digital

VAX/VMS
System Dump Analyzer Reference Manual

Order No. AA-J526A-TE

VAX11

March 1980

This document describes how the VAX/VMS System Dump Analyzer works and how to use it.

VAX/VMS
System Dump Analyzer Reference Manual

Order No. AA-J526A-TE

SUPERSESSION/UPDATE INFORMATION: This is a new document for this release.

OPERATING SYSTEM AND VERSION: VAX/VMS V02

SOFTWARE VERSION: VAX/VMS V02

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, March 1980

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

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

CONTENTS

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

The VAX/VMS System Dump Analyzer Reference Manual contains information useful in determining the cause of a VAX/VMS operating system failure.

INTENDED AUDIENCE

This reference manual is intended for users who possess extensive knowledge of VAX/VMS data structures. It assumes that the audience for this manual includes VAX/VMS developers and DIGITAL Software Support Specialists, as well as DIGITAL customers familiar with VAX/VMS internal design.

In addition, system programmers who are writing device drivers may need to use SDA. The system manager should also become familiar with SDA, usually to produce SDA listings after each crash and, more importantly, to save the system dump file for later analysis.

STRUCTURE OF THIS DOCUMENT

This reference manual consists of seven chapters:

- Chapter 1 provides an introduction to SDA and summarizes SDA operations.
- Chapter 2 describes the system dump file that SDA analyzes.
- Chapter 3 explains how to run SDA to analyze a dump file or examine the running system.
- Chapter 4 details the SDA command format.
- Chapter 5 describes the SDA commands, in alphabetical order.
- Chapter 6 gives guidelines for analyzing system failures and steps through a sample system crash.
- Chapter 7 lists and explains the messages related to SDA operation.

ASSOCIATED DOCUMENTS

This document has the following prerequisites:

VAX-11/780 Hardware Handbook
VAX/VMS Summary Description and Glossary

The following documents are associated with this manual:

VAX-11 Run-Time Library Reference Manual
VAX/VMS Guide to Writing a Device Driver
VAX/VMS System Manager's Guide
VAX/VMS System Services Reference Manual

For a complete list of all VAX-11 documents, including brief descriptions of each, see the VAX-11 Information Directory and Index.

CONVENTIONS USED IN THIS DOCUMENT

The following conventions are used in this document.

Convention	Meaning
SHOW CRASH	Uppercase words and letters, used in examples, indicate that you should type the word or letter exactly as shown.
symbol-name	Lowercase words and letters, used in format examples, indicate that you are to substitute a word or value of your choice.
[ ]	Square brackets indicate that the enclosed argument is optional, except for brackets used in directory specifications.
SET PROCESS {name / /INDEX = nn / /SYSTEM}	Braces are used to enclose lists from which one element is to be chosen.
. . .	A vertical ellipsis indicates that not all of the statements in an example or figure are shown.
SHOW SYMBOL TEN TEN = 00000010	In examples of commands you enter and SDA responses, all output lines and prompting characters that SDA prints or displays are shown in black letters. All the lines you type are shown in red letters.
RET	A symbol with a 1- to 3-character abbreviation indicates that you press a key on the terminal.

CHAPTER 1

INTRODUCTION TO SDA

The System Dump Analyzer (SDA) is a VAX/VMS utility that aids in determining the cause of an operating system failure.

When an internal error occurs that interferes with normal operations, the operating system writes information concerning its status at the time of the system failure to a predefined system dump file. SDA examines and formats the contents of this file.

With the help of the SDA commands, you can display parts of the formatted system dump file on a video display terminal, or you can create hard copy listings.

SDA performs the following operations:

- Assigns a value to a symbol
- Examines memory of any process
- Formats block of data
- Displays device data structures
- Displays memory management structures
- Displays a summary of all processes on the system
- Displays the SDA symbol table
- Copies the system dump file
- Sends output to a file or device
- Reads symbols from any object module

In addition to analyzing the system dump file, SDA can perform the operations listed above on a running system without interrupting that system's operation.

While SDA provides a great deal of information, it does not analyze all the various control blocks and data contained in memory. Therefore, in the event of system failure, it is extremely important that customers send a copy of the system dump file to DIGITAL along with a Software Performance Report (SPR).

CHAPTER 2

THE SYSTEM DUMP FILE

Before the VAX/VMS operating system can write information to the system dump file, the system parameter DUMPBUG must be set. Normally, this parameter is enabled by default; to reset DUMPBUG, as well as other system parameters, consult the VAX/VMS System Manager's Guide.

If the DUMPBUG parameter is set and the operating system fails, the system writes the contents of the error log buffers, processor registers, and physical memory to the contiguous file SYSDUMP.DMP. SDA analyzes this file and produces formatted displays of its contents.

SYSDUMP.DMP is furnished as an empty file in the VAX/VMS software distribution kit. It is located in the system directory [SYSEXE] and its file size is initially small.

2.1 SETTING THE SYSTEM DUMP FILE SIZE

To preserve the continuity of the error log file and save all of physical memory, it is important to make sure that the dump file's size in blocks matches the individual system configuration.

To change the size of SYSDUMP.DMP, the system manager (or a user with similar privileges) runs a command procedure in the directory [SYSUPD] called SWAPFILES.COM. The command line is:

```
$ @[SYSUPD]SWAPFILES
```

The command procedure prompts you for paging, swapping, and dump file sizes. You can enter a new file size or simply press RET . If you enter a new file size, the command procedure creates a new system dump file. This new file will not be used by the operating system until after a system reboot.

To calculate the correct dump file size for your configuration, use the formula:

```
blocks = physical-memory-size-in-pages + 4
```

The four additional blocks store hardware context and error log buffers. You can also use the table provided in the VAX/VMS System Manager's Guide to find the correct size. This table lists recommended sizes for the three files affected by the SWAPFILES command procedure. The system manager's guide also gives detailed information on SWAPFILES.COM and on changing dump file size.

2.2 SAVING SYSTEM DUMP FILES

Every time the operating system writes information to SYSDUMP.DMP, it writes over whatever was previously stored in the file. For this reason, the system manager should save the contents of SYSDUMP.DMP after a system failure has occurred. One way to accomplish this is to copy the file to another directory. Use the DIGITAL Command Language (DCL) command COPY, as shown in the following example:

```
$ COPY SYS$SYSTEM:SYSDUMP.DMP;1 [SYSERR]SAVEDUMP.DMP
```

SDA also provides a COPY command. This command can be included in the series of SDA commands in the site-specific start-up procedure. Section 3.6 discusses the start-up procedure in more detail. The COPY command is explained in Chapter 5.

CHAPTER 3

RUNNING SDA

SDA can analyze a dump file or examine the running system. To make it possible for SDA to read the dump file, you need:

- Read access to SYSDUMP.DMP
- Read access to a copy of the system symbol table
- Enough virtual space for SDA to map the entire system dump file

To ensure that SDA has the correct amount of virtual address space, the running system must have the system parameter VIRTUALPGCNT equal to the size of the dump file plus 1000 pages. In addition, your page file quota (PGFLQUOTA in the user's authorization record created by running the User Authorization Program) must be at least the size of the dump file plus 1000 pages. See the VAX/VMS System Manager's Guide for information on system parameters and the User Authorization Program (AUTHORIZE).

3.1 INVOKING SDA WITH THE RUN COMMAND

If the above conditions are satisfied, you can invoke SDA by typing the following 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 (SYS$SYSTEM:SYSDUMP.DMP), press RET in response to the prompt. SDA will search the system directory (logical name SYS$SYSTEM) for SYSDUMP.DMP. To examine an older dump file, enter its file specification:

```
Enter name of dump file > [WIZARD]ACFCRASH.DMP
```

The default file specification for the system dump file is SYS$DISK:[default-dir]SYSDUMP.DMP where SYS$DISK and [default-dir] represent, respectively, the device and directory specified by the last SET DEFAULT command. (See the VAX/VMS Command Language User's Guide for a description of the SET DEFAULT command.)

If you want to examine the running system, type an asterisk (*) in response to the dump file prompt. See Section 3.3 for further details.

3.2 INVOKING SDA AS A FOREIGN COMMAND

You can also invoke SDA as a foreign command by using the DCL assignment statement:

```
$ SDA := $SDA
```

A foreign command is a command not known to the command interpreter that can be executed by entering a command string.

The dollar sign ($) indicates to DCL that the expression is a foreign command. Now you can specify a file or the asterisk as a parameter to the SDA command:

```
$ SDA [DUMPS]BADUCB
```

Defining SDA as a foreign command abbreviates SDA initialization because it eliminates the need to respond to the dump file prompt. For further information on the foreign command feature of DCL, see Appendix A of the VAX/VMS Command Language User's Guide.

You can also invoke SDA from the site-specific start-up procedure; Section 3.6 describes this method of calling SDA.

3.3 EXAMINING THE RUNNING SYSTEM

Occasionally, VAX/VMS encounters an internal problem that hinders system performance without generating a system failure. By allowing you to examine the running system, SDA provides the means to search memory for the solution to the problem without disturbing the operating system.

To examine the running system, invoke SDA as described in Section 3.1. SDA automatically sets the process context to your process. (See the description of the SET PROCESS command in Chapter 5 for a discussion of process context.)

To analyze the system dump file, SDA maps the entire file. By contrast, when SDA examines a running system, it retrieves only the information necessary to process a given command.

Because of the system's dynamic nature, use extreme caution when examining the running system. Although you can safely reference most locations, accessing certain portions of memory, such as I/O address space or nonresident process header pages that the current process does not own, causes the system to fail.

3.4 READING THE SYSTEM DUMP FILE

When you invoke SDA and specify the name of a dump file (or press RET) SDA gathers the data needed to create the displays from that dump file. Under certain conditions, the contents of general purpose or processor registers may not be saved in SYSDUMP.DMP.

For example, during console restart bugchecks, such as HALT, the VAX-11 LSI-11 console program destroys the contents of all the general purpose registers except the program counter and the processor status longword. SDA indicates in the SHOW CRASH display that the registers were wiped out by the console.

Processor registers may also be lost if the error log buffers in memory are full. When the operating system writes data to SYSDUMP.DMP, it creates an error log entry in the error log buffer that stores the contents of the processor registers. If the buffers are full, the contents of the registers are lost because the operating system cannot create an error log entry for them. Again, SDA prints a message in the SHOW CRASH display indicating that an error log entry for the registers does not exist.

Although the system dump file must be contiguous for the operating system to write information to it successfully, the file need not be contiguous for SDA to read it. Thus, if your copy of the system dump file is not contiguous, you will still be able to run SDA.

3.5 BUILDING THE SDA SYMBOL TABLE

After locating and reading the system dump file, SDA next attempts to read the system symbol table file. This file, named SYS.STB, contains all the global symbols used by the operating system. SDA's ability to read global symbols makes it easier to analyze a dump because you can examine locations by symbol rather than by virtual address.

SDA first looks for SYS.STB in the directory and device containing the system dump. If the file is not there, SDA looks for it in the system directory SYS$SYSTEM. Once SDA finds SYS.STB, it copies the file's contents to the SDA symbol table. If SDA cannot find the system symbol table file, it will not run.

When SDA finishes building its symbol table, it prints out a message identifying itself and the immediate cause of the crash:

```
VAX/VMS System dump analyzer

Dump taken on 28-Feb-1979  01:22:58.43
MTXCNTNONZ, Mutex count nonzero at system service exit

SDA>
```

The SDA> prompt indicates that the utility is ready to accept SDA commands. You can now use SDA interactively, send selected information to a file, or print selected information on a line printer. Refer to the description of the SET OUTPUT command in Chapter 5 for directions on setting up output files.

3.6 INVOKING SDA IN THE SITE-SPECIFIC START-UP PROCEDURE

Because an SDA listing is an important tool in determining the general nature of a system failure, it is a good idea to make sure that one is produced after every crash. The system manager can ensure the creation of an SDA listing by modifying the SYSTARTUP.COM file in [SYSMGR] to invoke SDA when the system is booted.

When called by the start-up procedure, SDA scans the system dump file for a flag that indicates whether SDA has processed the file. This flag is cleared each time the operating system writes to SYSDUMP.DMP, except in the case of an emergency shutdown (OPCCRASH.EXE). If the flag is clear, SDA executes the commands designated in the command procedure and sets the flag. If, however, SDA finds that the dump file flag is set, it exits without performing any of the specified commands. Thus, SDA will execute only if the system just failed.

To allow you to run SDA from the site-specific start-up procedure, the system parameter PQL_DPGFLQUOTA must equal the size of the system dump file plus 1000 pages. See the VAX/VMS System Manager's Guide for more information on system parameters.

The example below shows commands that might be added to the site-specific start-up procedure to produce an SDA listing after each crash.

```
$ !
$ !       Print dump listing if system just failed
$ !
$ RUN SYS$SYSTEM:SDA
SYS$SYSTEM:SYSDUMP.DMP
   COPY SYS$SYSTEM:SAVEDUMP.DMP    ! Save dump file
   SET OUTPUT LPA0:SYSDUMP.LIS     ! Create listing file
   SHOW CRASH                      ! Display crash
                                   ! information
   SHOW STACK                      ! Show current stack
   SHOW SUMMARY                    ! List all active
                                   ! processes
   SHOW PROCESS/PCB/PHD/REG        ! Display current process
   SHOW SYMBOL/ALL                 ! Print system symbol
                                   ! table
   EXIT
```

CHAPTER 4

SDA COMMAND STRINGS

The following sections describe the SDA command format and the types of expressions SDA uses within commands.

4.1 GENERAL FORMAT

SDA uses a command string format similar to that of the DIGITAL Command Language (DCL) interpreter. You issue commands in the general format:

```
command [parameter][/qualifier]   [!comment]
```

command

The name of an SDA command that tells the utility to perform a certain function. Commands can consist of one or more words, and can be abbreviated to the number of characters that make the command unique. For example, SH stands for SHOW and SE stands for SET.

parameter

The target of the command. For example, SHOW PROCESS GORK tells SDA to display the process GORK.

When a parameter is a file specification, the current default device and directory are represented as listed below.

Default	Meaning
SYS$DISK	Device specified in the most recent SET DEFAULT command
[default-dir]	Directory specified in the most recent SET DEFAULT command

See the VAX/VMS Command Language User's Guide for a description of the DCL command SET DEFAULT.

/qualifier

The name of a command qualifier that modifies the action of an SDA command. A qualifier is always preceded by a slash (/). Multiple qualifiers can follow a single parameter but must be delimited by slashes. Qualifiers can be abbreviated as long as they remain unique.

!comment

A comment. SDA ignores the exclamation point and all characters appearing after it on the same line.

4.2 EXPRESSIONS

Certain SDA commands allow expressions as command parameters. To create expressions, you can use:

- Radix operators
- Unary operators
- Binary operators
- Special operators
- Symbols

4.2.1 Radix Operators

Radix operators determine which base SDA uses to evaluate expressions. You can use one of three radix operators to specify the radix for a numeric value.

Operator	Radix	Example
^X	Hexadecimal	^X10
^O	Octal	^O30
^D	Decimal	^D16

The default radix is hexadecimal. SDA displays hexadecimal values with leading zeros and decimal values with leading spaces.

4.2.2 Unary Operators

SDA recognizes the following unary operators:

Operator	Function
+	Assigns positive value
-	Assigns negative value
@	Uses contents of location
G	Adds 80000000 to value
H	Adds 7FFE0000 to value

The unary operator G corresponds the first virtual address in system space, while the unary operator H corresponds to a convenient base address in a process's control region.

4.2.3 Binary Operators

SDA performs integer arithmetic on 32-bit operands. The characters indicating arithmetic operations are:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
@	Arithmetic shift

SDA carries out multiplication, division, and arithmetic shift before addition and subtraction. In division, SDA does not round integers, nor does it retain a remainder.

4.2.4 Special Operators

SDA uses parentheses as special operators. Expressions enclosed in parentheses are evaluated first. In the case of nested parenthetical expressions, SDA evaluates from innermost to outermost.

4.2.5 Symbols

Symbols are composed of 1 to 31 alphanumeric characters that can include the special characters dollar sign ($) and underline(_).

SDA copies symbols into its symbol table from the SYS.STB file. They can also be created by the DEFINE and READ commands.

In addition, SDA provides the following special symbols:

Symbol	Meaning
.	Current location
G	80000000
H	7FFE0000
R0-R11	General purpose registers
AP	Argument Pointer
FP	Frame Pointer
KSP	Kernel Mode Stack Pointer
ESP	Executive Mode Stack Pointer
SSP	Supervisor Mode Stack Pointer
USP	User Mode Stack Pointer
P0BR	Program Region Base Register
P0LR	Program Region Length Register
P1BR	Control Region Base Register
P1LR	Control Region Length Register
PC	Program Counter
PSL	Processor status longword

The register symbols correspond to the registers saved in the hardware context of the current process (see the description of the SET PROCESS command in Chapter 5). For example,

```
SDA> EXAMINE @USP
```

This command displays the first longword on the user mode stack.

CHAPTER 5

SDA COMMANDS

Table 5-1 lists the SDA commands and gives a brief explanation of their functions. The underlined characters represent command abbreviations.

Table 5-1
Summary of SDA Commands

Command	Function
COPY	Copies the dump file
DEFINE	Defines symbols and their values
EVALUATE	Performs computations
EXAMINE	Examines memory locations
EXIT	Exits from the display or from SDA
FORMAT	Formats data blocks
HELP	Prints help files
READ	Copies object module symbols
REPEAT	Repeats the last command
SET OUTPUT	Sets output to the device or file specification
SET PROCESS	Sets the process context to a specific process
SHOW CRASH	Displays crash information
SHOW DEVICE	Displays I/O data structures
SHOW PAGE_TABLE	Displays the system page table
SHOW PFN_DATA	Displays the PFN data base
SHOW POOL	Displays dynamic memory
SHOW PROCESS	Displays specific process information
SHOW STACK	Displays process/interrupt stacks
SHOW SUMMARY	Displays a summary of all processes
SHOW SYMBOL	Displays the symbol table

COPY

Each time the system fails, new information is written over the contents of SYSDUMP.DMP. The COPY command allows you to preserve the contents of SYSDUMP.DMP by copying it to another file. (The resulting copy does not have to be a contiguous file; see Section 3.4.)

In most cases, the system manager will include the COPY command in the SYSTARTUP.COM command procedure so that each time the system fails, SDA will copy the system dump file to another file.

