EDUCATIONAL
SERVICES

VAX/VMS
Training

VAX/VMS
Device Driver
Debugging

digital

EY-2278E-MG-0001

DEBUGGING

Prepared by Educational Services
of
Digital Equipment Corporation

First Edition, October 1984

Printed in U.S.A.

The manuscript for this book was created using DIGITAL Standard Runoff. Book production was done by Educational Services Development and Publishing in Nashua, NH.

CONTENTS

FIGURES

TABLES

EXAMPLES

DEBUGGING DRIVER HINTS (test program)

1. Must test ALL code threads
   All processor types

2. START with QIOW
   All function codes
   ensure illegal ones trapped

3. Use IOSB

4. Use QIOs to queue IRPs

5. BCANCEL
   image rundown
   ↑Y Control Y

6. Handle all characters

7. RMS/COPY

INTRODUCTION

Since device drivers run in kernel mode, at high interrupt priority levels, any error is considered serious, and may cause the system to crash. Before a driver is put into operation, it must be thoroughly tested.

VMS offers several tools to aid in driver debugging. Among these are debuggers (DELTA and XDELTA) and a system dump analyzer (SDA). This module discusses the various debugging tools available, and how those tools can be used. In addition, this module presents some general techniques/suggestions for identifying where problems occur in driver routines. Also, some general driver testing hints are presented.

OBJECTIVES

Upon completion of this module, given the lecture notes, you will be able to:

- Observe the operation of a driver by examining various driver-related data structures using the executive debugger (XDELTA) and the system dump analyzer (SDA).
- Determine the cause of a driver-related crash by using XDELTA commands, analyzing console bugcheck listings and using the system dump analyzer.
- Describe what must be done to fully test a driver.

RESOURCES

1. Guide to Writing a Device Driver for VAX/VMS
2. VAX-11 Software Installation Support Manual
3. VAX/VMS System Dump Analyzer Reference Manual

TOPICS

I. Debugging Tools Overview

II. Analyzing Console Bugcheck listings

III. XDELTA Command Summary

IV. CHMK - Accessing Kernel mode with DELTA

V. SDA Command Summary

VI. Driver Testing Hints

VAX/VMS DEBUGGING

Under the VAX/VMS operating system, there are several ways to "debug" or analyze a problem that is occurring in a program. The method of analysis depends on the "environment" the program is running under and the nature of the analysis you wish to do (i.e., monitoring only, stepping through a program). The table below describes the type of analysis, program environment, and suggested method of analysis (the debugger).

Table 7-1 Environment vs Debugging Tools

Problem/Environment	Method of Analysis
Program IPL=Ø, User mode Examine perprocess memory	VAX/VMS SYMBOLIC DEBUGGER (linked with image or included at run time, User mode only)
Program IPL=Ø, User to Kernel mode Examine process and system memory	DELTA DEBUGGER (linked with image or included at run time, User to Kernel mode) Nonsymbolic
Examine active system	System Dump Analyzer (SDA) Activated from DCL, can examine active system
Examine a Crash file	System Dump Analyzer (SDA) Activated from DCL
Program IPL > Ø	XDELTA DEBUGGER (linked with VMS, run from console terminal only) Nonsymbolic

BUGCHECK OUTPUT

While debugging a driver, the system will probably crash several times. The Bugcheck output appearing on the system console (and in the SDA dump file) often can be used to locate the instruction in the driver causing the crash. The bug (causing the identified instruction to crash the system) may actually occur prior to the instruction's execution. By locating the instruction, you will at least have a handle on how far an I/O request got before the crash occurred.

Examine the Bugcheck output below.

```
$ RUN WRITEV
BEFORE ASSIGNMENT MADE
ASSIGNMENT SUCCESSFUL

**** FATAL BUG CHECK, VERSION = SSRVEXCEPT, Unexpected system service exception

    CURRENT PROCESS = VIKP

    REGISTER DUMP

        R0 = 7FFEFE35
        R1 = 8000B4E7
        R2 = 80058F30
        R3 = 00000002
        R4 = 80059A50
        R5 = 80059240
        R6 = 7FFE1DC0
        R7 = 00000020
        R8 = 8005A39C
        R9 = FFFFFFD0
        R10= 00000030
        R11= 80059A8E
        AP = 7FFEEBA4
        FP = 7FFEEB8C
        SP = 7FFEEB84
        PC = 8000B4ED
        PSL= 00020000

    KERNEL/INTERRUPT STACK

Kernel Stack -> 7FFEEB8C  00000000
                7FFEEB90  00000000
                7FFEEB94  00000460
                7FFEEB98  7FFEEBE4
                7FFEEB9C  80000014
                7FFEEBA0  80004F16
                7FFEEBA4  00000002   <- CONDITION HANDLER
                7FFEEBA8  7FFEEBC4
                7FFEEBAC  7FFEEBB0
                7FFEEBB0  00000004
                7FFEEBB4  7FFEEBE4 FRAME
                7FFEEBB8  FFFFFFFE DEPTH      MECHANISM ARRAY
                7FFEEBBC  80058F31 R0
                7FFEEBC0  00000030 R1
                7FFEEBC4  00000005
                7FFEEBC8  0000000C CONDITION = SS$_ACCVIO
                7FFEEBCC  00000004 MODIFY VIOLATION*
                7FFEEBD0  0000002E FAULTING VIRTUAL ADDRESS   SIGNAL ARRAY
                7FFEEBD4  8005A402 PC
                7FFEEBD8  00C20008 PSL
                7FFEEBDC  80009298
                7FFEEBE0  00C00004
                7FFEEBE4  00000000
                7FFEEBE8  00000000
                7FFEEBEC  00000444
```

