```
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
```

FILEID**DBGSTART

```
DDDDDDDD    BBBBBBBB      GGGGGGGG    SSSSSSSS  TTTTTTTTTT    AAAAAA    RRRRRRRR    TTTTTTTTTT
DDDDDDDD    BBBBBBBB      GGGGGGGG    SSSSSSSS  TTTTTTTTTT    AAAAAA    RRRRRRRR    TTTTTTTTTT
DD      DD  BB      BB  GG          SS              TT      AA      AA  RR      RR      TT
DD      DD  BB      BB  GG          SS              TT      AA      AA  RR      RR      TT
DD      DD  BB      BB  GG          SS              TT      AA      AA  RR      RR      TT
DD      DD  BB      BB  GG          SS              TT      AA      AA  RR      RR      TT
DD      DD  BBBBBBBB    GG            SSSSSS        TT      AA      AA  RRRRRRRR        TT
DD      DD  BBBBBBBB    GG            SSSSSS        TT      AA      AA  RRRRRRRR        TT
DD      DD  BB      BB  GG  GGGGGG          SS      TT      AAAAAAAAAA  RR  RR          TT
DD      DD  BB      BB  GG  GGGGGG          SS      TT      AAAAAAAAAA  RR  RR          TT
DD      DD  BB      BB  GG      GG          SS      TT      AA      AA  RR    RR        TT          ....
DD      DD  BB      BB  GG      GG          SS      TT      AA      AA  RR    RR        TT          ....
DDDDDDDD    BBBBBBBB      GGGGGG    SSSSSSSS        TT      AA      AA  RR      RR      TT          ....
DDDDDDDD    BBBBBBBB      GGGGGG    SSSSSSSS        TT      AA      AA  RR      RR      TT          ....

LL            IIIIII        SSSSSSSS
LL            IIIIII        SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

Table of contents

```
0000            .TITLE  DBGSTART
0000            .IDENT  'V04-000'
0000
0000
0000    ;****************************************************************************
0000    ;*                                                                          *
0000    ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000    ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000    ;*  ALL RIGHTS RESERVED.                                                    *
0000    ;*                                                                          *
0000    ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000    ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000    ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000    ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000    ;*  TRANSFERRED.                                                            *
0000    ;*                                                                          *
0000    ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000    ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    ;*  CORPORATION.                                                            *
0000    ;*                                                                          *
0000    ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000    ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    ;*                                                                          *
0000    ;*                                                                          *
0000    ;****************************************************************************
0000
0000    ;++
0000    ; FACILITY: DEBUG
0000    ;
0000    ; ABSTRACT:
0000    ;       Start-up module for DEBUG facility.
0000    ;
0000    ; ENVIRONMENT:
0000    ;       VAX/VMS User mode : Mapped and entered initially from the CLI. From
0000    ;       then on DEBUG runs as a (very complex) exception handler.
0000    ;
0000    ; VERSION:      4.00
0000    ;
```

```
0000   ; REVISION HISTORY:
0000   ;
0000   ;
0000   ;       MODIFIED BY:
0000   ;               John Francis, 30 November 1981
0000   ;               V. Holt, 2 June 1982
0000   ;               P. Sager, Oct. 1982
0000   ;               P. Sager, Aug. 1983
0000   ;+
0000   ; 1.01  05-OCT-78       DAR     Fixed bug in the way WINDOW_2 returns to VMS.
0000   ; 1.02  20-OCT-78       MCC     Changed size of writable storage to be
0000   ;                               calculated at link time.
0000   ; 1.03  24-OCT-78       DAR     PROBER transfer address before initializing
0000   ;                               the PSL's IV and DV bits.
0000   ; 1.04  10-NOV-78       DAR     Added routine DBG$THREAD_BPT for threaded BPT's.
0000   ; 1.05  07-DEC-78       DAR     Fixed exit handler to exit if SS$_CLIFRCEXT.
0000   ; 1.06  18-DEC-78       DAR     Added global label to threaded breakpoint routine
0000   ;                               to enable "GO %line x" to work in threaded code.
0000   ; 1.07  23-MAR-79       MCC     Modified dbg$out_message to write error messages
0000   ;                               to the LOG file
0000   ; 1.08   6-AUG-79       MCC     Modified dbg$final_handl to check for error msgs
0000   ;                               from shared msg file, before signaling "internal
0000   ;                               DEBUG coding error"
0000   ; 1.09   5-SEP-79       MCC     Made a fix to toggle system service failure
0000   ;                               mode when user program has set it and DEBUG
0000   ;                               gets control
0000   ; 1.10   7-APR-81       JF      Signal DBG$_SUPERDEBUG if testable debugger
0000   ;                               and any unexpected errors are encountered
0000   ; 3.00  30-NOV-81       JF      Tidy up entry and exit sequences to fix bugs
0000   ;                               with AST's and user termination handlers.
0000   ; 3.10   2-Jun-82       VJH     Removed all references to DBG$FAO_PUT and
0000   ;                               DBG$OUT_PUT, as these routines are now obsolete.
0000   ;                               Replaced them with calls to DBG$PRINT and
0000   ;                               DBG$NEWLINE, respectively.
0000   ; 3.B0   12-Oct-82      PS      Added some code to DBG$PSEUDO_HANDLER to
0000   ;                               release all the memory blocks for CALL command
0000   ; 3.B1  18-Jan-83       JF      Added DBG$GV_CONTROL state vector and modified
0000   ;                               handling of 'SS$_DEBUG' exception.
0000   ; 4.0   31-Aug-83       PS      Fixed a read error infinite loop reported by
0000   ;                               user through SPR.  Set up a count in
000C   ;                               DBG$COMMAND_PROC (DBGEXC), If we get bad
0000   ;                               status from $GET/Key Pad input, after 20
0000   ;                               tries, we'll force the DEBUG to take EXIT.
0000   ;                               (See code added in DBG$FINAL_HANDL, label
0000   ;                                FINAL_7).
0000   ; 4.0   01-Feb-84       PS      Added SSI for watch pointing
0000   ;--
```

```
          0000          .SBTTL  DECLARATIONS
          0000
          0000  ; *****SSI
          0000  ; SSI_USS is a privileged shareable image to set up system service
          0000  ; interception for watch pointing.  It must be installed in SYS$LIBRARY.
          0000  ; DEBUG can link with/without SSI_USS on VMS V4, indicated by link flag
          0000  ; DBG$GL_3B_SYSTEM and DBG$GL_SETSSI.  DBG$GL_3B_SYSTEM must be set to 1
          0000  ; to indicate VMS V4 system, DBG$GL_SETSSI must be set to 1 to indicate to
          0000  ; link with SSI_USS, set to 0 to indicate to link without SSI_USS.  On VMS
          0000  ; V3 system, DBG$GL_3B_SYSTEM must be set to 0, there is no SSI_USS active.
          0000  ; In this way, DEBUG will work both ways on VMS V4 and VMS V3.  Declare
          0000  ; SSI_USS to be weak reference, so that we won't get linker warnings at
          0000  ; link time.
          0000          .WEAK   SSI_USSK
          0000          .WEAK   SSI_USSU
          0000          .EXTRN  DBG$GL_3B_SYSTEM,DBG$GL_SETSSI
          0000          .EXTRN  DBG$GL_INPRAB,DBG$GL_OUTPRAB,DBG$RUNFRAME,DBG$GV_CONTROL
          0000          .EXTRN  DBG$GB_CALL_NORMAL_RET,DBG$SCR_SCREEN_TERM,PRT$C_UW
          0000          .EXTRN  DBG$END_OF_LINE,DBG$EXC_HANDLER,DBG$PRINT,DBG$RST_INIT
          0000          .EXTRN  DBG$REL_MEMORY,DBG$INIT_DEBUG,DBG$OUT_NUM_VAL,DBG$NEWLINE
          0000          .EXTRN  DBG$PUTMSG,DBG$INS_OPCODES,LIB$SIGNAL,SYS$DCLEXH,SYS$EXIT
          0000          .EXTRN  SYS$GETMSG,SYS$PUT,SYS$SETAST,SYS$SETPRT,SYS$UNWIND
          0000          .EXTRN  DBG$GL_LOGRAB,DBG$GB_DEF_OUT,DBG$FLUSHBUF,DBG$GB_UNHANDLED_EXC
          0000          .EXTRN  EVENT$PAGE_QUEUE
          0000
          0000  ;
          0000  ; invoke data definitions
          0000  ;
          0000          $CHFDEF                         ; Condition handler mnemonics
          0000          $CLIDEF                         ; CLI status bit definitions
          0000          $DBGDEF                         ; Debug definitions
          0000          $IFDDEF                         ; Image file definitions
          0000          $PSLDEF                         ; Processor Status Longword bits
          0000          $RABDEF                         ; RAB definitions
          0000          $SFDEF                          ; Stack Frame offset definitions
          0000          $SHRDEF                         ; Shared error messages
          0000          $SSDEF                          ; System error codes
          0000          $STSDEF                         ; Status code fields
          0000  ;
          0000  ; Equated symbols
          0000  ;
00000100  0000          buf_siz         = 256           ; length of getmsg, FAO, and $PUT buffers
00000002  0000          dbg_facility    = 2             ; DEBUG facility code.
```

```
          0000  ; *****SSI
          0000  ; SSI_USS can be called by the user, or by the DBG (TDBG), or by the SDBG.
          0000  ; Each level (user, DBG/TDBG, SDBG) delcares a interception routine which
          0000  ; runs at a priority (user - priority 1, 2, DBG/TDBG - priority 3, SDBG -
          0000  ; priority 4).  This vector is used to indicate which priority is active
          0000  ; at the moment.
          0000  ;
          0000  ;       Definitions of bits in DBG$GV_SSI_CONTROL running state vector
          0000  ;
00000000  0000          dbg$v_ssi_routine_1 = 0          ; Set if user declared prio. 1 routine
          0000                                           ;  is running
00000001  0000          dbg$m_ssi_routine_1 = 1@dbg$v_ssi_routine_1
          0000
00000001  0000          dbg$v_ssi_routine_2 = 1          ; Set if user declared prio. 2 routine
          0000                                           ;  is running
00000002  0000          dbg$m_ssi_routine_2 = 1@dbg$v_ssi_routine_2
          0000
00000002  0000          dbg$v_ssi_routine_3 = 2          ; Set if user declared prio. 3 routine
          0000                                           ;  is running
00000004  0000          dbg$m_ssi_routine_3 = 1@dbg$v_ssi_routine_3
          0000
00000003  0000          dbg$v_ssi_routine_4 = 3          ; Set if user declared prio. 4 routine
          0000                                           ;  is running
00000008  0000          dbg$m_ssi_routine_4 = 1@dbg$v_ssi_routine_4
          0000
          0000
          0000  ;       Definitions of bits in DBG$GV_CONTROL state vector
          0000  ;       ****    THESE MUST MATCH DEFINITIONS IN DBGLIB    ****
          0000
00000000  0000          dbg$v_control_tdbg =  0          ; Set if this is a testable DEBUG
00000001  0000          dbg$m_control_tdbg =  1@dbg$v_control_tdbg
          0000
00000001  0000          dbg$v_control_sdbg =  1          ; Set if this is SUPERDEBUG
00000002  0000          dbg$m_control_sdbg =  1@dbg$v_control_sdbg
          0000
00000002  0000          dbg$v_control_kdbg =  2          ;
00000004  0000          dbg$m_control_kdbg =  1@dbg$v_control_kdbg
          0000
00000003  0000          dbg$v_control_urun =  3          ; Set if user program has been run
00000008  0000          dbg$m_control_urun =  1@dbg$v_control_urun
          0000
00000004  0000          dbg$v_control_exit =  4          ; Set if DEBUG is about to EXIT
00000010  0000          dbg$m_control_exit =  1@dbg$v_control_exit
          0000
00000005  0000          dbg$v_control_fail =  5          ; Set by DEBUG internal errors
00000020  0000          dbg$m_control_fail =  1@dbg$v_control_fail
          0000
00000006  0000          dbg$v_control_done =  6          ; Set if user program completed
00000040  0000          dbg$m_control_done =  1@dbg$v_control_done
          0000
00000007  0000          dbg$v_control_allocate  =  7     ; Set if OK to allocate more memory
          0000                                           ;       (e.g., SET MODULE/ALLOCATE)
00000080  0000          dbg$m_control_allocate = 1@dbg$v_control_allocate
          0000
00000008  0000          dbg$v_control_user =  8          ; Set if user program is running
00000100  0000          dbg$m_control_user =  1@dbg$v_control_user
          0000
```

```
00000009  0000   dbg$v_control_stop =  9          ; Set by ^Y,DEBUG sequence
00000200  0000   dbg$m_control_stop =  1@dbg$v_control_stop
          0000
0000000A  0000   dbg$v_control_tbit =  10         ; Set during un-interruptable TBITs
00000400  0000   dbg$m_control_tbit =  1@dbg$v_control_tbit
          0000
0000000B  0000   dbg$v_control_screen = 11        ; Set if screen displays must be updated
          0000                                    ;          because user program has run
00000800  0000   dbg$m_control_screen = 1@dbg$v_control_screen
          0000
0000000C  0000   dbg$v_control_version_4 = 12     ; Set if VMS 3B or 4.0 is running
00001000  0000   dbg$m_control_version_4 = 1@dbg$v_control_version_4
          0000
          0000
```

```
                0000  ;+
                0000  ; Special vector that contains the beginning and end addresses of DEBUG's
                0000  ; writable storage area.  The third address in the vector is the transfer
                0000  ; address of DEBUG that is in this module.
                0000  ;-
            00000000          .PSECT  DBG$ABS_ZERO LONG, PIC, SHR, NOWRT, EXE
                0000  virtual_zero:
00000000        0000          .LONG   0          ;.ADDRESS writable_stor ; lowest writable location
00000000        0004          .LONG   0          ;.ADDRESS end_write_stor; highest writable location
00000002'       0008          .LONG   <beginhere+2> - virtual_zero     ; start address of mapped DEBUG
                000C
                000C
            00000000          .PSECT  DBG$GLOBAL LONG, PIC, NOSHR, NOEXE
                0000
                0000  writable_stor:                                   ; Define lowest writable address
00000004        0000  dbg$gl_runframe::       .BLKL   1
                0004
            00000000          .PSECT  ZZZ$ZZZZZZ LONG, PIC, NOSHR, NOEXE
                0000
                0000  end_write_stor:                                  ; Define highest writable address
```

```
          00000000        .PSECT  DBG$SSI  PIC, NOSHR, NOEXE, PAGE
              0000        .ALIGN  PAGE
              0000  ; *****SSI
              0000  ; Allocate one page of storage for the following variables.  These variables
              0000  ; are DEBUG variables, pass in SSI_USS as parameters, values are returned
              0000  ; from SSI_USS.  If we set the watch point on these variables (page is
              0000  ; write protected) will cause SSI_USS to fail (kernal mode accvio, PROBEW).
              0000  ; Any DEBUG variables is on the same page will be affected.  So, we put
              0000  ; these variables on a page all by themself, away from the other DEBUG
              0000  ; variables.  Restriction has to be set: NO WATCH POINT CAN BE SET ON THESE
              0000  ; VARIABLES.
              0000  ;
              0000 SSI_VAR_BEG::
 00000200     0000 DATA:   .BLKB    512
              0200 SSI_VAR_END::
              0200
 00000000     0200 DBG_ROUTINE_ID==SSI_VAR_BEG                   ; An ID returned from SSI_USS to
              0200                                               ;   indicate TDBG/DBG has declared its
              0200                                               ;   interception routine
 00000004     0200 SDBG_ROUTINE_ID==SSI_VAR_BEG+4                ; An ID returned from SSI_USS to
              0200                                               ;   indicate SDBG has declared its
              0200                                               ;   interception routine
 00000008     0200 SAVE_STATE==SSI_VAR_BEG+8                     ; An important communication state
              0200                                               ;   variable to keep the interception
              0200                                               ;   flow going between all the levels
 0000000C     0200 DUMMY=SSI_VAR_BEG+12                          ; Dummy arg.
              0200
              0200 ;
              0200 ; OWN STORAGE
              0200 ;
          00000000        .PSECT  DBG$OWN LONG, PIC, NOSHR, NOEXE
              0000
              0000 ; *****SSI
              0000 ; Variables are used in DEBUG to make SSI work.
              0000 DBG_SSI_CNT::                         ; A count to keep track how many SSV are seen
 00000000     0000        .LONG   0                      ;  by DBG/TDBG, or SDBG
              0004 DBG_ONCE_ONLY_CNT::                   ; Debug is highly re-entriant, is also hard
 00000000     0004        .LONG   0                      ;  to identify re-entriant point.  So we use
              0008                                       ;  this count to keep track of the entry point
              0008 DBG_SETUP::                           ; An important state flag to control SSI's
 00000000     0008        .LONG   0                      ;  activities for DBG/TDBG
              000C SDBG_SETUP::                          ; An important state flag to control SSI's
 00000000     000C        .LONG   0                      ;  activities for SDBG
              0010 PAGE_ENTRY::                          ; Pointer to watch variable's page list
 00000000     0010        .LONG   0
              0014 DBG$GB_SET_SSI_CNT::                  ; A flag to indicate watch pointing is active
       00     0014        .BYTE   0                      ;  DEBUG only intecepts if watch pointing is
              0015                                       ;  trigged
              0015 DBG$GV_SSI_CONTROL::                  ; A state vector to control which interception
       00     0015        .BYTE   0                      ;  routine is active at the moment
              0016 SAVE_SSI_STATE::                      ; A state vector is used in helping to set the
       00     0016        .BYTE   0                      ;  above state vector.  It serves the
              0017                                       ;  communication gap between TDBG and SDBG
```