Format

COPY output-file-spec

Qualifiers	Defaults
None	None

Parameters

output-file-spec

The device, directory, and file name to which SDA copies the system dump file. The default file specification is SYS$DISK:[default-dir]SYSDUMP.DMP. See the VAX/VMS Command Language User's Guide for more information about file specifications.

Examples

1. SDA> COPY SYS$SYSTEM:SAVEDUMP

   The COPY command takes the SYSDUMP.DMP file and copies it to the system device and directory SYS$SYSTEM under the file name SAVEDUMP.DMP.

DEFINE

The DEFINE command assigns a value to a symbol. SDA evaluates the expression before assigning it to the symbol. If the symbol is already defined, the new value simply replaces the old one.

Although both DEFINE and EVALUATE perform computations, DEFINE adds symbols used for temporary computations to the SDA symbol table, while EVALUATE simply performs the computation.

Format

DEFINE symbol { = / (SP) } expression

Qualifiers	Defaults
None	None

Parameters

symbol

A 1- to 31-alphanumeric character symbol you designate to represent a value. See Section 4.2.5 for a discussion of valid SDA symbols.

expression

An expression to be defined by the symbol. You can separate the expression from the symbol by a space or by an equal sign. See Section 4.2 for a discussion of SDA expressions.

Examples

1.
```
SDA> DEFINE BEGIN = 80058E00
SDA> DEFINE END = 80058E60
SDA> EXAMINE BEGIN:END
```

In this example, DEFINE delimits a range of address space. A subsequent EXAMINE command can then easily examine that section of memory locations. The symbols serve as reference points in memory.

2.
```
SDA> DEFINE NEXT = @PC
SDA> EXAMINE NEXT
00000454 :   1FDAF812   "...."
```

The temporary symbol NEXT defines the address contained in the program counter. SDA represents nonprinting characters by a period (.) and puts quotation marks around ASCII text. Refer to Section 4.2.5 for a discussion of SDA symbols.

3.
```
SDA> DEFINE VEC SCH$GL_PCBVEC
```

A symbol VEC has been assigned to a global symbol. Now you can access the memory location or value represented by the global symbol by specifying the symbol VEC.

4.
```
SDA> DEFINE COUNT = 4
SDA> DEFINE RESULT = COUNT*COUNT
SDA> EVALUATE RESULT
Hex =  00000010      Decimal =  16
```

The value 4 is symbolically defined and then used in an arithmetic expression.

EVALUATE

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

Format

EVALUATE expression

Qualifiers	Defaults
None	None

Parameters

expression

The expression to be evaluated. See Section 4.2 for a description of valid SDA expressions.

Examples

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

EVALUATE prints the values of negative 1 in hexadecimal and decimal.

2.
```
SDA> DEFINE TEN = A
SDA> EVALUATE TEN
Hex = 0000000A      Decimal = 10
```

EVALUATE computes and displays the value of the symbol TEN. In this example, the character "A" could also be a symbol. When SDA encounters a quantity that can either be a symbol or a hexadecimal expression, SDA first treats the quantity as a symbol and looks for it in the symbol table. If SDA cannot locate the quantity in the symbol table, it evaluates the quantity as a hexadecimal expression.

3.
```
SDA> EVALUATE ((TEN*6)+(-1/4))+(2+4)
Hex = 00000042      Decimal = 66
```

The EVALUATE command evaluates a complex expression and prints the result as hexadecimal and decimal values. See Sections 4.2.2 through 4.2.5 for a discussion of the expressions used in this example.

EXAMINE

The EXAMINE command displays the contents of a location or range of locations in physical memory.

You can use location parameters to examine specific locations or you can use qualifiers to display entire process and system regions. There are two ways to examine a range of locations: 1) designate starting and ending locations separated by a colon, for example, 80000040:80000200; or 2) specify a location and a byte length, separated by a semicolon, for example, 80000400;16.

If at any time you omit the location parameter from the EXAMINE command, SDA takes the location you last examined, increases it by 4 (one longword) and examines the resulting location.

Examining Specific Locations

A location can be represented by any valid SDA expression. When you use the EXAMINE command to look at a location, SDA displays the location, its symbolic representation (if possible), and its contents, in hexadecimal and ASCII formats.

SDA initially sets the current location to -4 (decimal) in the program region (P0) of the process. To examine memory locations in other processes, you must use the SET PROCESS command.

Examining Memory Regions

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

SDA formats the dump into columns of longwords, 4 for an 80-column device and 8 for a 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. You 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 that were not translated:

```
Virtual locations locl : loc2 are not in physical memory
```

In this message, locl and loc2 represent the starting and ending addresses of the range. This message also appears if you try to examine a single location that has not been mapped into physical memory.

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

```
Zeros suppressed from locl to loc2
```

Format

```
EXAMINE {[location]   [:location]}
        {             [;length]  }
```

Qualifiers	Defaults
/P0	None
/P1	
/SYSTEM	
/ALL	

Parameters

location

Expression that specifies the address in virtual memory at which data is stored.

length

Expression that specifies the number of bytes you want to display.

Qualifiers

/P0

Prints the entire program region for a given process. The default for this qualifier is the P0 region of the current process; you must use the SET PROCESS command to examine other processes' P0 regions.

/P1

Prints the entire control region for a given process. The default for this qualifier is the P1 region of the current process; use the SET PROCESS command to examine different P1 regions.

/SYSTEM

Prints portions of the writeable system region.

/ALL

Prints both the entire program and control regions for a given process, and portions of the writeable system region.

Examples

1.
```
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 hexadecimal and in ASCII formats. SDA represents nonprinting characters by a period (.) and puts quotation marks (" ") around ASCII text.

2.
```
SDA> EXAMINE PC
PC :  80008E22   "".,.."
SDA> EXAMINE @PC
EXE$RUNDWN+038 :  61772065   "e wa"
```

   SDA examines the program counter and the address contained in the program counter.

3.
```
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 hexadecimal) was given.

4.
```
SDA> EXAMINE/SYSTEM
```

   Figure 5-1 shows a portion of the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                                              21-MAY-1979 14:37:16.88
System region memory

00000000 00000003 00000009 00000003 00000010 0000000A 0000432B 00027393  .s..+C..........................  80000800
8005E800 80000848 00000000 7FFEEAF4 00000008 00000011 00000002 0000001A  ........................H.......  80000820
41424403 02313146 80060408 00060034 8000087C 80000948 80001046 80069090  ....F...H...|...4.......F11..DBA  80000840
800039A8 00000000 00000052 45564952 44424408 00000000 00000000 00000000  ..............DBDRIVER........9..  80000860
8005EB20 00010001 00000000 80017800 80017C00 8006086B 081000CC 800039A8  .9......k....|...x.......... ...  80000880
800008EC 032F1316 02000501 1C4D4008 800629A0 80068EB0 00000000 80000848  H.............)...@M......./.....  800008A0
00026617 000499F9 00001810 00000000 0315002A 80078230 00000000 800008BC  .........0...*...............f..  800008C0
800605C3 000C0000 8000388C 0C0C0000 003E0808 06000000 8011402C 000000CD  ....,@.........>......!..........  800008E0
00000000 00000000 00002000 00000100 00000000 00EE0F0A 10000000 0005328E  .2.................. ..........  80000900
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................  80000920
41504F03 00000000 80002964 00060034 8000097C 80000A88 00000000 00000000  .............|...4...d).......OPA  80000940
800039A8 00000000 00000052 4F544152 45504F08 00000000 00000000 00000000  ..............OPERATOR........9..  80000960
80000A28 00010001 00000000 00000019 0000000D 8000242D 081000AC 800039A8  .9......-$....................(...  80000980
8000095C 080002A0 00842042 0C040007 00000000 00000000 00000000 80000948  H..................B ..........  800009A0
00000028 00049477 00000010 00000000 00140000 8007A030 00000000 8000098C  .........0...............w...(...  800009C0
00000000 00000000 80002A09 00200000 00000000 00250001 8006B6C0 00000000  ...........%........ ..*.........  800009E0
00000006 00000006 0000006C 00000000 0000FFFF 00027BBA 80076700 8006F160  `....g...{...........]...........  80000A00
EF163FBB 00000000 00000001 00050044 00000000 00000000 080002A0 A0008420   ................D............?..  80000A20
800016EE 8000171F 800016D6 00000000 00000000 800016C0 80000A6C 00000CAB  ....1...........................  80000A40
8000097C 00000000 0209001C 00000000 00000000 80000A6C 00000CE3 EF163FBB  .?......1.......................|...  80000A60
41584403 03313146 80002A40 00060034 80000ABC 80000BE4 80000ABC 80000ABC  .................4...@*..F11..DXA  80000A80
800039A8 00000000 00000052 45564952 44584408 00000000 00000000 00000000  ..............DXDRIVER........9..  80000AA0
80000A28 00080020 FFF00000 80002C6D 00000000 80002AA7 08100128 800039A8  .9..(....*.......m,...... ...(...  80000AC0
80000AFC 004D011A 02000001 1C0D4008 80062CA0 00000000 00000000 80000A88  ..............,....@.........M.....  80000AE0
00000008 00000022 00041810 00000000 03140000 80078820 00000001 80000AFC  ........ ..................".......  80000B00
00000000 00000000 00000000 00000000 00000808 020000AC 800AD418 000000CC  ................................  80000B20
000001E5 80002B64 80002B64 00000000 00000000 0001000F 00000000 000001EE  ....................d...d.......  80000B40
E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 00800000  ................................  80000B60
E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5  ................................  80000B80
E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5  ................................  80000BA0
E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5  ................................  80000BC0
00000000 41424D03 00000000 8000127C 00060034 80000C90 80000D78 E5E5E5E5  ....x.......4...|.........MBA....  80000BE0
80000C1C 00000136 00000000 00000052 45564952 44424D08 00000000 00000000  ..........MBDRIVER.(.....6.......  80000C00
80000EFC 00010001 0000FFFF 00000014 00000000 00000000 08100074 80000C1C  ....t...........................  80000C20
80000C5C 00000000 010000A0 08150001 00000000 80000C90 00000000 80000BE4  ............................\...  80000C40
00000000 00000000 00010010 00000000 00080001 00000000 00000000 80000C5C  \...............................  80000C60
00000000 08100074 80000C90 80000C90 00000000 00000000 00000000 00000000  ........................t.......  80000C80
80000D04 00000000 80000BE4 80000EFC 00010004 FF0FFFFF 00000014 00000000  ................................  80000CA0
80078D20 00000001 80000CD0 80000CD0 00000000 010000A0 0C150001 00000000  ............................ ...  80000CC0
00B00000 00000000 00000000 000006FA 00000000 00010110 00000000 00080006  ................................  80000CE0
FF0FFFFF 00000014 00000000 00000000 08100074 80000D04 80000D04 00000000  ............t...................  80000D00
010000A0 0C150001 00000000 80062F50 00000000 80000BE4 80000EFC 00010004  .................P+..............  80000D20
00010110 00000000 00080002 80077420 00000002 80000D44 80000D44 00000000  ....D...D....... t..............  80000D40
80000DAC 80000E20 00000000 00C00000 00000000 00000000 00000434 00000000  ....4...................... .......  80000D60
444C4E08 00000000 00000000 00000000 414C4E03 00000000 80001608 00060034  4.............NLA..............NLD  80000D80
00000000 00000000 08100074 00000000 00000000 00000000 00000052 45564952  RIVER.................t..........  80000DA0
00000000 00000000 00000000 80000D78 80000EFC 00010001 00000000 00000000  ..................x.............  80000DC0
00080001 00000000 00000000 80000DEC 80000DEC 00000000 008400A0 0C150001  ................................  80000DE0
00000000 00000000 00000000 00000000 00000004 00000000 00000010 00000000  ................................  80000E00
00000000 00000000 54454E03 0054454E 80001648 00060034 80000E58 80063260  `2..X...4...H...NET..NET........  80000E20
00000000 00000000 000000C8 00000000 00005245 56495244 54454E09 00000000  .....NETDRIVER..................  80000E40
80000E20 80000EFC 00010001 00000000 00000000 00000000 00000000 08100080  ............................ ...  80000E60
80000E98 80000E98 00000000 00000000 0C1C2000 8006AC30 00000000 00000000  .........0.... .................  80000E80
00000000 00000000 00000000 00000010 00000000 00000001 00000000 00000000  ................................  80000EA0
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................  80000EC0
```

Figure 5-1 System Region Memory

EXIT

The EXIT command performs two functions: it discontinues SDA displays and exits from the utility. During interactive sessions, if a display has more than one page and is being shown on a video display terminal such as a VT100, SDA will issue the following message each time it reaches the bottom of a page:

```
        Press RETURN for more.
SDA>
```

If you want to discontinue the current display, type EXIT at the prompt. (On hard copy terminals, SDA does not prompt at the bottom of each page.) If you do not type EXIT at the screen overflow prompt and simply execute another command, SDA will accept the command as if you had exited from the display.

To stop SDA, type EXIT in response to the SDA prompt.

Format

EXIT

Qualifiers	Defaults
None	None

Parameters

None

FORMAT

The FORMAT command displays a formatted list of the contents of a specific block. It attempts to:

- Characterize a range of locations as a block
- Assign a symbol to each item of data within the block

Most VAX/VMS blocks contain a byte that indicates the block type. This byte is stored at offset 10 (decimal) from the first address of the block. The FORMAT command examines the byte stored at this offset as a block type. If the byte represents a valid block type, SDA tries to find its corresponding symbols. If the byte does not represent a valid block type, SDA issues the message:

```
invalid block type in specified block
```

Not every block contains a block type byte at offset 10. If this byte is absent, you must designate a block type at command level by using the qualifier /TYPE in order to format the block.

The display produced by FORMAT shows, from left to right, the virtual address of each item within the block, its symbolic name, and its hexadecimal representation.

Format

FORMAT location

Qualifiers	Defaults
/TYPE=	None

Parameters

location

The starting location of the block you want to format. The location can be any valid SDA expression.

Qualifiers

/TYPE=block-type

The symbolic prefix that corresponds to the type of block structure you want to format. SDA finds all symbols containing the specified prefix in the form:

```
block-type$field-type_field-name
```

The field types accepted by SDA are:

L	longword
W	word
B	byte
Q	quadword
T	counted ASCII string (0 through 31 characters)
C	constant

You can define your own block types and use the READ command to include them in the SDA symbol table. Thus, a valid block type is one that SDA can find in the symbol table. If SDA cannot find the symbols associated with the block type you have indicated, it will issue the message:

```
No "block-type" symbols found to format this block
```

Examples

1.
```
SDA> FORMAT @SCH$GL_CURPCB

80069550          PCB$L_SQFL     80002F48
80069554          PCB$L_SQBL     80002F48
80069558          PCB$W_SIZE         007C
8006955A          PCB$B_TYPE       0C
                      .
                      .
                      .
```

SDA takes the address pointed to by the global symbol, obtains the block type, and formats the block.

2.
```
$ RUN SYS$SYSTEM:SDA
            .
            .
            .

SDA> READ GLOBALS.STB

SDA> FORMAT @IOC$GL_DEVLIST

80000848      DDB$L_LINK          80000948
8000084C      DDB$L_UCB           8000087C
80000850      DDB$W_SIZE              0034
80000852      DDB$B_TYPE            06
80000853                          00
80000854      DDB$L_DDT           80060408
80000858      DDB$L_ACPD          02313146
8000085C      DDB$T_NAME          "DBA"
80000860                          00000000
80000864                          00000000
80000868                          00000000
8000086C      DDB$T_DRVNAME       "DBDRIVER"
SDA> FORMAT @.
          .
          .
          .

SDA> REPEAT
```

This example illustrates the use of SDA commands to format a list of blocks. The steps followed in the example are listed below:

- Invoke SDA.
- Use the READ command to read the DDB symbol definitions from GLOBALS.STB into the SDA symbol table. For a further discussion of object module files, see the description of the READ command.
- Use the FORMAT command to format the location pointed to by the global symbol IOC$GL_DEVLIST. When SDA finishes formatting this block, it sets the current location to the first byte of the block.

- Use the FORMAT command again to format the next block in the device list. Most blocks contain a pointer to the next block in a linked list. This pointer is usually the first longword in the block. In this step, the FORMAT command causes SDA to format the contents of the current location (the first longword of the block).

- Repeat the FORMAT command to format the next entry in the list. In this way, you can step through the entire device list, formatting each block.

3.
```
SDA> READ SYMDEF
SDA> FORMAT G10AC00/TYPE=PHD

8010AC00      PHD$Q_PRIVMSK       00308D08
8010AC04                          00000000
8010AC08      PHD$W_WSLIST            0046
8010AC0A      PHD$W_WSAUTH        042D
8010AC0C      PHD$W_WSLOCK            0051
8010AC0E      PHD$W_WSDYN         0051
8010AC10      PHD$W_WSNEXT            00C9
8010AC12      PHD$W_WSLAST        00DB
8010AC14      PHD$L_REFERFLT      00000000
8010AC18      PHD$W_WSQUOTA           042D
8010AC1A      PHD$W_DFWSCNT       00DB
8010AC1C      PHD$L_PAGFIL        03000000
              PHD$B_PAGFIL
8010AC20      PHD$W_PST               0600
              PHD$L_PSTBASOFF
8010AC22                          0000
8010AC24      PHD$W_PSTLAST           FFD8
8010AC26      PHD$W_PSTFREE       0000
8010AC28      PHD$L_FREPOVA       000AA200
8010AC2C      PHD$L_FREPTECNT     00001978
8010AC30      PHD$L_FREP1VA       7FFD9000
8010AC34      PHD$L_PGFLCNT       00270B2D
8010AC38      PHD$B_DFFFC               7F
8010AC39      PHD$B_PGTBPFC           02
8010AC31      PHD$W_FLAGS         0006
8010AC3C      PHD$L_CPUTIM        00022471
8010AC40      PHD$W_QUANT             FFF7
8010AC42      PHD$W_PRCLM         0008
8010AC44      PHD$W_ASTLM             0010
8010AC46      PHD$W_PHVINDEX      000F
8010AC48      PHD$W_BAK               0526
8010AC4A      PHD$W_WSLX          0500
              PHD$W_PSTBASMAX
8010AC4C      PHD$L_PAGEFLTS      0006451C
8010AC50      PHD$L_DIOCNT        00005631
8010AC54      PHD$L_BIOCNT        00009C28
8010AC58      PHD$L_CPULIM        00000000
                    .
                    .
                    .
```

This example shows the use of the qualifier /TYPE=. The READ command is issued to move PHD symbols to SDA's symbol table (see the description of the READ command for details on command syntax). Then, the FORMAT command can identify the process header block that starts at location 8010AC00.

HELP

The HELP command lists information about the SDA utility, its operation, and its command format. HELP has three command parameters. If you do not specify a parameter, HELP gives a brief description of SDA operations and lists SDA commands.

Format

```
HELP {[command-name]}
     {[EXPRESSION] }
     {[OPERATION]  }
```

Qualifiers	Defaults
None	None

Parameters

command-name

Specifies the SDA command for which you need information.

EXPRESSION

Prints a description of SDA expressions.

OPERATION

Describes how to operate SDA at your terminal and through the site-specific start-up procedure.

READ

The READ command lets you extract global symbols from any object module file and insert the definitions automatically into SDA's symbol table.

The object module file can be the output of a compiler or assembler or the output of the linker qualifier /SYMBOL_TABLE.

It is important to note that the READ command recognizes global symbols but ignores local symbols; hence, only global symbols are copied into the SDA symbol table.

The program below shows some sample definitions of global symbols.

