D 8

```
LLL                 IIIIIIIII     BBBBBBBBBBBB     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
LLL                 IIIIIIIII     BBBBBBBBBBBB     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
LLL                 IIIIIIIII     BBBBBBBBBBBB     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
LLL                    III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                    III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                    III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                    III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                    III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                    III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                    III        BBBBBBBBBBBB     RRRRRRRRRRRR           TTT        LLL
LLL                    III        BBBBBBBBBBBB     RRRRRRRRRRRR           TTT        LLL
LLL                    III        BBBBBBBBBBBB     RRRRRRRRRRRR           TTT        LLL
LLL                    III        BBB         BBB  RRR   RRR              TTT        LLL
LLL                    III        BBB         BBB  RRR   RRR              TTT        LLL
LLL                    III        BBB         BBB  RRR   RRR              TTT        LLL
LLL                    III        BBB         BBB  RRR      RRR           TTT        LLL
LLL                    III        BBB         BBB  RRR      RRR           TTT        LLL
LLL                    III        BBB         BBB  RRR      RRR           TTT        LLL
LLLLLLLLLLLLLLL     IIIIIIIII     BBBBBBBBBBBB     RRR         RRR        TTT        LLLLLLLLLLLLLLL
LLLLLLLLLLLLLLL     IIIIIIIII     BBBBBBBBBBBB     RRR         RRR        TTT        LLLLLLLLLLLLLLL
LLLLLLLLLLLLLLL     IIIIIIIII     BBBBBBBBBBBB     RRR         RRR        TTT        LLLLLLLLLLLLLLL
```

```
**FILE**ID**LIBFIXUPF                              D  2

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

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
```

LIB
2-0

```
0000            .TITLE  LIB$FIXUP_FLT - Fixup floating reserved operand
0000            .IDENT  /2-006/         ; File: LIBFIXUPF.MAR Edit: DGP2006
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    ;++
0000    ; FACILITY: General Utility Library
0000    ;
0000    ; ABSTRACT:
0000    ;
0000    ;       LIB$FIXUP_FLT fixes up floating reserved operands when a
0000    ;       reserved operand fault occurs so that the instruction may
0000    ;       may be continued.  It is designed to be a condition handler
0000    ;       or to be called from a condition handler.
0000    ;
0000    ; ENVIRONMENT: Runs at any access mode, AST Reentrant
0000    ;
0000    ; AUTHOR: Thomas N. Hastings, Version 1, CREATION DATE: 06-AUG-1977
0000    ;         Steven B. Lionel,   Version 2, CREATION DATE: 02-JAN-1980
0000    ;
0000    ; MODIFIED BY:
0000    ;
0000    ; 2-001 - Adapted from LIB$FIXUP_FLT version 1-004.  Use table lookup for
0000    ;         opcode and operand selection, remove PROBEs, improve operand
0000    ;         fixup logic.  SBL 2-JAN-1980
0000    ; 2-002 - MOVAB of a constant isn't PIC, use MOVL instead.  SBL 9-Jan-1980
0000    ; 2-003 - Correct the MOVB at NEXT_OPERAND to a MOVZBL so that there isn't
0000    ;         garbage in the upper part of R0.  SBL 12-Jan-1981
0000    ; 2-004 - Use local handler so that we only return SS$_ACCVIO and SS$_ROPRAND
0000    ;         as statuses, all other exceptions (like SS$_OPCDEC) resignal.
0000    ;         SBL 13-Oct-1981
0000    ; 2-005 - Use LIB$GET_OPCODE if BPT seen.  SBL 4-Jan-1982
0000    ; 2-006 - Defend the PROBE instruction by checking for SF$L_SAVE_REGS greater
0000    ;         than 512.  DGP 15-Mar-1982
```

```
                                 0000    58 ;--
```

```
                   0000          .SBTTL  DECLARATIONS
                   0000  ;
                   0000  ; LIBRARY MACRO CALLS:
                   0000  ;
                   0000          $SFDEF                  ; Stack frame symbols
                   0000          $CHFDEF                 ; Condition handling facility symbols
                   0000          $STSDEF                 ; Status value symbols
                   0000          $SSDEF                  ; System status values
                   0000  ;
                   0000  ; EXTERNAL DECLARATIONS:
                   0000  ;
                   0000          .DSABL  GBL             ; Force all external symbols to be declared
                   0000          .EXTRN  LIB$_BADSTA     ; Bad stack frame
                   0000          .EXTRN  LIB$SIG_TO_RET  ; Convert signals to return status
                   0000          .EXTRN  LIB$GET_OPCODE  ; Get debugger-modified opcode
                   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
         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
         00000044  0000          K_SAV_IMAGE_SIZ = 4*17          ; size of each image vector
         FFFFFFBC  0000          REG_IMAGE = -K_SAV_IMAGE_SIZ    ; FP offset for image vector of registers
         FFFFFF78  0000          ADR_IMAGE = -<K_SAV_IMAGE_SIZ>*2 ; FP offset for image vector of addresses
                   0000                                          ; where registers have been saved in stack
         FFFFFFFC  0000          IMAGE_PSL = -4                  ; FP offset of PSL image
         FFFFFFF8  0000          IMAGE_PC = -8                   ; FP offset of PC image
                   0000
                   0000
                   0000  ; Define first byte of two byte opcodes for G and H.  G instructions
                   0000  ; are ffFD where ff is a F floating opcode and H instructions are
                   0000  ; ddFD where dd is a D floating opcode.  For example, POLYG is 55FD
                   0000  ; and POLYH is 75FD.
                   0000
         000000FD  0000          G_H     = ^XFD                  ; first G and H opcode byte
                   0000
         00000003  0000          BPT     = ^X03                  ; Opcode for BPT instruction
                   0000
                   0000  ; Define field in floating data types to test for reserved operand.
                   0000
         00000009  0000          S_FMZERO = 9                    ; size for F
         00000007  0000          V_FMZERO = 7                    ; position for F
         0000000C  0000          S_GMZERO = 12                   ; size for G
         00000004  0000          V_GMZERO = 4                    ; position  for G
         00000010  0000          S_HMZERO = 16                   ; size for H
         00000000  0000          V_HMZERO = 0                    ; position for H
                   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_B    = 1                             ; Byte
            00000002 0000               OP_W    = 2                             ; Word
            00000003 0000               OP_F    = 3                             ; F_floating
            00000004 0000               OP_D    = 4                             ; D_floating
            00000005 0000               OP_G    = 5                             ; G_floating
            00000006 0000               OP_H    = 6                             ; H_floating
                     0000   ;
                     0000   ; OWN STORAGE:
                     0000   ;
                 00000000               .PSECT _LIB$CODE PIC, USR, CON, REL, LCL, SHR, -
                     0000                               EXE, RD, NOWRT, LONG
                     0000   ;
                     0000
                     0000   ; Tables of opcodes and operand types.  The first byte in each entry
                     0000   ; is the opcode (or second byte of a 2-byte opcode).  The remaining
                     0000   ; bytes (up to 3) are OP_x codes defined above that specify what datatype
                     0000   ; each operand is for that instruction.  If an operand type is 0, then
                     0000   ; no more operands are processed for that instruction.
                     0000   ;
                     0000   ; These tables are binary searched so the opcodes must be in ascending
                     0000   ; order and the entry addresses must be longword aligned.  This latter
                     0000   ; requirement is met by having these tables be first in this module.
