Company Confidential

Seahorse Diagnostics

First Draft - May 1983

d i g i t a l

First Draft Printing - May 1983

This manual describes the Seahorse diagnostic system. Your comments are requested. Please send them to:

Ken Western
Digital Equipment Corporation
2265 116 NE
Bellevue, Wa.
98004

Seahorse Diagnostics

Contents

Chapter 1: Diagnostics Overview

VAX Diagnostic System Structure 1

Console Environment 4

CPU Cluster Environment 7

System Environment 7

User Environment 8

Seahorse Diagnostic System Structure 8

Microverify 9

Macroverify 10

Standalone Diagnostics 10

Chapter 2: Troubleshooting Flowchart

TBS

Chapter 3: Loading and Bootstrapping Seahorse Diagnostics

TBS

Chapter 4: Building and Maintaining the Diagnostic System Disks

TBS

Chapter 5: Macroverify

Diagnostic Distribution 19

Distribution Media 19

Memory Requirements 19

Execution Time 20

Setting Up Procedures 20

Bootstrapping Macroverify 20

Macroverify Operation 20

Interpreting Program Output 22

Error Messages 23

Chapter 6: CPU Hard Core Diagnostic

Diagnostic Distribution 27

Distribution Media 27

Memory Requirements 27

Execution Time 28

Bootstrapping the Diagnostic 28

Operation 28

Testing Process 29
Interpreting Program Output 33
Further Action 34

Chapter 7: The Memory Diagnostic

Diagnostic Distribution 35
Distribution Media 35
Memory Requirements 36
Execution Time 36
Caution 36
Bootstrapping the Diagnostic 36
Memory Diagnostic Operation 36
Command Syntax 36
DISABLE 38
ENABLE 41
HELP 45
LOOP_TEST 46
MEMORY_SIZE 49
START 51
VIEW 52
Using the Commands 53
Testing Procedure 57
Interpreting Program Output 61
Error Messages 61

Further Action 65

Chapter 8: Using the VAX Diagnostic Supervisor

Bootstrap Procedure 69

Issuing Commands 69

The Command Set 70

Program and Test Sequence Commands 70

Debug and Utility Commands 71

Execution Control Commands 72

SCRIPT Command 72

ABORT 75

ATTACH 76

CLEAR BREAKPOINT 79

CLEAR EVENT FLAGS 80

CLEAR FLAGS 81

CONTINUE 85

DEPOSIT 86

DESELECT 88

EXAMINE 89

LOAD 91

NEXT 92

RUN 93

SCRIPT 97

SELECT 98

SET BASE 99

SET BREAKPOINT 100

SET DEFAULT 101

SET EVENT FLAGS 103

SET FLAGS 104

SET FLAGS DEFAULT 107

SET LOAD 108

SHOW BREAKPOINTS 109

SHOW DEVICE 110

SHOW EVENT FLAGS 111

SHOW FLAGS 112

SHOW LOAD 113

SHOW SECTION 114

SHOW SELECTED 115

START 116

SUMMARY 120

Chapter 9 Q22 Bus Diagnostics

DLV11-J Diagnostic 122

Diagnostic Distribution 122

Distribution Media 122

Memory Requirements 122

Execution Time 122

Setting Up Procedures 122

Bootstrapping the Diagnostic 123

Operation 123

Testing Process 129

Error Message Handling 130

Further Action 133

DZV11 Diagnostic TBS 134

RQDX1 RX50, RD51 Diagnostic TBS 135

QNA Diagnostic 136

Chapter 10: Guide to Writing a Device Diagnostic

TBS

Chapter 11: Interpreting and Controlling Diagnostic Messages

Diagnostic Messages 139

Finding the Relevent Documentation 139
Documentation File 140
Memory Allocation Map 141
Program Header and Test Modules 141
Error Analysis 142
Detailed Test and Subtest Descriptions 142

List of Figures

Figure 1. VAX Diagnostic System 3
Figure 2. VAX Diagnostic System Structure 5
Figure 3. Sample Macroverify Output 22
Figure 4. Sample CPU Diagnostic Output 29
Figure 5. Sample Memory Diagnostic Output 55
Figure 6. Memory Diagnostic Flow 59
Figure 7. Sample DLV11-J Output 127

List of Tables

Table 5-1 Macroverify Error Messages 25
Table 6-1 CPU Diagnostic Tests 31

Table 7-1 Editing Characters 37

Table 7-2 DISABLE Command Arguments 39

Table 7-3 ENABLE Command Arguments 43

Table 7-4 LOOP_TEST Command Arguments 47

Table 7-5 Memory Size Conversion 49

Table 7-6 Memory Diagnostic Error Messages 63

Table 8-1 UUT Naming Conventions 77

Table 8-2 Execution Control Flags 83

Table 8-3 DEPOSIT Command Qualifiers 86

Table 8-4 EXAMINE Command Qualifiers 89

Table 8-5 RUN Command Qualifiers 95

Table 8-6 SET DEFAULT Command Qualifiers 101

Table 8-7 Execution Control Flags 105

Table 8-8 START Command Qualifiers 117

Table 9-1 DLV11-J Diagnostic Error Messages 131

Chapter 1
Diagnostics Overview

All processors in the VAX family contain a comprehensive diagnostic system which enables all levels of user to verify that the entire system is serviceable. The diagnostic procedures enable you to investigate a suspected problem, isolate the fault, and carry out the required maintenance procedures.

In this chapter the overall VAX diagnostic strategy is described, this is followed by Seahorse specific information.

VAX Diagnostic System Structure

The VAX diagnostic system hierarchy consists of six program levels:

- Level 1 consists of operating system (usually VMS) based diagnostic programs (diagnostics).
- Level 2R consists of VAX Diagnostic Supervisor (VDS) based diagnostics. These can only run under an operating system, for example, peripheral diagnostics which are not supported by VDS in standalone mode.
- Level 2 consists of VDS based diagnostics that can be run either under an operating system (online) or in standalone mode, for example, bus interaction programs or formatter and reliability level peripheral diagnostic

programs.

- Level 3 consists of VDS based diagnostics that can be run in standalone mode only, for example, functional and repair level peripheral diagnostics, CPU cluster diagnostics.
- Level 4 consists of standalone macrodiagnostic programs that run without VDS in standalone mode, for example, the hard core instruction test (which tests the basic CPU functions required to run VDS). Although level 4 diagnostics are excellent troubleshooting tools, they are more difficult to use than level 2, 2R, and 3 diagnostics. Therefore, you should only use level 4 diagnostics when you are sure that you cannot obtain the same error information with other diagnostics.
- Console Level consists of console based diagnostics that can be run in standalone mode only, for example, microdiagnostics, the console program, and ROM resident power up tests such as Microverify.

The six program levels operate in the context of four user environments; user, system, cluster, and console. The four user environments run in two operating modes; online (under an operating system), standalone (without an operating system). Figure 1 illustrates these concepts.

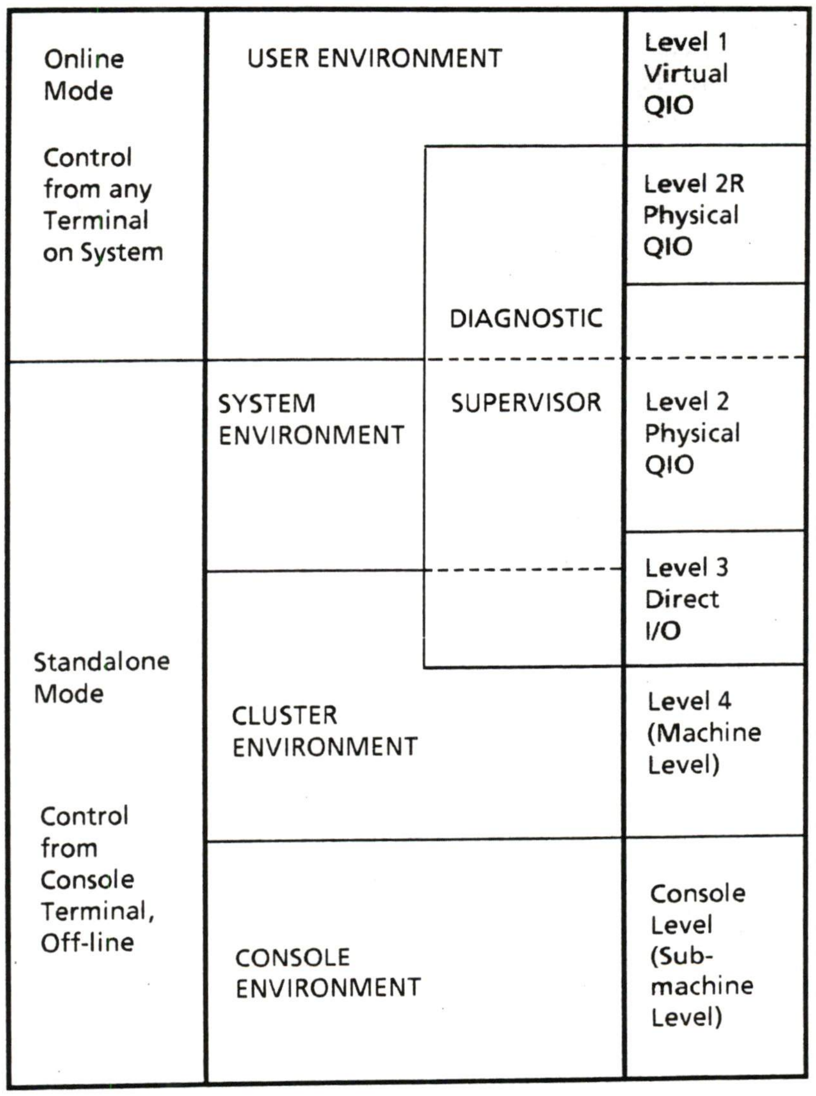

Figure 1. VAX Diagnostic System

The four environments form the basis of the building block diagnostic approach. For each environment a portion of the hardware functions as a hard core which is assumed to be fault-free. Specific diagnostics operate from this hard core environment to test the hardware in an area beyond the hard core. The hard core for each environment consists of the hard core of the next lower environment. Figure 2 shows the building block structure of the diagnostic environments. Note the overlap of the area under test in the different environments.

Console Environment

The console environment consists of submachine level hardware, software and firmware. It provides operator control functions, system programmer debugging functions, and basic machine diagnostic functions. This environment operates in standalone mode only. You control the system from the console terminal. The console hardware forms the hard core that must be fault free in order to run the microdiagnostics. Note that the CPU microcode remains untested in this environment.

The console environment is the most basic, installation-specific part of the diagnostic system. Microverify performs this function on Seahorse.

