

6. Call the DDC or other location that will provide the service and supply the following information.

    a. The customer name, address, and system ID
    b. The description of the problem to be serviced
    c. The telephone number of the dial-in line
    d. The baud rate of the dial-in line
    e. The password (if set) for the remote port
    f. The name of the account to be used and its password, if the service requires logging into the operating system

Optionally, VAX/VMS V4.1 provides the user a way to preserve their process, which will preserve their work when a dial-up line drops the carrier. Virtual terminals enable a VMS process to exist without being assigned a physical terminal. For instance, when the RD center loses connection to a system due to a noisy phone line, work can be saved. This is done by creating a virtual terminal and by assigning the physical terminal the /DISCONNECT attribute. These commands are needed when working with a system via an RD port if the operating system is running.

If the system has designated a virtual terminal and the RD line drops the carrier, the user can dial back in and login into the same account. Upon doing so, the user is asked if it is necessary to connect to the former process. By reconnecting to the former process, all work done in the previous work session is saved.

```
sysgen> connect vta0/noadapter /driver=ttdriver
                                !set up virtual terminal

sysgen> exit
$ set term opal: /perm/hang/auto/modem/DISCONNECT
```

At this point, a remote connection may be established and the service delivered. In addition to the services described in the previous section, the support engineer on the remote terminal can switch to CIO mode and run standalone micro and macrodiagnostics (if the Terminal Control Switch is set to the REMOTE position).

APPENDIX J
RL02 MAINTENANCE AND UTILITIES

J.1 OVERVIEW

This appendix provides information about the VAX 8600 and VAX 8650 diagnostics distribution strategy and the system utilities and procedures required to either build an RL02 diagnostic disk pack or update an existing one.

J.2 DIAGNOSTIC DISTRIBUTION STRATEGY

The system uses an RL02 as the front-end load device; this is used for loading console software, system microcode, microdiagnostics, and load path macrodiagnostics. All console software, microdiagnostics, and load path macrodiagnostics will reside on one RL02 pack. All other macro-level diagnostics will be in the [SYSMAINT] account on a VMS disk.

The distribution of diagnostics for the VAX 8600 and VAX 8650 systems will be different from current VAX CPUs being shipped. These systems will follow the new Digital Diagnostic Policy which states that diagnostics will no longer be shipped with the system from manufacturing. At the moment, this policy applies only to those products introduced after the establishment of the policy; however, in the future, this policy may be applied to all products.

To implement this policy, manufacturing will ship two RL02 disk packs with the system: one blank scratch pack and one Console No Diagnostics Pack. The Console No Diagnostics Pack will contain all the system microcode and console software necessary to boot and run the system; it will not contain diagnostics.

The VAX 8600 and VAX 8650 Diagnostics Kits will be shipped to those branches that will be maintaining VAX 8600 and VAX 8650 systems. The diagnostic kit will contain an RL02 console-w/ diagnostic pack, a VAX 8600/8650 diagnostic tape set, and a VAX 8600/8650 microfiche library. The RL02 disk pack will contain all the microcode, console software, microdiagnostics, and load path macrodiagnostics. The tapes will contain all the system microcode, console software, and diagnostics for the system and associated peripherals. The microfiche will contain the console software, microdiagnostics, the VAX 8600- and 8650-specific macrodiagnostics, and any generic macrodiagnostics which have been modified for the VAX 8600/8650 systems.

During installations, the branch will use the RL02 console-w/diagnostic pack to install the VAX 8600 or VAX 8650 system. Once the system installation is complete, a console-w/diagnostic pack must be built onto the scratch RL02 pack that is shipped with all systems under warranty or contract. A command procedure (RL02.COM) will be supplied for building this diagnostic pack. This command procedure resides on the console-w/diagnostic tape.

When the warranty/contract expires, Digital policy states that the branch must remove (delete) all Digital diagnostics from the site. A command procedure (RL02D.COM) will be supplied for deleting diagnostics from the RL02 pack.

For per-call customers, the branch engineer will have to bring a console/diagnostic pack to the site for diagnosing the system. When the branch engineer leaves the site, the diagnostics must be removed.

Self-maintenance customers, and customers with third-party maintenance agreements must purchase a diagnostic license along with the diagnostic pack or tape. For more details on this subject, refer to existing Digital Diagnostic Policy.

J.2.1 Diagnostic Update Distribution Process

The Standard VAX Field Service Update tape (VAXPAX) used today (order number ZE999-HM) has been split into two tapes with the introduction of the VAX 8600 system. Tape 1 (BB-S409Y-YE) contains all existing EVxxx generic diagnostics. Tape 2 (BB-T988A-YE) contains all of the VAX family CPU-specific console code, microdiagnostics, and load path diagnostics.

Note that both tapes may be ordered with the kit number ZE999-HM (the same number used today to order the Standard Field Service Update tape). The VAX 8600/8650 microfiche will be included in the standard VAX microfiche library. Thus, those people already receiving update tapes and microfiche will automatically receive the VAX 8600 and VAX 8650 diagnostics and microfiche.

The first distribution of VAX 8600 diagnostics and microfiche through the standard VAX update process will correspond with either VAX Diagnostic Release 2.1 or 2.2.

Prior to the formal release, VAX 8650 diagnostics will be distributed to the field by Marlboro Engineering. After joining the VAX release process, Marlboro will no longer support updates, and the normal VAX diagnostic process must be used to maintain the proper diagnostic and microfiche revisions in the field.

All basic VAX 8600/8650 system configurations will include a tape drive, and therefore the normal diagnostic update media will be tape. However, branches may subscribe to any combination of tape or RL02 media that they wish to receive.

US Area and GIA engineers may get on the automatic update of VAX diagnostics by contacting the library at Colorado Springs with either DTN 522-5050 or 1-800-525-6570. Europe may get on the automatic diagnostic update list by contacting the IDS. When contacting either Colorado or IDS, be prepared to furnish the following information.

1. Name
2. Cost Center
3. Badge Number
4. Address/Location
5. Diagnostic Media Type (or part number)

J.2.2 SDC Diagnostic Part Numbers

New part numbers have been created to support the VAX 8600 system. The new part numbers that have been created start with the letter B. The seventh character of the new part numbers (Bx-xxxxA-xx) is the revision. Thus, every time the media is updated the revision letter will change (for example, BC-T989A-DE would become BC-T989B-DE).

The console-w/diagnostic BB-T990A-DE tape and macro diagnostic BB-FF15A-DE tape contain all the diagnostics and console software to support the VAX 8600 system. These tapes contain neither diagnostics nor console software for the rest of the VAX family.

Also, new kit numbers have been created to package the necessary media, license, and documentation for customers. Tables J-1 through J-3 list the diagnostic media available from the SDC.

Table J-1 VAX 8600 Release Packages

SDC Part Numbers	Title (29 characters)	Description
BC-T987A-ME	VAX 8600 Console-No Diag Pack	RL02 pack (No Diag)
BB-FF58A-ME	VAX 8600 Console Update Tape	Console update tape
BC-T989A-DE	VAX 8600 Console-W/Diag Pack	RL02 pack (With Diag)
BB-T990A-DE	VAX 8600 Console-W/Diag Tape	VAX 8600-only tape
BB-FF15A-DE	VAX 8600 Macrodiag Tape	VAX 8600-only tape
BB-S409U-YE	Field Service Update Tape PT-1	(Existing EVXXX)
BB-T988A-YE	Field Service Update Tape PT-2	(VAX CPU Diag)

Table J-2 Field Service Automatic Update Kit

SDC Part Number	Description
ZE999-HM	Kit including BB-S409U-YE and BB-T988A-YE

Table J-3 Customer Kits

SDC Part Number	Description
ZK200-DZ	SPD for VAX 8600 (Single-Use Diag License)
ZK200-CQ	Kit including BC-T989A-DE, BB-T990A-DE, BB-FF15A-DE, and guides
ZK200-HQ	Update Kit of ZK200-CQ (depending upon what changed could contain any of the following: BC-T989A-DE, BB-T990A-DE, or BB-FF15A-DE)

J.2.3 VAX 8600 Diagnostic Release 2.0 and 3.0

VAX 8600 diagnostic releases 2.0 and 3.0 were shipped to the field on March 15, 1985. Only those branches receiving VAX 8600 systems received the diagnostic kits. Release 2.0 supports VAX 8600 systems with the MCC revision J module, while release 3.0 uses the MCC revision K module.

Release 3.0 will not run on a MCC revision J module. Therefore, if a MCC revision J module is replaced with a MCC revision K, or vice versa, a new diagnostic pack will have to be built to match the installed revision. However, if you replace a revision K module with a revision J, you should Pl a revision K and return the customer's system to the original revision. An FCO to upgrade all systems in the field to MCC revision K occurred around May or June 1985.

Some early released 2.0 kits were shipped with a BB-FF15A-DE tape instead of a BB-FF15B-DE. These tapes are basically the same. The BB-FF15A-DE had some unnecessary files which were deleted, creating a BB-FF15B-DE revision.

Tables J-4 and J-5 summarize release versions 2.0 and 2.3.

Table J-4 VAX 8600 Diagnostic Kit 2.0

SDC Part Number	Title
BC-T989B-DE	VAX 8600 Console-W/Diag Pack
BB-T990B-DE	VAX 8600 Console-W/Diag Tape
BB-FF15B-DE	VAX 8600 Macrodiag Tape
MD-8600B	VAX 8600 Fiche Library

Table J-5 VAX 8600 Diagnostic Kit 3.0

SDC Part Number	Title
BC-T989C-DE	VAX 8600 Console-W/Diag Pack
BB-T990C-DE	VAX 8600 Console-W/Diag Tape
BB-FF15C-DE	VAX 8600 Macrodiag Tape
MD-8600C	VAX 8600 Fiche Library

J.2.4 Speedy Update Process

A new update process has been implemented for the VAX 8600 and VAX 8650 systems which updates machine-critical microcode and console software quickly. These speedy updates will be provided via a tape (part number BB-FF58A-ME). Changes made through this procedure will bump the minor revision of the diagnostic release (for example, 2.0 would become 2.1).

