H 5

```
   SSSSSSSSSSSS     000000000     RRRRRRRRRRRR     TTTTTTTTTTTTTTT     333333333        222222222
   SSSSSSSSSSSS     000000000     RRRRRRRRRRRR     TTTTTTTTTTTTTTT     333333333        222222222
   SSSSSSSSSSSS     000000000     RRRRRRRRRRRR     TTTTTTTTTTTTTTT     333333333        222222222
SSS              000         000  RRR         RRR        TTT        333         333  222         222
SSS              000         000  RRR         RRR        TTT        333         333  222         222
SSS              000         000  RRR         RRR        TTT        333         333  222         222
SSS              000         000  RRR         RRR        TTT                    333              222
SSS              000         000  RRR         RRR        TTT                    333              222
SSS              000         000  RRR         RRR        TTT                    333              222
   SSSSSSSSS     000         000  RRRRRRRRRRRR           TTT                 333              222
   SSSSSSSSS     000         000  RRRRRRRRRRRR           TTT                 333              222
   SSSSSSSSS     000         000  RRRRRRRRRRRR           TTT                 333              222
            SSS  000         000  RRR   RRR              TTT                    333        222
            SSS  000         000  RRR   RRR              TTT                    333        222
            SSS  000         000  RRR   RRR              TTT                    333        222
            SSS  000         000  RRR      RRR           TTT        333         333     222
            SSS  000         000  RRR      RRR           TTT        333         333     222
            SSS  000         000  RRR      RRR           TTT        333         333     222
SSSSSSSSSSSS        000000000     RRR         RRR        TTT           333333333     222222222222222
SSSSSSSSSSSS        000000000     RRR         RRR        TTT           333333333     222222222222222
SSSSSSSSSSSS        000000000     RRR         RRR        TTT           333333333     222222222222222
```