Example 7-1 Sample Bugcheck (Console Output)

Use the following information to analyze the previous crash.

Base Address of Driver

```
$MCR SYSGEN
SYSGEN> SHOW/DEVICES=PT
  Driver    Start    End    Dev   DDB     CRB     IDB     Unit  UCB
PTDRIVER  8005A320 8005A7E0
                          PTA 80055810 80055850 80055880
                                                    0 80059240
```

Part of driver.MAP file

```
PTDRIVER                                                27-AUG-1979 18:43

                          +-------------------------+
                          ! OBJECT MODULE SYNOPSIS  !
                          +-------------------------+

MODULE NAME     IDENT           BYTES     FILE
-----------     -----           -----     ----
PTDRIVER        V03              1206 DRA0:[VTK.BUG]PTDRIVER.OBJ;48        27
SYS             .STB;1              0 DRA0:[SYSEXE]SYS.STB;1               22

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

P-SECT NAME     MODULE(S)       BASE      END           LENGTH          ALIGN
-----------     ---------       ----      ---           ------          -----
$$$105_PROLOGUE                 00000000 00000050 00000051 (      81.) BYTE 0 NOI
                PTDRIVER        00000000 00000050 00000051 (      81.) BYTE 0
code ---------> $$$115_DRIVER   00000054 000004B8 00000465 (    1125.) LONG 2 NOI
                PTDRIVER        00000054 000004B8 00000465 (    1125.) LONG 2

                          +-------------------+
                          ! SYMBOLS BY NAME   !
                          +-------------------+

SYMBOL          VALUE           SYMBOL          VALUE           SYMBOL
------          -----           ------          -----           ------
DYN$C_CRB       00000005
DYN$C_DDB       00000006
DYN$C_DPT       0000001E
DYN$C_UCB       00000010
ERL$DEVICFRR    8000A3E8
ERL$DEVICTMO    8000A3EC
EXE$ABORTIO     800092FF
EXE$ALLOCBUF    8000983B
EXE$BUFFRQUOTA  800097F8
EXE$FINISHIOC   8000930C
EXE$IOFORK      800095C4
EXE$QIOACPPKT   80009328
EXE$QIODRVPKT   80009324
EXE$READCHK     80008D55
EXE$WRITECHK    80008D80
IOC$CANCELIO    8000A551
IOC$REQCOM      8000A6B3
IOC$RETURN      8000A869
IOC$WFIKPCH     8000A86A
PCB$L_PHD       00000064
PCB$W_BYTCNT    00000042
PR$_IPL         00000012
PRV$V_MOUNT     00000011
PT$DDT          00000054-R

        KEY FOR SPECIAL CHARACTERS ABOVE:
                +-------------------+
                ! *  - UNDEFINED    !
                ! U  - UNIVERSAL    !
                ! R  - RELOCATABLE  !
                ! WK - WEAK         !
                +-------------------+
%LINK-W-USRTFR, Image "PTDRIVER" has no user transfer address
```

ADD THIS TO START ADDRESS IN SYSGEN SHOW/DEVICES COMMAND SO THAT LINES IN THE LISTING FILE CAN BE MATCHED TO ADDRESSES IN THE BUGCHECK OUTPUT.