                     0000
                     0000   ; Table for single byte opcodes
                     0000  SING_TAB:
         00 03 03 40 0000               .BYTE   ^X40, OP_F, OP_F, 0     ; ADDF2
         00 03 03 41 0004               .BYTE   ^X41, OP_F, OP_F, 0     ; ADDF3
         00 03 03 42 0008               .BYTE   ^X42, OP_F, OP_F, 0     ; SUBF2
         00 03 03 43 000C               .BYTE   ^X43, OP_F, OP_F, 0     ; SUBF3
         00 03 03 44 0010               .BYTE   ^X44, OP_F, OP_F, 0     ; MULF2
         00 03 03 45 0014               .BYTE   ^X45, OP_F, OP_F, 0     ; MULF3
         00 03 03 46 0018               .BYTE   ^X46, OP_F, OP_F, 0     ; DIVF2
         00 03 03 47 001C               .BYTE   ^X47, OP_F, OP_F, 0     ; DIVF3
         00 00 03 48 0020               .BYTE   ^X48, OP_F, 0   , 0     ; CVTFB
         00 00 03 49 0024               .BYTE   ^X49, OP_F, 0   , 0     ; CVTFW
         00 00 03 4A 0028               .BYTE   ^X4A, OP_F, 0   , 0     ; CVTFL
         00 00 03 4B 002C               .BYTE   ^X4B, OP_F, 0   , 0     ; CVTRFL
         03 03 03 4F 0030               .BYTE   ^X4F, OP_F, OP_F, OP_F  ; ACBF
         00 00 03 50 0034               .BYTE   ^X50, OP_F, 0   , 0     ; MOVF
         00 03 03 51 0038               .BYTE   ^X51, OP_F, OP_F, 0     ; CMPF
         00 00 03 52 003C               .BYTE   ^X52, OP_F, 0   , 0     ; MNEGF
         00 00 03 53 0040               .BYTE   ^X53, OP_F, 0   , 0     ; TSTF
         03 01 03 54 0044               .BYTE   ^X54, OP_F, OP_B, OP_F  ; EMODF
         00 00 03 55 0048               .BYTE   ^X55, OP_F, 0   , 0     ; POLYF
         00 00 03 56 004C               .BYTE   ^X56, OP_F, 0   , 0     ; CVTFD
         00 04 04 60 0050               .BYTE   ^X60, OP_D, OP_D, 0     ; ADDD2
         00 04 04 61 0054               .BYTE   ^X61, OP_D, OP_D, 0     ; ADDD3
         00 04 04 62 0058               .BYTE   ^X62, OP_D, OP_D, 0     ; SUBD2
         00 04 04 63 005C               .BYTE   ^X63, OP_D, OP_D, 0     ; SUBD3
         00 04 04 64 0060               .BYTE   ^X64, OP_D, OP_D, 0     ; MULD2
         00 04 04 65 0064               .BYTE   ^X65, OP_D, OP_D, 0     ; MULD3
         00 04 04 66 0068               .BYTE   ^X66, OP_D, OP_D, 0     ; DIVD2
         00 04 04 67 006C               .BYTE   ^X67, OP_D, OP_D, 0     ; DIVD3
```

```
00 00 04 68  0070              .BYTE   ^X68, OP_D, 0   , 0     ; CVTDB
00 00 04 69  0074              .BYTE   ^X69, OP_D, 0   , 0     ; CVTDW
00 00 04 6A  0078              .BYTE   ^X6A, OP_D, 0   , 0     ; CVTDL
00 00 04 6B  007C              .BYTE   ^X6B, OP_D, 0   , 0     ; CVTRDL
04 04 04 6F  0080              .BYTE   ^X6F, OP_D, OP_D, OP_D  ; ACBD
00 00 04 70  0084              .BYTE   ^X70, OP_D, 0   , 0     ; MOVD
00 04 04 71  0088              .BYTE   ^X71, OP_D, OP_D, 0     ; CMPD
00 00 04 72  008C              .BYTE   ^X72, OP_D, 0   , 0     ; MNEGD
00 00 04 73  0090              .BYTE   ^X73, OP_D, 0   , 0     ; TSTD
04 01 04 74  0094              .BYTE   ^X74, OP_D, OP_B, OP_D  ; EMODD
00 00 04 75  0098              .BYTE   ^X75, OP_D, 0   , 0     ; POLYD
00 00 04 76  009C              .BYTE   ^X76, OP_D, 0   , 0     ; CVTDF
             00A0      SING_END:
             00A0
             00A0      ; Table for 2-byte opcodes.  First byte (^XFD) is omitted.
             00A0
             00A0      DOUB_TAB:
00 00 04 32  00A0              .BYTE   ^X32, OP_D, 0   , 0     ; CVTDH
00 00 05 33  00A4              .BYTE   ^X33, OP_G, 0   , 0     ; CVTGF
00 05 05 40  00A8              .BYTE   ^X40, OP_G, OP_G, 0     ; ADDG2
00 05 05 41  00AC              .BYTE   ^X41, OP_G, OP_G, 0     ; ADDG3
00 05 05 42  00B0              .BYTE   ^X42, OP_G, OP_G, 0     ; SUBG2
00 05 05 43  00B4              .BYTE   ^X43, OP_G, OP_G, 0     ; SUBG3
00 05 05 44  00B8              .BYTE   ^X44, OP_G, OP_G, 0     ; MULG2
00 05 05 45  00BC              .BYTE   ^X45, OP_G, OP_G, 0     ; MULG3
00 05 05 46  00C0              .BYTE   ^X46, OP_G, OP_G, 0     ; DIVG2
00 05 05 47  00C4              .BYTE   ^X47, OP_G, OP_G, 0     ; DIVG3
00 00 05 48  00C8              .BYTE   ^X48, OP_G, 0   , 0     ; CVTGB
00 00 05 49  00CC              .BYTE   ^X49, OP_G, 0   , 0     ; CVTGW
00 00 05 4A  00D0              .BYTE   ^X4A, OP_G, 0   , 0     ; CVTGL
00 00 05 4B  00D4              .BYTE   ^X4B, OP_G, 0   , 0     ; CVTRGL
05 05 05 4F  00D8              .BYTE   ^X4F, OP_G, OP_G, OP_G  ; ACBG
00 00 05 50  00DC              .BYTE   ^X50, OP_G, 0   , 0     ; MOVG
00 05 05 51  00E0              .BYTE   ^X51, OP_G, OP_G, 0     ; CMPG
00 00 05 52  00E4              .BYTE   ^X52, OP_G, 0   , 0     ; MNEGG
00 00 05 53  00E8              .BYTE   ^X53, OP_G, 0   , 0     ; TSTG
05 02 05 54  00EC              .BYTE   ^X54, OP_G, OP_W, OP_G  ; EMODG
00 00 05 55  00F0              .BYTE   ^X55, OP_G, 0   , 0     ; POLYG
00 00 05 56  00F4              .BYTE   ^X56, OP_G, 0   , 0     ; CVTGH
00 06 06 60  00F8              .BYTE   ^X60, OP_H, OP_H, 0     ; ADDH2
00 06 06 61  00FC              .BYTE   ^X61, OP_H, OP_H, 0     ; ADDH3
00 06 06 62  0100              .BYTE   ^X62, OP_H, OP_H, 0     ; SUBH2
00 06 06 63  0104              .BYTE   ^X63, OP_H, OP_H, 0     ; SUBH3
00 06 06 64  0108              .BYTE   ^X64, OP_H, OP_H, 0     ; MULH2
00 06 06 65  010C              .BYTE   ^X65, OP_H, OP_H, 0     ; MULH3
00 06 06 66  0110              .BYTE   ^X66, OP_H, OP_H, 0     ; DIVH2
00 06 06 67  0114              .BYTE   ^X67, OP_H, OP_H, 0     ; DIVH3
00 00 06 68  0118              .BYTE   ^X68, OP_H, 0   , 0     ; CVTHB
00 00 06 69  011C              .BYTE   ^X69, OP_H, 0   , 0     ; CVTHW
00 00 06 6A  0120              .BYTE   ^X6A, OP_H, 0   , 0     ; CVTHL
00 00 06 6B  0124              .BYTE   ^X6B, OP_H, 0   , 0     ; CVTRHL
06 06 06 6F  0128              .BYTE   ^X6F, OP_H, OP_H, OP_H  ; ACBH
00 00 06 70  012C              .BYTE   ^X70, OP_H, 0   , 0     ; MOVH
00 06 06 71  0130              .BYTE   ^X71, OP_H, OP_H, 0     ; CMPH
00 00 06 72  0134              .BYTE   ^X72, OP_H, 0   , 0     ; MNEGH
00 00 06 73  0138              .BYTE   ^X73, OP_H, 0   , 0     ; TSTH
06 02 06 74  013C              .BYTE   ^X74, OP_H, OP_W, OP_H  ; EMODH
```

```
00 00 06 75  0140   231         .BYTE   ^X75, OP_H, 0   , 0     ; POLYH
00 00 06 76  0144   232         .BYTE   ^X76, OP_H, 0   , 0     ; CVTHG
00 00 03 98  0148   233         .BYTE   ^X98, OP_F, 0   , 0     ; CVTFH
00 00 03 99  014C   234         .BYTE   ^X99, OP_F, 0   , 0     ; CVTFG
00 00 06 F6  0150   235         .BYTE   ^XF6, OP_H, 0   , 0     ; CVTHF
00 00 06 F7  0154   236         .BYTE   ^XF7, OP_H, 0   , 0     ; CVTHD
             0158   237 DOUB_END:
             0158   238
             0158   239 ; Table of operand sizes listed in OP_x code order
             0158   240
             0158   241 OP_SIZES:
          00 0158   242         .BYTE   0                       ; Not used
          01 0159   243         .BYTE   1                       ; Byte
          02 015A   244         .BYTE   2                       ; Word
          04 015B   245         .BYTE   4                       ; F_floating
          08 015C   246         .BYTE   8                       ; D_floating
          08 015D   247         .BYTE   8                       ; G_floating
          10 015E   248         .BYTE   16                      ; H_floating
             015F   249
             015F   250 ;
             015F   251 ;  PSECT DECLARATIONS:
             015F   252 ;
             015F   253         .PSECT _LIB$CODE PIC, USR, CON, REL, LCL, SHR, -
     0000015F       254                         EXE, RD, NOWRT, LONG
             015F   255
          015F          .SBTTL  LIB$FIXUP_FLT - Fixup floating reserved operand
          015F   ;++
          015F   ; FUNCTIONAL DESCRIPTION:
          015F   ;
          015F   ; LIB$FIXUP_FLT finds the reserved operand of any  F,  D,  G  or  H
          015F   ; floating  instruction  (with  exceptions  stated  below)  after a
          015F   ; reserved operand fault has been signaled.   LIB$FIXUP_FLT changes
          015F   ; the  reserved operand from -0.0 to the parameter, new_operand, is
          015F   ; present;  or to +0.0 if new_operand is absent.