```
        .TITLE  GLOBALS, GLOBAL SYMBOLS FOR SYSTEM DUMP ANALYZER

;
;  Note: the macros in this program must use the
;  argument GLOBAL.  This argument defines them as
;  globals so that they will be automatically carried
;  into the object file.  Without the GLOBAL argument,
;  the macros would be local and SDA would not be
;  able to read them.
;
        $PHDDEF GLOBAL          ; PROCESS HEADER DEFINITIONS
        $DDBDEF GLOBAL          ; DEVICE DATA BLOCK
        $UCBDEF GLOBAL          ; UNIT CONTROL BLOCK
        $VCBDEF GLOBAL          ; VOLUME CONTROL BLOCK
        $ACBDEF GLOBAL          ; AST CONTROL BLOCK
        $IRPDEF GLOBAL          ; I/O REQUEST PACKET

;   (more macros can be inserted here)

        .END
```

Use the following DCL command to generate an object module file with the file type STB that contains the global symbols defined in the sample program GLOBALS.MAR, as shown above:

```
$ MACRO GLOBALS+SYS$LIBRARY:LIB/LIBRARY /OBJECT=GLOBALS.STB
```

Now you can invoke SDA and use the READ command to copy the symbols into the SDA symbol table, as shown in Example 1 below.

Format

READ file-spec

Qualifiers	Defaults
None	None

Parameters

file-spec

The device, directory, and file name of the file whose symbols you want copied to SDA. The default file specification for this parameter is SYS$DISK:[default-dir]filename.STB.

Examples

1. SDA> READ GLOBALS

   SDA searches for the file specification GLOBALS.STB in the current device and directory.

REPEAT

The REPEAT command repeats execution of the last command issued. This command is primarily used to step through a linked list of data structures or to examine a sequence of memory locations. On terminal devices, you can use the escape key (ESC) to perform the same function as the REPEAT command; ESC provides a faster means of executing the command.

Format

REPEAT

Qualifiers	Defaults
None	None

Parameters

None

Examples

1.
```
SDA> FORMAT @SCH$GQ_LEFWQ

800631E0          PCB$L_SQFL          80062D80
800631E4          PCB$L_SQBL          800030D0
800631E8          PCB$W_SIZE              007C
800631EA          PCB$B_TYPE            0C
800631EB          PCB$B_PRI           18
                      .
                      .
                      .

SDA> FORMAT @.

80062D80          PCB$L_SQFL          80062FB0
80062D84          PCB$L_SQBL          800631E0
80062D88          PCB$W_SIZE              007C
80062D8A          PCB$B_TYPE            0C
80062D8B          PCB$B_PRI           16
                      .
                      .
                      .

SDA> REPEAT

80062FB0          PCB$L_SQFL          800030D0
80062FB4          PCB$L_SQBL          80062D80
80062FB8          PCB$W_SIZE              007C
80062FBA          PCB$B_TYPE            0C
80062FBB          PCB$B_PRI           15
                      .
                      .
                      .
```

In this example, the FORMAT command is used to examine the local event flag wait queue. The first process control block (PCB) in the wait queue is formatted, then the rest of the queue can be examined by using REPEAT (or by pressing ESC).

SET OUTPUT

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

When you set SDA output to a file or device, SDA stops displaying commands at your terminal. If you finish directing SDA commands to an output file and wish to return to interactive display, you can issue another SET OUTPUT command using your terminal device as the file specification. You can also exit from SDA and then recall the utility.

Format

SET OUTPUT file-spec

Qualifiers	Defaults
None	None

Parameters

file-spec

The device, directory, and file to which SDA output will be written. The default file specification is SYS$DISK:[default-dir]SYSDUMP.LIS.

Examples

1.
```
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 on the current device and directory in a file called BROKEN.LIS.

SET PROCESS

The SET PROCESS command moves process context to a specific process. This command allows you to examine the data structures associated with any given process.

When you invoke SDA and specify a dump file, process context, that is, the virtual memory you will see upon executing SDA commands, defaults to the process that was executing when the system failed. If you are examining the running system, process context defaults to your process.

When you issue a SET PROCESS command, process context changes to the process you specify. Many of the SDA commands, for example, EXAMINE, SHOW PROCESS and SHOW STACK, operate on the current process, that is, the context of the process specified in the last SET PROCESS command.

SET PROCESS locates the information needed for the particular process but produces no output.

You must specify one of the three SET PROCESS parameters or SDA will generate a syntax error.

Format

```
SET PROCESS {name      }
            {/INDEX=nn }
            {/SYSTEM   }
```

Qualifiers	Defaults
None	None

Parameters

name

A 1 to 31 character alphanumeric string assigned to the process. The dollar sign ($) and underline (_) characters can be included in the string.

/INDEX=nn

The index to the software process control block (PCB). The index number (nn) is composed of the last four hexadecimal digits of the process identification (PID).

/SYSTEM

The system process control block. The system PCB and process header (PHD) are dummy structures that are located in system space and contain the system working set, global section table, global page table, and other system-wide data.

Examples

1.
```
SDA> SET PROCESS/INDEX=43
SDA> EXAMINE/P0
```

   SDA locates the process by means of the index number and displays the contents of its program region.

2.
```
SDA> SET PROCESS SMITH
SDA> SHOW STACK
```

   Setting the process to SMITH causes the SHOW STACK command to default to SMITH rather than to the currently executing process.

SHOW CRASH

The SHOW CRASH command displays fundamental information concerning the operating system and the currently executing process. The display can be divided into three sections:

- Operating system and process information
- General and special register contents
- Processor and hardware maintenance register contents

Operating System and Process Information

The first section of SHOW CRASH lists:

- Date and time of the crash
- Name and version number of the operating system
- Name of the currently executing process
- File specification of the image executing in the process context (left blank if no image is executing)
- Interrupt Priority Level (in decimal) of the processor

General and Special Register Contents

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

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

Process and Hardware Maintenance Register Contents

The third section 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 currently executing process, as well as registers that contain information used by the operating system. The second set of registers are the stack pointers for the processor access modes plus the interrupt stack. The third set of registers are used in hardware maintenance.

The process and system registers are:

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

The stack pointers are:

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

The hardware maintenance registers are:

ICCS	Interval Clock Control/Status Register
ICR	Interval Count Register
TODR	Time-of-Day Register
ACCS	Accelerator Control and Status Register
SBIFS	SBI Fault/Status Register
SBISC	SBI Silo Comparator Register
SBIMT	SBI Maintenance Register
SBIER	SBI Error Register
SBITA	SBI Timeout Address Register
SBIS	SBI Silo Register

Format

SHOW CRASH

Qualifiers	Defaults
None	None

Parameters

None

Examples

1. SDA> SHOW CRASH

   Figure 5-2 shows the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                          21-MAY-1979 14:42:49.46
System crash information


Time of system crash: 21-MAY-1979 10:57:48.99

Version of system: VAX/VMS VERSION 1.50

Reason for BUGCHECK exception: PGFIPLHI, Pagefault with IPL too high

Process currently executing: GALCHER

Current image file name: DB2:[F4V2.TOOL]BLISS32.EXE;43

Current IPL: 7  (decimal)



General registers:

        R0  = 0019CCAB   R1  = 00000000   R2  = 00008600   R3  = 8010C50C
        R4  = 8006BAC0   R5  = 7FFDFE00   R6  = 001FFFBC   R7  = 00000200
        R8  = 7FFDB998   R9  = 0001910B   R10 = 0000255C   R11 = 7FFE6C10
        AP  = 7FFEEBC4   FP  = 7FFEEB7C   SP  = 7FFEEB38   PC  = 80006074
        PSL = 00070000



Processor registers:

        P0BR   = 8010C400     PCBB   = 0012C678     ACCS   = 00008001
        P0LR   = 000003EE     SCBB   = 001D6A00     SBIFS  = 00040000
        P1BR   = 7F914400     ASTLVL = 00000004     SBISC  = 00000000
        P1LR   = 001FFEC9     SISR   = 00180000     SBIMT  = 00200200
        SBR    = 001FC000     ICCS   = 800000C1     SBIER  = 00008002
        SLR    = 00001000     ICR    = FFFFF88F     SBITA  = 00075CD8
                              TODR   = 585540BA     SBIS   = 00000000

        ISP    = 8007C200
        KSP    = 7FFEEB38
        ESP    = 7FFEF000
        SSP    = 7FFEF828
        USP    = 7FFDB65C
```

Figure 5-2 System Crash Information

SHOW DEVICE

The SHOW DEVICE command displays a formatted list of all data structures associated with a device. The display for each device is divided into three sections:

- Device data block lists
- Controller data structures
- Device unit data structures

For a detailed explanation of I/O data structures displayed by SDA, consult Appendix A of the VAX/VMS Guide to Writing a Device Driver.

Device Data Block (DDB) List

The DDB list shows information common to all devices associated with a single controller. It shows:

- Address of the controller status register (CSR)
- Name of the controller
- Name of the ancillary control process (ACP)
- Name of the I/O driver
- Address of the driver prologue table (DPT)
- Length of the I/O driver and DPT

Controller Data Structures

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

- Device Data Block (DDB) -- points to the driver dispatch table, the channel request block, and the first unit control block connected to the controller
- Channel Request Block (CRB) -- stores information used to arbitrate requests between devices attached to a single controller
- Interrupt Dispatch Block (IDB) -- contains controller status information used to dispatch interrupts to the proper driver
- Driver Dispatch Table (DDT) -- points to routines that 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 (UCB) for each device. If the device is handling file-structured requests, the display lists the Volume Control Block (VCB) and the ACP queue as well.

Unit Control Block (UCB) - 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:

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

The next six items on the list concern the fork block for the device driver:

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

The UCB contains device status information:

Device class
Device type
Device buffer size (DEVBUFSIZ)
Device dependent data (DEVDEPEND) longword
Device status (DEVSTS) longword
Device IPL
Reference count
Operations count

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

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

SDA also formats all the I/O request packets queued to the UCB. The packet currently being processed is flagged by an asterisk (*). Information contained in each I/O request packet is listed in the following order across the page:

Channel number (CHAN)
Function value (FUNC)
Window Control Block (WCB)
Event flag number (EFN)
Asynchronous system trap (AST)
I/O status block (IOSB)
Status flags (STATUS)

If the request queue is empty, SDA issues the message:

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

Volume Control Block and ACP Queue - If a volume was mounted on the device SDA reads and displays the contents of the volume control block (VCB) and the ancillary control process queue block (AQB). The VCB identifies the volume and contains counts and quotas concerning files on that volume.

The ACP queue block contains information about the ancillary control process (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 ***
```

Format

```
SHOW DEVICE [device-name]
```

Qualifiers	Defaults
None	None

Parameters

device-name

The name of a device whose data structures you want to display. The device name takes the form:

```
devcu
```

where

```
dev = 2-alphabetic character device code
c   = 1-alphabetic character controller designator
u   = 1- or 2-digit device unit number
```

You can display information about several devices by specifying a device code or a device code and controller. For example, SHOW DEVICE D lists all devices with device code Dn, where n corresponds to the second letter of the device code. SHOW DEVICE DBA lists all devices with device code DB and controller A. To display a single unit, specify the entire device name: SHOW DEVICE DBA1 displays the device associated with device name DBA1. If you do not specify a device name, SDA lists the data structures of every device on the system.

Examples

1. SDA> SHOW DEVICE D

   Figure 5-3 is a sample Device Data Block list of all the devices attached to the system whose device codes start with D. This is an example of the first section of the display produced by SHOW DEVICE.

```
VAX/VMS 2.0 -- System Dump Analysis                    18-DEC-1979 11:38:52.76
I/O data structures

                              DDB list
                              --------

  Address  Controller   ACP       Driver      DPT    DPT size
  -------  ----------   ---       ------      ---    --------

 80000848       DRA     F11ACP    DRDRIVER  80080410  0814
 8009A4C0       DMA     F11ACP    DMDRIVER  800821E0  08F0
 8009AD00       DYA     F11ACP    DYDRIVER  80082EC0  06F0
 8009C560       DBA     F11ACP    DBDRIVER  80087640  06F0
 8009C620       DBB     F11ACP    DBDRIVER  80087640  06F0
```

Figure 5-3 Device Data Block List for Dn Devices

2. SDA> SHOW DEVICE DBA

Figure 5-4 shows information on the data structures associated with DB device controller A. This is an example of the second section of the display produced by SHOW DEVICE.

```
VAX/VMS 2.0 -- System Dump Analysis                    16-AUG-1979 16:34:54.81
I/O data structures


Controller: DBA
---------------

Device Data Block (DDB):
        DDT address                 80060408
        First UCB address           8000087C
        CRB address                 8005EB20

Channel Request Block (CRB):
        UCB reference count               11
        Channel allocation mask           00
        Secondary CRB address       00000000
        IDB address                 80075E40
        Controller init. routine    80001271
        Unit init. routine          800609DE
        Unit start routine          00000000
        Unit disconnect routine     00000000

Interrupt Dispatch Block (IDB):
        CSR address                 80017800
        Owner UCB address           00000000
        Number of units                    8
        ADP address                 8005EB60

Driver Dispatch Table (DDT):
        Start I/O routine           00000102
        Unsol. interrupt routine    00000637
        Function Decision table     0000007A
        Cancel I/O routine          8000A869
        Register dump routine       00000592
        Diagnostic buffer size          0080
        Error buffer size               00AE
```

Figure 5-4 Controller Data Structures for DB Devices

3. SDA> SHOW DEVICE DBA1

Figure 5-5 shows an example of the last section of the display produced by SHOW DEVICE. The display lists the UCB, VCB, and AQB for the device DBA1.

```
VAX/VMS 1.0 -- System Dump Analysis                          15-AUG-1979 17:10:06.68
I/O data structures

DBA1
----

        UCB address:      80074B30
        Device status:      1810   online,valid,unload
        Characteristics:  1C4D4008

IRP address     80089350     Device class          01     SVPN          00000217
CRB address     8006B520     Device type           05     SVAPTE        8009835C
VCB address     80076140     DEVBUFSIZ            512     BOFF              0000
FQFL            80003A48     DEVDEPEND       032F1316     BCNT              0200
FQBL            80003A48     DEVSTS              0000     ERTCNT               8
Fork IPL               8     Device IPL            21     ERTMAX               8
Fork PC         80074831     Reference count        0     ERRCNT               1
Fork R3         80019C80     Operation count    83765     Owner UIC     [  1,  1]
Fork R4         80019800     AMB address     00000000     PID            00000000

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

Volume: VMSWORK2
Status:  80   system

AQB address         8006E4A0     Cluster size              3
Rel. Volume #              2     Reserved files            9
Transactions               3     Maximum files         25000
Mount count                1     Free blocks           19791
Window size                7     Record size               0
Default extension          5     RVT address        8008C820

ACP for volume: DRA5ACP

PID            00010042
ACP type              3
ACP class             0
Status               04   defsys
Mount count           7
AQB linkage    00000000

        *** ACP request queue is empty ***
```

Figure 5-5 Device Unit Data Structures for Device DBA1

SHOW PAGE_TABLE

The SHOW PAGE_TABLE command displays a formatted list of system page table entries which are used to map virtual pages to physical pages.

The display can be divided into left and right sections. The left section contains virtual page information. The right section contains physical page information.

Virtual Page Information

The left section of the display describes virtual pages using information contained in the system page table. Each line of this display lists characteristics of a particular virtual page as well as locations needed for address translation. The values listed are:

- ADDRESS -- system virtual address that marks the base of a virtual page
- SVAPTE -- system virtual address of the page table entry that maps this virtual page
- PTE -- page table entry; longword that describes a system virtual page
- Type -- type of virtual page; there are seven types:

VALID	Valid page (in main memory)
TRANS	Transitional page (between main memory and page lists)
DZERO	Demand-allocate-zero-fill page
PGFIL	Paging file page
STX	Section table index page
GPTX	Global page table index page
IOPAGE	I/O address space page

- PROT -- protection; a code derived from bits in the PTE that designate the type of access (read and/or write) granted to processor access modes (Kernel, Executive, Supervisor, or User).
- Bits -- letter(s) representing the value of a bit or a combination of bits in the PTE; indicates certain aspects of a page. The bit codes are:

M	Modify bit
L	Locked into working set
K,E,S or U	Access mode of owner of page (only one letter will appear)

Physical Page Information

If the virtual page has been mapped to a physical page, the right section of the display includes information from the Page Frame Number (PFN) data base. Otherwise, the section is left blank. SDA organizes the 18 bytes of PFN data into nine categories:

- PAGTYP -- type of physical page; there are six page types:

PROCESS	Process page
SYSTEM	System page
GLOBAL	Global section page
PPGTBL	Process page table page
GPGTBL	Global page table page
GBLWRT	Global writeable section page

- LOC -- location of page in system; there are eight locations:

ACTIVE	In working set
MDFYLST	In modified page list
FREELST	In free page list
BADLST	In bad page list
RELPEND	Release pending
RDERROR	Read error
PAGEOUT	Paging out
PAGEIN	Paging in

- STATE -- byte that describes the state of the physical page.

- TYPE -- byte that describes the type of virtual page (see PAGTYP).