```
                0017  ;
                0017  ; OWN STORAGE
                0017  ;
                0017  term_reason:
      00000000  0017          .LONG   0                ; Location for termination reason
                001B
                001B  term_block_one:
      00000000  001B          .LONG   0                ; Forward link
      000002F7' 001F  fix_1:  .ADDRESS term_handler    ; Address of termination handler
      00000001  0023          .LONG   1                ; Argument count
      00000017' 0027  fix_2:  .ADDRESS term_reason     ; Address of termination reason
                002B
                002B  term_block_two:
      00000000  002B          .LONG   0                ; Forward link
      000002B1' 002F  fix_3:  .ADDRESS restore_context; Address of termination handler
      00000001  0033          .LONG   1                ; Argument count
      00000017' 0037  fix_4:  .ADDRESS term_reason     ; Address of termination reason
```

```
          003B  ; **************** saved_AP and saved_FP must be contiguous ****************
          003B  ; **************** saved_R0 and saved_R1 must be contiguous ****************
          003B
00000000  003B  saved_AP:.LONG  0                  ; Original AP
00000000  003F  saved_FP:.LONG  0                  ; Original FP
00000000  0043  saved_R0:.LONG  0                  ; R0 and R1 are saved contiguously so that
00000000  0047  saved_R1:.LONG  0                  ;  they can be preserved across $EXIT_S
          004B
          004B  faobufdesc:
00000100  004B          .LONG   buf_siz            ; length of FAO buffer
00000000  004F          .LONG   0                  ; address of FAO buffer
          0053  msg_length:
    0000  0053          .WORD   0                  ; holds temporary buffer lengths
          0055  fao_buf:
00000155  0055          .BLKB   buf_siz            ; buffer for FAO messages
          0155  log_buf:
00000255  0155          .BLKB   buf_siz            ; buffer for LOG file
          0255  term_buf:
00000355  0255          .BLKB   buf_siz            ; buffer for EXIT reason messages
          0355
000000[illegible] 0355  const_0:.LONG   0
          0359  const_1:
00000001  0359  param_0:.LONG   1                  ; Parameter count
00000361  035D  param_1:.BLKL   1                  ; Actual parameter value
          0361
00000365  0361  user_pc:.BLKL   1                  ; Saved PC for fake LIB$SIGNAL entry
00000369  0365  user_fp:.BLKL   1                  ; Saved FP for fake LIB$SIGNAL entry
0000036D  0369  handler:.BLKL   1                  ; Address of user handler (or zero)
00000001  036D  dbg$gl_exit_status::.LONG 1        ; Status to be returned to DCL on exit
```

```
                              00000000   328         .PSECT  DBG$PLIT BYTE, PIC, SHR, NOWRT, EXE
                                  0000   329
                                  0000   330 routine_value:
72 75 74 65 72 20 65 75 6C 61 76 00' 0000   331         .ASCIC \value returned is \
             20 73 69 20 64 65 6E  000C
                                12  0000
```

```
    0013        .SBTTL  BEGINHERE       - called by DCL via DBGBOOT
00000000        .PSECT  DBG$CODE BYTE, PIC, SHR, NOWRT, EXE
    0000  ;++
    0000  ; FUNCTIONAL DESCRIPTION:
    0000  ;
    0000  ;        Routine "beginhere" is where DEBUG is given control from the CLI,
    0000  ;       either at the start of program execution or in response to the DCL
    0000  ;       "DEBUG" command (in the case of RUN/NODEBUG).
    0000  ;       The routine first resolves the two separate ways that DEBUG can be
    0000  ;       entered from the CLI, and coerces them to a common format. It then
    0000  ;       performs once-only DEBUG initialization, and finally it enters the
    0000  ;       exception handler that initiates command processing.
    0000  ;
    0000  ;       AP ---> -----------------------------
    0000  ;               !                         6 !
    0000  ;               -----------------------------
    0000  ;               ! transfer vector address   ! (Exception args if RUN/NODEBUG)
    0000  ;               -----------------------------
    0000  ;               ! parsing information       !
    0000  ;               -----------------------------
    0000  ;               ! image header information  !
    0000  ;               -----------------------------
    0000  ;               !  image file information   !
    0000  ;               -----------------------------
    0000  ;               ! LINK status bits          !
    0000  ;               -----------------------------
    0000  ;               !  CLI status bits          !
    0000  ;               -----------------------------
    0000  ;
    0000  ;       The transfer vector has three or less transfer addresses in it.
    0000  ;       They are ordered as in the picture below:
    0000  ;
    0000  ;               -----------------------------
    0000  ;               ! DEBUG transfer address    !
    0000  ;               -----------------------------
    0000  ;               ! OTS transfer address      !
    0000  ;               -----------------------------
    0000  ;               ! user transfer address     !
    0000  ;               -----------------------------
    0000  ;
    0000  ;       If the DEBUG or OTS transfer addresses are absent, the subsequent
    0000  ;       addresses are moved upward in the list.
```

```
                              0000  ; *****SSI
                              0000  ; Since DEBUG is highly re-entriant, (ie, signal back to DEBUG via SS$_NORMAL,
                              0000  ; Error message, build call frame stack on the fly, or via Branch, JMP etc.),
                              0000  ; to be safe, at each label in this module (potential entry point) we DISABLE
                              0000  ; SSI, so we won't intercept DEBUG'S own System Service calls.  DBG/TDBG
                              0000  ; watches user program's system service, SDBG watches TDBG and user program's
                              0000  ; system service calls.  But at each level DBG, TDBG or SDBG, one never
                              0000  ; intercepts its own system service calls.
                              0000  ;
                        0000  0000          .ENTRY  BEGINHERE,^M<>               ; Null entry mask
     00000B0E'EF   00   FB    0002          CALLS   #0,DISABLE_SSI
     00000000'EF   10   88    0009          BISB2   #dbg$m_control_exit,dbg$gv_control; Exit on startup errors
            09B6        30    0010          BSBW    fix_up_addresses             ; [TEMP] until VMS bug is fixed
     0000003B'EF   5C   7D    0013          MOVQ    AP,saved_AP                  ; Save pointer to CLI parameters
        2F 18 AC   10   E1    001A          BBC     #cli$v_dbgexcp,24(AP),3$; Branch if normal entry to DEBUG
                              001F  ;
                              001F  ; DEBUG has been entered after user program has been started. Find the
                              001F  ; call frame on the stack which will resturn control to SYS$IMGSTA and
                              001F  ; change the return address to point to dbg$user_exit so that DEBUG is
                              001F  ; given control if the user program exits via a RETURN.
                              001F  ;
          5C    04 AC   D0    001F          MOVL    4(AP),AP                     ; Get pointer to exception parameters
          50    08 AC   D0    0023          MOVL    chf$l_mcharglst(AP),R0       ; Get address of MECHANISM arguments
          50    04 A0   D0    0027          MOVL    chf$l_mch_frame(R0),R0       ; Get FP of establisher (SYS$IMGSTA)
     0000003F'EF   50   D0    002B          MOVL    R0,saved_FP                  ; Save for last-chance handler
               51  5D   D0    0032          MOVL    FP,R1                        ; Get current frame pointer
          51    0C A1   D0    0035  1$:     MOVL    sf$l_save_fp(R1),R1          ; Step back to previous frame
       61   14     00   0C    0039          PROBER  #0,#20,(RT)                  ; Can this stack frame be read ?
                   0C   13    003D          BEQL    2$                           ; No  - stack is corrupted ?
          50    0C A1   D1    003F          CMPL    sf$l_save_fp(R1),R0          ; Do we point back to SYS$IMGSTA ?
                   F0   12    0043          BNEQ    1$                           ; No  - go look at earlier frames
    10 A1    018A'CF    9E    0045          MOVAB   W^dbg$user_exit,sf$l_save_pc(R1); Yes - change return PC
               00A1     31    004B  2$:     BRW     setup                        ; Go perform common initialization
                              004E  ;
                              004E  ; DEBUG has been given control directly at start of program execution.
                              004E  ; We need to build a fake call-frame on the stack,  so that it appears
                              004E  ; to the user as though DEBUG had been given control after the CALL of
                              004E  ; his program but before execution of any user instructions.
                              004E  ;
  04 00000000'EF   01   E0    004E  3$:     BBS     #dbg$v_control_sdbg,dbg$gv_control,4$;Use this vector if SDBG
           04 AC   04   C0    0056          ADDL2   #4,4(AP)                     ; Otherwise step to OTS or USER entry
       50    01    10   78    005A  4$:     ASHL    #16,#1,R0                    ; Get all-zero default entry mask
          51    04 BC   D0    005E          MOVL    @4(AP),R1                    ; Get address of user transfer vector
       61    02    00   0C    0062          PROBER  #0,#2,(R1)                   ; Can transfer address be read ?
                   03   13    0066          BEQL    5$                           ; If not, don't try to read it !
               50  61   3C    0068          MOVZWL  (R1),R0                      ; Get user entry mask bits in R0
     00000043'EF   50   D0    006B  5$:     MOVL    R0,saved_R0                  ; Save entry-mask and flag bit
  00000047'EF  51  02   C1    0072          ADDL3   #2,R1,saved_R1               ; Save transfer address as well
     51    5E    02     00    EF    007A    EXTZV   #0,#2,SP,R1                  ; Get low two bits of stack pointer
               5E  03   CA    007F          BICL2   #3,SP                        ; Force stack to longword alignment
       50   F000 8F     AA    0082          BICW2   #^XF000,R0                   ; Mask to just bits 0-11 (registers)
                   50   BB    0087          PUSHR   R0                           ; Save registers given in entry mask
            018A'CF     9F    0089          PUSHAB  W^dbg$user_exit              ; Set up fake return address
            3000 8F     BB    008D          PUSHR   #^M<FP,AP>                   ; Save current context registers
    7E    51    FE 8F   9C    0091          ROTL    #-2,R1,-(SP)                 ; Set stack alignment bits
          02 AE    50   A8    0096          BISW2   R0,2(SP)                     ; Include register save mask
                   7E   D4    009A          CLRL    -(SP)                        ; Initialize stack exception handler
```

```
        6D   B1'AF   9E 009C   433        MOVAB   B^one_shot_handler,(FP) ; Establish handler in outer frame
            5D    5E   D0 00A0   434        MOVL    SP,FP                   ; Frame established - set pointer
   7E  00028001 8F   D0 00A3   435        MOVL    #dbg$_normal,-(SP)      ; Stack special exception code
   00000000'GF    01   FB 00AA   436        CALLS   #1,G^LIB$SIGNAL         ; SIGNAL into exception handler
                          00B1   437 ;
                          00B1   438 ;      Never returns here - PC changed within handler !
```

```
                                   00B1   440 one_shot_handler:
                           0FFC    00B1   441          .WORD    ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>     ; Full save entry mask
        00000B0E'EF    00    FB    00B3   442          CALLS    #0,DISABLE_SSI
             50    04 AC    D0    00BA   443          MOVL     chf$l_sigarglst(AP),R0  ; Get address of SIGNAL arg list
        04 A0    046C 8F    3C    00BE   444          MOVZWL   #ss$_debug,4(R0)        ; Change exception name to SS$_DEBUG
             51    60    01    C3    00C4   445          SUBL3    #1,chf$l_sig_args(R0),R1; Get offset to saved PC value
             50    6041    DE    00C8   446          MOVAL    (R0)[R1],R0             ; Get address of saved PC parameter
        80  00000047'EF    D0    00CC   447          MOVL     saved_R1,(R0)+          ; Change to user program start address
51   00000043'EF    01    0F    EF    00D3   448          EXTZV    #15,#1,saved_R0,R1      ; Get entry-mask decimal enable bit
        60    01    07    51    F0    00DC   449          INSV     R1,#psl$v_dv,#1,(R0)    ; Set decimal overflow bit in saved PSL
51   00000043'EF    01    0E    EF    00E1   450          EXTZV    #14,#1,saved_R0,R1      ; Get entry-mask integer enable bit
        60    01    05    51    F0    00EA   451          INSV     R1,#psl$v_iv,#1,(R0)    ; Set integer overflow bit in saved PSL
                                   00EF   452 ;
                                   00EF   453 ; fall through into common DEBUG once-only initialization code
                                   00EF   454 ;
                                   00EF   455 setup:
                    0125    30    00EF   456          BSBW     save_user_context       ; Establish known state of the world
             50    08 AC    D0    00F2   457          MOVL     chf$l_mcharglst(AP),R0  ; Get address of MECHANISM array
             51    04 A0    D0    00F6   458          MOVL     chf$l_mch_frame(R0),R1  ; Get frame of our establisher
             61    065B'CF    DE    00FA   459          MOVAL    W^dbg$final_handl,(R1)  ; Establish final exception handler
                                   00FF   460
             6D    0640'CF    DE    00FF   461          MOVAL    W^window_handler,(FP)   ; Establish local exception handler
                    00D0    30    0104   462          BSBW     setup_exit_handler      ; Establish DEBUGs final exit handler
                 3C 50    E9    0107   463          BLBC     R0,1$                   ; Go EXIT with error-status on failure
        00000000'EF    00    FB    010A   464          CALLS    #0,dbg$init_debug       ; initialize DEBUG context.
                                   0111   465          $SETEXV_S VECTOR=#2,ADDRES=last_chance; Declare last chance handler
                 1F 50    E9    0124   466          BLBC     R0,1$                   ; Go EXIT with error-status on failure
50     00000000'EF    01    00    EF    0127   467          EXTZV    #dbg$v_control_tdbg,#1,dbg$gv_control,R0 ; Get TEST DEBUG flag
                                   0130   468          $SETEXV_S VECTOR=R0,ADDRES=primary_handler; Declare 'primary' handler
                 0E 50    E8    0143   469          BLBS     R0,2$                   ; Carry on if declaration successful
        50    03    00    04    F0    0146   470 1$:      INSV     #4,#0,#3,R0             ; Otherwise make this a fatal error
                                   014B   471          $EXIT_S  R0                      ;  and report status via SYS$EXIT
                                   0154   472
             50    04 AC    D0    0154   473 2$:      MOVL     chf$l_sigarglst(AP),R0  ; Get address of SIGNAL arg list
             51    60    01    C3    0158   474          SUBL3    #1,chf$l_sig_args(R0),R1; Get offset to saved PC value
                    6041    DD    015C   475          PUSHL    (R0)[R1]                ; Stack PC value for dbg$rst_init
        50    0000003B'EF    D0    015F   476          MOVL     saved_AP,R0             ; Get address of CLI vector
                    0C A0    DD    0166   477          PUSHL    cli$a_imghdadr(R0)      ; Push address of image header info.
             50    10 A0    D0    0169   478          MOVL     cli$a_imgfiled(R0),R0   ; Get the address of the image file
             51    02 A0    3C    016D   479          MOVZWL   ifd$w_filnamoff(R0),R1  ; get offset to file name
        7E    51    50    C1    0171   480          ADDL3    R0,R1,-(SP)             ; Push address onto stack
             7E    08 A0    3C    0175   481          MOVZWL   ifd$w_chan(R0),-(SP)    ; Push channel number onto stack
        00000000'EF    04    FB    0179   482          CALLS    #4,dbg$rst_init         ; Initialize the symbol tables.
        00000000'EF    10    8A    0180   483          BICB2    #dbg$m_control_exit,dbg$gv_control ; Turn off exit flag
                    041B    31    0187   484          BRW      prim_4                  ; Act as though its a normal exception
```