Example 7-1 Sample Bugcheck (Cont) (driver.MAP)

```
'E READER/PUNCH DRIVER        27-AUG-1979 18:42:48    VAX-11 Macro V02.30
ich FDT routine               27-AUG-1979 18:42:18    DBA0:[VIK.BUG]PTDRIVER.MAR;8

    006A 15550
    006A 15600 ;++
    006A 15650 ; PT_FDT_ROUTINE, Punch FDT routine
    006A 15700 ;
    006A 15750 ; Functional description:
    006A 15800 ;
    006A 15850 ;       This FDT routine makes all the standard accessibility check:
    006A 15900 ;       for buffered I/O operations, allocates a buffer from system
    006A 15950 ;       pool, and copies the user data to be punched to the system
    006A 16000 ;       buffer.  The standard IRP and PCB fields are updated.  In t
    006A 16025 ;       case of a WRITEOF function code, the IRP is queued to the A
    006A 16050 ;
    006A 16100 ; Inputs:
    006A 16150 ;
    006A 16200 ;       R0-R2    = scratch registers
    006A 16250 ;       R3       = address of the IRP (I/O request packet)
    006A 16300 ;       R4       = address of the PCB (process control block)
    006A 16350 ;       R5       = address of the UCB (unit control block)
    006A 16400 ;       R6       = address of the CCB (channel control block)
    006A 16450 ;       R7       = bit number of the I/O function code
    006A 16500 ;       R8       = address of the FDT table entry for this routine
    006A 16550 ;       R9-R11   = scratch registers
    006A 16600 ;       AP       = address of the 1st function dependent QIO paramet
    006A 16650 ;
    006A 16700 ; Outputs:
    006A 16750 ;
    006A 16800 ;       The routine must preserve all registers except R0-R2, and
    006A 16850 ;       R9-R11.
    006A 16900 ;
    006A 16950 ;--
    006A 17000
    006A 17050 PT_FDT_ROUTINE:                          ; Punch FDT routine ala CRD
D0  006A 17100          MOVL    P1(AP),R0               ; GET ADD. OF BUFFE
3C  006D 17150          MOVZWL  P2(AP),R1               ; GET LENGTH OF BUF
13  0071 17200          BEQL    10$                     ; ZERO LENGTH TRANS
16  0073 17250          JSB     G^EXE$WRITECHK          ; CHECK ACCESS OF U
    0079 17300                                          ; BUFFER -- NO RETU
    0079 17350                                          ; NO ACCESSIBILITY
    0079 17400 ;        PUSHR   #^M<R0,R3>              ; SAVE REGISTERS
C0  0079 17450          ADDL2   #12,R1                  ; ACCOUNT FOR OVERH
16  007C 17500          JSB     G^EXE$BUFFQUOTA         ; CHECK BUFFER QUO
E9  0082 17550          BLBC    R0,20$                  ; ABORT IF INSUFFI
16  0085 17600          JSB     G^EXE$ALLOCBUF          ; ALLOCATE SYSTEM
E9  008B 17650          BLBC    R0,20$                  ; ABORT ON FAILURE
    008E 17700 ;        POPR    #^M<R0,R3>              ; RESTORE REGISTER.
D0  008E 17750          MOVL    R2,IRP$L_SVAPTE(R3)     ; STORE RETURNED B
B0  0092 17800          MOVW    R1,IRP$W_BOFF(R3)       ; AND BYTE QUOTA C
A2  0096 17850          SUBW2   R1,PCB$W_BYTCNT(R4)     ; CHARGE PROCESS F
    009B 17900 ;        PUSHR   #^M<R3,R4,R5>           ; SAVE REG.'S FOR
C1  009B 17950          ADDL3   #12,R2,R3               ; FIND SYSTEM DATA
    009E
D0  009F 18000          MOVL    R3,(R2)+                ; SAVE ITS ADDRESS
D0  00A2 18050          MOVL    R0,(R2)                 ; SAVE USER BUFFER
C2  00A5 18100          SUBL2   #12,R1                  ; SET TRANSFER LEN
28  00A8 18150          MOVC3   R1,(R0),(R3)            ; COPY USER BUFFER
    00AB
    00AC 18200                                          ; SYSTEM DATA BUFF
```

Example 7-1 Sample Bugcheck (Cont)
(Driver Listing File)

The following steps are keyed to the sample Console Bugcheck Output in Example 7-1.

1. The AP points to a condition handler. (Note that sometimes the AP does not contain such a pointer; a signal and mechanism array may still appear on the top of the kernel/interrupt stack.)

2. The mechanism array holds the values of RØ and R1 at the time of the crash. (Other registers are in the Register Dump section.)

3. The signal array contains the reason for the crash (an SS$_error_code), and the PC and PSL at the time of the crash. The System Services Reference Manual contains descriptions of what the other parameters mean for various SS$_error codes. For an access violation, SS$_ACCVIO, the other information includes the type of access violation, and the virtual address referenced (which caused the fault).

   The most important piece of information is the PC. If the PC indicates an address in your driver (i.e., lies in the range defined by the SYSGEN SHOW/DEVICES command for your driver), the following procedures can be used.

4. Find your driver's starting address (using information from the SYSGEN SHOW/DEVICES command, which you typed before testing your driver).

5. Find the base address for the $$$115_DRIVER PSECT in your driver.MAP file.

6. Add the two numbers found in steps 4 and 5. This gives you the starting address of your driver code.

7. Subtract the number computed in step 6 from the PC found in step 3. This gives you the offset into your driver listing file (driver.LIS) where you will find the instruction causing the crash.

For example

A. Driver start address = 8005A320

\+ $$$115_DRIVER PSECT base = 54
8005A374 (Note: addition is done in hex.)