- REFCOUNT -- reference count; word indicating the presence of a reference to this PFN. If the value of REFCOUNT is non-zero, the page is used in at least one working set. If the value is zero, the page is not used in any working set.

- BAK -- backing store address; location on a disk device to which pages can be written

- SVAPTE -- virtual address associated with this page frame. The two SVAPTEs indicate a valid link between physical and virtual address space.

- FLINK -- forward link within PFN data base; also acts as the share count of a global section.

- BLINK -- backward link within PFN data base; also acts as an index to the working set list.

SDA indicates pages that cannot be accessed with the message:

```
---------- n NULL PAGES
```

where n represents the number of inaccessible pages.

Format

SHOW PAGE_TABLE

Qualifiers	Defaults
/GLOBAL	
/SYSTEM	
/ALL	/ALL

Parameters

None

Qualifiers

/GLOBAL

Lists the global page table.

/SYSTEM

Lists the system page table.

/ALL

Lists both the global and system page tables. This is the default for the command.

Examples

1. SDA> SHOW PAGE_TABLE/SYSTEM

   Figure 5-6 shows one page of the display produced by this command.

ADDRESS	SVAPTE	PTE	TYPE	PROT	BITS	PAGTYP	LOC	STATE	TYPE	REFCNT	BAK	SVAPTE	FLINK	BLINK
8000D000	801F91A0	78000B7A	TRANS	UR	K	SYSTEM	FREELST	00	01	0	0040FFF8	801F91A0	01E5	0742
8000D200	801F91A4	F80004E9	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91A4	0000	0046
8000D400	801F91A8	78000134	TRANS	UR	K	SYSTEM	FREELST	00	01	0	0040FFF8	801F91A8	053E	0654
8000D600	801F91AC	F80009E5	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91AC	0000	0062
8000D800	801F91B0	F8000DB2	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91B0	0000	0058
8000DA00	801F91B4	F80001DD	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91B4	0000	0057
8000DC00	801F91B8	F80001E9	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91B8	0000	00A6
8000DE00	801F91BC	F8000257	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91BC	0000	007C
8000E000	801F91C0	F800098E	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91C0	0000	007D
8000E200	801F91C4	F8000A5B	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91C4	0000	00A3
8000E400	801F91C8	F800049A	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91C8	0000	004E
8000E600	801F91CC	F8000844	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91CC	0000	0094
8000E800	801F91D0	F8000075	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91D0	0000	0068
8000EA00	801F91D4	78000088	TRANS	UR	K	SYSTEM	FREELST	00	01	0	0040FFF8	801F91D4	0DB9	0394
8000EC00	801F91D8	F800020A	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91D8	0000	005C
8000EE00	801F91DC	F8000270	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91DC	0000	005B
8000F000	801F91E0	F8000A4C	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91E0	0000	006D
8000F200	801F91E4	78000096	TRANS	UR	K	SYSTEM	FREELST	00	01	0	0040FFF8	801F91E4	0909	05B5
8000F400	801F91E8	F800067B	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91E8	0000	0072
8000F600	801F91EC	780001E5	TRANS	UR	K	SYSTEM	FREELST	00	01	0	0040FFF8	801F91EC	0000	0B7A
8000F800	801F91F0	F800031A	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91F0	0000	006C
8000FA00	801F91F4	F800029A	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F91F4	0000	00A9
8000FC00	801F91F8	7800098C	TRANS	UR	K	SYSTEM	FREELST	00	01	0	0040FFF8	801F91F8	02BA	0159
8000FE00	801F91FC	7C40FFF8	STX	UR	K									
80010000	801F9200	F80002DC	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F9200	0000	0087
80010200	801F9204	78000159	TRANS	UR	K	SYSTEM	FREELST	00	01	0	0040FFF8	801F9204	098C	094D
80010400	801F9208	7C40FFF8	STX	UR	K									
80010600	801F920C	7C40FFF8	STX	UR	K									
80010800	801F9210	7C40FFF8	STX	UR	K									
80010A00	801F9214	7C40FFF8	STX	UR	K									
80010C00	801F9218	7C40FFF8	STX	UR	K									
80010E00	801F921C	F80009F7	VALID	UR	K	SYSTEM	ACTIVE	07	01	1	0040FFF8	801F921C	0000	0083
-------- 27 NULL PAGES														
80014600	801F928C	94100010	IOPAG	KW	M	K								
80014800	801F9290	94100030	IOPAG	KW	M	K								
80014A00	801F9294	90100031	IOPAG	KW		K								
80014C00	801F9298	90100032	IOPAG	KW		K								
80014E00	801F929C	90100033	IOPAG	KW		K								
80015000	801F92A0	94100034	IOPAG	KW	M	K								
80015200	801F92A4	94100035	IOPAG	KW	M	K								
80015400	801F92A8	90100036	IOPAG	KW		K								
80015600	801F92AC	90100037	IOPAG	KW		K								
80015800	801F92B0	941009F0	IOPAG	KW	M	K								
80015A00	801F92B4	901009F1	IOPAG	KW		K								
80015C00	801F92B8	901009F2	IOPAG	KW		K								
80015E00	801F92BC	901009F3	IOPAG	KW		K								
80016000	801F92C0	941009F4	IOPAG	KW	M	K								
80016200	801F92C4	901009F5	IOPAG	KW		K								
80016400	801F92C8	901009F6	IOPAG	KW		K								
80016600	801F92CC	901009F7	IOPAG	KW		K								

Figure 5-6 System Page Table

SHOW PFN_DATA

The SHOW PFN_DATA command displays a formatted list of values contained in the page lists and in the PFN data base that can be used to translate physical pages to virtual pages.

There are four data structures concerned with the management of physical memory:

- Free Page List -- pages available for use
- Modified Page List -- pages to be written to disk
- Bad Page List -- pages containing data errors
- PFN Data Base -- all pages in physical memory

To display a particular physical page, specify its page frame number (PFN). To list the pages of one or more data structures, use the qualifiers. If you do not specify a parameter or a qualifier, SDA will dump all the page lists and the entire PFN data base.

The format used to display physical page data is the same for each data structure. Figure 5-7 shows a page of the display produced by the command SHOW PFN_DATA/SYSTEM. SDA organizes the information for each page under the following headings:

- PFN -- page frame number; the absolute page number within physical memory
- PTE ADDRESS -- Page Table Entry address; the virtual address of the Page Table Entry (see the description of the SHOW PAGE_TABLE command for more details).
- BAK -- backing store address; location on a disk device to which pages can be written
- REFCNT -- reference count; a word whose value signals whether a page is part of a working set
- FLINK -- forward link; forward link within the PFN data base (also used as share count of a global section)
- BLINK -- backward link; backward link within the PFN data base (also used as an index to the working set list)
- TYPE -- type of page that was mapped into physical memory (see the description of the SHOW PAGE_TABLE command for a list of the different types)
- STATE -- current state of the page in the system (see the description of the SHOW PAGE_TABLE command for a list of states)

Format

SHOW PFN_DATA [number]

Qualifiers	Defaults
/FREE	
/MODIFIED	
/BAD	
/SYSTEM	
/ALL	/ALL

Parameters

number

The number of the physical page you want to display.

Qualifiers

/FREE

Displays the free page list.

/MODIFIED

Displays the modified page list

/BAD

Displays the bad page list.

/SYSTEM

Displays the PFN data base in order of PFN starting at page frame zero.

/ALL

Displays all of the above memory management data structures. This is the default for the command.

Examples

1. SDA> SHOW PFN_DATA/SYSTEM

   Figure 5-7 shows one page of the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                          21-MAY-1979 15:00:04.82
PFN data base

PFN    PTE ADDRESS   BAK       REFCNT  FLINK BLINK    TYPE          STATE
----   -----------   --------  ------  ----- -----    ----------    ----------

0000     00000000    00000000       0    0895 0AAF    00 PROCESS    00 FREELST
0001     800A3C10    03000000       1    0000 0053    00 PROCESS    87 ACTIVE
0002     80174EA0    03000000       1    0000 007C    00 PROCESS    87 ACTIVE
0003     8010CEE0    033FFFFF       1    0000 01A2    00 PROCESS    87 ACTIVE
0004     801F7F70    03000000       1    0000 00C9    00 PROCESS    07 ACTIVE
0005     8010CEB4    033FFFFF       1    0000 018D    00 PROCESS    87 ACTIVE
0006     80174CAC    03000000       1    0000 00CE    00 PROCESS    07 ACTIVE
0007     8016D200    0300E351       0    002D 0498    00 PROCESS    00 FREELST
0008     8010CE2C    0300E2DE       0    0D06 095F    00 PROCESS    81 MDFYLST
0009     801C0C58    03000000       1    0000 00DB    00 PROCESS    07 ACTIVE
000A     801FB890    03000000       1    0000 0054    04 PPGTBL     07 ACTIVE
000B     801FC6BC    03000000       1    0000 004B    04 PPGTBL     87 ACTIVE
000C     00000000    00000000       0    0566 08B1    00 PROCESS    00 FREELST
000D     801FD3E0    0040FFB8       1    0002 0000    02 GLOBAL     07 ACTIVE
000E     801FD474    0040FFA0       0    03FB 0D11    02 GLOBAL     00 FREELST
000F     8010CF1C    033FFFFF       1    0000 0084    00 PROCESS    87 ACTIVE
0010     80101040    033FFFFF       1    0000 008C    00 PROCESS    87 ACTIVE
0011     801FD524    0040FF90       1    0002 0000    02 GLOBAL     07 ACTIVE
0012     8017C9D0    03000000       1    0000 0047    00 PROCESS    07 ACTIVE
0013     80113F88    0300E2C6       1    0000 014B    00 PROCESS    07 ACTIVE
0014     8011F480    03000000       1    0000 00B8    00 PROCESS    07 ACTIVE
0015     801F0464    03000000       1    0000 006B    00 PROCESS    07 ACTIVE
0016     8010C610    0040FFE8       0    00D1 0195    00 PROCESS    00 FREELST
0017     8016D3BC    03000000       1    0000 00FE    00 PROCESS    07 ACTIVE
0018     800B50D8    03000000       1    0000 005D    00 PROCESS    07 ACTIVE
0019     8016D338    03000000       1    0000 0143    00 PROCESS    07 ACTIVE
001A     801FB774    03000000       1    0000 004E    04 PPGTBL     87 ACTIVE
001B     8016D254    033FFFFF       0    018E 0634    00 PROCESS    81 MDFYLST
001C     801D20D8    033FFFFF       1    0000 00B4    00 PROCESS    87 ACTIVE
001D     801FD5F0    0040FF90       1    0002 0000    02 GLOBAL     07 ACTIVE
001E     8010A8D8    03005305       1    0000 0175    00 PROCESS    87 ACTIVE
001F     801FD6DC    0040FF70       1    0001 0000    02 GLOBAL     07 ACTIVE
0020     8011D848    033FFFFF       1    0000 00D2    00 PROCESS    87 ACTIVE
0021     8010CFE0    033FFFFF       1    0000 00DC    00 PROCESS    87 ACTIVE
0022     801B59D4    03000000       1    0000 0046    00 PROCESS    07 ACTIVE
0023     00000000    00000000       0    020C 025B    00 PROCESS    00 FREELST
0024     80174CA4    03000000       1    0000 00CC    00 PROCESS    07 ACTIVE
0025     00000000    00000000       0    0A2E 08CB    00 PROCESS    00 FREELST
0026     8010CE34    0300E2E0       0    0590 0423    00 PROCESS    81 MDFYLST
0027     801FCB6C    03000000       1    0000 004F    04 PPGTBL     87 ACTIVE
0028     8010CA28    0040FFE8       1    0000 00C5    00 PROCESS    07 ACTIVE
0029     8010CC98    033FFFFF       0    0A25 0CC5    00 PROCESS    81 MDFYLST
002A     801601F4    03008FF2       1    0000 0058    00 PROCESS    87 ACTIVE
002B     8010C604    0040FFE8       0    07AC 0D1E    00 PROCESS    00 FREELST
002C     8016D348    03000000       1    0000 0135    00 PROCESS    07 ACTIVE
002D     8016D204    0300E352       0    04D1 0007    00 PROCESS    00 FREELST
002E     801A2838    0300861C       1    0000 00D7    00 PROCESS    87 ACTIVE
002F     00000000    00000000       0    0840 0B03    00 PROCESS    00 FREELST
0030     801FD2E8    0040FFD0       1    0003 0000    02 GLOBAL     07 ACTIVE
0031     00000000    00000000       0    093A 08CD    00 PROCESS    00 FREELST
0032     8010C7C4    0040FFE8       1    0000 017A    00 PROCESS    07 ACTIVE
```

Figure 5-7 PFN Data Base

SHOW POOL

The SHOW POOL command displays the contents of the I/O Request Packet (IRP) pool, the nonpaged dynamic storage pool, and the paged dynamic storage pool. This data is organized into blocks; SDA attempts to identify each block by its block type. SHOW POOL displays only allocated blocks, that is, blocks that are (or were) currently in use by the system.

The information contained in each of the three pools is shown in the same format. From left to right, the contents of the display are:

- Block type -- the type of information contained in the block. SDA tries to define the block type. If it is unable to do so, the message UNKNOWN is printed instead of the name of the block type.
- Starting address -- the virtual address that marks the start of the block.
- Block size -- the number (decimal) of bytes of nonpaged memory allocated to the block. The block size is fixed at 80 in the IRP pool and is variable in the paged and nonpaged pools.
- Contents (hexadecimal) -- the contents of the block in longwords. The contents are arranged four columns across.
- Contents (ASCII) -- the contents of the block in ASCII format.

Format

SHOW POOL

Qualifiers	Defaults
/IRP	
/NONPAGED	
/PAGED	
/SUMMARY	
/ALL	/ALL

Parameters

None

Qualifiers

/IRP

Prints the I/O request packet pool. Formats all blocks currently allocated (in use) within this pool.

/NONPAGED

Prints the nonpaged dynamic storage pool currently in use by the system.

/PAGED

Prints the paged dynamic storage pool currently in use by the system.

/SUMMARY

Prints a summary of the pools selected by the above qualifiers. /SUMMARY displays the different block types present and lists the total number and bytes used of each in decimal. This qualifier also prints the total number of bytes used in each pool.

/ALL

Prints IRP, nonpaged, and paged dynamic storage pools. This is the default for the command.

Examples

1. SDA> SHOW POOL/PAGED

Figure 5-8 shows a page of the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                          21-MAY-1979 16:21:47.68
Paged dynamic storage pool

GSD     80056FA0    48
                    00010004 00150028 80056FD0 80056B10 .k...o..(.......
                    5F424D53 5452500A 80056870 FFC8A000 ....ph...PRTSMB_
                    00000000 00000000 00000000 00313030 001.............
GSD     80056FD0    48
                    00010004 00150028 80057000 80056FA0 .o...p..(.......
                    59414C50 5349440B 80056970 FFC0A000 ....pi...DISPLAY
                    00000000 00000000 00000000 3130305F _001.............
GSD     80057000    48
                    00010004 00150028 80057320 80056FD0 .o.. s..(.......
                    59414C50 5349440B 80056970 FFB8A000 ....pi...DISPLAY
                    00000000 00000000 00000000 3230305F _002.............
KFH     80057030   240
                    0028007C 002600E4 80056970 80057114 .q..pi....&.|.(.
                    00000101 31303230 00000000 00440038 8.D.....0201....
                    02000000 00000000 00000000 00000000 ................
                    00000000 000022B5 0000373E 00000000 ....>7...".......
                    00000002 00000000 0000001D 00000000 ................
                    00000000 00000000 59414C50 53494407 .DISPLAY........
                    00000000 00000000 00000000 00313002 .01.............
                    00003032 2E313005 008644D4 064A0E20  .J..D...01.20..
                    00000001 00040010 00000000 00000000 ................
                    00000005 00020010 00000002 00000000 ................
                    00800007 0005000C 00000006 0000000A ................
                    00000000 0000000C 000A0010 0000000C ................
                    0000000A 00000016 000B0010 00000008 ................
                    FD00000C 003FFFEC 0014000C 00000012 .........?.....
                    00000000 00000000 00000000 FFFF0000 ................
KFI     80057120    64
                    00340000 04180040 80057160 800569F0 .i..`q..@.....4.
                    800630C0 00000233 00020387 00000001 ........3....0..
                    00000001 00000000 00108001 800574FC .t..............
                    00000000 00000057 4F485304 80057120  q...SHOW.......
KFI     80057160    64
                    00340000 04180040 800571A0 80057120  q...q..@.....4.
                    800630F0 000004DD 00008001 00000001 ..............0..
                    00000000 00000000 00000000 00000000 ................
                    00000000 00000000 50495003 00000000 .....PIP........
KFI     800571A0    64
                    00340000 04180040 800571E0 80057160 `q...q..@.....4.
                    80063120 000000A0 00018205 00000001 ............ 1..
                    00000001 00000000 00000000 00000000 ................
                    00000000 00000000 534F5303 800571A0 .q...SOS........
KFI     800571E0    64
                    00340000 04180040 80057220 800571A0 .q.. r..@.....4.
                    80062F50 000002F5 00010307 00000001 .............P/..
                    00000001 00000000 00000000 800578DC .x..............
                    00000000 00000000 58535203 800571E0 .q...RSX........
KFI     80057220    64
                    00340000 04180040 80057260 800571E0 .q..`r..@.....4.
                    80062F80 000004E1 00010307 00000001 ............../..
                    00000001 00000000 00000000 80057A0C .z..............
                    0000534E 4152544B 43414209 80057220  r...BACKTRANS..
KFI     80057260    64
                    00340000 04180040 800572A0 80057220  r...r..@.....4.
```

Figure 5-8 Paged Dynamic Storage Pool

SHOW PROCESS

The SHOW PROCESS command displays the software and hardware context of any process in the balance set.

Format

```
                  ⎧[name]      ⎫
SHOW PROCESS      ⎨[/INDEX=nn] ⎬
                  ⎩[/SYSTEM]   ⎭
```

Qualifiers	Defaults
/PCB	/PCB
/PHD	
/REGISTERS	
/WORKING_SET	
/PROCESS_SECTION_TABLE	
/PAGE_TABLES	
/ALL	

Parameters

name

A 1- to 15-character alphanumeric string assigned to the process. The name can include the symbols underline (_) and dollar sign ($).