```
                                018A  ;
                                018A  ;       When the user program runs to completion, then control runs back
                                018A  ;       to the label dbg$user_exit, and DEBUG forces a SYS$EXIT with the
                                018A  ;       user's value of R0. Our termination handler then reports this R0
                                018A  ;       as a system message.  The rest of this routine  (which starts at
                                018A  ;       beginhere_2) is jumped to from within the termination handler.
                                018A  ;       It recreates the original argument list with a new R0,R1 pair to
                                018A  ;       preserve them across $EXIT_S, and moves back through the restart
                                018A  ;       logic to the command processor.
                                018A
                                018A  dbg$user_exit::
 00000B0E'EF     00       FB    018A          CALLS   #0,disable_ssi
 0000036D'EF     50       D0    0191          MOVL    R0,dbg$gl_exit_status   ; Save user program's return status
 00000043'EF     50       7D    0198          MOVQ    R0,saved_R0             ; Stuff away for later restoration.
                                019F          $EXIT_S R0                      ; Force SYS$EXIT with user's R0 value.
                          04    01A8          RET                             ; Don't need to set R0 here !
                                01A9
                                01A9
                                01A9  reset_debug:
         50      8E       D0    01A9          MOVL    (SP)+,R0                ; Get back return address
 5C    0000003B'EF        7D    01AC          MOVQ    saved_AP,AP             ; Restore saved AP and FP
         5E      5D       D0    01B3          MOVL    FP,SP                   ; Restore SP to be the saved FP
       FFD0      CF       9F    01B6          PUSHAB  W^dbg$user_exit         ; Set up fake return address
       3000      8F       BB    01BA          PUSHR   #^M<FP,AP>              ; Save current context registers
                 7E       D4    01BE          CLRL    -(SP)                   ; Set register save mask & PSW
 7E    000003FF'EF        9E    01C0          MOVAB   term_window_handler,-(SP);Establish temporary window handler
         5D      5E       D0    01C7          MOVL    SP,FP                   ; Point to current frame
                 50       DD    01CA          PUSHL   R0                      ; Stack return address again
         7E      01       CE    01CC          MNEGL   #1,-(SP)                ; Replace all BPT's with their real
 00000000'EF     01       FB    01CF          CALLS   #1,dbg$ins_opcodes      ;  opcodes, and unprotect all pages
                          05    01D6          RSB                             ; Return to caller with a new frame
                                01D7
                                01D7  setup_exit_handler:
                                01D7          $DCLEXH_S DESBLK=term_block_one ; Declare a termination handler
      2F 50               E9    01E4          BLBC    R0,3$                   ; Return error-status to caller
   28 00000000'EF         E8    01E7          BLBS    dbg$gv_control,3$       ; No re-arranging if TEST DEBUG
 51    0000001B'EF        D0    01EE  1$:     MOVL    term_block_one,R1       ; Get link to first USER exit handler
                 1F       13    01F5          BEQL    3$                      ; Zero link means we are the last one
                 51       DD    01F7          PUSHL   R1                      ; Save address of control block
                                01F9          $CANEXH_S DESBLK=(R1)           ; Un-declare user exit handler
      02 50               E9    0202          BLBC    R0,2$                   ; Return error status to caller
                 E7       10    0205          BSBB    1$                      ; Repeat for all user exit handlers
         51      8E       D0    0207  2$:     MOVL    (SP)+,R1                ; Get back address of control block
      09 50               E9    020A          BLBC    R0,3$                   ; Report error-status to caller
                                020D          $DCLEXH_S DESBLK=(R1)           ; Re-establish handlers in LIFO order
                          05    0216  3$:     RSB                             ; Return status in R0
```

```
                              0217         .SBTTL  DEBUG entry and exit routines - save/restore state of user
                              0217   ;
                              0217   ;  This routine is called on entry to DEBUG to save the user's registers
                              0217   ; and sundry other user context in the current RUNFRAME,  and to set the
                              0217   ; context of DEBUG to a known state (AST's disabled, etc.).
                              0217   ;
                              0217   save_user_context:
      01 00000001'EF     E8   0217           BBCS    dbg$gv_control+1,save_user_context_always
                              021E                                            ; Only do this if user was 'in control'
                         05   021E           RSB                              ; Otherwise return immediately
                              021F   save_user_context_always:
             01FF 8F     BB   021F           PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8> ; Save all registers we use
    00000B0E'EF   00     FB   0223           CALLS   #0,disable_ssi
                              022A           $SETAST_S #0                    ; Disable AST interrupts
    56   00000000'EF     9E   0233           MOVAB   dbg$runframe,R6         ; Get pointer to current RUNFRAME
         57   04 AC      7D   023A           MOVQ    4(AP),R7                ;  and to SIGNAL & MECHANISM arrays
     48 A6   0820 8F     AA   023E           BICW2   #dbg$m_enab_fex+dbg$m_enab_ast,dbg$w_run_stat(R6)
  05 00000000'EF   0A    E4   0244           BBSC    #dbg$v_control_tbit,dbg$gv_control,2$; (set if ASTs held off)
            50   01      D1   024C           CMPL    #ss$_wasclr,R0          ; Were ASTs enabled?
                 04      13   024F           BEQL    3$                      ; No - flag is already clear
       48 A6     20      A8   0251   2$:     BISW2   #dbg$m_enab_ast,dbg$w_run_stat(R6) ; Yes - remember to reenable
                              0255   3$:     $SETSFM_S #0                    ; Disable sys service failure exception
            50   01      D1   025E           CMPL    #ss$_wasclr,R0          ; Was it enabled?
                 06      13   0261           BEQL    4$                      ; No - flag is already clear
     48 A6   0800 8F     A8   0263           BISW2   #dbg$m_enab_fex,dbg$w_run_stat(R6) ; Yes - remember to reenable
              04 A7      DD   0269   4$:     PUSHL   chf$l_sig_name(R7)      ; Stack actual exception code
     00000000'EF   01    FB   026C           CALLS   #1,dbg$exception_is_fault;Get type of exception (fault/trap)
  48 A6   01   0D   50   F0   0273           INSV    R0,#dbg$v_at_fault,#1,dbg$w_run_stat(R6) ; Remember exc type
         53   04 A6      DE   0279           MOVAL   dbg$l_user_regs(R6),R3  ; Get address for user's registers
         83   0C A8      7D   027D           MOVQ    chf$l_mch_savr0(R8),(R3)+;Copy R0,R1 from MECHANISM array
    63   14 AD   28      28   0281           MOVC3   #40,20(FP),(R3)         ; Save user registers R2 - R11
         83   08 AD      7D   0286           MOVQ    sf$l_save_ap(FP),(R3)+  ; Save user AP - FP
      50   01   67       C1   028A           ADDL3   (R7),#1,R0              ; Get signal arg count plus 1
         50   6740       DE   028E           MOVAL   (R7)[R0],R0             ; Calculate value of user SP
            83   50      D0   0292           MOVL    R0,(R3)+                ; Save user SP in RUNFRAME
            83   70      7D   0295           MOVQ    -(R0),(R3)+             ; Save last 2 SIGNAL args (PC & PSL)
                              0298           $DCLEXH_S DESBLK=term_block_two ; Declare temporary exit handler
    00000001'EF   01     8A   02A5           BICB2   #dbg$m_control_user@-8,dbg$gv_control+1 ; DEBUG is in control
             01FF 8F     BA   02AC           POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8> ; Restore all used registers
                         05   02B0           RSB                             ; Return with user context saved
                              02B1
                              02B1   restore_context:
                       0004   02B1           .WORD   ^M<R2>                  ; Save contents of register R2
 00000000'EF   0108 8F   A8   02B3           BISW2   #dbg$m_control_user+dbg$m_control_urun,dbg$gv_control
      52   00000000'EF   9E   02BC           MOVAB   dbg$runframe,R2         ; Get base of current run frame
     09 48 A2   0B       E1   02C3           BBC     #dbg$v_enab_fex,dbg$w_run_stat(R2),1$    ; Was SFM enabled ?
                              02C8           $SETSFM_S #1                    ; Yes - reenable exceptions
     11 48 A2   05       E1   02D1   1$:     BBC     #dbg$v_enab_ast,dbg$w_run_stat(R2),2$    ; Were AST's enabled ?
  11 00000000'EF   0A    E0   02D6           BBS     #dbg$v_control_tbit,dbg$gv_control,3$    ;   and not postponed ?
                              02DE           $SETAST_S #1                    ; Yes - reenable ASTs
  00 00000000'EF   0A    E5   02E7   2$:     BBCC    #dbg$v_control_tbit,dbg$gv_control,3$    ; No ASTs postponed !
                              02EF   ; *****SSI
                              02EF   ; Time to leave DEBUG, ENABLE SSI, This is the only place we enable SSI.
                              02EF   ;
    000009FD'EF   00     FB   02EF   3$:     CALLS   #0,enable_ssi
                         04   02F6           RET                             ; User context reset - return
```

```
                                02F7            .SBTTL  DEBUG Termination  and last-chance handlers
                                02F7
                                02F7    TERM_HANDLER:
                          0000  02F7            .WORD   ^M<>                    ; Null entry mask
                                02F9    DBG$TERM_HANDLR::
       00000B0E'EF    00    FB  02F9            CALLS   #0,disable_ssi
    07 00000000'EF          E9  0300            BLBC    DBG$GL_SCREEN_MODE,2$   ; Set up screen refresh and set scrol-
       00000000'EF    00    FB  0307            CALLS   #0,DBG$SCR_SCREEN_TERM  ;       ling region back to normal
    0D 00000000'EF    04    E0  030E    2$:     BBS     #DBG$V_CONTROL_EXIT,DBG$GV_CONTROL,3$   ; Exit if exit flag set
00000017'EF    00000980 8F  D1  0316            CMPL    #SS$_CLIFRCEXT,TERM_REASON              ; Check the exit status
                      2D    12  0321            BNEQ    5$                      ; Continue unless "CLI forced exit"
                                0323    ; Before we return to CLI, we check for this special case.
    24 00000000'EF    06    E0  0323    3$:     BBS     #DBG$V_CONTROL_DONE,DBG$GV_CONTROL,4$
                                032B    ; In testable debugger, if the user program does not run to an end, if we
                                032B    ; reach here via DBG>EXIT, if there are break points set along the path
                                032B    ; such as PRIMARY_HANDLER, DBG$EXCEPTION_HANDLER in SDBG, causing super
                                032B    ; debugger goes into an infinite loop.  ($EXIT calls all user declared
                                032B    ; handler after we reach RET instruction here, one of the user declared
                                032B    ; handler is the DBG$TERM_HANDLR in Super Debugger, Super debugger signals
                                032B    ; exit status, then causing break point faults to take place, note: at this
                                032B    ; point super debugger is no longer available).  So before we RET, we do
                                032B    ; some cleaning work.
0000036D'EF    00000017'EF  D0  032B            MOVL    TERM_REASON,DBG$GL_EXIT_STATUS  ; Save the $EXIT status code
       00000001'EF    01    8A  0336            BICB2   #dbg$m_control_user@-8,dbg$gv_control+1 ; DEBUG is in control
     00000000'EF   40 8F    88  033D            BISB2   #dbg$m_control_done,dbg$gv_control      ; User program complete
                    FE61    30  0345            BSBW    reset_debug             ; Re-establish a known context
        50    00000017'EF   D0  0348            MOVL    TERM_REASON,R0
                            04  034F    4$:     RET                             ; Don't intercept - return to CLI
                                0350
0000036D'EF    00000017'EF  D0  0350    5$:     MOVL    TERM_REASON,DBG$GL_EXIT_STATUS  ; Save the $EXIT status code
00000017'EF    10000000 8F  CA  035B            BICL2   #sts$m_inhib_msg,term_reason            ; Clear 'inhibit' bit
       00000001'EF    01    8A  0366            BICB2   #dbg$m_control_user@-8,dbg$gv_control+1 ; DEBUG is in control
     00000000'EF   40 8F    88  036D            BISB2   #dbg$m_control_done,dbg$gv_control      ; User program complete
                    FE31    30  0375            BSBW    reset_debug             ; Re-establish a known context
                    FE5C    30  0378            BSBW    setup_exit_handler      ; Re-establish final exit handler
   0000004B'EF    00FF 8F   3C  037B            MOVZWL  #buf_siz-1,faobufdesc   ; Try to convert the status
0000004F'EF    00000256'EF  9E  0384            MOVAB   term_buf+1,faobufdesc+4 ; to a system message
                                038F            $GETMSG_S MSGID=term_reason,MSGLEN=msg_length,BUFADR=faobufdesc
00000255'EF    00000053'EF  90  03AC            MOVB    msg_length,term_buf     ; Make counted string in TERM_BUF
               00000255'EF  9F  03B7            PUSHAB  term_buf                ; Address of counted string
                        01  DD  03BD            PUSHL   #1                      ; One FAO parameter for SIGNAL
               0002806B 8F  DD  03BF            PUSHL   #dbg$_exitstatus        ; Message number
       00000000'GF    03    FB  03C5            CALLS   #3,G^LIB$SIGNAL         ; SIGNAL exit status back to DEBUG
                                03CC
                                03CC    last_chance:
                          0000  03CC            .WORD   ^M<>                    ; Null entry mask
       00000B0E'EF    00    FB  03CE            CALLS   #0,DISABLE_SSI
       00000001'EF    01    8A  03D5            BICB2   #dbg$m_control_user@-8,dbg$gv_control+1 ; DEBUG is in control
       00000000'EF    6C    FA  03DC            CALLG   (AP),dbg$putmsg         ; Output signal message text
            50    08 AC     D0  03E3            MOVL    chf$l_mcharglst(AP),R0  ; Get address of MECHANISM array
   00000043'EF    0C A0     7D  03E7            MOVQ    chf$l_mch_savr0(R0),saved_R0 ; Save contents of user registers
                    FDB7    30  03EF            BSBW    reset_debug             ; Reset stack to a known state
               00028258 8F  DD  03F2            PUSHL   #dbg$_lastchance        ; Message number
       00000000'GF    01    FB  03F8            CALLS   #1,G^LIB$SIGNAL         ; SIGNAL back in to DEBUG
                                03FF
                                03FF    term_window_handler:
                          0FFC  03FF            .WORD   ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>     ; Full entry mask
```

```
         00000B0E'EF    00   FB  0401    647         CALLS   #0,DISABLE_SSI
               52   04 AC    7D  0408    648         MOVQ    chf$l_sigarglst(AP),R2  ; Get SIGNAL & MECHANISM addresses
   0C A3    00000043'EF      7D  040C    649         MOVQ    saved_R0,chf$l_mch_savr0(R3)     ; Set contents of user R0,R1
                    04 B3    D4  0414    650         CLRL    @chf$l_mch_frame(R3)    ; Remove link to this handler
         00000000'EF    6C   FA  0417    651         CALLG   (AP),dbg$putmsg         ; Output signal message text
            51   62     01   C3  041E    652         SUBL3   #1,(R2),R1              ; Get offset to saved PC value
                      8241   D4  0422    653         CLRL    (R2)+[R1]               ; Clear PC to make restart difficult
            62   046C   8F   3C  0425    654         MOVZWL  #ss$_debug,(R2)         ; Change signal name to 'SS$_DEBUG'
         00000001'EF    01   88  042A    655         BISB2   #dbg$m_control_user@-8,dbg$gv_control+1 ; User was in control
                      016E   31  0431    656         BRW     prim_3                  ; Go save context & issue DEBUG prompt
```