_$2

Pse

SOR

SOR

SOR

SOR

_LI

FILEID**LIBFIXUPD   D 5

```
LL            IIIIII    BBBBBBBB    FFFFFFFFFF    IIIIII    XX      XX  UU      UU  PPPPPPPP    DDDDDDDD
LL            IIIIII    BBBBBBBB    FFFFFFFFFF    IIIIII    XX      XX  UU      UU  PPPPPPPP    DDDDDDDD
LL              II      BB      BB  FF              II      XX      XX  UU      UU  PP      PP  DD      DD
LL              II      BB      BB  FF              II      XX      XX  UU      UU  PP      PP  DD      DD
LL              II      BB      BB  FF              II        XX  XX    UU      UU  PP      PP  DD      DD
LL              II      BB      BB  FF              II        XX  XX    UU      UU  PP      PP  DD      DD
LL              II      BBBBBBBB    FFFFFFFF        II          XX      UU      UU  PPPPPPPP    DD      DD
LL              II      BBBBBBBB    FFFFFFFF        II          XX      UU      UU  PPPPPPPP    DD      DD
LL              II      BB      BB  FF              II        XX  XX    UU      UU  PP          DD      DD
LL              II      BB      BB  FF              II        XX  XX    UU      UU  PP          DD      DD
LL              II      BB      BB  FF              II      XX      XX  UU      UU  PP          DD      DD     ....
LL              II      BB      BB  FF              II      XX      XX  UU      UU  PP          DD      DD     ....
LLLLLLLLLL    IIIIII    BBBBBBBB    FF            IIIIII    XX      XX  UUUUUUUUUU  PP          DDDDDDDD       ....
LLLLLLLLLL    IIIIII    BBBBBBBB    FF            IIIIII    XX      XX  UUUUUUUUUU  PP          DDDDDDDD       ....

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  LIB$FIXUP_DEC - Fixup decimal reserved operand
0000          .IDENT  /V04-000/              ; File: LIBFIXUPD.MAR Edit: PDG002
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    ;++
0000    ; FACILITY: General Utility Library
0000    ;
0000    ; ABSTRACT:
0000    ;
0000    ;       LIB$FIXUP_DEC fixes up decimal reserved operands when a
0000    ;       reserved operand fault occurs so that execution may continue
0000    ;       at that instruction or the next instruction.  It is designed
0000    ;       to be a condition handler or to be called from a condition handler.
0000    ;
0000    ; ENVIRONMENT: Runs at any access mode, AST Reentrant
0000    ;
0000    ; AUTHOR: Peter D Gilbert,       Version 1, CREATION DATE: 03-DEC-1980
0000    ;                                Adapted from LIB$FIXUP_DEC
0000    ;
0000    ; MODIFIED BY:
0000    ;
0000    ;       V02-002         PDG002          PDG     25-Oct-1983
0000    ;               Modify the source if possible.  If not, copy the sourc. before
0000    ;               attempting the instruction.  Also, store the condition codes.
0000    ;
0000    ;       V02-001         PDG001          PDG     10-Aug-1982
0000    ;               Fix a problem with searching the translation table.
0000    ;
0000    ;       V02-000         Original
0000    ;
0000    ;--
```

```
                    0000            .SBTTL  DECLARATIONS
                    0000    ;
                    0000    ; LIBRARY MACRO CALLS:
                    0000    ;
                    0000            $SFDEF                  ; Stack frame symbols
                    0000            $PSLDEF                 ; Processor Status Longword symbols
                    0000            $CHFDEF                 ; Condition handling facility symbols
                    0000            $STSDEF                 ; Status value symbols
                    0000            $SSDEF                  ; System status values
                    0000    ;
                    0000    ; EXTERNAL DECLARATIONS:
                    0000    ;
                    0000            .DSABL  GBL             ; Force all external symbols to be declared
                    0000            .EXTRN  SYS$UNWIND      ; Unwind stack frames
                    0000            .EXTRN  LIB$_BADSTA     ; Bad stack frame
                    0000            .EXTRN  SYS$CALL_HANDL  ; System routine that calls handlers
                    0000    ;
                    0000    ; MACROS:
                    0000    ;
                    0000            NONE
                    0000    ;
                    0000    ; EQUATED SYMBOLS:
                    0000    ;
          00000000  0000            R0_OFF  = 0*4                   ; R0 register offset in register image
          00000004  0000            R1_OFF  = 1*4                   ; R1 register offset
          00000008  0000            R2_OFF  = 2*4                   ; R2 register offset
          0000000C  0000            R3_OFF  = 3*4                   ; R4 register offset
          00000030  0000            AP_OFF  = 12*4                  ; AP register offset
          00000034  0000            FP_OFF  = 13*4                  ; FP register offset
          00000038  0000            SP_OFF  = 14*4                  ; SP register offset
          0000003C  0000            PC_OFF  = 15*4                  ; PC register offset
          00000040  0000            PSL_OFF = 16*4                  ; PSL offset
                    0000
          00000000  0000            STACK = 0                       ; Used by DCL macro
                    0000            .MACRO  DCL, SYM, LEN           ; Declare stack temp offsets
                    0000            STACK = STACK - 4*LEN           ; Allocate LEN longwords
                    0000            SYM =   STACK                   ; Define SYM
                    0000            .ENDM
                    0000
                    0000            DCL     REG_IMAGE, 17           ; FP offset for image vector of registers
                    0000            DCL     ADR_IMAGE, 17           ; FP offset for image vector of addresses
                    0000                                            ; where registers have been saved in stack
                    0000            DCL     OPD_IMAGE, 6            ; Addresses of operands
                    0000
          FFFFFFFC  0000            IMAGE_PSL = -4                  ; FP offset of PSL image
          FFFFFFF8  0000            IMAGE_PC = -8                   ; FP offset of PC image
                    0000
                    0000
                    0000    ; Define codes used to denote operand types in opcode/operand tables
                    0000    ; to follow.
                    0000
          00000000  0000            OP_Z    = 0                     ; No more operands to process
          00000001  0000            OP_W    = 1                     ; Word
          00000002  0000            OP_D    = 2                     ; Decimal
          00000003  0000            OP_P    = 3                     ; Packed
          00000004  0000            OP_A    = 4                     ; Address
                    0000    ;
```

```
                                0000   113 ; OWN STORAGE:
                                0000   114 ;
                            00000000   115         .PSECT _LIB$CODE PIC, USR, CON, REL, LCL, SHR, -
                                0000   116                         EXE, RD, NOWRT, LONG
                                0000   117 ;
                                0000   118
                                0000   119 ; Tables of opcodes and operand types.  The first byte in each entry
                                0000   120 ; is the opcode. The remaining bytes (up to 6) are OP_x codes defined
                                0000   121 ; above that specify what datatype each operand is for that instruction.
                                0000   122 ; If an operand type is 0, then no more operands are processed for that
                                0000   123 ; instruction.  The opcodes must be in decreasing order, and the final
                                0000   124 ; opcode byte must be a zero.
                                0000   125 ;
                                0000   126 ; Table for single byte opcodes.
                                0000   127 ;
                                0000   128 SING_TAB:
     00 00 03 01 04 02 01 26    0000   129         .BYTE   ^X26, OP_W, OP_D, OP_A, OP_W, OP_P, 0, 0 ; CVTTP
     00 00 00 03 01 02 01 09    0008   130         .BYTE   ^X09, OP_W, OP_D, OP_W, OP_P,    0, 0, 0 ; CVTSP
                          00    0010   131         .BYTE   0
                                0011   132
                                0011   133 ; Table for registers used in this instruction.
                                0011   134 ; The high order word is used for auto-increment/decrement.
                                0011   135 ; These entries must be in the same order as the SING_TAB entries.
                                0011   136 ;
                                0011   137         .ALIGN  LONG
                                0014   138 REGS_TAB:
                    7FFF000F    0014   139         .LONG   ^X7FFF000F                               ; CVTTP
                    7FFF000F    0018   140         .LONG   ^X7FFF000F                               ; CVTSP
                                001C   141
                                001C   142 ; Table of context amounts listed in OP_x code order
                                001C   143 ;
                                001C   144 OP_CONTEXT:
                          00    001C   145         .BYTE   0               ; OP_Z
                          02    001D   146         .BYTE   2               ; OP_W
                          01    001E   147         .BYTE   1               ; OP_D
                          01    001F   148         .BYTE   1               ; OP_P
                          01    0020   149         .BYTE   1               ; OP_A
                                0021   150
                                0021   151 ;
                                0021   152 ; PSECT DECLARATIONS:
                                0021   153 ;
