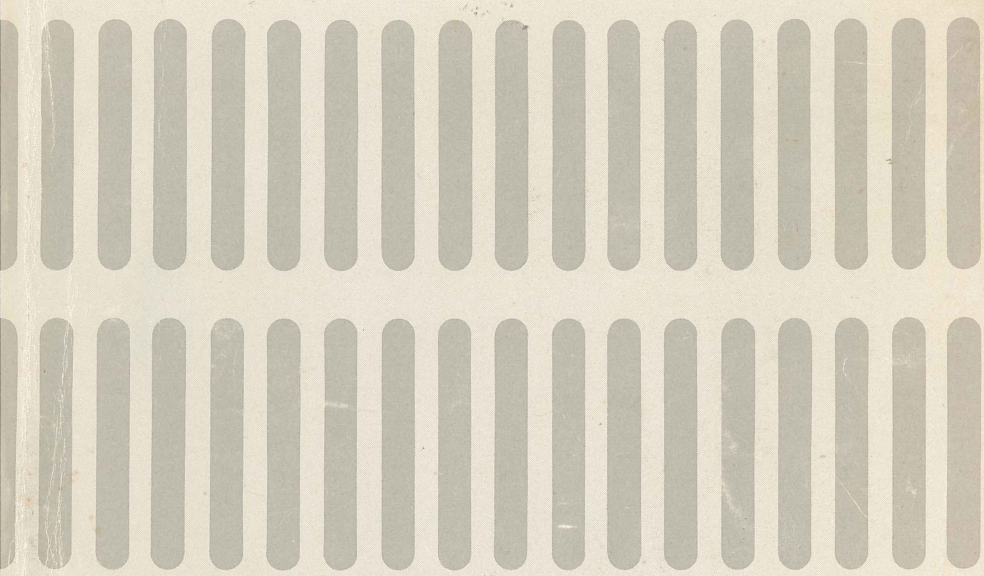
VAX

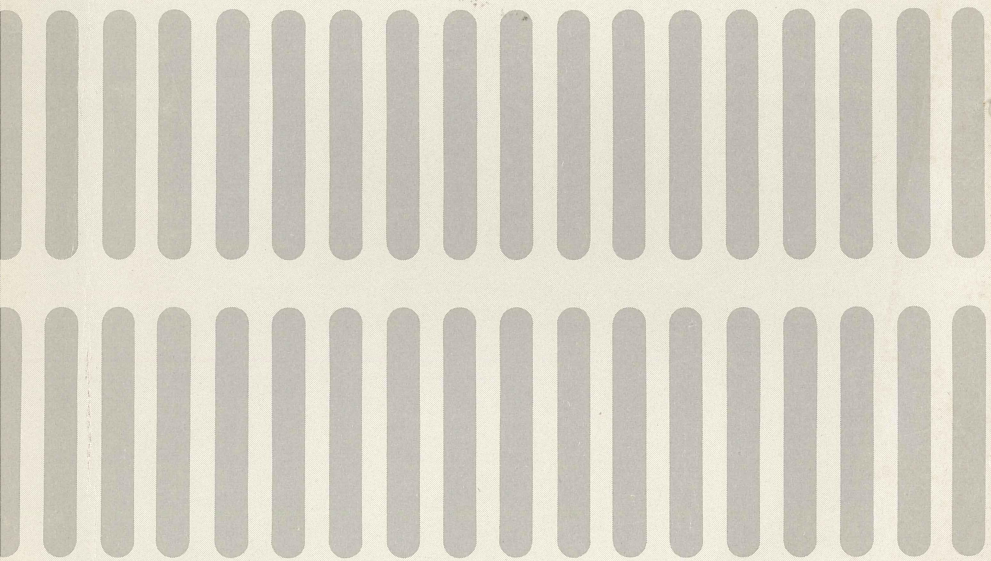
Diagnostic System User's Guide



digital

VAX

Diagnostic System User's Guide



digital

Diagnostic System User's Guide

EK-VX11D-UG-001

Digital Equipment Corporation
Maynard, Massachusetts

First Edition, September 1980

Printed in U.S.A.

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECSYSTEM-20	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	RSTS
UNIBUS	VAX	RSX
	VMS	IAS

Contents

FIGURES

TABLES

1

INTRODUCTION

The VAX diagnostic system is a hierarchy of programs with a wide range of capabilities. The diagnostic system provides field service engineers and customers with a powerful tool for VAX hardware verification, troubleshooting, and maintenance. The system's fault detection and isolation features speed repair time.

The programs range in function from general to specific. The system diagnostic control program (ESXBB) detects failing functions and subsystems. At the other extreme, microdiagnostic programs can isolate faults to field-replaceable units (printed circuit boards or LSI chips). The programs also execute in a range of environments. Some diagnostic programs run simultaneously with user application programs under the VMS operating system. Other diagnostic programs require exclusive use of the VAX computer system.

In addition, the VAX diagnostic system includes several versatile command sets. The commands provide the operator with a variety of selection and execution options.

DIAGNOSTIC SYSTEM STRUCTURE

The diagnostic system hierarchy consists of six program levels.

Level 1 – Operating system (VMS) based diagnostic programs

Level 2R – Diagnostic supervisor-based diagnostic programs restricted to running under VMS only

Peripheral diagnostic programs which are not supported by the supervisor in the standalone mode

System diagnostic control program

Level 2 – Diagnostic supervisor-based diagnostic programs that can be run either under VMS (on-line) or in the standalone mode

Bus interaction program

Formatter and reliability level peripheral diagnostic programs

Level 3 – Diagnostic supervisor-based diagnostic programs that can be run in standalone mode only

Functional level peripheral diagnostic programs
Repair level peripheral diagnostic programs
CPU cluster diagnostic programs

Level 4 – Standalone macrodiagnostic programs that run without the supervisor

Hard-core instruction test (This program tests the basic CPU functions necessary to running the supervisor.)

Console Level – Console-based diagnostic programs that can be run in the standalone mode only

Microdiagnostics
Console program
ROM resident power-up tests

The six program levels operate in the context of four environments: user, system, cluster, and console. These four environments, in turn, run within two operating modes: on-line (under VMS) and standalone (without VMS). Figure 1-1 gives a schematic representation of these relationships.

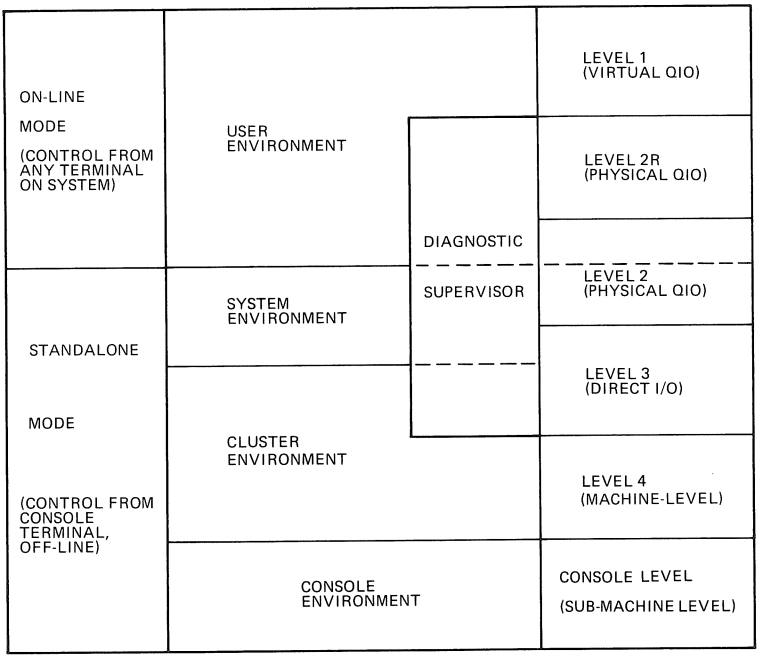


Figure 1-1 Schematic Representation of a VAX Diagnostic System

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 diagnostic programs operate from the hard-core of this 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 plus the area tested in that lower environment. Figure 1-2 shows the building block structure of the diagnostic environments. Notice the overlap between the areas under test in the different environments.

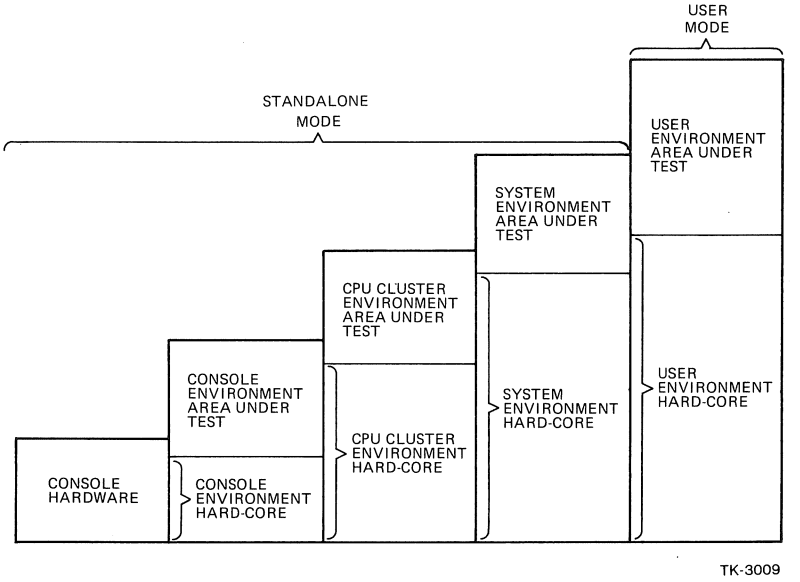


Figure 1-2 The Building Block Structure of the Diagnostic Environment

Console Environment

The console environment operates in the standalone mode only. The operator controls the system from the console terminal. This environment consists of submachine level hardware, software, and firmware. It provides fundamental operator control functions, system programmer debugging functions, and basic machine diagnostic functions. Figure 1-3 shows the console environment configuration. The console hardware forms the hard-core that must be fault-free in order to run the microdiagnostics. Notice that the CPU microcode remains untested in the console environment.

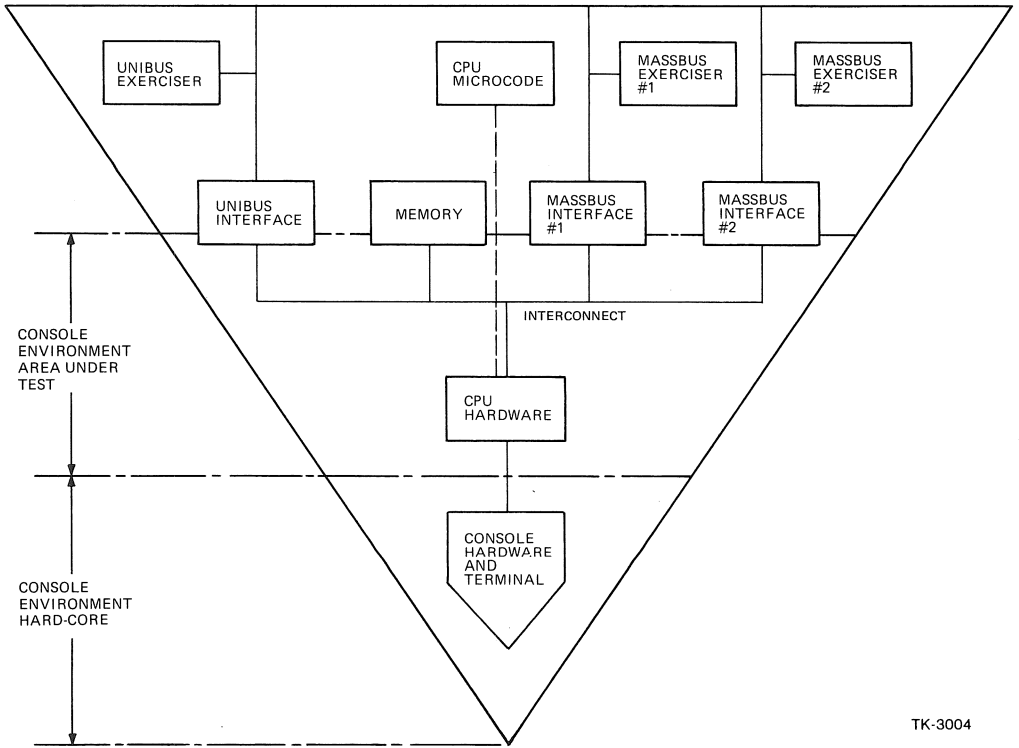


Figure 1-3 Console Environment

From a diagnostic strategy standpoint, the console environment is the most basic, most implementation-specific piece of the diagnostic system. It ranges from the extensive capability and functionality of the VAX-11/780 console (LSI-11 subsystem) to totally ROM-based quick verify tests in lower priced VAX CPUs.

CPU Cluster Environment

Like the console environment, the CPU cluster environment operates in the standalone mode only. The operator must use the console terminal. This environment consists of the console environment plus the machine level components (complete CPU, memory, I/O channels) that support standalone, macro level program execution. Figure 1-4 shows the CPU cluster environment configuration. 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 4 and level 3 diagnostic programs. In a building block fashion, these programs 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 channel loopback capability.

System Environment

The system environment also operates only in the standalone mode. The operator must use the console terminal. This environment contains a wide spectrum of diagnostic programs ranging from level 3 repair diagnostics through level 2 (QIO) device exercisers.

The VAX system environment level diagnostic strategy is to implement a series of level 3 repair diagnostic programs and level 2 functional diagnostic programs 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 2) to media reliability, acceptance, and multidevice exercisers (level 2).

Figure 1-5 shows the system environment configuration in a typical VAX system. The hardware tested in the CPU cluster environment forms the hard-core for the system environment.

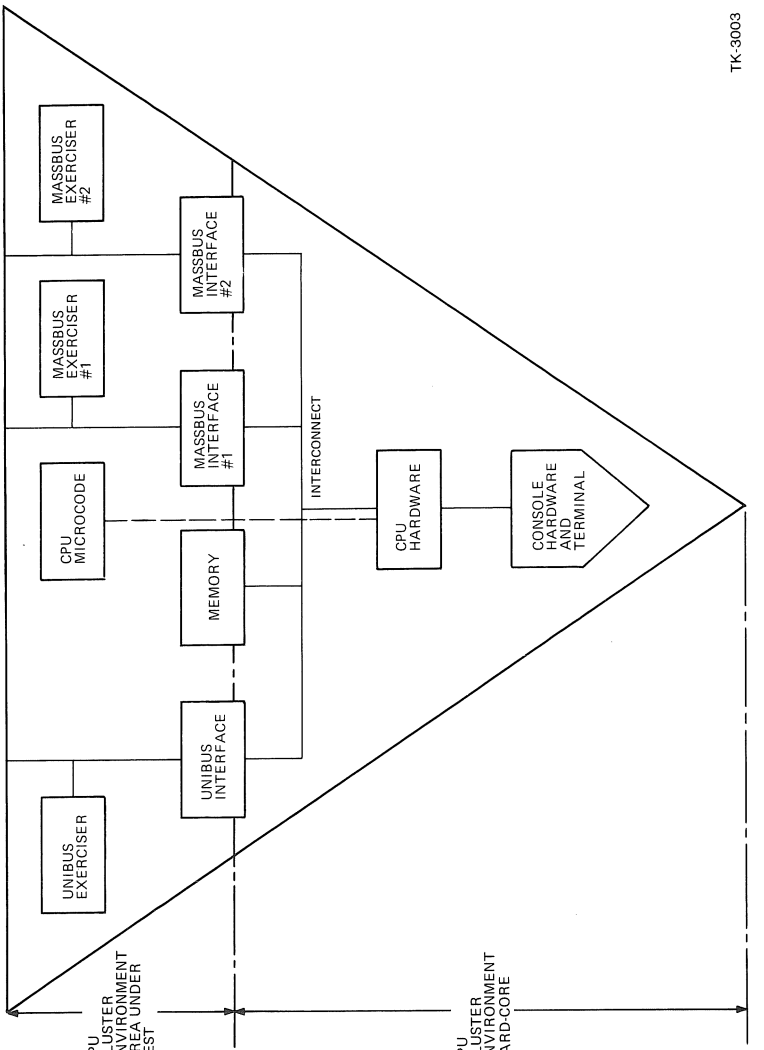


Figure 1-4 CPU Cluster Environment

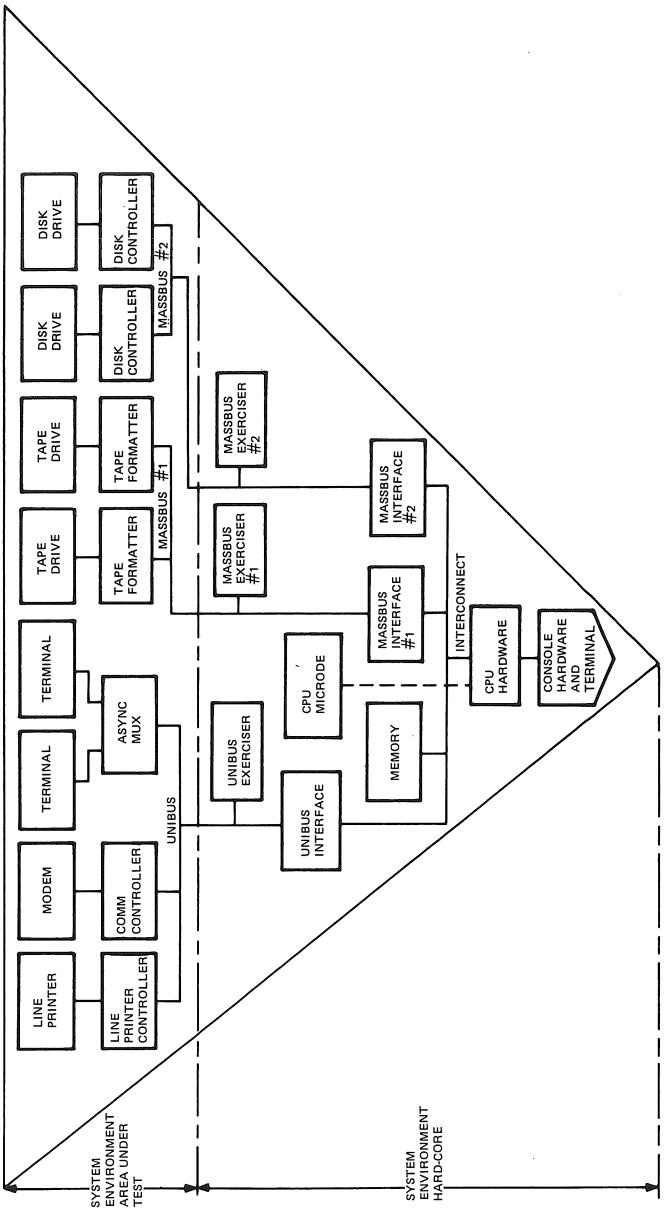


Figure 1-5 System Environment

User Environment

The VAX user environment operates in the on-line mode, under VMS. The operator can control the diagnostic process from any terminal on the system, including the console terminal. The user environment includes the level 2 diagnostic programs, which run in the system environment, as well as the level 2R programs, which do not. Many of the diagnostic programs that run in the user environment will run simultaneously with user application programs. However, some, like the system diagnostic control program, require exclusive use of the computer system.