B. PC = 8005A402
\- A = 8005A374
8E(hex)

The instruction causing the system to crash is (see the driver listing file):

```
MOVL R2,IRP$L_SVAPTE(R3)
```

In this case, there is nothing wrong with the instruction itself. The error occurred because R3 had not been saved prior to the call to EXE$ALLOCBUF (which destroys the contents of R3). As a check, note that the register dump shows that R3 contains the value 2 (clearly not an IRP address). The offset IRP$L_SVAPTE has a value of 2C, so the faulting virtual address in the signal array also makes sense (2C + 2 = 2E).

When the faulting PC is a system address, but not a driver address, the system map (SYS$SYSTEM:SYS.MAP) must be consulted to see which module in the system caused the crash. From this information, it is sometimes possible to determine where in the driver the problem occurred (e.g., a crash caused by module SYSQIOREQ might indicate that FDT routines are exiting with registers R3-R5 modified, possibly following a MOVC instruction for which R3-R5 were not saved and restored). If examination of the system map reveals no clues, or the faulting PC is not part of the executive/driver, you must place breakpoints in your driver (using XDELTA), and step through your driver code until you identify the faulting instruction(s).

In general, XDELTA is used as a last resort. The prime reason for using XDELTA is to set breakpoints to identify how far an I/O request went before a crash occurred. SDA (and console bugcheck listings) are better tools for looking at data structures after crashes. XDELTA is used dynamically, while SDA is used statically.

DELTA/XDELTA

	DELTA	XDELTA
USAGE	User images	Operating System Drivers
Terminal used for control	Any TTY	Console only (OPA0:)
IPL	= 0	$\geq$ 0
How activated	Linked or included at run time	Included at boot time
Access Mode	All modes	Kernel mode only

Both debuggers are:

- Nonsymbolic
- Use same command syntax
- No visible prompt
- Error message is "Eh?"

Bootstrapping VMS with XDELTA

```
VAX-11/780

>>> DEPOSIT R3 0                ; Unit number 0 in R3

SYSBOOT> SET BUGREBOOT 0
SYSBOOT> CONTINUE

VAX-11/750

>>>B/7 DMA0                     ; For B/F, if F=7 include XDELTA
                                ; see DD manual for more information

SYSBOOT> SET BUGREBOOT 0
SYSBOOT> CONTINUE

VAX-11/730

>>>D/G/L 3  1                   ; Deposit unit number 1 in R3

>>>@DQAXDT                      ; Boot from DQA1 with XDELTA

SYSBOOT> SET BUGREBOOT
SYSBOOT> CONTINUE
```

Bootstrapping VMS with XDELTA (Cont)

```
  P
>>>HALT

          HALTED AT 80008462

>>>@DB0XDT    DBzXDT for RM03 or RP06
             DMzXDT for RK07

!

!            DB0 CONVERSATIONAL/DEBUG BOOT COMMAND FILE - DB0XDT

!            BOOT FROM DB0 AND STOP IN SYSBOOT AT ALTER PARAMETERS

     .
     .
     .

DEPOSIT R5 7    ! SOFTWARE BOOT FLAGS (CONVERSATIONAL/DEBUG)

     .
     .
     .

SYSBOOT> SET BUGREBOOT 0

SYSBOOT> CONT

    VAX/VMS Version 3.0 27-MAY-82 10:30

1 BRK AT 800017D  ;P
```

Invoking XDELTA

To invoke XDELTA, enter the following commands at the console in the order given.

Command	Purpose
<control-P>	Activates XDELTA at the console
HALT	Halts the processor (780 only)
D/I 14 5	Deposits "5" into internal register 14 (software interrupt request register) to generate an interrupt at IPL 5.
CONTINUE	Resumes system execution

After issuing the CONTINUE command, the following happens:

- The IPL 5 interrupt occurs when the processor IPL drops below 5.
- XDELTA gains control at IPL 5, raises IPL to 31, and suspends execution at a breakpoint.
- At the breakpoint, XDELTA can accept an XDELTA command entered at the console and act accordingly.

Example:

```
>>>H

        HALTED AT 800082EE
>>>D/I 14 5

>>>CONTINUE

1 BRK AT 8000B17D

;P
```

Notice that the procedure described on the previous page works only for driver FDT routines running at IPL 2. For other driver routines, running at fork IPL or above, the XDELTA interrupt will not be granted, since XDELTA is invoked at IPL 5. To invoke XDELTA at elevated IPLs, use the following statement in your driver:

```
JSB G^INI$BRK
```

This may be used where you want to set XDELTA breakpoints in your driver (e.g., at the beginning of each driver routine, to isolate which routine is causing a system crash).

DELTA Debugger

To use the DELTA debugger, assemble and link a program in the following fashion:

1. $ MACRO prog-name + SYS$LIBRARY:LIB/LIB
2. $ LINK/DEBUG prog-name, SYS$SYSTEM:SYS.STB/SELECT
3. $ DEFINE LIB$DEBUG DELTA
4. $ RUN prog-name

Steps:

1. Assemble the program allowing system macros to be defined (SYS$LIBRARY:LIB/LIB)
2. Link the program with a debugger and resolve any system symbols (SYS$SYSTEM:SYS.STB)
3. Define the debugger used to be DELTA
4. Activate the program, mapping in DELTA

CHMK PROGRAM

It is often convenient to observe data structures changing dynamically. One way to gain access to kernel mode data structures is to run the CHMK program (see Example 7-2). This program allows any privileged process (with CMKRNL privilege) to change mode to kernel, and enter XDELTA commands (e.g., to look at system data structures). Extreme caution should be exercised so that data structures are not modified, since such modification could lead to a system crash.