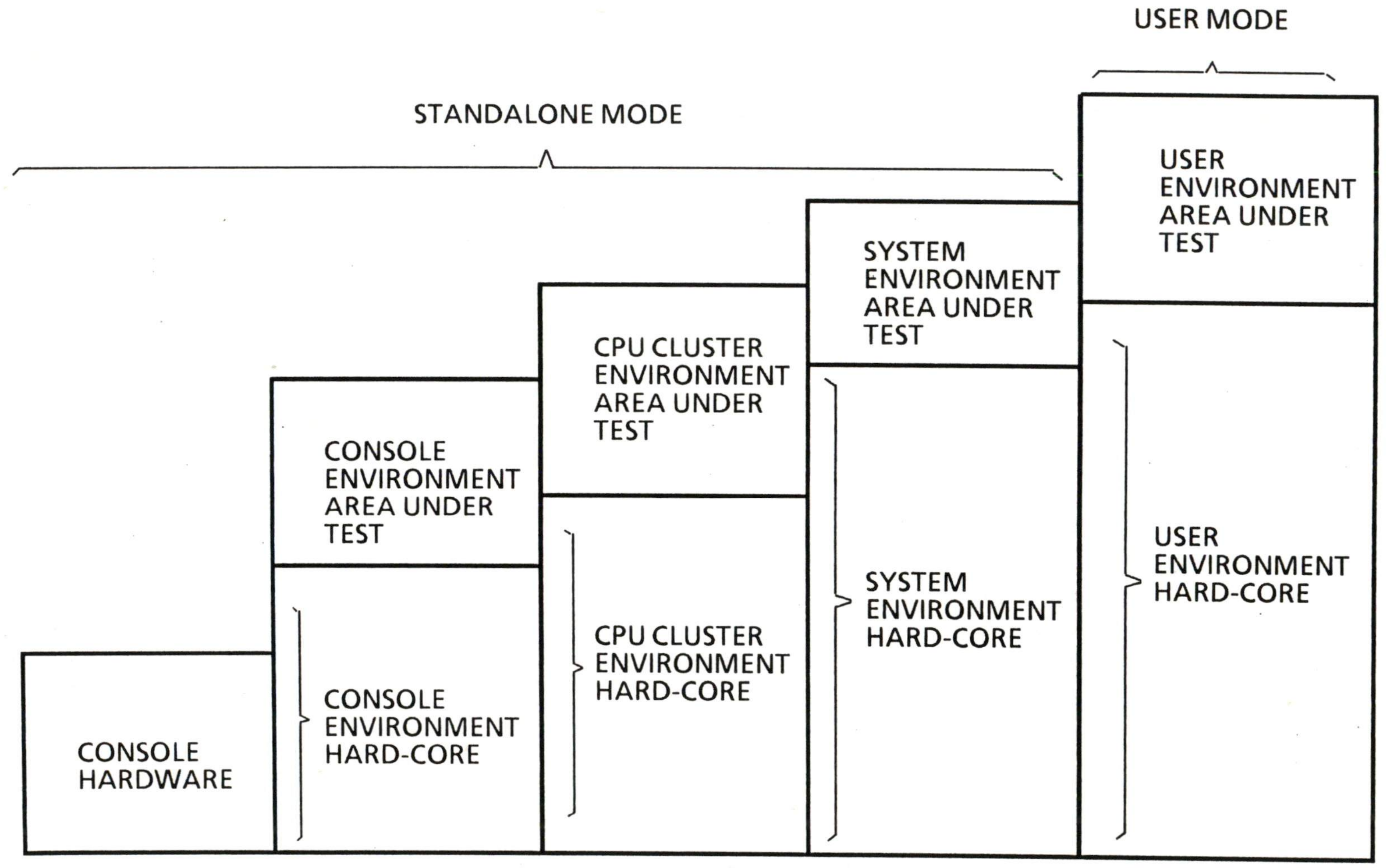
USER MODE
STANDALONE MODE
USER ENVIRONMENT AREA UNDER TEST
SYSTEM ENVIRONMENT AREA UNDER TEST
CPU CLUSTER ENVIRONMENT AREA UNDER TEST
CONSOLE ENVIRONMENT AREA UNDER TEST
USER ENVIRONMENT HARD-CORE
SYSTEM ENVIRONMENT HARD-CORE
CPU CLUSTER ENVIRONMENT HARD-CORE
CONSOLE ENVIRONMENT HARD-CORE
CONSOLE HARDWARE

Figure 2 . VAX Diagnostic System Structure

CPU Cluster Environment

The CPU cluster environment consists of the console environment plus the machine level components (CPU, Memory, I/O channels) that support standalone, macro level program execution. This environment operates in standalone mode only. You control the system from the console terminal. The hardware tested in the console environment, by the microdiagnostics, forms the hard core of the cluster environment.

The VAX CPU cluster environment provides a small number of level 3 and 4 diagnostic programs. In a building block fashion, these diagnostics test basic and extended CPU functions, memory functions and I/O channel functions. The I/O channel and cluster interaction diagnostic programs make full use of the channel loopback capability.

In Seahorse, examples of these diagnostics are the CPU and memory standalone diagnostics.

System Environment

The system environment consists of a variety of diagnostics, ranging from level 3 repair diagnostics to level 2 (QIO) device exercisers. This environment operates in standalone mode only. You control the system from the console terminal.

The VAX system environment level diagnostic strategy is to implement a series of level 3 repair diagnostics and level 2 functional diagnostics for each I/O subsystem. The diagnostic series (level 3 and level 2) for each I/O subsystem is designed to give building block test coverage. This evolves from static logic and maintenance loopback tests (level 3) through basic function and electromechanical timing tests (level 3 or level 2) to media reliability, acceptance, and multi-device exercisers (level 2). The hardware tested in the CPU cluster environment forms the hard core for the system environment.

In Seahorse, examples of these diagnostics are the Q22 bus device diagnostics.

User Environment

The user environment includes level 2 diagnostics, which run in the system environment, as well as level 2R programs which do not. Many of the diagnostics that run in the user environment will run simultaneously with user programs. However, some, for example, the system diagnostic control program, require exclusive use of the system. The user environment traditionally operates in online mode under VMS. You can control the diagnostic process from any terminal on the system, including the console terminal.

Seahorse also makes use of the VAX Élan operating system to implement user-environment level diagnostics, for example, Macroverify.

Seahorse Diagnostic System Structure

Seahorse diagnostics consists of three levels of diagnostics procedures, which correspond to the overall VAX diagnostic approach:

- Microverify
- Macroverify
- Standalone CPU, memory and Q22 bus diagnostics

Microverify is an automatic procedure, Macroverify and the standalone diagnostics must be explicitely loaded from floppy disks and then bootstrapped.

Microverify and Macroverify provide sufficient testing for you to isolate a failure down to a unit. You then select the most convenient method for repairing the fault. You may call Digital Field Service, use the DECmailer Service or take the faulty unit to one of the walk-in service centers in your area (these are listed in TBS).

Microverify

Microverify consists of permanently resident microcode that is executed automatically when system power is turned on, or when Seahorse is bootstrapped. If you require further verification after Microverify has successfully run, you may load and run Macroverify or the standalone diagnostics.

Microverify is the first attempt at detecting and isolating a problem. Microverify indicates a problem by illuminating Light Emitting Diodes (LEDs) on the Data Path Module (DAP). Three LEDs exist, they isolate the problem to either the DAP, the Memory Controller (MCT) or to a source other than the processor.

You are informed that Microverify is running by a percent (%) sign being printed on your console. When Microverify successfully concludes, a second percent sign is printed.

You may specifically invoke Microverify by issuing the TEST console command. You may inhibit Microverify during bootstrap by issuing the /X qualifier with the BOOT console command. The Console Commands are described in *Part Three: Operation*, Chapter 3.

Macroverify

Macroverify is designed to be a quick, high-level test. It executes macro instructions which exercise subsystems not verified by Microverify. After loading and bootstrapping Macroverify, no further user intervention is required. Loading Macroverify is described in Chapter 3.

Successful completion of Macroverify indicates that your program will run in the applicable operating system environment.

If Macroverify does not complete successfully, a message will be printed which will inform you of the failed subsystem. Based on this output, you may wish to run a device specific diagnostic to confirm the Macroverify diagnosis and to further isolate the problem.

Standalone Diagnostics

The standalone diagnostics consist of:

- The CPU hard core diagnostic.
- The memory diagnostic.
- The Q22 bus option diagnostics.

The CPU hard core diagnostic performs additional CPU testing to Microverify. Messages will be printed on the console informing you of the results of the tests. This diagnostic is described in Chapter 6.

The memory diagnostic exercises memory by writing and reading patterns while checking for errors. It isolates memory cell failures to the failing board. Messages will be printed on the console informing you of the results of the tests. This diagnostic is described in Chapter 7.

The Q22 bus option diagnostics are run under the control of VDS. These diagnostics enable you to check supported Q22 bus options. The diagnostics are invoked with commands which allow, for example, a diagnostic to be loaded, run, suspended, a statistical report to be printed, and the diagnostic to be restarted. VDS and Q22 bus diagnostics are described in Chapters 8 and 9.

Chapter 2
Troubleshooting Flow Chart

Details TBS

This chapter will contain a flowchart which describes the steps required to isolate problems to a Field Replaceable Unit (FRU) in the FRS Seahorse system. It will contain:

- Mechanical suggestions, for example, verifying that the power cord is connected.
- Suggestions on the order of running diagnostics.

Chapter 3 Loading and Bootstrapping Seahorse Diagnostics

Details TBS.

This chapter will describe loading the required diagnostic disks, how to bootstrap, how to recognize that the bootstrap failed, the BOOT/X and TEST commands. It will also list the contents of the disks and their part numbers.

Chapter 4
Building and Maintaining the Diagnostic System Disks

Details TBS.

This chapter will describe creating and customizing diagnostic disks from the distribution media.

Chapter 5 Macroverify

Macroverify is a high-level diagnostic program that is designed to verify the correct functioning of a Seahorse system. It is usually run as part of the Installation Verification Process and as the first step in diagnosing a system that is suspected of having a problem. Macroverify is normally run before other diagnostics, in order to provide a quick check and rule out applications software or user errors.

Macroverify is a functional level diagnostic, that is, after running it you should be assured that a typical VAX Élan application will run on the system tested without hardware problems. If the diagnostic fails to run, you should run additional diagnostics to further isolate the problem. See the Troubleshooting Flowchart in Chapter 2 for further details.

Macroverify is not compatible with Digital's Automated Product Test (APT) system.

Diagnostic Distribution

Distribution Media

Macroverify is distributed on the diskette labeled TBS, Part Number TBS.

Memory Requirements

Macroverify requires TBS Kb of memory to run.

Execution Time

Macroverify will require TBS minutes to run.

Setting Up Procedures

You should power up all devices in the configuration, and set all disk drives ready for I/O.

The disk-testing performed by Macroverify is non-destructive and no special software formatting is required for any of the online disks.

Bootstrapping Macroverify

Macroverify is a standalone diagnostic and is bootstrapped from the distribution media described earlier. See Chapter 3 for bootstrapping information.

Macroverify Operation

Macroverify is implemented as a VAX Élan application running in kernel mode. It contains a script which identifies the possibe Seahorse configuration. This script is followed during the test process.

For each device in the script, a test is made to see if the device responds to its assigned Q22 bus address. If it does not, the status message 'NOT PRESENT IN SYSTEM OR INCORRECTLY INSTALLED' is output in the table entry corresponding to the device. If the device does respond, a sequence of user level tests is then performed. The result of these tests are reported as either 'TEST SUCCEEDED' or 'TEST FAILED'.

The tests performed on each device are very simple operations and are typical of the types of operations that would be performed by a user application, for example, reading a block of data from a disk.

The sample output in Figure 3 illustrates Macroverify operation. It was carried out on a Seahorse system consisting of 1 Mb of memory, two diskette drives, and one fixed disk drive. Note that a DLV11-J four-channel communications controller is not part of the system and is flagged as such during the run.

After Macroverify is booted, Microverify is automatically run (this is identified by the double percent signs). Microverify is followed by:

- Headers which contain the VAX Élan and Macroverify version numbers.
- A table containing a line of information about each tested device and the result of the test.
- A message announcing that testing has finished. Any errors detected when Macroverify is running will be reported, these are described below.