```
                                 0434        .SBTTL  DBG$PSEUDO_PROG - Structure to implement CALL
                                 0434
                                 0434  dbg$pseudo_prog::
   00000052'FF   0000004E'FF  FA 0434        CALLG   @dbg$runframe+dbg$l_frame_ptr,@dbg$runframe+dbg$l_call_addr
                                 043F
                                 043F  dbg$pseudo_exit::; Label to detect STEPping off the end of the CALLed routine.
        00000B0E'EF    00  FB   043F        CALLS   #0,disable_ssi
        00000043'EF    50  7D   0446        MOVQ    R0,saved_R0             ; Save return value from user procedure
                                 044D        $SETAST_S #0                   ; Disable AST interrupts
                                 0456        $DCLEXH_S DESBLK=term_block_two ; Declare temporary exit handler
        00000001'EF    01  8A   0463        BICB2   #dbg$m_control[_user@-8,dbg$gv_control+1 ; DEBUG is in control
                  5C   6D  D0   046A        MOVL    (FP),AP                 ; Get pointer to current handler
             6D   7E'AF    3E   046D        MOVAW   B^pseudo_handler,(FP)   ; Get handler to bootstrap into DEBUG
          00028001 8F      DD   0471        PUSHL   #dbg$_normal            ; Get phony exception value
        00000000'GF    01  FB   0477        CALLS   #1,G^LIB$SIGNAL         ; SIGNAL back to proper context
                                 047E  ;      point of no return !
                                 047E
                        0FFC     047E  pseudo_handler: .WORD   ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Full save mask
                                 0480
             5A   04 AC    7D   0480        MOVQ    chf$l_sigarglst(AP),R10 ; Get pointers to SIGNAL & MECHANISM
          04 BB   08 AD    D0   0484        MOVL    sf$l_save_ap(FP),@chf$l_mch_frame(R11)  ; Restore handler
        00000000'EF    00  FB   0489        CALLS   #0,dbg$flushbuf         ; Initialize print buffer.
           00000000'EF     DF   0490        PUSHAL  routine_value           ; Report the value returned
        00000000'EF    01  FB   0496        CALLS   #1,dbg$print            ; Insert text in buffer
           00000043'EF     DD   049D        PUSHL   saved_R0                ;  call of DBG$OUT_NUM_VAL
        00000000'EF    01  FB   04A3        CALLS   #1,dbg$out_num_val      ; Insert returned numeric value
        00000000'EF    00  FB   04AA        CALLS   #0,dbg$newline          ; Output buffer contents
        5B    00000000'EF  9E   04B1        MOVAB   dbg$runframe,R11        ; Get address of routine's runframe
                  56   6B  D0   04B8        MOVL    dbg$l_next_link(R11),R6 ;  and address of previous runframe
             55   4E AB    D0   04BB        MOVL    dbg$l_frame_ptr(R11),R5 ; get address of routine argument list
                  54   65  D0   04BF        MOVL    (R5),R4                 ; Get number of parameters passed
                       25  13   04C2        BEQL    3$                      ; No data structure if no parameters !
         53   61 BB44      DE   04C4        MOVAL   @dbg$l_save_fld(R11)[R4],R3     ; Otherwise get MEMUSE vector
        0000035D'EF    73  D0   04C9  1$:   MOVL    -(R3),param_1           ; Get memory used for this parameter
                       0B  13   04D0        BEQL    2$                      ; Zero means none allocated !
   00000000'GF   00000359'EF  FA 04D2       CALLG   param_0,G^dbg$rel_memory; Otherwise release memory again
                E9 54      F5   04DD  2$:   SOBGTR  R4,1$                   ; Loop for all parameters
                       53  DD   04E0        PUSHL   R3                      ; Then point to MEMUSE vector area
        00000000'GF    01  FB   04E2        CALLS   #1,G^dbg$rel_memory     ;  and release that space as well
                       55  DD   04E9  3$:   PUSHL   R5                      ;  Push address of block.
        00000000'GF    01  FB   04EB        CALLS   #1,G^dbg$rel_memory     ; Free space used for argument list
      6B   66   0065 8F    28   04F2        MOVC3   #dbg$k_runfr_len,(R6),(R11); Restore previous context
                       56  DD   04F8        PUSHL   R6                      ; Push address of runframe
        00000000'GF    01  FB   04FA        CALLS   #1,G^dbg$rel_memory     ; Free this storage too
                       52  D4   0501        CLRL    R2                      ; Pop "unhandled exc" stack
 00000000'EF42   00000001'EF42  90 0503  4$: MOVB   DBG$GB_UNHANDLED_EXC+1[R2],DBG$GB_UNHANDLED_EXC[R2]
             EF 52     08  F3   0510        AOBLEQ  #8,R2,4$
        00000000'EF    01  91   0514        CMPB    #1,DBG$GB_CALL_NORMAL_RET; Set CALL flag to indicate a normal
                       06  12   051B        BNEQ    5$                      ;        return from a CALL command call
           00000000'EF     96   051D        INCB    DBG$GB_CALL_NORMAL_RET  ;        (used to suppress screen update)
        51    6A   01      C3   0523  5$:   SUBL3   #1,(R10),R1             ; Get offset to saved SIGNAL PC
             51    6A41    DE   0527        MOVAL   (R10)[R1],R1            ; Get actual address for PC & PSL
       04 AA    046C 8F    3C   052B        MOVZWL  #ss$_debug,4(R10)       ; Change signal name to 'SS$_DEBUG'
             61   40 AB    7D   0531        MOVQ    dbg$l_user_pc(R11),(R1) ; Restore PC & PSL to SIGNAL array
                       6E  11   0535        BRB     prim_4                  ; Rejoin common exception flow
```

```
0537          .SBTTL  PRIMARY_HANDLER
0537  ;++
0537  ; FUNCTIONAL DESCRIPTION:
0537  ;       Exception handler declared in the primary vector. Simply resignals
0537  ;       if the exception occurred during the execution of a debug command.
0537  ;       If the exception occurred in the user program being debugged, this
0537  ;       routine disables ASTs (if they were enabled),  saves the registers
0537  ;       from the user program at the time of the exception, and then calls
0537  ;       a routine to handle the exception.
0537  ;       When the called routine returns, the registers are restored,  ASTs
0537  ;       are re-enabled (if they were disabled),  and the exception handler
0537  ;       returns with the resignal value received from the called routine.
0537  ;
0537  ; CALLING SEQUENCE:
0537  ;       4(AP)    - Address of SIGNAL ARRAY
0537  ;       8(AP)    - Address of MECHANISM ARRAY
0537  ;
0537  ; IMPLICIT INPUTS:
0537  ;       The global flag dbg$v_control_user, which indicates whether the
0537  ;       user program was running, or DEBUG was executing a debug command.
0537  ;
0537  ; IMPLICIT OUTPUTS:
0537  ;       The "dbg$v_enab_ast" flag indicates whether asts were enabled
0537  ;       at the time of the interrupt
0537  ;
0537  ; ROUTINE VALUE:
0537  ;       SS$_RESIGNAL or the value returned by DBG$EXC_HANDLER
0537  ;
0537  ; SIDE EFFECTS:
0537  ;       The user may get control under DEBUG
0537  ;--
```

```
                          0FFC  0537        .ENTRY  PRIMARY_HANDLER,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
                                0539  prim_handl_2::
      00000B0E'EF   00   FB     0539        CALLS   #0,disable_ssi
         50   0918 8F    3C     0540        MOVZWL  #ss$_resignal,R0          ; We usually want to resignal exception
           51   04 AC    D0     0545        MOVL    chf$l_sigarglst(AP),R1    ; Get address of SIGNAL argument list
 04 A1   00028001 8F     D1     0549        CMPL    #dbg$_normal,4(R1)        ; Fake DEBUG-generated signal ?
                   36    13     0551        BEQL    2$                        ; Yes - just resignal it at once
 04 A1   0000046C 8F     D1     0553        CMPL    #ss$_debug,4(R1)          ; Generated by ^Y,DEBUG sequence ?
                   25    12     055B        BNEQ    1$                        ; No  - pass on to next handler
              50   01    3C     055D        MOVZWL  #ss$_continue,R0          ; Get ready to ignore 'SS$_DEBUG'
    22 00000000'EF       E8     0560        BLBS    dbg$gv_control[,2$        ; Always ignore if testable debugger
 13 00000000'EF   09     E3     0567        BBCS    #dbg$v_control_stop,dbg$gv_control,1$    ; Set STOP flag
    14 00000001'EF       E8     056F        BLBS    dbg$gv_control[+1,3$      ; Continue unless DEBUG was running
    00000000'EF   20     88     0576        BISB2   #dbg$m_control_fail,dbg$gv_control ; and the STOP flag was set
         50   0918 8F    3C     057D        MOVZWL  #ss$_resignal,R0          ;  when we resignal to final handler
    01 00000001'EF       E8     0582  1$:   BLBS    dbg$gv_control[+1,3$      ; Are debug commands being executed?
                   04           0589  2$:   RET                               ; Yes - return to exception dispatch
                                058A
 10 00000000'EF   01     E1     058A  3$:   BBC     #dbg$v_control_sdbg,dbg$gv_control,prim_3 ; If SUPERDEBUG,
 04 A1   00028352 8F     D1     0592        CMPL    #dbg$_superdebug,4(R1)    ; See if this SUPERDEBUG signal
                   06    12     059A        BNEQ    prim_3                    ; Some other signal - look at it
    04 A1     046C 8F    3C     059C        MOVZWL  #ss$_debug,4(R1)          ; SUPERDEBUG gets changed to DEBUG
                 FC72    30     05A2  prim_3: BSBW  save_user_context         ; Establish known state of the world
       6D   0640'CF      DE     05A5  prim_4: MOVAL W^window_handler,(FP)     ; Establish temporary exception handler
    00000000'EF   6C     FA     05AA        CALLG   (AP),dbg$exc_handler      ; Call inner exception handler
               24 50     E9     05B1        BLBC    R0,return_to_user         ; Just return if re-signalling
 1C 00000000'EF   0A     E0     05B4        BBS     #dbg$v_control_tbit,dbg$gv_control,return_to_user; Single-Step ?
 14 00000000'EF   09     E1     05BC        BBC     #dbg$v_control_stop,dbg$gv_control,return_to_user; ^Y, DEBUG ?
00000361'EF  00000040'EF D0     05C4        MOVL    dbg$runframe+dbg$l_user_regs+60,user_pc             ; Yes - save PC
 00000040'EF    0862'CF  9E     05CF        MOVAB   W^Pseudo_Signal,dbg$runframe+dbg$l_user_regs+60  ;  & set new PC
                                05D8  return_to_user:
           5A   04 AC    7D     05D8        MOVQ    4(AP),R10                 ; Get address of SIGNAL & MECHANISM
                   01    BB     05DC        PUSHR   #^M<R0>                   ; Save resignal value
      51   00000004'EF   DE     05DE        MOVAL   dbg$runframe+dbg$l_user_regs,R1 ; Get address of user regs
           0C AB   81    7D     05E5        MOVQ    (R1)+,12(R11)             ; And restore R0 - R1,
     14 AD    61   28    28     05E9        MOVC3   #40,(R1),20(FP)           ;  user R2 - R11
           08 AD   81    7D     05EE        MOVQ    (R1)+,8(FP)               ;   and user AP - FP
        52    6A   01    C3     05F2        SUBL3   #1,(R10),R2               ; Get offset to saved SIGNAL PC
           52    6A42    DE     05F6        MOVAL   (R10)[R2],R2              ; Get actual address for PC & PSL
        62    04 A1      7D     05FA        MOVQ    4(R1),(R2)                ; Restore PC & PSL to SIGNAL array
                   6D    D4     05FE        CLRL    (FP)                      ; Remove stack-frame exception handler
                                0600        $CANEXH_S DESBLK=term_block_two   ; Un-declare temporary exit handler
      FC9F CF     00     FB     060D        CALLS   #0,restore_context        ; Go reset user AST/SFM enables
                   01    BA     0612        POPR    #^M<R0>                   ; Get the resignal value back
                   04           0614        RET                               ;  and return
```

```
                           0615          .SBTTL  DBG$THREAD_BPT  - Entry to DEBUG for threaded BPT's
                           0615  ;++
                           0615  ; FUNCTIONAL DESCRIPTION:
                           0615  ;       This routine is called by a "JMP @(R11)+" instruction when the "thread"
                           0615  ;       processor encounters the thread where the user has set a breakpoint.
                           0615  ;       An exception frame is built on the stack to describe the breakpoint
                           0615  ;       using R11 -4 as the address of the "PC".  The PRIMARY_HANDLER is called
                           0615  ;       which will announce the breakpoint and process user commands.
                           0615  ;
                           0615  ;       When the user issues a GO or STEP command the actual thread is moved
                           0615  ;       onto the stack from R11 and the exception frame is removed.  The user
                           0615  ;       program is then restarted by an REI to the next thread.
                           0615  ;
                           0615  ; CALLING SEQUENCE:
                           0615  ;       R11     - Contains the address after the thread where the breakpoint
                           0615  ;                 was set.
                           0615  ;
                           0615  ; IMPLICIT INPUTS:
                           0615  ;       The routine was called with a JMP @(R11)+ instruction.
                           0615  ;
                           0615  ; IMPLICIT OUTPUTS:
                           0615  ;       R11 is still the thread pointer.
                           0615  ;
                           0615  ; ROUTINE VALUE:
                           0615  ;       None.
                           0615  ;
                           0615  ; SIDE EFFECTS:
                           0615  ;       None.
                           0615  ;
                           0615  ;--
                           0615
                           0615  dbg$thread_bpt::
             7E    DC      0615          MOVPSL  -(SP)                           ; Save the current PSL
             7B    DF      0617          PUSHAL  -(R11)                          ; Treat R11 as the PC, and set it to
                           0619                                                  ; the address of the thread.
   00000414  8F    DD      0619          PUSHL   #ss$_break                      ; This is a breakpoint exception.
             03    DD      061F          PUSHL   #3                              ; Exception frame has 3 longwords
       7E    50    7D      0621          MOVQ    R0,-(SP)                        ; Save R0,R1
             7E    7C      0624          CLRQ    -(SP)                           ; next 2 longwords of mechanism array
             04    DD      0626          PUSHL   #4                              ; Mechanism array has 4 longwords.
             6E    DF      0628          PUSHAL  (SP)                            ; Build arg list for primary handler
          18 AE    DF      062A          PUSHAL  24(SP)                          ;
   FF05 CF   02    FB      062D          CALLS   #2,primary_handler              ; Call primary handler
                           0632  dbg$thread_ret::                                ; Label where threaded breakpoint returns
      1C AE  8B    D0      0632          MOVL    (R11)+,28(SP)                   ; Save user's actual thread in case
                           0636                                                  ; he changed it during the breakpoint
       5E    0C    C0      0636          ADDL2   #12,SP                          ; Get address of new R0,R1
       50    8E    7D      0639          MOVQ    (SP)+,R0                        ; Restore R0,R1
       5E    08    C0      063C          ADDL2   #8,SP                           ; Remove all but new PC - PSL pair
                   02      063F          REI                                     ; Transfer control to "thread" routine
```

```
                         0640          .SBTTL  WINDOW_HANDLER  - Call frame exception handler
                         0640  ;++
                         0640  ; FUNCTIONAL DESCRIPTION:
                         0640  ;       This handler is put up by the primary handler to be used during the
                         0640  ;       "window" during the processing of an exception and before the DEBUG
                         0640  ;       prompt is output. SS$_DEBUG signals are ignored (we are trying hard
                         0640  ;       to get back to DEBUG command level). everything else causes a jump
                         0640  ;       to FINAL_HANDLER to report the error.
                         0640  ;
                         0640  ; CALLING SEQUENCE:
                         0640  ;       4(AP)   - Address of SIGNAL ARRAY
                         0640  ;       8(AP)   - Address of MECHANISM ARRAY
                         0640  ;
                         0640  ; IMPLICIT INPUTS:
                         0640  ;       NONE
                         0640  ;
                         0640  ; IMPLICIT OUTPUTS:
                         0640  ;       NONE
                         0640  ;
                         0640  ; ROUTINE VALUE:
                         0640  ;       NONE
                         0640  ;
                         0640  ; SIDE EFFECTS:
                         0640  ;       Errors reported by FINAL_HANDLER
                         0640  ;--
                         0640  window_handler:
                   OFFC  0640          .WORD   ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
  00000B0E'EF   00   FB  0642          CALLS   #0,DISABLE_SSI
          50   04 AC  D0  0649          MOVL    4(AP),R0                      ; Get address of SIGNAL ARRAY
 0000046C 8F   04 A0  D1  064D          CMPL    4(R0),#ss$_debug              ; Is this the DEBUG exception ?
              06'  12  0655          BNEQ    dbg$final_handl+2             ; No  - transfer to the final handler
         50    01  3C  0657          MOVZWL  #ss$_continue,R0              ; Yes - load CONTINUE code
                   04  065A          RET                                   ; Ignore extra 'SS$_DEBUG' signals
```