                                0021   154         .PSECT _LIB$CODE PIC, USR, CON, REL, LCL, SHR, -
                            00000021   155                         EXE, RD, NOWRT, LONG
                                0021   156
```

```
0021        .SBTTL  LIB$FIXUP_DEC - Fixup decimal reserved operand
0021  ;++
0021  ; FUNCTIONAL DESCRIPTION:
0021  ;
0021  ;       LIB$FIXUP_DEC finds the reserved operand of the decimal instructions
0021  ;       CVTTP or CVTSP after a reserved operand fault has been signaled.
0021  ;       If possible, LIB$FIXUP_DEC will change the reserved digit(s) to "zero".
0021  ;       Otherwise, execution proceeds with the next instruction.
0021  ;
0021  ; Exceptions:
0021  ;
0021  ;       LIB$FIXUP_DEC can not handle the following cases and will return
0021  ;       a status of SS$_RESIGNAL if any of them occur.
0021  ;
0021  ;           1.  The currently active signaled condition is not SS$_ROPRAND.
0021  ;           2.  The reserved operand's datatype is not Decimal or Packed.
0021  ;
0021  ; CALLING SEQUENCE:
0021  ;
0021  ;      ret_status.wlc.v = LIB$FIXUP_DEC (chf$l_sigarglst.rl.ra,
0021  ;                                        chf$l_mcharglst.rl.ra )
0021  ;
0021  ; FORMAL PARAMETERS:
0021  ;
0021  ;       CHF$L_SIGARGLST = Address of signal argument vector.
0021  ;       CHF$L_MCHARGLST = Address of mechanism argument vector.
0021  ;
0021  ; IMPLICIT INPUTS:
0021  ;
0021  ;       The stack frames back to that of the instruction which faulted.
0021  ;       The instruction which faulted and its operands.
0021  ;
0021  ; IMPLICIT OUTPUTS:
0021  ;
0021  ;       The reserved decimal operand, if found, is replaced by "zero".
0021  ;
0021  ; COMPLETION STATUS:
0021  ;
0021  ;       SS$_CONTINUE - continue execution at point of condition
0021  ;           Routine successfully completed.  The  reserved  operand  was
0021  ;           found and was fixed up.
0021  ;
0021  ;       SS$_ACCVIO - access violation
0021  ;           An argument to LIB$FIXUP_DEC or an operand of  the  faulting
0021  ;           instruction could not be read or written.
0021  ;
0021  ;       SS$_RESIGNAL - resignal condition to next handler
0021  ;           The condition signaled  was not SS$_ROPRAND or the  reserved
0021  ;           operand  was not a decimal value.
0021  ;
0021  ;       LIB$_BADSTA - bad stack
0021  ;           The stack frame linkage had been corrupted since the time of
0021  ;           the reserved operand exception.
0021  ;
0021  ;       Note:  If the status value returned from LIB$FIXUP_DEC is seen by
0021  ;       the  condition  handling  facility,  (as  would  be  the  case if
0021  ;       LIB$FIXUP_DEC was the handler), any success value  is  equivalent
```

```
                               0021  ;        to  SS$_CONTINUE,  which  causes the instruction to be restarted.
                               0021  ;        Any failure value is equivalent to SS$_RESIGNAL, which will cause
                               0021  ;        the  condition  to  be  resignalled to the next handler.  This is
                               0021  ;        because the condition handler (LIB$FIXUP_DEC)  failed  to  handle
                               0021  ;        the condition correctly.
                               0021  ;
                               0021  ; SIDE EFFECTS:
                               0021  ;
                               0021  ;        If the reserved operand is fixed up, the instruction which
                               0021  ;        faulted is restarted.
                               0021  ;
                               0021  ;--
                               0021
                               0021  ;+
                               0021  ; Registers used:
                               0021  ;
                               0021  ;        R0 =    scratch
                               0021  ;        R1 =    scratch
                               0021  ;        R2 =    pointer into opcode/operand table
                               0021  ;        R3 =    context index or 0
                               0021  ;        R4 =    OA1 (operand address) of bits 31:0
                               0021  ;        R5 =    OA2 (operand address) of bits 63:32 which may not be
                               0021  ;                OA1+4 since registers not necessarily saved contiguously.
                               0021  ;        R6 =    register number of operand specifier
                               0021  ;        R7 =    pointer into operand image block
                               0021  ;        R8 =    scratch
                               0021  ;        R9 =    mask of registers used in operands
                               0021  ;-
                               0021
                         0FFC  0021          .ENTRY LIB$FIXUP_DEC, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
                               0023                                           ; save all registers so that all will be
                               0023                                           ; found in stack during back scan.
                               0023                                           ; disable IV (content index multiply)
             6D    59'AF   9E  0023          MOVAB   B^SIG_TO_RET, (FP)       ; Enable condition handler
             50    04 AC   D0  0027          MOVL    CHF$L_SIGARGLST(AP), R0  ; R0 = adr. of signal arg list array
0000008A 8F   04 A0  19  03  ED  002B        CMPZV   #STS$V_COND_ID, -        ; position of message identification
                               0035                  #STS$S_COND_ID, -        ; size of id
                               0035                  CHF$L_SIG_NAME(R0), -    ; compare 29-bit VAX-11 signal code
                               0035                  #<SS$_ROPRAND@-STS$V_COND_ID> ; with reserved operand code
                   1C      12  0035          BNEQ    RESIGNAL                 ; resignal the error
             5E  E338 CE   9E  0037  2$:     MOVAB   STACK(SP), SP            ; allocate stack space
                  0242     30  003C          BSBW    GET_REGS                 ; setup the two image vectors in local stora
                               003F                                           ; do not return here if error, instead RET w
                               003F                                           ; error completion status
                               003F  ;+
                               003F  ; Get instruction opcode.  Determine if this is an instruction which
                               003F  ; we can handle.  If not, resignal.  If so, load R2 with the address
                               003F  ; of the operand table entry for that opcode.
                               003F  ;-
                  0130     30  003F          BSBW    NEXT_BYTE                ; Get first opcode byte
             52  FFBA CF   9E  0042          MOVAB   W^SING_TAB, R2           ; Table base
             50    62      91  0047  3$:     CMPB    (R2), R0                 ; Is this the opcode?
                   34      13  004A          BEQL    MATCH                    ; Yes, we have a match
             52    08      C0  004C          ADDL2   #8, R2                   ; Skip to next entry
                   62      95  004F          TSTB    (R2)                     ; At end of table?
                   F4      12  0051          BNEQ    3$                       ; No, continue searching
                               0053  RESIGNAL:
```

```
          50   0918 8F    3C  0053          MOVZWL  #SS$_RESIGNAL, R0       ; We can't handle this exception, return SS$
                          04  0058          RET                             ; R0 = RESIGNAL error completion code
                        0000  0059  SIG_TO_RET:     .WORD   0
               51    04 AC    D0  005B          MOVL    4(AP),R1
      00000920 8F    04 A1    D1  005F          CMPL    4(R1),#SS$_UNWIND
                     04   12  0067          BNEQ    1$
               50    01   D0  0069          MOVL    #SS$_NORMAL,R0
                          04  006C          RET
               50    08 AC    D0  006D  1$:     MOVL    8(AP),R0
            0C A0    04 A1    D0  0071          MOVL    4(R1),12(R0)
                     7E   7C  0076          CLRQ    -(SP)
       00000000'GF   02   FB  0078          CALLS   #2,G^SYS$UNWIND
                          04  007F          RET
                              0080  MATCH:
          57   E338 CD    9E  0080          MOVAB   OPD_IMAGE(FP), R7       ; Address of operand address block
                     59   D4  0085          CLRL    R9                      ; No registers are used yet
                              0087  ;+
                              0087  ; Scan the operand list, getting the addresses of all operands
                              0087  ;-
                              0087  SCAN:
                     52   D6  0087          INCL    R2                      ; Get next operand type byte
                     62   95  0089          TSTB    (R2)                    ; No more operands to test?
                     07   13  008B          BEQL    ALLOPDS                 ; Yes, we have all the operands
                     18   10  008D          BSBB    NEXT_OPERAND            ; Look at next operand
               87    54   D0  008F          MOVL    R4, (R7)+               ; Save address of operand
                     F3   11  0092          BRB     SCAN
                              0094
                              0094  ALLOPDS:                                ; All operand addresses are available
                   00F5   30  0094          BSBW    TRY_TO_FIX              ; Try to fix the error
                  B9 50   E9  0097          BLBC    R0, RESIGNAL            ; If we can't, resignal the error