          015F   ;
          015F   ;
          015F   ;
          015F   ; Exceptions:
          015F   ;
          015F   ; LIB$FIXUP_FLT can not handle the following cases and will  return
          015F   ; a status of SS$_RESIGNAL if any of them occur.
          015F   ;
          015F   ;     1.  The   currently  active  signaled   condition   is   not
          015F   ;         SS$_ROPRAND.
          015F   ;
          015F   ;     2.  The reserved operand's datatype is not  F,  D,  G  or  H
          015F   ;         floating.
          015F   ;
          015F   ;     3.  The  reserved  operand  is  an  element   in   a   POLYx
          015F   ;         table.
          015F   ;
          015F   ; CALLING SEQUENCE:
          015F   ;
          015F   ;     ret_status.wlc.v = LIB$FIXUP_FLT (sig_args_adr.rl.ra,
          015F   ;                             mch_args_adr.rl.ra [, new_operand.rf.r])
          015F   ;
          015F   ; FORMAL PARAMETERS:
          015F   ;
00000004  015F   sig_args_adr = 4
          015F   ;     Address of signal argument vector.
00000008  015F   mch_args_adr = 8
          015F   ;     Address of mechanism argument vector.
0000000C  015F   new_operand = 12
          015F   ;     Optional.  Address of an F_floating  value  to  replace  the
          015F   ;     reserved operand.
          015F   ;
          015F   ; IMPLICIT INPUTS:
          015F   ;
          015F   ;      The stack frames back to that of the instruction which faulted.
          015F   ;      The instruction which faulted and its operands.
          015F   ;
          015F   ; IMPLICIT OUTPUTS:
          015F   ;
          015F   ;      The reserved floating operand, if found, is replaced by
          015F   ;      "new_operand" or zero.
          015F   ;
          015F   ; COMPLETION STATUS:
          015F   ;
          015F   ; SS$_CONTINUE - continue execution at point of condition
          015F   ;     Routine successfully completed.  The  reserved  operand  was
          015F   ;     found and was fixed up.
          015F   ; SS$_ACCVIO - access violation
```

```
                          015F     ;       An argument to LIB$FIXUP_FLT or an operand of  the  faulting
                          015F     ;       instruction could not be read or written.
                          015F     ; SS$_RESIGNAL - resignal condition to next handler
                          015F     ;       The condition signaled  was not SS$_ROPRAND or the  reserved
                          015F     ;       operand  was not a floating point value or was an element in
                          015F     ;       a POLYx table.
                          015F     ; SS$_ROPRAND - reserved operand fault
                          015F     ;       The optional  argument  new_operand  was  supplied  but  was
                          015F     ;       itself an F_floating reserved operand.
                          015F     ; LIB$_BADSTA - bad stack
                          015F     ;       The stack frame linkage had been corrupted since the time of
                          015F     ;       the reserved operand exception.
                          015F     ;
                          015F     ; Note:  If the status value returned from LIB$FIXUP_FLT is seen by
                          015F     ; the  condition  handling  facility,  (as  would  be  the  case if
                          015F     ; LIB$FIXUP_FLT was the handler), any success value  is  equivalent
                          015F     ; to  SS$_CONTINUE,  which  causes the instruction to be restarted.
                          015F     ; Any failure value is equivalent to SS$_RESIGNAL, which will cause
                          015F     ; the  condition  to  be  resignalled to the next handler.  This is
                          015F     ; because the condition handler (LIB$FIXUP_FLT)  failed  to  handle
                          015F     ; the condition correctly.
                          015F     ;
                          015F     ; SIDE EFFECTS:
                          015F     ;
                          015F     ;       If the reserved operand is fixed up, the instruction which
                          015F     ;       faulted is restarted.
                          015F     ;
                          015F     ;--
                          015F
                          015F     ;+
                          015F     ; Registers used:
                          015F     ;
                          015F     ;       R0 =    scratch
                          015F     ;       R1 =    scratch
                          015F     ;       R2 =    pointer into opcode/operand table
                          015F     ;       R3 =    context index or 0
                          015F     ;       R4 =    OA1 (operand address) of bits 31:0
                          015F     ;       R5 =    OA2 (operand address) of bits 63:32 which may not be
                          015F     ;               OA1+4 since registers not ne:essarily saved contiguously.
                          015F     ;       R6 =    register number of operand specifier
                          015F     ;       R7 =    scratch
                          015F     ;       R8 =    OA3 (operand address) of bits 95:64 (H opcodes only)
                          015F     ;       R9 =    OA4 (operand address) of bits 127:96 (H opcodes only)
                          015F     ;-
                          015F
                    0FFC  015F             .ENTRY LIB$FIXUP_FLT, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
                          0161                                                   ; save all registers so that all will be
                          0161                                                   ; found in stack during back scan.