```
                         065B          .SBTTL  DBG$FINAL_HANDL - Call frame exception handler
                         065B   ;
                         065B   ; Functional description:
                         065B   ;       This handler is the ultimate exception handler for exceptions
                         065B   ;       that occur under DEBUG control or during execution of the user
                         065B   ;       program. Any exception that gets here has already passed thru
                         065B   ;       primary handlers, secondary handlers, and user-declared
                         065B   ;       stack handlers. This handler stops the exception from causing
                         065B   ;       an exit to the operating system, and drops the user back at
                         065B   ;       DEBUG command level.
                         065B   ;
                         065B   ;       This routine first determines whether it was called because of
                         065B   ;       a hard/software exception condition, or because of a software
                         065B   ;       generated SIGNAL. The identification of the error is from the
                         065B   ;       signal-arg-list.
                         065B   ;
                         065B   ;       The handler outputs DEBUG generated messages and operating
                         065B   ;       system generated conditions in distinct manners. The latter
                         065B   ;       conditions are reported, analyzed for source of error, and
                         065B   ;       then the user regains control. DEBUG messages are output, and
                         065B   ;       control is returned to the user or to CLI according to the
                         065B   ;       severity of the message.
                         065B   ;
                         065B   ; Calling sequence:
                         065B   ;       4(AP)   - Address of SIGNAL ARRAY for an exception. Contains the
                         065B   ;                 exception name, the PC of the exception, and the PSL
                         065B   ;                 and any additional FAO arguments required by the
                         065B   ;                 particular message to be generated.
                         065B   ;       8(AP)   - Address of MECHANISM ARRAY for an exception. Contains
                         065B   ;                 R0 and R1.
                         065B   ;
                         065B   ; Implicit inputs:
                         065B   ;       The global flag DBG$V_CONTROL_USER says whether DEBUG or the user
                         065B   ;       was running when the exception occurred. The severity of the error
                         065B   ;       is determined by the low three bits in the error identifier.
                         065B   ;
                         065B   ; Implicit outputs:
                         065B   ;       The name of the exception is changed if PRIMARY_HANDLER is
                         065B   ;       called.
                         065B   ;
                         065B   ; Routine value:
                         065B   ;       SS$_RESIGNAL is returned when the exception was SS$_UNWIND.
                         065B   ;       SS$_CONTINUE is returned when the exception occurred in DEBUG
                         065B   ;               code (unless the error was fatal).
                         065B   ;       Otherwise, this code JMPs to primary_handler and the return is
                         065B   ;               dependent upon many subsequent things.
                         065B   ;
                         065B   ; Side effects:
                         065B   ;       An error message is output to the terminal.
                         065B   ;
                         065B
                   0FFC  065B          .ENTRY  DBG$FINAL_HANDL,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
00000B0E'EF    00    FB  065D          CALLS   #0,disable_ssi
     52    04  AC    D0  0664          MOVL    CHF$L_SIGARGLST(AP),R2   ; Get address of signal argument list
00000920 8F    04  A2  D1  0668        CMPL    4(R2),#SS$_UNWIND        ; Is this a SYSTEM unwind exception ?
           06        12  0670          BNEQ    1$                       ; If not unwind, look at it further.
     50  0918 8F     3C  0672          MOVZWL  #SS$_RESIGNAL,R0         ; if unwind, just resignal condition
```

```
                            04  0677          RET
                                0678
             6D    E0 AF    9E  0678  1$:     MOVAB   B^DBG$FINAL_HANDL,(FP)  ; Establish ourselves as a handler
       53    00000001'EF    9A  067C          MOVZBL  DBG$GV_CONTROL+1,R3     ; Save current state of DEBUG/USER flag
                    FB91    30  0683          BSBW    SAVE_USER_CONTEXT       ; Establish known state of the world
 00000004'EF   80000000 8F  C8  0686          BISL2   #RAB$M_CCO,DBG$GL_OUTPRAB+RAB$L_ROP ; Cancel control-O
   0000046C 8F      04 A2   D1  0691          CMPL    4(R2),#SS$_DEBUG        ; Is this the DEBUG exception ?
                       27   13  0699          BEQL    2$                      ; Yes - suppress message output
   00028001 8F      04 A2   D1  069B          CMPL    4(R2),#DBG$_NORMAL      ; Special DEBUG initialization ?
                       23   13  06A3          BEQL    3$                      ; Yes - suppress message output
     00000000'EF       6C   FA  06A5          CALLG   (AP),DBG$PUTMSG         ; Write system message to DBG$OUTPUT
   02    04 A2    0C   10   ED  06AC          CMPZV   #STS$V_FAC_NO,#STS$S_FAC_NO,4(R2),#DBG_FACILITY ;
                       0E   12  06B2          BNEQ    2$                      ; Skip if facility is not DEBUG
       1000 8F      04 A2   B1  06B4          CMPW    4(R2),#SHR$_APPENDEDB   ; Not DEBUG if bit 15 is clear unless
                       06   1F  06BA          BLSSU   2$                      ;       this is a "shared" message
          0000074E'EF       17  06BC          JMP     FINAL_2
                                06C2
                                06C2  ; Come here if not a DEBUG-specific exception (System or User generated).
                                06C2  ; If the user was running, jump into the register saving exception handler.
                                06C2  ; If DEBUG was running, output a message saying that DEBUG caused the error.
                                06C2  ;
                    03 53   E8  06C2  2$:     BLBS    R3,3$                   ; Report error if DEBUG was running
                     00B3   31  06C5          BRW     FINAL_3                 ;       by branching to FINAL_3
           54    62    01   C3  06C8  3$:     SUBL3   #1,(R2),R4              ; Get address of saved PC
                  FD6F CF   9F  06CC          PUSHAB  DBG$PSEUDO_EXIT         ; Get address of DBG$PSEUDO_EXIT
               8E    6244   D1  06D0          CMPL    (R2)[R4],(SP)+          ; See if CALLed routine has finished
                       36   13  06D4          BEQL    4$                      ; If so, just return CONTINUE
                 32 04 A2   E8  06D6          BLBS    4(R2),4$                ; Continue if INFORMATION or SUCCESS
     00000361'EF     6244   D0  06DA          MOVL    (R2)[R4],USER_PC        ; Save actual user PC for error
    0000036D'EF     04 A2   D0  06E2          MOVL    4(R2),DBG$GL_EXIT_STATUS; Remember error status (for EXIT)
   04    04 A2    03   00   ED  06EA          CMPZV   #0,#3,4(R2),#4          ; Check for severe error
                       07   12  06F0          BNEQ    35$                     ; If not, don't fill in global
      00000000'EF      01   90  06F2          MOVB    #1,DBG$GB_UNHANDLED_EXC ; Remember that an unhandled exception
                                06F9                                          ;       has occured in the user program
                                06F9  35$:    $UNWIND_S  DEPADR=CONST_0,NEWPC=PSEUDO_SIGNAL   ; Unwind the stack
                                070C  ;
                                070C  ; After the UNWIND, return to the user, but do so without restoring the
                                070C  ; registers.  The Exception-Handling Facility requires that we not change
                                070C  ; the saved PC, and in any event there is no need to restore any registers
                                070C  ; since they cannot be changed (via DEPOSIT, etc.) in the Final Handler.
                                070C  ;
                 50    01   3C  070C  4$:     MOVZWL  #SS$_CONTINUE,R0        ; Return status "CONTINUE"
                    23 53   E8  070F          BLBS    R3,6$                   ; Restore registers on user exit
           00000000'EF      D5  0712          TSTL    DBG$GL_SCREEN_ERROR     ; Do not purge type-ahead if error msg
                       0F   12  0718          BNEQ    5$                      ;       went to a screen display
                 0B 04 A2   E8  071A          BLBS    4(R2),5$                ; Do not purge type-ahead if severity
 00000004'EF   20000000 8F  C8  071E          BISL2   #RAB$M_PTA,DBG$GL_INPRAB+RAB$L_ROP      ;   is INFO or SUCCESS
 00000004'EF   80000000 8F  CA  0729  5$:     BICL2   #RAB$M_CCO,DBG$GL_OUTPRAB+RAB$L_ROP     ; Un-cancel Ctrl-O
                            04  0734          RET                             ; Return to exception mechanism
                       01   BB  0735  6$:     PUSHR   #^M<R0>                 ; Save resignal value
                       6D   D4  0737          CLRL    (FP)                    ; Remove stack-frame exception handler
                                0739          $CANEXH_S DESBLK=TERM_BLOCK_TWO ; Un-declare temporary exit handler
        FB66 CF        00   FB  0746          CALLS   #0,RESTORE_CONTEXT      ; Go reset user AST/SFM enables
                       01   BA  074B          POPR    #^M<R0>                 ; Get the resignal value back and
                            04  074D          RET                             ;       return from the Final Handler
                                074E  ;
                                074E  ; Arrive here because the error was generated by a DEBUG signal.
```

```
                              074E  ;
                              074E  FINAL_2:
          1F 00000000'EF   E9 074E          BLBC    DBG$GV_CONTROL,6$             ; Only SIGNAL if testable DEBUG
    04    04 A2    03   00 ED 0755          CMPZV   #STS$V_SEVERITY,#STS$S_SEVERITY,4(R2),#STS$K_SEVERE ; Fatal ?
                       0A  13 075B          BEQL    5$                            ; If so, signal condition
    00028362 8F    04 A2   D1 075D          CMPL    4(R2),#DBG$_INTERR            ; Check for either of the two
                       0D  12 0765          BNEQ    6$                            ;  'DEBUG internal coding error'
             00028352 8F   DD 0767  5$:     PUSHL   #DBG$_SUPERDEBUG              ; Get special signal for SUPERDEBUG
           00000000'GF  01 FB 076D          CALLS   #1,G^LIB$SIGNAL               ; Alert the superdebugger
                 44 04 A2  E9 0774  6$:     BLBC    4(R2),FINAL_4                 ; Exit, but allow DEBUG to continue
                     008D  31 0778          BRW     FINAL_5                       ;       if message is 'INFORMATION'
                              077B
                              077B  FINAL_3:
    0000046C 8F    04 A2   D1 077B          CMPL    4(R2),#SS$_DEBUG              ; Is this the DEBUG exception ?
                       37  13 0783          BEQL    FINAL_4                       ; Yes -  UNWIND to DEBUG command level
      1B 00000000'EF    05 E2 0785          BBSS    #DBG$V_CONTROL_FAIL,DBG$GV_CONTROL,8$   ; Set failure flag
             00028322 8F   DD 078D          PUSHL   #DBG$_DBGERR                  ; Message blaming DEBUG for the error
                       01  DD 0793          PUSHL   #1                            ; Number of parameters
                       00  DD 0795          PUSHL   #0                            ; No facility string
                  08CE'CF  9F 0797          PUSHAB  W^DBG$OUT_MESSAGE             ; Action routine name to output message
                    08 AE  9F 079B          PUSHAB  8(SP)                         ; Address of argument list
           00000000'GF  03 FB 079E          CALLS   #3,G^SYS$PUTMSG               ; Get message formatted and output
                    5E  08 C0 07A5          ADDL2   #8,SP                         ; Remove temporary argument list
          0D 00000000'EF   E9 07A8  8$:     BLBC    DBG$GV_CONTROL,FINAL_4        ; Are we a testable DEBUG ?
             00028352 8F   DD 07AF          PUSHL   #DBG$_SUPERDEBUG              ; Yes - get special signal
           00000000'GF  01 FB 07B5          CALLS   #1,G^LIB$SIGNAL               ; Alert the superdebugger
                              07BC  ;
                              07BC  ; The messages are all out. Unless the exit flag is set, do end-of-command
                              07BC  ; processing and unwind the stack to the caller of the command processor,
                              07BC  ; (or whoever established DBG$FINAL_HANDL as the exception handler)
                              07BC  ; and return a continue code.
                              07BC  ;
                              07BC  FINAL_4:
      70 00000000'EF    04 E0 07BC          BBS     #DBG$V_CONTROL_EXIT,DBG$GV_CONTROL,FINAL_6 ; EXIT if flag is set
  50    00010000 8F     02 C5 07C4          MULL3   #2, #65536, R0                ; Get DEBUG facility code in R0
     50    000010B0 8F     C0 07CC          ADDL2   #SHR$_READERR, R0             ; Change SHR$_READERR to DBG$_READERR
                    50  04 C8 07D3          BISL2   #^X0004, R0                   ; Set the fatal bit on.
             50    04 A2   D1 07D6          CMPL    4(R2), R0                     ; If the message is DBG$_READERR
                       0A  13 07DA          BEQL    FINAL_4_1
    00028138 8F    04 A2   D1 07DC          CMPL    4(R2), #DBG$_INPREADERR ; Keypad input error?
                       09  12 07E4          BNEQ    FINAL_4_2                     ; No, continue
                              07E6  FINAL_4_1:
       14     00000000'EF  D1 07E6          CMPL    DBG$GL_READERR_CNT, #20 ; Tried 20 times, get the same error
                       57  18 07ED          BGEQ    FINAL_7                       ; Yes, force exit
                              07EF  FINAL_4_2:
           00000000'EF  00 FB 07EF          CALLS   #0,DBG$END_OF_LINE            ; Clean up DEBUG internal status
             50    08 AC   D0 07F6          MOVL    8(AP),R0                      ; Get address of mechanism array
                 0C A0     D4 07FA          CLRL    CHF$L_MCH_SAVR0(R0)           ; Make sure returned value is 0 !!
                              07FD          $UNWIND_S                             ; Unwind to caller of the routine
                              0808                                                ;         that declared this handler
                              0808  FINAL_5:
                    50  01 3C 0808          MOVZWL  #SS$_CONTINUE,R0              ; Return status "CONTINUE"
                    23  53 E8 080B          BLBS    R3,12$                        ; Restore registers on user exit
              00000000'EF  D5 080E          TSTL    DBG$GL_SCREEN_ERROR           ; Do not purge type-ahead if error msg
                       0F  12 0814          BNEQ    11$                           ;        went to a screen display
                 0B 04 A2  E8 0816          BLBS    4(R2),11$                     ; Do not purge type-ahead if severity
00000004'EF    20000000 8F C8 081A          BISL2   #RAB$M_PTA,DBG$GL_INPRAB+RAB$L_ROP      ;     is INFO or SUCCESS
```

```
   00000004'EF    80000000 8F   CA  0825  1048 11$:    BICL2   #RAB$M_CCO,DBG$GL_OUTPRAB+RAB$L_ROP     ; Un-cancel Ctrl-O
                                04  0830  1049         RET                                     ; Return to exception mechanism
                         FDA4   31  0831  1050 12$:    BRW     RETURN_TO_USER                  ; Go restore user context
                                    0834  1051
                                    0834  1052
                                    0834  1053 FINAL_6:
                    50    04 A2 D0  0834  1054         MOVL    4(R2),R0                        ; Unrecoverable error - get code
          50    03    00    04  F0  0838  1055         INSV    #4,#0,#3,R0                     ; Change severity to FATAL
                                    083D  1056         $EXIT_S R0                              ;  and take an exit.
                                    0846  1057
                                    0846  1058 FINAL_7:
       00000000'EF    10        88  0846  1059         BISB2   #DBG$M_CONTROL_EXIT,DBG$GV_CONTROL; Set Exit bit on
          50    00028128 8F     D0  084D  1060         MOVL    #DBG$_READERR, R0               ; Set the exit status
          50    03    00    04  F0  0854  1061         INSV    #4,#0,#3,R0                     ; Change severity to FATAL
                                    0859  1062         $EXIT_S R0                              ;  and take an exit.
```