Program Load and Control Sequences

The VAX diagnostic system provides some flexibility concerning the load paths and execution control of different program levels. For example, level 2 and level 3 programs can be loaded from the console load device or from the system disk. Microdiagnostic programs, on the other hand, are usually loaded from the console load device. Although level 2 programs are flexible and will run in user or standalone mode, level 2R, 3, and 4 programs are less flexible. Figure 1-6 shows the load and control sequences and operating modes for the VAX diagnostic system.

DIAGNOSTIC STRATEGY

The wide range of diagnostic programs and program levels in the VAX diagnostic system provides users with flexibility in fault isolation procedures. Detailed knowledge of the computer and the diagnostic system will enable you to troubleshoot effectively.

However, if you are troubleshooting a machine where a failure is not obvious, you should use on-line tools as a first step, when possible. SYE, the system error log program, helps you to analyze the recent performance of the machine. SDA, the system dump analyzer, helps you to analyze VAX system crashes. UETP provides a confidence check for the entire VAX system (Chapter 7). And on-line diagnostic programs help you to identify the failing subsystem. Once you know what subsystem is at fault, run level 3 programs and then level 2 programs (bottom up approach) to isolate the failure to a field-replaceable unit.

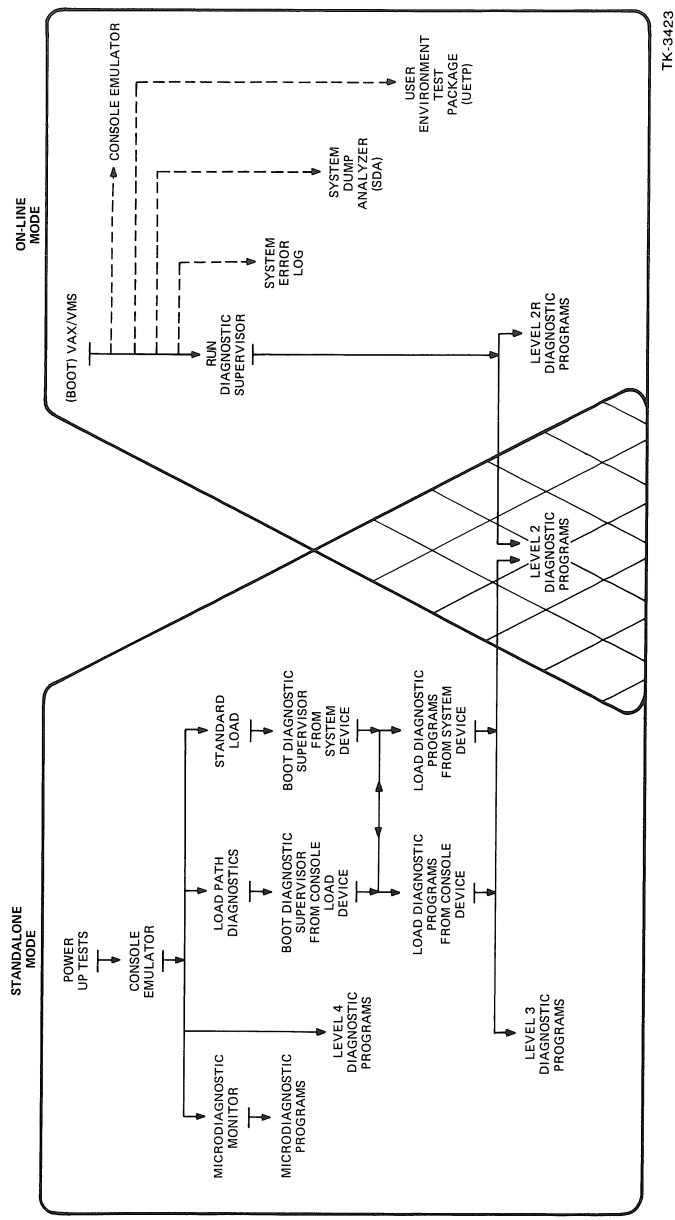


Figure 1-6 VAX Diagnostic System Load and Control Sequences

If you cannot run VMS or boot the diagnostic supervisor from a system disk, try booting the supervisor and loading programs from the secondary load path (the console load device). If this also fails, run microdiagnostics or level 4 programs, as appropriate. See the appendix for detailed troubleshooting guidelines.

DIAGNOSTIC FILE MAINTENANCE

DIGITAL Field Service is responsible for building and maintaining the diagnostic directory (SYSMAINT) on VAX systems for customers who have a maintenance contract. Customers who have bought their own diagnostic system licenses must install and maintain the diagnostic files themselves.

Build the SYSMAINT directory when you install the system. Keep the file up to date by transferring new versions of programs to the SYSMAINT directory as you receive them. DIGITAL distributes diagnostic updates on magnetic media periodically. In some cases VMS indirect command files may be provided to aid in the installation and updating of SYSMAINT. See the diagnostic system overview manual (the second level of documentation shown in Figure 1-7) that refers to your VAX system for details on SYSMAINT build and update procedures.

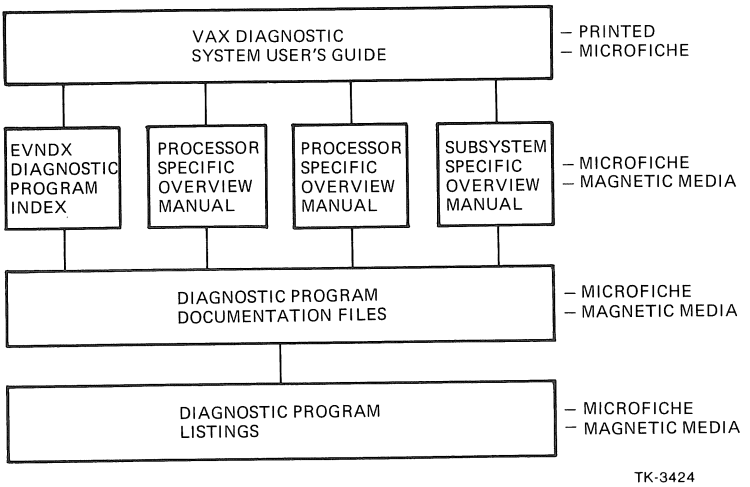


Figure 1-7 VAX Diagnostic System Documentation

DIAGNOSTIC DOCUMENTATION HIERARCHY

Digital supplies four levels of documentation for the VAX diagnostic system, as shown in Figure 1-7.

This manual, the *VAX Diagnostic System User's Guide*, contains stable information that applies across all VAX implementations. Since this manual is general, it provides no information on processor-specific diagnostic features such as microdiagnostic programs or diagnostic and operating system bootstraps. This manual is available in hard copy and on microfiche.

Processor and subsystem-specific diagnostic overview manuals provide the details not covered in this VAX manual. Explanations of the console command language, microdiagnostic program use, and boot procedures are included in the processor and subsystem-specific overview manuals. The overview manuals are available on microfiche. DIGITAL updates the overview manuals periodically to reflect changes in the hardware and diagnostic programs.

A documentation file for each VAX diagnostic program is available on microfiche and magnetic media. These documentation files are grouped together on several microfiche sheets. Each documentation file contains the following information categories.

- coversheet
- table of contents
- program maintenance history
- program abstract
- program execution requirements
- assumptions
- operating instructions
- program functional description
- bibliography
- glossary

The diagnostic program listings are available on microfiche and magnetic media. The programs are organized into tests and subtests. Each

test is preceded by a test description in the listing. See Chapter 6 of this manual for details. Changes in the programs are reflected in the listings and distributed periodically.

Table 1-1 lists other related manuals that are available in hard copy.

Table 1-1 Related Documentation

Title	Document Number
VAX/VMS Documentation Kit	QE001-GZ
VAX Diagnostic Design Guide	EK-1VAXD-TM
VAX-11 Architecture Handbook	EB-17580
VAX-11 Software Handbook	EB-15485
PDP-11 Peripherals Handbook	EB-07667
Terminals and Communications Handbook	EB-15486
BLISS-32 Language Guide	AA-H019A-RE

EVNDX, VAX DEVELOPMENT MAINDEC INDEX

DIGITAL distributes EVNDX, the VAX Development MAINDEC Index, on magnetic media and microfiche. EVNDX enables users to keep up with the periodic changes and additions to the VAX diagnostic system. Once you are familiar with the format and content of this index, a quick review of each new release will show you what's new and what changes may be important for your VAX system. The general format for EVNDX follows.

- Coversheet
- Table of Contents

- Introduction

  Media descriptor. Codes and part numbers for magnetic media containing executable programs and documentation.

  Program code naming convention

- VAX Diagnostic Program Codes

  Revision number

  Update status. One asterisk means that the revision listed is new; two asterisks indicate a new program.

  Program level (2R, 2, 3, or 4)

  Program title

- Magnetic Media MAINDEC Codes

  Revision number

  Update status. One asterisk means that the revision listed is new; two asterisks indicate a new disk or tape.

- AIDS Problem Reports

  Reports of problems encountered by program users and solutions provided by DIGITAL Diagnostic Engineering, where applicable.

- Release Notes

  General information on the VAX diagnostic system

  Notes on revisions to diagnostic programs

- VAX Diagnostic Media Contents

  A list of files contained on each diagnostic disk and tape.

- VAX Diagnostic Hardware Option Kit List

  A list of diagnostic media packages arranged for specific VAX hardware configurations. Each option list contains a part number, a MAINDEC code, and a title for each disk and tape in the kit.

- ECO/DECO Cross Reference History File

  A list of hardware revisions and compatible diagnostic program revisions.

REMOTE DIAGNOSIS

Customers who have bought VAX remote diagnosis contracts and have remote diagnosis option kits installed should call a DIGITAL Diagnostic Center (DDC) when they suspect hardware failures. The dispatcher at the DDC will provide customers with the information necessary to proceed with the remote diagnostic session. See the appropriate remote diagnosis manual for details.

2

CONSOLE COMMAND LANGUAGE

All VAX computer systems include an ASCII console which provides an interface between an operator at a terminal or an automatic test system and the:

- CPU hardware
- CPU microcode
- instruction set processor (macro level) software.

The console interprets commands typed on the terminal. And it performs appropriate operations for each command by means of the console software or the CPU microcode.

The console normally operates in one of two modes. When the console is in the console I/O mode, it interprets all console terminal output in order to perform the lights, switches, and maintenance functions, and to implement the console command language. When the console is waiting for input in the console I/O mode, it displays the symbol >>> as a prompt on the terminal. When the console is in the program I/O mode, it functions as a user terminal, passing characters between the operator's keyboard and the software operating in the instruction set processor (ISP).

The console is an important diagnostic tool. It gives the operator visibility into and control of memory, processor registers, and I/O registers. It also enables the operator to start and stop the CPU instruction set processor and to initialize the CPU.

COMMAND DESCRIPTION TERMS AND SYMBOLS

Table 2-1 defines the special symbols used to describe the syntax of the console commands.

Table 2-1 Term and Symbol Definitions

Term/Symbol	Definition
< >	Denotes a category name. For example, the category name <address> represents a valid address.
[ ]	Indicates the part of an expression that is optional.
<SP>	Represents one or more spaces or tabs.
<address>	Represents an address argument (hexadecimal).
<data>	Represents a numeric argument (hexadecimal).
<qualifier-list>	Represents a list of command modifiers (switches).
<CR>	Represents a console terminal carriage return.
/	Delimits a command from its qualifiers.

CONSOLE COMMAND QUALIFIERS

VAX console commands take two types of qualifiers: data length and address. Table 2-2 lists the data length qualifiers. Table 2-3 lists the address type qualifiers.

Table 2-2 Data Length Qualifiers

Qualifier	Meaning
/B	byte data length (8 bits)
/W	word data length (16 bits)
/L	longword data length (32 bits) (this is the default length qualifier following initialization)

Table 2-3 Address Type Qualifiers

Qualifier	Meaning
/G	general register address space
/I	internal processor register address space
/P	physical address space (this is the default address type qualifier following initialization)
/V	virtual address space

CONSOLE COMMANDS COMMON TO ALL VAX SYSTEMS

Seven standard console commands are common to all VAX systems. Detailed explanations of these comands follow. The radix (base) for all values in console commands is hexadecimal. In the examples that follow, operator input is printed in color.

NOTE

See the appropriate processor-specific overview manual for an explanation of the complete command set for any VAX processor.

CTRL/P Command

^P

This command causes the console to enter the console I/O mode. On some VAX implementations CTRL/P halts the processor.

When the console is in the console I/O mode, the console fields characters typed on the console terminal. If the console is already in the console I/O mode, CTRL/P causes the console to display another input prompt, >>>. CTRL/P is normally used when the central processor is running.

```
                                ! Processor is running.
^P                              ! Enter console I/O mode.
>>>                             ! Console input prompt.
 .
 .
 .
>>>C                            ! Continue with the interrupted
                                ! Process.
```

Example 2-1 CTRL/P Command

Continue Command

C<CR>

The continue (C) command causes the CPU instruction set processor (ISP) to begin execution at the address currently contained in the program counter (PC). If the CPU is already running, it will continue running in response to a continue command. Continue does not initialize the CPU.

The console enters the program I/O mode after issuing the continue command to the ISP. You may use the continue command to return the console to the program I/O mode even if the CPU is already running.

```
>>>C                          ! Console input prompt.
                             ! The operator types C and
                             ! carriage return.  The console
                             ! enters the program I/O
                             ! mode.
<CR>                         !
Username:                    ! Operating system prompts for
                             ! login.
```

Example 2-2 Continue Command

Deposit Command

D[<qualifier-list>]<SP><address><SP><data><CR>

The deposit (D) command accepts the following qualifiers (Tables 2-2 and 2-3):

/B, /W, /L, /P, /V, /G, /I

This command writes <data> into the <address> specified.The address space used (physical, virtual, internal register, or general register) depends on the qualifier given. Note that the deposit virtual command (D/V) will not work unless mapping is set up for the virtual address referenced.

If no qualifiers are given, the current address type default determines the address space to be used. The data length used will be the current default, unless the operator specifies a data length qualifier (byte, word, or longword). If the default data length or the data length specified by a qualifier is shorter than the number of digits typed as <data>, the console will respond in one of two ways, depending on the VAX processor implementation. The console will either ignore the command and issue an error message, or it will use only the low-order digits in the assembled data quantity.

```
>>>D/W/P FE 4C09              ! Deposit the hexadecimal
                             ! word value 4C09
                             ! in the physical
                             ! location FE.
>>>
```

Example 2-3 Deposit Command

In addition, certain symbolic addresses are accessible to both the deposit and the examine commands on some VAX processors, as shown in Table 2-4.

Table 2-4 Symbolic Addresses for Use with Deposit and Examine

Symbol	Function
PSL	Reference the processor status longword
PC	Reference the program counter
SP	Reference the current stack pointer
Rn	Reference general register n, where n is a decimal number from 0 to 15.
+	Reference the location immediately following the last location referenced. For physical and virtual references, the location referenced will be the last address plus n. n=1 for byte, n=2 for word, and n=4 for longword. For all other address spaces n is always equal to 1.
−	Reference the location immediately preceding the last location referenced.
*	Reference the location last referenced.
@	Reference the location represented by the last data examined or deposited.

Examine Command

E[<qualifier-list>][<SP><address>]<CR>

The examine (E) command accepts the following qualifiers (Tables 2-2 and 2-3):

/B, /W, /L, /P, /V, /G, /I

The examine command causes the console to display on the terminal the contents of the specified <address>. If you do not specify an address,

the console displays the contents of the current default address. Examine accepts the symbolic addresses listed in Table 2-4.

```
>>>E/L/P 6B                      ! Examine the longword at
    P 0000006B 58455359         ! physical address 6B.
>>>E                            ! Examine the next longword,
    P 0000006F 59535D45         ! the default.
>>>E +                          ! Examine the next longword.
    P 00000073 4F4F4253         ! +,-, and * are not available
                                ! on all VAX-11 processors
>>>E -                          ! Examine the longword preceding
    P 0000006F 59535D45         ! the last longword referenced.
>>>E *                          ! Examine the last referenced
    P 0000006F 59535D45         ! location.
>>>D * FC                       ! Deposit FC in the last
                                ! referenced location.

>>>E @                          ! Examine the location
    P 000000FC 0000043A         ! represented by the data
>>>                             ! last referenced.
```

Example 2-4 Examine Command

Halt Command

H<CR>

The CPU instruction set processor (ISP) will stop after completing the instruction being executed when the console presents the halt (H) request to the CPU. If the CPU is running and the console is in the console I/O mode (note that this is not possible on all VAX processors), you must type H before you can execute some of the other console commands, including initialize (I) and binary load/unload (X).

```
                                ! CPU is running.  Console
                                ! is in program I/O mode.
^P                              ! Enter console I/O mode.

>>>H                            ! Halt the CPU.

>>>
```

Example 2-5 Halt Command

Initialize Command

I<CR>

This command causes the console to initialize the CPU system. You must halt the CPU before you can use initialize.

```
>>>I                          ! The initialize command
                              ! is given before the
                              ! CPU has been halted.

          ?15                 ! Error message: Illegal
                              ! command while CPU was running.
>>>H                          ! Halt the CPU.

>>>I                          ! Initialize the CPU.

>>>
```

Example 2-6 Initialize Command

Binary Load/Unload Command

This command loads or dumps a block of binary data. It is intended for down-line or up-line data transfers in manufacturing environments only.

CONSOLE ERROR MESSAGES

If the console encounters an error (in a command, in the hardware, or in the software) which prevents execution of a specified function, the console will print an error message. All error messages have the following format.

?|<error number>||<sp><message text>|<CR><LF>

The error number is a two-digit hexadecimal number. Table 2-5 lists the numbers and their meanings. The words in upper case letters show the message text.

Table 2-5 Console Error Codes

Number	Meaning
01	SYNTAX ERROR. The console does not recognize this command.
10	ILLEGAL GPR. An invalid general register number was typed.
11	ILLEGAL IPR. An invalid internal register number was typed.
15	ILLEGAL COMMAND WHILE CPU RUNNING.
19	INVALID DEFAULT NEXT ADDRESS. Attempt to wrap around from SP to R0.
20	EXECUTION ERROR. A non-existent memory location may have been referenced. Or some other error may have been encountered during a command execution.
21	HALT TIMEOUT. The CPU did not respond to a halt request within the alloted time.
30	APT CHECKSUM ERROR. The checksum received either following an X command or following the data did not match the expected value.
33	UNRECOGNIZED BOOT DEVICE.
81	CONSOLE INSTRUCTION TEST FAILURE.
82	CONSOLE ROM1 CHECKSUM FAILURE.
83	CONSOLE ROM2 CHECKSUM FAILURE.
84	CONSOLE ROM3 CHECKSUM FAILURE.
85	CONSOLE RAM DATA TEST ERROR.
86	CONSOLE RAM ADDRESS TEST FAILURE.

HALT CODES

When the instruction set processor halts, the console will type out a one- or two-digit hexadecimal code. Table 2-6 lists the halt codes and their meanings.

Table 2-6 Instruction Set Processor Codes

Code	Meaning
0	A halt command was executed from the console.
2	A CTRL/P from the console automatically halted the CPU.
4	Interrupt stack not valid, or unable to read SCB.
6	Halt MACRO-32 instruction was executed with PSL <CUR MODE>=0.
7	An SCB vector was encountered with bits <1:0>=3.
8	An SCB vector with bits <1:0>=2 was executed but no WCS is present.
9	Undefined.
A	A change mode was attempted while the interrupt stack was active.
B	A change mode was attempted and SCB vector<1:0> not equal to 0.