                          0161                                                   ; disable IV (content index multiply)
                          0161             ASSUME  SF$L_SAVE_REGS LE 512         ; Assembly time check to be sure that the
                          0161                                                   ; PROBEs later will work correctly
          6D   03EE'CF  9E  0161           MOVAB   W^OUR_HANDLER, (FP)           ; Enable condition handler
            50    04 AC  D0  0166          MOVL    SIG_ARGS_ADR(AP), R0          ; R0 = adr. of signal arg list array
0000008A 8F  04 A0  19  03  ED  016A       CMPZV   #STS$V_COND_ID, -             ; position of message identification
                          0174                     #STS$S_COND_ID, -             ; size of id
                          0174                     CHF$L_SIG_NAME(R0), -         ; compare 29-bit VAX-11 signal code
                          0174                     #<SS$_ROPRAND@-STS$V_COND_ID> ; with reserved operand code
```

```
LIB
Sym

ADR
AP
AUT
AUT
AUT
BAD
BPT
BYT
BYT
CHF
CHF
CHF
CHF
CHF
CHF
CHK
DES
DES
DES
DES
DIS
DIS
DOU
DOU
END
EXI
FIX
GET
G_H
IMA
IND
K_S
LIB
LIB
LIB
LIB
LON
LON
LOO
LOO
LOO
MCH
NEW
NEX
NEX
NEX
NEX
NEX
NOT
OP_
OP_
OP_
OP_
OP_
OP_
OP_
OUR
```

```
              03    13  0174                BEQL    2$                              ; It is, continue
            0073    31  0176                BRW     RESIGNAL                        ; Not, so resignal
     5E  FF78 CE    9E  0179      2$:       MOVAB   -<K_SAV_IMAGE_SIZ>*2(SP), SP ; allocate two local vectors:
                        017E                                                    ; REG_IMAGE - image of registers at time of
                        017E                                                    ; ADR_IMAGE - image of address where regs ar
                        017E                                                    ; saved in stack in case they need fixup.
            01DA    30  017E                BSBW    GET_REGS                        ; setup the two image vectors in local stora
                        0181                                                    ; do not return here if error, instead RET w
                        0181                                                    ; error completion status
                        0181
                        0181      ;+
                        0181      ; Get instruction opcode.  Determine if this is an instruction which
                        0181      ; we can handle.  If not, resignal.  If so, load R2 with the address
                        0181      ; of the operand table entry for that opcode.
                        0181      ;-
                        0181
            01BD    30  0181                BSBW    NEXT_BYTE                       ; Get first opcode byte
       FD 8F  50    91  0184                CMPB    R0, #G_H                        ; Is this a G or H instruction?
              27    13  0188                BEQL    10$                             ; Yes
          03  50    91  018A                CMPB    R0, #BPT                        ; Is this a BPT instruction?
              12    12  018D                BNEQ    5$                              ; Skip if not
  7E   F8 AD  01    C3  018F                SUBL3   #1, IMAGE_PC(FP), -(SP)         ; Push PC on stack
 00000000'GF  01    FB  0194                CALLS   #1, G^LIB$GET_OPCODE            ; See what opcode really is.
       FD 8F  50    91  019B                CMPB    R0, #G_H                        ; Is it a G or H instruction?
              10    13  019F                BEQL    10$                             ; Skip if so
              51    D4  01A1      5$:       CLRL    R1                              ; Binary search low limit
  57   000000A0 8F  D0  01A3                MOVL    #<SING_END-SING_TAB>, R7        ; Binary search high limit
     53  FE52 CF    9E  01AA                MOVAB   W^SING_TAB, R3                  ; Table base
              11    11  01AF                BRB     SEARCH
            018D    30  01B1      10$:      BSBW    NEXT_BYTE                       ; Get second opcode byte
              51    D4  01B4                CLRL    R1                              ; Binary search low limit
  57   000000B8 8F  D0  01B6                MOVL    #<DOUB_END-DOUB_TAB>, R7        ; Binary search high limit
     53  FEDF CF    9E  01BD                MOVAB   W^DOUB_TAB, R3                  ; Table base
                        01C2
          57  51    D1  01C2      SEARCH:   CMPL    R1, R7                          ; Not in table?
              25    13  01C5                BEQL    RESIGNAL                        ; Yes, resignal
      52  57  51    C1  01C7                ADDL3   R1, R7, R2                      ; Get middle entry
  52  52   FF 8F    78  01CB                ASHL    #-1, R2, R2
          52  03    CA  01D0                BICL2   #3, R2                          ; Longword offset
      50    6342    91  01D3                CMPB    (R3)[R2], R0                    ; Is this the opcode?
              08    1A  01D7                BGTRU   10$                             ; No, too high
              0B    1F  01D9                BLSSU   20$                             ; No, too low
          52  53    C0  01DB                ADDL2   R3, R2                          ; Compute actual address
            0011    31  01DE                BRW     SCAN                            ; Now scan the operands
          57  52    D0  01E1      10$:      MOVL    R2, R7                          ; New high limit is last try
              DC    11  01E4                BRB     SEARCH                          ; Continue search
      51  52  04    C1  01E6      20$:      ADDL3   #4, R2, R1                      ; New low limit is last try + 4
              D6    11  01EA                BRB     SEARCH                          ; Continue search
                        01EC
                        01EC
                        01EC      ;+
                        01EC      ; If we can't handle this exception, return SS$_RESIGNAL.
                        01EC      ;-
                        01EC
                        01EC      RESIGNAL:
      50   0918 8F  3C  01EC                MOVZWL  #SS$_RESIGNAL, R0               ; R0 = RESIGNAL error completion code
                    04  01F1                RET                                     ; error return
```

```
LIB
Pse

PSE
---
.
$AB
_LI

Pha
---
Ini
Com
Pas
Sym
Pas
Sym
Pse
Cro
Ass

The
376
The
918
12

Mac
---
_$2

561

The

MAC
```

```
                    01F2   428
                    01F2   429 ;+
                    01F2   430 ; Now scan the operand list, looking for a reserved operand.
                    01F2   431 ;-
                    01F2   432
                    01F2   433 SCAN:
          52   D6   01F2   434          INCL    R2                      ; Get next operand type byte
     52   03   93   01F4   435          BITB    #3, R2                  ; If the low two bits are clear,
                    01F7   436                                          ; then we are at the start of the
                    01F7   437                                          ; next table entry, thus we are
                    01F7   438                                          ; done.  Remember that the tables
                    01F7   439                                          ; are longword aligned.
          F3   13   01F7   440          BEQL    RESIGNAL                ; No reserved operand found
          62   95   01F9   441          TSTB    (R2)                    ; No more operands to test?
          EF   13   01FB   442          BEQL    RESIGNAL                ; Yes, no reserved operand found
        0052   30   01FD   443          BSBW    NEXT_OPERAND            ; Look at next operand
       EF 50   E9   0200   444          BLBC    R0, SCAN                ; If reserved operand not found,
                    0203   445                                          ; continue looking
          00   11   0203   446          BRB     FIXUP                   ; Fixup reserved operand
```

**F

```
                      0205  ;+
                      0205  ; Fixup reserved operand
                      0205  ;-
                      0205
                      0205  FIXUP:
            50    D4  0205            CLRL    R0                          ; default fixup value is zero
      03    6C    91  0207            CMPB    (AP), #<NEW_OPERAND/4>      ; is new operand present?
            09    1F  020A            BLSSU   10$                         ; no
   50    0C AC    D0  020C            MOVL    NEW_OPERAND(AP), R0         ; yes, get address
            03    13  0210            BEQL    10$                         ; omitted by reference?
      50    60    D0  0212            MOVL    (R0), R0                    ; get F_floating fixup value
03    03    62    8F  0215  10$:      CASEB   (R2), #OP_F, #<OP_H-OP_F>        ; select on operand type
               0009'  0219  11$:      .WORD   DEST_F-11$                  ; F_floating
               000E'  021B            .WORD   DEST_D-11$                  ; D_floating
               0019'  021D            .WORD   DEST_G-11$                  ; G_floating
               0025'  021F            .WORD   DEST_H-11$                  ; H_floating
                  00  0221            HALT                                ; should never get here
      64    50    50  0222  DEST_F:   MOVF    R0, (R4)                    ; move F_floating
            27    11  0225            BRB     EXIT
      7E    50    56  0227  DEST_D:   CVTFD   R0, -(SP)                   ; convert F to D and store
      64    8E    D0  022A            MOVL    (SP)+, (R4)
      65    8E    D0  022D            MOVL    (SP)+, (R5)
            1C    11  0230            BRB     EXIT
      7E    50  99FD  0232  DEST_G:   CVTFG   R0, -(SP)                   ; convert F to G and store
      64    8E    D0  0236            MOVL    (SP)+, (R4)
      65    8E    D0  0239            MOVL    (SP)+, (R5)
            10    11  023C            BRB     EXIT
      7E    50  98FD  023E  DEST_H:   CVTFH   R0, -(SP)                   ; convert F to H and store
      64    8E    D0  0242            MOVL    (SP)+, (R4)
      65    8E    D0  0245            MOVL    (SP)+, (R5)
      68    8E    D0  0248            MOVL    (SP)+, (R8)
      69    8E    D0  024B            MOVL    (SP)+, (R9)
                      024E
                      024E  EXIT:
      50    01    D0  024E            MOVL    #SS$_CONTINUE, R0           ; success
                  04  0251            RET                                 ; return
```

```
                              0252          .SBTTL NEXT_OPERAND - Get next operand and check for floating opcode
                              0252    ;++
                              0252    ; FUNCTIONAL DESCRIPTION:
                              0252    ;
                              0252    ;         NEXT_OPERAND interprets the instruction stream and
                              0252    ;         gets the next operand. It returns 1 in R0
                              0252    ;         if operand is floating or double reserved operand, else 0.