```
                                  0862  PSEUDO_SIGNAL:
      00000B0E'EF      00    FB   0862          CALLS   #0,disable_ssi
00000004'EF  20000000 8F     C8   0869          BISL2   #RAB$M_PTA,DBG$GL_INPRAB+RAB$L_ROP    ; Purge type-ahead
      00000365'EF      5D    D0   0874          MOVL    FP,USER_FP                ; Save user's frame pointer
      00000369'EF      6D    D0   087B          MOVL    (FP),HANDLER              ; Save pointer to stack handler
      50   00000000'EF       9E   0882          MOVAB   DBG$RUNFRAME,R0           ; Restore
              50   04 A0     DE   0889          MOVAL   DBG$L_USER_REGS(R0),R0    ;  user registers
                  50   60    7D   088D          MOVQ    (R0),R0                   ;  R0 and R1.
              6D    A1'AF    9E   0890          MOVAB   B^LOCAL_HANDLER,(FP)      ; Set up a one-shot handler
            00028001 8F      DD   0894          PUSHL   #DBG$_NORMAL              ; Get special exception value
      00000000'GF      01    FB   089A          CALLS   #1,G^LIB$SIGNAL           ; SIGNAL back to DEBUG context
                                  08A1
                                  08A1  LOCAL_HANDLER:
                          0FFC    08A1          .WORD   ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
      00000B0E'EF      00    FB   08A3          CALLS   #0,disable_ssi
00000365'FF    00000369'EF   D0   08AA          MOVL    HANDLER,@USER_FP          ; Restore user's original handler
              50   04 AC     D0   08B5          MOVL    CHF$L_SIGARGLST(AP),R0    ; Get address of signal argument list
        04 A0    046C 8F     3C   08B9          MOVZWL  #SS$_DEBUG,4(R0)          ; Change signal name to SS$_DEBUG
              51   60   01   C3   08BF          SUBL3   #1,(R0),R1                ; Get offset to PC in SIGNAL args
      6041   00000361'EF     D0   08C3          MOVL    USER_PC,(R0)[R1]          ; Restore actual User Error PC
                     FCD4    31   08CB          BRW     PRIM_3                    ; Go save context & issue DEBUG prompt
```

```
                                         08CE        .SBTTL  DBG$OUT_MESSAGE - Write SYS$PUTMSG output to DBG$OUTPUT
                                         08CE
                                         08CE  ;++
                                         08CE  ; FUNCTIONAL DESCRIPTION:
                                         08CE  ;       This routine is called as an action routine from EXE$PUTMSG to output
                                         08CE  ;       the string that EXE$PUTMSG has just formatted.  The string is output
                                         08CE  ;       to the logical device DBG$OUTPUT, and a value of zero is returned to
                                         08CE  ;       EXE$PUTMSG preventing it from outputing the message also.
                                         08CE  ;
                                         08CE  ; CALLING SEQUENCE:
                                         08CE  ;       4(AP)   - Address of a quadword string descriptor
                                         08CE  ;
                                         08CE  ; IMPLICIT INPUTS:
                                         08CE  ;       The output RAB for DBG$OUTPUT at location DBG$GL_OUTPRAB
                                         08CE  ;
                                         08CE  ; ROUTINE VALUE:
                                         08CE  ;       R0 = 0  - To inhibit further typing of the message
                                         08CE  ;
                                         08CE  ;--
                                         08CE
                                   0000  08CE        .ENTRY  DBG$OUT_MESSAGE,^M<>
                                         08D0
                      50    04 BC   9E   08D0        MOVAB   @4(AP),R0                      ; Get address of string descriptor
             00000022'EF    60      B0   08D4        MOVW     (R0),dbg$gl_outprab+rab$w_rsz  ; Load string length into RAB
         00000028'EF    04 A0      D0   08DB        MOVL    4(R0),dbg$gl_outprab+rab$l_rbf  ; Load address of string
               3F 00000000'EF       E9   08E3        BLBC    dbg$gb_def_out,1$              ; Check if LOG file being written
             00000155'EF    21      90   08EA        MOVB    #^A"!",log_buf                 ; Put "!" into first byte of LOG buf
20    00000028'FF    00000022'EF    2C   08F1        MOVC5   dbg$gl_outprab+rab$w_rsz,@<dbg$gl_outprab+rab$l_rbf>, -
         00000156'EF    00FF 8F          08FD
                                         0905                #^A" ",#buf_siz-1,log_buf+1       ; Copy message to LOG buffer
00000022'EF    00000022'EF    01    A1   0905        ADDW3   #1,dbg$gl_outprab+rab$w_rsz,dbg$gl_lograb+rab$w_rsz ; Length
     00000028'EF    00000155'EF     9E   0911        MOVAB   log_buf,dbg$gl_lograb+rab$l_rbf ; Load address of string
                                         091C        $PUT    RAB = dbg$gl_lograb          ; Write string to LOG file
                  00000000'EF       D5   0929  1$:   TSTL    dbg$gl_screen_error          ; If errors are redirected to a screen
                           10       13   092F        BEQL    2$                           ;       display, call screen WRITE_ERROR
                  00000000'EF       9F   0931        PUSHAB  dbg$gl_outprab               ;       routine instead of $PUT to out-
             00000000'EF    01      FB   0937        CALLS   #1,dbg$scr_write_error       ;       put the message
                        10 50       E8   093E        BLBS    R0,3$                        ; On success, skip the $PUT call
                                         0941  2$:   $PUT    RAB = dbg$gl_outprab         ; Write string to DBG$OUTPUT
                        03 50       E9   094E        BLBC    R0,4$                        ; Exit if we encountered an error
                           50       D4   0951  3$:   CLRL    R0                           ; Otherwise return 0
                           04            0953        RET                                  ;
            50    03    00    04    F0   0954  4$:   INSV    #4,#0,#3,R0                  ; Change severity to FATAL
                                         0959        $EXIT_S R0                           ;  and take an exit.
```

```
                            0962          .SBTTL  DBG$CHECK_PROT  - Makes page writable
                            0962  ;++
                            0962  ; FUNCTIONAL DESCRIPTION:
                            0962  ;         Probes a single byte to see whether it can be written.  If it can,
                            0962  ;       the value dbg$k_no_reset is returned.  Otherwise, the SETPRT system
                            0962  ;       service is called to change the protection of the page in which the
                            0962  ;       byte is located to user read/write and we return dbg$k_reset_prt as
                            0962  ;       the value of this routine.  If the system service returns an error,
                            0962  ;       zero is returned.
                            0962  ;
                            0962  ; CALLING SEQUENCE:
                            0962  ;       4(AP)   - The address of the byte to make writeable
                            0962  ;       8(AP)   - The address of a byte in which the system service can put
                            0962  ;                 the previous protection of the page
                            0962  ;
                            0962  ; IMPLICIT INPUTS:
                            0962  ;       NONE
                            0962  ;
                            0962  ; IMPLICIT OUTPUTS:
                            0962  ;       NONE
                            0962  ;
                            0962  ; ROUTINE VALUE:
                            0962  ;       dbg$k_no_reset  - Protection was already writeable, no change made
                            0962  ;       dbg$k_reset_prt - Protection changed, old protection stored
                            0962  ;       0               - Error in system service call
                            0962  ;
                            0962  ; SIDE EFFECTS:
                            0962  ;       NONE
                            0962  ;--
                            0962
                      0000  0962          .ENTRY  DBG$CHECK_PROT,^M<>
                            0964
  04 BC   01   00   0D      0964          PROBEW  #0,#1,@4(AP)            ; See if this byte can be written
               04   13      0969          BEQL    1$                      ; No, must change protection
          50   01   D0      096B          MOVL    #dbg$k_no_reset,R0      ; Yes, set return value
                    04      096E          RET                             ;  to no change, and return
                            096F
             04 AC  DD      096F  1$:     PUSHL   4(AP)                   ; Get address passed as argument
             04 AC  DD      0972          PUSHL   4(AP)                   ; Same address for end of area
          50   5E   D0      0975          MOVL    SP,R0                   ; Save stack address
                            0978          $SETPRT_S INADR=(R0),PROT=#prt$c_uw,PRVPRT=@8(AP); Change protection
             04 50  E9      098E          BLBC    R0,2$                   ; Return error if service failed
          50   02   D0      0991          MOVL    #dbg$k_reset_prt,R0     ; Service succeeded, set return value
                    04      0994          RET                             ; And return
                            0995
               50   D4      0995  2$:     CLRL    R0                      ; Set error return value
                    04      0997          RET
                            0998
```

```
                            0998        .SBTTL  DBG$REDO_PROT   - Sets page to read only
                            0998  ;++
                            0998  ; FUNCTIONAL DESCRIPTION:
                            0998  ;       Sets the protection of a page to a specified protection.
                            0998  ;
                            0998  ; CALLING SEQUENCE:
                            0998  ;       4(AP)   - Address of byte whose protection must be changed
                            0998  ;       8(AP)   - Address of byte which contains the new protection
                            0998  ;
                            0998  ; IMPLICIT INPUTS:
                            0998  ;       NONE
                            0998  ;
                            0998  ; OUTPUTS:
                            0998  ;       NONE
                            0998  ;
                            0998  ; IMPLICIT OUTPUTS:
                            0998  ;       NONE
                            0998  ;
                            0998  ; ROUTINE VALUE:
                            0998  ;       NONE
                            0998  ;
                            0998  ; SIDE EFFECTS:
                            0998  ;       SIGNAL ''DBG$_NOWPROT'' if page cannot be write protected.
                            0998  ;--
                            0998
                      0000  0998        .ENTRY  DBG$REDO_PROT,^M<>       ; Null entry mask
                            099A
             04 AC    DD    099A        PUSHL   4(AP)                    ; Get address passed as argument
             04 AC    DD    099D        PUSHL   4(AP)                    ; Same address for end of area
          50    5E    D0    09A0        MOVL    SP,R0                    ; Save stack address
      51     08 BC    9A    09A3        MOVZBL  @8(AP),R1                ; Get protection of this page
                            09A7        $SETPRT_S INADR=(R0),PROT=R1     ; Reset protection
             0D 50    E8    09B8        BLBS    R0,1$                    ; Return if service succeeded
       000284C4 8F    DD    09BB        PUSHL   #dbg$_nowprot            ; If not, tell user that protection
  00000000'GF    01   FB    09C1        CALLS   #1,G^LIB$SIGNAL          ; Resetting did not work
                      04    09C8  1$:   RET                              ; And return
```

```
                            09C9  1216 fix_up_addresses:
     0000001F'EF    F92A CF   9E  09C9  1217         MOVAB   term_handler,fix_1       ; [TEMP]
 00000027'EF   00000017'EF    9E  09D2  1218         MOVAB   term_reason,fix_2        ; [TEMP]
     0000002F'EF    F8D0 CF   9E  09DD  1219         MOVAB   restore_context,fix_3    ; [TEMP]
 00000037'EF   00000017'EF    9E  09E6  1220         MOVAB   term_reason,fix_4        ; [TEMP]
 00000000'EF   00000000'EF    9E  09F1  1221         MOVAB   dbg$runframe,dbg$gl_runframe
                              05  09FC  1222         RSB
                                  09FD  1223
```

```
09FD  ; *****SSI
09FD  ;
09FD  ; Abstract:
09FD  ;
09FD  ; In VAX DEBUG, watchpoints are implemented by write-protecting the page
09FD  ; containing the watched variable.  An access violation on that page signals
09FD  ; to DEBUG that the watched variable may be changed.  The problem with this
09FD  ; implementation is that it can cause system services that write to locations
09FD  ; on the write-protected page to fail.
09FD  ;
09FD  ; The way we have solved this problem is to intercept system services.  We have
09FD  ; changed the System Service vector to jump into intercept code, which then
09FD  ; calls a interception DEBUG routine.  In the DEBUG routine, we unprotect the
09FD  ; write-protected page and set bit 15 of the saved PSW in the system service
09FD  ; call frame on the stack. The system service then executes.  When the system
09FD  ; service returns, the bit in the saved PSW caused a reserved operand fault
09FD  ; which DEBUG catches.  DEBUG can then check for changes to the watched
09FD  ; variables and reset the page protections.
09FD  ;
09FD  ; In DEBUG, the code itself to take to implement this scheme is not much.
09FD  ; The difficulty is to put all the interactions together and make it all to
09FD  ; work properly.  Both DEBUG and System Service Intercept code are highly
09FD  ; re-entriant in an unpredicatable way.  System service can originated
09FD  ; from user program, from DBG/TDBG or from SDBG. System service can call
09FD  ; system service.  DEBUG has it own events (Break points, Stepping, Go,
09FD  ; RET, etc.) at DBG/TDBG and SDBG levels.  The communication between the levels
09FD  ; is important (in the past, TDBG/SDBG acts quite seperately), the orderring of
09FD  ; the instruction sequences is important.  It is likely things are working
09FD  ; fine in 2 levels' interactions (user and DBG), one should really test
09FD  ; 3 levels interactions by hand (user, TDBG and SDBG).
09FD  ;
09FD  ; DBGSSISHR.EXE is a priveleged shareable image at the moment which sets up
09FD  ; the system service interception.  This is the communication path between
09FD  ; user program, DBG/TDBG, and SDBG.  It is important to call this image
09FD  ; to find out what are the others doing at the moment and to tell the others
09FD  ; what am I doing at the moment.  It is not necessary to activate the
09FD  ; interception each time this image is called.  This image is intended to be
09FD  ; an unsupported system service in the future (or never will be).
09FD  ; In the meantime, this image is part of the DEBUG source.  This image runs
09FD  ; in kernal mode, so any changes made to this image should be tested on
09FD  ; stand alone machine.
09FD  ;
09FD  ;
09FD  ; There are things still not working correctly:
09FD  ;
09FD  ; 1. If one sets watch points in both TDBG and SDBG (interceptions are
09FD  ;    active), system service is originated from user:
09FD  ;
09FD  ;    user         dbg_ssi_routine            sdbg_ssi_routine
09FD  ;    sys$xxx  --> sees it (set bit 15)  --> sees it (bit 15 set)  -->:
09FD  ;                                                                    :
09FD  ;             <-- RET      (bit 15 off,  <-- RET      (bit 15 set)  <--V
09FD  ;                            t-bit over)
09FD  ;
09FD  ;    At each level, normal DEBUG interactions are going on.  The problem
09FD  ;    here is after sdbg_ssi_routine is called, bit 15 is off (where it
09FD  ;    should have been set), SDBG went into a loop.
```