Perform the following steps to use the CHMK program:

1. Assemble and link CHMK
2. Run the CHMK program
3. Enter a breakpoint in the program and tell it to proceed

Corresponding Commands:

1. $ MACRO CHMK and $ LINK CHMK
2. $ RUN CHMK.EXE
3. 215;B;P

   Note on Step 3, no prompt from DELTA is given.

After you receive the "stopped at breakpoint" message, you are in kernel mode, and may proceed to examine system data structures. To leave the program, type ';P', followed by 'EXIT' (if you just type EXIT, you will be logged off, since kernel mode exit implies process deletion).

```
GO:     .WORD  Ø                    ; null entry mask
        $CMKRNL_S  ROUTIN = 1Ø$     ; enter kernel mode
        RET                         ; all done
1Ø$:    .WORD  Ø                    ; null entry mask
        NOP                         ; where BPT instruction
        NOP                         ; is placed, by 215;B
                                    ; issued by user
        MOVZBL #SS$_NORMAL,RØ       ; return success status
        RET                         ; all done
        .END GO
```

Example 7-2 The CHMK program, which must be run with DELTA

Debugging DELTA/XDELTA

Table 7-2 DELTA/XDELTA Functions and Commands

Function	Command	Example
Display contents of given address	address/	GA88/00060034
Display contents as instruction	address!	
Replace contents of given address	addr/contents new	GA88/00060034 GA88 GA88/00060034 'A' (replace as ASCII)
Display contents of previous location	<ESC>	80000A88/80000BE4 <ESC> 80000A84/00000000
Display contents of next location	addr/contents <LF> addr'/contents	80000004/8FBC0FFC 80000005/50E9002C
Display range of locations	addr,addr'/contents	G4,GC/8FBC0FFC 80000008/50E9002C 8000000C/00000400
Display indirect	<TAB> or /	80000A88/80000BE4 <TAB> 80000BE4/80000078 80000A88/80000BE4/80000078
Single step command	S	1 brk at 8000B17D S
Step Over Subroutine	O	8000B17E/9A0FBB05
Set Breakpoint	addr;N;B <ret> (N is a number 2-8)	800055F6;2;B
Display Breakpoint	;B	;B 1 8000B17D 2 800055F6

Table 7-2 **DELTA/XDELTA Functions and Commands (Cont)**

Function	Command	Example
Clear Breakpoint	Ø;N;B <ret>	Ø;2;B
Proceed from Breakpoint	;P	;P
Set Base Register	'value',N,X	80000000,0,X
Display Base Register	Xn <ret> or Xn=	XØ ØØØØØØØ3 XØ=ØØØØØØØ3
Display General Register	Rn/ (n is in Hex)	RØ/ØØØØØØØ3
Show Value	expression=	1+2+3+4=ØØØØØØØA (+,-,*,%(divide))
Executing stored command strings	addr;E <ret>	8ØØØØE58;E
Change display mode	[B [W [L [" [I	byte width word width longword width ASCII display Instruction display

Debugging SDA

To activate SDA:

```
$ ANALYZE/qualifier
```

What you use as a qualifier will determine what you will be examining.

Table 7-3 ANALYZE Qualifiers

To Examine	Qualifier	Comment
Current System	/SYSTEM	CMKRNL privilege needed
System Dump file or Other Dump file	/CRASH_DUMP	Read access to file needed

SDA Functions

- examine locations by address or symbol
- displays process/system data
- formats and displays data known data structures
- assigns values to symbols as requested

Command Format

```
command          [parameter]        [/qualifier]
```

Table 7-4 SDA Commands

Information	Command
Provides help using SDA	HELP
Displays specific data/information	SHOW
Format and display data structures	FORMAT
Display contents of location(s)	EXAMINE

Manipulation	Command
Preserve second copy of dump file	COPY
Create and define symbols	DEFINE
Perform computations	EVALUATE
Set/Reset defaults	SET
Define other VMS symbols	READ
Repeat last command	REPEAT or <ESC>