  FB78 DD   08000000 8F   CA  009A          BICL2   #PSL$M_FPD, @PSL_OFF+ADR_IMAGE(FP)      ; Clear FPD bit
               50    01   D0  00A3          MOVL    #SS$_NORMAL, R0         ; Everything is okay
                          04  00A6          RET                             ; return
```

```
                           00A7         .SBTTL NEXT_OPERAND - Get next operand
                           00A7   ;++
                           00A7   ; FUNCTIONAL DESCRIPTION:
                           00A7   ;
                           00A7   ;       Interpret the instruction stream and gets the next operand.
                           00A7   ;
                           00A7   ; CALLING SEQUENCE
                           00A7   ;
                           00A7   ;       JSB     NEXT_OPERAND
                           00A7   ;
                           00A7   ; INPUT PARAMETERS:
                           00A7   ;
                           00A7   ;       R2 = address of operand type table
                           00A7   ;
                           00A7   ; IMPLICIT INPUTS:
                           00A7   ;
                           00A7   ;       REG_IMAGE(FP)           ; The image of the registers including PC
                           00A7   ;       instruction stream
                           00A7   ;
                           00A7   ; OUTPUT PARAMETERS:
                           00A7   ;
                           00A7   ;       R4 = OA1 (operand address of bits 31:0 of operand)
                           00A7   ;       R5 = OA2 (operand address of bits 63:32 of operand) if R1 = 8
                           00A7   ;       R9 = mask of registers used in the operands
                           00A7   ;
                           00A7   ; IMPLICIT OUTPUT:
                           00A7   ;
                           00A7   ;       Saved image of PC is updated as operand stream is interpreted.
                           00A7   ;
                           00A7   ; COMPLETION STATUS
                           00A7   ;
                           00A7   ;       NONE
                           00A7   ;
                           00A7   ; SIDE EFFECTS:
                           00A7   ;
                           00A7   ;       NONE - uses registers R0:R9 - see LIB$FIXUP_DEC for register usage
                           00A7   ;--
                           00A7
                           00A7   NEXT_OPERAND:
                  53   D4  00A7         CLRL    R3                      ; R3 = initial context index register
              50  62   9A  00A9         MOVZBL  (R2), R0                ; Get operand type byte
     51    FF6B CF40   9A  00AC         MOVZBL  W^OP_CONTEXT[R0], R1    ; Get context amount
                           00B2   ;+
                           00B2   ; Loop to get operand specifier - loop back here (once) if operand specifier is inde
                           00B2   ;-
                           00B2
                           00B2   LOOP_OP:
                00BD   30  00B2         BSBW    NEXT_BYTE               ; R0 = next I-stream byte (sign extended)
   56   50   04   00   EF  00B5         EXTZV   #0, #4, R0, R6          ; R6 = register field
   50   50   04   04   EF  00BA         EXTZV   #4, #4, R0, R0          ; R0 = operand specifier 7:4
             50   0C   93  00BF         BITB    #^B1100, R0             ; Do we use the register?
                  33   13  00C2         BEQL    LITERAL                 ; branch if not
        58   01   56   78  00C4         ASHL    R6, #1, R8              ; Mask of register used
 FC AE  01C00000 8F   50   78  00C8         ASHL    R0, #^X01C00000, -4(SP) ; Is a register modified by this?
                  05   18  00D1         BGEQ    1$                      ; branch if not
   58   10   10   58   F0  00D3         INSV    R8, #16, #16, R8        ; Also set the register modified bit
             59   58   C8  00D8   1$:   BISL2   R8, R9                  ; Include into other modified registers
```

```
      0B    04    50    8F  00DB             CASEB   R0, #4, #15-4              ; Dispatch on operand specifier code
                     0028'  00DF     10$:    .WORD   INDEXED-10$                ; 4
                     0032'  00E1             .WORD   REG-10$                    ; 5
                     0044'  00E3             .WORD   REG_DEF-10$                ; 6
                     003F'  00E5             .WORD   AUTO_DECR-10$              ; 7
                     004B'  00E7             .WORD   AUTO_INCR-10$              ; 8
                     0057'  00E9             .WORD   AUTO_INCR_DEF-10$          ; 9
                     0066'  00EB             .WORD   BYTE_DISPL-10$             ; 10
                     006A'  00ED             .WORD   BYTE_DISPL_DEF-10$         ; 11
                     006E'  00EF             .WORD   WORD_DISPL-10$             ; 12
                     0072'  00F1             .WORD   WORD_DISPL_DEF-10$         ; 13
                     0076'  00F3             .WORD   LONG_DISPL-10$             ; 14
                     0080'  00F5             .WORD   LONG_DISPL_DEF-10$         ; 15
                            00F7
                            00F7     LITERAL:                                   ; We may want to reference the literal
                55    7E    DE  00F7             MOVAL   -(SP), R5                  ; Address of high half of operand
           54    04 AE    DE  00FA             MOVAL   4(SP), R4                  ; Address of low half of operand
                7E    64    D0  00FE             MOVL    (R4), -(SP)                ; Push the return address back
64    56    10    50    7A  0101             EMUL    R0, #16, R6, (R4)          ; Store operand
                      05  0106             RSB                                ; All done
                            0107
                            0107     INDEXED:                                   ; Save context index and loop back
53    51    BC AD46    C5  0107             MULL3   REG_IMAGE(FP)[R6],R1,R3 ; R3 = context index
                      51    D4  010D             CLRL    R1                         ; See if already had an index
                      A1    11  010F             BRB     LOOP_OP                    ; Go back and get next specifier
                            0111
      54   FB38 CD46    D0  0111     REG:    MOVL    ADR_IMAGE(FP)[R6], R4      ; R4 = OA1 = adr where Rn  saved in stack
      55   FB3C CD46    D0  0117             MOVL    ADR_IMAGE+4(FP)[R6], R5 ; R5 = OA2 = adr where Rn+1 saved in stack
                      05  011D             RSB
                            011E
                            011E     AUTO_DECR:
      BC AD46    51    C2  011E             SUBL    R1, REG_IMAGE(FP)[R6]      ; decrement Rn by operand size
                            0123     REG_DEF:
      54    BC AD46    D0  0123             MOVL    REG_IMAGE(FP)[R6], R4      ; R4 = OA = contents of Rn
                      40    11  0128             BRB     SET_OA2                    ; set OA2, check op and RSB
                            012A
                            012A     AUTO_INCR:
      54    BC AD46    D0  012A             MOVL    REG_IMAGE(FP)[R6], R4      ; R4 = OA = contents of Rn
      BC AD46    51    C0  012F             ADDL    R1, REG_IMAGE(FP)[R6]      ; increment Rn by operand size
                      34    11  0134             BRB     SET_OA2                    ; set OA2, check op and RSB
                            0136
                            0136     AUTO_INCR_DEF:
      54    BC AD46    D0  0136             MOVL    REG_IMAGE(FP)[R6], R4      ; R4 = contents of Rn
                54    64    D0  013B             MOVL    (R4), R4                   ; R4 = OA
      BC AD46    04    C0  013E             ADDL    #4, REG_IMAGE(FP)[R6]      ; increment Rn by 4 (size of address)
                      25    11  0143             BRB     SET_OA2                    ; set OA2, check op, and RSB
                            0145
                            0145     BYTE_DISPL:
                      2B    10  0145             BSBB    NEXT_BYTE                  ; R0 = next I-stream byte
                      0E    11  0147             BRB     DISPL                      ; add to PC
                            0149
                            0149     BYTE_DISPL_DEF:
                      27    10  0149             BSBB    NEXT_BYTE                  ; R0 = next I-stream byte
                      14    11  014B             BRB     DISPL_DEF                  ; add to PC and defer
                            014D
                            014D     WORD_DISPL:
                      2B    10  014D             BSBB    NEXT_WORD                  ; R0 = next I-stream word
                     06  11 014F          BRB     DISPL                       ; add to PC
                            0151
                            0151  WORD_DISPL_DEF:
                     27  10 0151          BSBB    NEXT_WORD                   ; R0 = next I-stream word
                     0C  11 0153          BRB     DISPL_DEF                   ; add to PC and defer
                            0155
                            0155  LONG_DISPL:
                     2C  10 0155          BSBB    NEXT_LONG                   ; R0 = next I-stream longword
      54    BC AD46  50  C1 0157  DISPL:  ADDL3   R0, REG_IMAGE(FP)[R6], R4         ; R4 = OA = (Rn) + displacement
                     0B  11 015D          BRB     SET_OA2                     ; set OA2, check OP, and RSB
                            015F
                            015F  LONG_DISPL_DEF:
                     22  10 015F          BSBB    NEXT_LONG                   ; R0 = Next I-stream longword
                            0161  DISPL_DEF:                                  ; here for displacement deferred