```
                                    09FD  ;
                                    09FD  ;  2. If one sets same watch point on a user variable from both TDBG and SDBG.
                                    09FD  ;     This variable is changed by a system service originated from user.
                                    09FD  ;     Because the problem I mentioned above, I have changed the picture
                                    09FD  ;     a bit, so the SDBG won't go into a loop:
                                    09FD  ;
                                    09FD  ;     user          dbg_ssi_routine                sdbg_ssi_routine
                                    09FD  ;     sys$xxx  -->  sees it (set bit 15)   -->  sees it (pass)          -->|
                                    09FD  ;                                                                         |
                                    09FD  ;              <-- RET       (bit 15 off,  <-- RET       (pass)          <--V
                                    09FD  ;                             t-bit over)
                                    09FD  ;
                                    09FD  ;      Now the problem is not be able to report the watched variable correctly
                                    09FD  ;      in this case.  (I consider this is a feature for now).
                                    09FD  ;
                                    09FD  ;  3. One can step into user program from SDBG level, this can confuse the
                                    09FD  ;     things.
                                    09FD  ;
                                    09FD  ;  4. Unknown problems?
                                    09FD  ;
                                    09FD
                                    09FD  ; Main interface to enable the intercept system service.
                                    09FD  ;
                              0000  09FD          .ENTRY  ENABLE_SSI,^M<>
         00000000'8F    01    D1    09FF          cmpl    #1,#dbg$gl_3b_system      ; VMS 4 system? (link flag check)
                        01    13    0A06          beql    enable_ssi_3b             ; Yes, next check
                              04    0A08          ret                               ; No, simply return
                                    0A09  enable_ssi_3b:
         00000000'8F    01    D1    0A09          cmpl    #1,#dbg$gl_setssi         ; VMS 4 system linked with
                                    0A10                                            ;  DBGSSISHR.EXE? (link flag check)
                        01    13    0A10          beql    enable_start              ; Yes, things start to happen
                              04    0A12          ret                               ; No, simply return
                                    0A13  ENABLE_START:
                        01    BB    0A13          PUSHR   #^M<R0>                   ; R0 is used randomly in DBGSTART
                                    0A15                                            ;  for purpose, save it to be safe
              00000004'EF     D4    0A15          CLRL    DBG_ONCE_ONLY_CNT         ; Since we are leaving DEBUG, clear
                                    0A1B                                            ;  this re-entriant count
              00000BC9'EF     16    0A1B          JSB     TRIGGER_SSI               ; Is any watch point active?
     6C 00000000'EF     01    E0    0A21          BBS     #DBG$V_CONTROL_SDBG,DBG$GV_CONTROL,2$; If SDBG is running
                                    0A29
                                    0A29  ; DBG/TDBG is running now.
                                    0A29  ;
00000008'EF     03000300 8F   D0    0A29          MOVL    #^X03000300,DBG_SETUP     ; Initialize the variable, from
                                    0A34                                            ;  left to right, user mode, not
                                    0A34                                            ;  active, priority 3, SSI disabled.
              00000000'EF     D5    0A34          TSTL    DBG_SSI_CNT               ; Have we intercepted before?
                                    0A3A                                            ;  (DBG_SSI_ROUTINE called?)
                        0B    13    0A3A          BEQL    1$                        ; No
00000015'EF     00000016'EF   88    0A3C          BISB2   SAVE_SSI_STATE,DBG$GV_SSI_CONTROL; Yes, merge (OR) running
                                    0A47                                            ;  status from intercept code in P0,
                                    0A47                                            ;  we got to make sure the info. has
                                    0A47                                            ;  flowed through all levels during
                                    0A47                                            ;  one service call, regardless the
                                    0A47                                            ;  interactions happen in the back.
0000000A'EF     00000015'EF   88    0A47  1$:     BISB2   DBG$GV_SSI_CONTROL,DBG_SETUP+2; Merge (OR) again into state
                                    0A52                                            ;  vector 3rd byte now
     09 00000014'EF             E8  0A52          BLBS    DBG$GB_SET_SSI_CNT,11$    ; Any watch pointing active at this
```

```
                                  0A59                                            ; level?
       0000000A'EF     04    8A   0A59         BICB2   #DBG$M_SSI_ROUTINE_3,DBG_SETUP+2; No, Tell the 3rd bit in 3rd
                                  0A60                                            ; byte in state vector that priority
                                  0A60                                            ; 3 is not active
                       07    11   0A60         BRB     12$
       0000000A'EF     04    88   0A62  11$:   BISB2   #DBG$M_SSI_ROUTINE_3,DBG_SETUP+2; Yes, Tell the 3rd bit in 3rd
                                  0A69                                            ; byte in state vector that priority
                                  0A69                                            ; 3 is active
00000008'EF   00000014'EF    88   0A69  12$:   BISB2   DBG$GB_SET_SSI_CNT,DBG_SETUP; Now, finally set the enable/disable
                                  0A74                                            ; bit depending on whether the watch
                                  0A74                                            ; point is set or not
              00000008'EF    9F   0A74         PUSHAB  SAVE_STATE                 ; Save old state, not used here
              00000000'EF    9F   0A7A         PUSHAB  DBG_ROUTINE_ID             ; Must keep this ID around, retunred
                                  0A80                                            ; ID value from SSI_USS
              00000C22'GF    9F   0A80         PUSHAB  G^DBG_SSI_ROUTINE          ; user supplied routine to be called
                                  0A86                                            ; at the time system service is
                                  0A86                                            ; intercepted
              00000008'EF    DD   0A86         PUSHL   DBG_SETUP                  ; Enable/Disable DEBUG routine
       00000000'GF     04    FB   0A8C         CALLS   #4,G^SSI_USSU              ; Invoke routine in privileged library
                                  0A93                                            ; to setup intercept system service
                       6A    11   0A93         BRB     4$                         ; Join the common code
                                  0A95
                                  0A95
                                  0A95  ; SDBG is running now.
                                  0A95  ;
                                  0A95  2$:
0000000C'EF   03000400 8F    D0   0A95         MOVL    #^X03000400,SDBG_SETUP     ; Initialize the variable, from left
                                  0AA0                                            ; to right, user mode, not active,
                                  0AA0                                            ; priority 4, SSI disabled
              00000000'EF    D5   0AA0         TSTL    DBG_SSI_CNT                ; Have intercept before?
                       0B    13   0AA6         BEQL    3$                         ; No
00000015'EF   00000016'EF    88   0AA8         BISB2   SAVE_SSI_STATE,DBG$GV_SSI_CONTROL; Yes, merge (OR) running
                                  0AB3                                            ; status from intercept code in P0
0000000E'EF   00000015'EF    88   0AB3  3$:    BISB2   DBG$GV_SSI_CONTROL,SDBG_SETUP+2; Merge (OR) again into state
                                  0ABE                                            ; vector
           09 00000014'EF    E8   0ABE         BLBS    DBG$GB_SET_SSI_CNT,31$     ; Any watch pointing active at this
                                  0AC5                                            ; level?
       0000000E'EF     08    8A   0AC5         BICB2   #DBG$M_SSI_ROUTINE_4,SDBG_SETUP+2; No
                       07    11   0ACC         BRB     32$
       0000000E'EF     08    88   0ACE  31$:   BISB2   #DBG$M_SSI_ROUTINE_4,SDBG_SETUP+2; Yes
0000000C'EF   00000014'EF    88   0AD5  32$:   BISB2   DBG$GB_SET_SSI_CNT,SDBG_SETUP; Finally, enable/disable SSI
              00000008'EF    9F   0AE0         PUSHAB  SAVE_STATE                 ; Save old state, not used here
              00000004'EF    9F   0AE6         PUSHAB  SDBG_ROUTINE_ID            ; Must keep this ID around, retunred
                                  0AEC                                            ; ID value from SSI_USS
              00000C22'GF    9F   0AEC         PUSHAB  G^DBG_SSI_ROUTINE          ; user supplied routine to be called
                                  0AF2                                            ; at the time system service is
                                  0AF2                                            ; intercepted
              0000000C'EF    DD   0AF2         PUSHL   SDBG_SETUP                 ; Enable/Disable SDBG routine
       00000000'GF     04    FB   0AF8         CALLS   #4,G^SSI_USSU              ; Invoke routine in privileged library
                                  0AFF                                            ; to setup intercept system service
                                  0AFF  4$:
                    09 50    E8   0AFF         BLBS    R0,5$                      ; Test to see if SSI_USS failed
                                  0B02         $EXIT_S R0                         ; Yes, out!!!
                                  0B0B
                                  0B0B  5$:
                       01    BA   0B0B         POPR    #^M<R0>                    ; Pop R0, RET
                             04   0B0D         RET
```

```
                               0B0E
                               0B0E   ; Main interface to disable the intercept system service.
                               0B0E   ; NOTE: DEBUG interception routine is always not active.  SSI is enabled
                               0B0E   ; only if there is a watch point set.
                               0B0E   ;
                         0000  0B0E           .ENTRY  DISABLE_SSI,^M<>
        00000000'8F    01   D1 0B10           cmpl    #1,#dbg$gl_3b_system     ; VMS 4 system? (link flag check)
                       01   13 0B17           beql    disable_ssi_3b           ; Yes, next check
                            04 0B19           ret                              ; No, simply return
                               0B1A   disable_ssi_3b:
        00000000'8F    01   D1 0B1A           cmpl    #1,#dbg$gl_setssi        ; VMS 4 system linked with
                               0B21                                            ;  DBGSSISHR.EXE? (link flag check)
                       01   13 0B21           beql    disable_start            ; Yes, things start to happen
                            04 0B23           ret                              ; No, simply return
                               0B24   DISABLE_START:
                       03   BB 0B24           PUSHR   #^M<R0,R1>
              00000004'EF   D6 0B26           INCL    DBG_ONCE_ONLY_CNT        ; Keep track of re-entriant times
              00000BC9'EF   16 0B2C           JSB     TRIGGER_SSI              ; Is any watch point active?
     37 00000000'EF    01   E0 0B32           BBS     #DBG$V_CONTROL_SDBG,DBG$GV_CONTROL,2$
                               0B3A                                            ; If SDBG is running
                               0B3A
                               0B3A   ; DBG or TDBG is running
00000008'EF   03000300 8F   D0 0B3A           MOVL    #^X03000300,DBG_SETUP    ; Initialize the variable, from
                               0B45                                            ;    left to right, user mode, not
                               0B45                                            ;    active, priority 3, SSI disabled
00000008'EF   00000014'EF   88 0B45           BISB2   DBG$GB_SET_SSI_CNT,DBG_SETUP; Enable/Disable SSI depending on
                               0B50                                            ; whether watch point is set or not
              00000008'EF   9F 0B50           PUSHAB  SAVE_STATE               ; Save the old state, must remember
                               0B56                                            ;  the state when DEBUG is first time
                               0B56                                            ;  entered
              00000000'EF   9F 0B56           PUSHAB  DBG_ROUTINE_ID           ; Must keep this ID around, retunred
              00000C22'GF   9F 0B5C           PUSHAB  G^DBG_SSI_ROUTINE        ; user supplied routine to be called
                               0B62                                            ;  at the time system service is
                               0B62                                            ;  intercepted
                               0B62                                            ; ID value from SSI_USS
              00000008'EF   DD 0B62           PUSHL   DBG_SETUP                ; Setup the SSI_USS
        00000000'GF    04   FB 0B68           CALLS   #4,G^SSI_USSU            ; Invoke routine in privileged library
                               0B6F                                            ;  to setup intercept system service
                       35   11 0B6F           BRB     4$
                               0B71
                               0B71   ; SDBG is running
                               0B71   2$:
0000000C'EF   03000400 8F   D0 0B71           MOVL    #^X03000400,SDBG_SETUP   ; Initialize the variable, from left
                               0B7C                                            ;   to right, user mode, not active,
                               0B7C                                            ;   priority 4, SSI disabled
0000000C'EF   00000014'EF   88 0B7C           BISB2   DBG$GB_SET_SSI_CNT,SDBG_SETUP; Enable SSI if watch point set
              00000008'EF   9F 0B87           PUSHAB  SAVE_STATE               ; Save the old state, must
              00000004'EF   9F 0B8D           PUSHAB  SDBG_ROUTINE_ID          ; Must keep this ID around, retunred
                               0B93                                            ;  ID value from SSI_USS
              00000C22'GF   9F 0B93           PUSHAB  G^DBG_SSI_ROUTINE        ; user supplied routine to be called
                               0B99                                            ;  at the time system service is
                               0B99                                            ;  intercepted
              0000000C'EF   DD 0B99           PUSHL   SDBG_SETUP               ; Setup the SSI_USS
        00000000'GF    04   FB 0B9F           CALLS   #4,G^SSI_USSU            ; Invoke routine in privileged library
                               0BA6                                            ;  to setup intercept system service
                               0BA6   4$:
                    09 50   E8 0BA6           BLBS    R0,5$                    ; Test to see if SSI_USS failed
```

```
                                    0BA9              $EXIT_S R0                        ; Yes, out!!!
                                    0BB2       5$:
            01  00000004'EF    D1   0BB2              CMPL    DBG_ONCE_ONLY_CNT,#1       ; First time enter DEBUG
                          0B   12   0BB9              BNEQ    6$                         ; No
  00000015'EF  00000008'EF     90   0BBB              MOVB    SAVE_STATE,DBG$GV_SSI_CONTROL; Yes, remember its original state
                                    0BC6       6$:
                          03   BA   0BC6              POPR    #^M<R0,R1>
                               04   0BC8              RET
                                    0BC9
                                    0BC9       ; This routine is used to determine whether a watch point is set or not.
                                    0BC9       ;
                                    0BC9       TRIGGER_SSI::
                 00000014'EF   94   0BC9              CLRB    DBG$GB_SET_SSI_CNT         ; Assume there is no watch point active
                 00000000'EF   D5   0BCF              TSTL    EVENT$PAGE_QUEUE           ; A list of watched pages is there?
                          4A   13   0BD5              BEQL    3$                         ; No
  00000010'EF  00000000'EF     D0   0BD7              MOVL    EVENT$PAGE_QUEUE,PAGE_ENTRY; Loop through the list
            50  00000000'EF    9E   0BE2       1$:    MOVAB   EVENT$PAGE_QUEUE,R0        ;
            50  00000010'EF    D1   0BE9              CMPL    PAGE_ENTRY,R0              ; End of list?
                          13   13   0BF0              BEQL    2$                         ; Yes
                 00000014'EF   96   0BF2              INCB    DBG$GB_SET_SSI_CNT         ; There is one watch point
  00000010'EF  00000010'FF     D0   0BF8              MOVL    @PAGE_ENTRY,PAGE_ENTRY     ; Next
                          DD   11   0C03              BRB     1$
                 00000014'EF   95   0C05       2$:    TSTB    DBG$GB_SET_SSI_CNT         ; Any watched page found?
                          14   13   0C0B              BEQL    3$                         ; No
       00000014'EF        01   90   0C0D              MOVB    #1,DBG$GB_SET_SSI_CNT      ; Yes, flag it
                 0000000C'EF   9F   0C14              PUSHAB  DUMMY
       00000000'GF        01   FB   0C1A              CALLS   #1,G^SSI_USSK              ; Call the Kernel mode routine to
                               05   0C21       3$:    RSB                                ;  Set up the interception, if
                                    0C22                                                 ;  interception is already setup
                                    0C22                                                 ;  call simply returns
                                    0C22
```

```
                                 0C22   ; DEBUG Interception routine.
                                 0C22   ;
                          0000   0C22            .ENTRY  DBG_SSI_ROUTINE,^M<>
                                 0C24   DBG$PSEUDO_SSI::
       6D   00000C61'EF    DE    0C24            MOVAL   DBG_SSI_ROUTINE_HANDLER,(FP)
                                 0C2B                                              ; Declare its own stack handler
                                 0C2B                                              ;   so primary handler can resignal
                                 0C2B                                              ;   DBG$_NORMAL, so stack handler
                                 0C2B                                              ;   has a chance to catch this signal
                                 0C2B                                              ;   and change it to DBG$_SS_INT
            00000000'EF    D6    0C2B            INCL    DBG_SSI_CNT               ; Mark the fact we see one
   00000016'EF    18 AC    90    0C31            MOVB    24(AP),SAVE_SSI_STATE     ; Get the communication variable
                                 0C39                                              ;   at the time this routine is called
                  14 AC    DD    0C39            PUSHL   20(AP)                    ; Pass in Address of the system service
                                 0C3C                                              ;   count (this system service may be
                                 0C3C                                              ;   nested)
                  10 AC    DD    0C3C            PUSHL   16(AP)                    ; Pass in Address of the system service
                                 0C3F                                              ;   RET count (this system service may
                                 0C3F                                              ;   return from SDBG, then TDBG)
                  0C AC    DD    0C3F            PUSHL   12(AP)                    ; FP of the system service
                  08 AC    DD    0C42            PUSHL   8(AP)                     ; AP of the system service
                  04 AC    DD    0C45            PUSHL   4(AP)                     ; System service index
                     05    DD    0C48            PUSHL   #5                        ; # of arguments
            00028001 8F    DD    0C4A            PUSHL   #DBG$_NORMAL              ; A way to get back into DEBUG
       00000000'GF   07    FB    0C50            CALLS   #7,G^LIB$SIGNAL           ; Signal it.
                 50  01    D0    0C57            MOVL    #1,R0                     ; Interception done, return
            00000016'EF    94    0C5A            CLRB    SAVE_SSI_STATE            ; No use for the communication variable
                                 0C60                                              ;   clear it, make sure there is no
                                 0C60                                              ;   side facts after using it
                           04    0C60            RET
                                 0C61
                                 0C61   ; DEBUG Interception routine handler
                                 0C61   ;
                          0000   0C61            .ENTRY  DBG_SSI_ROUTINE_HANDLER,^M<>
                     F5B9  30    0C63            BSBW    SAVE_USER_CONTEXT_ALWAYS; Save user context
               52   04 AC  D0    0C66            MOVL    CHF$L_SIGARGLST(AP),R2    ; Change DBG$_NORMAL to
   04 A2    00028793 8F    D0    0C6A            MOVL    #DBG$_SS_INT,4(R2)        ; DBG$_SSI_INT
                     F930  31    0C72            BRW     PRIM_4                    ; Act as though its a normal exception
                                 0C75
                                 0C75            .END    beginhere
```