3

LEVEL 4 DIAGNOSTICS

Level 4 diagnostics are macro level (instruction set processor level) diagnostic programs that run without the diagnostic supervisor, in the stand-alone mode. The load, control, and error reporting features of level 4 programs are primitive. Although level 4 programs are excellent trouble-shooting tools, they are more difficult to use than microdiagnostic programs and level 2, 2R, and 3 programs. In general, therefore, you should run level 4 programs only when you are sure that you cannot obtain the same error information from other types of programs.

Level 4 programs always load at physical address 0 and start at physical address 200. See the appropriate processor-specific diagnostic system overview manual for an explanation of how to run level 4 diagnostic programs.

4

DIAGNOSTIC SUPERVISOR COMMANDS

The diagnostic supervisor provides a context for running level 3, 2, and 2R diagnostic programs. Run the supervisor in the standalone mode to execute level 3 programs. Run it in the on-line mode (under VMS) to execute level 2R programs. Level 2 programs will run under the supervisor when it is in either mode.

The supervisor provides nondiagnostic services, such as message control, to diagnostic programs. And it handles processor-specific features of the VAX system so that nearly all level 3, 2, and 2R diagnostic programs can be run on any VAX processor.

The supervisor implements a set of commands and control flags that allow the operator to control diagnostic program loading and execution. In order to make good use of the VAX diagnostic system the operator should be familiar with the supervisor commands and with the diagnostic programs applicable to his/her computer system.

DIGITAL intends to enhance the supervisor functions and broaden the command set in future diagnostic releases. See the release notes in EVNDX, the VAX Development MAINDEC Index for details.

The diagnostic supervisor commands are grouped in four sets.

Program and test sequence control
Scripting features
Execution control
Debug and utility features

Commands, switches, and literal arguments can be abbreviated to the minimum number of characters necessary to retain their unique identity. For example, the load command can be specified by a single L, whereas the start command requires a minimum of ST.

In the command descriptions which follow, certain special characters are employed that require some explanation. Angle brackets, <>, are used to enclose symbolic arguments that are satisfied by a numeric expression or character string. Optional arguments are enclosed by square brackets, []. An OR function is indicated with an exclamation point, !. Literal arguments such as ALL, OFF, and FLAGS are capitalized.

Use the hyphen, -, as a continuation character at the end of a line to continue a command from one line to the next. Use an exclamation point, !, to separate a comment from a command in a command line.

In the examples that follow, operator input is printed in color.

PROGRAM AND TEST SEQUENCE CONTROL COMMANDS

These commands enable the operator to select programs and portions of programs and to control the sequence of test execution.

Set Load Command

SET LOAD<device>:[directory]<CR>

The set load command establishes the storage device from which the supervisor will load diagnostic programs. Initially the default load device is the device from which the supervisor was booted. Use set load when you wish to load diagnostic programs from a different device. Set load

establishes a new default. Use the set load command in combination with the load command or the run command.

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

DS>SET LOAD DMA0:[SYSMAINT]
DS>RUN ESDXA
```

Example 4-1 Set Load Command

NOTE

The directory name, and the square brackets around it, are necessary in the set load command.

Show Load Command

SHOW LOAD<CR>

The show load command causes the supervisor to display the names of the storage device and directory from which diagnostic programs are to be loaded when the load command is given.

```
DS>SHOW LOAD
DMA0:[SYSMAINT]
DS>
```

Example 4-2 Show Load Command

Load Command

LOAD<file-spec><CR>

This command loads the specified file into main memory from the default load device. The default file extension is .EXE. The storage device from which the program is loaded is the device established on the previous set load command. Note that you need supply only the five-letter code that identifies each diagnostic program for the command line argument <file-spec>.

```
DS> LOAD EVTAA                  ! Load the local terminal
DS>                             ! diagnostic program.
```

Example 4-3 Load Command

Attach Command

ATTACH<UUT-type><link-name><generic-device-name>...<CR>

Before starting a diagnostic program, the operator must use several attach commands to define each unit under test (UUT). You must also define for the supervisor the devices that link it to the system bus. If you are testing several units at once, repeat the attach command for each device. Every unit under test is uniquely defined by a hardware designation and a link. See the explanation of the attach command in the diagnostic system overview manual that applies to your VAX system.

The first parameter, <UUT-type>, is the hardware designation of the unit under test. For example, RH, TM03, TE16, and DZ11 are hardware designations.

The second parameter, <link-name>, is the name of the piece of hardware that links the unit under test, in most cases through intermediate links, to the main system bus. For example, an RH is linked to the interconnect; an MTa is linked to an RH; a TU45 is linked to an MTa; and a DZ11 is linked to a DWn. You must attach each piece of hardware (with the exception of the main system bus) before you can use it as a link in an attach command.

The third parameter is the generic device name, which identifies to the supervisor the particular unit to be tested. Use the form <GGan> for the device name.

<GG> is a two-character generic device name (alphabetic).

<a> is an alphabetic character, specifying the device controller.

<n> is a decimal number in the range of 0–255, specifying the number of the unit with respect to the controller.

Use the unit number, <n> or <a>, only if it is applicable to the device. You must supply additional information for some types of hardware to enable the diagnostic program to address the device. For example, you must supply the TR and BR numbers for an RH780, the controller number for a TM03, and the CSR vector and BR for a UNIBUS device. If you do not include additional information, but the information is necessary, the supervisor will prompt you for it.

Select Command

SELECT <generic-device-name>[:],-<CR>
[<generic-device-name>[:] . . .] !ALL<CR>

You must select each unit to be tested with the select command, after attaching it. For each unit, supply the appropriate generic device name, as shown in the appropriate processor-specific overview manual. The select command adds the specified device to the list of units to be tested. The command takes effect the next time the diagnostic program is started.

```
DS> SELECT   TTA:
DS>
```

Example 4-4 Select Command

Deselect Command

DESELECT <device>[:][, <device>[:] . . .] ! ALL<CR>

Use the deselect command to remove the name of one or more devices from the list of units to be tested.

```
DS> DESELECT   TTA:
DS> DESELECT   ALL
DS>
```

Example 4-5 Deselect Command

Show Device Command

SHOW DEVICE <device>[:][,<device>[:] . . .]<CR>

The show device command causes the supervisor to display the characteristics of the specified devices on the operator's terminal. If you omit the device name, the supervisor will list the characteristics of all attached devices (Example 4-6).

Show Selected Command

SHOW SELECTED<CR>

The show selected command causes the display of information in the same format as the show device command. However, the information is shown only for the devices that have been previously selected.

```
DS> SHOW DEVICE
_DW0   DW780        60006000   TR=3. BR=4. NUMBER=0.
_DMA   RK611  _DW0 6013FF20   CSR=00000777440(O) VECTOR=00000000210(O) BR=5.
_DMA0  RK07   _DMA 00000000
_TTA   DZ11   _DW0 6013E050   CSR=00000760120(O) VECTOR=00000000320(O) BR=4.

DS> SHOW SELECTED

DS> SELECT TTA:
DS> SHOW SELECTED
_TTA   DZ11   _DW0 6013E050   CSR=00000760120(O) VECTOR=00000000320(O) BR=4.
DS> DESELECT TTA:
DS> SHOW SELECTED
DS>
```

Example 4-6 Show Device and Show Selected Commands

Start Command

START [/SECTION:<section-name>]-<CR>
[/TEST:<first>[:<last>!/SUBTEST:<num>]]-<CR>
[/PASSES:<count>]<CR>

The start command causes the diagnostic supervisor to pass control to the initialize routine in the diagnostic program in memory, thus beginning program execution.

Each diagnostic program is organized in discrete tests. The tests are grouped in sections, according to their functions, execution times, and whether or not there is need for operator interaction.

If the start command is given without switches, the program will run the tests in the default section. In other words, the initial setting for SECTION is DEFAULT. The supervisor calls only those tests that have been designed by the diagnostic engineer to run in the default section. Default section tests do not require operator intervention. When a section is selected in conjunction with the start command, only the tests that it contains will be executed.

The TEST switch is used in two distinctly different ways. If the *first* and *last* arguments are specified, the supervisor sequentially passes control to tests *first* through *last*, inclusively. If the *first* argument is combined with the SUBTEST switch, program execution begins at the beginning of the *first* test and terminates at the end of the subtest *num*. If the SUBTEST switch is used in conjunction with the PASSES switch, the operator is provided with a loop-on-subtest capability. In this case, only the subtest named in the command line is executed, once looping begins.

If the TEST switch is not specified, all tests within the named section of the program are executed. In other words, the default for TEST is TEST *a* through TEST *n*, where TEST *n* is the highest numbered test in the section. If only the first argument is specified with the TEST switch, the *last* argument is assumed by default to be the highest numbered test within the selected section of the program.

Tests are run only if they are included in the section named. If the PASSES switch is not used, the default value is 1. Test and pass numbers are decimal. The minimum value for passes is 1. The maximum value is 0, which means infinity in this context.

For example:

```
DS> START                          ! Start execution of the diag-
                                   ! nostic program in memory.

DS> START/SEC:MANUAL               ! Start execution of the manual
                                   ! section of the program.

DS> START/SEC:MANUAL/TEST:32:33    ! Run tests 32 and 33 if they
                                   ! are in the manual section.
                                   ! Some tests may not be executed
                                   ! unless the section is speci-
                                   ! fied.

DS> START/TEST:6:12                ! Run tests 6, 7, 8, 9, 10, 11,
                                   ! 12.

DS> START/TEST:9/SUBTEST:5         ! Run test 9, subtests 1, 2, 3,
                                   ! 4, 5.

DS> START/TEST:9                   ! Run tests 9 through N, where
                                   ! N is the last test in the de-
                                   ! fault section.

DS> START/PASS:3                   ! Run 3 passes of the default
                                   ! section.

DS> START/TEST:9/SUBTEST:5/PASS:0  ! Execute test 9, subtests 1, 2,
                                   ! 3, 4, and then loop on sub-
                                   ! test 5 indefinitely.
```

Example 4-7 Start Command

Run Command

RUN <file-spec>[/SECTION:<section name>]-<CR>
[/TEST:<first>[:<last>!/SUBTEST:<num>]]-<CR>
[/PASSES:<count>]<CR>

Run is the equivalent to a load and start command sequence. The run command switches are identical to those in the start command.

For example:

```
DS> RUN EVTAA                                  ! Load and run the local
                                               ! terminal diagnostic.

DS> RUN EVTAA/SEC:MANUAL                       ! Load the local terminal
                                               ! diagnostic and run the
                                               ! manual section.

DS> RUN EVTAA/SEC:MANUAL/TEST:32:33            ! Load the local terminal
                                               ! diagnostic and run tests
                                               ! 32 and 33 in the manual
                                               ! section.

DS> RUN EVTAA/TEST:6:12                        ! Load the local terminal
                                               ! diagnostic and run tests
                                               ! 6, 7, 8, 9, 10, 11, 12.

DS> RUN EVTAA/TEST:9/SUBTEST:5                 ! Load the local terminal
                                               ! diagnostic and run test 9,
                                               ! subtests 1, 2, 3, 4, 5.

DS> RUN EVTAA/TEST:9                           ! Load the local terminal
                                               ! diagnostic and run tests 9
                                               ! through N, where N is the
                                               ! last test in the default
                                               ! section.

DS> RUN EVTAA/PASS:3                           ! Load the local terminal
                                               ! diagnostic and run three
                                               ! passes of the
                                               ! default section.

DS> RUN EVTAA/TEST:9/SUBTEST:5/PASS:0          ! Load the local terminal
                                               ! diagnostic, execute test
                                               ! 9, subtests 1,2,3,4, and
                                               ! then loop on test 9, sub-
                                               ! test 5 indefinitely.
```

Example 4-8 Run Command

Summary Command

SUMMARY<CR>

This command causes the execution of the program's summary report code section, which prints statistical reports. Note that this command is generally used only after running a pass of a diagnostic program. However, the summary command can be used at any time, and would be useful, for example, when the disk reliability program is run. Type CTRL/C first to return control to the command line interpreter (CLI).

Then type SUMMARY to obtain a statistical report on the program. CONTINUE can be typed at this point, if the operator wishes to resume program execution.

CTRL/C

Normally CTRL/C returns control from a diagnostic program to the command line interpreter in the diagnostic supervisor. The supervisor then enters a command wait state and displays the DS> prompt on the operator's terminal. The operator may then issue any valid command. CTRL/C is the only diagnostic supervisor command that may be issued while a program is running. When a diagnostic program is running in conversation mode, CTRL/C returns control to a command interpreter within the program for the conversation mode.

Continue Command

CONTINUE<CR>

This command causes program execution to resume at the point at which the program was suspended. This command is used to proceed from a breakpoint, error halt, summary, or CTRL/C situation.

The following example shows how CTRL/C, summary, and continue can be used together to obtain statistics on the program being run and to then resume execution.

```
                          ...Program is running...

 ^C
DS>  SUMMARY                    ! Operator types CTRL/C.
                                ! Supervisor prompt.
                                ! Operator requests
                                ! statistical report.

                              Statistical
                                Report
                             Printed Here

DS>  CONTINUE                   ! Supervisor prompt.
                                ! Operator requests
                                ! resumption of program.

                        ...Program is running...
```

Example 4-9 Use of CTRL/C, Summary, and Continue Commands

Abort Command

ABORT<CR>

This command passes control to the program's cleanup code and then returns control to the supervisor, which enters a command wait state and displays the supervisor prompt, DS>. At this point the operator may issue any command except CONTINUE. Example 4-10 shows how the abort command can be used together with CTRL/C and the summary command.

```
                          ...Program is running...

^C                                  ! Operator types CTRL/C.
DS> SUMMARY                         ! Supervisor prompt.
                                    ! Operator requests
                                    ! statistical report.

                                 Statistical
                                   Report
                                 Printed Here

DS> ABORT                           ! Supervisor prompt.
                                    ! Operator requests program
                                    ! cleanup and termination.

DS>                                 ! Supervisor prompt.
```

Example 4-10 Use of CTRL/C, Summary, and Abort Commands

SCRIPTING

The scripting feature in the supervisor enables the computer operator to run predefined sequences of diagnostic programs automatically. Supervisor commands normally solicited from the operator's terminal are instead taken from a text file.

Scripting Command

@[load-device] [ ]directory]]<file-spec> <CR>

This command causes the supervisor to execute the commands that it finds in the command file specified. You should build the command file

with a text editor before starting the supervisor. See the *VAX-11 Text Editing Reference Manual* (AA-D029A-TE). Type DS> at the beginning of each line of the script. Then copy the command file on the diagnostic program load device. When you execute the command file from the supervisor, the supervisor assumes that the load device for the command file is the device from which the supervisor was loaded. If the load device is different, specify the device and the directory for the file, either with the scripting command or with a preceding set load command.

Example 4-11 shows a typical command file. Example 4-12 shows how the file can be used. Notice that in Example 4-12 the load device is specified, but the file type and version are not specified. When the operator does not supply the file type and version number, the supervisor applies the defaults ".COM" and latest version number.

```
DS> ATTACH DW780 SBI DW0 3 4
DS> ATTACH DZ11 DW0 TTA 760120 320 4 EIA
DS> SELECT TTA:
DS> RUN ESDAA/PASS:3
```

Example 4-11 A Typical Command File

```
$ COPY CMD.COM DMA0:[TEST]      ! Copy the command file.
$ RUN DMA0:[TEST]ESSAA          ! Run the diagnostic supervisor.
.
.
.
DS>@CMD                         ! Execute the command file.
```

Example 4-12 Execution of a Typical Command File

NOTE

The square brackets around the directory name, [TEST], are necessary. COPY and RUN are VMS commands. COPY makes a copy of the first named file (CMD.COM) in the [TEST] directory. RUN loads and starts the program named (ESSAA).

Diagnostic programs do not solicit information from the operator, except under unusual circumstances. Exceptions are manual intervention tests and volume verification failures for programs that write on disks. Responses to questions of this nature should come from the operator, not from a script. Therefore, script files contain only supervisor commands.

@ Command Processing

The supervisor processes the @ command roughly as follows.

1. The supervisor aborts the current program if necessary.
2. The supervisor reads the whole script at once into a buffer.
3. The supervisor initializes a pointer to the first line of the script.
4. The supervisor sets a flag to indicate that the next command is to be taken from the script.
5. As the supervisor processes the commands in the script, it displays the prompt and command text on the operator's terminal.
6. When the script has been exhausted, the supervisor types "@<EOF>".

Buffer Allocation and Script Nesting

The supervisor dynamically allocates the memory buffer for script text and control and position information. Each script descriptor is linked to previous script descriptors. This allows you to nest scripts. The amount of memory available on a given computer system limits the number of nesting levels possible. If you exceed the memory capacity, the supervisor will type UNKNOWN ERROR PC = 00000124.

You can invoke script nesting with an "@<file-spec>" command within a script. The supervisor processes commands from the second script file until it reaches the end of the script. The supervisor then releases the second script and resumes processing commands from the first script. If no previous script is left unprocessed, control returns to the operator's terminal.

Interrupting the Script

You may type CTRL/C on the terminal to interrupt the script, if necessary. CTRL/C causes the supervisor to suspend the script and stop the current program, if a program is running. You can issue any command while the script is suspended. However, if you want to resume the script eventually, by typing CONTINUE, the selection of commands is limited.

These commands can be followed by resumption of a script or program.

SET	SHOW
CLEAR	SUMMARY
EXAMINE	NEXT
DEPOSIT	CONTINUE

The following commands flush all scripts and return control to the command line interpreter in the supervisor.

ATTACH	START
SELECT	RUN
DESELECT	ABORT

In general, a command flushes a script if it would be meaningless to continue the script after the command has been executed.

Command File Format

A command procedure must be a contiguous ASCII file created by VAX RMS (record management services) on an ODS-1 or ODS-2 disk file structure. The file must be line oriented and records must not exceed 72 characters. You can create a command procedure file with any editor or with the VMS CREATE command. The supervisor treats all records as supervisor commands. Any legitimate supervisor command is valid in a script.

EXECUTION CONTROL FUNCTIONS

The execution control functions allow you to alter the characteristics of the diagnostic programs and the diagnostic supervisor. These functions are implemented by command flags and event flags. The command flags are used to control the printing of error messages, ringing the bell, and halting and looping of the program.

Set Flags Command

SET [FLAGS] <arg-list><CR>

This command results in the setting of the execution control flags specified by <arg-list>. No other flags are affected. <Arg-list> is a string of flag mnemonics from the following table, separated by commas.

HALT	Halt on error detection. When the program detects a failure and this flag is set, the supervisor enters a command wait state after all error messages associated with the failure have been output. You may then continue, rerun, or abort the program. This flag takes precedence over the LOOP flag.
LOOP	Loop on error. When set, this flag causes the program to enter a predetermined scope loop on a test or subtest that detects a failure. Set the IE1 flag if you want to inhibit error messages. Note that you cannot inhibit messages forced by the program or the supervisor. Looping will continue until the operator returns control to the supervisor by using the CTRL/C command. You may then continue, clear the flag and continue, or abort the program.
BELL	Bell on error. When set, this flag causes the supervisor to send a bell to the operator whenever the program detects a failure.

IE1	Inhibit error messages at level 1. When set, this flag suppresses all error messages, except those that are forced by the program or supervisor.
IE2	Inhibit error messages at level 2. When set, this flag suppresses basic and extended information concerning the failure. Only the header information message (first three lines) is output for each failure.
IE3	Inhibit error messages at level 3. When set, this flag suppresses extended information concerning the failure. The header and basic information messages are output for each failure.
IES	Inhibit summary report. When set, this flag suppresses statistical report messages.
QUICK	Quick verify. When set, this flag indicates to the program that the operator wants a quick verify mode of operation. The interpretation of this flag is program dependent.
TRACE	Report the execution of each test. When set, this flag causes the supervisor to report the execution of each individual test within the program as the supervisor dispatches control to that test.
OPERATOR	Operator present. When set, this flag indicates to the supervisor that operator interaction is possible. When cleared, the supervisor takes appropriate action to ensure that the test session continues without an operator.
PROMPT	Display long dialogue. When set, this flag indicates to the supervisor that the operator wants to see the limits and defaults for all questions printed by the program.
ALL	All flags in this list.

Clear Flags Command

CLEAR [FLAGS]<arg-list><CR>

This command results in the clearing of the flags specified by <arglist>. No other flags are affected. <Arg-list> is a string of flag mnemonics separated by commas. See the set command for supported arguments.

Set Flags Default Command

SET FLAGS DEFAULT<CR>

This command returns all flags to their initial default status. The default flag settings are OPERATOR and PROMPT.

Show Flags Command

SHOW FLAGS<CR>

This command displays all the execution control flags and their current status. The flags are displayed as two mnemonic lists; one list is for those flags that are set, the other for those that are clear.

The following example shows how the set flags, clear flags, and show flags commands can be coordinated.