      54    BC AD46  50  C1 0161          ADDL3   R0, REG_IMAGE(FP)[R6], R4         ; R4 = (Rn) + displacement
                54   64  D0 0167          MOVL    (R4), R4                    ; R4 = OA = (OA) (do defer)
                            016A
                            016A  ;+
                            016A  ; add context index or 0
                            016A  ; Set OA2 (operand address 2) from OA+4 since
                            016A  ; operand is in memory not a register and therefore is contiguous
                            016A  ;-
                            016A
                            016A  SET_OA2:
                54   53  C0 016A          ADDL    R3, R4                      ; R4 = OA + context index or 0
          55    04   54  C1 016D          ADDL3   R4, #4, R5                  ; R5 = OA2 = OA + 4
                         05 0171          RSB
```

```
                             0172   448 ;+
                             0172   449 ; routines to get next byte, word, or long from I-stream and sign extend
                             0172   450 ;-
                             0172   451
                             0172   452 NEXT_BYTE:
      50    F8 BD     98     0172   453         CVTBL   @IMAGE_PC(FP), R0       ; R0 = next byte
            F8 AD     D6     0176   454         INCL    IMAGE_PC(FP)            ; update PC
                      05     0179   455         RSB                             ; return
                             017A   456
                             017A   457 NEXT_WORD:
      50    F8 BD     32     017A   458         CVTWL   @IMAGE_PC(FP), R0       ; R0 = next word
      F8 AD    02     C0     017E   459         ADDL    #2, IMAGE_PC(FP)        ; update PC
                      05     0182   460         RSB                             ; return
                             0183   461
                             0183   462 NEXT_LONG:
      50    F8 BD     D0     0183   463         MOVL    @IMAGE_PC(FP), R0       ; R0 = next longword
      F8 AD    04     C0     0187   464         ADDL    #4, IMAGE_PC(FP)        ; update PC
                      05     018B   465         RSB                             ; return
```

```
                                    018C          .SBTTL TRY_TO_FIX - Try to fix the operands of the instruction
                                    018C     ;++
                                    018C     ; FUNCTIONAL DESCRIPTION:
                                    018C     ;
                                    018C     ;       Try to fix the operands of the instruction.
                                    018C     ;
                                    018C     ; CALLING SEQUENCE
                                    018C     ;
                                    018C     ;       JSB     TRY_TO_FIX
                                    018C     ;
                                    018C     ; INPUT PARAMETERS:
                                    018C     ;
                                    018C     ;       R2 = address in operand type table
                                    018C     ;       R9 = mask of registers used
                                    018C     ;
                                    018C     ; IMPLICIT INPUTS:
                                    018C     ;
                                    018C     ;       REG_IMAGE(FP)           ; The image of the registers including PC
                                    018C     ;       instruction stream
                                    018C     ;
                                    018C     ; OUTPUT PARAMETERS:
                                    018C     ;
                                    018C     ;       R0 = 1 if successful, 0 otherwise
                                    018C     ;       R9 = mask of registers used in the operands
                                    018C     ;
                                    018C     ; IMPLICIT OUTPUT:
                                    018C     ;
                                    018C     ;       NONE
                                    018C     ;
                                    018C     ; COMPLETION STATUS
                                    018C     ;
                                    018C     ;       NONE
                                    018C     ;
                                    018C     ; SIDE EFFECTS:
                                    018C     ;
                                    018C     ;       NONE
                                    018C     ;--
                                    018C
                                    018C     TRY_TO_FIX:
                                    018C     ;+
                                    018C     ; Find which registers are clobbered by the instruction
                                    018C     ;-
      50   FE70 CF     9E    018C             MOVAB   SING_TAB, R0            ; Base address of R2
      52     52   50   C3    0191             SUBL3   R0, R2, R2              ; R2 less SING_TAB
   51   52   FD 8F     78    0195             ASHL    #-3, R2, R1             ; divided by 8
    52   FE61 CF41     7E    019A             MOVAQ   SING_TAB[R1], R2        ; Restore pointer to opcode value
    59   FE6F CF41     D3    01A0             BITL    REGS_TAB[R1], R9        ; Did we use any clobbered registers?
                  54   12    01A6             BNEQ    100$                    ; Yes, we can't find the source
                             01A8     ;+
                             01A8     ; Try to find the invalid byte.
                             01A8     ;-
      54   E338 CD     7D    01A8             MOVQ    OPD_IMAGE(FP), R4       ; Get the source
             54   64   D0    01AD             MOVL    (R4), R4
             56   55   D0    01B0             MOVL    R5, R6                  ; Grab original source address
             09   62   91    01B3             CMPB    (R2), #^X09             ; Was the instruction CVTSP?
                  02   12    01B6             BNEQ    40$                     ; No, don't check the sign
                  54   D6    01B8             INCL    R4
```

```
            65   54    00    0D  01BA    524 40$:       PROBEW  #0, R4, (R5)              ; Check if source is writable
                       1F    12  01BE    525            BNEQ    43$                       ; Branch if writable
                                 01C0    526 ;+
                                 01C0    527 ; Copy the source string onto the stack
                                 01C0    528 ;-
                 50    8E    D0  01C0    529            MOVL    (SP)+, R0                 ; Save return address
                 5E    54    C2  01C3    530            SUBL2   R4, SP                    ; Make stack space for the string
                 51    5E    D0  01C6    531            MOVL    SP, R1                    ; New source string address
                       50    DD  01C9    532            PUSHL   R0                        ; Push return address
                 50    54    D0  01CB    533            MOVL    R4, R0                    ; Copy byte-length to a temporary
         E33C CD       51    D0  01CE    534            MOVL    R1, 4+OPD_IMAGE(FP)       ; Address of new source
                       03    11  01D3    535            BRB     42$                       ; Jump to end of loop
                 81    85    90  01D5    536 41$:       MOVB    (R5)+, (R1)+              ; Move a byte
                    FA 50    F4  01D8    537 42$:       SOBGEQ  R0, 41$                   ; More bytes to move?
            55   51    54    C3  01DB    538            SUBL3   R4, R1, R5                ; Get new source address
                       54    D7  01DF    539 43$:       DECL    R4
                 09    62    91  01E1    540            CMPB    (R2), #^X09               ; Was the instruction CVTSP?
                       26    12  01E4    541            BNEQ    60$                       ; No, don't check the sign
                 20    65    91  01E6    542            CMPB    (R5), #^A/ /
                       0D    13  01E9    543            BEQL    30$
                 2D    65    91  01EB    544            CMPB    (R5), #^A/-/
                       08    13  01EE    545            BEQL    30$
                 2B    65    91  01F0    546            CMPB    (R5), #^A/+/
                       03    13  01F3    547            BEQL    30$
                 65    20    90  01F5    548            MOVB    #^A/ /, (R5)              ; Put a space into the sign position
                       55    D6  01F8    549 30$:       INCL    R5                        ; Skip the sign byte
                       10    11  01FA    550            BRB     60$
                       50    D4  01FC    551 100$:      CLRL    R0                        ; Indicate an error
                             05  01FE    552            RSB                               ; And return
            51   85    30    83  01FF    553 50$:       SUBB3   #^A/0/, (R5)+, R1
                 0A    51    91  0203    554            CMPB    R1, #10                   ; Valid byte?
                       04    1F  0206    555            BLSSU   60$                       ; Branch if so
              FF A5    30    90  0208    556            MOVB    #^A/0/, -1(R5)            ; Move a zero to that byte
                    F0 54    F4  020C    557 60$:       SOBGEQ  R4, 50$                   ; Try for more bytes
                                 020F    558 ;+
                                 020F    559 ; See if the instruction now works.
                                 020F    560 ;-
            57   E338 CD     9E  020F    561            MOVAB   OPD_IMAGE(FP), R7
                 09    62    91  0214    562            CMPB    (R2), #^X09               ; Was the instruction CVTSP?
                       3C    13  0217    563            BEQL    70$                       ; Yes, don't check the trailing byte