                              0252    ;
                              0252    ; CALLING SEQUENCE
                              0252    ;
                              0252    ;         JSB     NEXT_OPERAND
                              0252    ;
                              0252    ; INPUT PARAMETERS:
                              0252    ;
                              0252    ;         R2 = address of operand type table
                              0252    ;
                              0252    ; IMPLICIT INPUTS:
                              0252    ;
                              0252    ;         REG_IMAGE(FP)                       ; The image of the registers including PC
                              0252    ;         instruction stream
                              0252    ;
                              0252    ; OUTPUT PARAMETERS:
                              0252    ;
                              0252    ;         R4 = OA1 (operand address of bits 31:0 of operand)
                              0252    ;         R5 = OA2 (operand address of bits 63:32 of operand) if R1 = 8
                              0252    ;         R8 = OA3 (H opcodes only)
                              0252    ;         R9 = OA4 (H opcodes only)
                              0252    ;
                              0252    ; IMPLICIT OUTPUT:
                              0252    ;
                              0252    ;         Saved image of PC is updated as operand specific is interpreted
                              0252    ;
                              0252    ; COMPLETION STATUS
                              0252    ;
                              0252    ;         R0 = 1 if operand is floating or double reserved operand, else 0
                              0252    ;
                              0252    ; SIDE EFFECTS:
                              0252    ;
                              0252    ;         NONE - uses registers R0:R9 - see LIB$FIXUP_FLT for register usage
                              0252    ;--
                              0252
                              0252  NEXT_OPERAND:
                   53    D4   0252          CLRL    R3                          ; R3 = initial context index register
               50  62    9A   0254          MOVZBL  (R2), R0                    ; Get operand type byte
     51    FEFC CF40    9A   0257          MOVZBL  W^OP_SIZES[R0], R1          ; Get operand size
                              025D
                              025D    ;+
                              025D    ; Loop to get operand specifier - loop back here (once) if operand specifier is inde
                              025D    ;-
                              025D
                              025D  LOOP_OP:
                 00E1    30   025D          BSBW    NEXT_BYTE                   ; R0 = next I-stream byte (sign extended)
     56    50   04   00    EF   0260          EXTZV   #0, #4, R0, R6              ; R6 = register field
     50    50   04   04    EF   0265          EXTZV   #4, #4, R0, R0              ; R0 = operand specifier 7:4
           0B   04   50    8F   026A          CASEB   R0, #4, #15-4               ; dispatch on operand specifier code
                              026E                                          ; literal 0-3 falls through
                001B'         026E  10$:    .WORD   INDEXED-10$                 ; 4
```

```
LIB$
1-0
```

```
                          0023'  0270          .WORD   REG-10$                         ; 5
                          003B'  0272          .WORD   REG_DEF-10$                     ; 6
                          0042'  0274          .WORD   AUTO_DECR-10$                   ; 7
                          0049'  0276          .WORD   AUTO_INCR-10$                   ; 8
                          0055'  0278          .WORD   AUTO_INCR_DEF-10$               ; 9
                          0064'  027A          .WORD   BYTE_DISPL-10$                  ; 10
                          0068'  027C          .WORD   BYTE_DISPL_DEF-10$              ; 11
                          006C'  027E          .WORD   WORD_DISPL-10$                  ; 12
                          0070'  0280          .WORD   WORD_DISPL_DEF-10$              ; 13
                          0074'  0282          .WORD   LONG_DISPL-10$                  ; 14
                          007E'  0284          .WORD   LONG_DISPL_DEF-10$              ; 15
                                 0286
                                 0286  ;+
                                 0286  ; Literal - can't be reserved, just return failure
                                 0286  ;-
                                 0286
                   00B5   31     0286          BRW     NOT_RESRV                       ; return - not reserved operand
                                 0289
                                 0289  ;+
                                 0289  ; Indexed - save context index and loop back
                                 0289  ;
                                 0289
                                 0289  INDEXED:
   53    51    BC AD46    C5     0289          MULL3   REG_IMAGE(FP)[R6], R1,- ; R3 = context index
                                 028F          R3
                     CC   11     028F          BRB     LOOP_OP                         ; go back and get next specifier
                                 0291
                                 0291  ;+
                                 0291  ; Register
                                 0291  ;-
                                 0291
       54     FF78 CD46   D0     0291  REG:    MOVL    ADR_IMAGE(FP)[R6], R4     ; R4 = OA1 = adr where Rn  saved in stack
       55     FF7C CD46   D0     0297          MOVL    ADR_IMAGE+4(FP)[R6], R5   ; R5 = OA2 = adr where Rn+1 saved in stack
         58     80 AD46   D0     029D          MOVL    ADR_IMAGE+8(FP)[R6], R8   ; R8 = OA3
         59     84 AD46   D0     02A2          MOVL    ADR_IMAGE+12(FP)[R6], R9  ; R9 = OA4
                     5D   11     02A7          BRB     CHK_OP_RSB                ; check operand for reserved and RSB
                                 02A9
                                 02A9  ;+
                                 02A9  ; Register Deferred
                                 02A9  ;-
                                 02A9
                                 02A9  REG_DEF:
         54     BC AD46   D0     02A9          MOVL    REG_IMAGE(FP)[R6], R4     ; R4 = OA = contents of Rn
                     47   11     02AE          BRB     SET_OA2                   ; set OA2, check op and RSB
                                 02B0
                                 02B0  ;+
                                 02B0  ; Auto Decrement
                                 02B0  ;-
                                 02B0
                                 02B0  AUTO_DECR:
         BC AD46     51   C2     02B0          SUBL    R1, REG_IMAGE(FP)[R6]     ; decrement Rn by operand size
                     F2   11     02B5          BRB     REG_DEF                   ; go do register deferred
                                 02B7
                                 02B7  ;+
                                 02B7  ; Auto Increment
                                 02B7  ;-
                                 02B7
```

```
                         02B7  AUTO_INCR:
     54    BC AD46   D0  02B7          MOVL    REG_IMAGE(FP)[R6], R4    ; R4 = OA = contents of Rn
           BC AD46   51  C0  02BC      ADDL    R1, REG_IMAGE(FP)[R6]    ; increment Rn by operand size
                 34  11  02C1          BRB     SET_OA2                  ; set OA2, check op and RSB
                         02C3
                         02C3  ;+
                         02C3  ; Auto Increment Deferred
                         02C3  ;-
                         02C3
                         02C3  AUTO_INCR_DEF:
     54    BC AD46   D0  02C3          MOVL    REG_IMAGE(FP)[R6], R4    ; R4 = contents of Rn
              54 64  D0  02C8          MOVL    (R4), R4                 ; R4 = OA
      BC AD46    04  C0  02CB          ADDL    #4, REG_IMAGE(FP)[R6]    ; increment Rn by 4 (size of address)
                 25  11  02D0          BRB     SET_OA2                  ; set OA2, check op, and RSB
                         02D2
                         02D2  ;+
                         02D2  ; Byte Displacement
                         02D2  ;-