Macroverify runs until completion without the need for you to intervene. When it has finished running, a HALT instruction will be executed and the console command (>>>) prompt will be displayed.

```
>>>B DUO
%%
VAX Élan V1.0
Macroverify V1.0
Testing                 Results
Memory                  TEST SUCCEEDED(1.0 Mb)
Disk Drive 0
(floppy drive DU0)      TEST SUCCEEDED
Disk Drive 1
(floppy drive DU1)      TEST SUCCEEDED
Disk Drive 2
(fixed drive DU2)       TEST SUCCEEDED
DLV11-J                 NOT CURRENTLY IN CONFIGURATION
                        OR INCORRECTLY INSTALLED
Macroverify test completed
?06 PC = nnnnnnnn
>>>
```

Figure 3. Sample Macroverify Output

Interpreting Program Output

You first verify that the amount of memory is the same as the amount installed on your system (1 Mb in this example). If the value differs you should consider running Memory Diagnostics, see Chapter 7, as your memory may be incorrectly configured or have other hardware errors.

If a device is reported as not present or is installed incorrectly, you should determine if the device should be present. If the device should be present, you should verify that all of the module switches on the interface board are set correctly and that the interface is correctly connected to the Q22 bus. See the Installation Instructions provided with the device for further information. If the interface is correctly set up, then you should run the device specific diagnostic to continue the troubleshooting procedure.

Error Messages

All error messages are reported in the form:

```
Macroverify - text
```

The text will list and describe the problem and direct you further. Table 5-1 lists the error messages and the suggested action.

Table 5-1 Macroverify Error Messages

Message	Meaning	Action
TBS		

Chapter 6
CPU Hard Core Diagnostic

The CPU hard core diagnostic (EHxxx - name TBS) is a level 4 standalone diagnostic. It is designed to verify that the CPU is functional to the extent that the memory diagnostic or VDS will run correctly. This portion of the CPU is termed the 'hard core'.

This diagnostic is normally run when CPU problems are suspected due to the failure of Microverify or Macroverify, the presence of intermittent system errors, or a failure during bootstrapping. It is usually run before attempting to run any of the other CPU diagnostics, see Chapter 9, Q22 Bus Diagnostics.

This diagnostic is compatible with Digital's Automated Product Test (APT) system.

Diagnostic Distribution

Distribution Media

This diagnostic is distributed on the diskette labeled TBS, Part Number TBS

Memory Requirements

This diagnostic requires 60 Kb of memory to run.

Execution Time

This diagnostic requires 20 seconds to run.

Bootstrapping the Diagnostic

This diagnostic is a standalone diagnostic and is bootstrapped from the distribution media described earlier. See Chapter 3 for bootstrapping information.

Operation

Once the diagnostic is bootstrapped, Microverify is automatically run (this is identified by the two percent signs being listed). The CPU tests are then run continuously. After each 100 passes of the diagnostic (this takes approximately 20 seconds), a single line of output containing the version number of the diagnostic is listed. The diagnostic will continue until an error is detected or until you terminate the program by pressing the BREAK key .

Any errors detected will cause the diagnostic to execute a HALT instruction. No text will be output. Figure 4 illustrates an error free sample diagnostic run.

```
>>>B DUO
%%
EHXXX - CPU Hardcore Diagnostic V1.0
EHXXX - CPU Hardcore Diagnostic V1.0
EHXXX - CPU Hardcore Diagnostic V1.0
                    .
                    .
```

Figure 4. Sample CPU Diagnostic Output

Testing Process

The diagnostic is composed of 19 tests. Table 6-1 lists and describes these tests. Each test is executed in sequence, starting at test 1. The strategy is to test a large enough portion of the CPU so that standalone or VDS diagnostics will run on the CPU without CPU problems injecting spurious errors into other diagnostics. This is not an exhaustive architectural test, but it does verify a large amount of functionality.

Table 6-1 CPU Diagnostic Tests

Number	Meaning
1	General Register Addressing Test. Each general register is written with its address and the contents are verified. Any stuck-at-1, stuck-at 0, or shorted address bits will be explicitly shown.
2	Conditional Branch Test. Condition codes are preset to particular patterns and all of the conditional branches are executed. This is repeated for all 16 combinations of NZVC condition code bits.
3	Operand Specifier Test. This test consists of a series of subtests which verify address modes and operand specifier routine flows.
4	Data Size Test. This test verifies that the correct data size is used in autoincrement, autodecrement, etc.
5	AOBLEQ, AOBLS Test. Functionally verifies these instructions.
6	SOBGEQ, SOBGTR Test. Functionally verifies these instructions.
7	BBS, BBC Test. Functionally verifies these instructions.
8	BBSS, BBCS Test. Functionally verifies these instructions.
9	BLBS, BLBC Test. Functionally verifies these instructions.
10	CASEB, CASEW, CASEL Test. Functionally verifies these instructions.
11	BSBB, BSBW, JSB, RSB, BRB, JMP Test. Functionally verifies these instructions.
12	Integer Arithmetic and Logic Test. Verifies the following instructions; ADDB2, ADDL2, ADDL3, ADDW2, ADDW3, ADWC, ASHL, ASHQ, BICB2 BICB3, BICL2, BISB2, BISL2, BISW2, BITL, BITW, CLRL, CLRQ, CMPB, CMPL, CMPW, CVTBL, CVTWL, CVTLW, DECL, DECW, DIVL2, DIVL3, DIVW3, EDIV, EMUL, INCL, INCW, MNEGL, MCOML, MOVB, MOVW, MOVQ, MOVZBL, MOVZWL, MULL2, MULL3, MULW2, MULW3, PUSHL, ROTL, SUBB3, SUBL2, SUBL3, SUBW2, SUBW3,SBWC, TSTB, TSTL, TSTW, XORB3.
13	Procedure Call Instruction Test. The CALLG, CALLS, and RET instructions are verified.
14	Miscellaneous Instruction Test. The PUSHR, and POPR instructions are verified.
15	Exception and Interrupt Test. Verify CPU ability handle exceptions and interrupts. A reserved opcode tests exceptions and software interrupts test interrupt processing.
16	Interval Timer Test. Functionally tests the timer.
17	Console Transmitter Test. Functionally tests the console terminal transmitter.
18	Variable Bit Field Instruction Test. Functionally verifies the INSV, EXTV, EXTZV, CMPV, and CMPZV instructions.
19	Queue Instruction Test. The INSQUE and REMQUE instructions are functionally tested.

Interpreting Program Output

If the diagnostic detects an error, it executes a HALT instruction. No diagnostic message is produced. This is because if this level of CPU functionality fails, it is probable that the operations required to format and list a readable error message will also fail. If the diagnostic does execute a HALT instruction, the console will output text in the form:

```
?06 PC = nnnnnnnn

>>>
```

Console halt code 06 indicates that a HALT instruction was executed. The hexadecimal value represented by 'nnnnnnnn' is the value of the Program Counter (PC) at the time the HALT instruction was executed, that is, it points to the first byte of the instruction after the HALT.

If the CPU fails, replace the DAP board and rerun the diagnostic.

If you wish to determine the test and subtest which failed, you must consult the program listing for this diagnostic. The PC value may be converted to an offset into the diagnostic listing by subtracting the value of R10, which is always set to the base address of the program. For example:

```
?06 PC = 00040023

>>> E/G/L A

G 0000000A 00000200
```

The offset into the listing that will help locate the test that failed is 3FE23 (40023 - 200).

Continuing execution after the HALT instruction has been executed will result in executing the failing test again.

If necessary the diagnostic may be restarted at location TBS.

Further Action

The Troubleshooting Flowchart in Chapter 2 illustrates how this diagnostic is used in conjunction with other troubleshooting techniques.

Chapter 7 The Memory Diagnostic

The memory diagnostic (EHxxx - name TBS) is a level 4 (standalone) diagnostic and is designed to verify the correct functioning of MSV11-PL memory modules.

Each memory module has 512 Kb of MOS memory. Each byte has one additional internal parity bit. The MSV11-PL is described in the *MSV11-P User Guide*(EK-MSV0P-UG-001).

This diagnostic is a functional level diagnostic, that is, after running it you should be able to identify the memory board that failed. It is usually run when memory errors are detected by VAX Élan or microVMS or when intermittent program failures suggest that a problem exists in the memory subsystem. Additionally, you may choose to run this diagnostic if you discover that one or more memory module's red error LED indicators are lit. It is usually run after running the CPU hard core diagnostic to verify that the CPU is functioning correctly.

This diagnostic is not compatible with Digital's Automated Product Test (APT) system.

Diagnostic Distribution

Distribution Media

The diagnostic is distributed on the diskette labeled TBS, Part Number TBS.

Memory Requirements

The diagnostic requires TBS Kb of memory to run.

Execution Time

The diagnostic requires TBS minutes per 512 Kb for full testing (default), and TBS minutes per Kb with no memory parity testing (described later).

Caution

The previous contents of the memory will be destroyed.

Bootstrapping the Diagnostic

This is a standalone diagnostic and is bootstrapped from the distribution media described earlier. See Chapter 3 for bootstrapping information.

Memory Diagnostic Operation

Once the diagnostic is booted, Microverify is automatically run (this is identified by the double percent signs). Microverify is followed by a header which contains the memory diagnostic version number. You are then prompted to issue commands which will control the diagnostic.

Command Syntax

You may issue commands either in upper or lower case. They are echoed in upper case. Before terminating a line by entering Carriage Return, you may enter any of the following editing characters. Note that the convention ^U or ^R means that you hold down the CTRL (Control) key and press the U or R key at the same time.

Table 7-1. Editing Characters

Character	Function
DELETE	Backspaces one character and deletes it. A backslash (\\) is printed, followed by the deleted character and another backslash.
^U	You may use ^U instead of Delete to delete an entire line. It ignores what you have typed on the current line and performs a Carriage Return. You may then reenter the line.
^R	Performs a Carriage Return and displays the current line. The cursor is left at the end of the line so that you can continue typing input. You may use ^R when you have deleted a lot of characters on the line and cannot easily read its contents.

Commands and option keywords may be abbreviated to the first letter.

DISABLE

The DISABLE command enables you to disable test feature selected with the ENABLE command.

Format

```
EHxxx> DISABLE option.
```

Arguments

option. This argument lists the test features to be disabled. Table 7-2 describes the test features. Note that the DISABLE and ENABLE options are identical.

Example

```
EHxxx> DISABLE BELL
```

Table 7-2 DISABLE Command Arguments

Feature	Use	Feature	Use
BELL	Sounds the bell when a failure is detected. When disabled no bell is output. Default - enabled.	PARITY	Causes the diagnostic to execute test number 10 as part of the test sequence. This test forces writing bad parity throughout memory in order to test the parity error detection logic. When disabled, this test is skipped. Disabling parity testing causes the diagnostic to run faster by TBS minutes per 512 Kb. Default - enabled.
ERRORS	Outputs error messages to the console terminal when errors are detected. When disabled, these messages are suppressed. Has no effect on messages produced as a result of invalid user input. Default - enabled.	RELOCATION	Causes the diagnostic to relocate itself in memory, allowing the vacated memory to be tested. When disabled the memory in which the diagnostic resides is not tested. Default - enabled.
HALT	Halts when a failure is detected. When disabled an error message will be produced and processing will continue. Default - disabled.		
LOOP	Loops on the first test that encounters an error. Looping continues until ^C is typed when the diagnostic will return and prompt for user comands. When disabled, testing will continue at the next test. Default - disabled.		
MAP	Causes a memory map to be produced before the first test is executed. Default - enabled.		

ENABLE

The ENABLE command enables you to select test features.