```
DS> SET FLAGS TRACE                ! Set the TRACE flag.
DS> CLEAR FLAGS QUICK              ! Clear the QUICK flag.
DS> SHOW FLAGS
CONTROL FLAGS SET: PROMPT, OPER, TRACE
CONTROL FLAGS CLEAR: QUICK, IES, IE3, IE2, IE1, BELL, LOOP, HALT

DS>
```

Example 4-13 Use of the Flag Control Commands

Set Event Flags Command

SET EVENT [FLAGS] <arg-list>!ALL<CR>

This command results in the setting of the event flags specified by <arglist>. No other event flags are affected. <Arg-list> is a string of flag numbers in the range of 1–23, separated by commas. ALL may be specified instead of <arg-list>.

Event flags are status posting bits maintained by VMS and the supervisor. Diagnostic programs can use event flags to perform a variety of signaling functions, including communication with the operator.

The VMS operating system specifies the functions of the first two event flags.

Event Flag 1 enables (when set) and disables (when clear) error logging under VMS. The default is clear.

Event Flag 2 enables (when set) and disables (when clear) retries under VMS. The default is clear.

The other available event flags (3–23) are not permanently defined. Diagnostic programs that use event flags for interaction with the operator will specify their functions in program documentation files. For example, a program might use an event flag to enable the operator to specify the use of a particular data pattern.

Clear Event Flags Command

CLEAR EVENT [FLAGS] <arg-list>!ALL<CR>

This command clears the event flags specified by <arg-list>. No other event flags are affected. <Arg-list> is a string of flag numbers in the range of 1–23, separated by commas. You may specify ALL instead of <arg-list>

Show Event Flags Command

SHOW EVENT [FLAGS]<CR>

This command causes the supervisor to display a list of the event flags that are currently set.

Example 4-14 shows how the set event flags, clear event flags, and show event flags commands can be coordinated.

```
DS> SET EVENT FLAGS 1, 9, 15
DS> CLEAR EVENT FLAGS 2, 6
DS> SHOW EVENT FLAGS
EVENT FLAGS SET: 15, 9, 1
DS>
```

Example 4-14 Event Flag Control Commands

DEBUG AND UTILITY COMMANDS

This group of commands provides the operator with the ability to isolate errors and to alter diagnostic program code. The supervisor allows up to 15 simultaneous breakpoints within the program. The operator can also examine and modify the program image in memory.

Set Base Command

SET BASE <address><CR>

This command loads the address specified into a software register. This number is then used as a base to which the address specified in the set breakpoint, clear breakpoint, examine, and deposit commands is added. The set base command is useful when referencing code in the diagnostic program listings. The base should be set to the base address (see the program memory allocation map) of the program section referenced. Then the PC numbers provided in the listings can be used directly in referencing locations in the program sections.

For example:

```
DS> SET BASE E00                ! Set the base
                                ! address to the
                                ! beginning of the psect for
                                ! the routine under
                                ! examination.
DS>
```

Example 4-15 Set Base Command

NOTE

Virtual address = physical address (normally) when memory management is turned off.

You can show the base by typing E 0.

For example:

```
DS>    SET BASE        3800
DS>    E 0
     00003800:         43190003
DS>
```

Example 4-16 Showing the Base Address

Set Breakpoint Command

SET BREAKPOINT <address><CR>

This command causes control to pass to the supervisor when the program counter points to the <address> previously specified by this command. A maximum of 15 simultaneous breakpoints can be set within the diagnostic program.

For example:

```
DS> SET BREAKPOINT    30        ! Set a breakpoint
                                ! at an offset of
                                ! 30 from the
                                ! base address.
```

Example 4-17 Set Breakpoint Command

Clear Breakpoint Command

CLEAR BREAKPOINT <address>! ALL<CR>

This command clears the previously set breakpoint at the memory location specified by <address>. If no breakpoint existed at the specified address, no error message is given. An optional argument of ALL clears all previously defined breakpoints.

For example:

```
DS> CLEAR BREAKPOINT    30      ! Clear the breakpoint
                                ! at the location which
                                ! is offset 30 from
                                ! the base address.
DS>
```

Example 4-18 Clear Breakpoint Command

Show Breakpoints Command

SHOW BREAKPOINTS<CR>

This command displays all currently defined breakpoints.

For example:

```
DS> SHOW BREAKPOINTS                ! Display breakpoints
                                    ! currently set.
CURRENT BREAKPOINTS:
        00000E30(X)
DS>
```

Example 4-19 Show Breakpoints Command

Set Default Command

SET DEFAULT <arg-list><CR>

This command causes setting of default qualifiers for the examine and deposit commands. The <arg-list> argument consists of data length default and/or radix default qualifiers. If both qualifiers are present, they are separated by a comma. If only one default qualifier is specified, the other one is not affected. Initial defaults (if you don't change them) are HEX and LONG. Default qualifiers may be:

Data Length: Byte, Word, Long

Radix: Hexadecimal, Decimal, Octal

For example:

```
DS> SET DEFAULT BYTE, DECIMAL  ! Set the default data
                               ! length qualifier as
                               ! byte and the default
                               ! radix qualifier as
                               ! decimal.
DS>
```

Example 4-20 Set Default Command

Examine Command

EXAMINE [<qualifiers>][<address>]<CR>

The examine command displays the contents of memory in the format described by the qualifiers. If no qualifiers are specified, the default qualifiers set by a previous set default command are used. The applicable qualifiers are described in Table 4-1.

Table 4-1 Examine Command Qualifier Descriptions

Qualifier	Description
/B	Address points to a byte
/W	Address points to a word
/L	Address points to a longword
/H	Display in hexadecimal radix
/D	Display in decimal radix
/O	Display in octal radix
/A	Display in ASCII bytes

When specified, the <address> argument is accepted in hexadecimal format unless some other radix has been set with the set default command. Optionally, <address> may be specified as decimal, octal, or hexadecimal by immediately preceding the address argument with %D, %O, or %H, respectively. <address> may also be one of the following: R0–R11, AP, FP, SP, PC, PSL. Note that the supervisor will also accept the names R12–R15 for AP, FP, SP, and PC, respectively.

For example:

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

00000E30:  D0513D01
DS>
```

Example 4-21 Examine Command

Deposit Command

DEPOSIT [<qualifiers>]<address><data><CR>

This command accepts data and writes it into the memory location specified by <address> in the format described by the qualifiers. If no qualifiers are specified, the default qualifiers are used. The applicable qualifiers are identical to those of the examine command described in Table 4-2.

The <address> argument is accepted in hexadecimal format unless some other radix has been set with the set default command. Optionally, <address> may be specified as decimal, octal, or hexadecimal by immediately preceding <address> with %D, %O, or %H, respectively.

For example:

```
DS> DEPOSIT/W/H 30 0001          ! Deposit 0001 (hex)
                                 ! in the word
                                 ! offset 30 from
                                 ! the base address.
00000E30:  0001
DS>

DS> DEPOSIT/W/D %HFF 1009        ! Deposit the value 1009
                                 ! decimal in the word offset
                                 ! FF (hexadecimal) from
                                 ! the base address.
```

Example 4-22 Deposit Command

Next Command

NEXT [number-of-instructions]<CR>

This command causes the supervisor to execute one macro instruction. If you specify a number (decimal) after NEXT, the supervisor will execute that number of macro instructions. The supervisor displays the PC of the next instruction and the contents of the next four bytes, after execution of each instruction.

Use this command to step through an area of a program where you suspect a problem.

NOTE

Do not use NEXT unless you have stopped the program at a breakpoint.

For example:

```
DS> NEXT                              ! Execute the next instruction.
00000E31:  D0513D01
DS>
```

Example 4-23 Next Command

5

DIAGNOSTICS UNDER THE SUPERVISOR

Whether you run diagnostics on-line or standalone, you must first start the supervisor to run level 3, 2, or 2R programs. Then, using supervisor commands, describe the system configuration to the supervisor, set and clear flags as appropriate, and run the diagnostic programs needed.

ON-LINE DIAGNOSTIC SUPERVISOR LOAD PROCEDURES

When you wish to run diagnostic programs in the on-line mode, ensure that a properly formatted diagnostic volume with the SYSMAINT directory is on-line and mounted. Then type RUN [SYSMAINT]ESSAA to the VMS command interpreter to load and start the diagnostic supervisor.

```
$ RUN [SYSMAINT]ESSAA
  .
  .
  .
DS>
```

Example 5-1 Running the Supervisor On-Line

The supervisor is loaded and started. It prompts the operator with DS>. If the SYSMAINT directory is not available, VMS will display an error message indicating that the file (ESSAA) was not found.

VAX/VMS Privileges and Quotas Required (as a minimum) for Running Diagnostics On-Line

The system manager must use the Authorize program to set up the field service user account with the following privileges and quotas.

Privileges:

GPRNAM	PRMCEB
ALLSPOOL	PRMMBX
DETACH	PSWAPM
DIAGNOSE	TMPMBX
LOG_IO	WORLD
GROUP	PHY_IO

Quotas:

CLI:	DCL
ASTLM:	1000
DIOLM:	1000
WSDEFAULT:	256
PCRCLM:	100
BIOLM:	1000
FILLM:	100
WSQUOTA:	512
PRI:	4
BYTLM:	65000
TQELM:	1000
PGFLQUOTA:	40

STANDALONE BOOT

The VAX diagnostic system provides two methods for booting the supervisor and loading diagnostic programs in the standalone mode. The standard method involves booting the supervisor and loading the diagnostic program from the system device on the system bus. Use this method unless a fault in the load path prevents booting the supervisor.

If a load path problem exists, boot the supervisor and load the diagnostic programs from the load path diagnostic package on the console load device. See the appropriate processor-specific diagnostic system overview manual for standalone diagnostic boot procedures.

USING THE SCRIPT FILES

The SYSMAINT area on the system disk contains at least two script files. These files make it easy to run diagnostic programs on-line or standalone from the system disk. However, they are not available when you run load path diagnostics.

The script files contain sequences of commands to the supervisor that make it possible for you to run a series of diagnostic programs with one or more commands. This means that you need not deal with the names of the hardware components and the diagnostic programs in order to test the system. The scripts for each VAX system are tailored to the hardware configuration of that system.

The configuration script file (CONFIG) consists of a series of attach commands that describe the hardware configuration to the supervisor. The program execution script file (SYSTEST) includes a call to CONFIG, selects devices for test, controls flags, and executes appropriate diagnostic programs. The SYSTEST script file runs in the standalone mode only.

Type @SYSTEST or @CONFIG to execute these scripts. Example 5-2 shows the listing for a typical CONFIG script. Example 5-3 shows a corresponding listing for a SYSTEST script.

```
DS>    !CONFIGURATION FILE FOR SYSTEM TYPE SV-AXHAA DUAL RK07
DS>    !PACKAGED SYSTEM    VERSION: 1.0    01-MAY-79
DS>    !CONFIG.COM;3
DS>    !
DS>    !Define processor...
DS>    ATTACH KA780 SBI KA0 NO NO 0 0
DS>    !
DS>    !Define Memory
DS>    ATTACH MS780 SBI MS0 1
DS>    !
DS>    !Define UNIBUS adapters...
DS>    ATTACH DW780 SBI DW0 3 4 0
DS>    !
DS>    !Define UNIBUS disks...
DS>    ATTACH RK611 DW0 DMA 777440 210 5
DS>    ATTACH RK07 DMA DMA0
DS>    ATTACH RK07 DMA DMA1
DS>    !
DS>    !Define terminals...
DS>    ATTACH DZ11 DW0 TTB 760110 310 5 EIA
DS>    !
DS>    ATTACH VT100 TTB TTB0
DS>    ATTACH VT100 TTB TTB1
DS>    ATTACH VT100 TTB TTB2
DS>    ATTACH VT100 TTB TTB3
DS>    ATTACH VT100 TTB TTB4
DS>    ATTACH VT100 TTB TTB5
DS>    ATTACH VT100 TTB TTB6
DS>    ATTACH VT100 TTB TTB7
```

Example 5-2 A Typical CONFIG Script Listing

```
DS>    !SYSTEM TEST SCRIPT FOR SYSTEM TYPE SV-AXHHA DUAL RK07
DS>    !PACKAGED SYSTEM
DS>    !FOR STANDALONE USE ONLY...
DS>    !SYSTEST.COM;3   VERSION:1.0   01-MAY-79
DS>    @CONFIG
DS>    !
DS>    SELECT ALL          ! Select everything.
DS>    !
DS>    RUN ESKAX           ! Cluster Exerciser Quick Verify.
DS>    RUN ESKAY           ! Cluster Exerciser Native Mode Inst.
DS>    RUN ESKAZ           ! Cluster Exerciser MEM-MGT/PDP-11 Inst.
DS>    !
DS>    RUN ESCBA           ! DW780 Test
DS>    !
DS>    RUN EVRAA/SEC:QUAL ! Verify disk functionality.
DS>    RUN EVDAA/SEC:QUAL ! Verify DZ11 functionality.
DS>    !
DS>    RUN EVXBA           ! Verify integrity of system buses.
DS>    !
DS>    SET FLAGS QUICK
DS>    RUN EVRAA           ! Run disk reliability in quick mode.
DS>    CLEAR FLAGS QUICK
DS>    !
DS>    !END OF SYSTEST...
```

Example 5-3 A Typical SYSTEST Script Listing

CREATING AND MODIFYING SCRIPT FILES

DIGITAL distributes script files for packaged systems. You will need to update these script files in order to reflect the addition of new peripheral devices. Use an editor (such as SOS) under VMS to modify your script files or create new scripts. See Chapter 4 for details. The file names for the existing script files are: [SYSMAINT]CONFIG.COM, and [SYSMAINT]SYSTEST.COM.

RUNNING DIAGNOSTICS UNDER THE SUPERVISOR WITHOUT SCRIPT FILES

You will probably, on occasion, need to run diagnostics without the benefit of script files, for example, when you run load path diagnostics. Load path diagnostics are programs which test for failures in the primary load path (system disk, channel adapter, CPU).

The CONFIG script file is of no use when you are using load path diagnostics or when you are testing a device not mentioned in the CONFIG file. In either case, use the attach command to make the device known to the supervisor. Then select the device for testing with the select command and run the appropriate program, as shown in Example 5-4.

```
DS> ATTACH RH780 SBI RH0 8 5     ! Attach MASSBUS interface.
DS> ATTACH RP06 RH0 DBA7         ! Attach system disk.
DS> SELECT DBA7                  ! Select system disk.

DS> RUN EVRAA/SECTION:QUAL       ! Load the disk reliability
                                 ! program from the console load
                                 ! device and run it to verify disk
                                 ! functionality. The console load
                                 ! device is the default load
                                 ! device if the supervisor has
                                 ! been booted from the floppy.
```

Example 5-4 Use of Attach, Select, and Run

The SYSTEST script file is unavailable when you run diagnostics on-line or when you run load path diagnostics, and useless when you wish to run one test or a portion of one test only. If the device to be tested is

mentioned in the CONFIG file, type @CONFIG to make the device known to the supervisor. Then select the device and run the appropriate diagnostic program, as shown in Example 5-5. See Chapter 4 for details on the supervisor commands.

```
DS> @CONFIG                          ! Invoke the Configuration
      .                              ! script file.
      .
      .

DS> SELECT DBA7:                     ! Select the device to be tested.

DS> RUN EVRAA/SECTION:CONVERSATION

                                     ! Load the disk reliability
                                     ! program and run the
                                     ! manual intervention section.
```

Example 5-5 Running a Diagnostic Program Without Use of the SYSTEST Script File

Notice that when you use the load or run commands, the program will always be loaded from the default load device. The default load device is the device from which the supervisor has been loaded, unless you change the default with the set load command.

6

DIAGNOSTIC PROGRAM INTERPRETATION

ERROR REPORT FORMATS

VAX diagnostic programs provide informative error messages. In most cases these messages will help you to quickly indentify a failing subsystem, function, or module. Example 6-1 shows a message from a repair level program. Example 6-2 shows one from a functional level program.

```
******** MAINDEC ZZ-ESCAA-6.0          RH780 DIAGNOSTIC - 6.0
PASS 1 TEST 3 SUBTEST 2 ERROR 2        11-SEP-1979 13:30:23.18
HARD ERROR WHILE TEST RH0:             FAILING-MODULE: MIR(M8276)

(RHCR) = 00000004 ERROR:
EXPECTED:          BIT17,BIT16,IE
RECEIVED:          IE
XOR:               BIT17,BIT16
EXPECTED:          00030004 RECEIVED: 00000004 XOR: 00030000
```

Example 6-1 Sample Error Message with Failing Module Callout

```
******** ZZ-EVRCA RP0X/DCL DIAGNOSTIC - 4.1 ********
PASS 1 TEST 1 SUBTEST 0 ERROR 2  10-MAR-1980 08:26:20.26
DEVICE FATAL WHILE TESTING DBA0: CONTROL BUS PARITY ERROR DETECTED

MBA CHANNEL STATUS DUMP

MBA_CSR:[20010000]        00000020(X);
MBA_CR:[20010004]         00000000(X);
MBA_SR:[20010008]         00020000(X);    MCPE
MBA_VAR:[2001000C]        00000200(X);
MBA_MAP(80):              00000000(X);
MBA_BCNT:[20010010]       00000000(X);
```

Example 6-2 Sample Error Message with Failing Function Callout

In each case, these messages would probably give you enough information to identify the problem and repair the machine. And as you become more familiar with the VAX computer system, the diagnostic system, and the program being run, the error messages should become even more useful.

Occasionally, however, a message may be insufficient or even misleading. For example, you may replace module M8276 in accordance with the message in Example 6-1. If you then rerun the program and get the same message, error isolation will obviously require more work.

FINDING THE RELEVANT DOCUMENTATION

Error messages always give the failing test number, the subtest number, and the error number. This information provides an index into the program documentation.

The documentation for each VAX diagnostic program falls into five categories. All are available on microfiche.

- EVNDX, for AIDS Problem Reports and Release Notes
- documentation file
- memory allocation map
- header module listing
- 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.