                                 0219    564 ;+
                                 0219    565 ; Check the overpunch character.  If it's not valid, change it.
                                 0219    566 ;-
                 51    65    9A  0219    567            MOVZBL  (R5), R1                          ; The overpunch byte
            52   E340 CD     D0  021C    568            MOVL    8+OPD_IMAGE(FP), R2               ; The translation table
               51    6241    90  0221    569            MOVB    (R2)[R1], R1                      ; The translated byte
              A0 8F    51    91  0225    570            CMPB    R1, #^XA0                         ; Is the digit valid?
                       07    1E  0229    571            BGEQU   61$                               ; Branch if not valid
        0A  51    04   00    ED  022B    572            CMPZV   #0, #4, R1, #^X0A                 ; Is the sign valid?
                       1B    1E  0230    573            BGEQU   64$                               ; Branch if okay
                       51    D4  0232    574 61$:       CLRL    R1                                ; Look for a good character
                       04    11  0234    575            BRB     63$                               ; Jump into the loop
                       51    96  0236    576 62$:       INCB    R1                                ; Less characters to try
                       C2    1F  0238    577            BCS     100$                              ; We couldn't find a good byte
                 50    82    90  023A    578 63$:       MOVB    (R2)+, R0                         ; Grab this translated byte
              A0 8F    50    91  023D    579            CMPB    R0, #^XA0                         ; Is the digit valid?
                       F3    1E  0241    580            BGEQU   62$                               ; Branch if not valid
```

```
      0A   50   04   00   ED  0243           CMPZV   #0, #4, R0, #^X0A                   ; Is the sign valid?
                     EC   1F  0248           BLSSU   62$                                 ; Branch if valid
                65   51   90  024A           MOVB    R1, (R5)                            ; Store the valid byte
 97   97   97   97   97   26  024D  64$:     CVTTP   @(R7)+, @(R7)+, @(R7)+, @(R7)+, @(R7)+
                     07   11  0253           BRB     110$
      97   97   97   97   09  0255  70$:     CVTSP   @(R7)+, @(R7)+, @(R7)+, @(R7)+
                     00   11  025A           BRB     110$
                              025C      ;+
                              025C      ; We really shouldn't be using the state of the output registers to zero
                              025C      ; the destination, but we do so anyway.
                              025C      ;-
                              025C      ;        SUBL3   R2_OFF+REG_IMAGE(FP), #1, R2        ; 2 + nibbles
                              025C      ;        ASHL    #-1, R2, R1                         ; bytes
                              025C      ;        SUBL3   R1, R3_OFF+REG_IMAGE(FP), R3        ; dst + bytes - bytes
                              025C      ;        SUBL2   #2, R2                              ; nibbles
                              025C      ;        MOVB    #^X0C, -(SP)
                              025C      ;        ASHP    #0, #0, (SP)+, #0, R2, (R3)         ; Clear destination
                     50   DC  025C  110$:    MOVPSL  R0
 FB78 DD   04   00   50   F0  025E           INSV    R0, #0, #4, @PSL_OFF+ADR_IMAGE(FP)         ; Store NZVC bits
               FB38  DD   D4  0265           CLRL    @R0_OFF+ADR_IMAGE(FP)               ; R0 = 0
           FB3C DD   56   D0  0269           MOVL    R6, @R1_OFF+ADR_IMAGE(FP)           ; R1 = address of source
               FB40  DD   D4  026E           CLRL    @R2_OFF+ADR_IMAGE(FP)               ; R2 = 0
           FB44 DD   53   D0  0272           MOVL    R3, @R3_OFF+ADR_IMAGE(FP)           ; R3 = address of destination
       FB74 DD   F8  AD   D0  0277           MOVL    IMAGE_PC(FP), @PC_OFF+ADR_IMAGE(FP)
                50   01   D0  027D           MOVL    #1, R0                              ; Indicate success
                          05  0280           RSB
```

```
0281          .SBTTL GET_REGS Get contents and addresses of all save registers in stack
0281    ;++
0281    ; FUNCTIONAL DESCRIPTION:
0281    ;
0281    ;       GET_REGS scans the stack and finds all registers saved
0281    ;       in call frames back to the signal facility. Thus it
0281    ;       makes an image of the registers at the time of the
0281    ;       exception or CALL LIB$SIGNAL/STOP. Because a double
0281    ;       operand may be saved in two different places, an image
0281    ;       array of addresses where the registers are saved is also created.
0281    ;       Note: GET_REGS assumes:
0281    ;       caller has saved R2:R11 in frame using its entry mask so all registers
0281    ;       are in memory somewhere. Stack scan is defensive against bad stacks.
0281    ;       Note:
0281    ;       To reconstruct contents of SP at time of exception or call LIB$SIGNAL,
0281    ;       Use the fact that the signal args list is pushed on stack first.
0281    ;       That is SP is = adr of last signal arg/ +4.
0281    ;       Also depends on saved PC being SYS$CALL_HANDL+4.
0281    ;
0281    ; CALLING SEQUENCE:
0281    ;
0281    ;       JSB     GET_REGS
0281    ;
0281    ; INPUT PARAMETERS:
0281    ;
0281    ;       NONE
0281    ;
0281    ; IMPLICIT INPUTS:
0281    ;
0281    ;       CHF$L_SIGARGLST.(AP)            ; Adr. of array of signal args
0281    ;       CHF$L_MCHARGLST.(AP)            ; Adr. of array of mechanism args
0281    ;
0281    ; OUTPUT PARAMETERS:
0281    ;
0281    ;       NONE
0281    ;
0281    ; IMPLICIT OUTPUTS:
0281    ;
0281    ;       REG_IMAGE(FP)                   ; set reg image array R0:PC/PSL
0281    ;       ADR_IMAGE(FP)                   ; Set adr where reg saved R0:PC/PSL
0281    ;                                       ; except adr. where SP SAVED = 0, since not
0281    ;
0281    ; COMPLETION CODES:
0281    ;
0281    ;       NONE JSB
0281    ;
0281    ; SIDE EFFECTS:
0281    ;
0281    ;       If error, RET with error code
0281    ;--
0281
0281    ;+
0281    ; Registers used:
0281    ;
0281    ;       R0 = scratch
0281    ;       R1 = pointer to register image array (REG_IMAGE)
0281    ;       R2 = stack frame pointer
```

```
                               0281   ;       R3 = Adr. of register save area in frame
                               0281   ;       R4 = Loop count
                               0281   ;       R5 = pointer to address image array (ADR_IMAGE)
                               0281   ;       R6 = register save mask
                               0281   ;-
                               0281
                               0281   GET_REGS:                                ; get register image
                               0281
                               0281   ;+
                               0281   ; Setup loop to scan back through stack
                               0281   ;-
                               0281
        51   BC AD    DE       0281           MOVAL   REG_IMAGE(FP), R1        ; R1 = Adr. reg image vector
           52   5D    D0       0285           MOVL    FP, R2                   ; R2 = Adr. of current frame
                               0288                                            ; where all callers register saved
        54   01   10    78     0288           ASHL    #16, #1, R4              ; R4 = max loop count = 65K
        55   FB38 CD    DE     028C           MOVAL   ADR_IMAGE(FP), R5        ; R5 = adr. of array of address where
                               0291                                            ; registers are saved.
                               0291   ;+
                               0291   ; Loop to scan call stack back to signal exception
                               0291   ;-
                               0291
        53   14   52    C1     0291   LOOP:   ADDL3   R2, #SF$L_SAVE_REGS, -   ; stack frame adr + offset to first reg save
                               0295                   R3                       ; R3 = adr. of first saved reg.
                50    D4       0295           CLRL    R0                       ; R0 = first possible register # saved
 56   06 A2   0C   00    EF    0297           EXTZV   #SF$V_SAVE_MASK, -       ; position of save mask
                               029D                   #SF$S_SAVE_MASK, -       ; size of save mask
                               029D                   SF$W_SAVE_MASK(R2), R6   ; R6 = register save mask
                               029D
                               029D   ;+
                               029D   ; loop to copy saved registers R0:R11 from one call stack frame
                               029D   ; to register image array also set address of register image array.
                               029D   ;-
                               029D
        56   0C   50    EA     029D   LOOP1:  FFS     R0, #12, -               ; find next register in saved bit mask
                50             02A1                   R6, R0                   ; R0 = register number of next saved reg.
                               02A2
             12   13           02A2           BEQL    10$                      ; branch if finished 12-bit reg mask
        63   04   00    0D     02A4           PROBEW  #0, #4, (R3)             ; check if stack still writeable
             24   13           02A8           BEQL    BAD_STACK1               ; branch if stack bad
          6540   53    D0      02AA           MOVL    R3, (R5)[R0]             ; store address of where Rn saved
          6140   83    D0      02AE           MOVL    (R3)+, (R1)[R0]          ; copy saved Rn to image + Rn