```
$$.TMP1                        = 00000001
$$.TMP2                        = 000000EF
$$T1                           = 00000000
BEGINHERE                        00000000 RG    08
BUF_SIZ                        = 00000100
CHF$L_MCHARGLST                = 00000008
CHF$L_MCH_FRAME                = 00000004
CHF$L_MCH_SAVR0                = 0000000C
CHF$L_SIGARGLST                = 00000004
CHF$L_SIG_ARGS                 = 00000000
CHF$L_SIG_NAME                 = 00000004
CLI$A_IMGFILED                 = 00000010
CLI$A_IMGHDADR                 = 0000000C
CLI$V_DBGEXCP                  = 00000010
CONST_0                          00000355 R     06
CONST_1                          00000359 R     06
DATA                             00000000 R     05
DBG$B_BPT_INS                    00000060
DBG$B_PREV_PRO1                  0000005E
DBG$B_PREV_PRO2                  0000005F
DBG$B_USER_OPCO                  00000040
DBG$CHECK_PROT                   00000962 RG    08
DBG$C_RUNFR_LEN                  00000065
DBG$END_OF_LINE                  ********   X   00
DBG$EXCEPTION_IS_FAULT           ********   X   08
DBG$EXC_HANDLER                  ********   X   00
DBG$FINAL_HANDL                  0000065B RG    08
DBG$FLUSHBUF                     ********   X   00
DBG$GB_CALL_NORMAL_RET           ********   X   00
DBG$GB_DEF_OUT                   ********   X   00
DBG$GB_SET_SSI_CNT               00000014 RG    06
DBG$GB_UNHANDLED_EXC             ********   X   00
DBG$GL_3B_SYSTEM                 ********   X   00
DBG$GL_EXIT_STATUS               0000036D RG    06
DBG$GL_INPRAB                    ********   X   00
DBG$GL_LOGRAB                    ********   X   00
DBG$GL_OUTPRAB                   ********   X   00
DBG$GL_READERR_CNT               ********   X   08
DBG$GL_RUNFRAME                  00000000 RG    03
DBG$GL_SCREEN_ERROR              ********   X   08
DBG$GL_SCREEN_MODE               ********   X   08
DBG$GL_SETSSI                    ********   X   00
DBG$GV_CONTROL                   ********   X   00
DBG$GV_SSI_CONTROL               00000015 RG    06
DBG$INIT_DEBUG                   ********   X   00
DBG$INS_OPCODES                  ********   X   00
DBG$K_NO_RESET                 = 00000001
DBG$K_RESET_PRT                = 00000002
DBG$K_RUNFR_LEN                  00000065
DBG$L_BPT_PC                     0000004A
DBG$L_CALL_ADDR                  00000052
DBG$L_FRAME_PTR                  0000004E
DBG$L_NEXT_LINK                  00000000
DBG$L_SAVE_FLD                   00000061
DBG$L_USER_AP                    00000034
DBG$L_USER_FP                    00000038
DBG$L_USER_PC                    00000040
DBG$L_USER_PSL                   00000044
DBG$L_USER_R0                    00000004
DBG$L_USER_R1                    00000008
DBG$L_USER_R10                   0000002C
DBG$L_USER_R11                   00000030
DBG$L_USER_R2                    0000000C
DBG$L_USER_R3                    00000010
DBG$L_USER_R4                    00000014
DBG$L_USER_R5                    00000018
DBG$L_USER_R6                    0000001C
DBG$L_USER_R7                    00000020
DBG$L_USER_R8                    00000024
DBG$L_USER_R9                    00000028
DBG$L_USER_REGS                  00000004
DBG$L_USER_SP                    0000003C
DBG$L_WATCHPT                    00000056
DBG$L_WATCHPTEN                  0000005A
DBG$M_CONTROL_ALLOCATE         = 00000080
DBG$M_CONTROL_DONE             = 00000040
DBG$M_CONTROL_EXIT             = 00000010
DBG$M_CONTROL_FAIL             = 00000020
DBG$M_CONTROL_KDBG             = 00000004
DBG$M_CONTROL_SCREEN           = 00000800
DBG$M_CONTROL_SDBG             = 00000002
DBG$M_CONTROL_STOP             = 00000200
DBG$M_CONTROL_TBIT             = 00000400
DBG$M_CONTROL_TDBG             = 00000001
DBG$M_CONTROL_URUN             = 00000008
DBG$M_CONTROL_USER             = 00000100
DBG$M_CONTROL_VERSION_4        = 00001000
DBG$M_ENAB_AST                 = 00000020
DBG$M_ENAB_FEX                 = 00000800
DBG$M_SSI_ROUTINE_1            = 00000001
DBG$M_SSI_ROUTINE_2            = 00000002
DBG$M_SSI_ROUTINE_3            = 00000004
DBG$M_SSI_ROUTINE_4            = 00000008
DBG$NEWLINE                      ********   X   00
DBG$OUT_MESSAGE                  000008CE RG    08
DBG$OUT_NUM_VAL                  ********   X   00
DBG$PRINT                        ********   X   00
DBG$PSEUDO_EXIT                  0000043F RG    08
DBG$PSEUDO_PROG                  00000434 RG    08
DBG$PSEUDO_SSI                   00000C24 RG    08
DBG$PUTMSG                       ********   X   00
DBG$REDO_PROT                    00000998 RG    08
DBG$REL_MEMORY                   ********   X   00
DBG$RST_INIT                     ********   X   00
DBG$RUNFRAME                     ********   X   00
DBG$SCR_SCREEN_TERM              ********   X   00
DBG$SCR_WRITE_ERROR              ********   X   08
DBG$TERM_HANDLR                  000002F9 RG    08
DBG$THREAD_BPT                   00000615 RG    08
DBG$THREAD_RET                   00000632 RG    08
DBG$USER_EXIT                    0000018A RG    08
DBG$V_AT_FAULT                 = 0000000D
DBG$V_CONTROL_ALLOCATE         = 00000007
DBG$V_CONTROL_DONE             = 00000006
```

```
DBG$V_CONTROL_EXIT             = 00000004                FIX_4                          00000037 R       06
DBG$V_CONTROL_FAIL             = 00000005                FIX_UP_ADDRESSES               000009C9 R       08
DBG$V_CONTROL_KDBG             = 00000002                HANDLER                        00000369 R       06
DBG$V_CONTROL_SCREEN           = 0000000B                IFD$W_CHAN                   = 00000008
DBG$V_CONTROL_SDBG             = 00000001                IFD$W_FILNAMOFF              = 00000002
DBG$V_CONTROL_STOP             = 00000009                LAST_CHANCE                    000003CC R       08
DBG$V_CONTROL_TBIT             = 0000000A                LIB$SIGNAL                     ********    X   00
DBG$V_CONTROL_TDBG             = 00000000                LOCAL_HANDLER                  000008A1 R       08
DBG$V_CONTROL_URUN             = 00000003                LOG_BUF                        00000155 R       06
DBG$V_CONTROL_USER             = 00000008                MSG_LENGTH                     00000053 R       06
DBG$V_CONTROL_VERSION_4        = 0000000C                ONE_SHOT_HANDLER               000000B1 R       08
DBG$V_ENAB_AST                 = 00000005                PAGE_ENTRY                     00000010 RG      06
DBG$V_ENAB_FEX                 = 0000000B                PARAM_0                        00000359 R       06
DBG$V_SSI_ROUTINE_1            = 00000000                PARAM_1                        0000035D R       06
DBG$V_SSI_ROUTINE_2            = 00000001                PRIMARY_HANDLER                00000537 RG      08
DBG$V_SSI_ROUTINE_3            = 00000002                PRIM_3                         000005A2 R       08
DBG$V_SSI_ROUTINE_4            = 00000003                PRIM_4                         000005A5 R       08
DBG$W_RUN_STAT                   00000048                PRIM_HANDL_2                   00000539 RG      08
DBG$_DBGERR                    = 00028322                PRT$C_UW                       ********    X   00
DBG$_EXITSTATUS                = 0002806B                PSEUDO_HANDLER                 0000047E R       08
DBG$_INPREADERR                = 00028138                PSEUDO_SIGNAL                  00000862 R       08
DBG$_INTERR                    = 00028362                PSL$V_DV                     = 00000007
DBG$_LASTCHANCE                = 00028258                PSL$V_IV                     = 00000005
DBG$_NORMAL                    = 00028001                RAB$L_RBF                    = 00000028
DBG$_NOWPROT                   = 000284C4                RAB$L_ROP                    = 00000004
DBG$_READERR                   = 00028128                RAB$M_CCO                    = 80000000
DBG$_SS_INT                    = 00028793                RAB$M_PTA                    = 20000000
DBG$_SUPERDEBUG                = 00028352                RAB$W_RSZ                    = 00000022
DBG_FACILITY                   = 00000002                RESET_DEBUG                    000001A9 R       08
DBG_ONCE_ONLY_CNT                00000004 RG      06     RESTORE_CONTEXT                000002B1 R       08
DBG_ROUTINE_ID                 = 00000000 RG      05     RETURN_TO_USER                 000005D8 R       08
DBG_SETUP                        00000008 RG      06     ROUTINE_VALUE                  00000000 R       07
DBG_SSI_CNT                      00000000 RG      06     SAVED_AP                       0000003B R       06
DBG_SSI_ROUTINE                  00000C22 RG      08     SAVED_FP                       0000003F R       06
DBG_SSI_ROUTINE_HANDLER          00000C61 RG      08     SAVED_R0                       00000043 R       06
DISABLE_SSI                      00000B0E RG      08     SAVED_R1                       00000047 R       06
DISABLE_SSI_3B                   00000B1A R       08     SAVE_SSI_STATE                 00000016 RG      06
DISABLE_START                    00000B24 R       08     SAVE_STATE                   = 00000008 RG      05
DUMMY                          = 0000000C R       05     SAVE_USER_CONTEXT              00000217 R       08
ENABLE_SSI                       000009FD RG      08     SAVE_USER_CONTEXT_ALWAYS       0000021F R       08
ENABLE_SSI_3B                    00000A09 R       08     SDBG_ROUTINE_ID              = 00000004 RG      05
ENABLE_START                     00000A13 R       08     SDBG_SETUP                     0000000C RG      06
END_WRITE_STOR                   00000000 R       04     SETUP                          000000EF R       08
EVENT$PAGE_QUEUE                 ********    X    00     SETUP_EXIT_HANDLER             000001D7 R       08
FAOBUFDESC                       0000004B R       06     SF$L_SAVE_AP                 = 00000008
FAO_BUF                          00000055 R       06     SF$L_SAVE_FP                 = 0000000C
FINAL_2                          0000074E R       08     SF$L_SAVE_PC                 = 00000010
FINAL_3                          0000077B R       08     SHR$_APPENDEDB               = 00001000
FINAL_4                          000007BC R       08     SHR$_READERR                 = 000010B0
FINAL_4_1                        000007E6 R       08     SS$_BREAK                    = 00000414
FINAL_4_2                        000007EF R       08     SS$_CLIFRCEXT                = 00000980
FINAL_5                          00000808 R       08     SS$_CONTINUE                 = 00000001
FINAL_6                          00000834 R       08     SS$_DEBUG                    = 0000046C
FINAL_7                          00000846 R       08     SS$_RESIGNAL                 = 00000918
FIX_1                            0000001F R       06     SS$_UNWIND                   = 00000920
FIX_2                            00000027 R       06     SS$_WASCLR                   = 00000001
FIX_3                            0000002F R       06     SSI_USSK                       ********W GX     00
```

```
SSI_USSU                 ********W GX    00
SSI_VAR_BEG              00000000 RG     05
SSI_VAR_END              00000200 RG     05
STS$K_SEVERE           = 00000004
STS$M_INHIB_MSG        = 10000000
STS$S_FAC_NO           = 0000000C
STS$S_SEVERITY         = 00000003
STS$V_FAC_NO           = 00000010
STS$V_SEVERITY         = 00000000
SYS$CANEXH               ********  GX    08
SYS$DCLEXH               ********  GX    00
SYS$EXIT                 ********  GX    00
SYS$GETMSG               ********  GX    00
SYS$PUT                  ********  GX    00
SYS$PUTMSG               ********   X    08
SYS$SETAST               ********  GX    00
SYS$SETEXV               ********  GX    08
SYS$SETPRT               ********  GX    00
SYS$SETSFM               ********  GX    08
SYS$UNWIND               ********  GX    00
TERM_BLOCK_ONE           0000001B R      06
TERM_BLOCK_TWO           0000002B R      06
TERM_BUF                 00000255 R      06
TERM_HANDLER             000002F7 R      08
TERM_REASON              00000017 R      06
TERM_WINDOW_HANDLER      000003FF R      08
TRIGGER_SSI              00000BC9 RG     08
USER_FP                  00000365 R      06
USER_PC                  00000361 R      06
VIRTUAL_ZERO             00000000 R      02
WINDOW_HANDLER           00000640 R      08
WRITABLE_STOR            00000000 R      03

                              +----------------+
                              ! Psect synopsis !
                              +----------------+

PSECT name          Allocation          PSECT No.  Attributes
----------          ----------          ---------  ----------
.  ABS  .           00000000  (    0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD  NOWRT NOVEC BYTE
$ABS$               00000065  (  101.)  01 (  1.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD    WRT NOVEC BYTE
DBG$ABS_ZERO        0000000C  (   12.)  02 (  2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC LONG
DBG$GLOBAL          00000004  (    4.)  03 (  3.)    PIC   USR   CON   REL   LCL NOSHR NOEXE   RD    WRT NOVEC LONG
ZZZ$ZZZZZZ          00000000  (    0.)  04 (  4.)    PIC   USR   CON   REL   LCL NOSHR NOEXE   RD    WRT NOVEC LONG
DBG$SSI             00000200  (  512.)  05 (  5.)    PIC   USR   CON   REL   LCL NOSHR NOEXE   RD    WRT NOVEC PAGE
DBG$OWN             00000371  (  881.)  06 (  6.)    PIC   USR   CON   REL   LCL NOSHR NOEXE   RD    WRT NOVEC LONG
DBG$PLIT            00000013  (   19.)  07 (  7.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC BYTE
DBG$CODE            00000C75  ( 3189.)  08 (  8.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC BYTE

                           +------------------------+
                           ! Performance indicators !
                           +------------------------+

Phase                  Page faults    CPU Time       Elapsed Time
-----                  -----------    --------       ------------
Initialization                  16    00:00:00.05    00:00:01.23
Command processing              96    00:00:00.82    00:00:03.71
```

```
Pass 1                          499      00:00:17.36     00:00:57.95
Symbol table sort                 8      00:00:02.38     00:00:07.61
Pass 2                          448      00:00:04.56     00:00:14.30
Symbol table output              31      00:00:00.24     00:00:01.40
Psect synopsis output             3      00:00:00.03     00:00:00.04
Cross-reference output            0      00:00:00.00     00:00:00.00
Assembler run totals           1103      00:00:25.46     00:01:26.39

The working set limit was 1950 pages.
95992 bytes (188 pages) of virtual memory were used to buffer the intermediate code.
There were 90 pages of symbol table space allocated to hold 1516 non-local and 63 local symbols.
1524 source lines were read in Pass 1, producing 65 object records in Pass 2.
32 pages of virtual memory were used to define 30 macros.

                                  +----------------------------+
                                  ! Macro library statistics !
                                  +----------------------------+

Macro library name                        Macros defined
------------------                        --------------
_$255$DUA28:[DEBUG.OBJ]DBGMSG.MLB;1                    1
_$255$DUA28:[SYS.OBJ]LIB.MLB;1                         1
_$255$DUA28:[SYSLIB]STARLET.MLB;2                     25
TOTALS (all libraries)                                27

1626 GETS were required to define 27 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:DBGSTART/OBJ=OBJ$:DBGSTART MSRC$:DBGSTART/UPDATE=(ENH$:DBGSTART)+EXECML$/LIB+LIB$:DBGMSG/LIB
```

0094 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

DBGSSV
LIS

DBGSTEPGO
LIS

DBGSTART
LIS

DBGSOURCE
LIS