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

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

Memory Allocation Map

Two parts of the memory allocation map (link map) are useful for troubleshooting.

First, the program section synopsis lists the starting and ending address of each program section (psect) in the program. Since each test begins with a new psect, its starting address is given in the program section synopsis. Figure 6-1 shows parts of the link map program section synopsis for the RH780 diagnostic, ESCAA.

Consider the psect for Test 3 (TEST_003). The map gives two entries for Test 3. In this case they are identical. The first entry shows the base and end addresses and the length of the Test 3 psect. The second entry shows what part of the psect was contributed by the RH780_TESTS1 module.

Second, 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 value indicates that the symbol is relocatable.

Figure 6-2 shows a section of the symbol cross reference in the map. The symbol PGM_INIT has been assigned a value of 1. PISR10, on the other hand, is the symbol for address 2CFC.

```
                                    +-------------------------+
                                    ! PROGRAM SECTION SYNOPSIS !
                                    +-------------------------+

P-SECT NAME     MODULE(S)          BASE     END      LENGTH          ALIGN                ATTRIBUTES
-----------     ---------          ----     ---      ------          -----                ----------

$HEADER                            00000200 00000281 00000082 (     130.) PAGE 9 NOPIC,USR,CON,REL,LCL,NOSHR,NOEXE,  RD,NOWRT
                RH780_HEADER       00000200 00000281 00000082 (     130.) PAGE 9

    •
    •
    •

INITIALIZE                         00002E3C 00003091 00000256 (     598.) LONG 2 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT
                RH780_HEADER       00002E3C 00003091 00000256 (     598.) LONG 2

SUMMARY                            00003094 0000309D 0000000A (      10.) LONG 2 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT
                RH780_HEADER       00003094 0000309D 0000000A (      10.) LONG 2

TEST.001                           00003200 00003355 00000156 (     342.) PAGE 9 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,NOWRT
                RH780_TESTS1       00003200 00003355 00000156 (     342.) PAGE 9

TEST.002                           00003400 00003698 0000029C (     668.) PAGE 9 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,NOWRT
                RH780_TESTS1       00003400 00003698 0000029C (     668.) PAGE 9

TEST.003                           00003800 00003987 00000188 (     392.) PAGE 9 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,NOWRT
                RH780_TESTS1       00003800 00003987 00000188 (     392.) PAGE 9

TEST.004                           00003A00 00004152 00000753 (    1875.) PAGE 9 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,NOWRT
                RH780_TESTS1       00003A00 00004152 00000753 (    1875.) PAGE 9
```

TK-3904

Figure 6-1 Link Map Program Section Synopsis

```
$SYMBOL                 VALUE           DEFINED BY              REFERENCED BY ...
------                  -----           ----------              -----------------
NIBBLE7                 F0000000        RH780_HEADER            RH780_TESTS1
NODRIVE                 000002E0-R      RH780_HEADER            RH780_TESTS1
NON_XIST_DRIVE          00040000        RH780_HEADER            RH780_TESTS2
NOOP                    00000000        RH780_HEADER            RH780_TESTS1            RH780_TESTS2
NO_RESP_CONF            40000000        RH780_HEADER            RH780_TESTS1            RH780_TESTS2
NO_UNITS                00000283-R      RH780_HEADER
NUMBER_BUFFER           00000540-R      RH780_HEADER
OCC                     00000019        RH780_HEADER
OPI                     00002000        RH780_HEADER
PAGE_BYTE_MSK           000001FF        RH780_HEADER
PF_FIELD                00000009        RH780_HEADER            RH780_TESTS1            RH780_TESTS2
PF_WIDTH                00000015        RH780_HEADER
PGM_INIT                00000001        RH780_HEADER            RH780_TESTS1            RH780_TESTS2
PIP                     00002000        RH780_HEADER
PISR10                  00002CFC-R      RH780_HEADER
PISR10X                 00002D07-R      RH780_HEADER
PISR11                  00002D08-R      RH780_HEADER
```

TK-3903

Figure 6-2 Link Map Symbol Cross Reference

Program Modules

The program header module is the next item in the listing. This part of the listing has little value in system troubleshooting. The header module contains the following items.

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

Gobal 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.

Test modules follow, and they make up the greater part of most diagnostic programs. Each test module generally contains the following items.

- module preface
- declaration section
- test description for each test
- code for each test
- a module symbol table and symbol cross reference

ERROR ANALYSIS

Begin your analysis of an error by reading the description of the failing test in the program documentation file. The description explains the purpose and methods of the test, and it should give a general idea 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, 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. The description normally includes debug procedures and troubleshooting aids, as well as a step by step statement of the test algorithm. Read this before proceeding. Then locate the program code for the failing function by looking through the test routine listing for the appropriate subtest and error numbers. Line comments and block comments explain the functions of each line or group of lines.

If you need still more details concerning the failure, several procedures are available.

- Loop on error by rerunning the program with LOOP flag set.
- Analyze the code.
- Set breakpoints and step through the test.
- Use examine and deposit commands to alter code and data after setting breakpoints.

Looping and stepping through the program require good familiarity with the diagnostic supervisor commands (Chapter 4). Analyzing the code requires an understanding of the appropriate programming language – MACRO-32 or BLISS-32.

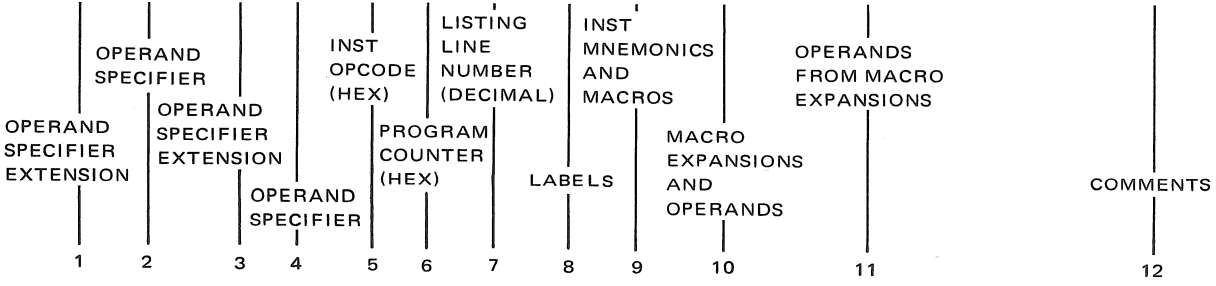
MACRO-32 CODE INTERPRETATION – A SAMPLE TEST

MACRO-32 is the assembly language for VAX computers. Each line of MACRO-32 code produces machine code which is exactly equivalent.

Listing Column Format Description

Figure 6-3 shows the program listing for the MBA RH780 diagnostic program (ESCAA), Test 3, subtest 1. The sixth column from the left contains the relative address of each instruction. These numbers begin at 0 with the beginning of each program section. Note that the address offset of the program section containing Test 3 (3800_{16}), found in the memory allocation map, must be added to the relative address to find the physical memory address of the instruction.

```
                                        0046   329 ; TEST CLEARING VIA D-INPUTS
                                        0046   330 ;--
                                        0046   331         $DS_BGNSUB
                                        0046       T3_S1::
00000000'9F  00000000'EF      FA        0046               CALLG   $S$, @#DS$BGNSUB
    0000'C2  00000000'8F      C8  0051  332 10$:   BISL    #PGM_INIT,CR(R2)        ; INIT RH UNDER TEST
    0000'C2            0E     9A  005A  333         MOVZBL  #^XE,CR(R2)             ; WRITE ONE'S VIA D INPUTS TO CR
                 0000'C2      D4  005F  334         CLRL    CR(R2)                  ; CLEAR VIA D INPUTS
          53     0000'C2      D0  0063  335         MOVL    CR(R2),R3               ; READ 0'S FROM CONTROL REGISTER
                       1D     13  0068  336         BEQL    20$                     ; SKIP IF NO ERRORS
                       54     D4  006A  337         CLRL    R4                      ; CLEAR EXPECTED RESULTS REGISTER
                                  006C  338         $DS_ERRHARD_S   #1,LUN,MIR,-    ; PRINT ERROR
                                  006C  339                         PRINT_SBE
                 00000000'EF      DF  006C                         PUSHAL  PRINT_SBE
                 00000000'EF      DF  0072                         PUSHAL  MIR
                 00000000'EF      DD  0078                         PUSHL   LUN
                          01      DD  007E                         PUSHL   #1
00000000'9F               04     FB  0080                         CALLS   #$$M, @#DS$ERRHARD
                                  0087  340 20$:     $DS_CKLOOP      10$             ; SCOPE LOOP?
00000000'9F            C7 AF      FA  0087                         CALLG   10$, @#DS$CKLOOP
                                  008F  341         $DS_ENDSUB
                                  008F       T3_S1_X:
00000000'9F  00000000'EF      FA        008F               CALLG   $S$, @#DS$ENDSUB
                                  009A  342 ;++
                                  009A  343 ; TEST CONTROL REGISTER FOR STUCK AT ZEROES
                                  009A  344 ;--
                                  009A  345         $DS_BGNSUB
```



TK-0736

Figure 6-3 RH780 (MBA) Diagnostic Program Test 3, Subtest 1, Listing

The seventh column from the left contains the listing line numbers. These numbers begin at 0 for each module of the program. Note that the line number increments for each line of the source program. The relative address increases according to the amount of memory space required for the instructions and operands. Line numbers are present only for lines entered by the program developer. Macro expansions do not have line numbers.

The eighth column from the left contains labels used by the programmer as symbolic addresses.

The ninth column from the left contains instruction mnemonics and macro calls. Note that the macro calls themselves require no memory space (the relative address does not change), and that in the macro expansion which follows, the line numer is not incremented (see lines 331 to 333). At assembly time, the assembler program has responded to the macro calls, expanding the macro according to the definition listed at the beginning of the file or in a macro library.

Column ten contains operands for those intructions located in column nine; and it contains instruction mnemonics and parameters for the macro expansions.

The eleventh column from the left contains operands from the macro expansions.

Column five contains the op codes (hex) for the instructions contained in columns nine and ten.

Columns one through four contain the hexadecimal code for the operands specified in columns ten and eleven. Columns two and four contain operand specifiers. Columns one and three contain operand specifier extensions. Numbers followed by an apostrophe (e.g., 00000000') are the machine code for symbolic operands. They are modified by the linker at link time. However, the program listing is always created before the program modules are linked. Therefore these symbols are represented as 00000000'.

The twelfth column contains comments describing the functions of the instructions. A semicolon precedes each comment.

Analysis of Typical Lines

Line 332, the BISL instruction, sets bits in the destination according to the mask provided. PGM_INIT is the symbol for the mask. The symbol table at the end of the module gives ****** as its value. The asterisks indicate that the symbol is global. You must look in the symbol cross reference in the map for the value (Figure 6-2). The value is 00000001. CR is the symbol for the relative address (offset from the MBA base register) of the control register of the MBA under test. Its value (00000004) is also listed in the symbol cross reference in the map. This value is added to the contents of R2, the base address of the MBA under test, to produce the physical address of the control register. The instruction thus sets bit zero of the control register. The comment, INIT, indicates the function that setting bit zero performs.

Note that it may often be easier to find the value assigned to a symbol by using the examine command in the diagnostic supervisor. You could determine the value of PGM_INIT, for instance, as follows.

```
DS>  SET BASE              3800
DS>  EXAMINE/L 53
     00003853:       00000001
DS>
```

Example 6-3 Using Examine to Find the Value of a Symbol

Line 338, $DS_ERRHARD_S in line 338, is a macro call. The symbols that follow it are arguments to be used in the call. The five lines that follow line 339 show the expansion of the macro. These instructions push the arguments on the stack and call the DS$ERRHARD subroutine in the supervisor, which sets the error flag and prints an error message based on the stored arguments.

Test Procedure and Flow

The test procedure used most commonly in VAX diagnostic programs involves writing data to the device under test and then reading it back. This method exercises the logic circuits or the functions to be tested. The code compares data received with the data expected. The test may perform the I/O and comparison functions directly, or it may call a subroutine to perform these functions. Level 2 and 2R programs call VMS

system services to perform all I/O functions (QIO). If the comparison indicates a failure, the test routine calls an error routine, which, in turn, sends a message to the operator.

Subroutines and system services normally return a status code via general register RO to the test routine.

1 = success
0 = error

When the code indicates an error, the test routine normally calls an error print routine to send a message to the operator. If you must analyze the code of the subroutine, proceed as follows.

- Find the call, branch, or jump instruction which passes control to the subroutine. This may be in a macro expansion.

- Find the subroutine name in the local symbol table. Some global symbols will require you to go to the symbol cross reference in the memory allocation map.

- Locate the subroutine in the header module, the global subroutine module, or the test module.

Diagnostic supervisor service routines and VMS service routines are called with macros. Supervisor routine names begin with the characters DS$. VMS service routines begin with SYS$. You should not normally need to analyze the code of the service routines.

Scope Loops

Scope loops are available in most subtests. If the LOOP flag is set and the test detects an error, the program will loop on the error indefinitely. Look in the subtest listing for the macro:

$SDS_CKLOOP <label>

This statement forms the end of the loop. The label forms the beginning. In Figure 6-3, line 340 contains a $SDS_CKLOOP macro. Control branches from line 340 back to line 332, 10$, at the beginning of each loop.

BLISS-32 CODE INTERPRETATION

BLISS-32 is a block-structured, medium level language that has many features of a high level language. It uses algebraic notation for calculations, with operations for arithmetic, shifting, comparison, and logic. It supports a rich variety of data structures and provides the developer with the facility to create his own data structures.

However, BLISS-32 has no built-in I/O routines. BLISS-32 is a machine-dependent language, and this permits a high degree of coupling between the software and hardware. This feature of BLISS-32 makes it suitable for implementing hardware diagnostic programs. See the *BLISS-32 LANGUAGE GUIDE* (AA-H019A-RE).

Data Representation and Manipulation in BLISS-32

The default data element in BLISS-32 is the longword. The BLISS-32 compiler will always attempt to use the most efficient machine code to handle a calculation, and this is often done by using longword instructions. For example, consider the following BLISS-32 expressions (the variable names represent consecutive bytes of memory storage).

```
BYTE_ONE = 0;

BYTE_TWO = 0;

BYTE_THREE = 0;

BYTE_FOUR = 0;
```

The BLISS-32 compiler, in all probability, would generate a single line of machine code to accomplish those four assignments.

```
CLRL    BYTE_ONE
```

In a hardware diagnostic program, it is often undesirable to allow the compiler free choice. For example, in a read or write on a UNIBUS device, the developer must be sure the compiler does not try to use a longword reference (which would cause a trap). This is easily accomplished in BLISS-32. Any size of bus reference can be effected simply by

an explicit specification of the field size of the operand. In general, the compiler will use branch-on-bit instructions in dealing with single-bit fields, byte instructions for byte aligned 8-bit fields, and word instructions for byte or word aligned 16-bit fields.

The Fetch Operator

Because addresses can be manipulated in the same manner as data in BLISS-32, an operator, represented by a period, distinguishes a value in an expression as either an address or the contents of the address. In BLISS-32, a variable name by itself is always an address. A period preceding a variable name represents the contents of that address. For example, the expression

```
X = 123;
```

stores the decimal value 123 in location X, whereas the expression

```
.X = 123;
```

stores the decimal value 123 in the location specified by the contents of X. As an additional example, consider the representation of an increment instruction in BLISS-32.

```
COUNTER = .COUNTER + 1;
```

This expression increments the contents of the location COUNTER by one, and replaces the contents of COUNTER with the sum.

Value Assignments

The assignment operator, = , is used to assign a value to a storage location. For example, the expression

```
TEMP = 1000;
```

causes the longword value representing decimal 1000 to be stored in the four consecutive bytes of storage beginning at the byte whose address is TEMP.

Expressions

Most high level languages distinguish between statements and expressions. A statement performs an action without producing a value, and an expression calculates a value. Thus an assignment such as

```
A = 1;
```

is a statement, whereas the sum

```
.B + 1;
```

is an expression. The combination of the two is typically permitted on the same line, as

```
A = .B + 1;
```

in which the value of the expression .B + 1 is assigned to A. However, in BLISS-32, there is no distinction made between statements and expressions. Thus, an assignment can legally appear anywhere in an expression. For example,

```
X = .Y * (A =.B + 1);
```

computes the value .B + 1 and multiplies it by the contents of Y. The resulting value is stored in location X. However, at the same time, with no additional computation, the value .B + 1 is stored in location A.

Blocks

Blocks provide for the organization of a program into units. The typical block is delimited by the pair of keywords BEGIN and END. Blocks can also be legally delimited by a pair of parentheses. A block delimited by a pair of parentheses is generally termed a parenthesized expression. A block which contains no declarations is generally termed a compound expression. There are two main categories of blocks: blocks that compute a value, and blocks that only take action.

Declarations

Before any name may be used in a BLISS-32 program, it must be declared. The initial declaration of a name provides the compiler with information about the attributes to be associated with the name. A declaration is altogether distinct from an expression. For instance, the declaration

```
LITERAL A = 1234;
```

means that whenever A appears in the program, the value 1234 decimal is to be used. The value 1234 is not stored in location A.

Storage is allocated to a program with a declaration such as

```
OWN BETA:
```

In this example, four consecutive bytes are allocated to the program, and the name BETA is associated with the address of the first byte. A declaration may also specify attributes to be associated with a name, as in the following example:

```
OWN STATUS: VECTOR [4];
```

which allocates four longwords of storage and tells the compiler that STATUS has the attribute of a longword vector.

Data Segments

Values in a BLISS-32 program are stored in data segments. A data segment is defined as one or more bytes of memory. There are two main categories of data segments: scalars and structures. A scalar is a single value, and can be either a byte, word, or longword in length. It may also be signed or unsigned. The default attributes are longword, unsigned. For example,

```
OWN ALPHA;
```

allocates four consecutive bytes of memory storage. In contrast,

```
OWN ALPHA: BYTE;
```

allocates a single byte of memory storage.

BLISS-32 defines several structures. These predeclared structures include:

1. Vectors
2. Bitvectors
3. Blocks
4. Blockvectors

A vector structure is a sequence of elements. The number of elements is the extent of the vector, and is given as part of the declaration of the data segment name. Thus,

```
OWN ALPHA: VECTOR[3];
```

allocates three consecutive longwords of storage. Reference to a particular element in the program would typically appear as

```
B=.ALPHA[1];.
```

This expression would store the contents of the second element in the location whose address is B.

A vector can also be declared with an allocation unit. This allocation unit can be byte, word, or longword. Thus,

```
OWN GAMMA: VECTOR[11, BYTE];
```

allocates eleven consecutive bytes of storage.

The bitvector structure is similar to the vector, except that each element of the vector is always one bit in length. For example,

```
OWN DELTA: BITVECTOR [15];
```

allocates two consecutive bytes of storage, even though one bit will never be referenced.

A block structure is a sequence of components. The block has a name, and each component of the block has a name. A block is declared with a size and an allocation unit. The size will specify the total amount of required storage for the entire block. The allocation unit specifies the units in which the size is measured. Thus,

```
OWN TOTAL: BLOCK [6, BYTE];
```

allocates six consecutive bytes of storage.

The blockvector is a sequence of elements (just like a vector). The number of elements is the extent of the blockvector, and is given as part of the declaration of the segment name. Each element of the blockvector is a sequence of components (just like a block).