J.3 THE EXCHANGE UTILITY

Building or updating an RL02 diagnostic disk pack requires using the VMS utility, EXCHANGE. The EXCHANGE utility program is used with mass storage volumes that are formatted for operating systems other than VAX/VMS. It performs file transfers and format conversions for the following file structures.

1. DOS-11 magnetic tape volumes
2. Files-11 volumes
3. RT-11 block-addressable volumes (VAX 8600 RL02 disk)

In addition to transferring files, EXCHANGE allows the user to perform the following operations.

1. Initialize foreign volumes
2. List directories of volumes
3. Delete files from block-addressable volumes
4. Rename files on block-addressable volumes
5. Write boot blocks on VAX-11 processor consoles
6. Mount and dismount foreign volumes

To obtain more detailed information about the EXCHANGE utility, use the VMS HELP EXCHANGE command after logging on to the system.

Figure J-1 shows how to use EXCHANGE to obtain a directory listing of all files on the RL02 with a ".TXT" extension, while Figure J-2 shows how to copy the NOTICE.TXT file from the RL02 to the [SYSMAINT] account on the system disk using the EXCHANGE utility.

```
$EXCHANGE DIR CSA1:
```

Figure J-1 EXCHANGE Directory Command Example

```
$ EXCHANGE COPY [SYSMAINT]NOTICE.TXT CSA1:NOTICE.TXT
```

Figure J-2 EXCHANGE Copy Command Example

J.4 OPERATING PROCEDURES

The next two sections describe the procedures for maintaining the RL02 disk, building a diagnostic disk pack from the distribution tape, and updating existing files.

J.4.1 Building an RL02 Pack

This is a simple, two-step procedure that is entirely automated by a command procedure contained in the file RL02.COM, which resides on the distribution magnetic tape. Figure J-3 shows a listing of this file. The first step is to copy all of the tape files onto the system disk, and then copy selected files from the system disk to the RL02 disk pack. The following procedure outlines the process, which may require some modifications depending upon the actual system configuration.

1. Load the RL02 build tape onto the magtape unit (SDC Part No. BB-T990B-DE or BB-T990C-DE).

2. Log into the field service account using the username FIELD and the password assigned by the system manager.

NOTE

Approximately 50,000 blocks of free disk space is required to perform the build. Consult the system manager if you need this additional space. Remember that this allocation is only temporary, since the command procedure will delete all the files read from the tape after the disk pack is built.

```
$ ! RL02.COM  version V2.0
$ UPDATE_ID := "D-"
$ TEMP_DIR := "RL2CREATE"
$ TAPE_DIR := "RL2TAPE"
$ PROC = ""
$ NEW_TYPE = ""
$ LAST_TYPE = ""
$ SAVE_DEF = ""
$ TAPE_DEF = ""
$ DISK_DEF = ""
$ COUNTER = 0
$ !
$ ! This command procedure builds console media from magtape.
$ ! Modified 14-MAY-1985 to accomodate 8600's which do not
$ ! have a tape drive on their system and to be able to more
$ ! quickly build more than one pack on the system.
$ !
$ on control_y then goto EXIT
$ SET PROCESS/PRIV=(CMK,VOLPRO)
$ save_def =  f$environment("default")
$ IF  P1 .EQS. "" THEN GOTO INIT
$ IF F$LOCATE("CON", P1) .NE. 0 .AND. F$LOCATE("NEW",P1) .NE. 0 THEN GOTO INIT
$ IF F$LOCATE("CON", P1) .EQ. 0 THEN PROC = "CON"
$ IF F$LOCATE("NEW", P1) .EQ. 0 THEN PROC = "NEW"
$ TAPE := 'F$EXTRACT(0,F$LENGTH(SAVE_DEF) - 1,SAVE_DEF)'.'TAPE_DIR']
$ GOTO SYSGEN
$INIT:
$ type sys$input:
   THIS PROCEDURE WILL BUILD CONSOLE MEDIA FOR THE VAX 8600 FROM MAGTAPE.
IN ORDER FOR THIS PROCEDURE TO OPERATE CORRECTLY, YOU MUST HAVE APPROXIMATELY
50000 BLOCKS OF FREE SPACE. THE PROCEDURE WILL:
        1) CREATE A SUBDIRECTORY
        2) COPY FROM TAPE TO THAT SUBDIRECTORY
        3) COPY FROM THAT SUBDIRECTORY TO THE RL02
        4) DELETE THE SUBDIRECTORY AND ITS FILES
$ WRITE SYS$OUTPUT "**************************************************"
$ WRITE SYS$OUTPUT "*                                                *"
$ WRITE SYS$OUTPUT "*  THE FOLLOWING SHOULD BE VERIFIED.             *"
$ WRITE SYS$OUTPUT "*            1. CSA1: MUST BE DISMOUNTED         *"
$ WRITE SYS$OUTPUT "*            2. SOURCE TAPE MUST BE MOUNTED      *"
$ WRITE SYS$OUTPUT "*                                                *"
$ WRITE SYS$OUTPUT "**************************************************"
$ !
$START:
$ inquire cont "DO YOU WISH TO CONTINUE? [Y/N]"
$ if .not. cont then exit
$ INQUIRE CONT "Is the tape drive on the VAX 8600? [Y]"
$ IF F$LOCATE("Y",CONT) .EQ. 0 THEN GOTO SYSGEN
$ CREATE/DIRECTORY [.'TAPE_DIR']
$ SET DEFAULT [.'TAPE_DIR']
$ TAPE_DEF .EQS. F$ENVIRONMENT("DEFAULT")
$ inquire tape "ENTER MAG TAPE DEVICE (INCLUDING COLON)"
$ if tape .eqs. "" then goto start
$ WRITE SYS$OUTPUT "Copying tape into ''tape_def'.  Files will not be deleted"
$ WRITE SYS$OUTPUT "by this process.  Delete by hand later if you wish."
$ COPY 'TAPE'*.* []
$ DISMOUNT 'TAPE'
$ WRITE SYS$OUTPUT "Log off this system and Log on to the VAX 8600 whose"
$ WRITE SYS$OUTPUT "RL02 you are updating.  Set default to ''save_def'."
$ WRITE SYS$OUTPUT "Then type @RL02 CONTINUE.  This will complete the build."
$ WRITE  SYS$OUTPUT ""
$ set default [-]
$ exit
$ !
```

Figure J-3 Command File to Create a New RL02 Pack
(Sheet 1 of 4)

```
$SYSGEN:
$ mcr sysgen
CONNECT CONSOLE
EXIT
$ create/dir [.'TEMP_DIR']
$ set def [.'TEMP_DIR']
$ disk_def = f$environment("default")
$!
$! 1st build or another build different kind
$!
$ NEXT2:
$ IF NEW_TYPE .EQS. "DIAG" THEN GOTO NEXT5
$ IF NEW_TYPE .EQS. "NODIAG" THEN GOTO NEXT5
$ INQUIRE KIND "DO YOU WANT TO BUILD A DIAG OR NO DIAG PACK? [DIAG/NODIAG]"
$ IF KIND .EQS. "DIAG" THEN GOTO NEXT3
$ IF KIND .NES. "NODIAG" THEN GOTO NEXT2
$ NEXT3:-
$ IF PROC .NES. "CON" .AND. PROC .NES. "NEW" -
                THEN inquire tape "ENTER MAGTAPE DEVICE (INCLUDING COLON)"
$ IF PROC .NES. "NEW" THEN copy 'tape'*.* []
$ IF PROC .NES. "NEW" THEN  WRITE SYS$OUTPUT -
      "Copying files into ''disk_def'.  Files will not be deleted"
$ IF PROC .NES. "NEW" THEN  WRITE SYS$OUTPUT -
      "by this process.  Delete by hand later if you wish."
$ IF KIND .EQS. "DIAG" THEN open/read rlist CONLIST.LIS
$ IF KIND .EQS. "NODIAG" THEN open/read rlist CONLIST_NODIAG.LIS
$ NEXT5:
$ EXCH INIT/CREATE/SEG=31/ALLOC=20480 CONSOL.DSK
$ IF NEW_TYPE .EQS. "DIAG" THEN open/read rlist CONLIST.LIS
$ IF NEW_TYPE .EQS. "NODIAG" THEN open/read rlist CONLIST_NODIAG.LIS
$GET_FILES:
$ OPEN/WRITE TEMP RL02TEMP.COM
$ WRITE TEMP "$ EXCH/NOMESS"
$ WRITE TEMP "MOUNT/VIRTUAL RL02: CONSOL.DSK"
$ WRITE SYS$OUTPUT "CREATING MASTER LIST FOR EXCHANGE. PLEASE WAIT ............."
$ MACFLAG=0
$ readlist:
$ read/end=donelist rlist entry
$ A=F$LOCATE(".",ENTRY)
$ A=A+1
$ EXT=F$EXTRACT(A,3,ENTRY)
$ IF ENTRY .EQS. "EVRLA.DAT" THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "CI780.BIN" THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "KKTMAD.PAK" THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "COPY." THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "BOOT." THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "EXE" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "SAV" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "BPN" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "OBJ" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "DCI" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "DCA" THEN GOTO IMAGE_COPY
$ WRITE TEMP "COPY/LOG ''ENTRY' RL02:*"
$ GOTO READLIST
$ IMAGE_COPY:
$ WRITE TEMP "COPY/LOG ''ENTRY'/TRANS=BLOCK RL02:*/TRANS=BLOCK"
$ goto readlist
$ donelist:
$ close rlist
$ IF NEW_TYPE .EQS. "NODIAG" THEN GOTO NEXT4
$ IF KIND .EQS. "NODIAG" THEN GOTO NEXT4
$ IF MACFLAG .EQS. 1 THEN GOTO NEXT1
```