         E7 56   50    E4      02B2           BBSC    R0, R6, LOOP1            ; clear bit n for Rn, get next bit
                               02B6
                               02B6   ;+
                               02B6   ; check if frame just saved is that of call to handler from signal or exception
                               02B6   ;-
                               02B6
 00000004'8F   10 A2    D1     02B6   10$:    CMPL    SF$L_SAVE_PC(R2), -      ; saved PC the one from call to handler?
                               02BE                   #SYS$CALL_HANDL+4        ; absolute system vector adr
                16    13       02BE           BEQL    END_SCAN                 ; branch if yes
                               02C0
                               02C0   ;+
                               02C0   ; step (cautiously) to previous frame
                               02C0   ;-
                               02C0
             14   00    0D     02C0           PROBEW  #0, #SF$L_SAVE_REGS,-    ; check if fixed part of previous frame ok
```

```
               0C B2           02C3                         @SF$L_SAVE_FP(R2)           ;
                  07      13   02C5                 BEQL    BAD_STACK1                  ; branch if frame not writeable
        52     0C A2      D0   02C7                 MOVL    SF$L_SAVE_FP(R2), R2        ; R2 = adr. of previous frame
               C3 54      F5   02CB                 SOBGTR  R4, LOOP                    ; go back if haven't scanned too many frames
                               02CE
                               02CE         ;+
                               02CE         ; here if bad stack - return LIB$_BADSTA to caller of LIB$FLT_FIXUP
                               02CE         ;-
                               02CE
                               02CE         BAD_STACK1:
   50    00000000'8F      D0   02CE                 MOVL    #LIB$_BADSTA, R0            ; R0 = BAD STACK completion code
                          04   02D5                 RET                                 ; return to caller of LIB$FIXUP_DEC
                               02D6                                                     ; not JSB caller of GET_REGS
                               02D6
                               02D6         ;+
                               02D6         ; Here when scanned all frames back to call to handler
                               02D6         ; Copy R0:R1 from mechanism vector. Set AP,FP,SP,PC,PSL
                               02D6         ; Also set address where each of these registers is saved
                               02D6         ;-
                               02D6
                               02D6         END_SCAN:
        50     08 AC      D0   02D6                 MOVL    CHF$L_MCHARGLST(AP), R0     ; R0 = adr. of signal mechanism arglist
        65     0C A0      DE   02DA                 MOVAL   CHF$L_MCH_SAVR0(R0), -      ; adr. where R0 saved
                               02DE                         R0_OFF(R5)                  ; to vector of addresses
     04 A5     10 A0      DE   02DE                 MOVAL   CHF$L_MCH_SAVR1(R0), -      ; adr. where R1 saved
                               02E3                         R1_OFF(R5)                  ; to image address vector
        61     0C A0      7D   02E3                 MOVQ    CHF$L_MCH_SAVR0(R0), -      ; saved R0/R1
                               02E7                         R0_OFF(R1)                  ; to register image vector
           51     30      C0   02E7                 ADDL    #AP_OFF, R1                 ; R1 = adr. in image vector of AP/FP
           55     30      C0   02EA                 ADDL    #AP_OFF, R5                 ; R5 = adr. in image address vector of AP/FP
        85     08 A2      DE   02ED                 MOVAL   SF$L_SAVE_AP(R2), -         ; adr of saved AP
                               02F1                         (R5)+                       ; to image address vector
        85     0C A2      DE   02F1                 MOVAL   SF$L_SAVE_FP(R2), -         ; adr of saved FP
                               02F5                         (R5)+                       ; to image address vector
        81     08 A2      7D   02F5                 MOVQ    SF$L_SAVE_AP(R2), -         ; saved AP/FP
                               02F9                         (R1)+                       ; to image register vector
        50     04 BC      9A   02F9                 MOVZBL  @CHF$L_SIGARGLST(AP), R0          ; R0 = # of signal args
     50     04 BC40       DE   02FD                 MOVAL   @CHF$L_SIGARGLST(AP)[R0], R0      ; R0 = adr of last signal arg
           50     04      C0   0302                 ADDL    #4, R0                      ; R0 = SP at time of exception or call LIB$S
                               0305                                                     ; NOTE: this a spec from LIB$SIGNAL and
                               0305                                                     ; exception processing of operating system!!
                  85      D4   0305                 CLRL    (R5)+                       ; SP not saved anywhere so set IMAGE _ADR TO
           81     50      D0   0307                 MOVL    R0, (R1)+                   ; set image SP
           81     70      7D   030A                 MOVQ    -(R0), (R1)+                ; copy PC/PSL to image (always last
                               030D                                                     ; 2 signal arguments)
           85     80      DE   030D                 MOVAL   (R0)+, (R5)+                ; set adr. where PC saved
           85     80      DE   0310                 MOVAL   (R0)+, (R5)+                ; set adr. where PSL saved
                          05   0313                 RSB                                 ; return (to LIB$FIXUP_DEC)
                               0314
                               0314                 .END                                ; end of LIB$FIXUP_DEC
```

```
ADR_IMAGE                       = FFFFFB38                SF$L_SAVE_FP                  = 0000000C
ALLOPDS                           00000094 R      02      SF$L_SAVE_PC                  = 00000010
AP_OFF                          = 00000030                SF$L_SAVE_REGS                = 00000014
AUTO_DECR                         0000011E R      02      SF$S_SAVE_MASK                = 0000000C
AUTO_INCR                         0000012A R      02      SF$V_SAVE_MASK                = 00000000
AUTO_INCR_DEF                     00000136 R      02      SF$W_SAVE_MASK                = 00000006
BAD_STACKT                        000002CE R      02      SIG_TO_RET                      00000059 R      02
BYTE_DISPL                        00000145 R      02      SING_TAB                        00000000 R      02
BYTE_DISPL_DEF                    00000149 R      02      SP_OFF                        = 00000038
CHF$L_MCHARGLST                 = 00000008                SS$_NORMAL                    = 00000001
CHF$L_MCH_SAVR0                 = 0000000C                SS$_RESIGNAL                  = 00000918
CHF$L_MCH_SAVR1                 = 00000010                SS$_ROPRAND                   = 00000454
CHF$L_SIGARGLST                 = 00000004                SS$_UNWIND                    = 00000920
CHF$L_SIG_NAME                  = 00000004                STACK                         = FFFFE338
DISPL                             00000157 R      02      STS$S_COND_ID                 = 00000019
DISPL_DEF                         00000161 R      02      STS$V_COND_ID                 = 00000003
END_SCAN                          000002D6 R      02      SYS$CALL_HANDL                  ********   X    00
FP_OFF                          = 00000034                SYS$UNWIND                      ********   X    00
GET_REGS                          00000281 R      02      TRY_TO_FIX                      0000018C R      02
IMAGE_PC                        = FFFFFFF8                WORD_DISPL                      0000014D R      02
IMAGE_PSL                       = FFFFFFFC                WORD_DISPL_DEF                  00000151 R      02
INDEXED                           00000107 R      02
LIB$FIXUP_DEC                     00000021 RG     02
LIB$_BADSTA                       ********   X    00
LITERAL                           000000F7 R      02
LONG_DISPL                        00000155 R      02
LONG_DISPL_DEF                    0000015F R      02
LOOP                              00000291 R      02
LOOP1                             0000029D R      02
LOOP_OP                           000000B2 R      02
MATCH                             00000080 R      02
NEXT_BYTE                         00000172 R      02
NEXT_LONG                         00000183 R      02
NEXT_OPERAND                      000000A7 R      02
NEXT_WORD                         0000017A R      02
OPD_IMAGE                       = FFFFE338
OP_A                            = 00000004
OP_CONTEXT                        0000001C R      02
OP_D                            = 00000002
OP_P                            = 00000003
OP_W                            = 00000001
OP_Z                            = 00000000
PC_OFF                          = 0000003C
PSL$M_FPD                       = 08000000
PSL_OFF                         = 00000040
R0_OFF                          = 00000000
R1_OFF                          = 00000004
R2_OFF                          = 00000008
R3_OFF                          = 0000000C
REG                               00000111 R      02
REGS_TAB                          00000014 R      02
REG_DEF                           00000123 R      02
REG_IMAGE                       = FFFFFFBC
RESIGNAL                          00000053 R      02
SCAN                              00000087 R      02
SET_OA2                           0000016A R      02
SF$L_SAVE_AP                    = 00000008
```