                         02D2
                         02D2  BYTE_DISPL:
                 6D  10  02D2          BSBB    NEXT_BYTE                ; R0 = next I-stream byte
                 0E  11  02D4          BRB     DISPL                    ; add to PC
                         02D6
                         02D6  ;+
                         02D6  ; Byte Displacement Deferred
                         02D6  ;-
                         02D6
                         02D6  BYTE_DISPL_DEF:
                 69  10  02D6          BSBB    NEXT_BYTE                ; R0 = next I-stream byte
                 14  11  02D8          BRB     DISPL_DEF                ; add to PC and defer
                         02DA
                         02DA  ;+
                         02DA  ; Word Displacement
                         02DA  ;-
                         02DA
                         02DA  WORD_DISPL:
                 6D  10  02DA          BSBB    NEXT_WORD                ; R0 = next I-stream word
                 06  11  02DC          BRB     DISPL                    ; add to PC
                         02DE
                         02DE  ;+
                         02DE  ; Word Displacement Deferred
                         02DE  ;-
                         02DE
                         02DE  WORD_DISPL_DEF:
                 69  10  02DE          BSBB    NEXT_WORD                ; R0 = next I-stream word
                 0C  11  02E0          BRB     DISPL_DEF                ; add to PC and defer
                         02E2
                         02E2  ;+
                         02E2  ; Long displacement
                         02E2  ;-
                         02E2
                         02E2  LONG_DISPL:
                 6E  10  02E2          BSBB    NEXT_LONG                ; R0 = next I-stream longword
   54    BC AD46     50  C1  02E4  DISPL:  ADDL3   R0, REG_IMAGE(FP)[R6], R4        ; R4 = OA = (Rn) + displacement
                 0B  11  02EA          BRB     SET_OA2                  ; set OA2, check OP, and RSB
                         02EC
                         02EC  ;+
                             02EC   ; Long Displacement deferred
                             02EC   ;-
                             02EC
                             02EC   LONG_DISPL_DEF:
                  64    10   02EC           BSBB    NEXT_LONG                    ; R0 = Next I-stream longword
                             02EE   DISPL_DEF:                                   ; here for displacement deferred
    54   BC AD46  50    C1   02EE           ADDL3   R0, REG_IMAGE(FP)[R6], R4          ; R4 = (Rn) + displacement
              54  64    D0   02F4           MOVL    (R4), R4                     ; R4 = OA = (OA) (do defer)
                             02F7
                             02F7   ;+
                             02F7   ; add context index or 0
                             02F7   ; Set OA2 (operand address 2)from OA+4 since
                             02F7   ; operand is in memory not a register and therefore is contiguous
                             02F7   ; Also set OA3 and OA4
                             02F7   ;-
                             02F7
                             02F7   SET_OA2:
          54      53    C0   02F7           ADDL    R3, R4                       ; R4 = OA + context index or 0
    55    04      54    C1   02FA           ADDL3   R4, #4, R5                   ; R5 = OA2 = OA + 4
    58    04      55    C1   02FE           ADDL3   R5, #4, R8                   ; R8 = OA3
    59    04      58    C1   0302           ADDL3   R8, #4, R9                   ; R9 = OA4
                             0306
                             0306   ;+
                             0306   ; check for reserved operand
                             0306   ;-
                             0306
                             0306   CHK_OP_RSB:
          50      01    D0   0306           MOVL    #1, R0                       ; Indicate success, initially
    03    03      62    8F   0309           CASEB   (R2), #OP_F, #<OP_H-OP_F>          ; Case on operand type
                   000A'     030D   1$:     .WORD   10$-1$                       ; F_floating
                   000A'     030F           .WORD   10$-1$                       ; D_floating
                   0017'     0311           .WORD   20$-1$                       ; G_floating
                   0024'     0313           .WORD   30$-1$                       ; H_floating
                  27    11   0315           BRB     NOT_RESRV                    ; wrong datatype
    64    09      07    ED   0317   10$:    CMPZV   #V_FMZERO, #S_FMZERO, -      ; Check F_floating and D_floating
       00000100   8F         031B                   (R4), #^X100
                  1E    13   0320           BEQL    NEXT_OPERANDX                ; Found
                  1A    11   0322           BRB     NOT_RESRV                    ; Not found
    64    0C      04    ED   0324   20$:    CMPZV   #V_GMZERO, #S_GMZERO, -      ; Check G_floating
       00000800   8F         0328                   (R4), #^X800
                  11    13   032D           BEQL    NEXT_OPERANDX                ; Found
                  0D    11   032F           BRB     NOT_RESRV                    ; Not found
    64    10      00    ED   0331   30$:    CMPZV   #V_HMZERO, #S_HMZERO, -      ; Check H_floating
       00008000   8F         0335                   (R4), #^X8000
                  04    13   033A           BEQL    NEXT_OPERANDX                ; Found
                  00    11   033C           BRB     NOT_RESRV                    ; Not found
                             033E   NOT_RESRV:
                  50    D4   033E           CLRL    R0                           ; R0 = failure
                             0340   NEXT_OPERANDX:
                        05   0340           RSB                                  ; return R0 indicating success or failure
                             0341
                             0341
                             0341
                             0341   ;+
                             0341   ; routines to get next byte, word, or long from I-stream and sign extend
                             0341   ;-
                             0341
```

```
                            0341      NEXT_BYTE:
             50   F8 BD  98 0341              CVTBL   @IMAGE_PC(FP), R0         ; R0 = next byte
                  F8 AD  D6 0345              INCL    IMAGE_PC(FP)              ; update PC
                         05 0348              RSB                               ; return
                            0349
                            0349      NEXT_WORD:
             50   F8 BD  32 0349              CVTWL   @IMAGE_PC(FP), R0         ; R0 = next word
             F8 AD   02  C0 034D              ADDL    #2, IMAGE_PC(FP)          ; update PC
                         05 0351              RSB                               ; return
                            0352
                            0352      NEXT_LONG:
             50   F8 BD  D0 0352              MOVL    @IMAGE_PC(FP), R0         ; R0 = next longword
             F8 AD   04  C0 0356              ADDL    #4, IMAGE_PC(FP)          ; update PC
                         05 035A              RSB                               ; return
                            035B
```

```
035B          .SBTTL GET_REGS Get contents and addresses of all save registers in stack
035B  ;++
035B  ; FUNCTIONAL DESCRIPTION:
035B  ;
035B  ;       GET_REGS scans the stack and finds all registers saved
035B  ;       in call frames back to the signal facility. Thus it
035B  ;       makes an image of the registers at the time of the
035B  ;       exception or CALL LIB$SIGNAL/STOP. Because a double
035B  ;       operand may be saved in two different places, an image
035B  ;       array of addresses where the registers are saved is also created.
035B  ;       Note: GET_REGS assumes:
035B  ;       caller has saved R2:R11 in frame using its entry mask so all registers
035B  ;       are in memory somewhere. Stack scan is defensive against bad stacks.
035B  ;       Note: to reconstruct contents of SP at time of exception or call LIB$SIGNAL,
035B  ;       Use of the fact that the signal args list is pushed on stack first is made.
035B  ;       That is SP is = adr of last signal arg/ +4. Also depends on saved PC being
035B  ;       SYS$CALL_HANDL+4.