Figure J-3 Command File to Create a New RL02 Pack (Sheet 2 of 4)

```
$ MACFLAG=1
$ OPEN/READ RLIST MACLIST.LIS
$ GOTO READLIST
$ NEXT1:
$ ON ERROR THEN EXIT
$ WRITE TEMP "COPY/LOG/BOOT=DL RL02:RT11FB.SYS RL02:"
$ CLOSE TEMP
$ @RL02TEMP
$ MAKE_PHYS_PACK:
$ inquire rl1 "ENTER RL02 DEVICE (INCLUDING COLON)"
$ IF RL1 .EQS. "" THEN GOTO MAKE_PHYS_PACK
$ MOUNT/FOREIGN/DATA_CHECK  'RL1'
$ EXCH MOUNT/FOREIGN 'RL1'
$ EXCH INIT 'RL1'
$ COPY/WRITE_CHECK/LOG CONSOL.DSK 'RL1'
$ DISMOUNT 'RL1'
$ !
$ INQUIRE CONT -
"Is this the last VAX 8600 Console to be built on this system at this time? [Y]"
$ WRITE SYS$OUTPUT ""
$ IF F$LOCATE("Y",CONT) .EQ. 0 THEN WRITE SYS$OUTPUT -
        "To rebuild next time, log in and type @RL02 NEW."
$ WRITE SYS$OUTPUT ""
$ IF F$LOCATE("Y",CONT) .EQ. 0 THEN GOTO DEL_RL02_BUILD_AREA
$ INQUIRE NEW_TYPE:
$ COUNTER=COUNTER+1
$ IF COUNTER .LE. 1 THEN LAST_TYPE = KIND
$ IF COUNTER .GT. 1 THEN LAST_TYPE = NEW_TYPE
$ IF F$LOCATE("N",CONT) .EQ. 0 THEN INQUIRE NEW_TYPE -
            "WHAT TYPE DO YOU WISH TO CREATE [DIAG/NODIAG]?"
$ IF NEW_TYPE .EQS. LAST_TYPE THEN GOTO MAKE_PHYS_PACK
$ GOTO RESET
$!
$! Deleting tape area function has been disabled for quicker rebuilding of packs
$!
$!DEL_TAPE_AREA:
$! write sys$output "CURRENT (DELETE) OPERATION TAKES SOME TIME-PLEASE WAIT"
$! SET DEFAULT 'TAPE'
$! DELEE *.*;*
$! set def [-]
$! set prot 'TAPE_DIR'.dir/prot=(OWNER:D,G:D,W:D)
$! delEE 'TAPE_DIR'.dir;
$DEL_RL02_BUILD_AREA:
$ set def 'save_def'
$ set def [.'TEMP_DIR']
$ delEE CONSOL.DSK;*
$ set def [-]
$! DELEE RL02.COM;
$! set prot 'TEMP_DIR'.dir/prot=(OWNER:D,G:D,W:D)
$! delEE 'TEMP_DIR'.dir;
$ WRITE SYS$OUTPUT ""
$ WRITE SYS$OUTPUT "Console has been updated. If you are on an 8600 you can "
$ WRITE SYS$OUTPUT "shut down the system using site specific procedures and "
$ WRITE SYS$OUTPUT "reboot the system with the new version of the console."
$ WRITE SYS$OUTPUT "Exiting..."
$ WRITE SYS$OUTPUT ""
$ SET PROCESS/PRIV=(NOCMK,NOVOLPRO)
$ EXIT
$RESET:
```

Figure J-3 Command File to Create a New RL02 Pack (Sheet 3 of 4)

```
$ set def 'save_def'
$ set def [.'TEMP_DIR']
$ DEL CONSOL.DSK;
$ GOTO NEXT2
$ EXIT:
$ CLOSE RLIST
$ CLOSE TEMP
$ SET PROCESS/PRIV=(NOCMK,NOVOLPRO)
$ SET DEF 'SAVE_DEF'
$ exit
$ NEXT4:
$ WRITE TEMP "COPY/LOG CI780.BIN/TRANS=BLOCK RL02:*/TRANS=BLOCK"
$ GOTO NEXT1
$
```

Figure J-3 Command File to Create a New RL02 Pack (Sheet 4 or 4)

3. Dismount the RL02 tape by typing the following command.

```
$ DISMOUNT CSA1:
```

If the response to this command is "no such device," don't panic; the procedure will connect the console automatically.

4. Mount the magtape with the following command:

```
$ MOUNT HSC004$MUA0:     (TA78 on an HSC50)
_Label: VAXPAX
_Log name: <CR>
```

5. Copy the RL02 command file from tape to disk with the following command.

```
$ COPY HSC004$MUA0:RL02.COM *
```

CAUTION

Be sure to remove the console pack from the RL02 drive and replace it with a formatted scratch pack before proceeding. Failure to do this will result in the destruction of the console pack's contents, which will ultimately crash the system and prevent restoring normal operation until a new pack is available.

6. Finally, execute the command file by typing the following command.

```
$ @RL02
```

The command file will now take over and execute the command sequences required to build the new RL02 disk pack. It will take approximately one hour to complete the entire build process.

After the command file terminates, there are two files on the disk pack that may need to be modified to permit booting the customer's system with the new pack. Use the following command procedure to update the files DEFBOO.COM and CIBOO.COM to reflect the customer's system node and load device drive numbers.

```
$ EXCHANGE
EXCHANGE> COPY CSA1:DEFBOO.COM *
EXCHANGE> EXIT
$ EDIT DEFBOO.COM

    (edit the file to reflect the
    specific customer configuration)

$ EXCHANGE
EXCHANGE> COPY DEFBOO.COM CSA1:*
EXCHANGE> EXIT
$
```

As a final check, the system should be shut down and rebooted to test whether the new pack performs correctly.

J.4.2 Updating an RL02 Pack

The following procedure outlines the process for updating an existing disk pack. It is similar to the build process just described except that a different tape and a different command file are used. Figure J-4 shows a listing of the command file.

1. Load the RL02 update tape onto the magtape unit (SDC Part No. BB-ZZKHA-JS or BB-FF58A-ME V1.0).

2. Log into the field service account using the username FIELD and the password assigned by the system manager.

3. Mount the magtape with the following command.

```
$ MOUNT HSC004$MUA0:     (TA78 on an HSC50)
_Label: VAXPAX
_Log name: <CR>
```

4. Copy the RL02 command file from tape to disk with the following command.

```
$ COPY HSC004$MUA0:RLUPDATE.COM *
```

5. Finally, execute the command file by typing the following command.

```
$ @RLUPDATE
```

The command file will now take over and execute the command sequences required to update the RL02 disk pack. As a final check, the system should be shut down and rebooted to test whether the updated pack performs correctly.

```
$ ! RLUPDATE.COM  version V1.0
$ UPDATE_ID := "BB-FF58A-ME"
$ TEMP_DIR := "RL2UPDATE"
$ TAPE_DIR := "RL2TAPE"
$ PROC = ""
$ !
$ ! This command procedure updates console media with new microcode and
$ ! console software.  Written 4-APR-1985 11:07:22
$ !
$ on control_y then goto EXIT
$ SET PROCESS/PRIV=(CMK,VOLPRO)
$ save_def =  f$environment("default")
$ IF  P1 .EQS. "" THEN GOTO INIT
$ IF F$LOCATE("CON", P1) .NE. 0 THEN GOTO INIT
$ PROC = "CON"
$ TAPE := 'F$EXTRACT(0,F$LENGTH(SAVE_DEF) - 1,SAVE_DEF)'.'TAPE_DIR']
$ GOTO SYSGEN
$INIT:
$ type sys$input:
This procedure updates files on the console pack.  Certain files will be
be replaced.  The procedure will take only a few minutes.  Please ensure
that the tape is mounted.
$ !
$START:
$ inquire cont "DO YOU WISH TO CONTINUE? (Y/N)"
$ if .not. cont then exit
$ INQUIRE CONT "Is the tape drive on the VAX 8600? [Y]"
$ IF F$LOCATE("Y",CONT) .EQ. 0 THEN GOTO SYSGEN
$ CREATE/DIRECTORY [.'TAPE_DIR']
$ SET DEFAULT [.'TAPE_DIR']
$ inquire tape "ENTER MAG TAPE DEVICE (INCLUDING COLON)"
$ if tape .eqs. "" then goto start
$ WRITE SYS$OUTPUT "Copying tape..."
$ COPY 'TAPE'*.* []
$ DISMOUNT 'TAPE'
$ WRITE SYS$OUTPUT "Log off this system and Log on to the VAX 8600 whose"
$ WRITE SYS$OUTPUT "RL02 you are updating.  Set default to ''save_def'.
$ WRITE SYS$OUTPUT "Then type @RLUPDATE CONTINUE."
$WRITE  SYS$OUTPUT "This will complete the installation."
$ set default [-]
$ exit
```