Format

```
EHxxx> ENABLE option
```

Arguments

option. This argument lists the test features to be enabled. Table 7-3 describes the test features. Note that the DISABLE and ENABLE options are identical.

Example

```
EHxxx> ENABLE  HALT
```

Table 7-3 ENABLE Command Arguments

Feature	Use	Feature	Use
BELL	Sounds the bell when a failure is detected. When disabled no bell is output. Default - enabled.	PARITY	Causes the diagnostic to execute test number 10 as part of the test sequence. This test forces writing bad parity throughout memory in order to test the parity error detection logic. When disabled, this test is skipped. Disabling parity testing causes the diagnostic to run faster by TBS minutes per 512 Kb. Default - enabled.
ERRORS	Outputs error messages to the console terminal when errors are detected. When disabled, these messages are suppressed. Has no effect on messages produced as a result of invalid user input. Default - enabled.	RELOCATION	Causes the diagnostic to relocate itself in memory, allowing the vacated memory to be tested. When disabled the memory in which the diagnostic resides is not tested. Default - enabled.
HALT	Halts when a failure is detected. When disabled an error message will be produced and processing will continue. Default - disabled.		
LOOP	Loops on the first test that encounters an error. Looping continues until ^C is typed when the diagnostic will return and prompt for user comands. When disabled, testing will continue at the next test. Default - disabled.		
MAP	Causes a memory map to be produced before the first test is executed. Default - enabled.		

HELP

The HELP command enables you to list a description of the memory diagnostic commands, options and syntax.

Format

```
EHxxx> HELP
```

Example

```
EHxxx> HELP
output TBS
```

LOOP_TEST

The LOOP_TEST command enables you to change the test sequence executed by the diagnostic after the START command is issued.

Format

```
EHxxx> LOOP_TEST n
```

Arguments

n. This argument specifies the decimal test number. Table 7-4 list the test numbers and their meaning. Specifying test number 0 will disable the loop testing feature.

Example

```
EHxxx> LOOP_TEST 10
```

Table 7-4 LOOP_TEST Command Arguments

Number	Meaning
1	Verify CSR Function. Determine the number of CSR's present and verify that they retain settings and clear correctly when the Q22 bus is initialized.
2	Verify Configuration. Verify size of memory and memory contiguity. Verify CSR/memory correlation.
3	Memory Address Test. Write memory address into memory one word at a time.
4	Memory Address Test. Write memory address into memory one byte at a time.
5	Memory Address Test. Write one's complement of memory address into memory one word at a time.
6	Memory Address Test. Write the 16 Kb bank number into memory one byte at a time.
7	Memory Address Test. Write the one's complement of the 16 Kb bank number into memory one byte at a time.
8	Data Test. Write a known constant throughout memory a word at a time.
9	Worst Case Noise Test. Write a series of exercising worst case patterns throughout memory a word at a time. These pattern include testing for stuck-at-0, stuck-at-1 and worst case word parity cases
10	Worst Case Noise Parity Byte Test. Force writing bad parity in each byte of memory with a set of worst case patterns. This test is only executed if the PARITY option is enabled.
11	Instruction Execution Test. Execute a series of simple instruction sequences throughout memory.
12	'Marching' 1's and 0's Test. Each 16 Kb bank of memory is exercised by running several passes of alternating bytes of 1's and 0's through the bank.
13	Memory Checkerboard Test. A series of patterns are written throughout memory and are checked after two seconds to verify memory is refreshed properly.
14	Block Mode Test. A series of block mode tests are performed throughout memory.

MEMORY_SIZE

The MEMORY_SIZE command enables you to specify the amount of memory available on the system. This is the only mandatory command. It must be specified before the START command is issued.

Format

```
EHxxx> MEMORY_SIZE n
```

Arguments

n. This argument specifies the memory size in Kb. Table 7-5 shows how to convert from Mb to Kb. Specifying a zero memory size causes the diagnostic to size the memory for you (autosizing). Autosizing should be used with care as improperly installed or faulty memory may cause the autosizing operation to miscalculate memory, thereby reducing test coverage.

Table 7-5 Memory Size Conversion

Size (Mb)	Size (Kb)
0.5	512
1.0	1024
1.5	1536
2.0	2048
2.5	2560
3.0	3072
3.5	3584
4.0	4096

Example

```
EHxxx> MEMORY_SIZE 1024
```

START

The START command enables you to start the diagnostic testing sequence.

Format

```
EHxxx> START
```

Example

```
EHxxx> START
```

VIEW

The VIEW command enables you to list the status of all the ENABLE and DISABLE command options and the specified memory size.

Format

```
EHxxx> VIEW
```

Example

```
EHxxx> VIEW
```

Using the Commands

You may enter a series of commands in any order you wish, the only constraint is that the START command must be the last command issued. No testing occurs until the START command is issued.

All commands are optional except the MEMORY_SIZE command. An error message and a prompt will be issued if you specify a START command without first specifying the size of memory. The most common use of the command set will be to issue a MEMORY_SIZE command followed by a START command. This will begin execution of the full diagnostic with all of the default ENABLE and DISABLE options.

If the LOOP_TEST command is not specified, (or if LOOP_TEST 0 is specified), the diagnostic will print a memory map (if one has been requested), then begin testing at the first test and continue until all tests have been executed or an error is encountered. At the end of the last test, the diagnostic relocates itself in memory (if the RELOCATE option is enabled) and the test sequence is repeated on the memory that was occupied by the diagnostic. After this second test sequence, the diagnostic is moved back to the memory it previously occupied (See Figure ....TBS). At the end of the test pass, a message is output indicating that testing is completed and the entire test is then repeated.

If the LOOP_TEST command is specified, testing will start at the specified test, and continue executing that test until you stop it (See Figure...TBS).

Typing ^C at any time during the test process will cause the diagnostic to restart execution and prompt for command input. The ENABLE and DISABLE command options and the memory size are saved. You may rerun the same test sequence by issuing a START command on its own, or you may change the commands to run the test in a different way. You may terminate the diagnostic by pressing the BREAK key.

You may restart the diagnostic by starting it at location TBS. When restarting in this fashion all options are reset and must be reentered.

Figure 5 illustrates a sample console output. The test system has 2Mb of memory.

```
>>>B DU0
%%

EHXXX - MSV11-PL Memory Diagnostic V1.0

EHXXX>MEMORY 2048
EHXXX>START

Memory Map

                              TBS


                         ------------------------------------------------
                         | Key:  *     No memory in this 16 Kb bank     |
                         |       0-F   Memory controller number         |
                         ------------------------------------------------

Test Started.
Test Ended - no errors detected

Test Started.
Test Ended - no errors detected
               .
               .
```

Figure 5. Sample Memory Diagnostic Output .

Testing Procedure

The memory modules are configured to start on any 16 Kb boundary. Consequently, testing is performed in multiples of 16 Kb units. Up to 256 16 Kb units may reside in the 4 Mb memory address space on Seahorse. A maximum of 16 MSV11-PL memory modules may be installed on one Seahorse system up to a total of 4 Mb of memory. Memory must also be contiguous in the physical address space and the first memory module installed must be jumpered to start at physical address 0 (module configuration jumpers P,N,M,L,T,X,W,V all 'out').

The optional memory map produced by the diagnostic provides a picture of how memory modules have been installed and is used to correlate a failing memory location to a particular MSV11-PL memory module. The map is a matrix of 256 elements, one element for each possible 16 Kb memory element in the 22-bit memory address space. The map is organized in four rows by 64 columns, each row represents 1 Mb of memory. The row and column headings may be used to form the base physical address of the 16 Kb element in memory. The row headings provide the most significant 2-bits of the 22-bit physical address. They are represented as 32-bit hexadecimal numbers in the actual map output, with five placeholding X's for the 20 least significant bits (five hexadecimal digits) of the memory address. The five least significant hexadecimal digits may be read vertically downwards as the column headings.

For example, the physical base address of the memory described by the intersection of row two and column four is 0010C000 (001X XXXX + 0C000). The address range is from the base address for 16 Kb, or from 0010CO00 to 0010FFFF inclusive.

In each memory element of the matrix there is a flag describing the presence of memory at that location. The entry is either '*' which signifies that no memory is installed at that address or is a hexadecimal digit from '0' to 'F' containing the memory controller number associated with that memory.

For example, in the example map output, the entry for 0010C0000 is 2, signifying that memory is contained on the third memory card (controller 2) on the system.

The diagnostic is composed of a series of 14 individual tests. See the LOOP_TEST command for a description of the tests. Each test is independent of the other tests and shares a common structure with the others. The flowchart in Figure 6 illustrates the structure and also shows how the BELL, ERROR, HALT and LOOP ENABLE/DISABLE options affect the diagnostic control flow.

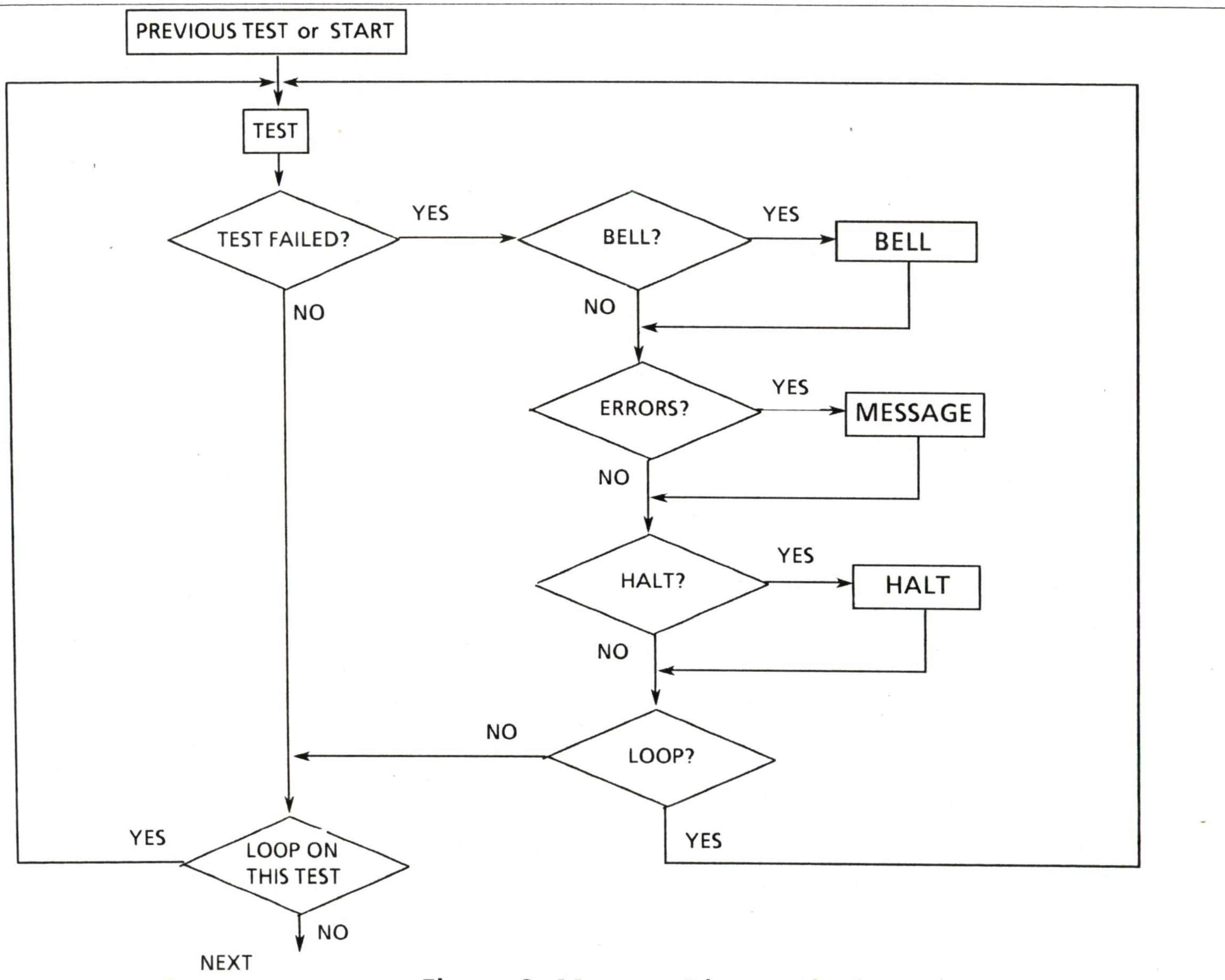

Figure 6 . Memory Diagnostic Flow

Interpreting Program Output

Any error encountered by the diagnostic will cause it to output an error message. The format of the error message will depend upon whether the error was caused by an erroneous command entered by you or an error in the memory under test.