/INDEX=nn

The index to the software PCB; nn consists of the last four hexadecimal digits of the Process Identification (PID).

/SYSTEM

The system process control block. The system PCB and process header (PHD) are dummy structures that are located in system space and contain the system working set, global section table, global page table, and other system-wide data. When you specify this parameter, SDA displays the system PCB rather than a given process.

If no parameter is specified, the command displays the current process. See the description of the SET PROCESS command for the definition of the current process.

Qualifiers

/PCB

Produces a formatted list of the data contained in the software process control block (PCB). The software PCB is the central control mechanism for process swapping and scheduling.

The display produced by the /PCB qualifier lists:

- Software context for the process
- Condition handling information
- Interprocess communication data
- Counts and quotas

/PCB is the default display for the command.

/PHD

Lists information included in the process header. The process header contains a process's vital statistics and is swapped into memory when a process becomes part of the balance set. Each item listed by the /PHD qualifier gives a quantity, count, or limit for the process concerning:

- Process memory
- Pager
- Scheduler
- Asynchronous system traps
- I/O activity
- CPU activity

/REGISTERS

Lists the process's hardware context. When a process executes, its hardware context is saved in the processor registers (see the description of the SHOW CRASH command). If the process is not executing, its hardware context is stored in the hardware PCB, which is part of the process header. The /REGISTERS qualifier organizes the saved process registers into:

- General-purpose registers
- Stack pointers
- Special-purpose registers
- Base and length registers

/WORKING_SET

Displays the working set list for the process. The working set list is a table for all virtual pages residing in physical memory that the process can access without a page fault. The values exhibited by this command are:

- INDEX -- index used in PFN data base to access the entry
- ADDRESS -- address of a virtual page in the process address space
- STATUS -- a 3-part section that lists the location of the page in physical memory, the type of page (see the description of the SHOW PAGE_TABLE command), and whether the page is locked into the working set

When SDA locates an unused working set entry, it issues the message:

```
---- n empty entries
```

The value of n is the number (in decimal) of unused entries that SDA has found.

/PROCESS_SECTION_TABLE

Lists data within the process section table. The process section table contains information needed to locate a page in a process section. SDA notes the boundary of the Process Section Table in the "Process Section Table Information" section of the listing and then displays the actual process section table in readable format. The parts of the process section table are:

- INDEX -- the word that locates the corresponding process section table entry
- ADDRESS -- the virtual address in the program region that marks the location of a process section table page
- PAGES -- the length of a process section in pages
- WINDOW -- the mapping window that translates virtual block numbers to logical block numbers
- VBN -- virtual block number; the number of a block on a mass storage device (the block number is relative to a file rather than to a device)
- CLUSTER -- the cluster factor used when faulting pages in the corresponding process section
- CHANNEL -- the channel number connecting a process section to a device unit
- REFCNT -- a number that indicates whether the page is part of a working set
- FLINK -- the forward link word in the PST list
- BLINK -- the backward link word in the PST list
- FLAGS -- the flags that describe the process section during process execution

/PAGE_TABLES

Displays the program and control region page tables. /PAGE_TABLES produces a list in the same format as the SHOW PAGE_TABLE command.

/ALL

Displays the information produced by the /PCB, /PHD, /REGISTERS, /WORKING_SET, /PROCESS_SECTION_TABLE, and /PAGE_TABLES qualifiers.

Examples

1. SDA> SHOW PROCESS/PCB

   The top portion of Figure 5-9 shows the display produced by this command.

2. SDA> SHOW PROCESS/PHD

   The middle portion of Figure 5-9 shows the display produced by this command.

3. SDA> SHOW PROCESS/REGISTERS

   The bottom portion of Figure 5-9 shows the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                          3-JAN-1980 16:54:45.57
Process 2B dump: ELDRIDGE

Process status:  00040001   RES,PHDRES

PCB address              8008BD20    JIB address              8009DB80
Master PID               0004002B    Creator PID              00000000
PID                      0004002B    Subprocess count                0
PHD address              80180A00    Swapfile disk address    00000000
State                       LEF      Termination mailbox          0000
Current priority              9      AST's enabled                KESU
Base priority                 4      AST's active                 NONE
UIC                      [011,013]   AST's remaining                14
Mutex count                   0      Buffered I/O count/limit       5/6
Waiting EF cluster            0      Direct I/O count/limit         6/6
Starting wait time       00000000    BUFIO byte count/limit     12056/12336
Event flag wait mask     F7FFFFFF    # open files allowed left      10
Local EF cluster 0       4000031B    Timer entries allowed left     10
Local EF cluster 1       80000041    Active page table count         0
Global cluster 2 pointer 00000000    Process WS page count         235
Global cluster 3 pointer 00000000    Global WS page count           14


Process header
--------------

First free P0 address      0002C800    Accumulated CPU time       00001446
Free PTEs between P0/P1        7478    CPU since last quantum         0031
First free P1 address      7FFD3200    Subprocess quota                  8
Free page file pages          25252    AST limit                        16
Page fault cluster size         127    Process header index           0018
Page table cluster size           2    Backup address vector          03A5
Flags                          0002    WSL index save area            0380
Direct I/O count                727    PTs having locked WSLs            2
Buffered I/O count             4958    PTs having valid WSLs             6
Limit on CPU time          00000000    Active page tables                7
Maximum page file count       25600    Maximum active PTs                7
Total page faults              7154    Guaranteed fluid WS pages        20
File limit                       16    Extra dynamic WS entries        360
Timer queue limit                10    Locked WSLE counts array       0FB8
Paging file index          03000000    Valid WSLE counts array        0FF8


Saved process registers
-----------------------

R0   = 08000000    R1   = 00000000    R2   = 80001AD0    R3   = 8008BD70
R4   = 8008BD20    R5   = 00000000    R6   = 000008DC    R7   = 00000003
R8   = 7FFD5A58    R9   = 7FFDC120    R10  = 7FFDC008    R11  = 7FFDBA8C
AP   = 7FFD590C    FP   = 7FFD5914    PC   = 80000328    PSL  = 03C00000
KSP  = 7FFEE400    ESP  = 7FFEF000    SSP  = 7FFEF878    USP  = 7FFD590C
P0BR = 80181C00    P0LR = 00000164    P1BR = 7F989C00    P1LR = 001FFE9A
```

Figure 5-9 Process Information

4. SDA> SHOW PROCESS/WORKING_SET

Figure 5-10 shows the display produced by this command. The size of the working set and its boundaries are listed at the head of the display. The actual working set list follows this information.

```
VAX/VMS 2.0 -- System Dump Analysis                          21-MAY-1979 15:19:08.57
Process 34 dump: GROVE

Working set information
-----------------------

First WSL entry          0046        Current authorized working set size   1000
First locked entry       0051        Default (initial) working set size    350
First dynamic entry      0052        Maximum working set allowed (quota)   1000
Last entry replaced      015A
Last entry in list       01A3

Working set list
----------------

        INDEX   ADDRESS    STATUS

         0046  7FFEEA00     VALID PROCESS WSLOCK
         0047  7FFEE800     VALID PROCESS WSLOCK
         0048  7FFEE600     VALID PROCESS WSLOCK
         0049  7FFEFE00     VALID PROCESS WSLOCK
         004A  8010AC00     VALID PPGTBL  WSLOCK
         004B  8010AE00     VALID PPGTBL  WSLOCK
         004C  8010B000     VALID PPGTBL  WSLOCK
         004D  8010C000     VALID PPGTBL  WSLOCK
         004E  8010C200     VALID PPGTBL  WSLOCK
         004F  80113E00     VALID PPGTBL  WSLOCK
         0050  80114000     VALID PPGTBL  WSLOCK
         0051  8010B200     VALID PPGTBL  WSLOCK
         0052  00054400     VALID PROCESS
         0053  00054A00     VALID PROCESS
         0054  00055400     VALID PROCESS
         0055  00054E00     VALID PROCESS
         0056  00054C00     VALID PROCESS
         0057  00055000     VALID PROCESS
         0058  00055200     VALID PROCESS
         0059  00063E00     VALID PROCESS
         005A  00056600     VALID PROCESS
         005B  00055600     VALID PROCESS
         005C  00068E00     VALID PROCESS
         005D  00055C00     VALID PROCESS
         005E  0005BE00     VALID PROCESS
         005F  00069000     VALID PROCESS
         0060  00056200     VALID PROCESS
         0061  00056400     VALID PROCESS
         0062  00069200     VALID PROCESS
         0063  00056A00     VALID PROCESS
         0064  00069400     VALID PROCESS
         0065  00056C00     VALID PROCESS
         0066  00056E00     VALID PROCESS
         0067  00064000     VALID PROCESS
         0068  00069600     VALID PROCESS
         0069  0005C400     VALID PROCESS
         006A  00057200     VALID PROCESS
         006B  00057400     VALID PROCESS
         006C  00059200     VALID PROCESS
         006D  00059000     VALID PROCESS
```

Figure 5-10 Working Set List

5. SDA> SHOW PROCESS/PROCESS_SECTION_TABLE

Figure 5-11 shows the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                    21-MAY-1979 15:29:24.56
Process 34 dump: GROVE


Process section table information
---------------------------------

       Last entry allocated     FFE0
       First free PST entry     0000

Process section table
---------------------

       INDEX      ADDRESS   PAGES     WINDOW     VBN     CLUSTER  CHANNEL   REFCNT  FLINK BLINK   FLAGS

        FFF8     00000200 0000000F  8006B980  00000002       0  7FFE1DE0      0     FFE8  FFE8
        FFF0     00076600 00000008  80078050  00000002       0  7FFE1D90      7     FFF0  FFF0
        FFE8     00003C00 00000188  8006B980  00000013       0  7FFE1DE0    355     FFF8  FFF8
        FFE0     00077600 00000033  80079220  0000019C       0  7FFE1D60     51     FFE0  FFE0
```

Figure 5-11 Process Section Table

6. SDA> SHOW PROCESS/PAGE_TABLES

Figure 5-12 shows a portion of the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                    22-MAY-1979 10:40:52.99
Process 01 dump: SWAPPER

P0 page table
-------------

  ADDRESS      SVAPTE    PTE      TYPE  PROT  BITS PAGTYP    LOC STATE TYPE  REFCNT   BAK       SVAPTE   FLINK  BLINK

00000000    8005EE30 D0000B6D    VALID SRKW      K PPGTBL   ACTIVE  87   04       1   03000000  801FA7D8  0000   004A
00000200    8005EE34 D000066E    VALID SRKW      K PPGTBL   ACTIVE  87   04       1   03000000  801FA900  000F   0050
00000400    8005EE38 D40009DD    VALID SRKW M    K PROCESS  ACTIVE  07   00       1   03000000  800C81FC  0000   0049
00000600    8005EE3C D0000075    VALID SRKW      K PROCESS  ACTIVE  87   00       1   03000000  800C803C  0000   0051
00000800    8005EE40 D0000B77    VALID SRKW      K PROCESS  ACTIVE  87   00       1   03000000  800C80C8  0000   0052
00000A00    8005EE44 D0000B73    VALID SRKW      K PROCESS  ACTIVE  07   00       1   03000000  800C81D4  0000   0046
00000C00    8005EE48 D00005FB    VALID SRKW      K PROCESS  ACTIVE  07   00       1   03000000  800C81D0  0000   0047
00000E00    8005EE4C D4000C4C    VALID SRKW M    K PROCESS  ACTIVE  07   00       1   03000000  800C81CC  0000   0048
00001000    8005EE50 D00009DD    VALID SRKW      K PROCESS  ACTIVE  07   00       1   03000000  800C81FC  0000   0049
00001200    8005EE54 D0000075    VALID SRKW      K PROCESS  ACTIVE  87   00       1   03000000  800C803C  0000   0051
00001400    8005EE58 D0000B77    VALID SRKW      K PROCESS  ACTIVE  87   00       1   03000000  800C80C8  0000   0052
00001600    8005EE5C D4000290    VALID SRKW M    K PPGTBL   ACTIVE  87   04       1   03000000  801FA7E0  0000   004C
00001800    8005EE60 D4000397    VALID SRKW M    K PPGTBL   ACTIVE  87   04       1   03000000  801FA7DC  0000   004B

-------- 1 NULL PAGES

00001C00    8005EE68 D0000AA6    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  05E4   0288
00001E00    8005EE6C D0000960    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  01C9   00AE
00002000    8005EE70 D00001CB    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0271   0456
00002200    8005EE74 D0000D04    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0BC1   0C32
00002400    8005EE78 D00005A0    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0596   0523
00002600    8005EE7C D0000271    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0331   01CB
00002800    8005EE80 D0000BC7    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  049E   0BCA
00002A00    8005EE84 D0000066    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  06DF   0384
00002C00    8005EE88 D0000076    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  00CB   09F0
00002E00    8005EE8C D0000C32    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0D04   080C
00003000    8005EE90 D000080C    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0C32   0539
00003200    8005EE94 D0000539    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  080C   0038
00003400    8005EE98 D0000BC1    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0456   0D04
00003600    8005EE9C D00006AA    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0DD8   0B24
00003800    8005EEA0 D0000DE9    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0DD1   0C8C
00003A00    8005EEA4 D0000A9B    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  09A7   0044
00003C00    8005EEA8 D0000012    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0779   08BA
00003E00    8005EEAC D000094D    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  04E7   03FC
00004000    8005EEB0 D0000448    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0763   0839
00004200    8005EEB4 D00007E5    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0557   0AC3
00004400    8005EEB8 D0000456    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  01CB   0BC1
00004600    8005EEBC D000006F    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0952   0DE8
00004800    8005EEC0 D0000DCC    VALID SRKW      K GLOBAL   FREELST 00   02       0   0040FF08  801FE2E4  054C   075E
00004A00    8005EEC4 D0000DB9    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  070F   00CE
00004C00    8005EEC8 D0000234    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0A8B   067B
00004E00    8005EECC D0000A59    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  009A   0557
00005000    8005EED0 D000074B    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0407   040A
00005200    8005EED4 D000083B    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0050   0522
00005400    8005EED8 D0000900    VALID SRKW      K GLOBAL   FREELST 00   02       0   004CFFD0  801FD34C  074D   08E0
00005600    8005EEDC D000070F    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  086A   0DB9
00005800    8005EEE0 D00002CB    VALID SRKW      K PROCESS  FREELST 00   00       0   00000000  00000000  0511   0889
```

Figure 5-12 Program Region Memory

SHOW STACK

The SHOW STACK command displays the location and contents of the four stacks used by a given process as well as the system-wide interrupt stack.

Each qualifier displays one of four stacks that correspond to the four VAX/VMS processor access modes for a specific process. The /INTERRUPT qualifier displays the system-wide interrupt stack. The default for SHOW STACK is the stack that is currently being used or that was in use when the system failed.

Figure 5-13 shows the display produced by the SHOW STACK command. The display is the same for each stack, and is composed of four sections:

- Stack Pointer -- the stack pointer identifies the top of the stack. The display indicates the stack pointer by the symbol:

```
SP => 7FFEF868   00000001
```

- Stack address -- SDA lists all the virtual addresses allocated to the stack by the operating system. The stack addresses are listed in a column which increases by 4 (one longword).

- Stack contents -- SDA lists the contents of the stack in a column next to the stack addresses.

- Global symbols -- SDA attempts to display the contents of a location symbolically using a symbol and an offset. For example:

```
7FFEF868     7FFEE200  MMG$HDRBUF
7FFEF86C     7FFEE208  MMG$HDRBUF+08
```

  If the value is not within range of any existing symbols, the field will be left blank.

If a stack is empty, the stack pointer will point to the message:

```
SP =>              (THE STACK IS EMPTY)
```

SDA will display only five pages of any stack.

Format

SHOW STACK

Qualifiers	Defaults
/INTERRUPT	Stack currently in use (running system) or in use when system failed
/KERNEL	
/EXECUTIVE	
/SUPERVISOR	
/USER	
/ALL	

Parameters

None

Qualifiers

/INTERRUPT

Displays the interrupt stack. This stack is always resident in memory and is used during hardware interrupt processing.

/KERNEL

Displays the kernel stack for the current process.

/EXECUTIVE

Displays the executive stack for the current process.

/SUPERVISOR

Displays the supervisor stack for the current process.

/USER

Displays the user stack for the current process.

/ALL

Displays all the stacks described above.

Examples

1. SDA> SHOW STACK/KERNEL

   Figure 5-13 shows a portion of the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                          21-MAY-1979 15:34:40.49
Current operating stack

Current operating stack (KERNEL):

               7FFEEB18  00001000
               7FFEEB1C  7FFEEB20      CTL$GL_KSTKBAS+520
               7FFEEB20  7FFEF000      CTL$GL_KSPINI+400
               7FFEEB24  7FFEF828      CTL$GL_KSPINI+C28
               7FFEEB28  7FFDB65C
               7FFEEB2C  8007C200
               7FFEEB30  80006074      EXE$SWAPINIT+9F8
               7FFEEB34  00070000

         SP => 7FFEEB38  8006BAC0
               7FFEEB3C  7FFDFE00
               7FFEEB40  00000000
               7FFEEB44  801F1868
               7FFEEB48  80007814      MMG$EXTRADYNWS+164
               7FFEEB4C  00070000
               7FFEEB50  00000000
               7FFEEB54  00000002
               7FFEEB58  00020004
               7FFEEB5C  8000C4DF      EXE$DELTVA+0AA
               7FFEEB60  8000C487      EXE$DELTVA+052
               7FFEEB64  3FFFFFFF
               7FFEEB68  00000000
               7FFEEB6C  8000771F      MMG$EXTRADYNWS+06F
               7FFEEB70  00000000
               7FFEEB74  00000000
               7FFEEB78  00000000
               7FFEEB7C  00000000
               7FFEEB80  00000000
               7FFEEB84  7FFEEBC4      CTL$GL_KSTKBAS+5C4
               7FFEEB88  7FFEEB98      CTL$GL_KSTKBAS+598
               7FFEEB8C  8000B4E3      EXE$CMODEXEC+0D3
               7FFEEB90  80000116      SYS$DELTVA+006
               7FFEEB94  00800000
               7FFEEB98  00000000
               7FFEEB9C  20FC0000
               7FFEEBA0  7FFEF854      CTL$GL_KSPINI+C54
               7FFEEBA4  7FFEEBE4      CTL$GL_KSTKBAS+5E4
               7FFEEBA8  8000BBF7      MMG$IMGRESET+030
               7FFEEBAC  7FFEEBD4      CTL$GL_KSTKBAS+5D4
               7FFEEBB0  7FFE1DF0      CTL$GL_CCB
               7FFEEBB4  8006BAC0
               7FFEEBB8  7FFDFE00
               7FFEEBBC  FFFFFFFF
               7FFEEBC0  00000003
               7FFEEBC4  00000003
               7FFEEBC8  7FFEEBD4      CTL$GL_KSTKBAS+5D4
               7FFEEBCC  00000000
               7FFEEBD0  00000000
               7FFEEBD4  3FFFFFFF
               7FFEEBD8  00000000
               7FFEEBDC  00000000
               7FFEEBE0  80008E38      EXE$RUNDWN+04E
               7FFEEBE4  00000000
```