Figure J-4 File to Update an Existing RL02 Pack (Sheet 1 of 3)

```
$SYSGEN:
$ mcr sysgen
CONNECT CONSOLE
EXIT
$ create/dir [.'TEMP_DIR']
$ set def [.'TEMP_DIR']
$ inquire rl1 "ENTER RL02 DEVICE (INCLUDING COLON)"
$ RL1 := "CSA1:"
$ MOUNT/FOREIGN/DATA_CHECK  'RL1'
$ EXCH MOUNT/FOREIGN 'RL1'
$ EXCH COPY 'RL1'NOTICE.TXT []
$ !
$ ! This section determines out which version of the console that we have.
$ !
$check_version:
$ !TYPE NOTICE.TXT
$ OPEN/READ/ERR=BAD_VERSION V_NOTICE NOTICE.TXT
$GET_LINE:
$ READ/END=BAD_VERSION V_NOTICE LINE
$ IF F$LOCATE("VAX 8600 CONSOLE",LINE) .EQS. F$LENGTH(LINE) THEN GOTO GET_LINE
$ READ/END=BAD_VERSION V_NOTICE LINE
$ CON_VER := "V
$ VERSION = 30
$ KIND = "DIAGS"
$ IF F$LOCATE("BC-T989C-DE V3.0",LINE) .NES. F$LENGTH(LINE) THEN CON_VER := "D30
"
$ KIND = "NODIAGS"
$ IF F$LOCATE("BC-T987C-ME V3.0",LINE) .NES. F$LENGTH(LINE) THEN CON_VER := "C30
"
$ VERSION = 20
$ IF F$LOCATE("BC-T987B-DE V2.0",LINE) .NES. F$LENGTH(LINE) THEN CON_VER := "C20
"
$ KIND = "DIAGS"
$ IF F$LOCATE("BC-T989B-DE V2.0",LINE) .NES. F$LENGTH(LINE) THEN CON_VER := "D20
"
$ IF CON_VER .EQS. "" THEN GOTO BAD_VERSION
$ !
$ !  Check if already updated
$ !
$ READ/END=BAD_VERSION V_NOTICE LINE
$ CLOSE V_NOTICE                          ! Finished reading notice
$ IF F$LOCATE(UPDATE_ID,LINE) .EQS. F$LENGTH(LINE) THEN GOTO GET_FILES
$ WRITE SYS$OUTPUT "The console pack already has this release on it."
$ WRITE SYS$OUTPUT "Exiting..."
$ GOTO DELETE
$ !
$ ! Start update
$ !
$GET_FILES:
$ IF F$PROC .NES. "CON" THEN inquire tape "ENTER MAG TAPE DEVICE (INCLUDING COLON)
"
$ copy 'tape'CONLIST_'CON_VER'.LIS *.*
$ open/read rlist CONLIST_'CON_VER'.LIS
$ OPEN/WRITE TEMP RL02TEMP.COM
$ WRITE TEMP "$ EXCH/NOMESS"
$ WRITE SYS$OUTPUT "CREATING MASTER LIST FOR EXCHANGE. PLEASE WAIT ............
$ readlist:
$ read/end=donelist rlist entry
$ ENTRY = ENTRY - "_NODIAG"
$ A=F$LOCATE(".",ENTRY)
$ NAME = F$EXTRACT(0,A,ENTRY)
$ A=A+1
$ EXT=F$EXTRACT(A,3,ENTRY)
```

Figure J-4 File to Update an Existing RL02 Pack (Sheet 2 of 3)

```
$ COPY 'TAPE''NAME' 'CON_VER'.'EXT' 'ENTRY'                    ! COPY TO DISK
$ IF ENTRY .EQS. "EVRLA.DAT" THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "CI780.BIN" THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "KKTMAD.PAK" THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "COPY." THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "BOOT." THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "COPY.COM" THEN GOTO IMAGE_COPY
$ IF ENTRY .EQS. "ANYBOO.COM" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "EXE" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "SAV" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "BPN" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "OBJ" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "DCI" THEN GOTO IMAGE_COPY
$ IF EXT .EQS. "DCA" THEN GOTO IMAGE_COPY
$ WRITE TEMP "COPY/REPLACE/LOG ''ENTRY' ''RL1'*"
$ GOTO READLIST
$ IMAGE_COPY:
$ WRITE TEMP "COPY/REPLACE/LOG ''ENTRY'/TRANS=BLOCK ''RL1'*/TRANS=BLOCK"
$ goto readlist
$ donelist:
$ close rlist
$ ON ERROR THEN EXIT
$ CLOSE TEMP
$ IF PROC .NES. "CON" THEN DISMOUNT 'TAPE'
$ @RL02TEMP
$ WRITE SYS$OUTPUT "Console has been updated."
$ WRITE SYS$OUTPUT "Shut down the system using site specific procedures and rebo
ot the system."
$ WRITE SYS$OUTPUT "Exiting..."
$DELETE:
$ delEE *.*;*
$ set def [-]
$ set prot 'TEMP_DIR'.dir/prot=(owner:d,G:D,W:D)
$ delEE 'TEMP_DIR'.dir;
$ IF PROC .NES. "CON" THEN GOTO  RESET
$ inquire CONT "Is this the last VAX 8600 Console to be updated? [Y]"
$ IF CONT .EQS. "" THEN GOTO DEL_TAPE
$ IF F$LOCATE("N",CONT) .EQ. 0 THEN GOTO RESET
$DEL_TAPE:
$ SET DEFAULT 'TAPE'
$ DELEE *.*;*
$ set def [-]
$ set prot 'TAPE_DIR'.dir/prot=(owner:d,G:D,W:D)
$ delEE 'TAPE_DIR'.dir;
$RESET:
$ SET PROCESS/PRIV=(NOCMK,NOVOLPRO)
$ DELEE RLUPDATE.COM;
$ exit
$BAD_VERSION:
$ CLOSE V_NOTICE                          ! Finished reading notice
$ WRITE SYS$OUTPUT "This pack is INVALID for this update."
$ WRITE SYS$OUTPUT "To apply this update you must have a V3.0 or V2.0 pack."
$ WRITE SYS$OUTPUT "If you have a later version you cannot apply this update."
$ WRITE SYS$OUTPUT "Otherwise you must rebuild the pack before appplying the upd
ate."
$ GOTO DELETE
$EXIT:
$ close rlist
$ CLOSE TEMP
$ CLOSE V_NOTICE
$ SET PROCESS/PRIV=(NOCMK,NOVOLPRO)
$ exit
$
```

Figure J-4 File to Update an Existing RL02 Pack
(Sheet 3 of 3)

APPENDIX K
CONSOLE SOFTWARE ERROR MESSAGES

K.1 INTRODUCTION

This appendix describes console messages for Version 7.0 through Version 9.0 of the console software. It begins with a description of the console message format and is followed by a list of tables. The tables identify both the program or routine running at the time the message was printed and the type of message (error, warning, etc.). Finally, there are tables which list, in alphabetical order, most messages and what they mean. In many cases, the message descriptions include a statement explaining how the user might response to the message.

K.2 CONSOLE MESSAGE FORMAT

Console messages consist of four parts, as shown in the following example.

```
?CSM-F-ACCVIO  ACV Condition
```

1. Part 1: The Calling Routine -- begins with a question mark and includes the three-character routine name that initiated the message. For example, in the above message, the Console Support Microcode (CSM) initiated the message print out. The following is a list of console routines that display messages.

   a. CSM -- Console Support Microcode
   b. DCN -- General Console
   c. DCP -- Diagnostic Console Program
   d. ECR -- Error Correction Routine
   e. EMM -- Environmental Monitor Module
   f. HEX -- Hexadecimal Debugger
   g. MCP -- Macro Control Program

2. Part 2: Severity Code -- Consists of a letter (E, F, W, or I) preceded and followed by a hyphen. The letter indicates the general severity of the condition, as summarized below.

   a. E (Error) -- Indicates that the routine printing the message is responding to a device or hardware error of some sort.

b. F (Fatal) -- Indicates that the routine printing the message either detected an internal consistency error (e.g., bad parameters passed to a routine) or a totally unexpected or unserviceable error (e.g., errors from RT-11).

c. W (Warning) -- Indicates that the routine printing the message was able to complete some, but not all, of the operation. Check the result before proceeding.

d. I (Information) -- Indicates that the routine printing the message completed the operation successfully and the user should be aware of the information that follows the message header.

3. Part 3: Message ID -- Is a six-letter mnemonic that identifies the message.

4. Part 4: Message Text -- Is the text of the message. It contains a line of text explaining the reason for the message.

K.3 CONSOLE MESSAGE TABLES

The console messages have been further defined and organized into tables, as shown in Table K-1.

IMPORTANT

Refer to the Console Software Specification for further information and specific details on console operation. Also, keep in mind that these message are subject to console revisions.