035B  ;
035B  ; CALLING SEQUENCE:
035B  ;
035B  ;       JSB     GET_REGS
035B  ;
035B  ; INPUT PARAMETERS:
035B  ;
035B  ;       NONE
035B  ;
035B  ; IMPLICIT INPUTS:
035B  ;
035B  ;       SIG_ARGS_ADR.(AP)                       ; Adr. of array of signal args
035B  ;       MCH_ARGS_ADR.(AP)                       ; Adr. of array of mechanism args
035B  ;
035B  ; OUTPUT PARAMETERS:
035B  ;
035B  ;       NONE
035B  ;
035B  ; IMPLICIT OUTPUTS:
035B  ;
035B  ;       REG_IMAGE(FP)                           ; set reg image array R0:PC/PSL
035B  ;       ADR_IMAGE(FP)                           ; Set adr where reg saved R0:PC/PSL
035B  ;                                               ; except adr. where SP SAVED = 0, since not
035B  ;
035B  ; COMPLETION CODES:
035B  ;
035B  ;       NONE JSB
035B  ;
035B  ; SIDE EFFECTS:
035B  ;
035B  ;       If error, RET with error code
035B  ;--
035B
035B  ;+
035B  ; Registers used:
035B  ;
035B  ;       R0 = scratch
035B  ;       R1 = pointer to register image array (REG_IMAGE)
035B  ;       R2 = stack frame pointer
035B  ;       R3 = Adr. of register save area in frame
```

```
                         035B  ;         R4 = Loop count
                         035B  ;         R5 = pointer to address image array (ADR_IMAGE)
                         035B  ;         R6 = register save mask
                         035B  ;-
                         035B
                         035B  GET_REGS:                                   ; get register image
                         035B
                         035B  ;+
                         035B  ; Setup loop to scan back through stack
                         035B  ;-
                         035B
       51    BC AD   DE  035B          MOVAL   REG_IMAGE(FP), R1           ; R1 = Adr. reg image vector
          52    5D   D0  035F          MOVL    FP, R2                      ; R2 = Adr. of current frame
                         0362                                              ; where all callers register saved
      54   01   10   78  0362          ASHL    #16, #1, R4                 ; R4 = max loop count = 65K
      55 FF78 CD     DE  0366          MOVAL   ADR_IMAGE(FP), R5           ; R5 = adr. of array of address where
                         036B                                              ; registers are saved.
                         036B  ;+
                         036B  ; Loop to scan call stack back to signal exception
                         036B  ;-
                         036B
      53   14   52   C1  036B  LOOP:   ADDL3   R2, #SF$L_SAVE_REGS, -  ; stack frame adr + offset to first reg saved
                         036F                  R3                          ; R3 = adr. of first saved reg.
                50   D4  036F          CLRL    R0                          ; R0 = first possible register # saved
56   06 A2   0C   00   EF  0371          EXTZV   #SF$V_SAVE_MASK, -          ; position of save mask
                         0377                  #SF$S_SAVE_MASK, -          ; size of save mask
                         0377                  SF$W_SAVE_MASK(R2), R6    ; R6 = register save mask
                         0377
                         0377  ;+
                         0377  ; loop to copy saved registers R0:R11 from one call stack frame
                         0377  ; to register image array also set address of register image array.
                         0377  ;-
                         0377
      56   0C   50   EA  0377  LOOP1:  FFS     R0, #12, -                  ; find next register in saved bit mask
                50       037B                  R6, R0                      ; R0 = register number of next saved reg.
                         037C
                12   13  037C          BEQL    10$                         ; branch if finished 12-bit reg mask
      63   04   00   0D  037E          PROBEW  #0, #4, (R3)                ; check if stack still writeable
                24   13  0382          BEQL    BAD_STACK1                  ; branch if stack bad
        6540    53   D0  0384          MOVL    R3, (R5)[R0]                ; store address of where Rn saved
        6140    83   D0  0388          MOVL    (R3)+, (R1)[R0]             ; copy saved Rn to image + Rn
       E7 56    50   E4  038C          BBSC    R0, R6, LOOP1               ; clear bit n for Rn, get next bit
                         0390
                         0390  ;+
                         0390  ; check if frame just saved is that of call to handler from signal or exception
                         0390  ;-
                         0390
  00000004'8F   10 A2   D1  0390  10$:    CMPL    SF$L_SAVE_PC(R2), -         ; saved PC the one from call to handler?
                         0398                  #SYS$CALL_HANDL+4           ; absolute system vector adr
                16   13  0398          BEQL    END_SCAN                    ; branch if yes
                         039A
                         039A  ;+
                         039A  ; step (cautiously) to previous frame
                         039A  ;-
                         039A
           14   00   0D  039A          PROBEW  #0, #SF$L_SAVE_REGS,-       ; check if fixed part of previous frame ok
             0C B2       039D                  @SF$L_SAVE_FP(R2)           ;
```

```
                07   13  039F               BEQL    BAD_STACK1               ; branch if frame not writeable
      52     0C A2   D0  03A1               MOVL    SF$L_SAVE_FP(R2), R2     ; R2 = adr. of previous frame
             C3 54   F5  03A5               SOBGTR  R4, LOOP                 ; go back if haven't scanned too many frames
                         03A8
                         03A8   ;+
                         03A8   ; here if bad stack - return LIB$_BADSTA to caller of LIB$FLT_FIXUP
                         03A8   ;-
                         03A8
                         03A8   BAD_STACK1:
   50   00000000'8F  D0  03A8               MOVL    #LIB$_BADSTA, R0         ; R0 = BAD STACK completion code
                     04  03AF               RET                              ; return to caller of LIB$FIXUP_FLT
                         03B0                                                ; not JSB caller of GET_REGS
                         03B0
                         03B0   ;+
                         03B0   ; Here when scanned all frames back to call to handler
                         03B0   ; Copy R0:R1 from mechanism vector. Set AP,FP,SP,PC,PSL
                         03B0   ; Also set address where each of these registers is saved
                         03B0   ;-
                         03B0
                         03B0   END_SCAN:
      50     08 AC   D0  03B0               MOVL    MCH_ARGS_ADR(AP), R0     ; R0 = adr. of signal mechanism arglist
                         03B4
      65     0C A0   DE  03B4               MOVAL   CHF$L_MCH_SAVR0(R0), -   ; adr. where R0 saved
                         03B8                       R0_OFF(R5)               ; to vector of addresses
   04 A5     10 A0   DE  03B8               MOVAL   CHF$L_MCH_SAVR1(R0), -   ; adr. where R1 saved
                         03BD                       R1_OFF(R5)               ; to image address vector
      61     0C A0   7D  03BD               MOVQ    CHF$L_MCH_SAVR0(R0), -   ; saved R0/R1
                         03C1                       R0_OFF(R1)               ; to register image vector
           51    30  C0  03C1               ADDL    #AP_OFF, R1              ; R1 = adr. in image vector of aP/FP
           55    30  C0  03C4               ADDL    #AP_OFF, R5              ; R5 = adr. in image address vector of AP/FP
      85     08 A2   DE  03C7               MOVAL   SF$L_SAVE_AP(R2), -      ; adr of saved AP
                         03CB                       (R5)+                    ; to image address vector
      85     0C A2   DE  03CB               MOVAL   SF$L_SAVE_FP(R2), -      ; adr of saved FP
                         03CF                       (R5)+                    ; to image address vector
      81     08 A2   7D  03CF               MOVQ    SF$L_SAVE_AP(R2), -      ; saved AP/FP
                         03D3                       (R1)+                    ; to image register vector
      50     04 BC   9A  03D3               MOVZBL  @SIG_ARGS_ADR(AP), R0    ; R0 = # of signal args
     50    04 BC40   DE  03D7               MOVAL   @SIG_ARGS_ADR(AP)[R0],-  ;
                         03DC                       R0                       ; R0 = adr of last signal arg
      50     04      C0  03DC               ADDL    #4, R0                   ; R0 = SP at time of exception or call LIB$S
                         03DF                                                ; NOTE: this a spec from LIB$SIGNAL and
                         03DF                                                ; exception processing of operating system!!
                85   D4  03DF               CLRL    (R5)+                    ; SP not saved anywhere so set IMAGE _ADR TO
           81    50  D0  03E1               MOVL    R0, (R1)+                ; set image SP
           81    70  7D  03E4               MOVQ    -(R0), (R1)+             ; copy PC/PSL to image (always last
                         03E7                                                ; 2 signal arguments)
           85    80  DE  03E7               MOVAL   (R0)+, (R5)+             ; set adr. where PC saved
           85    80  DE  03EA               MOVAL   (R0)+, (R5)+             ; set adr. where PSL saved
                     05  03ED               RSB                              ; return (to LIB$FIXUP_FLT)
                         03EE
```

```
                              03EE  ;++
                              03EE  ; OUR_HANDLER - Local condition handler
                              03EE  ;
                              03EE  ;       This condition handler is enabled by LIB$FIXUP_FLT.  If the signal
                              03EE  ;       depth is zero and if the exception is SS$_ACCVIO or SS$_ROPRAND, then
                              03EE  ;       LIB$SIG_TO_RET is called to cause LIB$FIXUP_FLT to return the
                              03EE  ;       exception condition as a status.  All other exceptions are resignalled.