SDA> SHOW DEVICE TTC7

DDB LIST

```
SDA> SHOW DEVICE  TTC7

                                DDB list
                                --------

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

         8009DA20      TTC               DZDRIVER   80078BF0  1A46
Controller: TTC
---------------

Device Data Block (DDB):
        DDT address                  80078DE8
        First UCB address            8007E450
        CRB address                  8009DA80

Channel Request Block (CRB):
        UCB reference count                 8
        Channel allocation mask            00
        Secondary CRB address        00000000
        IDB address                  8009DAE0
        Controller init. routine     8007A27C
        Unit init. routine           8007A2FF
        Unit start routine           8007A406
        Unit disconnect routine      8007A35F

Interrupt Dispatch Block (IDB):
        CSR address                  80017A68
         Owner UCB address            00000000
        Number of units                     8
        ADP address                  80077200

Driver Dispatch Table (DDT)::
        Start I/O routine            00000498
        Unsol. interrupt routine     8000981B
        Function Decision table      00000020
        Cancel I/O routine           000010E8
        Resister dump routine        8000981B
        Diagnostic buffer size           0000
        Error buffer size                0000
TTC7
----

        UCB address:      8007E990
        Device status:        0110   online,bsy
        Characteristics:  0C040007

IRP address     800940C0     Device class          42     SVPN          0000000C
CRB address     8009DA80     Device type           60     SVAPTE        800838FC
UCB address     00000000     DEVBUFSIZ             80     BOFF              0014
FQFL            800024C8     DEVDEPEND      181013B0     BCNT              000C
FQBL            800024CC8     DEVSTS              0001     ERTCNT            1?
Fork IPL               8     Device IPL            21     ERTMAX
Fork PC         8007A019     Reference count        1     ERRCNT            0
Fork R3         0000000D     Operation count     1425     Owner UIC     [000,000]
Fork R4         0000001A     AMB address     00000000     PID           0002003C

                                I/O request queue
                                -----------------

CHF     IRP       PID  MODE CHAN  FUNC    WCB     EFN    AST     IOSB    STATUS
```

DRIVER TESTING HINTS

1. You will need to write one or more programs to test your driver. Often, it is convenient to write test programs in higher-level languages (such as FORTRAN). Your test programs must:

   - test all function-codes supported by your driver.
   - test all paths through your driver (including timeout, powerfail, and cancel I/O).
   - test all conditions relevant to your driver (e.g., error conditions, multiple program access, and high-speed access).

   If possible, test your test programs on some other device (terminal) by changing the $ASSIGN system service to assign a channel to a terminal, rather than your device, in the DEVNAM argument. That way, you can debug only your driver, not both your driver and your test program. As always, it is good programming practice to document your test programs.

2. Be systematic in your approach. Develop FDT routines first. Test them by temporarily changing

   ```
   JMP G^EXE$QIODRVPKT
   ```

   to

   ```
   JMP G^EXE$FINISHIO
   ```

   In this way, you can develop sections of your driver at a time.

3. You may want to include a mask at the end of your Function Decision Table to transfer control to an error routine (to prevent FDT dispatching errors). The mask should specify all legal function codes.

4. If you plan on using the XDELTA stored command feature, leave room in your UCB for stored command strings.

5. Place a (commented) JSB G^INI$BRK at the start of each driver routine, so that if you have to use XDELTA, you can quickly locate which routine is causing system crashes.

6. To test for powerfail recovery, turn off the power on the front cabinet by using the key.

7. Observe your device while testing for clues as to how far an I/O request got before the system crashed. Was the device powered-up?, did the device initiate some hardware activity?, did the system crash after the I/O completed?, etc.

   If, while observing your device, your driver causes the device to behave incorrectly, or to loop (e.g., never stop punching tape), you may have to type control-P on the console (followed by HALT) to halt the processor. Or, if looping at too high an IPL for the console control-P interrupt to be granted, turn the power off using the key on the front cabinet. To force a dump file to be written, you can crash the system on the system console as follows:

   type control-P
   ```
   >>> HALT
   >>> @CRASH
   ```

8. Specify a real CSR to SYSGEN when you CONNECT a unit unless you use the /NOADAPTER qualifier. If your hardware is not yet installed, specify an existing CSR (such as the lineprinter CSR, which will not be affected by a test to see if it is there). Also, a vector address that is not currently in use must be specified, even in the /NOADAPTER case.

9. When using XDELTA, set up a base register to point to the start address of your driver (plus the $$$115_DRIVER PSECT base address), so that you can reference instructions as offsets from the base register (corresponding to offsets in your driver listing file).

10. When calling a DIGITAL representative (field service, software support, etc.) for assistance in device driver or device hardware problems, it is always helpful to have a driver testing program available. Such a program can be used as a "first pass" diagnostic for your device driver or device hardware.