Table K-1 Console Error Messages Tables

Table	Message Descriptions
K-2	Console Support Microcode (CSM) Fatal Messages
K-3	General Console (DCN) Error Messages
K-4	General Console (DCN) Fatal Messages
K-5	General Console (DCN) Information Messages
K-6	General Console (DCN) Warning Messages
K-7	Diagnostic Console (DCP) Error Messages
K-8	Diagnostic Console (DCP) Fatal Message
K-9	Diagnostic Console (DCP) Information Messages
K-10	Diagnostic Console (DCP) Warning Messages
K-11	Error Correction Routine (ECR) Error Messages
K-12	Environmental Monitor Module (EMM) Error Messages
K-13	Environmental Monitor Module (EMM) Fatal Messages
K-14	Hexadecimal Debugger (HEX) Warning Messages
K-15	Macro Control Program (MCP) Error Messages
K-16	Macro Control Program (MCP) Fatal Messages
K-17	Macro Control Program (MCP) Information Messages
K-18	Macro Control Program (MCP) Warning Messages

Table K-2 Console Support Microcode (CSM) Fatal Messages

Header	Message Description
?CSM-F-ACCVIO	ACV Condition -- The console has no access to any of the data requested. By raising the PSL CUR_MODE Field Encoding and issuing the command again, it may be possible to get a successful response. Also, try using the physical address with memory management turned off. If the problem persists, there may be a fault in the Access RAM.
?CSM-F-BADIPR	Bad IPR Number -- The IPR number argument is currently unassigned.
?CSM-F-BDCKSM	Bad Packet Checksum -- The computed checksum of the RBUF packets does not match the console checksum. This could be a dual-port RAM, CBus, or EBox microcode problem. Try reinitializing the CPU. If that doesn't work, run the MHC diagnostic.
?CSM-F-BDVADR	Bad Virtual Address -- The virtual address either has no translation or a bad translation.
?CSM-F-CF64KB	Can't Find 64 KB -- CSM was unable to find a good 64 KB section of memory. There is a good chance that the MBox/Array either has a serious problem or it has not been initialized properly.
?CSM-F-CFDRPB	Can't Find RPB -- CSM was unable to find the Restart Parameter Block (RPB) in memory. Either that section of memory was overwritten, the BBU power to support the Array Refresh was switched off, or the 10-minute (BBU) Array Refresh limit expired. In any case, a cold restart is necessary. Note: This is a normal response if, after a power-up, the Restart Control switch is in the RESTART BOOT or RESTART HALT position.
?CSM-F-HERABT	Hardware Error Abort -- During the FIND 64KB operation, the error handling microcode detected a hardware-related error causing the FIND 64KB operation to abort. The machine check stack frame built by the EHM (ESC: <17:2F> should provide information about the error). If the error persists, run the microdiagnostics.

Table K-3 General Console (DCN) Error Messages

Header	Message Description
?DCN-E-BA11PF	BA11 Power Failure -- The console detected a BA11 power failure when it read the status of the RL02. Check power at the BA11.
?DCN-E-CSPERR	"RAM ID" Control Store Parity Error -- While in console I/O mode, a CS/DRAM parity error was detected in the RAM specified. Use the VERIFY command (in debug context) to determine the good/bad data and take corrective action.
?DCN-E-DVCERR	Read/Write Disk Error -- A Read/Write error was detected while reading or writing a file on the RL02. Examine the RLV12 control and status registers for specific details. If the error is unique to a specific file, reinstall the file from tape or, if necessary, replace the pack.
?DCN-E-FULERR	"Device Name" Device Full -- The RL02 is full. This may be due to fragmented files. A file may need to be deleted or the pack may need to be rebuilt.

Table K-4 General Console (DCN) Fatal Messages

Header	Message Description
?DCN-F-ACCERR	"Filename" Illegal Access -- The user does not have the necessary privileges to access the specified file (e.g., the file may be read only protected).
?DCN-F-CHNERR	Read/Write IO Channel Invalid -- (Software Problem).
?DCN-F-EIARG	Invalid Argument on "aaaaaa" Call -- (Software Problem).
?DCN-F-EOFERR	Read/Write End_Of_File -- The console software encountered an unexpected EOF while reading a file. There is something wrong with the file format. Restore the file from tape.
?DCN-F-INVPD	Invalid Parameter Detected -- (Software Problem).
?DCN-F-TIMEXP	Timer Either Expired or Not Set -- (Software Problem).
?DCN-F-TIMMAX	Maximum Number of Timers Set -- (Software Problem).

Table K-5 General Console (DCN) Information Messages

Header	Message Description
?DCN-I-CCABRT	<CTRL/C> Abort -- The user typed <CTRL/C> which interrupted the command in progress.
?DCN-I-CLKNLK	CPU Clock Frequency Unstable (Not Locked) -- The system clock is more than 60 ns out phase with the 1 MHz Reference. This message will only occur for Rev C5 clock modules. Either change to a more stable clock frequency or use the SET CLOCK FREQ QUIET command to suppress the message.
?DCN-I-CSLFAL	"RAM File" Already Loaded -- The file specified has already been loaded in the control store or Dispatch RAM.
?DCN-I-VERINC	"RAM File" Vn Incorrect, Should Have Vn -- The CS/DRAM file just loaded is not compatable with the revision level of the system (as specified in the RL02 file, VENUS.REV). This message will also be displayed if specifying an incorrect filename. For example, typing the console command LOAD/ECS CSM040 will result in this message because CSM040 is a CSM overlay and not considered part of the main EBox microcode load file. The console will still load the file, however, but beware of erroneous responses.
?DCN-I-VFYERC	"RAM File" Verify Found "n" Microwords Bad -- During CS/DRAM verification (using the VERIFY command), "n" microwords were found to have errors. Note: the maximum error count displayed will be 256. If the errors persist, then run the MHC diagnostic to isolate the fault and then take corrective action.*
?DCN-I-VFYERR	"RAM File" Verification Fails -- If the console is in debug mode during CS/DRAM verification, this message will be displayed followed by the good/bad data for the first 256 errors.*

*The VERIFY command must be executed in debug context (>>>> VERIFY or DC>> VERIFY) in order to display the good/bad data. Otherwise, only the total number of errors will be displayed.

Table K-6 General Console (DCN) Warning Messages

Header	Message Description
?DCN-W-CLKRER	Command Invalid with CPU Clock Running -- The CPU clock must be stopped (i.e., STOP CPU) in order for the command to execute properly.
?DCN-W-COMABT	Command Procedure Aborted -- The abort flag was set and an error was detected while executing a command file. For troubleshooting purposes, this condition can be bypassed (and force the console to execute the command file anyway) by using the SET ABORT OFF command. See the ?DCN-W-COMCWE message next.
?DCN-W-COMCWE	Command Procedure Completed with Errors -- The Abort on Error switch was not set when an error was detected during command file execution. The console continued to execute the file, but the results are unpredictable.
?DCN-W-COMERR	Nesting Depth Exceeded -- The maximum number of nested command files (4) has been exceeded.
?DCN-W-CSLERR	"RAM File" CS Load/Verify Failure (File Bad) -- The contents of the RAM file specified does not match the format of the CS/DRAM being loaded. Entering a command such as "LOAD/ECS MCF.BPN<cr>" would cause the message because the MCF RAM data format differs from that of the EBox CS.
?DCN-W-EIRID	Reg_ID Undefined -- The register ID that the user supplied as a command argument does not exist. Use the SHOW REGISTERS command to display all defined registers and register IDs.
?DCN-W-EISID	SDB_ID Not Found in CAD Tables -- The SDB_ID that the user supplied as a command argument does not exist in the CAD tables.*
?DCN-W-EUREG	Register Name Undefined -- The register name that the user supplied as a command argument has not been defined. Use the SHOW REGISTERS command to display all defined registers and register IDs.
?DCN-W-EUSIG	Signal Name Not Found in CAD Tables -- The SDB signal name that the user supplied as a command argument does not exist in the CAD tables.*

*Check the CONFIG.DAT file to be sure it contains the right CAD filenames for the revision of the machine. Use the SHOW CONFIG/ASCII command.

Table K-6 General Console (DCN) Warning Messages (Cont.)

Header	Message Description
?DCN-W-EUSYM	Symbol Name Not Found in CAD Tables -- The V$ symbol that the user supplied as a command argument does not exist in the CAD tables.*
?DCN-W-FILNAM	The "aaaaaa" File Not Found -- The file specified does not exist on the RL02.
?DCN-W-INVCLK	Command Invalid for This Version Clock Module
?DCN-W-INVRMT	Command Invalid from Remote Terminal -- The last command entered cannot be executed from a remote terminal (e.g., changing the baud rate).
?DCN-W-INVXTL	Unassigned Crystal Mnemonic
?DCN-W-OPNERR	"Device Name" Too Many Files Opened -- The maximum number of files that can be opened at the same time (4) has been exceeded.
?DCN-W-PARSER	Ambiguous Command -- Two or more commands match the command abbreviation. Be more specific.
?DCN-W-PARSER	Invalid Command -- Either the command was entered in the wrong context or the command does not exist. Check for proper context and spelling.
?DCN-W-USESTP	Use INIT/POWER Command to Initialize EMM

*Check the CONFIG.DAT file to be sure it contains the right CAD filenames for the revision of the machine. Use the SHOW CONFIG/ASCII command.

Table K-7 Diagnostic Console (DCP) Error Messages

Header	Message Description
?DCP-E-ALIVEE	Invalid DSM Alive Byte -- This message should only appear after a START command has been issued to a diagnostic. It means that the diagnostic should have finished running its current test, but has not. The microcode may be hung, or the test may have gotten into an infinite loop. In either case, it is a lot like a DSM-DCP communication failure and needs to have the same sort of information collected. The user needs to know what caused the condition so that it can be fixed. Do the following. 1. Enable HARDCOPY if a hardcopy terminal is available. 2. Type STOP CPU. 3. Type MIC. This will cause the current microsequencer PCs to be typed on the terminal. 4. Press the space bar 10 more times. This causes a sequence of microsequencer PCs to be typed. This helps us to find out what the CPU thinks it's doing. 5. Press <RETURN>. This gets the user out of MIC mode. 6. Type RESET. 7. Type START CPU. 8. Type EXAMINE/ESCRATCH 70. 9. Type EXAMINE/ESCRATCH 73. 10. Type SHOW DATA. 11. Now reexecute the command file for the microdiagnostic that got the error. Type @EDK--. 12. Type START to see if the diagnostic fails consistently.