Figure 5-13 Current Operating Stack (Kernel)

SHOW SUMMARY

The SHOW SUMMARY command displays a formatted list of all active processes. The display shows the values used in swapping and scheduling for these processes. Figure 5-14 is an example of the display produced by the SHOW SUMMARY command. The information listed in the display includes:

- PID -- the 32 bit quantity that uniquely identifies the process
- PROCESS NAME -- the name assigned to the process
- IMAGE NAME -- the VAX/VMS file specification of the image currently executing in the process's context
- STATE -- the condition of the process (see the VAX/VMS System Manager's Guide for a description of possible states)
- PRI -- the current scheduling priority of the process
- UIC -- User Identification Code
- WKSET -- the total number (in decimal) of pages currently in the working set

If the process has been swapped out of the balance set, this message appears in the "Image Name" column:

```
---- SWAPPED OUT ----
```

Format

Qualifiers	Defaults
None	None

Parameters

None

Examples

1. SDA> SHOW SUMMARY

   Figure 5-14 shows the display produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                          21-MAY-1979 15:36:26.03
Current process summary

   PID    PROCESS NAME    IMAGE NAME                      STATE PRI    UIC    WKSET
   ---    ------------    ----------                      ----- ---    ---    -----

00010000 NULL                                             COM     0 [000,000]     0
00010001 SWAPPER                                          HIB    16 [000,000]     0
00010019 _TTA3:          DBA0:[SYSEXE]VMOUNT.EXE;         LEF     4 [017,022]    59
0003001A _TTF7:          DBA0:[SYSEXE]MAIL.EXE;           COM     4 [320,100]    74
0005001B MANDERLEY       DBA0:[SYSEXE]RSX.EXE;            LEF     7 [360,007]   148
000B001C _TTA1:          DBA0:[SYSEXE]LOGINOUT.EXE;       LEF     4 [010,040]    39
0014001D KAREN                                            LEF     4 [361,006]    41
000C001E CRAIG           DBA0:[SYSEXE]BACKTRANS.EXE;      LEF     4 [320,111]    70
0001001F DERF                                             LEF     8 [320,114]    41
000B0020 USER                                             LEF     7 [304,003]   150
00100021 NOAH            DBA0:[SYSEXE]LOGINOUT.EXE;       LEF     7 [361,002]    67
00050022 LYNN            DBA0:[SYSEXE]SUBMIT.EXE;8        LEF     4 [320,110]    68
001E0023 LAMONT          DBA0:[SYSEXE]SHOW.EXE;           LEF     4 [360,003]    61
00170024 OZZIE           DBA0:[SYSEXE]TYPE.EXE;31         LEF     4 [360,002]    70
00060025 CLEO            DBA0:[SYSEXE]DELETE.EXE;4        LEF     4 [361,004]    68
00020026 MAJA            DBA0:[SYSEXE]COPY.EXE;           LEF     4 [304,002]    55
00170027 BOUSQUET        DBA0:[SYSEXE]TALK.EXE;           LEF     4 [011,016]    53
00020028 BACH                                             LEF     7 [360,016]    45
00040029 WIZARD          DBA0:[SYSEXE]BACKTRANS.EXE;      LEF     5 [017,022]   149
000F002A BOUFFON                                          COM     4 [300,041]   300
000B002B HARLEY          DBA0:[SYSEXE]SHOW.EXE;           LEF     4 [017,022]    58
0018002C DAVIDSON        DBA0:[SYSEXE]BACKTRANS.EXE;      LEF     8 [361,013]    43
000F002D RMS             DBA0:[SYSEXE]COPY.EXE;           LEF     4 [011,016]    70
0007002E _TTG4:          DBA0:[SYSEXE]DISPLAY.EXE;        LEF     7 [311,001]    39
0013002F KURT            --- SWAPPED OUT ---              LEFO    4 [361,003]    61
000F0030 MEYERS                                           LEF     4 [360,005]    65
000F0031 EDWIN                                            LEF     8 [360,001]   150
00040032 _TTG3:          --- SWAPPED OUT ---              LEFO    4 [311,001]    45
00090033 WOODROW         DBA0:[SYSEXE]USERS.EXE;          LEF     4 [201,201]    50
00200034 FRED            DB2:[F4V2.TOOL]BLISS32.EXE;43    CUR     5 [320,100]   338
00190035 REID            --- SWAPPED OUT ---              LEFO    7 [361,010]   106
00160036 LOWELL                                           LEF     5 [301,021]    45
00210037 OPPENHEIM       --- SWAPPED OUT ---              LEFO    4 [360,023]    43
00010038 NETACP                                           HIB     9 [001,001]    54
00010039 PRTSYMB4        DBA0:[SYSEXE]PRTSMB.EXE;         HIB     8 [001,004]    41
0001003A PRTSYMB3        DBA0:[SYSEXE]PRTSMB.EXE;         HIB     8 [001,004]    41
0001003B PRTSYMB2        DBA0:[SYSEXE]PRTSMB.EXE;         HIB     8 [001,004]    41
0001003C PRTSYMB1        DBA0:[SYSEXE]PRTSMB.EXE;         HIB     8 [001,004]    41
0001003D DBA2ACP         DBA0:[SYSEXE]F11BACP.EXE;        HIB    11 [001,003]   104
0031003E GUITAR                                           HIB     7 [361,010]    32
0001003F ERRFMT                                           HIB     9 [001,006]    30
00010040 OPCOM                                            LEF     8 [001,004]    38
00010041 JOB_CONTROL                                      HIB    13 [001,004]   100
00010042 DBA0ACP                                          HIB    11 [001,003]   100
00020043 DBA1ACP         DBA0:[SYSEXE]F11BACP.EXE;        HIB    10 [001,003]   105
```

Figure 5-14 Summary of Active Processes

SHOW SYMBOL

The SHOW SYMBOL command displays a local or global symbol and the value associated with it. If the value is a valid memory location, SDA examines that address and displays its contents.

Format

SHOW SYMBOL symbol-name

Qualifiers	Defaults
/ALL	None

Parameters

symbol-name

Specifies an SDA symbol that corresponds to an SDA expression. See Section 4.2.5 for more information on SDA symbols.

Qualifiers

/ALL

Displays two lists of the entire SDA symbol table. The first list organizes the local and global symbols in alphabetical order. The second list organizes these symbols by their values, starting at the lowest value. If the value represents an address, the contents of the memory location will be displayed:

```
TTY$A_CTRLZ 80002A12 => 000D5A5E
```

If you specify a symbol name and the /ALL qualifier, SHOW SYMBOL displays a list of all the symbols that begin with the specified symbol name. For example, SHOW SYMBOL IOC$GL displays all the symbols with starting characters IOC$GL.

Examples

1.
```
SDA> SHOW SYMBOL BUG$FATAL
BUG$FATAL = 80008256 :  08309F9E
```

In this example, the global symbol, its system virtual address, and the value stored at the address are shown.

2.
```
SDA> DEFINE START = 00000000
SDA> SHOW SYMBOL START
START = 00000000 :  009A029A
```

In this example, a local symbol is defined. See the description of the DEFINE command for more information about symbol definition.

3.
```
SDA> SHOW SYMBOL/ALL
```

Figure 5-15 shows a page of the listing produced by this command.

```
VAX/VMS 2.0 -- System Dump Analysis                    21-MAY-1979 16:04:51.71
System global symbols by name

IO$M_FCODE       0000003F
IO$_LOGICAL      0000002F
IO$_PHYSICAL     0000001F
IO$_READLBLK     00000021
IO$_READPBLK     0000000C
IO$_READVBLK     00000031
IO$_WRITELBLK    00000020
IO$_WRITEPBLK    0000000B
IO$_WRITEVBLK    00000030
IOC$ALOUBAMAP    8000A807 => A53C38BB
IOC$ALOUBAMAPN   8000A803 => 151138BB
IOC$ALTUBAMAP    8000A7E5 => 5326A19A
IOC$APPLYECC     8000A8AC => C53C18BB
IOC$CANCELIO     8000A551 => 58A508E1
IOC$CREATE_UCB   8000B7F6 => 5108A53C
IOC$CVTLOGPHY    8000A920 => 34A350D0
IOC$CVT_DEVNAM   8000B87A => D07E507D
IOC$DELMBX       8000A568 => 5024A5D0
IOC$DIAGBUFILL   8000A5EE => 534CA5D0
IOC$DIRPOST1     80005544 => 5024A5D0
IOC$DISMOUNT     8000B665 => A6D028BB
IOC$FFCHAN       8000B8AF => 463BEFC1
IOC$FILSPT       8000A3C0 => 04C553DD
IOC$GL_ADPLIST   8000083C => 8005E800
IOC$GL_AQBLIST   80003A10 => 80063310
IOC$GL_DEVLIST   80000838 => 80000848
IOC$GL_DIALUP    80003A1C => 800688A0
IOC$GL_DPTLIST   80000840 => 80069090
IOC$GL_IRPBL     80003A0C => 8007B390
IOC$GL_IRPFL     80003A08 => 80076F70
IOC$GL_MUTEX     80003B28 => 0000FFFF
IOC$GL_PSBL      80003A04 => 80003A00
IOC$GL_PSFL      80003A00 => 80003A00
IOC$GQ_BRDCST    80003A20 => 80003A20
IOC$GQ_MOUNTLST  80003A14 => 80056A60
IOC$GW_MAXBUF    80004466 => 04000400
IOC$GW_MBXBFQUO  80004468 => 01000400
IOC$GW_MBXMXMSG  8000446A => 00100100
IOC$GW_MBXNMMSG  8000446C => 00100010
IOC$INITDRV      800126FF
IOC$INITIATE     8000A6F4 => 4CA553D0
IOC$IOPOST       80005078 => 7D7E547D
IOC$LOADMBAMAP   8000AA04 => A53C53DD
IOC$LOADUBAMAP   8000AA4E => 3C7E537D
IOC$MAPVBLK      8000A947 => 120AA291
IOC$MOVFRUSER    8000A388 => A5A83610
IOC$MOVTOUSER    8000A3A4 => A5A81A10
IOC$PTETOPFN     8000AAB7 => 00008FCA
IOC$QNXTSEG      800053C0 => 5218A3D0
IOC$QNXTSEG1     800053CC => 3044A4B7
IOC$REINITDRV    80012705
IOC$RELCHAN      8000A628 => 5020A5D0
IOC$RELDATAP     8000A729 => 5020A5D0
IOC$RELMAPREG    8000A799 => 5120A5D0
IOC$RELSCHAN     8000A61E => 5020A5D0
IOC$REQCOM       8000A6B3 => 534CA5D0
IOC$REQDATAP     8000A762 => A5D001DD
IOC$REQDATAPNW   8000A75E => 021100DD
IOC$REQMAPREG    8000A7D3 => 50E83210
IOC$REQPCHANH    8000A67F => 5020A5D0
IOC$REQPCHANL    8000A688 => 5020A5D0
IOC$REQSCHANH    8000A66B => 5020A5D0
IOC$REQSCHANL    8000A675 => 5020A5D0
IOC$RETURN       8000A869 => 6E02C005
IOC$SEARCHDEV    8000B8D6 => 115202D0
IOC$SEARCHGEN    8000B8DB => BB5201D0
IOC$UNLOCK       8000B9F9 => EFD050DD
IOC$UPDATRANSP   8000A9C8 => 6EA550A2
IOC$VERIFYCHAN   8000BA0B => 13500FAA
IOC$WAKACP       800053F4 => DA7E12DB
IOC$WFIKPCH      8000A86A => 7D6E02C0
IOC$WFIRLCH      8000A88A => 7D6E02C0
IPL$_ASTDEL      00000002
IPL$_HWCLK       00000018
IPL$_IOPOST      00000004
IPL$_MAILBOX     0000000B
IPL$_POWER       0000001F
IPL$_QUEUEAST    00000006
IPL$_SCHED       00000003
IPL$_SYNCH       00000007
IPL$_TIMER       00000007
KFI$GL_F11AACP   80003B40 => 80056830
LIB$CVT_DTB      8000F03E => 0AD0003C
LIB$CVT_HTB      8000F04C => 10D0003C
LIB$CVT_OTB      8000F045 => 08D0003C
LOG$AL_DISKLOG   80011A08
LOG$AL_LOGTBL    800039C8 => 800039E8
LOG$AL_MUTEX     800039D4 => 0000FFFF
LOG$DELETE       8000BA3E => D050610F
LOG$GL_SLTFL     800039E8 => 80056A30
LOG$INSLOGN      8000BA60 => 530BA59A
LOG$LOCKR        8000BAA7 => DF64CF9F
LOG$LOCKW        8000BAAD => DF44CF9F
LOG$SEARCHLOG    8000BAC8 => 00608FBB
LOG$TRNSLOGNAME  8000BB14 => 1350613C
LOG$UNLOCK       8000BAB3 => DFB7CF9F
MB$DDT           8000127C => 0000031B
MB$DPT           80000F34 => 80001046
MB$GL_DDB        80000BE4 => 80000D78
MB$GL_UCB1       80000C90 => 80000C90
MB$GL_UCB2       80000D04 => 80000D04
MB$UCB0          80000C1C => 80000C1C
MBA$INITIAL      80001271 => 04A401D0
MBA$INT          80001200 => 5300BED0
MMG$ALCPAGFIL    80006FFD => 16DF41D0
MMG$ALCPHD       8000BD5E => 5024A532
MMG$ALCSTX       8000BD35 => 5126A532
MMG$ALLOCPFN     80006B82 => 923052D4
MMG$AL_BEGDRIVE  80001200 => 5300BED0
MMG$AL_ENDDRIVE  80002E00 => 01AD0983
MMG$AL_PGDCOD    8000B400 => D500CF31
MMG$AL_PGDCODEN  80011000
MMG$AL_SBICONF   80003264 => 00000000
MMG$AL_SYSPCB    800038D8 => 800038D8
MMG$A_ENDVEC     80000600 => 0000007A
MMG$A_PAGFIL     800031D8 => 80060B7C
MMG$A_SYSPARAM   80004400 => E0BECC20
MMG$CRECOM1      8000C39F => 547D5510
MMG$CRECOM2      8000C3A7 => DA00EE30
MMG$CREPAG       8000C51D => 1EE150DD
MMG$CRETVA       8000C380 => AC9A01FC
MMG$DALCPAGFIL   800070C4 => 4FDF41D0
MMG$DALCSTX      8000BD1E => 5520A5C1
MMG$DALCSTXSCN   8000BCA5 => 3AA501E7
MMG$DALLOCPFN    80006D23 => 40B552D4
MMG$DECPHDREF    80006B2C => 5146A53C
MMG$DECPHDREF1   80006B30 => 2BDF41B7
MMG$DECPTREF     80006AC5 => 531509EF
MMG$DECSECREF    8000712E => 5520A5C1
MMG$DELCONPFN    80006B97 => F8DF40D0
MMG$DELGBLSEC    80007963 => 501CA3D0
MMG$DELGBLWCB    8000D639 => 6409FF0F
MMG$DELPFNLST    80006C8C => 10889610
MMG$DELWSLEPPG   800069B9 => A4B70710
MMG$DELWSLEX     80006995 => 526541D0
MMG$DGBLSC1      8000D518 => 071100FC
MMG$EXTRADYNWS   800076B0 => A56CA5A3
MMG$FREWSLE      8000682F => EF327E7C
MMG$FREWSLX      800067E7 => 7E7C01DD
MMG$FRSTRONLY    80004800 => E15622DB
MMG$GETPTIPAG    80007A12 => DD7E12DB
MMG$GL_CRDCNT    80003FB0 => 00000001
MMG$GL_CTLBASVA  80004540 => 7FFDFE00
MMG$GL_FRESVA    80004558 => 80200000
MMG$GL_GPTBASE   8000454C => 801F9000
MMG$GL_GPTE      80004550 => 801FD000
MMG$GL_IACLOCK   80002E84 => 00000000
MMG$GL_MAXGPTE   80004554 => 80200000
MMG$GL_MAXPFN    80004584 => 00000E17
MMG$GL_MAXSYSVA  80004558 => 80200000
MMG$GL_NPAGEDYN  8000457C => 8005E800
MMG$GL_PAGEDYN   80004580 => 80056800
MMG$GL_PAGSWPVC  80003218 => 8000322C
MMG$GL_PFNLOCK   80002E88 => 40000000
MMG$GL_PHYPGCNT  8000445E => 00004000
MMG$GL_RMSBASE   800032A4 => 80019C00
MMG$GL_SBR       80004578 => 001FC000
MMG$GL_SPTBASE   8000455C => 801F9000
MMG$GL_SPTLEN    80004560 => 00001000
MMG$GL_SYSPHD    80004564 => 801F8400
MMG$GL_SYSPHDLN  80004568 => 00000C00
MMG$GSDMTXULK    8000D503 => 5C1BEFD0
MMG$GSDSCN       8000D599 => 51D5527C
MMG$IMGACTBUF    7FFEDA00 => 00000000
MMG$IMGHDRBUF    7FFEE200 => 7FFEE208
```

Figure 5-15 Global Symbols

CHAPTER 6

ANALYZING SYSTEM FAILURES -- GUIDELINES AND EXAMPLES

This chapter discusses how VAX/VMS handles internal errors and suggests procedures that can aid in determining the cause of these errors. The final sections of the chapter illustrate, through detailed analysis of a sample system failure, how SDA helps you find operating system problems.

6.1 GENERAL PROCEDURE FOR SOLVING SYSTEM FAILURES

When the VAX/VMS operating system detects an internal error so severe that normal operation cannot continue, it signals a condition known as a fatal bugcheck and shuts itself down. A bugcheck describes the error discovered by the system; each error is associated with a particular bugcheck code.

To resolve the condition, you must find the reason for the bugcheck. You generally need the VAX/VMS source code to locate the error, unless the error exists in a driver that is not supplied by DIGITAL. If this is the case, you may simply need the driver listings.

The best way to start the search for the error is to locate the line of code that signaled the bugcheck. The address of this instruction is usually contained in the Program Counter register (PC). Invoke SDA and give the SHOW CRASH command. The display SDA produces gives the contents of the PC.

Next, examine the system map file SYS$SYSTEM:SYS.MAP. This file lists the addresses of each VAX/VMS module that resides in system space (the part of the operating system that performs basic system services and scheduling). Compare the address in the PC with the addresses in the system map file to locate the module that uses the instruction pointed to by the PC.

If you do not have the system map file, you can use SDA's symbol table. All the system global symbols that appear in SYS.MAP also exist in the SYS.STB file that SDA reads during the initialization process. To determine the offset from the closest global symbol, issue the command:

```
SDA> EXAMINE @PC
```

Once you have narrowed the search to a particular module, subtract the module's starting address from the address in the PC to get the offset into the code.

Now, to determine the general cause of the system failure, examine the code that signaled the bugcheck.

6.2 FATAL BUGCHECK CONDITIONS

If a bugcheck is signaled, it is usually caused by one of two conditions:

- A fatal exception
- An illegal page fault

6.2.1 Fatal Exceptions

A fatal exception is an event that causes VAX/VMS to signal a fatal bugcheck. An exception is fatal when it occurs while a process is:

- Using the interrupt stack
- Executing above IPL 2 (IPL$_ASTDEL)
- Executing in a privileged (kernel or executive) processor access mode

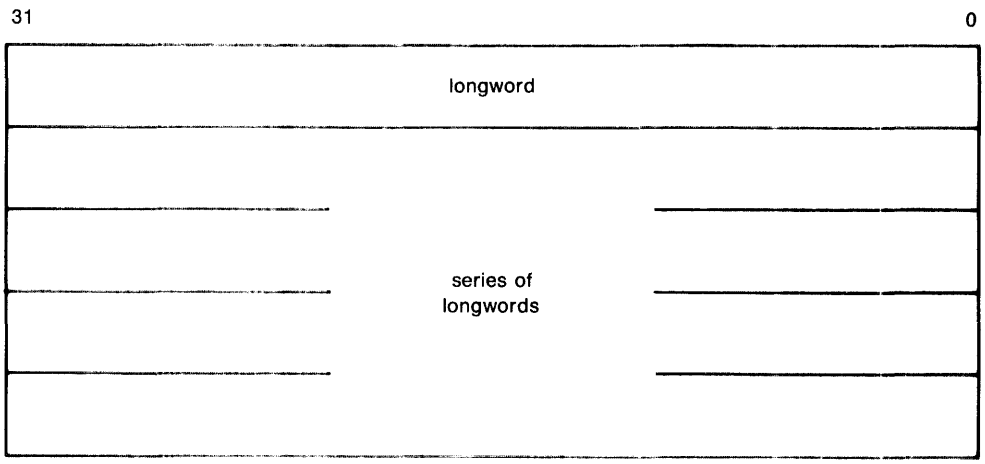
When the system fails, it lists the immediate cause of the failure on the LSI-11 console. For fatal exceptions, the messages appear as follows:

```
FATALEXCPT, Fatal executive or kernel mode exception

INVEXCEPTN, Exception while above ASTDEL or on interrupt stack
```

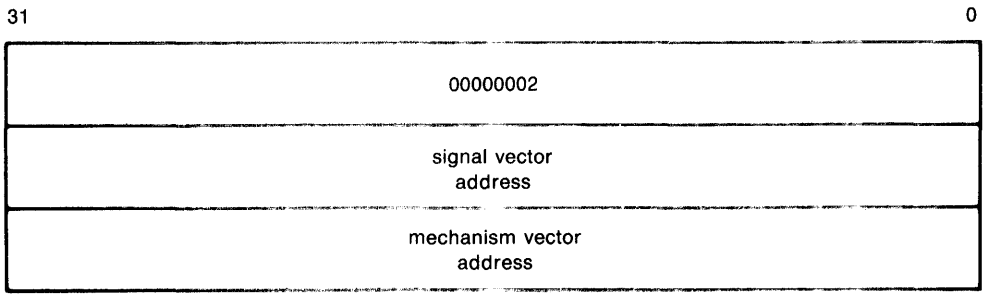
Although there are several possible exception conditions, the type that most commonly occurs is the access violation. The rest of this section discusses the access violation in detail. For more information on other kinds of exceptions, see the VAX-11 Run-Time Library Reference Manual.

When an access violation is detected by the VAX-11 hardware, information useful in finding the cause of the violation is pushed onto the current operating stack, that is, the stack that the process was using when the access violation occurred. This information is described by three structures, referred to as vectors. A vector is structured as follows:

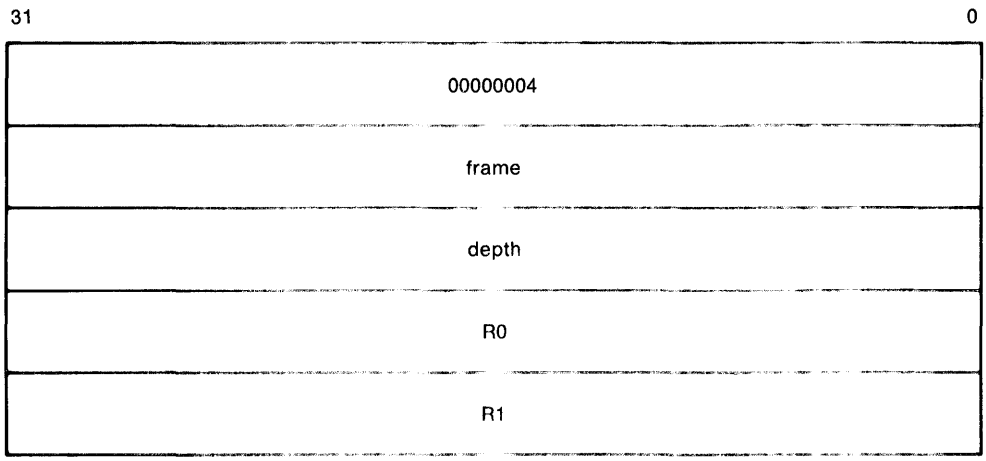


The first longword in the vector shows the number of longwords that follow. Each longword in the series contains information describing conditions at the time of the exception.

The first vector that appears on the stack gives the addresses of the next two vectors:



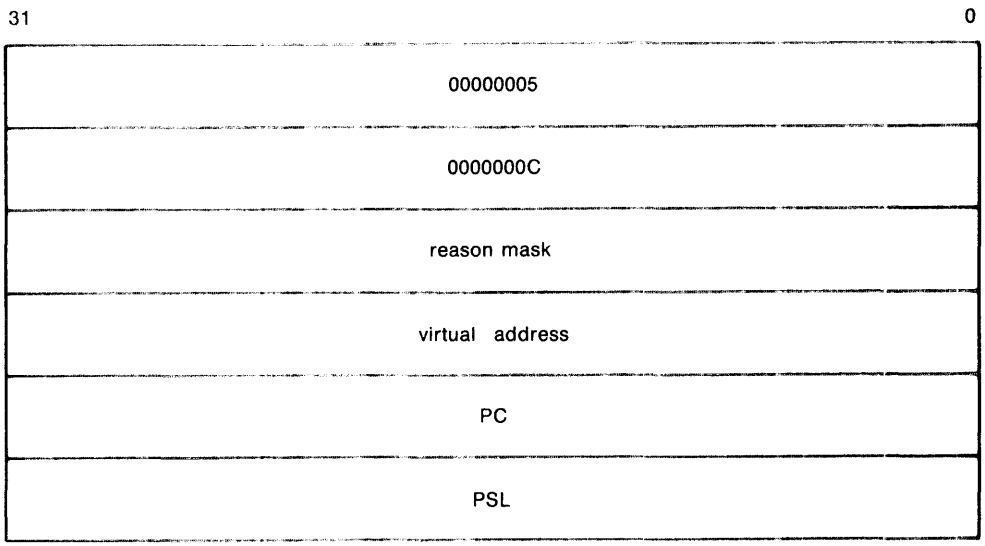
The mechanism vector follows the first vector. This structure describes the process that was executing when the exception occurred. The diagram below illustrates the sequence of longwords in a mechanism vector:



The values contained in this vector are:

- 00000004 -- the number of longwords that follow. In a mechanism vector, this value is always four.
- Frame -- the address of the stack frame.
- Depth -- the stack depth.
- R0 -- the contents of R0 at the time of the exception.
- R1 -- the contents of R1 at the time of the exception.

The next vector created on the stack is the signal vector. For access violations, the signal vector is set up as follows:



The parameters shown in the above diagram are:

- 00000005 -- the number of longwords that follow. For access violations, this value is always five.
- 0000000C -- the exception code. This value is C (hexadecimal) to represent an access violation.
- Reason mask -- the longword whose lowest three bits, if set, indicate that the instruction caused a length violation (bit 0), referenced the process page table (bit 1), and attempted a read/modify operation (bit 2).
- Virtual address -- the virtual address that the system tried to reference.
- PC -- the Program Counter. The PC contains the address of the instruction that signaled the exception.
- PSL -- the processor status longword at the time of the exception.

Signal vectors differ in length, depending on the kind of exception the system detects. See the VAX-11 Run-Time Library Reference Manual for details.

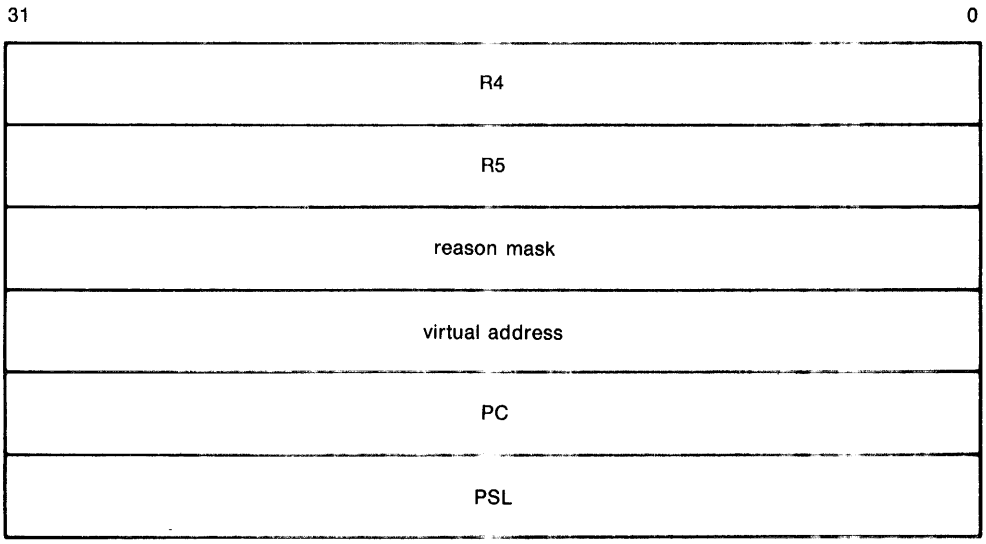
If VAX/VMS encounters a fatal exception, you can find the code that signaled it by examining the PC placed in the signal vector. Issue the SHOW STACK command to display the current operating stack, then locate the vectors. Once you obtain the PC, which points to the instruction that signaled the exception, you can identify the module by the procedure outlined in Section 6.1.

6.2.2 Illegal Page Faults

VAX/VMS also signals a bugcheck when a page fault occurs while the Interrupt Priority Level (IPL) is greater than two (IRP$_ASTDEL). When VAX/VMS fails because of an illegal page fault, it issues the following message on the console:

```
PGFIPLHI, Page fault with IPL too high
```

In this case, information is pushed on the stack as longwords in the following sequence:



The longwords pushed on the stack are:

- R4 -- the contents of R4 at the time of the bugcheck.
- R5 -- the contents of R5 at the time of the bugcheck.
- Reason mask -- see Section 6.2.1.
- Virtual address -- the virtual address that caused the page fault.
- PC -- the Program Counter. The PC contains the address of the instruction that was executing when the page fault was issued.
- PSL -- the processor status longword at the time of the bugcheck.

If the operating system detects a page fault while the IPL is higher than two, you can obtain the faulting instruction by examining the PC pushed on the current operating stack. Follow the steps outlined in Section 6.1 to determine which module issued the instruction.

6.3 DEBUGGING A SYSTEM FAILURE -- AN EXAMPLE

This section steps through the analysis of a system failure. The events that lead up to this failure are:

- The line printer goes offline for three hours.
- The line printer comes back online.
- The operating system signals a bugcheck, writes information to the system dump file, and shuts itself down.

6.3.1 Identifying the Bugcheck

Invoke SDA to analyze the system dump file. The initialization message indicates the type of bugcheck signaled:

```
VAX/VMS System dump analyzer
Dump taken on 31-JUL-1979    20:43:13.32
INVEXCEPTN, Exception while above ASTDEL or on interrupt stack

SDA>
```

VAX/VMS encountered an exception that caused it to signal a bugcheck. Signal and mechanism vectors are created on the current operating stack.

6.3.2 Identifying the Exception

Issue the SHOW STACK command to display the current operating stack, which, in this case, is the interrupt stack. Figure 6-1 shows the interrupt stack and highlights the three vectors.

```
Current operating stack (INTERRUPT)

                8006A378  80008448        ACP$WRITEBLK+0A0
                   .
                   .
                   .
        SP =>   8006A398  7FFDC340
                8006A39C  8006A3A0
                8006A3A0  80004E7D        EXE$RELECT+0D4
                8006A3A4  04080009
                8006A3A8  00000004
mechanism       8006A3AC  7FFDC368
vector          8006A3B0  FFFFFFFD
                8006A3B4  8001774E
                8006A3B8  0000074F
                8006A3BC  00000005
                8006A3C0  0000000C
signal          8006A3C4  00000000
vector          8006A3C8  80069E00
                8006A3CC  8005D003
                8006A3D0  04080000
                8006A3D4  80009604        EXE$FORKDSPTH+01C
                   .
                   .
                   .
```

Figure 6-1 Interrupt Stack and Vectors

Examination of the signal vector shows that:

- The exception code is C (hexadecimal) which means that an access violation occurred.
- The reason mask is zero, which means that the instruction generated a protection violation (instead of a length violation) when it tried to read the location (rather than write to it).
- The virtual address is 80069E00 and is the address that the instruction tried to reference.
- The PC is 8005D003 and is the address of the instruction that signaled the exception.
- The IPL was eight at the time of the exception (shown by bits 16 through 20 of the PSL).
- The current operating stack was the interrupt stack (bit 26 of the PSL is set to 1).
- The process was executing in kernel mode at the time of the exception (shown by bits 24 and 25 of the PSL).

Use the SHOW PAGE_TABLE command to display the system page table, as shown in Figure 6-2. The page containing location 80069E00 is not available to any access mode (a null page); thus, the virtual address is not valid.

```
VAX/VMS 1.0 -- System Dump Analysis                    10-DEC-1979 19:21:00.32    Page 17
System page table

 ADDRESS      SVAPTE    PTE      TYPE  PROT  BITS PAGTYP    LOC STATE TYPE  REFCNT   BAK       SVAPTE   FLINK  BLINK

-------- 1 NULL PAGE

8006A000    800CD340 B0000SED    VALID ERKW     K
8006A200    800CD344 B4000SEE    VALID ERKW M   K

-------- 1 NULL PAGE

8006A600    800CD34C D4000389    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD34C 0000   004A
8006A800    800CD350 D4000144    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD350 0000   004B
8006AA00    800CD354 D4000028    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD354 0000   004C

-------- 3 NULL PAGES

8006B200    800CD364 D40000A7    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD364 0000   004D
8006B400    800CD368 D4000446    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   0300242A  800CD368 0035   00E0
8006B600    800CD36C D40004A8    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   033FFFFF  800CD36C 0021   00E3
8006B800    800CD370 D40002F5    VALID SRKW     K PPGTBL   ACTIVE  87   04       1   0300264B  800CD370 0000   0082
8006BA00    800CD374 50000000    DZERO SRKW     K
8006BC00    800CD378 50000000    DZERO SRKW     K
8006BE00    800CD37C 50000000    DZERO SRKW     K
8006C000    800CD380 50000000    DZERO SRKW     K
8006C200    800CD384 50000000    DZERO SRKW     K
8006C400    800CD388 50000000    DZERO SRKW     K
8006C600    800CD38C 50000000    DZERO SRKW     K
8006C800    800CD390 50000000    DZERO SRKW     K
8006CA00    800CD394 50000000    DZERO SRKW     K
8006CC00    800CD398 50000000    DZERO SRKW     K
8006CE00    800CD39C 50000000    DZERO SRKW     K
8006D000    800CD3A0 50000000    DZERO SRKW     K
8006D200    800CD3A4 50000000    DZERO SRKW     K
8006D400    800CD3A8 50000000    DZERO SRKW     K
8006D600    800CD3AC 50000000    DZERO SRKW     K
8006D800    800CD3B0 50000000    DZERO SRKW     K
8006DA00    800CD3B4 50000000    DZERO SRKW     K
8006DC00    800CD3B8 50000000    DZERO SRKW     K
8006DE00    800CD3BC 50000000    DZERO SRKW     K
8006E000    800CD3C0 50000000    DZERO SRKW     K
8006E200    800CD3C4 50000000    DZERO SRKW     K
8006E400    800CD3C8 50000000    DZERO SRKW     K
8006E600    800CD3CC 50000000    DZERO SRKW     K
8006E800    800CD3D0 50000000    DZERO SRKW     K
8006EA00    800CD3D4 50000000    DZERO SRKW     K
8006EC00    800CD3D8 50000000    DZERO SRKW     K
8006EE00    800CD3DC D40002E9    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   033FFFFF  800CD3DC 0003   00B5
8006F000    800CD3E0 D4000353    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD3E0 0040   004E
8006F200    800CD3E4 50000000    DZERO SRKW     K
8006F400    800CD3E8 D4000194    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD3E8 0000   004A
8006F600    800CD3EC D400005D    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD3EC 0000   004B
8006F800    800CD3F0 D4000122    VALID SRKW M   K PPGTBL   ACTIVE  87   04       1   03000000  800CD3F0 0000   004C