Control Expressions

There are five basic types of flow of control in BLISS-32: sequential, conditional, iterative, subroutine, and condition handling.

Sequential flow of control is the evaluation of expressions strictly in the order that the expressions appear in the program. This applies to expressions that are contained in a block and separated by semicolons. For example:

```
BEGIN
A = .X + 1;
B = .A * 3;
C = 24;
END;
```

This block consists of three assignments, each of which is evaluated in turn.

Conditional flow of control produces the evaluation of one of two expressions on the basis of the outcome of a test. For example:

```
IF .EXPECTED EQL .RECEIVED
THEN
   FLAG = 1
ELSE
   FLAG = 0;
```

This expression sets FLAG equal to 1 if the contents of EXPECTED are equal to the contents of RECEIVED. Otherwise, it sets FLAG equal to 0. BLISS-32 has three main categories of conditional flow expressions: conditional expressions, case expressions, and select expressions.

Iterative flow of control is the repeated evaluation of an expression. For example:

```
INCR 1 FROM 0 to 99 DO
  A[.I] = B [.I];
```

In this example, the loop is evaluated 100 times.

Subroutine flow of control is the evaluation of an expression (usually a block) that is written somewhere else in the program. For example:

```
ROUTINE INCREMENT (ALPHA)=
   ALPHA = .ALPHA + 1;
. . .
. . .

INCREMENT (ALPHA):
. . .
. . .
```

When the program reaches the routine call INCREMENT (ALPHA), the flow jumps to the routine, evaluates the assignment, and returns the flow to the next line following the routine call.

Condition handling flow of control is the evaluation of an expression (usually a block) in response to an unusual condition detected either by the VAX hardware or by software. The expression that is evaluated, which must be coded as the body of a routine, is determined by the stack of routine calls that lead to the detection of the condition. Different routines can be invoked for the same condition at different times. Flow of control may or may not return to the point at which the conditon was detected.

ASSEMBLY LANGUAGE LISTINGS IN BLISS-32 PROGRAMS

Each section of BLISS-32 source code in the program listing is followed by corresponding assembly language code. If you wish to bypass the BLISS-32 code and read the assembly language code for a failing test, first scan the BLISS-32 code for the failing test. Then continue looking through the BLISS-32 code until you find the error number. The error number will appear in the listing in the following format:

```
; %PRINT   ERROR NUMBER N
```

Go to the line immediately following the error number, and note the source line number. For example, consider the listing fragment shown in Example 6-4.

```
;  1511          IF .TEMP1 NEQ .TEMP2
;  1512          THEN
;  1513                BEGIN
;  %PRINT:       ERROR NUMBER 1
;  1514                $DS_ERRHARD(UNIT=.LUN,MSG_ADR=RCS_CPU_RW):
;  1515                $DS_PRINTX(FMT_CPU_RW):
```

Example 6-4 Sample BLISS-32 Source Code

The source line number of interest is 1514. Advance through the listing until you locate the MACRO-32 listing for this test. The MACRO-32 listing can be found immediately following the BLISS-32 source code listing. Scan the extreme right of the listing until you locate the source

line number (1514 in Example 6-4). All of the MACRO-32 code associated with that line appears sequentially in the listing, starting at the line identified with the source line number, and continuing until the next source line number is encountered. Example 6-5 is a MACRO-32 listing fragment for the BLISS-32 code shown in Example 6-4.

```
00561   BEQL    2$                  ;
00563   CLRQ    -(SP)               ;  1514
00565   CLRQ    -(SP)               ;
00567   CLRQ    -(SP)               ;
00569   CLRL    -(SP)               ;
0056B   PUSHAB  RCS_CPU_RW          ;
0056F   MOVZBL  LUN, -(SP)          ;
00574   PUSHL    1                  ;
00567   CALLS    10, @#DS$ERRHARD   ;
0057D   PUSHAB  FMT_CPU_RW          ;  1515
00581   CALLS    1, @#DS$PRINTX     ;
```

Example 6-5 Assembler Equivalent of BLISS-32 Code

The column of five-digit numbers is the assembler location counter. Note that from location counter 00563 to 00567 is code associated with line 1514 of the BLISS-32 source listing.

Now, to find the beginning of the subtest, having located the error call, simply scan the MACRO-32 listing in reverse until the first BGNSUB call is located. This appears in the MACRO-32 listing as:

```
CALLS   2, @#DS$BGNSUB
```

Block comments describing the program action are interspersed with the MACRO-32 code. These comments, coupled with the functional description of the test, should provide the user with enough information to understand the program.

7

SYSTEM VERIFICATION AND ANALYSIS

VMS provides several tools that aid VAX customers and DIGITAL Field Service engineers in hardware maintenance. The User Environment Test Package (UETP) is a collection of tests which demonstrate whether the hardware and software components of a VAX/VMS system are in working order. The UETP is not a diagnostic program.

The System Dump Analyzer (SDA) is a VAX/VMS utility which aids in determining the cause of an operating system crash. When the operating system fails, SDA writes information concerning the operating system status at the time of the crash to a predefined system dump file. SDA also examines the formats and contents of this file.

The VAX/VMS error logging facility gathers and maintains information on system errors and events as they occur. This information provides a detailed record of system activity. By running the report generator program (SYE), you can obtain a record of errors and events that have occurred within a specified time period. Both SDA and SYE have been designed for efficient use at a video terminal for the experienced user.

This chapter gives a brief summary of the operating procedures for each of these facilities. See the appropriate VAX/VMS manuals for details.

UETP:

VAX/VMS UETP User's Guide (AA-D643A-TE)

SDA:

VAX/VMS System Dump Analyzer Reference Manual (AA-J562A-TE)

Error Logging:

VAX/VMS System Manager's Guide, Chapter 16, (AA-D027A-TE)
VAX/VMS Operator's Guide, Section 2.4.1 (AA-D025A-TE)

RUNNING THE USER ENVIRONMENT TEST PACKAGE (UETP)

The UETP leads the system through a series of exercises. At the end of the series most hardware and software components have been requested to perform one or more tasks.

When the tests run successfully, they show not only that individual components work, but also that those components work together as an integrated system. The UETP is not a diagnostic program. Instead, it is a VAX system verification tool.

Logging Out

Log out from the field service or user account:

```
$ LOGOUT <CR>
```

the system responds:

```
VAX/VMS LOGOUT at 12:43:10 17-JUL-1978
```

Logging In

Log into the SYSTEST account as follows:

```
<CR>
Username: SYSTEST <CR>
Password: <CR>
```

Note that the system does not echo the password.

Preparing Devices for Testing
This section tells you how to prepare different kinds of devices for testing by the UETP.

Disk Drives – To prepare each disk for testing, perform the following steps.

- Physically mount a scratch disk.
- Start up the drive.
- Issue one or more of the following commands as required.

```
$ INITIALIZE/DATA _ CHECK <device-name:label><CR>
$ MOUNT/SYSTEM<device-name:label><CR>
$ CREATE/DIRECTORY <device-name:>[SYSTEST]<CR>
```

Magnetic Tape Drives –To prepare each magnetic tape drive for testing, perform the following steps.

- Turn power on to the device.
- Physically mount a write-enabled scratch tape at least 600 feet long.
- Position the tape at the BOT marker.
- Press the ONLINE switch.
- If necessary, initialize the tape by entering the command:

```
$ INITIALIZE <device-name:label><CR>
```

- Mount the tape by entering the command:

```
$ MOUNT<device-name:label> <CR>
```

Terminals and Line Printers – Prepare terminals and line printers for testing by performing the following steps.

- Turn on power to the device.
- Check the paper supply if the device produces hard copy (two pages for each pass of the UETP).
- Press the ONLINE switch.
- Check baud rates and terminal characteristics.

Other Devices – The UETP does not test the following devices.

- Card reader
- Network devices (DMC11s)
- Null devices
- Mailboxes
- The console terminal and the console load device
- The terminal used to initiate the UETP tests
- Dialup terminal lines
- Nonstandard devices

Running the Entire UETP

To initiate the UETP test package, enter a call to the UETP master command procedure and respond to the three prompts shown below:

```
$ @UETP[/OUTPUT=<filespec>]<CR>

VAX/VMS UETP STARTED dd-mmm-yy  hh:mm
ENTER NUMBER OF LOAD TEST USERS [D]:<n><CR>

ENTER NUMBER OF COMPLETE UETP RUNS [D]:<n><CR>

ENTER SCRATCH MAGTAPE (E.G. MTO:) OR A <CR>:<device-
name:><CR>
```

See Chapter 2 of the *VAX/VMS UETP User's Guide* for information on choosing the number of load test users. This parameter varies according to the system memory size and system disk drive type.

Use CTRL/Y or CTRL/C to interrupt the tests.

RUNNING THE SYSTEM DUMP ANALYZER (SDA)

When the operating system crashes, the kernel routine writes the contents of the error log file, the processor registers, and physical memory to a contiguous file called SYSDUMP.DMP. With the help of the SDA commands you can analyze and display parts of the formatted system dump file on a video display terminal. Or, you can create hard copy listings.

Any user may run SDA by typing the DCL command:

```
$ RUN SYS$SYSTEM:SDA
```

When you issue this command, SDA will prompt for the name of the system dump file you want to examine.

```
Enter name of dump file >
```

To examine the most recent system dump, press RETURN at the prompt

```
Enter name of dump file >
```

SDA will search the system directory [SYSEXE] (logical name SYS$SYSTEM) for the SYSDUMP.DMP file. To examine an older system dump, enter its file specification.

```
Enter name of dump file > DBO.[EYSEXE]SYSDUMP.OLD
```

Saving the System Dump File

When the kernel routine writes data into SYSDUMP.DMP, it destroys the previous contents of the file. Therefore, be sure to make a copy of the file under another name. Use the VMS copy command or the SDA copy command to save the file.

```
$ COPY SYSDUMP.DMP SAVEDUMP.DMP
```

or

```
SDA> COPY SYS$SYSTEM:SAVEDUMP.DMP
```

SDA Commands

Table 7-1 gives a summary of the SDA commands.

Table 7-1 Summary of SDA Commands

Command	Function
<escape key>	Repeat last command
COPY	Copy dump file
DEFINE	Define symbols and their values
EVALUATE	Perform computations
EXAMINE	Examine memory locations
EXIT	Exit from display or utility
FORMAT	Format data blocks
READ	Copy object module symbols
SET OUTPUT	Set output to device or file specification
SET PROCESS	Set to the current process context
SHOW CRASH	Display crash information
SHOW DEVICE	Display I/O data structures
SHOW PAGE_TABLE	Display system page table
SHOW PFN_DATA	Display PFN data base
SHOW POOL	Display dynamic memory
SHOW PROCESS	Display specific process information
SHOW STACK	Display process/interrupt stacks
SHOW SUMMARY	Display process summary
SHOW SYMBOL	Display symbol table

Three help files within SDA will provide explanations.

- HELP<command-name> briefly explains a command
- HELP SDA briefly explains command format
- HELP briefly explains the SDA utility

Detailed explanations of several SDA commands follow.

Evaluate Command – This command computes the value of an SDA expression.

EVALUATE <expression>

<expression> specifies the expression to be evaluated.

EVALUATE computes the value of any SDA expression and displays the results in hexadecimal and decimal.

```
SDA> EVALUATE -1
Hex = FFFFFFFF     Decimal = -1

! SDA prints the value of negative 1 in hex and decimal.

SDA> DEFINE TEN = A
SDA> EVALUATE A
Hex = 0000000A     Decimal = 10

! SDA evaluates the symbol TEN and prints the results.

SDA> EVALUATE ((TEN*6)+(-1/3))*(2+4)
Hex = 00000166     Decimal = 358

! SDA evaluates a complex expression and prints the results in hex
! and decimal.
```

Example 7-1 SDA Evaluate Command

Examine Command – This command displays the contents of a location or range of locations in physical memory.

EXAMINE <location>[:<location>]

<location>[;<length>]

Command Qualifiers

/PO

Prints the entire program region for a given process. You must SET PROCESS to the process whose PO region you want to examine. Otherwise, you will get a dump of the PO region belonging to the process which was executing at the time of the crash.

/P1
Prints the entire control region for a given process. You must SET PROCESS to examine different P1 regions. The default P1 space is the process to examine different P1 regions. The default P1 space is the process that was executing when the system crashed.

/SYSTEM
Prints portions of the writable system region.

/ALL

Parameters

<location>
Specifies the address in virtual memory at which data is stored.

<length>
Specifies the number of bytes you want to display.

You may use command parameters to examine specific locations or command qualifiers to display entire process and system regions. There are two ways to examine a range of locations.

1. Designate a location followed by a colon and an ending location (e.g., 80000000:80000030).
2. Specify a location and a byte length, separating the two values with a semicolon.

If, at any time, you omit the command parameter from the examine command, SDA takes the location you last examined, increments it by four (one longword), and examines the resulting location.

Examining Specific Locations

A location may be represented by any valid SDA expression. When you examine a location, SDA displays the location and its contents in hexadecimal and in ASCII.

SDA initially sets the "current location" to –4 (decimal) in the program (P0) region of the process which was executing at the time of the crash. To examine memory locations in different processes, you must SET PROCESS to the process whose memory you want to examine.

```
SDA> EXAMINE 80000200
SYS$SETEF  :  8FBC003C     "<..."

! The system virtual address is defined by a global symbol.  The
! information stored at this address is given in hex and in ASCII.
! SDA represents unprintable characters by ".".

SDA> EXAMINE PC
PC : 8002484C  "LH.."
SDA> EXAMINE @PC
8002484C  :  00DD00DD "...."

! SDA examines the program counter and the location referenced by
! the program counter.

SDA> EXAMINE 80000008;11

! SDA displays a range of bytes starting at address 80000008 and
! ending at 80000027.  SDA displays byte ranges in units of 16
! (decimal) bytes.  In this case, SDA displays two lines of 16
! bytes even though a value of 17 (11 hex) was given.
```

Example 7-2 SDA Examine Command

Examining Memory Regions

You may dump an entire region of virtual memory by adding one or more of the command qualifiers to the examine command.

SDA organizes the dump into columns of longwords, 4 for an 80 column device and 8 for 132 column device, and prints the ASCII value of the longwords on the right side of the display. The final column contains the address of the first longword in each line. Read the dump display from right to left.

If a series of virtual addresses does not exist in physical memory, SDA prints a message specifying the range of addresses which were not translated:

Virtual locations : (addr) are not in physical memory

If a range of virtual locations contains only zeros, SDA prints the message:

Zeros suppressed from (addr) to (addr).

Exit Command – The exit command stops SDA displays and ends use of SDA.

EXIT

The exit command has two functions: discontinuing SDA displays, and exiting from the utility. During interactive sessions with SDA, if a display has more than one page and is being shown on a terminal such as a VT52 or VT100, SDA will issue this message each time it reaches the bottom of a page:

Press RETURN for more.

SDA>

At this point, you type EXIT if you want to discontinue the current display.

To stop running SDA, type EXIT at the regular SDA prompt.

NOTE

If you do not type EXIT at the RETURN message and simply execute another command, SDA will accept the command as if you had exited from the display.

Set Output – This command causes SDA to write displays to a file or device.

SET OUTPUT <file-spec>

<File-spec> specifies the device, directory, and/or file to which SDA output will be written. The default file specification is SYSDUMP.LIS.

SET OUTPUT writes the output of SDA commands to a file or device of your choice. If you have set output to a file, SDA will create a table of contents that identifies the displays you selected.

Once you set SDA output to a file or device, you are locked out of interactive operation. After you have issued all the commands you want to execute, you must exit from the utility and then recall SDA, or issue another SET OUTPUT to the terminal device.

```
SDA> SET OUTPUT BROKEN
SDA> SHOW CRASH
SDA> SHOW PROCESS/ALL
SDA> SHOW SUMMARY
SDA> EXIT

! SDA stores the displays produced by the commands following SET
! OUTPUT in a file called BROKEN.LIS.
```

Example 7-3 SDA Set Output Command

Set Process Command – This command sets the process default to a specific process so that information and memory locations can be examined by later commands.

SET PROCESS [<name>] [/INDEX=<nn>]

<name>
A 1 through 15 character alphanumeric string assigned to the process. The symbols "$" and "_" may be included in the string.

INDEX=<nn>
nn is the index to the software PCB and is composed of the last two hexadecimal digits of the process identification number (PID). You may specify the process using either name or index number, but you must use only one of them or SDA will issue a syntax error message.

The main function of the SET process command is to permit you to examine per process virtual memory and data structures. SET PROCESS is a functional command. It locates the information needed for the particular process but produces no output.

```
SDA> SET PROCESS/I=43
SDA> EXAMINE/P0

! SDA locates the process via the index number and displays the
! contents of its program region.

SDA> SET PROCESS GROVE
SDA> SHOW DEVICE

! Setting the process to GROVE causes the SHOW DEVICE command to
! default to GROVE rather than the process which was executing at
! the time of the crash.
```

Example 7-4 SDA Set Process Command

Show Crash Command – The show crash command lists fundamental information concerning the operating system and the process currently executing at the time of the crash.

SHOW CRASH

The display produced by SHOW CRASH can be divided into three parts.

- operating system and process information
- general and special register contents
- processor and hardware register contents

Operating System and Process Information

The first section of SHOW CRASH lists the following.

- date and time of crash
- name and version number of operating system
- name of process executing at time of crash
- file specification of image executing in process context (left blank if no image was executing)
- interrupt priority level (in decimal) of the processor

General Purpose and Special Register Contents

The second part of the SHOW CRASH display lists the contents of the general purpose and special registers.

- R0-R11
- AP (Argument Pointer)
- FP (Frame Pointer)
- SP (Stack Pointer)
- PC (Program Counter)
- PSL (Processor Status Longword)

Process and Hardware Maintenance Register Contents

The third part of the SHOW CRASH display lists the contents of three sets of registers. The first set includes registers that store the vital statistics of the process executing when the system crashed, as well as registers that contain information used by the operating system. The second set of registers contains the stack pointers for the processor access modes and the interrupt stack. The third set of registers is used in hardware maintenance.

Per Process and System Registers:

- P0BR Program Region Base Register
- P0LR Program Region Length Register
- P1BR Control Region Base Register
- P1LR Control Region Length Register
- SBR System Region Base Register
- SLR System Region Length Register
- PCBB Process Control Block Base Register
- SCBB System Control Block Base Register
- ASTLVL Asynchronous System Trap Level
- SISR Software Interrupt Summary Register

Stack Pointers:

- ISP Interrupt Stack Pointer
- KSP Kernel Stack Pointer
- ESP Executive Stack Pointer
- SSP Supervisor Stack Pointer
- USP User Stack Pointer

Hardware maintenance registers are processor dependent.

Show Device Command – This command displays a formatted list of all data structures associated with a device.

SHOW DEVICE [<device-name>]

<device-name>
Specifies the name of a device whose data structures you want to display. Device-name takes the form "ddcu", where:

dd = device type (two characters)
c = controller
u = device unit.

You may display information about several devices or a single unit by specifying device-type, (for example, DB), device-type and controller (DBA), or the full device-name (DBA0). If you do not specify a device-name, SDA will list the data structures of every device on the system as they existed at the time of the crash.

The display for each device is divided into three sections.

- device data block list
- controller data structures
- device unit data structures

The sections that follow outline the information contained in these areas.

Device Data Block List

The device data block list shows information common to all devices associated with a single controller.

- address of controller
- name of controller
- name of ancillary control process (ACP)
- name of I/O driver

Controller Data Structures

SDA displays the contents of four data structures associated with each controller.