Table K-7 Diagnostic Console (DCP) Error Messages (Cont.)

Header	Message Description
	13. If the microdiagnostic hangs again, type DIAG and then reexecute the command file one more time. 14. Save all of this data and include it with a problem report.
?DCP-E-BADCHK	Bad Chksum -- After six retries, the computed checksum still did not match the checksum sent with the DSM message packet. This could be a dual-port RAM, CBus or EBox microcode problem. Try reinitializing the CPU. If that doesn't work, run the MHC diagnostic.
?DCP-E-CONDER	Invalid Conditional Statement (failed to isolate)*
?DCP-E-ILLDSM	Invalid DSM Function Code*
?DCP-E-INVDAT	Invalid SDB data, Failed to Isolate*
?DCP-E-INVDCB	Invalid .DCB Isolation File*
?DCP-E-INVDCI	Invalid .DCI Isolation File*
?DCP-E-INVDID	Invalid ID (Failed to Isolate)*
?DCP-E-NOANSD	DSM-DCP Communication Failure -- This message means the EBox microsequencer has stopped listening to the console. The cause of this condition can be investigated. This could happen because of programming faults, because the hardware is not initialized properly, or because the hardware is broken. This message can occur at almost any time when the user is at the console terminal. No matter what the reason, the cause of the condition must be determined so that it can be fixed, or another test must be written to catch the fault earlier in the testing sequence. Follow the procedure outlined under ?DCP-E-ALIVEE.

*This is most likely a software problem (DSM, DC, or the isolation file). Ensure that the CPU revision level matches the revision level in VENUS.REV on the RL02.

Table K-7 Diagnostic Console (DCP) Error Messages (Cont.)

Header	Message Description
?DCP-E-UMICTP	Unexpected Micro Trap at Vector XX -- The user should get this message only after starting a microdiagnostic. It means there is something wrong in the hardware that is causing microtraps in the EBox the current test has not requested or has tried to force. A fault is in the machine that should have been caught by a previous diagnostic, or perhaps the machine has not been initialized properly. Do the following. 1. Enable HARDCOPY if a hardcopy terminal is available. 2. Type SHOW SWITCHES. 3. Type SHOW DATA. 4. Type EXAMINE/WBUS 6. 5. Type EXAMINE/WBUS 7. 6. Type EXAMINE/WBUS 9. 7. Type EXAMINE/WBUS 11. 8. Type EXAMINE/WBUS 12. 9. Type EXAMINE/WBUS 13. 10. Type START. Typing START and causing the tests to be run again will show the user if the problem was a one-time event or if the test microcode has an initialization or setup problem.

Table K-8 Diagnostic Console (DCP) Fatal Message

Header	Message Description
?DCP-F-LDFAIL	DSM Load Failure -- DCP was either unable to load DSM or unable to start DSM. Reinitalize the CPU and DC. If the problem persists, run the MHC Diagnostic.

Table K-9 Diagnostic Console (DCP) Information Messages

Header	Message Description
?DCP-I-BADMWD	Stop on Umark Bit -- A micromark bit was detected while running diagnostics.
?DCP-I-FAPAUS	Fault Detected, Pausing -- The user set the /FAULT:PAUSE switch and the diagnostic detected a fault.
?DCP-I-PAUSEI	Pausing -- The user typed <CTRL/P>.
?DCP-I-SWDATA	Set Data Command Not Invoked -- The user must first enter a Set Data command before the SHOW DATA command.

Table K-10 Diagnostic Console (DCP) Warning Messages

Header	Message Description
?DCP-W-CLKSTP	Clock Not Running -- The CPU clock has stopped and must be started so the command can execute properly.
?DCP-W-DIASTA	Diags Not Started -- The user issued a CONTINUE or STEP command before starting the diagnostic.
?DCP-W-EOTBLE	End of Set Data Table Space -- The user has issued more than the maximum (16) SET DATA commands.
?DCP-W-LMTRUN	Limit of Set Data ASCII Text, Truncate -- The user has exceeded the maximum (30) ASCII characters in a string.
?DCP-W-FAILIS	Isolation Algorithm Failed to Isolate -- There is a problem in the isolation file and the DCP is unable to execute the isolation algorithm.
?DCP-W-ISOEOF	End of Isolation File Encountered -- DCP unexpectedly encountered an EOF in an isolation file. There is something wrong with the isolation file.

Table K-11 Error Correction Routine (ECR) Error Messages

Header	Message Description
?ECR-E-INTERR	CSPE Interrupt, Code Invalid (0) -- The console was interrupted to correct a CS/DRAM parity error; when the interrupt code was read, it was zero which is unassigned. This will result in a KAF. If the problem persists, the console module or the EBE module is the most likely cause (see CL07-CL09 and EBE3).
?ECR-E-MBTERR	"RAM ID" Multi-Bit-Error, Uncorrectable -- More than one bit was found to be in error in the CS/DRAM specified. This will result in a KAF.*
?ECR-E-MUNREC	MCS MBox Not Recoverable -- MBox CS parity errors are correctable but not recoverable. This will result in a KAF.
?ECR-E-NOECCD	"RAM ID" No ECC Data in Table -- The ECC data needed to correct the CS/DRAM parity error was not specified in the corresponding ECC Table. This will result in a KAF.
?ECR-E-PCFAIL	"RAM ID" Correction Attempted and Failed -- The console was unable to correct the CS/DRAM parity error specified after five attempts. This will result in a KAF.*
?ECR-E-SYNGTR	"RAM ID" Syndrome > Ram_Size, Uncorrectable -- The syndrome generated by the CS/DRAM parity error specified indicates a nonexistent bit was at fault (e.g., bit 22 in the IBox DRAM which is only 20 bits wide). This will result in a KAF.*
?ECR-E-SYNZRO	"RAM ID" Syndrome = 0, Uncorrectable -- Most likely the CPU detected a transient error associated with the CS/DRAM specified. That is, a parity error was detected and latched in the appropriate CS/DRAM logic, but when the data latch was read (or the CS RAM re-read) the data was ok. There is, however, a very low possibility that double bit error occurred, producing a real zero syndrome. Currently this error will result in a KAF. In the future, however, the console will report this as a transient error and the console will take no further action.

*The console software is being modified to report the following for uncorrectable CS/DRAM parity errors: "Good Data/Bad Data, Syndrome, and Address."

Table K-11 Error Correction Routine (ECR) Error Messages (Cont.)

Header	Message Description
?ECR-E-UPCERR	"RAM ID" Can't Read Box Address -- The CS/DRAM correction routine was unable to read the microaddress associated with the error. This will result in a KAF. This is either an SDB failure or a failure in the addressing logic associated with the CS/DRAM specified.

Table K-12 Environmental Monitor Module (EMM) Error Messages

Header	Message Description
?EMM-E-EMMACK	No TRANSPORT_ACK from EMM
?EMM-E-EMMACL	EMM_LAT_AC_LO failed to deassert
?EMM-E-EMMAFD	REGULATOR_A_OK failed to deassert
?EMM-E-EMMAFF	AIR_FLOW fault failed to deassert
?EMM-E-EMMANO	MODULE_A not ok on EMM powerup
?EMM-E-EMMAOK	REGULATOR_A is not ok
?EMM-E-EMMBFD	REGULATOR_B_OK failed to deassert
?EMM-E-EMMBOK	REGULATOR_B is not ok
?EMM-E-EMMBUF	EMM has no protocol buffer
?EMM-E-EMMCFD	REGULATOR_C_OK failed to deassert
?EMM-E-EMMCOK	REGULATOR_C is not ok
?EMM-E-EMMCOL	Data errors (collisions) on EMM bus
?EMM-E-EMMDCL	EMM_LAT_DC_LO failed to deassert

Table K-12 Environmental Monitor Module (EMM) Error Messages (Cont.)

Header	Message Description
?EMM-E-EMMDED	Console/EMM communication temporarily suspended -- Communication with the EMM has been suspended for 30 seconds due to excessive errors with the EMM or the EMM communication link (these errors are reported prior to the message stating that communications is suspended). While EMM communication is suspended, the ALERT LED will flash double-time (1/4 second on, 1/4 second off). After 30 seconds, the console tries to reestablish communication with the EMM. It will only report the error once (but the LED will continue to flash until the link is reestablished).
?EMM-E-EMMDFD	REGULATOR_D_OK failed to deassert
?EMM-E-EMMDOK	REGULATOR_D is not ok
?EMM-E-EMMEFD	REGULATOR_E_OK failed to deassert
?EMM-E-EMMEOK	REGULATOR_E is not ok
?EMM-E-EMMFFD	REGULATOR_F_OK failed to deassert
?EMM-E-EMMFOK	REGULATOR_F is not ok
?EMM-E-EMMHFD	REGULATOR_H_OK failed to deassert
?EMM-E-EMMHOK	REGULATOR_H is not ok
?EMM-E-EMMI55	EMM 5.5 interrupt broken -- Replace EMM
?EMM-E-EMMI65	EMM encountered unexpected 6.5 interrupt
?EMM-E-EMMINV	Invalid exception code from EMM
?EMM-E-EMMJFD	REGULATOR_J_OK failed to deassert
?EMM-E-EMMJOK	REGULATOR_J is not ok
?EMM-E-EMMKAC	MOD_K_AC_LO is asserted
?EMM-E-EMMKOK	REGULATOR_K is not ok
?EMM-E-EMMLAC	MOD_L_AC_LO is asserted
?EMM-E-EMMLOK	REGULATOR_L is not ok

Table K-12 Environmental Monitor Module (EMM) Error Messages (Cont.)