-------- 3 NULL PAGES
```

Figure 6-2 Page Table Display Showing Invalid Location 80069E00

6.3.3 Locating the Source of the Exception

Because the line printer went offline and then online, the problem may exist in the driver code. To determine which driver might contain the faulty code, take the address contained in the PC on the stack and compare it with the bounds of each driver.

6.3.3.1 **Finding the Driver Using the DPT List** - The Driver Prologue Table (DPT) is a data structure that describes each driver. All the driver prologue tables form a linked list; each DPT is followed directly by driver code. The location IOC$GL_DPTLIST contains the address of the first DPT. Figure 6-3 illustrates the linked structure of the driver prologue tables.

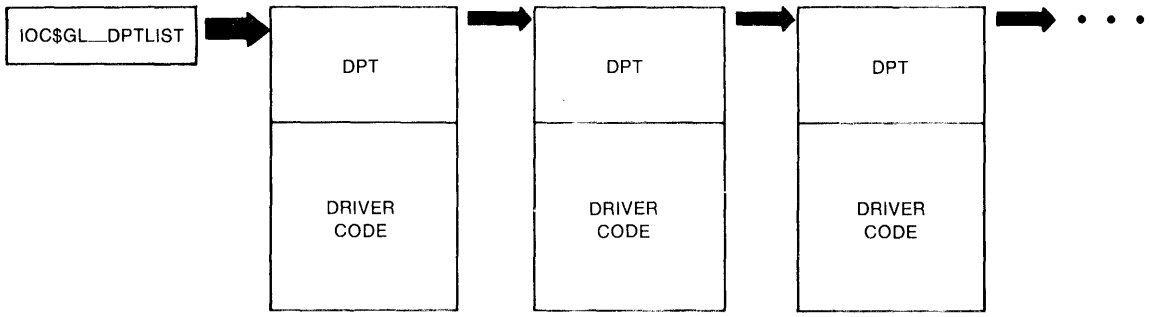


Figure 6-3 Linked List of Driver Prologue Tables

Use the FORMAT command and specify the contents of IOC$GL_DPTLIST as a parameter:

```
SDA> FORMAT@IOC$GL_DPTLIST

80060500        DPT$L_FLINK             8005F400
80060504        DPT$L_BLINK             80000000
80060508        DPT$W_SIZE                  07F0
8006050A        DPT$B_TYPE                1E
8006050B        DPT$B_REFC              01
8006050C        DPT$B_ADPTYPE                01
8006050D        DPT$B_FLAGS                02
8006050E        DPT$W_UCBSIZE           00F0
80060510        DPT$W_INITTAB                001F
80060512        DPT$W_REINITTAB         0062
80060514        DPT$W_UNLOAD                 0000
80060516        DPT$T_NAME              "DPDRIVER"
SDA>
```

The formatted display identifies the size of the driver by the symbol DPT$W_SIZE.

Calculate the end of the driver by adding the value of DPT$W_SIZE to the starting address of the DPT for the driver. The driver code begins just after the DPT.

Next, determine whether the address in the PC falls within the range of addresses that contain the driver code. If the address is not part of the driver you are examining, continue on to the next driver by stepping through the linked list with the FORMAT command (see the description of the FORMAT command in Chapter 5 for an example of the commands used to step through a linked list of data structures).

In this example, the instruction that caused the exception falls within the range of addresses that contain the line printer driver code.

6.3.3.2 **Calculating the Offset into the Driver** - Once you have identified the driver, you can locate the instruction in the source code by subtracting the starting address of the driver prologue table from the address contained in the PC. Match the resulting offset with the offsets in the driver code listing.

After you have located the routine that caused the exception, you should examine memory to make sure that the instruction in the routine matches the instruction that signaled the exception.

6.3.4 Finding the Problem within the Routine

Examine the line printer driver code. The instruction that caused the exception is MOVB (R3)+,(R0), as shown in Figure 6-4. To check the contents of R3, use the SHOW CRASH command. The invalid virtual address 80069E00 is indeed stored in R3.

```
                        029F    480 ;   START NEXT OUTPUT SEQUENCE
                        029F    481 ;
                        029F    482
50      54      02      C1      029F    483 10$:    ADDL3       #LP_DBR,R4,R0           ;CALCULATE ADDRESS OF DATA BUFFER REGISTER
  51      6C    A5      3C      02A3    484         MOVZWL      UCB$W_BOFF(R5),R1       ;GET NUMBER OF CHARACTERS REMAINING
52      8080    8F      B0      02A7    485         MOVW        #^X8080,R2              ;GET CONTROL REGISTER TEST MASK
                08      11      02AC    486         BRB         25$                     ;
        64      52      B3      02AE    487 20$:    BITW        R2,(R4)                 ;PRINTER READY OR HAVE PAPER PROBLEM?
                08      15      02B1    488         BLEQ        30$                     ;IF LEQ NOT READY OR PAPER PROBLEM
        60      83      90      02B3    489         MOVB        (R3)+,(R0)              ;OUTPUT NEXT CHARACTER
          F5    51      F4      02B6    490 25$:    SOBGEQ      R1,20$                  ;ANY MORE CHARACTERS TO OUTPUT?
                70      11      02B9    491         BRB         70$                     ;
                                02BB    492
                                02BB    493 ;
                                02BB    494 ;   PRINTER IS NOT READY OR HAS PAPER PROBLEM
                                02BB    495 ;
                                02BB    496
                21      12      02BB    497 30$:        BNEQ    40$                     ;IF NEQ PAPER PROBLEM
          51    01      A1      02BD    498             ADDW3   #1,R1,UCB$W_BOFF(R5)    ;SAVE NUMBER OF CHARACTERS REMAINING
            6C  A5              02C0
                                02C2    499     DSBINT                                  ;DISABLE INTERRUPTS
        64  40  8F      88      02C8    500             BISB    #^X40,LP_CSR(R4)        ;SET INTERRUPTS
                                02CC    501             WFIKPCH 40$,#12                 ;WAIT FOR INTERRUPT
                                02D6    502             IOFORK                          ;CREATE A FORK PROCESS
                C1      11      02DC    503             BRB     10$                     ;
                                02DE    505 ;
                                02DE    506 ;   PRINTER HAS PAPER PROBLEM
                                02DE    508
          7A    A5      94      02DE    509 40$:        CLRB    UCB$B_LP_OFLCNT(R5)     ;CLEAR OFFLINE COUNTER
          51    01      A1      02E1    510             ADDW3   #1,R1,UCB$W_BOFF(R5)    ;SAVE NUMBER OF CHARACTERS REMAINING
            6C  A5              02E4
                64  B4          02E6    511 50$:        CLRW    LP_CSR(R4)              ;DISABLE PRINTER INTERRUPT
                                02E8    512             SETIPL  UCB$B_FIPL(R5)          ;LOWER TO FORK LEVEL
                64  B5          02EC    513             TSTW    LP_CSR(R4)              ;PRINTER STILL HAVE PAPER PROBLEM?
                AF      14      02EE    .1              BGTR    10$                     ;IF GTR NO
3E  58  A5      03      E0      02F0    515             BBS     #UCB$V_CANCEL,UCB$W_STS(R5),80$ ;IF SET, CANCEL I/O OPERATION
        01      0F      9D      02F5    516             ACBB    #15,#1,UCB$B_LP_OFLCNT(R5),80$  ;SKIP UNTIL TIMEOUT
    0017    7A  A5              02F8
            7A  A5      94      02FC    517             CLRB    UCB$B_LP_OFLCNT(R5)     ;RESET COUNTER
                18      BB      02FF    518             PUSHR   #^M<R3,R4>              ;SAVE REGISTERS
          54    05      9A      0301    519             MOVZBL  #MSG$_DEVOFFLIN,R4      ;SET UP MESSAGE TYPE
   00000000'GF      9E      0304    520             MOVAB   G^SYS$GL_OPRMBX,R3      ;ADDRESS TARGET MAILBOX
                53              030A
```

Figure 6-4 Location of Instruction in Driver Routine

6.3.4.1 **Stepping through the Routine** - The MOVB instruction is part of a routine that reads characters from a buffer and writes them out to the line printer. The routine executes the following steps for each character in the buffer:

- The driver gets a character from the buffer, moves it to the device data register (pointed to by R0 in this example), and autoincrements.
- The preceding step is repeated until the byte count is exhausted or the printer signals that it is NOT READY.
- If the printer gives the NOT READY signal, the driver waits for an interrupt from the printer.
- When the printer becomes READY, it interrupts the driver and the loop is resumed.

Examine the code to determine which variables control the loop. In this case, the byte count (BCNT) is the number of characters in the buffer. This value controls the number of times the loop is executed. (BCNT is set by a Function Decision Table (FDT) routine to the number of characters in the buffer.) The number of characters left to be printed is represented by the byte offset (BOFF).

Because the exception is an access violation, you can infer that R3 is outside the range of the buffer. It seems likely that the MOVB instruction has executed too many times, that is, a number of times greater than BCNT. To prove this theory, you must examine BOFF and BCNT.

6.3.4.2 **Checking the Values of Key Variables** - If you examine the code, you can see that R5 contains the address of the Unit Control Block (UCB) of the device that was active when the system failed. If you use the FORMAT command to display the contents of R5, SDA will display the values of BCNT and BOFF:

```
SDA> FORMAT @R5

8005D160        UCB$L_RQFL              800039A8
                UCB$L_FQFL
8005D164        UCB$L_RQBL              800039A8
                UCB$L_FQBL
8005D168        UCB$W_SIZE                  0080
8005D16A        UCB$B_TYPE                10
8005D16B        UCB$B_FIPL              08
                    .
                    .
                    .
8005D1C8        UCB$L_SVAPTE            80062720
8005D1CC        UCB$W_BOFF                  0795
8005D1CE        UCB$W_BCNT              006D
8005D1D0        UCB$B_ERTCNT                  00
8005D1D1        UCB$B_ERTMAX                00
8005D1D2        UCB$W_ERRCNT            0000
                    .
                    .
                    .
SDA>
```

If you have only one line printer in your system configuration, you need not use the FORMAT command. Issue the SHOW DEVICE command with device code LP as the parameter; since there is only one line printer device connected to the VAX-11 processor there is only one line printer UCB to display.

The output produced by the FORMAT @R5 command shows that BOFF contains a value greater than BCNT, when it should be the reverse. This means that an illegal value is being stored in BOFF. Thus, the value of BOFF is not the number of remaining characters in the buffer but some meaningless number that eventually causes the system to fail when it tries to access a null page (unreadable to all access modes).

6.3.4.3 **Identifying and Fixing the Defective Code** - Examine the line printer driver code again to locate all instructions that modify BOFF. The value changes in two important places.

1. Immediately after the driver detects that the printer is not ready.

2. When the wait for interrupt (WFIKPCH) routine timeout count of 12 seconds is exhausted. At this time, R1+1 is stored in BOFF.

The second modification to BOFF should not be made because R4 and R5 are the only registers that retain their values after the WFIKPCH routine is executed. To correct the problem, change the WFIKPCH line to transfer control to 50$ rather than 40$ (see Figure 6-5) if the timeout count expires.

```
                                029F     480 ;   START NEXT OUTPUT SEQUENCE
                                029F     481 ;
                                029F     482
50      54      02      C1      029F     483 10$:    ADDL3     #LP_DBR,R4,R0          ;CALCULATE ADDRESS OF DATA BUFFER REGISTER
  51      6C    A5      3C      02A3     484         MOVZWL    UCB$W_BOFF(R5),R1      ;GET NUMBER OF CHARACTERS REMAINING
52      8080    8F      B0      02A7     485         MOVW      #^X8080,R2             ;GET CONTROL REGISTER TEST MASK
                08      11      02AC     486         BRB       25$                    ;
        64      52      B3      02AE     487 20$:    BITW      R2,(R4)                ;PRINTER READY OR HAVE PAPER PROBLEM?
                08      15      02B1     488         BLEQ      30$                    ;IF LEQ NOT READY OR PAPER PROBLEM
        60      83      90      02B3     489         MOVB      (R3)+,(R0)             ;OUTPUT NEXT CHARACTER
          F5    51      F4      02B6     490 25$:    SOBGEQ    R1,20$                 ;ANY MORE CHARACTERS TO OUTPUT?
                70      11      02B9     491         BRB       70$                    ;
                                02BB     492
                                02BB     493 ;
                                02BB     494 ;   PRINTER IS NOT READY OR HAS PAPER PROBLEM
                                02BB     495 ;
                                02BB     496
                21      12      02BB     497 30$:    BNEQ      40$                    ;IF NEQ PAPER PROBLEM
          51    01      A1      02BD     498         ADDW3     #1,R1,UCB$W_BOFF(R5)   ;SAVE NUMBER OF CHARACTERS REMAINING
            6C  A5              02C0
                                02C2     499     DSBINT                             ;DISABLE INTERRUPTS
        64  40  8F      88      02C8     500         BISB      #^X40,LP_CSR(R4)       ;SET INTERRUPTS
                                02CC     501         WFIKPCH   40$,#12                ;WAIT FOR INTERRUPT
                                02D6     502         IOFORK                           ;CREATE A FORK PROCESS
                C1      11      02DC     503         BRB       10$                    ;
                                02DE     505 ;-----------
                                02DE     506 ;|  PRINTER HAS PAPER PROBLEM
                                02DE     508 ;|
            7A  A5      94      02DE     509 40$:    |CLRB     UCB$B_LP_OFLCNT(R5)    ;CLEAR OFFLINE COUNTER
          51    01      A1      02E1     510         |ADDW3    #1,R1,UCB$W_BOFF(R5)   ;SAVE NUMBER OF CHARACTERS REMAINING
            6C  A5              02E4                 |
                64  B4          02E6     511 50$:<---CLRW     LP_CSR(R4)             ;DISABLE PRINTER INTERRUPT
                                02E8     512         SETIPL    UCB$B_FIPL(R5)         ;LOWER TO FORK LEVEL
                64  B5          02EC     513         TSTW      LP_CSR(R4)             ;PRINTER STILL HAVE PAPER PROBLEM?
                AF  14          02EE      .1         BGTR      10$                    ;IF GTR NO
3E  58  A5      03  E0          02F0     515         BBS       #UCB$V_CANCEL,UCB$W_STS(R5),80$ ;IF SET, CANCEL I/O OPERATION
      01        0F  9D          02F5     516         ACBB      #15,#1,UCB$B_LP_OFLCNT(R5),80$  ;SKIP UNTIL TIMEOUT
    0017  7A    A5              02F8
            7A  A5      94      02FC     517         CLRB      UCB$B_LP_OFLCNT(R5)    ;RESET COUNTER
                18      BB      02FF     518         PUSHR     #^M<R3,R4>             ;SAVE REGISTERS
        54      05      9A      0301     519         MOVZBL    #MSG$_DEVOFFLIN,R4     ;SET UP MESSAGE TYPE
  00000000'GF           9E      0304     520         MOVAB     G^SYS$GL_OPRMBX,R3     ;ADDRESS TARGET MAILBOX
                53              030A
```

Figure 6-5 Location of Defective Code in Driver Routine

6.4 INDUCING A SYSTEM FAILURE

If the operating system is not performing well and you want to create a system dump file so that you can examine it later, you can induce a system failure by typing the following commands at the console:

```
$ CTRL/P

>>> HALT

>>> EXAMINE PSL

>>> DEPOSIT PC = -1

>>> DEPOSIT PSL = 1F0000

>>> CONTINUE
```

The system responds to the HALT command by displaying the PC; it responds to the EXAMINE PSL command by displaying the PSL. Immediately after you type this command sequence, the system signals a fatal bugcheck, writes information to SYSDUMP.DMP, shuts itself down, and automatically reboots.

Make a note of the PC and PSL displayed on the console before you perform the procedure outlined above. When you induce a system failure, the values you deposit into these registers destroy their previous contents, and you will need the pre-failure values contained in the PC and PSL when you begin to examine the system dump file, as described in Section 6.1.

CHAPTER 7

SDA ERROR MESSAGES

SDA error messages can be divided into messages that occur during SDA initialization and messages that occur during SDA operation. Messages that appear before SDA is initialized indicate problems encountered by SDA as it tries to run. SDA prints the message but does not execute. Messages that appear when SDA is operating concern problems encountered during command execution.

7.1 INITIALIZATION ERROR MESSAGES

The dump file contains no valid dump

This message appears if SDA cannot read the contents of the system dump file. The file may be unreadable because the data is bad or because the file is empty.

The dump only contains n pages of physical memory

This message occurs if the system dump file is not large enough to accommodate all of physical memory. The number of physical pages SDA can analyze is represented by n. To change the size of the system dump file, see Section 2.1.

Symbol symbol-name not found in SDA symbol table

This message appears if SDA cannot find a symbol in the SYS.STB file which is vital to its initialization.

7.2 OPERATIONAL ERROR MESSAGES

Invalid block type in specified block

This message appears if SDA is unable to identify the block type of a particular block. The invalid block type message most usually occurs when the FORMAT command tries to identify a block type using a byte offset. See the description of the FORMAT command in Chapter 5 for further information about byte offsets.

No "block-type" symbols found to format this block

This message appears if SDA cannot locate the symbols needed to format a block as a particular block type.

You may need to use the READ command to include the specific block type symbols in the SDA symbol table.

No such process

This message occurs if the process name specified in a SHOW PROCESS or SET PROCESS command refers to a process that does not exist.

Process swapped out

This message occurs if the process name specified in a SHOW PROCESS or SET PROCESS command represents a process that was swapped out of the balance set when the system failed.

Unable to access location location

This message indicates that SDA is unable to read a certain location. The inaccessible location may be an implied reference to memory made during the execution of an SDA command.

Unknown symbol symbol-name

This message occurs if SDA cannot identify a specified symbol.

Unknown type of GSD entry: GSD

This message occurs when SDA encounters a type of global symbol that it does not recognize, either in the SYS.STB file or in a file specified in the READ command. The type of global symbol definition GSD is represented by a byte. This message can occur during either initialization or operation of SDA, and usually means that the file being read has been corrupted.

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

VAX/VMS
System Dump Analyzer
Reference Manual
AA-J526A-TE

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
or

Please cut along this line.

— — — Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

— — Do Not Tear - Fold Here —