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

PSECT name                       Allocation          PSECT No.  Attributes
----------                       ----------          ---------  ----------
.  ABS  .                        00000000  (     0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD  NOWRT NOVEC BYTE
$ABS$                            00000000  (     0.)  01 (  1.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD    WRT NOVEC BYTE
_LIB$CODE                        00000314  (   788.)  02 (  2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC LONG

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

Phase                      Page faults    CPU Time       Elapsed Time
-----                      -----------    --------       ------------
Initialization                    29      00:00:00.04    00:00:01.81
Command processing               106      00:00:00.46    00:00:05.54
Pass 1                           232      00:00:06.35    00:00:20.85
Symbol table sort                  0      00:00:00.88    00:00:02.09
Pass 2                           140      00:00:01.91    00:00:07.48
Symbol table output               10      00:00:00.08    00:00:00.36
Psect synopsis output              2      00:00:00.02    00:00:00.02
Cross-reference output             0      00:00:00.00    00:00:00.00
Assembler run totals             521      00:00:09.75    00:00:38.16

The working set limit was 1200 pages.
37212 bytes (73 pages) of virtual memory were used to buffer the intermediate code.
There were 30 pages of symbol table space allocated to hold 579 non-local and 20 local symbols.
771 source lines were read in Pass 1, producing 14 object records in Pass 2.
13 pages of virtual memory were used to define 12 macros.

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

Macro library name                           Macros defined
------------------                           --------------
_$255$DUA28:[SYSLIB]STARLET.MLB;2                     8

598 GETS were required to define 8 macros.

There were no errors, warnings or information messages.

MACRO/DISABLE=TRACE/LIS=LIS$:LIBFIXUPD/OBJ=OBJ$:LIBFIXUPD MSRC$:LIBFIXUPD/UPDATE=(ENH$:LIBFIXUPD)
```

0363 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

SORCOMMAN
LIS

LIBFIXUPD
LIS

REQSYM
R32

CRETRANS
LIS

SFKEYWRD
LIS

SORCOLLAT
LIS

OPCODES
LIS

SORARCHAI
LIS

SORCOLUTI
LIS