Header	Message Description
?EMM-E-EMMNEG	EMM rejected command request due to present conditions
?EMM-E-EMMPER	EMM encountered RAM parity error
?EMM-E-EMMRES	No response from EMM
?EMM-E-EMMRST	EMM encountered restart 1 instruction
?EMM-E-EMMTRP	EMM encountered unexpected trap interrupt
?EMM-E-EMMUNK	EMM encountered unexpected trap to PC 0
?EMM-E-EMMURC	Unknown restart code in RTDREG

Table K-13 Environmental Monitor Module (EMM) Fatal Messages

Header	Message Description
?EMM-F-EMMMTL	EMM protocol message too long -- (Software Problem).
?EMM-F-EMMTIP	EMM transmission already in progress -- (Software Problem).
?EMM-F-EMMXTO	Console-to-EMM protocol message transmit timeout -- Either the switch controlling BBU power to the TOY is off or there is a TOY chip problem on the console module.

Table K-14 Hexadecimal Debugger (HEX) Warning Messages

Header	Message Description
?HEX-W-ADRFOR	Address field out of range -- The microaddress specified exceeds the size of the CS or DRAM.*
?HEX-W-ANFERR	Microaddress not found in file -- The microaddress specified does not exist in the RAM file (e.g., CSM is not included as part of the EBox.BPN file).*
?HEX-W-CLKRUN	Command invalid with CPU clock running -- The CPU clock must be stopped (STOP CPU) before the command can be executed properly.*
?HEX-W-DATFOR	Data field out of range -- The data specified exceeds the size of the CS or DRAM.
?HEX-W-INCHNO	Invalid channel number -- The SDB control channel specified does not exist.*
?HEX-W-INVRPT	Invalid repeat function -- The command cannot be executed with the Repeat function.*
?HEX-W-INVSID	Invalid ID or name -- The last SDB ID or SDB name entered does not exist. Make sure the correct CDF files are loaded for the CPU revision level (SHOW CONFIG/ASCII).*

*Use the console software HELP command or refer to the Console Software Specification for more details on command usage.

Table K-15 Macro Control Program (MCP) Error Messages

Header	Message Description
?MCP-E-C40ICE	CSM040 hung during CPU initialization -- CSM was unable to initialize the IPRs properly. This indicates that there is an interaction problem between the EBox and one of the other boxes. If the problem persists, run MHC and the microdiagnostics to isolate the fault. If all else fails, try single stepping through the CSM overlay.
?MCP-E-CBADCK	CSM sent bad checksum -- The computed checksum did not match the checksum sent with the last CSM message. This could be a dual-port RAM, CBus, or EBox microcode problem. Try reinitializing the CPU. If that doesn't work, run the MHC diagnostic.
?MCP-E-CSMLOP	CSMs console loop not running -- The EBox is either hung or not started. To clear this condition, either start the CPU (if <CTRL/P> was typed) or use the INIT/CPU if the EBox is hung.*
?MCP-E-NOCSMR	No acknowledgment from CSM -- CSM failed to respond to a request. If this occurred during a INIT/CPU command, chances are CSM is hung trying to initialize an IPR.*
?MCP-E-NODATA	CSM sent no data packet -- The command protocol includes a data packet but CSM failed to send one.*
?MCP-E-NODATW	CSM sent unexpected data packet -- CSM either sent an extra or an unexpected data packet when a packet was not part of the command protocol.†
?MCP-E-NORPKT	No response packet received from CSM -- CSM is hung trying to perform an initialization.*
?MCP-E-PK2CNS	CSM pkt2, cntl not set†
?MCP-E-UNEXPD	CSM sent data for non-data response†
?MCP-E-UNKCSM	Unknown CSM packet code†
?MCP-E-XCCHKS	X command cmd_check_sum failure -- The checksum calculated by MCP did not match the checksum associated with the command portion of the X command.

Table K-15 Macro Control Program (MCP) Error Messages (Cont.)

Header	Message Description
?MCP-E-XDCHKS	X command data_check_sum failure -- The checksum MCP calculated did not match the checksum associated with the data portion of the X command.
?MCP-E-XRTIMO	X command receiver time_out -- Once the X command has established a link, the console expects a byte no less than once a second. This message indicates that the X command has not send a byte during the last second.

*To determine where CSM is hung, do the following.

1. Type STOP CPU.
2. Type MIC. This will cause the current microsequencer PCs to be typed on the terminal.
3. Press the space bar 10 more times. A sequence of microsequencer PCs will be typed out. This helps us to find out what the CPU thinks its doing.
4. Press <RETURN>. This gets the user out of MIC mode.
5. Type UNHANG.
6. Type @STKFRM. The EHM may have built a machine check stack frame for this error condition. This command will dump ESC:17 through 2F.
7. Refer to the Console Software Spec to determine which CSM overlay was loaded.

†These are CSM protocol problems. First try reloading the EBox microcode. If that doesn't work, run the microdiagnostics.

Table K-16 Macro Control Program (MCP) Fatal Messages

Header	Message Description
?MCP-F-ABSDED	ABus Dead -- Indicates that ABus Dead was detected while in program I/O mode.
?MCP-F-INVCSM	Invalid CSM overlay number -- (Software Problem).
?MCP-F-PWRFAL	Power Fail -- Indicates that a Power Fail condition was detected while in program I/O mode.

Table K-17 Macro Control Program (MCP) Information Messages

Header	Message Description
?MCP-I-BBUINV	Battery backup unit invalid
?MCP-I-HDECOL	Hardware not up to proper ECO level -- According to the System ID Register (SID) and VENUS.REV the files on the RL02 are not compatable with the system. Check the contents of the VENUS.REV file against the system ID.
?MCP-I-MCLDST	Aborting redundant cold-start attempt -- A cold restart has been attempted and failed. Subsequent automatic cold restarts are aborted.
?MCP-I-MWRMST	Aborting redundant warm-start attempt -- A warm restart has been attempted and failed. A second warm restart attempt will be aborted and a cold restart will be attempted if enabled via the System Control Panel.
?MCP-I-LARPWR	Lost array refresh power (warm-start not possible) -- Either the BBU is switched off or the 10 minute time limit expired for the battery backup unit to supply power to the array refresh circuit. A warm restart is no longer possible. A cold restart will be attempted, if enabled via the System Control Panel.
?MCP-I-NOPAMM	No physical address memory map
?MCP-I-RPBBSY	RPB restart-in-progress flag set -- The warm restart attempt failed. If the Restart Control Switch is in the correct position, a cold restart will be attempted.
?MCP-I-RPBINV	RPB invalid/not found -- The RPB in memory is invalid. The warm restart must be aborted. If the Restart Control Switch is in the correct position, a cold restart will be attempted.

Table K-18 Macro Control Program (MCP) Warning Messages

Header	Message Description
?MCP-W-ADROOR	Address out of range -- The address specified is outside the range for examine/deposit commands.*
?MCP-W-DATOOR	Data out of range -- The data specified exceeds 32 bits.*
?MCP-W-INVIPR	Invalid IPR number -- The IPR number argument is currently unassigned.*
?MCP-W-INVRRW	Read or write not allowed -- The command specified is not appropriate because the IPR is either read or write only.*
?MCP-W-XINCOM	X command in .com file invalid -- X commands cannot be executed in a command file.*

*Use the console software HELP command or refer to the Console Software Specification for more details on command usage.

K.4 KEEP ALIVE FAIL MESSAGES

The program module MCPSNP, the KAF snapshot routine, provides the messages described below. It first prints the following banner.

"Attempting to save machine state due to:"

One of the following reason messages is appended to the banner.

1. DOUBLE ESCRATCH PARITY ERROR (KAF Reason Code: 18) -- The Error Handling Microcode (EHM) determined that there was a parity error in both copies of the EBox scratchpad RAMs. This condition probably occurred when the EHM was trying to correct a GPR parity error by copying the good GPR to the bad GPR. This is a nonrecoverable error condition. The EHM responded to this error condition by looping at EBox UPC 20, which resulted in this KAF.

   This is a sticky problem. There is a good chance that the GPR parity error will once again be detected when the KAF routine uses CSM to read the EBox scratchpad RAMs. If that is the case, then the Escratch record of the Snap file will contain the contents of the Escratch up to the point where the parity error occurred. The remainder of that section will contain all 1s (FFF...). In addition to being unable to copy some or all of the Escratch, the KAF routine will be unable to copy the CPU IPRs, the PAMM, the top 64 longwords on the interrupt stack, and the SBIA/Nexus registers. Table K-19 lists the probable cause of this fault.

Table K-19 Double Escratch Parity Error Fault (KAF)

Module	Probability	RAMs (GPRs)
L0209/EDP	High	E500, E501, E502, E503 E613, E614, E615, E617 E712, E713, E714, E715 E812, E813, E814, E815 E903, E907
L0219/EBE	Low	

2. MACHINE DOUBLE ERROR (KAF Reason Code: 19) -- EHM was in the process of handling an error when a second EBox-related error was detected. This is a nonrecoverable error condition. The EHM responded to this error condition by looping at EBox UPC 21, which resulted in this KAF.

   The user should contact the RDC and request they analyze the Snap file. Meanwhile, if possible, use VSRBLD to translate the Snap file. Otherwise, translate the Snap file manually. The ESC should contain a partial machine check stack frame for the first error. Beware, however, the EHM over wrote the following Escratch locations with status from the second error.

   ESC: 12 (Trap Vector)
   ESC: 15 (EBCS)
   ESC: 19 (VMQ)
   ESC: 2F (PSL)

3. WBUS PARITY ERROR (KAF Reason Code: 1A) -- The EBox updated all copies of the GPRs with bad parity. This is a nonrecoverable error condition. The EHM responded to this error condition by looping at EBox UPC 24, which resulted in this KAF.