- Device Data Block (DDB). This points to the driver dispatch table, the channel request block, and the first unit control block connected to the controller.
- Channel Request Block (CRB). This stores information used to arbitrate requests between devices attached to a single controller.
- Interrupt Dispatch Block (IDB). This contains controller status information used to dispatch interrupts to the proper driver.
- Driver Dispatch Table (DDT). This points to routines used to process the I/O request.

Device Unit Data Structures

The final section of the SHOW DEVICE display itemizes the contents of the unit control block for each device. If the device handles file-structured requests, the display lists the volume control block and the ACP queue as well.

Unit Control Block

SDA organizes the data stored in the UCB into a list of items. Heading the list are the address of next UCB, the status of the device, and the longword whose bits express various characteristics of the device.

Following the heading, SDA lists pointers to other block types.

- IRP (I/O Request Packet) address
- CRB (Channel Request Block) address
- VCB (Volume Control Block) address

The next six items on the list deal with the fork block for the device driver.

- FQFL (Fork Queue Forward Link)
- FQBL (Fork Queue Backward Link)
- Fork IPL (Interrupt Priority Level)
- Fork PC, R3 and R4

The UCB also contains device status information.

- Device class
- Device type
- DEVBUFSIZ (Device Buffer Size)
- DEVDEPEND (Device Dependent data longword)
- DEVSTS (Device Status longword)
- Device IPL

The next two quantities displayed are counters.

- reference count
- operations count

The final items detailed concern mailboxes and information obtained from the I/O request packet.

- AMB address (Associated Mail Box)
- SVPN (System Virtual Page Number)
- SVAPTE (System Virtual Page Table Entry)
- BOFF (Byte Offset)
- BCNT (Byte Count)
- ERTCNT (Error Retry Count)
- ERTMAX (Error Retry Maximum)
- ERRCNT (Error Count)
- Owner UIC
- PID (Process ID)

SDA also formats the I/O request queue, another area of data stored in the UCB. Information concerning the request at the head of the queue is listed in the following order across the page.

- CHAN (Channel Number)
- FUNC (Function value)
- WCB (Window Control Block)
- EFN (Even Flag Number)
- AST (Asynchronous System Trap)
- IOSB (I/O Status Block)
- STATUS

If the request queue is empty, SDA will issue the message.

*** I/O request queue is empty ***

Volume Control Block and ACP Queue

If a volume has been mounted on a device, SDA will read and display the contents of the volume control block and the ACP queue block. The volume control block (VCB) identifies the volume and contains counts and quotas concerning files on the volume.

The ACP queue block (AQB) contains information about the ACP associated with the volume. SDA reads the AQB and lists its contents in readable format.

If the request queue is empty, SDA prints the message.

*** ACP request queue is empty ***

Show Summary Command – This command displays a formatted list of processes which were active when the system crashed.

SHOW SUMMARY

SHOW SUMMARY displays values used in swapping and scheduling for all processes. The information listed in the display includes the following.

- PID – The 32-bit quantity which uniquely identifies the process. This is the process identification and sequence number.
- PROCESS NAME – The name assigned to the process. This may be from 1 through 15 alphanumeric characters long.
- IMAGE NAME – The VAX/VMS file specification of the image currently executing under the process.
- STATE – The condition of the process at the time of the crash.
- PRI – The current scheduling priority of the process.
- UIC – User Identification Code.
- WKSET – The total number of pages currently in the working set.

NOTE

If the process has been swapped out of the balance set, the message --- SWAPPED OUT --- will appear in the IMAGE NAME column.

USING THE ERROR LOGGING FACILITY AND SYE

The VAX/VMS error logging facility detects a variety of hardware and software errors. When an error occurs, the facility gathers significant information about the current state of the system. The type of information gathered depends on the type of error detected. In addition to detecting actual errors, the facility monitors events that reflect other aspects of system performance. The recording of such events helps to define the system context in which actual errors occur.

The errors detected include :

- device errors
- asynchronous write errors

- corrected read data
- interconnect errors
- fatal hardware errors such as parity errors in cache memory or in a translation buffer (machine checks)
- fatal software errors (bugchecks).

The system events recorded include:

- normal system start up (cold start)
- recovery after a power failure (warm start)
- cold start after a crash (crash restart)
- volume mounts and dismounts
- error log messages sent by an operator or by a process that issues a send message to error logger system device ($SNDERR)
- Time stamps that indicate that no errors or events have occurred within a given period of time.

The error logging facility stores information on all these errors and events in a file on the system disk. This file becomes the input to the report generator program SYE. Depending on how a user invokes it, SYE reports either on all the errors in the file or on a specific subset of errors. Parameters to SYE also determine the level of detail to be included in a report.

Error reports allow the system manager to track system performance and to anticipate failures. Field service engineers should interpret the reports as aids to both corrective and preventive maintenance.

Error Logging Facility Components

The error logging facility consists of three working parts.

- a set of executive routines that detect errors and events and write relevant information into error log buffers in memory

- a process called ERRFMT that periodically empties the error log buffers, transforms the descriptions of the errors into standard formats, and stores the formatted information in a file on the system disk
- a process called SYE that generates readable reports from the information formatted by ERRFMT

Printing the Error Log File

The following procedure shows how you can create an error log report and how to obtain a copy of it.

1. Set the default disk to the system disk and the default directory [SYSERR] by typing the following commands:

   ```
   $ SET DEFAULT SYS$SYSTEM
   $ SET DEFAULT SYS$DISK: [SYSERR]
   ```

2. Examine the [SYSERR] directory to see what versions of the ERRLOG.SYS file are on disk by typing.

   ```
   $ DIRECTORY ERRLOG.SYS;*
   ```

3. Rename all the versions of the ERRLOG.SYS file to ERRLOG.OLD by issuing the command.

   ```
   $ RENAME ERRLOG.SYS;* ERRLOG.OLD;*/NEW VERSION
   ```

4. Invoke the SYE utility by typing the command:

   ```
   $ RUN SYS$SYSTEM:SYE
   ```

5. Enter the following file name in response to the input file prompt (input file:):

   ```
   ERRLOG.OLD
   ```

   This is the file created in step 3 of this procedure. By default, SYE uses the highest version of the ERRLOG.OLD file.

6. Obtain a copy of the error log report by entering the following command:

   $PRINT <filename>

   The file name is the name of the file entered in response to the output file prompt (output file.). Example 7-5 shows the whole procedure.

```
$ SET DEFAULT SYS$SYSTEM
$ SET DEFAULT SYS$DISK:[SYSERR]
$ DIRECTORY ERRLOG.SYS

DIRECTORY DBB2:[SYSERR]
18-JUL-78  15:13

ERRLOG.SYS;1       14.       18-JUL-80            13:48

TOTAL OF 14./18. BLOCKS IN 1. FILE

$ RENAME ERRLOG.SYS;* ERRLOG.OLD;*/NEW VERSION
$ RUN SYS$SYSTEM:SYE

SYE x0.6-0

_input file:       [[1,6]ERRLOG.SYS]          ? ERRLOG.OLD
_output file:      [SYS$OUTPUT]               ? ERRLOG.DAT
_options:          [ROLL-UP]                  ? R
_device name:      [<all>]                    ? <CR>
_after date:       [17-NOV-1858]              ? <CR>
                                              17-NOV-1858
                                              00:00:00.00
_before date:      [31-DEC-9999]              ? <CR>
                                              31-DEC-9999
                                              23:59:59.99

Successful completion
$
```

Example 7-5 Error Logging Commands

The SET DEFAULT commands set the operator's default disk and directory to DBB2:[SYSERR]. The DIRECTORY command lists all the ERRLOG.SYS files contained in the [SYSERR] directory. In this example, [SYSERR] contains only one version of ERRLOG.SYS. The RENAME command renames ERRLOG.SYS to ERRLOG.OLD. The/NEW_VERSION qualifier requests that ERRLOG.OLD be assigned a new version number if a file of this name already exists.

The operator then invokes the SYE utility by typing RUN SYS$SYSTEM:SYE. SYE prompts for the following six parameters:

- The name of the file to be manipulated. The operator responds with ERRLOG.OLD.
- The name of the file that is to contain the error log report. The operator responds with ERRLOG.DAT.
- The type of report that SYE should generate. The operator responds with R, which indicates the ROLL UP report.
- The devices on which SYE should report errors. The operator responds by pressing the RETURN key, which requests SYE to report on all devices.
- The time after which SYE should report errors. The operator responds by pressing the RETURN key, which requests SYE to report on all errors occurring after November 17,1858.
- The time up to which SYE should report errors. The operator responds by pressing the RETURN key, which requests SYE to report the occurrence of errors until December 31, 9999.

SYE creates the error log report and stores it in the ERRLOG.DAT file. The operator obtains a hard copy of this report by using the print command.

Device Errors

You may request that all device errors be reported, or only those which occur on one or more devices specified by a device name. SYE prompts for the device name by typing

```
device name:  [<all>]  ?
```

By default, errors on all devices are reported (that is, if only a carriage return is typed, all error types are inspected).

If a device name is specified, then device errors and mount/dismount messages whose device names match are selected for further inspection.

SYE will accept generic device names, allowing you to specify that errors be reported for all devices of a particular type (for example, DB:) for devices attached to a particular controller (for example, DBA:), or for a particular device (for example, DBA1:).

When you specify a device name to SYE, you may also request that it report one of three special classes of errors.

CP Hardware errors other than device errors which include machine checks, corrected read data, read data substitutes, and asynchronous write errors.

CO Configuration changes which include mount and dismount messages.

SY System information which includes system startup, power recovery, crash/restart, system service and network messages, and system and user bugchecks.

Although time stamp messages are included in the summary totals under system information, they are not included in this option.

You can also use a device-name to deselect one device class or special class by prefixing the name with a minus sign(–). For example:

```
device-name:  [<all>]  ?  -DMA1:
```

This command string instructs SYE to report on all errors other than DMA1: errors. The device-name -SY would cause all errors except system information entries to be reported.

This method can be used only to exclude one device or one special class of device.

Appendix

TROUBLESHOOTING

The flowcharts that follow should help you repair VAX computer systems quickly, with minimum adverse impact on the customer's application. See the appropriate processor-specific diagnostic system overview manual for more detailed information. The DIGITAL Diagnostic Center (DDC) is always available for technical assistance.

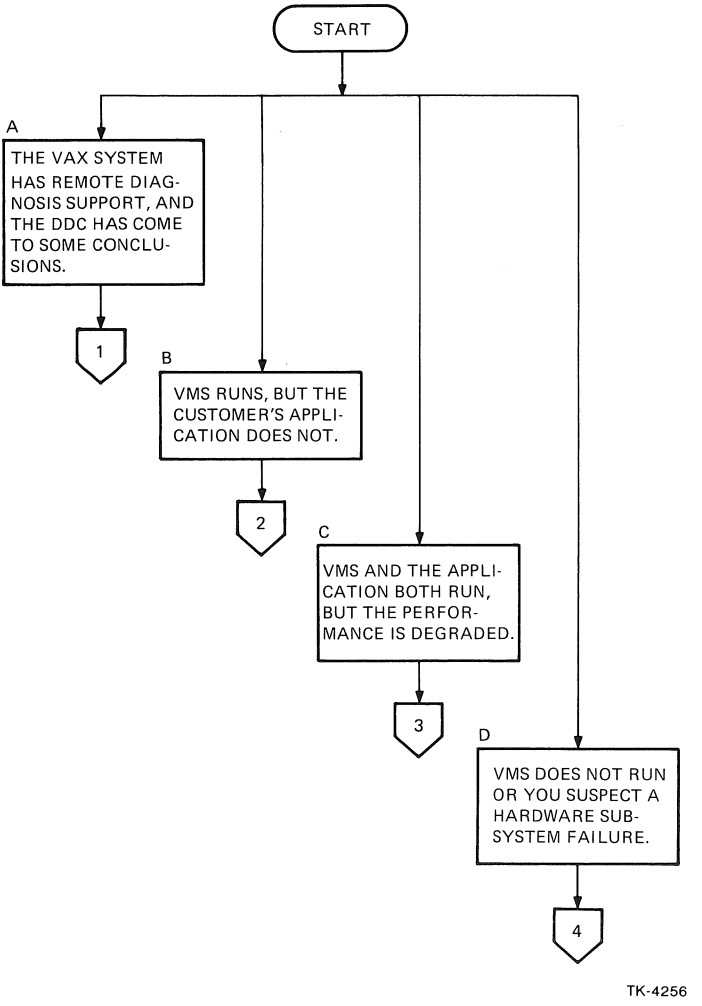
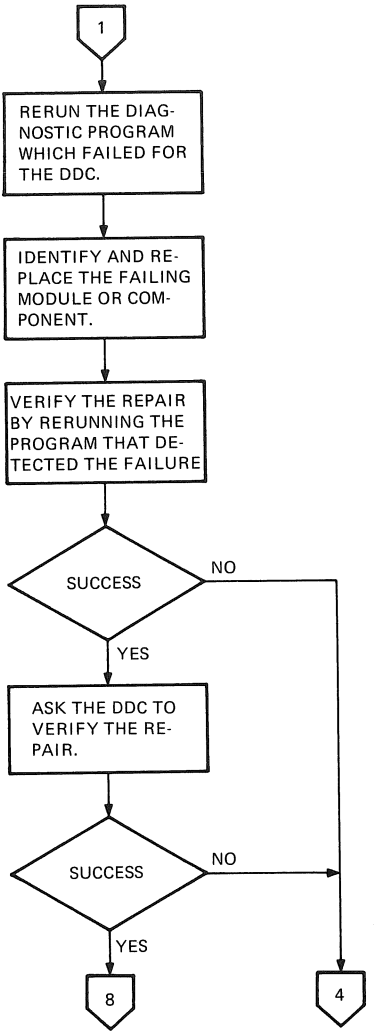


Figure A-1 VAX Troubleshooting Flowchart (Sheet 1 of 11)

1 THE VAX SYSTEM HAS REMOTE DIAGNOSIS (RD) SUPPORT, AND THE DDC HAS COME TO SOME CONCLUSIONS.



TK-4257

Figure A-1 VAX Troubleshooting Flowchart (Sheet 2 of 11)

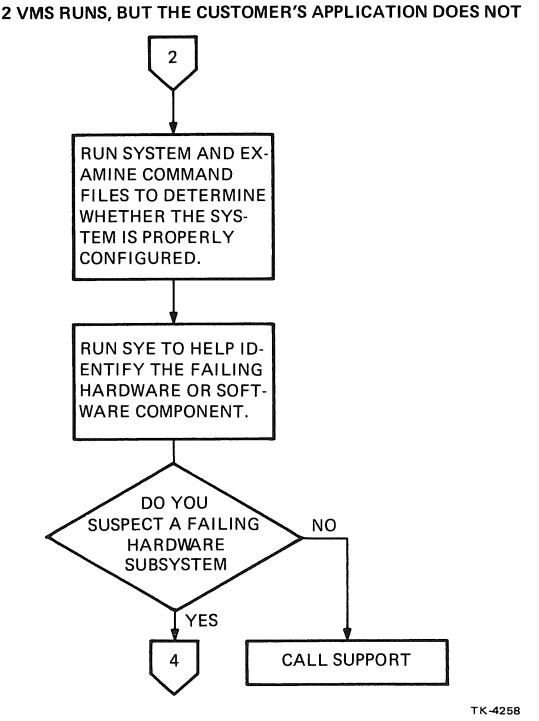


Figure A-1 VAX Troubleshooting Flowchart (Sheet 3 of 11)

3 VMS AND THE CUSTOMER'S APPLICATION BOTH RUN, BUT PERFORMANCE IS DEGRADED. THIS SITUATION IMPLIES THAT THE CUSTOMER HAS IDENTIFIED AND CONFIGURED AROUND THE PROBLEM.

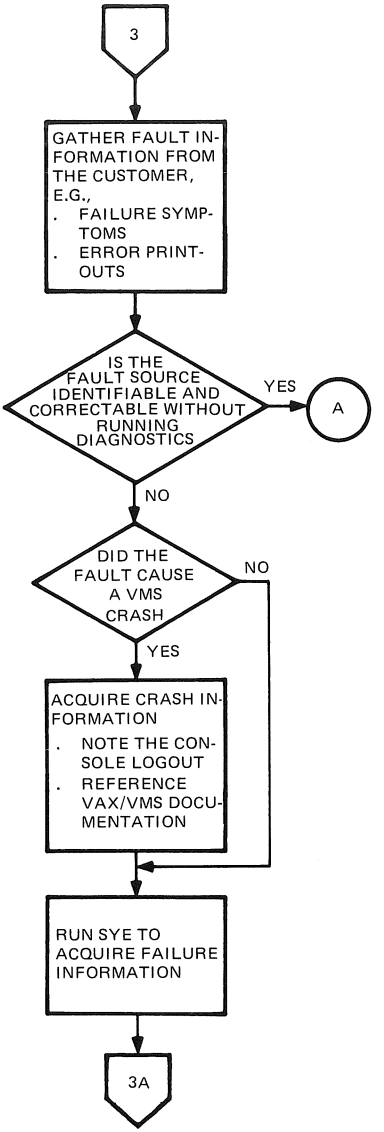


Figure A-1 VAX Troubleshooting Flowchart (Sheet 4 of 11)

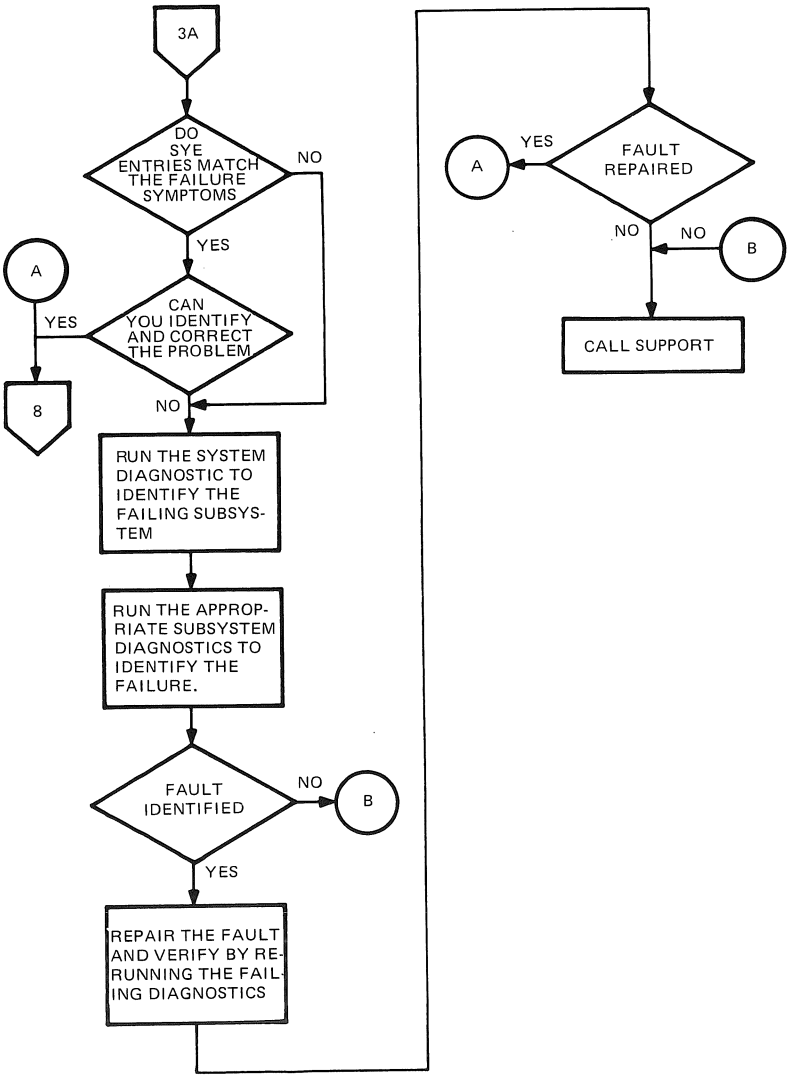


Figure A-1 VAX Troubleshooting Flowchart (Sheet 5 of 11)