Error messages produced as a result of incorrect input have the format:

```
EHXXX- text
```

'text' is self-explanatory, see the next section.

Error messages produced as a result of an error in the memory under test are only output if the MESSAGE option is enabled with the ENABLE command. These messages have the format:

```
EHXXX- Error during test x, subtest x
EHXXX- text
```

The first line of the error message identifies the test and subtest numbers of the test item that failed. The x's represent a series of digits.

The second line of the error message is a description of the error detected.

Error Messages

Table 7-6 lists the error messages and the suggested action.

Table 7-6 Memory Diagnostic Error Messages

Message	Meaning	Action
TBS		

Further Action

If the execution of this diagnostic fails or if you cannot load it into memory, you may wish to carry out the following manual tests to further isolate the problem.

First, open the Seahorse cabinet and locate the MSV11-PL memory modules. These are located by the module number 'M8067' stamped on their removal handles. Verify that the green LEDs on each memory module are lit. If any or all of the LEDs are not lit then the modules are not receiving 5 volts power. This condition must be rectified before any further tests may be attempted.

Next, if new memory modules have recently been installed, remove each newly installed module and check the option jumper/switch settings. Note that in Seahorse systems, the First Address Range (FAR) jumpers (X,W,V) are used to select which of eight 512 Kb addressing boundaries the memory is configured on and that the Partial Starting Address (PSA) jumpers (P,N,M,L,T) are all 'out' (PSA = 0). Also, verify that the Control and Status Register (CSR) jumpers (D,C,B,A) are set correctly and that the lowest addressed memory module is set with jumpers D,C,B, and A 'out' (controller number is 0). If you have more than one memory module in your system, you may wish to remove them all and reinsert them one at a time, testing each by hand (described next), until installing a module causes the test to fail.

Testing memory manually involves using the console to store and examine memory locations. You should attempt to store a known pattern in known locations in memory, use increments of 16 Kb (4000 hexadecimal) or 512 Kb (800000 hexadecimal) to attempt to discover if any memory module will respond.

The following test program may be entered and run. It causes a selected portion of memory to contain the physical address of each longword of memory in that memory longword.

Write Address

```
80    50   D0        1$:   MOVL      R0,(R0)+
  FO  51   F5              SOBGTR    R1,1$
           00              HALT
```

The following program causes the ones complement of the physical address to be stored in each selected longword.

Write Complement Address

```
80    50   D2        1$:   MCOML     R0,(R0)+
  FO  51   F5              SOBGTR    R1,1$
           00              HALT
```

These programs contain Position Independent Code (PIC) and may be loaded anywhere in memory. Before starting these programs you should set R0 to contain the base address of the block of memory to be initialized, and set R1 to contain the number of longwords of memory to initialize. The values of R0 and R1 are destroyed and must be reentered each time the program is invoked. The only difference between the programs is in the first byte of code, the MOVL instruction is D0 (hexadecimal) and the MCOML instruction is D2 (hexadecimal). For example, the following commands are used to load and execute the first program into memory locations 0-7 on a system with 1 Mb of memory.

```
>>>D/P/L 0 F58050D0
>>>D 4 FA51
>>>D/G/L 0 8
>>>D 1 3FFFE
>>>S 0
```

Chapter 8

Using the VAX Diagnostic Supervisor

Bootstrap Procedure

The VAX Diagnostic Supervisor (VDS) is bootstrapped from the distribution media. See Chapter 3 for a description of this procedure.

Issuing Commands

Commands are issued in response to the VDS prompt, DS>, for example:

```
DS> SHOW DEVICES <CR>
```

Carriage Return (the key labeled <CR> or Return) is the normal line terminator and is omitted from the command syntax descriptions and examples.

You can abbreviate commands and qualifiers to the minimum number of characters that uniquely identify them, for example, L for the LOAD command, ST for the START command, B for the BYTE qualifier.

Qualifiers are preceded by a slash (/) character, for example, /LONGWORD (/L).

CTRL/C (which is invoked by pressing the key marked C with the CTRL or Control key held down) is used to return control from a diagnostic to the DS> prompt. You may then issue any valid command. CTRL/C is the only VDS command that you may issue while a program is running. If the program is running in conversation mode, CTRL/C will return control to a command interpreter in the program.

A hyphen (-) is used to continue a command string from one line to the next, for example:

```
DS> SET FLAGS, BELL, HALT, OPERATOR-
PROMPT, TRACE
```

An exclamation mark (!) is used to identify the beginning of a comment, for example:

```
DS> EXAMINE 30    ! Display the contents of the
                  ! longword which is offset
                  ! 30 from base address E00
```

The Command Set

The commands are divided into four functional groups:

- Program and test sequence control commands.
- Debug and utility commands.
- Execution control commands.
- Script facility command.

Program and Test Sequence Commands

These commands allow you to select programs, or portions of programs, and control the sequence of test execution. This group contains the following commands:

- ABORT
- ATTACH
- CLEAR EVENT FLAGS

- CLEAR FLAGS
- CONTINUE FLAGS
- DESELECT
- LOAD
- RUN
- SELECT
- SET LOAD
- SHOW DEVICE
- SHOW LOAD
- SHOW SECTION
- SHOW SELECTED
- START
- SUMMARY

Debug and Utility Commands

These commands allow you to isolate errors and to examine and modify the diagnostic in memory. This group contains the following commands:

- CLEAR BREAKPOINT
- DEPOSIT
- EXAMINE
- NEXT
- SET BASE
- SET BREAKPOINT
- SET DEFAULT
- SHOW DEFAULT

VDS allows you to include up to 15 simultaneous breakpoints in a program.

Execution Control Commands

These commands allow you to modify the characteristics of VDS and the diagnostic programs. These modifications are implemented using execution control and event flags.

Execution control flags control, for example, printing error messages, ringing the bell, halting and looping the program.

Event flags are status bits maintained by VDS. They are used to signal events.

This group contains the following commands:

- CLEAR EVENT FLAGS
- CLEAR FLAGS
- SET EVENT FLAGS
- SET FLAGS
- SET FLAGS DEFAULT
- SHOW EVENT FLAGS
- SHOW FLAGS

SCRIPT Command

This command enables you to execute a command file. The command file contains a predefined sequences of commands, called a script. The script is automatically executed. You can create the command file using an editor or with the VMS CREATE command.

The command file must be a contiguous ASCII file (as created by VAX-11 Record Management Services) on an ODS-1 or ODS-2 disk file structure. The command file must be line oriented and records must not exceed 72 characters. VDS treats all records as commands. The command file should only contain strings of VDS commands.

VDS processes the file as follows:

1. The current program is aborted if necesssary.
2. The file is read into a buffer.
3. A pointer is initialized which points to the first line of the script.
4. A flag is set to indicate that the next command is to be taken from the script.
5. As the commands in the script are processed, the prompt and command text are displayed on your terminal.
6. When the end of the script is reached, VDS outputs the message '@<EOF>'.

VDS dynamically allocates the memory buffer for control and position information, and for the script text. Each script descriptor is linked to previous descriptors, allowing scripts to be nested. The number of nesting levels permitted is limited by the availability of memory. You can invoke nesting by including another SCRIPT command inside a script.

You may interrupt a script while it is running by typing CTRL/C. You can issue any command while the script is suspended, however, if you wish to eventually resume the script with the CONTINUE command, only the following commands may be issued:

- CLEAR
- CONTINUE
- DEPOSIT

- EXAMINE
- NEXT
- SET
- SHOW
- SUMMARY

Any other command will result in all script text being flushed and control being returned to VDS. In general, a command will flush a script if it would be meaningless to continue with the script after the command has been executed.

ABORT

The ABORT command enables you to terminate any currently running diagnostic and return control to VDS. The clean-up code contained in the diagnostic is executed; when execution is complete the DS> prompt is displayed. You may then enter any command except the CONTINUE command.

Format

```
DS> ABORT
```

Example

The following example shows how the ABORT command may be used with CTRL/C and the SUMMARY command:

```
...........diagnostic is running.............
^C
DS> SU

...........statistical report is printed.............

DS> AB
```

In this example CTRL/C halts the diagnostic, a statistical report is printed, the diagnostic is then terminated and the clean-up code is executed.

ATTACH

The ATTACH command enables you to define each Unit Under Test (UUT). The command is repeated for each UUT prior to running the diagnostic program. Each UUT is uniquely defined by a hardware designation and a link. Table 8-1 lists the UUT-types, their links and device names.

Format

```
DS> ATTACH UUT-type link-name device-name
```

Arguments

UUT-type. This argument supplies the hardware designation of the UUT, for example, RX50, DZV11.

link-name. This argument supplies the name of the device that links the UUT to the Q22 bus. In most cases the UUT is linked via intermediary links, for example, an RX50 is linked via a RQDX1 Controller. Each link device, with the exception of the Q22 bus, must be attached before it can be used as a link.

device-name. This argument supplies the name of the specific unit to be tested. The device name is specified in the form XYannn:

- XY is the device name.
- a is an alphabetic character representing the device controller.
- nnn is a unit number in the range 0-255 specifying the UUT with respect to the controller, for example, LPA0.

Table 8-1 UUT Naming Conventions

UUT-type	Link-name	Device-name	Additional Information
KD32	QBUS	KD0	
DZV11	QBUS	TTa	qcsr qvector
DLV11	QBUS	TTa	qcsr qvector ndata nstop parity even parity
QNA	QBUS	??a	qcsr qvector
RQDX1	QBUS	DUa	qcsr qvector
RX50	DUa	DUan	
RD51	DUa	DUan	

Additional information is required for some devices in order to further define the device to VDS and is supplied with the ATTACH command or in response to prompts:

- qcsr is an octal Q22 bus CSR address in the range 760000-777776.
- qvector is an octal Q22 bus vector in the range 2-776.
- ndata is the number of data bits transmitted/received (7/8).
- nstop is the number of stop bits transmitted/received (1/2).

- parity is 'yes' if parity detection is enabled, 'no' otherwise.
- even parity is 'yes' if even parity is selected, 'no' otherwise.

Example

In the following example, the DZV11 is attached:

```
DS> AT DZV11 QBUS TTA
CSR?       760010
VECTOR?  300
```

CLEAR BREAKPOINT

The CLEAR BREAKPOINT command clears a previously set breakpoint. The breakpoint is specified by its address.

Format

```
DS> CLEAR BREAKPOINT     ALL
                         address
```

Arguments

You must specify either the ALL or address argument.

ALL. This argument clears all previously set breakpoints.

address. This argument specifies the address of a specific breakpoint. No error message is returned if a breakpoint does not exist at the specified address. See the SET BASE command for further information about specifying addresses.

Example

In the following example the breakpoint is cleared at the address which is offset 30 from the base address:

```
DS> CL B 30
```

CLEAR EVENT FLAGS

The CLEAR EVENT FLAGS command clears previously set event flags. Event flags are status posting bits maintained by VDS. Event flags are used by diagnostics to perform a variety of signaling functions, including communicating with you.

Format

```
DS> CLEAR EVENT FLAGS    ALL
                         flag-list
```

Arguments

You must specify either the ALL or flag-list argument.

ALL. This argument clears all previously set event flags.

flag-list. This argument consists of a string of event flag numbers in the range 1-23, each event flag must be separated by a comma.

Example

```
DS> CL E ALL
```

CLEAR FLAGS

The CLEAR FLAGS command clears execution control flags that have been previously set with the SET FLAGS command.

Format

```
DS> CLEAR FLAGS flag-list
```

Arguments

flag-list. Is one or more of the flag mnemonics listed in Table 8-2. The list elements are separated by commas.

Example

```
DS> CL LOOP, TRACE
```

Table 8-2 Execution Control Flags

Flag	Use	Flag	Use
ALL	Clears all flags in this list.	LOOP	Causes the program to enter a predefined scope loop on a test or subtest which detects a failure. Looping continues until you type CTRL/C, you may then continue, clear the flag and continue, or abort the program.
BELL	Sounds the bell when a failure is detected.	OPERATOR	Informs VDS that the operator is present and that operator interaction is possible. If this flag is not set, VDS will take actions to ensure that the test continues when the operator is not present.
HALT	Halts when a failure is detected. After all messages associated with the error have been sent, VDS enters a command wait state, you may then continue, restart or abort the program. This flag takes precedence over the LOOP flag.	PROMPT	Indicates to VDS that a long form of dialogue is to be output, i.e., the limits and defaults for user responses are to be displayed.
IE1	Inhibits error messages at level 1. All messages will be suppressed except those forced by the diagnostic or VDS.	QUICK	Indicates to the program that the operator wants a quick verify mode of operation. The interpretation of this flag is program dependent.
IE2	Inhibits error messages at level 2. Basic and extended information concerning the failure is suppressed, only header information (the first three lines)is output.	TRACE	Causes VDS to report the operation of each individual test within a program as VDS passes control to that test.
IE3	Inhibits error messages at level 3. Extended information concerning the failure is suppressed, only header and basic information is output.	SEARCH	TBS
IES	Inhibits the summary report.	VERIFY	TBS

CONTINUE

The CONTINUE command causes a diagnostic to resume execution from where it was suspended. You may use this command to proceed from a breakpoint, error halt, summary report printing or after CTRL/C is typed.

Format

```
DS> CONTINUE
```

Example

The following example shows how the CTRL/C, SUMMARY and CONTINUE commands are used together to obtain statistics on the program being run, after the statistics are printed, execution is continued:

```
.......diagnostic is running.......

^C
DS> SU

.......statistical report is printed.......

DS> CO

.......diagnostic continues running.......
```

DEPOSIT

The DEPOSIT command enables you to deposit data in a specified memory location or processor register.

Format

```
DS> DEPOSIT qualifier-list   address data
```

Arguments

qualifier-list. This argument defines the format of the data to be deposited. The applicable qualifiers are shown in table 8-3

Table 8-3 DEPOSIT Command Qualifiers

Qualifier	Effect
BYTE	A byte is to be deposited.
WORD	A word is to be deposited.
LONGWORD	A longword is to be deposited.
HEXADECIMAL	Deposit in hexadecimal radix format.
DECIMAL	Deposit in decimal radix format.
OCTAL	Deposit in octal radix format.
NEXT=n	Deposit in the next 'n' locations. 'n' is decimal.

address. This argument specifies the memory location in which the data is to be deposited. It is interpreted as hexadecimal unless another default radix is specified with the SET DEFAULT command. You may optionally specify the address as decimal, octal, or hexadecimal by preceding it with %D, %O or %X respectively. If you wish to deposit data in one of the various processor registers, the address argument may be specified as one of R0-R11, AP, FP, SP, PC, or PSL. VDS will also accept R12-R15 for AP, FP, SP and PC respectively. See the SET BASE command for additional information about specifying addresses.

data. Is the data to be deposited.

Example

The following example deposits 0001 (hexadecimal) in the word located at offset 30 from the base address:

```
DS> DEP/W/H 30 0001
00000E30:  0001
```

DESELECT

The DESELECT command removes one or more devices from the list of units to be tested.

Format

```
DS> DESELECT    ALL
                device-list
```

Arguments

You must specify either the ALL or device-list argument.

ALL. The ALL argument deselects all devices.

device-list. This argument consists of a list of the devices to be deselected. If more than one device is specified, they must be separated by commas.

Example

```
DS> DES TTA:
```

EXAMINE

The EXAMINE command displays the contents of a specified memory location or processor register. The data is displayed in the format defined by the qualifiers.

Format

```
DS> EXAMINE qualifier-list  address
```

Arguments

qualifier-list. This parameter defines the format of the data to be displayed. The applicable qualifiers are shown in table 8-4

Table 8-4 EXAMINE Command Qualifiers

Qualifier	Effect
BYTE	A byte is to be examined.
WORD	A word is to be examined.
LONGWORD	A longword is to be examined.
HEXADECIMAL	Display in hexadecimal radix format.
DECIMAL	Display in decimal radix format.
OCTAL	Display in octal radix format.
NEXT=n	Deposit in the next 'n' locations. 'n' is decimal.

address. Specifies the memory location to be examined. It is interpreted as hexadecimal unless another default radix is specified with the SET DEFAULT command. You may optionally specify the address as decimal, octal, or hexadecimal by preceding the address with %D, %O or %X respectively. If you wish to deposit data in one of the various processor registers, the address argument may be specified as one of R0-R11, AP, FP, SP, PC, or PSL. VDS will also accept R12-R15 for AP, FP, SP and PC respectively. See the SET BASE command for additional information about specifying addresses.

Example

The following example displays the contents of the longword which is located at offset 30 from the base address of E00:

```
DS> E 30
00000E30:  DO513DO1
```

LOAD

The LOAD command loads a file, containing a diagnostic, from the default load device into main memory.

Format

```
DS> LOAD file-specification
```

Arguments

file-specification. This argument supplies the name of the diagnostic to be loaded. You can supply it as the five-letter code assigned to the diagnostic, for example, EHBAV. The default file extension is EXE. The storage device from which the file is loaded is the device established in a previous SET LOAD command.

Example

The following example loads the EHBAV diagnostic into memory:

```
DS> L EHBAV
```

NEXT

The NEXT command causes one or more macroinstructions to be executed. This command is useful when you wish to step through a diagnostic, for example, when a problem is suspected. You should only use this command when you have stopped the diagnostic at a breakpoint.

Format

```
DS> NEXT number-of-instructions
```

Arguments

number-of-instructions. Is the decimal number of instructions to be executed. If you do not specify a number, only one instruction will be executed. After the execution of each instruction, the Supervisor displays the PC of the next instruction and the contents of the next four bytes.

Example

The following example executes the next instruction:

```
DS> N

00000E31:    D0513D01
```

RUN

The RUN command loads a file from a specified load device and executes the diagnostic program contained in the file. It is equivalent to the LOAD and START command sequence.

Format

```
DS> RUN file-specification   qualifier-list
```

Arguments

file-specification. This argument specifies the file containing the diagnostic program. You need only supply the five-letter code that identifies the program, for example, ESTAA. The file is loaded from the default load device established with the previous SET LOAD command. The default file extension is EXE.

qualifier-list. This argument consists of optional qualifiers listed in Table 8-5.

Example

In the following example, the diagnostic named ESTAA is run:

```
DS> R ESTAA
```

In the following example, the diagnostic named ESTAA, manual section is run:

```
DS> R ESTAA/SEC:MANUAL
```

In the following example, the diagnostic named ESTAA tests 32 and 33 in the manual section are run:

```
DS> R ESTAA/SEC:MANUAL/TEST:32:33
```

In the following example, the diagnostic named ESTAA tests 6-12 are run:

```
DS> R ESTAA/TEST:6:12
```

In the following example, the diagnostic named ESTAA test 9, subtests 1-5 are run:

```
DS> R ESTAA/TEST:9/SUBTEST:5
```

In the following example, the diagnostic named ESTAA tests 9-n are run, n is the highest numbered test in the default section:

```
DS> R ESTAA/TEST:9
```

In the following example, the diagnostic named ESTAA test is run with three passes:

```
DS> R ESTAA/PASS:3
```

In the following example, the diagnostic named ESTAA test 9, subtests 1-4 are run, and then test 9, subtest 5 loops indefinitely:

```
DS> R ESTAA/TEST:9/SUBTEST:5/PASS:0
```

Table 8-5 RUN Command Qualifiers

Qualifier	Effect
SECTION:sec-nam	A section is a group of discrete tests. The section identifies function, execution times, and whether or not there is need for user intervention. If a section--name is supplied with the RUN command, the tests contained in that section will be run. If a section-name is not supplied, the tests contained in the section named DEFAULT are run. The diagnostic will contain a list of the available sections, or you may use the SHOW SECTIONS command after loading the diagnostic.
TEST:first:last SUBTEST:num PASSES:count	The TEST qualifier may be used on its own or in conjunction with the SUBTEST and PASSES qualifiers. If the TEST qualifier is specified with the first test and last test arguments, VDS sequentially passes control to the first through last tests inclusively. If the first test argument is combined with the SUBTEST qualifier, execution begins at the first test and concludes at the endof the subtest identified by num. The SUBTEST qualifier may also be combined with the PASSES qualifier to give you a loop-on-subtest capability. The value supplied with the PASSES qualifier must be a decimal in the range 1-0 where 0 is infinity. If the PASSES qualifier is not specified a default of one pass is used. If the TEST qualifier is not specified, all tests in the specified section are executed. If only the first test argument is specified, execution begins at the specified first test and concludes after the highest numbered test in the section.

SCRIPT

The SCRIPT command enables you to execute a script (a sequence of commands) contained in a command file. You should create the command file with a microVMS text editor such as EDT and then copy it onto the diagnostic load device.

Format

```
DS> @ file-specification
```

Arguments

file-specification. This argument supplies the device, directory, and name of the file to be loaded. The SET LOAD command could also be used to specify the device. If you do not specify the load device, VDS will assume the load device is the device from which VDS was loaded.

Example

The following is the contents of a command file named DUA0:[TEST]CMD.COM

```
DS> ATTACH DLV11J QBUS TTA 776500 300 8 1-
     NO NO
DS>SELECT TTA
DS>RUN EHXXX
```

The following example shows how this command file may be used:

```
DS> @ DUA0:[TEST]CMD
```

If you do not specify the file type and version, it is defaulted to COM and the highest numbered version.

SELECT

The SELECT command enables you to select each unit to be tested. This command takes effect the next time the diagnostic program is started. The units must have been previously attached using the ATTACH command.

Format

```
DS> SELECT   ALL
             device-list
```

Arguments

You must specify either the ALL or device-list argument.

ALL. This argument selects all devices.

device-list. This argument specifies the device or devices to be tested. If more than one device is specified, they are separated by commas.

Example

```
DS> SEL TTA:
```

SET BASE

The SET BASE command enables you to load a specified address into a software register (the 'base' register). The address is then used as a base to which the address specified in a SET BREAKPOINT, CLEAR BREAKPOINT, EXAMINE, or DEPOSIT command is added. The SET BASE command is useful when referencing code in diagnostic listings. The base should be set to the base address (see the program link map) of the program section referenced, the PC numbers provided in the listings can then be used directly when referencing locations in the program sections.

Format

```
DS> SET BASE address
```

Arguments

address. This argument specifies the base address. Note that a virtual address is normally equivalent to a physical address when memory management is not enabled. The current base address can be determined by issuing a SHOW BASE command and specifying address 0.

Example

In the following example, the base address is set to the beginning of the program section of the routine under examination:

```
DS> SET BA E00
```

SET BREAKPOINT

The SET BREAKPOINT command enables you to define when control is to be passed to VDS. Control will be passed when the PC points to the specified address. Up to 15 simultaneous breakpoints may be set within a program.

Format

```
DS> SET BREAKPOINT address
```

Arguments

address. This argument specifies the address at which the breakpoint is set. See the SET BASE command for further information about specifying addresses.

Example

In the following example, the breakpoint is set at the address which is offset 30 from the base address:

```
DS> SET BR 30
```

SET DEFAULT

The SET DEFAULT command enables you to define default qualifiers for the DEPOSIT and EXAMINE commands.

Format

```
DS> SET DEFAULT argument-list
```

Arguments

argument-list. This argument consists of a default data length and/or radix. If both are specified, they are separated by a comma. The defaults are initially set to longword and hexadecimal. You may specify a byte, word or longword data length, and an octal, decimal or hexadecimal radix.

Table 8-6 SET DEFAULT Command Qualifiers

Qualifier	Effect
BYTE	A byte is to be deposited or examined.
WORD	A word is to be deposited or examined.
LONGWORD	A longword is to be deposited or examined.
HEXADECIMAL	Deposit or examine in hexadecimal radix format.
DECIMAL	Deposit or examine in decimal radix format.
OCTAL	Deposit or examine in octal radix format.

Example

```
DS> SET D WORD,OCTAL
```

SET EVENT FLAGS

The SET EVENT FLAGS command sets the specified event flags. Event flags are status posting bits maintained by VDS. Event flags are used by diagnostics to perform a variety of signaling functions, including communicating with you.

Format

```
DS> SET EVENT FLAGS ALL
                    flag-list
```

Arguments

You must specify either the ALL or flag-list argument.

ALL. This optional argument sets all event flags.

flag-list. This optional argument consists of a string of numbers in the range 1-23, separated by commas.

Example

```
DS> SET E  ALL
```

SET FLAGS

The SET FLAGS command enables you to set execution control flags.

Format

```
DS> SET FLAGS flag-list
```

Arguments

flag-list. This parameter consists of one or more of the flag mnemonics listed in Table 8-7. The list elements are separated by commas.

Example

```
DS> SE BELL
```

Table 8-7 Execution Control Flags

Flag	Use	Flag	Use
ALL	Clears all flags in this list.	LOOP	Causes the program to enter a predefined scope loop on a test or subtest which detects a failure. Looping continues until you type CTRL/C, you may then continue, clear the flag and continue, or abort the program.
BELL	Sounds the bell when a failure is detected.		
HALT	Halts when a failure is detected. After all messages associated with the error have been sent, VDS enters a command wait state, you may then continue, restart or abort the program. This flag takes precedence over the LOOP flag.	OPERATOR	Informs VDS that the operator is present and that operator interaction is possible. If this flag is not set, VDS will take actions to ensure that the test continues when the operator is not present.
IE1	Inhibits error messages at level 1. All messages will be suppressed except those forced by the diagnostic or VDS.	PROMPT	Indicates to VDS that a long form of dialogue is to be output, i.e., the limits and defaults for user responses are to be displayed.
IE2	Inhibits error messages at level 2. Basic and extended information concerning the failure is suppressed, only header information (the first three lines) is output.	QUICK	Indicates to the program that the operator wants a quick verify mode of operation. The interpretation of this flag is program dependent.
		TRACE	Causes VDS to report the operation of each individual test within a program as VDS passes control to that test.
IE3	Inhibits error messages at level 3. Extended information concerning the failure is suppressed, only header and basic information is output.	SEARCH	TBS
		VERIFY	TBS
IES	Inhibits the summary report.		

SET FLAGS DEFAULT

The SET FLAGS DEFAULT command enables you to return all flags to their initial default setting. The default settings are OPERATOR and PROMPT.

OPERATOR informs VDS that the operator is present and that operator interaction is possible. If this flag is not set, VDS will take action to ensure that the test continues when the operator is not present.

PROMPT indicates to VDS that a long form of dialogue is to be output, that is, the limits and defaults for user responses are to be displayed.

Format

```
DS> SET FLAGS DEFAULT
```

Example

```
DS> SET F D
```

SET LOAD

The SET LOAD command enables you to establish the storage device from which VDS will load diagnostics. This command is used in conjunction with the LOAD or RUN commands.

Format

```
DS> SET LOAD   device   directory
```

Arguments

device and directory. These parameters specify the load device and directory. If the device and directory are not specified, the default load device is the device from which VDS was booted.

Examples

```
DS> SET L DMA0:[SYSMAINT]
DS> L ESDXA

DS> SET L DMA2:[SYSMAINT]
DS> R ESDXA
```

SHOW BREAKPOINTS

The SHOW BREAKPOINTS command enables you to display all currently defined breakpoints.

Format

```
DS> SHOW BREAKPOINTS
```

Example

```
DS> SH B
CURRENT BREAKPOINTS:
          00000E30 (X)
```

SHOW DEVICE

The SHOW DEVICE command enables you to obtain a display of the characteristics of the specified devices. If you do not supply a device name, the characteristics of all attached devices will be displayed.

Format

```
DS> SHOW DEVICE device-list qualifiers
```

Arguments

device-list. This argument specifies the device or devices to be displayed. If more than one device is to be displayed, the device names are separated by commas.

qualifiers. This argument enables you to specify the qualifiers /ADAPTER=name, /BRIEF. TBS.

Example

```
DS> SH D
```

response .....

SHOW EVENT FLAGS

The SHOW EVENT FLAGS command enables you to display a list of the event flags that are currently set. Event flags are status posting bits maintained by VDS. Event flags are used by diagnostics to perform a variety of signaling functions, including communicating with you.

Format

```
DS> SHOW EVENT FLAGS
```

Example

The following example shows how the SET EVENT FLAGS, CLEAR EVENT FLAGS and SHOW EVENT FLAGS commands can be coordinated:

```
DS> SET E 1,9,15
DS> CL E 2,6
DS> SH E
EVENT FLAGS SET: 15,9,1
```

SHOW FLAGS

The SHOW FLAGS command enables you to display a list of the execution control flags that are currently set. The flags are displayed as two lists of flag mnemonics. One list contains flags that are set, the other flags that are clear.

Format

```
DS> SHOW FLAGS
```

Example

The following example shows how the SET FLAGS, CLEAR FLAGS and SHOW FLAGS commands can be coordinated:

```
DS> SET F TRACE
DS> CL F QUICK
DS> SH F
CONTROL FLAGS SET: PROMPT, OPER, TRACE
CONTROL FLAGS CLEAR: QUICK, IES, IE3, IE2,-
IE1,BELL, LOOP, HALT, SEARCH, VERIFY
```

SHOW LOAD

The SHOW LOAD command enables you to display the storage device from which the diagnostic is to be loaded when the LOAD command is issued.

Format

```
DS> SHOW LOAD
```

Example

```
DS> SH L
DMA0:[SYSMAINT]
```

SHOW SECTION

The SHOW SECTION command enables you to display the section names supported by the currently loaded diagnostic.

Format

```
DS> SHOW
```

Example

```
DS> LOAD  ABCDEF
DS> SH SEC
ABCDEF contains the following sections:
DEFAULT, MANUAL, ALL, TEST–AB, TEST–CDEF
```

SHOW SELECTED

The SHOW SELECTED command enables you to display the characteristics of devices that have been selected.

Format

```
DS> SHOW SELECTED qualifier
```

qualifier. This argument enables you to specify the /BRIEF qualifier. TBS.

Example

```
DS> SEL TTA:
DS> SH SEL
output TBS
```

START

The START command causes VDS to pass control to the initialize routine of the diagnostic currently stored in memory. The diagnostic will then begin to execute.

Format

```
DS> START qualifier-list
```

Arguments

qualifier-list. This argument consists of one or more of the optional command qualifiers listed in Table 8-8.

Examples

In the following example, the diagnostic program in memory is started:

```
DS> ST
```

In the following example, the program's manual section is started:

```
DS> ST/SEC:MANUAL
```

In the following example, the tests 32 and 33 are started if they are in the manual section:

```
DS> ST/SEC:MANUAL/TEST:32:33
```

In the following example, tests 6-12 are started:

```
DS> ST/TEST:6:12
```

Table 8-8 START Command Qualifiers

Qualifier	Effect
SECTION:sec-nam	A section is a group of discrete tests. The section identifies function, execution times, and whether or not there is need for user intervention. If a section--name is supplied with the RUN command, the tests contained in that section will be run. If a section-name is not supplied, the tests contained in the section named DEFAULT are run. The diagnostic will contain a list of the available sections, or you may use the SHOW SECTIONS command after loading the diagnostic.
TEST:first:last SUBTEST:num PASSES:count	The TEST qualifier may be used on its own or in conjunction with the SUBTEST and PASSES qualifiers. If the TEST qualifier is specified with the first test and last test arguments, VDS sequentially passes control to the first through last tests inclusively. If the first test argument is combined with the SUBTEST qualifier, execution begins at the first test and concludes at the endof the subtest identified by num. The SUBTEST qualifier may also be combined with the PASSES qualifier to give you a loop-on-subtest capability. The value supplied with the PASSES qualifier must be a decimal in the range 1-0 where 0 is infinity. If the PASSES qualifier is not specified a default of one pass is used. If the TEST qualifier is not specified, all tests in the specified section are executed. If only the first test argument is specified, execution begins at the specified first test and concludes after the highest numbered test in the section.

In the following example, test 9, subtests 1-5 are started:

```
DS> ST/TEST:9/SUBTEST:5
```

In the following example, tests 9-n are started, n is the highest numbered test in the default section:

```
DS> ST/TEST:9
```

In the following example, the default section is started with three passes:

```
DS> ST/PASS:3
```

In the following example, test 9, subtests 1-4 are started, and then test 9, subtest 5 loops indefinitely:

```
DS> ST/TEST:9/SUBTEST:5/PASS:0
```

SUMMARY

The SUMMARY command enables you to print statistical information compiled by a diagnostic. This command may only be used if the diagnostic has a summary section. You would normally use this command after running a pass of a diagnostic, see the RUN command. The format and contents of the report are diagnostic specific.

Format

```
DS> SUMMARY
```

Example

The following example shows how the CTRL/C, SUMMARY and CONTINUE commands are used together to obtain statistics on the program being run and then execution to continue:

```
  .....diagnostic is running.....
  ^C
  DS>SU
  .....statistical report is produced.....
  DS> CO
......diagnostic continues running.....
```

Chapter 9 Q22 Bus Diagnostics

The Q22 bus diagnostic procedures enable you to test the supported Q22 bus devices. These procedures are performed by issuing VAX Diagnostic Supervisor (VDS) commands described in the previous chapter. The following Q22 bus devices can be tested:

- DLV11-J
- DZV11
- RQDX1, RX50, RD51
- QNA

DLV11-J Diagnostic

The DLV11-J Diagnostic (EHxxx - name TBS) is a level 3 diagnostic designed to verify the operation of the DLV11-J four-channel asynchronous interface. This diagnostic is used in standalone mode. This is a functional level diagnostic; after running it you will be able to identify a DLV11-J module that failed.

This diagnostic is compatible with Digital's Automated Product Test (APT)system.

Diagnostic Distribution

Distribution Media

This diagnostic is distributed on the diskette labeled TBS, Part Number TBS

Memory Requirements

This diagnostic requires TBS Kb of memory for installation, including VDS.

Execution Time

Execution time will depend on the line speed selected. It will require TBS minutes (all lines configured at 150 baud). It will require TBS minutes (all lines configured at 38,400 baud).

Setting Up Procedures

All DLV11-J units to be tested must be configured according to the rules defined in *Part Five: System Configuration.* All channels on each DLV11-J module must be configured identically for each of the following options:

- Number of data bits (D jumpers).
- Number of stop bits (S jumpers).
- Parity detection (P jumpers).
- Even/Odd parity (E jumpers).

Special loopback connectors (Part Number TBS) are required to run the complete diagnostic.

Bootstrapping the Diagnostic

This diagnostic is run under the control of VDS. See Chapter 3 for bootstrapping information.

Operation

Before invoking the diagnostic you should attach and select all of the units to be tested, using the ATTACH and SELECT commands. All four channels of each selected DLV11-J will be tested. The following information is required in order to attach a DLV11-J, the first three item are defined as part of the ATTACH command, the other six items are specific to the DLV11-J. The items are supplied in the order shown:

- The device type. This is specified as DLV11J.
- The link type. This is specified as QBUS.
- The device name. This specified as is TTx, where x is either A, B, etc. for successive DLV11-J modules under test.
- Base Control and Status Register (CSR) Address (specified in octal). This is the Q22 bus address of the group of 16 CSR's that the DLV11-J provides for programmed control.

The CSR's for all four-channels must be contiguous. The Seahorse configuration procedures do not allow the fourth channel to be designated as the console. See *Part Five: System Configuration.*

- Base Vector Address (specified in octal). This is the base vector address for the group of 8 interrupt vectors that the DLV11-J provides for programmed control. The vectors for all four-channels on each DLV11-J must be contiguous. The Seahorse configuration procedures do not allow the fourth channel to be designated as the console. See *Part Five: System Configuration.*
- Number of Data Bits Configured (specified as 7 or 8). This value must agree with the settings of the D jumpers for all four channels. 7 is specified if the jumpers are all wirewrapped from X to 0. 8 is specified if the jumpers are wirewrapped from X to 1.
- Number of Stop Bits Configured (specified as 1 or 2). This value must agree with the settings of the S jumpers for all four channels. 1 is specified if the jumpers are all wirewrapped from X to 0. 2 is specified if the jumpers are wirewrapped from X to 1.
- Parity Detection Enabled (specified as 'yes' or 'no'). This value must agree with the settings of the P jumpers for all four channels. 'yes' is specified if the jumpers are all wirewrapped from X to 0. 'no' is specified if the jumpers are wirewrapped from X to 1.
- Even Parity Enabled (specified as 'yes' or 'no'). This value must agree with the settings of the E jumpers for all four channels. 'yes' is

specified if the jumpers are all wirewrapped from X to 1. 'no' is specified if the jumpers are wirewrapped from X to 0.

For example, the following ATTACH command defines the first DLV11-J installed. Note that the device parameters are correct for a device shipped by Digital. VDS will prompt for any missing arguments:

```
DS> ATTACH DLV11J QBUS TTA 776500 300 8 1-
    NO NO
```

The HELP command may be used to obtain information about the diagnostic, and the format of the ATTACH command for the DLV11-J, for example:

```
DS> HELP EHxxx

DS> HELP DEVICE DLV11-J
```

The diagnostic is composed of the following overlapping test sections:

- DEFAULT. The default test section. This section includes the tests contained in the LOOPBACK and REGISTER sections.
- LOOPBACK. Contains the tests that require data loopback connectors be installed on all ports of the DLV11-J modules under test. These connectors should be installed prior to beginning the test.
- REGISTER. Contains the tests that do not require the data loopback connectors be installed in order to function correctly. Note that test coverage is severely limited if the loopback connectors are not installed.

There are no special VDS flag or event flag settings for this diagnostic.

The diagnostic will produce a summary report if the SUMMARY command is issued. Figure 7 illustrates the information listed for each channel.

```
>>>B DU0
%%

DIAGNOSTIC SUPERVISOR.   ZZ-EHSAA-6.12-308   1-OCT-1983  03:16:55

DS> ATTACH DLV11J QBUS TTA 776500 300 8 1 NO NO
DS> SELECT TTA
DS> RUN EHXXX

.. PROGRAM: DLV11-J Functional Test, ZZ-EHXXX, Revision 1.0,   9 tests at 08:55:03.12
Testing _TTA

.. End of run, 0 errors detected, pass count is 1,    time is 11-JAN-1984  09:03:45.64

DS> SUMMARY

Summary of DLV11-J Functional Test, ZZ-EHXXX, Rev. 1.0:

Device      Channel     Total       Total       Parity      Framing     Overrun
Name        Number      Reads       Writes      Errors      Errors      Errors

_TTA        0           2500        2800        0           0           0
            1           2500        2800        0           0           0
            2           2500        2800        0           0           0
            3           2500        2800        0           0           0

DS>
```

Figure 7. DLV11-J Sample. Output

Testing Process

The DLV11-J diagnostic consists of the following tests. The test numbers which have an (L) after them require the data loopback connectors to be installed and are part of the LOOPBACK test section:

1. CSR Addressability Test. This test accesses all 16 device CSR's and verifies that they all respond.
2. CSR Bit Function Test.

3(L). CSR Bit Function Test. This the same as test 2, with the exception that the loopback connector is required.

4. Transmitter Interrupt Logic Test.

5(L). Receiver Interrupt Logic Test.

6(L). Full Speed Data Loopback Test.

7(L). Break Genearation Test.

8(L). Multichannel Interrupt Priority Test.

9(L). Multichannel Full Speed Data Loopback Test.

Tests 1-7 are performed on each channel of each DLV11-J separately. Tests 8 and 9 are performed with all four channels of each DLV11-J operating concurrently.

Error Message Handling

Diagnostic messages output by this diagnostic are in the standard VDS format. See Table 9-1 for a list of the error messages, their meaning and the suggested action.

One possible error that the diagnostic is unable to detect will result in the premature termination of the diagnostic during test 7. This is the BREAK generation test. The symptom of this condition is that after the test has started, the console terminal is forced into console mode, as if the BREAK key had been pressed. The console halt code for this occurrence is 02.

If this type of halt occurs, you should restart the diagnostic with the TRACE flag set, and verify that the halt occurs after test 7 is started. If it does not occur at this point, some other problem is causing the error. If the halt does reoccur, then the DLV11-J board is misconfigured. You must remove the board and correct the setting of the B jumper by removing the jumper at B from X to H.

Table 9-1 DZV11-J Diagnostic Error Messages

Message	Meaning	Action
TBS		

Further Action

If the diagnostic fails to execute properly on a particular DLV11-J module, recheck the jumper settings on the module, especially the B jumper. Also check the ATTACH command input.

DZV11 Diagnostic

TBS

RQDX1, RX50, RD51 Diagnostic

TBS

QNA Diagnostic

TBS

Chapter 10

Guide to Writing a Device Diagnostic

Details TBS.

This chapter will describe Seahorse specific device diagnostic programming issues, for example, interrupts and Q22 bus access. It will address Seahorse specific issues as they affect VDS, for example, the interval clock. It will reference the *VAX-11 Diagnostic Design Guide* (EK-1VAXD-TM-002) for all VDS-related general information.

Chapter 11
Interpreting and Controlling Diagnostic Messages

Diagnostic Messages

Seahorse diagnostic programs provide informative error messages. In most cases these messages will enable you to quickly identify a failing subsystem, function or module. Figure 11-1 (TBS) illustrates a message from a repair level program. Figure 11-2 (TBS) illustrates one from a functional level program.

In each case, these messages will probably give you enough information to isolate the problem. Most users will swap boards that are identified in error after verifying any configuration information, for example, jumpers, switches are set correctly. Occasionally, however, the message may not provide sufficient information for you to isolate the fault. For example, you may replace the module TBS in Figure 11-2 and still get the same message when you rerun the diagnostic. If this occurs, you will have to carry out additional procedures to isolate the fault. The following sections will help you investigate the fault.

Finding the Relevent Documentation

Diagnostic messages always give the failing test number, the subtest number and the error number. This information will provide you with an index to the program documentation.

The documentation for each diagnostic falls into five categories, all are available on microfiche.

- The AIDS problem reports and release notes, EVNDX.
- The documentation file.
- The memory allocation map.
- The header module listing.
- The test module listings.

Documentation File

The documentation files for all VAX diagnostic programs are grouped together on microfiche. Ideally, you should be familiar with the documentation file before running any program. The documentation file includes the following items.

- The program maintenance history.
- The program abstract.
- The requirements necessary for program execution.
- The assumptions concerning previous testing.
- The program operating instructions.
- The program functional description.
- The bibliography.
- The glossary.

To make use of any diagnostic, you must read the requirements, assumptions, operating instructions, and functional description.

Memory Allocation Map

Two parts of the memory allocation map (the link map) are useful for troubleshooting; the program section synopsis, and the symbol cross reference.

The program section synopsis lists the starting and ending address of each program section in the diagnostic. Since each test begins with a new program section, its starting address is given in the program section. Figure 11-3 (TBS) shows part of the link map synopsis for the TBS diagnostic.

The symbol cross reference alphabetically lists the global symbols used in the program. A value is given for each symbol. The symbols may be equivalent to literal values or to addresses. An R beside a symbol indicates that the symbol is relocatable.

Program Header and Test Modules

The program header module contains:

- The module preface.
- The declarations.
- The data.
- The working storage allocation.
- The initialization routine.
- The cleanup routine.
- The summary routine.

Global subroutines may be in the header module or they may be in a separate module. A local symbol table follows the header module, and each of the other program modules.

The test modules follow the header module. They generally contain the following items:

- The module preface.
- The declaration section.
- The test description for each test.
- The code for each test.
- The module summary table and symbol cross reference.

Error Analysis

You should begin analysing the error by reading the description of the failing test in the program documentation file. This will explain the purpose of the test and the methods used. It will also provide a general indication of the nature of the fault. Further analysis of the program is not normally necessary.

Detailed Test and Subtest Descriptions

If you require more information on the test, you must find the location of the microfiche frame number for the listing containing the failing test. The index in the lower right corner of the microfiche sheet will help you to find the right frame. Each test begins with a functional description which normally contains debugging procedures and troubleshooting aids, as well as a step-by-step statement of the test algorithm. You should read this information before proceeding. You must then locate the program code for the failing function by searching the test routine listing for the appropriate subtest and error numbers. Line and block comments explain the function of each line or group of lines.

If still more information concerning the failure is required, you may try the following procedures:

- Produce a loop-on-error by rerunning the program with the LOOP flag set. This procedure requires use of the VDS commands, see Chapter 8.
- Analyze the code. This requires an understanding of the programming language (usually MACRO-32).
- Set breakpoints and step through the test.
- Use EXAMINE and DEPOSIT commands to alter code and data after setting breakpoints..