   The symptoms indicate that the parity calculated on the WBus data did not match the parity calculated at the output of the WREG. Most likely, the IBox or FBox is either contaminating the WBus or one of the parity generators is bad. Table K-20 lists the most probable causes of WBus parity errors.

Table K-20 WBus Parity Error Faults (KAF)

Module	Probability
L0209/EDP	High
L0219/EBE	High
L0206/IDP	Low
L0212/FBA	Low
L0223/FTM	Low (FBox Terminator)

4. CPU ERROR HALT (KAF Reason Code: 1B) -- The KAF condition was initiated by the Console Support Microcode (CSM). The specific reason for the KAF is contained in the Master Header Record Byte 13 (CSM Status Word Entry Code). See list below.

   a. Code=0 -- CSM could not be forced to run by the console program after the KAF. The EBox microcode may have been corrupted, so reinitialize the system. If that doesn't work, run the microdiagnostics.

   b. Code=4 -- Interrupt Stack not valid. The CPU was processing an interrupt or an exception, but when it attempted to push error status information on the interrupt stack, it discovered that the interrupt stack was mapped NO ACCESS or NOT VALID. This generally indicates that the CPU got into loop handling an interrupt or exception, causing the interrupt stack to overflow to a page mapped NO ACCESS or NOT VALID.

      Translate the Snap file. Look at the last 64 longwords on the interrupt stack. That may provide a clue to the loop that the CPU was in. Also, look at the contents of the CSLINT register. If the CPU was in an interrupt loop it may be possible to get some idea of the source. Finally, look at EHSR and the machine check section of the Escratch record to determine if the CPU was handling an error.

c. Code=5 -- Non-Ebox or VMS ENTERED double error. There are two ways this condition can occur.

   (1) If the EHM was processing an error and a second (non-EBox) error was detected, the EHM will call CSM with a code of 5 in CSM.STATUS. In this case, there will be a partially built stack frame (for the first error) in the Escratch. The vector address in EHSR will identify the port that reported the second error.

   (2) If the VMS machine check handler was handling an error and a second error was detected, the EHM will build a stack frame for the second error and then call CSM with a code of 5 in CSM.STATUS. In this case, the first machine check stack frame will be on the interrupt stack, and the second machine check stack frame in the Escratch.

d. Code=6 -- Kernel mode HALT. The processor executed a HALT instruction while in kernel mode. Use the PC and the interrupt stack to determine the reason.

e. Code=7 -- SCB vector with <1:0> = 3. The vector code field in the system control block <1:0> was equal to 3, which is a reserved code. Either the SCB was overwritten or a wrong vector address was generated.

f. Code=8 -- SCB vector with <1:0> = 2. The vector code field in the system control block <1:0> was equal to 2, which means service this event in Writable Control Store (WCS). However, the WCS either does not exist or was not loaded. The result of the operation is a HALT. Again, either the SCB was overwritten or a wrong vector address was generated.

g. Code=9 -- Pending error on HALT. The CPU was in the process of handling an error condition when P was typed on the console. The interrupt stack, the Machine Check section of the Escratch, and the contents of ERSR indicate the type of error the CPU was processing when P was typed.

h. Code=A -- CHMx with IS = 1. The CPU executed a change mode instruction when PSL <26> (interrupt stack) was set.

i. Code=B -- CHMx vector <1:0> not 0. The CPU executed a change mode instruction and the SCB vector code <1:0> was not equal to zero.

5. UNCORRECTABLE CS PARITY ERROR (KAF Reason Code: 1C) -- The console was unable to correct a control store or Dispatch RAM for one of the reasons shown in Table K-21.

Table K-21 Uncorrectable CS Parity Error Faults (KAF)

Reason	Description
INTERR	CSPE interrupt, code invalid (0)
MBTERR	"RAM ID" multi-bit-error, uncorrectable
MUNREC	MCS MBox not recoverable
NOECCD	"RAM ID" no ECC data in table
PCFAIL	"RAM ID" correction attempted and failed
SYNGTR	"RAM ID" syndrome > ram_size, uncorrectable
SYNZRO	"RAM ID" syndrome = 0, uncorrectable
UPCERR	"RAM ID" can't read box address

NOTE

Refer to Table K-11 for a description of uncorrectable control store and dispatch parity errors. Examine the contents of CSES to determine the RAM, address, and syndrome. Then refer to the RAM callout tables in the section on stack frame analysis to identify the failing RAM.

6. POWER SYSTEM FAILURE (DC LOW) (KAF Reason Code: 1D) -- During a normal power failure, ac low will precede dc low by approximately 10 ms. This allows VMS enough time to sweep the cache, save the state of the GPRs, CPU registers, etc. However, if a DC regulator fails (resulting in dc low only), then this message is printed and a KAF results. Review the EMM record of the Snap file to determine the exact cause of the failure.

7. UNKNOWN MACHINE HANG (KAF Reason Code: 1E) -- The KAF timer expired and the KAF routine read the EBox UPC and the CSM status word, but was unable to determine the cause of the KAF. Most likely, the system is stalled or hung in a loop waiting for some event to occur.

   Contact the RDC and request they analyze the Snap file. Meanwhile, if possible, use VSRBLD to translate the Snap file or translate the snap file manually. The ESC may contain a partial machine check stack frame.

There are two additional messages displayed by the KAF routine, as described below.

1. Both Snap files still valid -- The KAF routine was called to capture (snapshot) the state of the system but both Snap files were still valid. That is, they still contained status captured during the previous two KAF conditions. As a result, the console will not capture the state of the machine. Instead, it will attempt to restart CSM at 100D and call the reboot routine.

2. SNAPx.DAT created -- Currently there two Snap filenames used by the console: SNAP1.DAT and SNAP2.DAT2. This message shows the name of the Snap file that the KAF routine created as a result of the KAF condition.

APPENDIX L
GLOSSARY OF TERMS

The following terms and mnemonics are used throughout the manual for simplicity of expression. This appendix defines the meaning of each term or mnemonic.

CBus -- The CPU bus is a 32x8-bit dual access RAM used for CPU/console communication. The CBus RAM is the means by which the RX, TX, STX, and TODR register functions are provided to the CPU.

CCS -- Console Command Strings are diagnostic commands issued from the VAX processor, under program control, to the console program. The console interprets the string as a series of console commands to be executed.

CIO mode -- Console I/O mode. One of two modes that the console program can be running in (see also PIO mode). In CIO mode, the console program accepts command input from the LCL and RD terminal ports.

CPU -- In this document, CPU refers to the VAX central processor running macroinstructions.

CSM -- Console Support Microcode running in the EBox while the console program is in CIO mode (MACRO context) provides the console program access to various address spaces in the CPU.

DSM -- Diagnostic Support Microcode runs in the EBox while the console program is in CIO mode (DIAGNOSTIC context) and provides the console access to various address spaces in the CPU.

EMM -- Environmental Monitoring Module. This microprocessor-controlled unit is part of the power system. It communicates to the console program to provide control and status of the power system.

EMM port -- This is a PCI port used for EMM/console communication.

FRU -- Field Replaceable Unit. Refers to a module, subassembly, cable, or circuit component that can easily be replaced to repair a hardware problem.

KAF -- Refers to a CPU Keep Alive Failure. This is a condition detected in PIO mode where the CPU stops executing instructions. In most cases, the only recourse for the console program is to reinitialize and reboot the CPU. A restart of the CPU is attempted if the KAF was one of those that was defined as a CPU ERROR HALT in Digital Standard 032.

keyword -- A keyword is an argument on a console command line (that is not a switch).

LCL port -- This is a PCI port used for console communications with the local terminal device.

LCL terminal -- The terminal device connected to the LCL port.

logical register -- A set of visibility bits grouped together to form a register. The EXAM/SDB command is one way of displaying the state of a logical register.

MPS -- Modular Power System.

PAMM -- The Physical Address Memory Map is a RAM in the MBox that is initialized by the INIT/PAMM command. The PAMM assigns memory arrays and I/O adapters to specific ranges of physical memory addresses.

PCI -- Programmable Communications Interface (generic term for USART).

PIO mode -- Program I/O mode. One of two modes that the console program can be running in (see also CIO mode). In PIO mode, the console is a dedicated I/O device to the CPU.

QBA -- The Q-Bus Adapter interfaces the 16-bit Q-Bus with the 8-bit console memory bus, and also performs the necessary handshaking with the Q-Bus device (RLV12 controller) for DMA and register transfers.

RD port -- This is the PCI port used for console communications with the remote (RDY) terminal device.

RD terminal -- The terminal device connected to the remote port. Note that this connection could be made through a modem.

register ID -- A unique 16-bit number (with the most significant bit set) assigned to a logical register. May be used as an argument to the EXAMINE/SDB command, but intended as an "tag" used internally by the console program.

SDB -- The Serial Diagnostic Bus provides the console with visibility into the CPU. It is also used for loading control stores and state information into the CPU.

SDB ID -- A unique 16-bit number (with the most significant bit clear) assigned by the Chaser utility to single visibility signal.

snapshot -- The process of the console collecting data from the CPU after a KAF and saving it on the console's disk.

TODR -- The Time-Of-Day Register is a 32-bit integer in the CBus that the console increments once each 10 ms to provide the CPU with Time-of-Year information.

TOY -- Time-Of-Year, in terms of 10 ms increments from the beginning of the year.

UPC -- Short for micro-PC.

visibility register -- See logical register.

VTERM -- Visibility Terminator. Special logic device used to terminate up to 8 ECL signals and provide visibility to each bit.