4 VMS DOES NOT RUN, OR YOU SUSPECT A HARDWARE SUBSYSTEM FAILURE

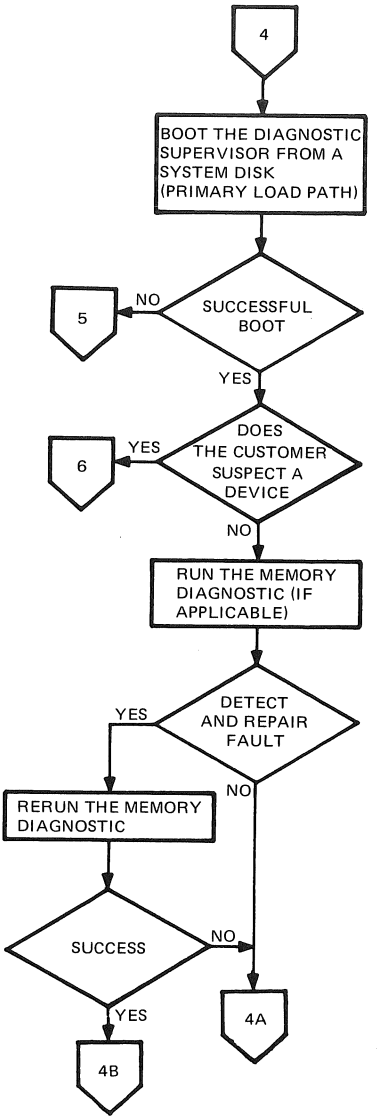


Figure A-1 VAX Troubleshooting Flowchart (Sheet 6 of 11)

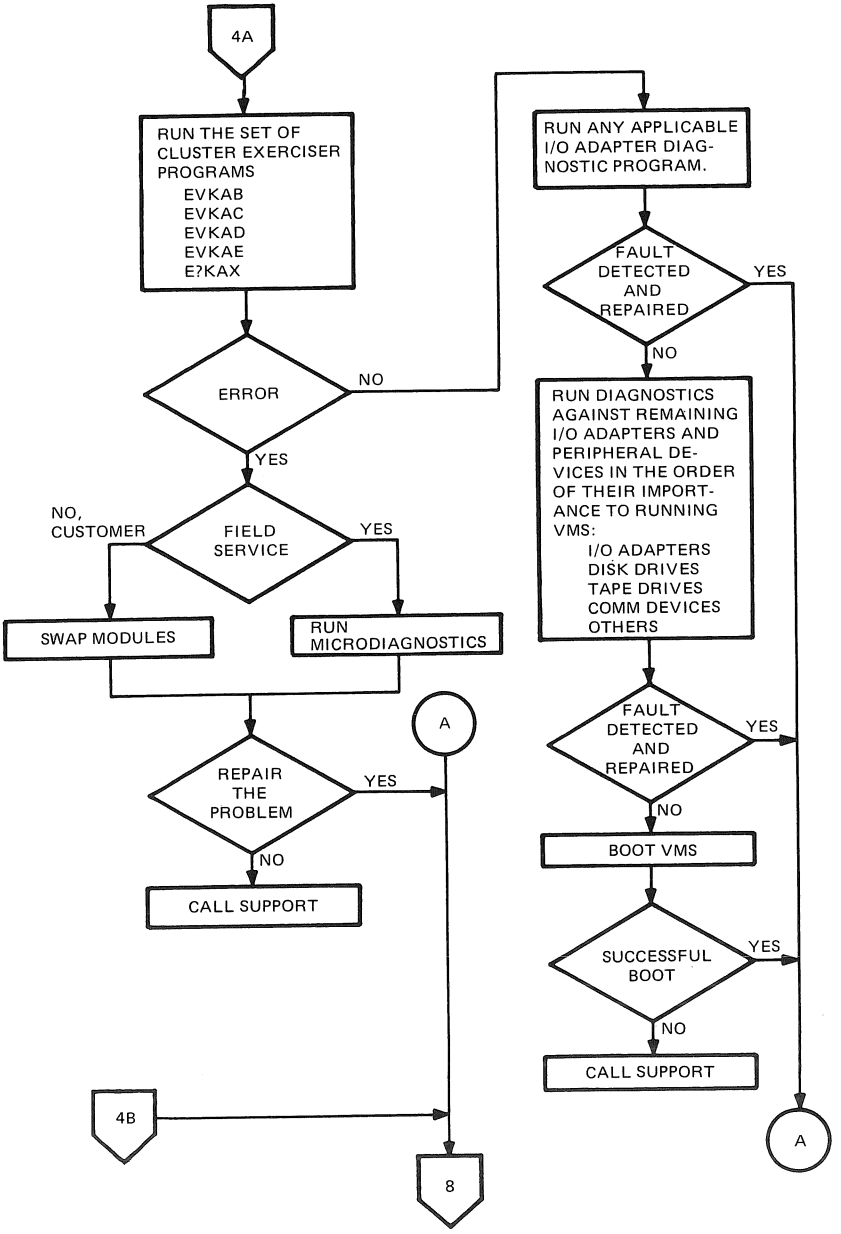


Figure A-1 VAX Troubleshooting Flowchart (Sheet 7 of 11)

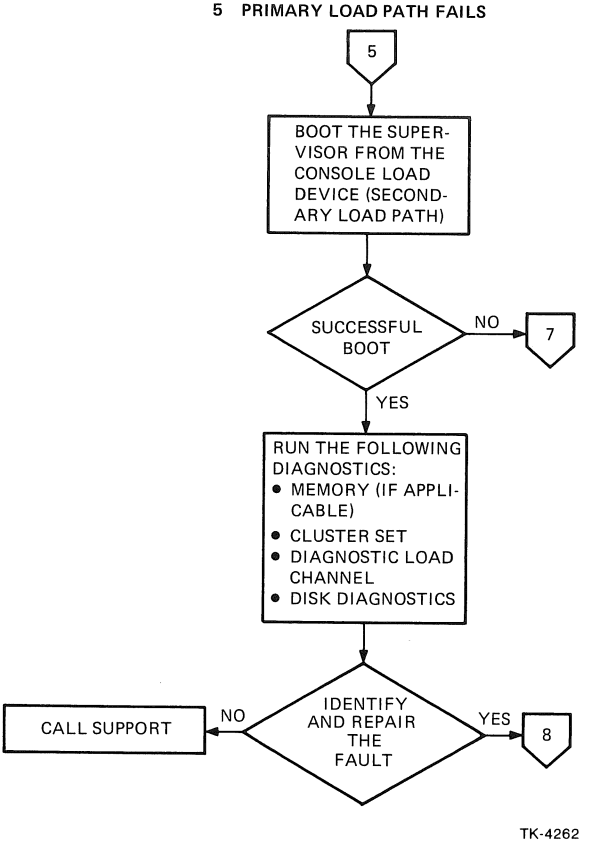


Figure A-1 VAX Troubleshooting Flowchart (Sheet 8 of 11)

6 THE CUSTOMER SUSPECTS A DEVICE AND YOU CAN BOOT THE SUPERVISOR

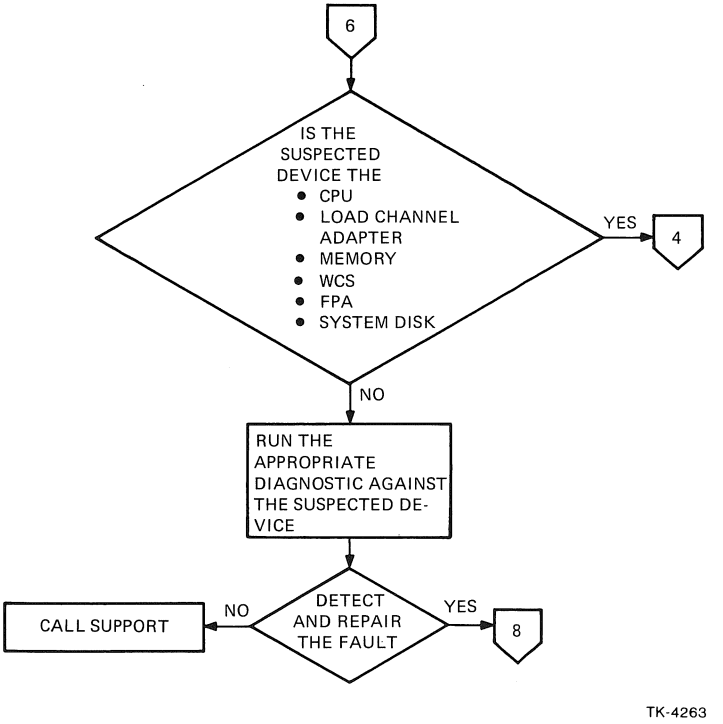


Figure A-1 VAX Troubleshooting Flowchart (Sheet 9 of 11)

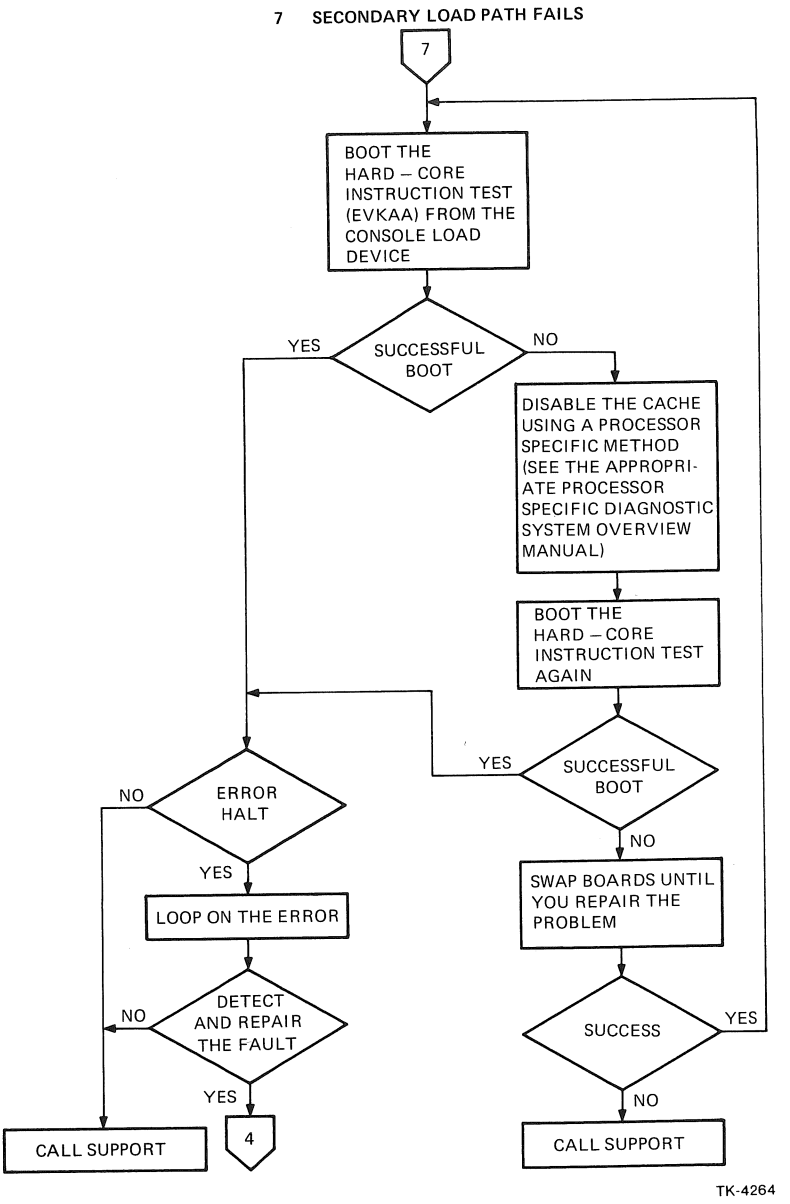


Figure A-1 VAX Troubleshooting Flowchart (Sheet 10 of 11)

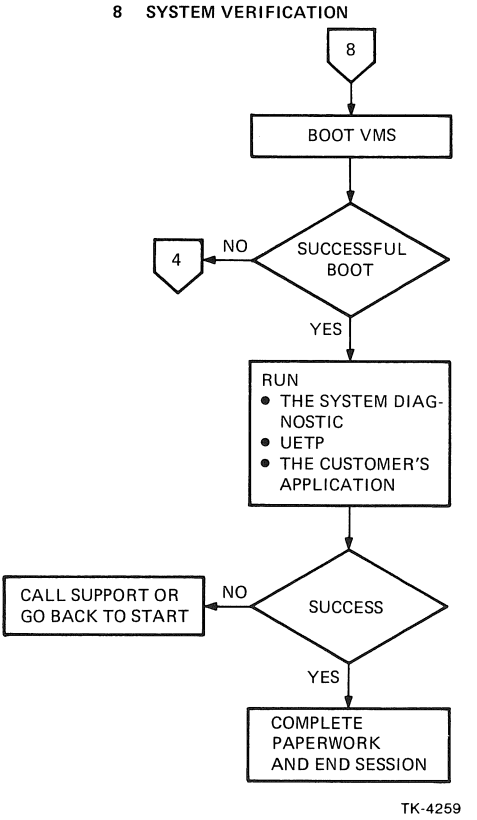


Figure A-1 VAX Troubleshooting Flowchart (Sheet 11 of 11)

Glossary

APT – An automated product test application used throughout DIGITAL manufacturing.

Argument – An independent value within a command statement that specifies where, how, or on what the command will operate. Parameter.

Assembler – The program that translates source language code into object language code. On VAX computers the assembler is MACRO-32.

BLISS-32 – A middle level software development language, having advantages of both higher and lower level languages.

Boot (bootstrap) – A program that loads another (usually larger) program into memory.

Breakpoint – In diagnostics, an address assigned through the diagnostic supervisor. When the PC equals the value of the breakpoint, control returns to the diagnostic supervisor.

Buffer – A temporary data storage area.

Channel – A logical path connecting a user process to a physical device unit. The MASSBUS adapters and UNIBUS adapters are channels.

CPU Cluster Environment – The console, CPU, memory, and I/O channels that support macro level program execution.

Console Environment – Hardware, software, and firmware in the console and CPU that operate without macro level program execution.

Command File – A file containing command strings.

Command Line Interpreter – Code that receives, checks, and passes commands typed by the user at a terminal or submitted in a command file.

CPU – Central processor unit.

DDC – DIGITAL Diagnostic Center; the location from which DIGITAL Field Service conducts remote diagnosis.

Diagnostic Supervisor – A program that is loaded in memory to provide a framework for control and execution of diagnostic programs. It provides non-diagnostic services to diagnostic programs.

Direct I/O – A mode of access to peripheral devices in which the program accesses device registers directly, without relying on support from the operating system drivers.

Driver – The set of operating system code that handles physical I/O to a device through QIO services.

Documentation File – A listing available on microfiche that describes a given diagnostic program.

Event Flag – Status posting bits maintained by VMS and the diagnostic supervisor. Diagnostic programs often use event flags to perform a variety of signaling functions, including communication with the operator.

File Specification – A unique name for a file on a mass storage medium.

Hard-Core – That portion of a computer system assumed to be fault-free in a given diagnostic situation. A diagnostic program will yield valid results only when the hard-core for that program is fault-free.

ISP – Instruction set processor. That portion of a computer system that executes macro level instructions.

Level 1 – Operating system (VMS) based diagnostic programs using logical or virtual I/O references with QIO services.

Level 2 – Diagnostic supervisor based diagnostic programs that can be run either under VMS (on-line) or in the standalone mode, using physical I/O references with QIO services.

Level 2R – Diagnostic supervisor based diagnostic programs that can be run only under VMS, using physical I/O references with QIO services.

Level 3 – Diagnostic supervisor based diagnostic programs that can be run in standalone mode only, using direct I/O.

Level 4 – Standalone macro level diagnostic programs that run without the supervisor.

Link Map – A listing that shows the virtual memory allocation of the program image. The first part of a program listing contains the link map.

Load Path Diagnostics – A set of diagnostic programs designed to run when the primary load path fails. You can load the load path diagnostics from the console load device, the secondary load path. Use these programs to test the primary load path.

MACRO-32 – The native assembler for VAX computers.

Microdiagnostics – Diagnostic programs that test the CPU cluster but for which control remains in the console processor.

On-Line Diagnostics – Diagnostic programs that run under the operating system (VMS).

Parameter – An independent value within a command statement that specifies where, how, or on what the command will operate. Argument.

Physical QIO – A set of I/O functions that allow access to all device level operations except maintenance mode operations.

Primary Load Path – That portion of a VAX system that includes the CPU cluster and the mass storage device from which VMS or the diagnostic supervisor is booted.

Program Module – A portion of a program that is assembled as a unit and then linked with other modules. Program listings are arranged according to modules.

Prompt – The symbol displayed on the operator's terminal by a program which tells the operator that he should type a command or other information. The diagnostic supervisor prompt is DS>.

Qualifier – An argument used to modify the function of a command.

QIO – Queue I/O, the VMS and diagnostic supervisor service that enables a program to communicate with a device via a device driver routine.

Scope Loop – A portion of a diagnostic test that enables the operator to loop on a hardware failure at machine speed.

Script File – A line-oriented ASCII file that contains a list of commands.

SDA – System Dump Analyzer, a program under VMS that enables you to analyze and display parts of a formatted system dump file after recovery from a crash.

Secondary Load Path – The console and that portion of the CPU cluster necessary to boot diagnostic programs from the console load device.

Section – A portion of a program that consists of a group of tests. If you run a diagnostic program without specifying a section, only the default section will run.

Standalone Mode – The mode of VAX diagnostic system operation that enables you to run diagnostic programs without VMS.

Subtest – A portion of a test in a diagnostic program that tests a small section of logic or a specific function. With arguments to the start and run commands in the supervisor, you can cause a program to loop on a specific subtest.

SYE – System Error log report generator program. You can use SYE to report all errors or a subset of errors stored in the system error log.

Symbol Cross Reference – An alphabetical list of global symbols used in the program. A value is given for each symbol.

SYSMAINT – The name of the directory containing the diagnostic system files on disk storage.

System Environment – The environment provided by the standalone diagnostic supervisor. Level 2 and level 3 programs run in the system environment.

Test – A unit of a diagnostic program that checks a specific function or portion of the hardware.

UETP – User Environment Test Package, a program under VMS that checks the integrity of the VAX system hardware and software. This is not a diagnostic program.

User Environment – The environment provided by the diagnostic supervisor when it runs under VMS. Level 2 and level 2R diagnostic programs will run in the user environment.

User Mode – A mode of diagnostic testing that runs under VMS while customer applications are running.

Virtual QIO – A set of I/O functions that must be interpreted by an ancillary control process.

VMS – Virtual Memory System, the operating system for VAX computers.