Common Problems and Reasons When Writing Device Drivers

Problem	Reason
Access violation	1. UCB$L...(R5) Empty register
	2. Didn't save/restore registers before/after MOVC
NULL process "caught" in IOPOST routine	Didn't set buffered I/O bit IOPOST, tried to unlock "bad" pages
Process requesting I/O in MWAIT state (AST wait)	Didn't IOFORK before REQCOM
Random behavior not reproducible	1. Using registers other than R3 R4 before/after IOFORK
	2. Leaving data on stack when calling system macros (causing fork process to wait)

APPENDIX
SAMPLE STACK OUTPUTS

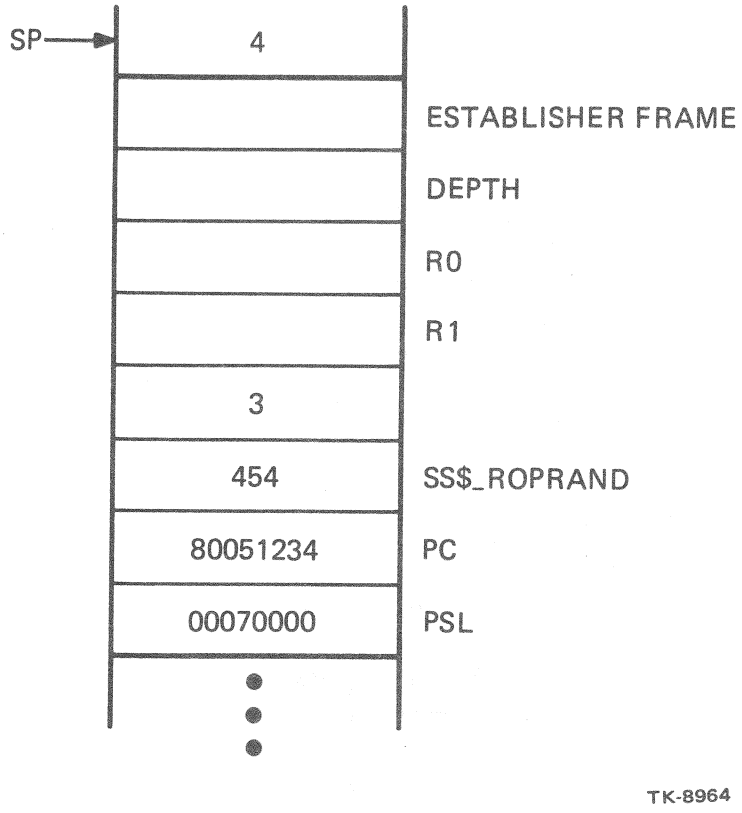

Figure 7-1 Reserved Operand

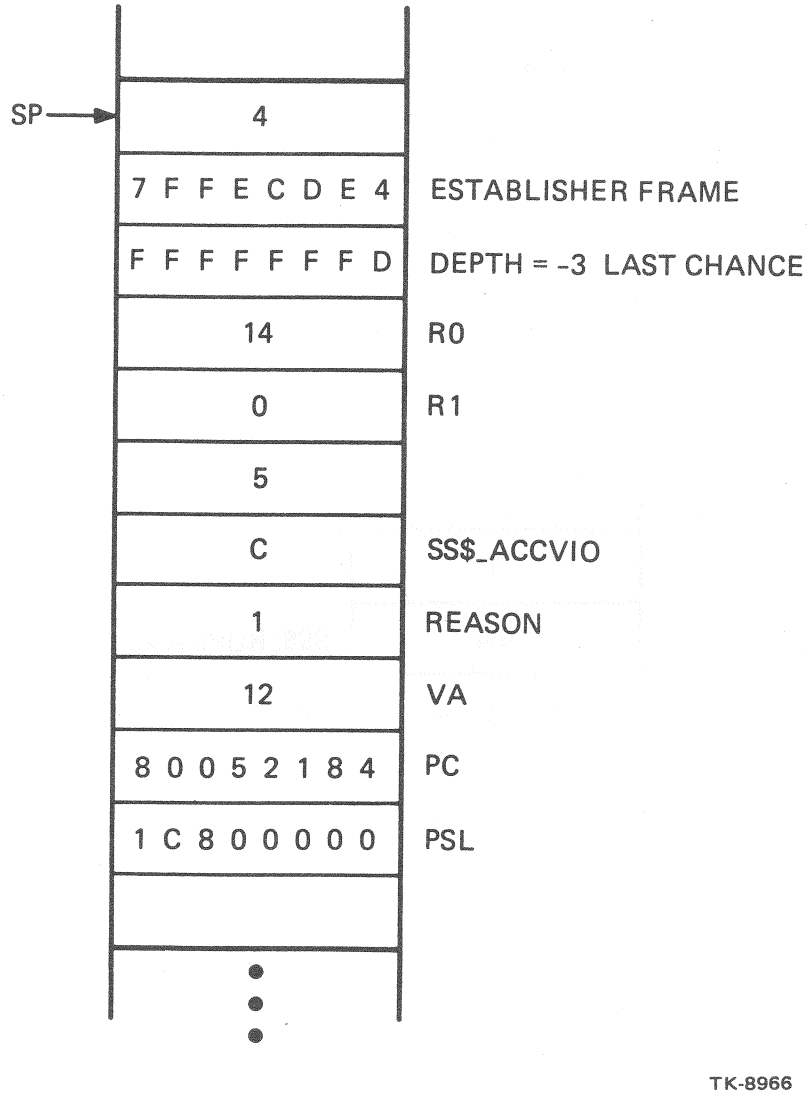

Figure 7-2 Access Violation

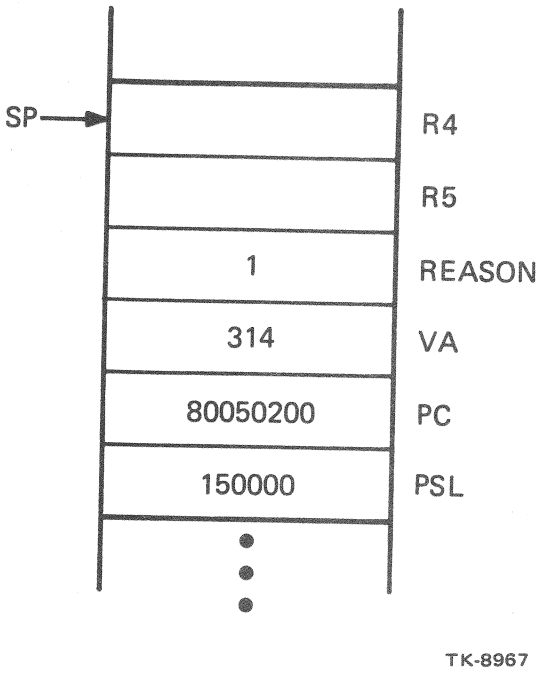

Figure 7-3 Pagefault Above IPL2

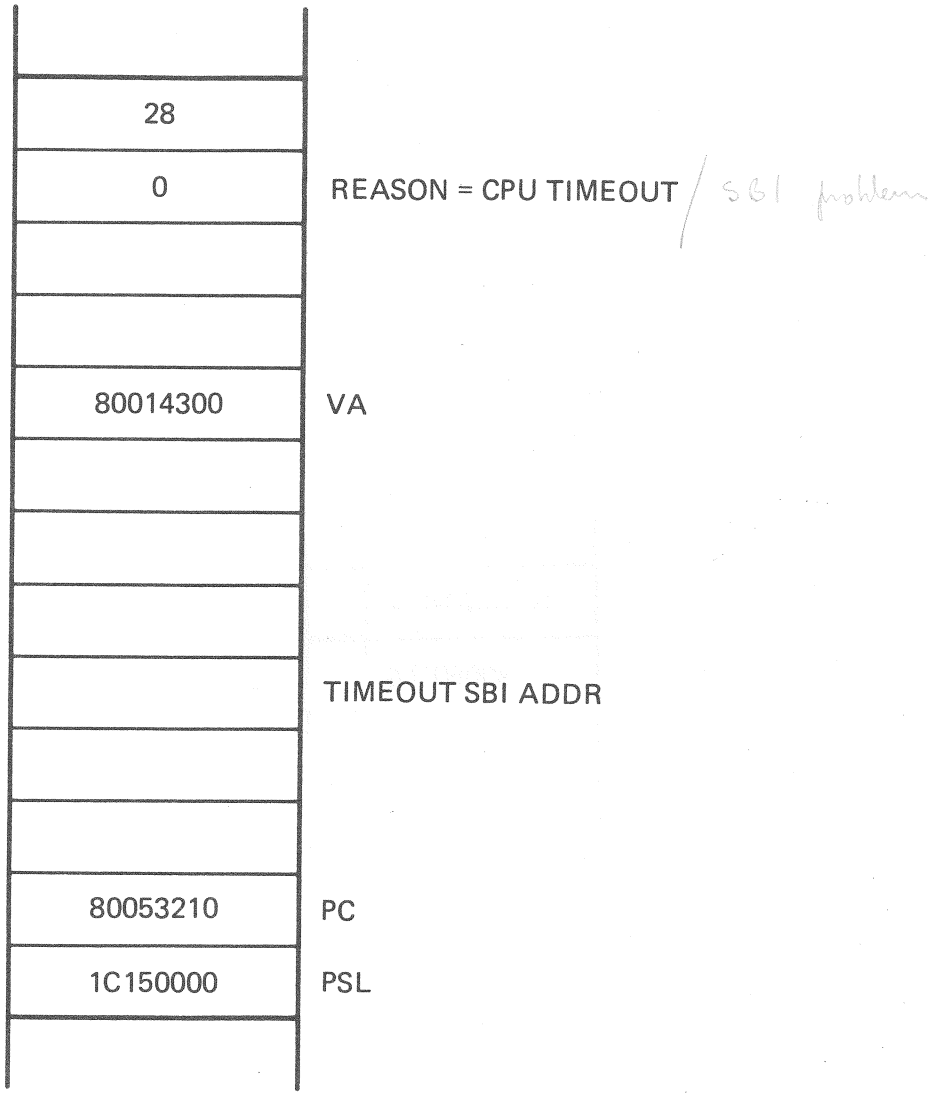

Figure 7-4 Machine Check in Kernel

digital

EY-2278E-MG-0001
Printed in U.S.A.