                              03EE  ;--
                              03EE
                              03EE  OUR_HANDLER:
                        0000  03EE          .WORD   ^M<>                        ; Save nothing
           50   0918 8F   3C  03F0          MOVZWL  #SS$_RESIGNAL, R0           ; Resignal if nothing else
             51    08 AC  D0  03F5          MOVL    CHF$L_MCHARGLST(AP), R1     ; Get mechanism args list
                   08 A1  D5  03F9          TSTL    CHF$L_MCH_DEPTH(R1)         ; Is depth zero?
                      1D  12  03FC          BNEQ    90$                         ; If not, resignal
             51    04 AC  D0  03FE          MOVL    CHF$L_SIGARGLST(AP), R1     ; Get signal args list
             51    04 A1  D0  0402          MOVL    CHF$L_SIG_NAME(R1), R1      ; Get signal name
                0C    51  D1  0406          CMPL    R1, #SS$_ACCVIO             ; Access violation?
                09    13      0409          BEQL    10$                         ; If so, call LIB$SIG_TO_RET
   00000454 8F    51  D1  040B          CMPL    R1, #SS$_ROPRAND            ; Reserved operand?
                07    12      0412          BNEQ    90$                         ; If not, resignal
   00000000'GF    6C  FA  0414  10$:    CALLG   (AP), G^LIB$SIG_TO_RET      ; Unwind to LIB$FIXUP_FLT's caller
                      04      041B  90$:    RET                                 ; Return to CHF
                              041C
                              041C          .END                                ; end of LIB$FIXUP_FLT
```

```
LIB$FIXUP_FLT                    - Fixup floating reserved operand        16-SEP-1984 00:09:10  VAX/VMS Macro V04-00          Page  21
Symbol table                                                               6-SEP-1984 11:07:14  [LIBRTL.SRC]LIBFIXUPF.MAR;1         (7)

ADR_IMAGE                       = FFFFFF78                  R0_OFF                          = 00000000
AP_OFF                          = 00000030                  R1_OFF                          = 00000004
AUTO_DECR                         000002B0 R      02        REG                               00000291 R      02
AUTO_INCR                         000002B7 R      02        REG_DEF                           000002A9 R      02
AUTO_INCR_DEF                     000002C3 R      02        REG_IMAGE                       = FFFFFFBC
BAD_STACKT                        000003A8 R      02        RESIGNAL                          000001EC R      02
BPT                             = 00000003                  SCAN                              000001F2 R      02
BYTE_DISPL                        000002D2 R      02        SEARCH                            000001C2 R      02
BYTE_DISPL_DEF                    000002D6 R      02        SET_OA2                           000002F7 R      02
CHF$L_MCHARGLST                 = 00000008                  SF$L_SAVE_AP                    = 00000008
CHF$L_MCH_DEPTH                 = 00000008                  SF$L_SAVE_FP                    = 0000000C
CHF$L_MCH_SAVR0                 = 0000000C                  SF$L_SAVE_PC                    = 00000010
CHF$L_MCH_SAVR1                 = 00000010                  SF$L_SAVE_REGS                  = 00000014
CHF$L_SIGARGLST                 = 00000004                  SF$S_SAVE_MASK                  = 0000000C
CHF$L_SIG_NAME                  = 00000004                  SF$V_SAVE_MASK                  = 00000000
CHK_OP_RSB                        00000306 R      02        SF$W_SAVE_MASK                  = 00000006
DEST_D                            00000227 R      02        SIG_ARGS_ADR                    = 00000004
DEST_F                            00000222 R      02        SING_END                          000000A0 R      02
DEST_G                            00000232 R      02        SING_TAB                          00000000 R      02
DEST_H                            0000023E R      02        SS$_ACCVIO                      = 0000000C
DISPL                             000002E4 R      02        SS$_CONTINUE                    = 00000001
DISPL_DEF                         000002EE R      02        SS$_RESIGNAL                    = 00000918
DOUB_END                          00000158 R      02        SS$_ROPRAND                     = 00000454
DOUB_TAB                          000000A0 R      02        STS$S_COND_ID                   = 00000019
END_SCAN                          000003B0 R      02        STS$V_COND_ID                   = 00000003
EXIT                              0000024E R      02        SYS$CALL_HANDL                    ********   X  00
FIXUP                             00000205 R      02        S_FMZERO                        = 00000009
GET_REGS                          0000035B R      02        S_GMZERO                        = 0000000C
G_H                             = 000000FD                  S_HMZERO                        = 00000010
IMAGE_PC                        = FFFFFFF8                  V_FMZERO                        = 00000007
INDEXED                           00000289 R      02        V_GMZERO                        = 00000004
K_SAV_IMAGE_SIZ                 = 00000044                  V_HMZERO                        = 00000000
LIB$FIXUP_FLT                     0000015F RG     02        WORD_DISPL                        000002DA R      02
LIB$GET_OPCODE                    ********   X    00        WORD_DISPL_DEF                    000002DE R      02
LIB$SIG_TO_RET                    ********   X    00
LIB$_BADSTA                       ********   X    00
LONG_DISPL                        000002E2 R      02
LONG_DISPL_DEF                    000002EC R      02
LOOP                              0000036B R      02
LOOP1                             00000377 R      02
LOOP_OP                           0000025D R      02
MCH_ARGS_ADR                    = 00000008
NEW_OPERAND                     = 0000000C
NEXT_BYTE                         00000341 R      02
NEXT_LONG                         00000352 R      02
NEXT_OPERAND                      00000252 R      02
NEXT_OPERANDX                     00000340 R      02
NEXT_WORD                         00000349 R      02
NOT_RESRV                         0000033E R      02
OP_B                            = 00000001
OP_D                            = 00000004
OP_F                            = 00000003
OP_G                            = 00000005
OP_H                            = 00000006
OP_SIZES                          00000158 R      02
OP_W                            = 00000002
OUR_HANDLER                       000003EE R      02
```

```
                                                              N  3
LIB$FIXUP_FLT                    - Fixup floating reserved operand        16-SEP-1984 00:09:10  VAX/VMS Macro V04-00         Page  22
Psect synopsis                                                             6-SEP-1984 11:07:14  [LIBRTL.SRC]LIBFIXUPF.MAR;1         (7)

                                              +----------------+
                                              ! 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                        0000041C  (  1052.)  02 (  2.)    PIC    USR    CON    REL    LCL   SHR   EXE   RD   NOWRT NOVEC LONG

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

Phase                      Page faults    CPU Time       Elapsed Time
-----                      -----------    --------       ------------
Initialization                      29    00:00:00.03    00:00:01.85
Command processing                 103    00:00:00.31    00:00:02.19
Pass 1                             233    00:00:04.26    00:00:18.30
Symbol table sort                    0    00:00:00.53    00:00:01.85
Pass 2                             169    00:00:01.40    00:00:06.12
Symbol table output                 11    00:00:00.10    00:00:01.04
Psect synopsis output                3    00:00:00.02    00:00:00.02
Cross-reference output               0    00:00:00.00    00:00:00.00
Assembler run totals               550    00:00:06.65    00:00:31.37

The working set limit was 1500 pages.
37666 bytes (74 pages) of virtual memory were used to buffer the intermediate code.
There were 30 pages of symbol table space allocated to hold 559 non-local and 15 local symbols.
918 source lines were read in Pass 1, producing 15 object records in Pass 2.
12 pages of virtual memory were used to define 11 macros.

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

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

561 GETS were required to define 8 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:LIBFIXUPF/OBJ=OBJ$:LIBFIXUPF MSRC$:LIBFIXUPF/UPDATE=(ENH$:LIBFIXUPF)
```

0207 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY