```
EEEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                AAAAAAAAA     TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                AAAAAAAAA     TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                AAAAAAAAA     TTTTTTTTTTTTTTT
EEE               MMMMMMM   MMMMMMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMMMMMM   MMMMMMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMMMMMM   MMMMMMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMM MMM MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMM MMM MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMM MMM MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEEEEEEEEEEE      MMM     MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEEEEEEEEEEE      MMM     MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEEEEEEEEEEE      MMM     MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMM     MMM  UUU         UUU  LLL              AAAAAAAAAAAAAAA         TTT
EEE               MMM     MMM  UUU         UUU  LLL              AAAAAAAAAAAAAAA         TTT
EEE               MMM     MMM  UUU         UUU  LLL              AAAAAAAAAAAAAAA         TTT
EEE               MMM     MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMM     MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEE               MMM     MMM  UUU         UUU  LLL              AAA         AAA         TTT
EEEEEEEEEEEEEEEE  MMM     MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA         TTT
EEEEEEEEEEEEEEEE  MMM     MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA         TTT
EEEEEEEEEEEEEEEE  MMM     MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA         TTT
```

```
FFFFFFFFFF  PPPPPPPP    EEEEEEEEEE  MM      MM  UU      UU  LL            AAAAAA    TTTTTTTTTT  EEEEEEEEEE
FFFFFFFFFF  PPPPPPPP    EEEEEEEEEE  MM      MM  UU      UU  LL            AAAAAA    TTTTTTTTTT  EEEEEEEEEE
FF          PP      PP  EE          MMMM  MMMM  UU      UU  LL          AA      AA      TT      EE
FF          PP      PP  EE          MMMM  MMMM  UU      UU  LL          AA      AA      TT      EE
FF          PP      PP  EE          MM  MM  MM  UU      UU  LL          AA      AA      TT      EE
FF          PP      PP  EE          MM  MM  MM  UU      UU  LL          AA      AA      TT      EE
FFFFFFFF    PPPPPPPP    EEEEEEE     MM      MM  UU      UU  LL          AA      AA      TT      EEEEEEE
FFFFFFFF    PPPPPPPP    EEEEEEE     MM      MM  UU      UU  LL          AA      AA      TT      EEEEEEE
FF          PP          EE          MM      MM  UU      UU  LL          AAAAAAAAAA      TT      EE
FF          PP          EE          MM      MM  UU      UU  LL          AAAAAAAAAA      TT      EE
FF          PP          EE          MM      MM  UU      UU  LL          AA      AA      TT      EE          ....
FF          PP          EE          MM      MM  UU      UU  LL          AA      AA      TT      EE          ....
FF          PP          EEEEEEEEEE  MM      MM  UUUUUUUUUU  LLLLLLLLLL  AA      AA      TT      EEEEEEEEEE  ....
FF          PP          EEEEEEEEEE  MM      MM  UUUUUUUUUU  LLLLLLLLLL  AA      AA      TT      EEEEEEEEEE  ....

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     1          .TITLE  VAX$EMULATE_FP - Emulate floating-point instructions
0000     2          .IDENT  /V04-000/
0000     3
0000     4  ;
0000     5  ;****************************************************************************
0000     6  ;*                                                                          *
0000     7  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000     8  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9  ;*  ALL RIGHTS RESERVED.                                                    *
0000    10  ;*                                                                          *
0000    11  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    12  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    13  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    14  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    15  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    16  ;*  TRANSFERRED.                                                            *
0000    17  ;*                                                                          *
0000    18  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    19  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    20  ;*  CORPORATION.                                                            *
0000    21  ;*                                                                          *
0000    22  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    23  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    24  ;*                                                                          *
0000    25  ;*                                                                          *
0000    26  ;****************************************************************************
0000    27  ;
0000    28
0000    29  ;++
0000    30  ; FACILITY: VMS Executive floating point emulation
0000    31  ;
0000    32  ; ABSTRACT:
0000    33  ;
0000    34  ;       Loadable code that emulates F, D, G and H floating instructions on
0000    35  ;       any processor.  Octaword integer emulation is included.
0000    36  ;
0000    37  ; ENVIRONMENT: Runs at any access mode, AST Reentrant
0000    38  ;
0000    39  ; AUTHOR: Steven B. Lionel, 22-March-1983
0000    40  ;
0000    41  ;       Emulation code based on LIB$EMULATE by Derek Zave.
0000    42  ;
0000    43  ; MODIFIED BY:
0000    44  ;
0000    45  ;       JCW1010         Jeffrey C. Wiener       21-Aug-1984
0000    46  ;       Added a missing CLRL REG_R1(FP) instruction to the POLYF emulation.
0000    47  ;
0000    48  ;       RNG0009         Rod N. Gamache          14-Aug-1984
0000    49  ;       Only require a P1 control region on exceptions if the exception
0000    50  ;       did not occur in kernel mode (i.e. we have to switch stacks), in
0000    51  ;       the routine VAX$OPCDEC.
0000    52  ;
0000    53  ;       JCW1008         Jeffrey C. Wiener       10-Aug-1984
0000    54  ;       Corrected a BBS instruction in MULTIPLY_FFLOAT. It had referred
0000    55  ;       to FRACTION1+8(FP). It should have been FRACTION1+12(FP)
0000    56  ;
0000    57  ;       LJK0025         Lawrence J. Kenah       8-Mar-1984
```

```
0000     58 ;       Change PRVMOD field in PSL that is in effect while emulator is
0000     59 ;       executing so that PROBEs work with correct access mode when the
0000     60 ;       emulator is used in exec or supervisor mode. Fix incorrect
0000     61 ;       register usage in READ_FAULT.
0000     62 ;
0000     63 ;       LJK0015         Lawrence J. Kenah       2-Feb-1984
0000     64 ;       Fix error destinations for inaccessible instruction stream or
0000     65 ;       exception stack. Use G^ addressing for SYS.STB symbols.
0000     66 ;
0000     67 ;       JCW1005         Jeffrey C. Wiener       11-January-1984
0000     68 ;       Corrected the discription of the algorithm used to divide unsigned
0000     69 ;       multiple length integers. A fix was also added to the associated
0000     70 ;       code to fix an outstanding DIVG/DIVH bug. The fix checks a "carry-
0000     71 ;       over". If the "carry-over" is negative, then the "carry-over" is
0000     72 ;       zeroed.
0000     73 ;
0000     74 ;       SBL1004         Steven B. Lionel        19-October-1983
0000     75 ;       Fix FLOAT_LONG to properly convert -2**32.
0000     76 ;
0000     77 ;       SBL1003         Steven B. Lionel        24-June-1983
0000     78 ;       Correct "emulated opcode" check for 2-byte opcodes.
0000     79 ;
0000     80 ;       SBL1002         Steven B. Lionel        23-May-1983
0000     81 ;       Add check for no P1 region.
0000     82 ;
0000     83 ;       SBL1001         Steven B. Lionel        22-March-1983
0000     84 ;       Adapt LIB$EMULATE for integration into the VMS executive.
0000     85 ;--
```

```
0000     87          .SBTTL  DECLARATIONS
0000     88 ;
0000     89 ; LIBRARY MACRO CALLS:
0000     90 ;
0000     91          $SSDEF                  ; System Status Codes
0000     92          $PSLDEF                 ; Processor Status Longword definitions
0000     93          $PRDEF                  ; Processor register definitions
0000     94          $ARCDEF                 ; Architecture support flag definitions
0000     95
0000     96 ;
0000     97 ; EXTERNAL DECLARATIONS:
0000     98 ;
0000     99          .DSABL  GBL             ; Force all external symbols to be declared
0000    100          .EXTRN  EXE$SRCHANDLER  ; System routine that looks for handlers
0000    101          .EXTRN  EXE$UNWIND      ; Unwind exception
0000    102          .EXTRN  EXE$OPCDEC      ; VMS entry to handle SS$_OPCDEC
0000    103          .EXTRN  EXE$ACVIOLAT    ; VMS entry to handle SS$_ACCVIO
0000    104          .EXTRN  EXE$GL_ARCHFLAG ; Architecture support flags
0000    105          .EXTRN  CTL$AL_STACK    ; Stack limit arrays
0000    106          .EXTRN  CTL$AL_STACKLIM ; Stack limit arrays
0000    107
0000    108 ;
0000    109 ; MACROS:
0000    110 ;
0000    111
0000    112 ;        Macro for Comparing Condition Codes
0000    113          .MACRO  CMPCOND COND,LOC
0000    114          CMPZV   #3,#26,LOC,#COND@-3
0000    115          .ENDM
0000    116
0000    117 ;        Macro for loading OP_TYPES with the types of the various operands,
0000    118          .MACRO  SET_OP_TYPES    OP1=F,OP2=F,OP3=F,OP4=F,OP5=F,OP6=F
0000    119          .NARG   N_ARGS
0000    120          .IF     EQ N_ARGS-6     ; 6 arguments?
0000    121          MOVW    #<TYP_'OP5'+<TYP_'OP6'@8>>, OP_TYPES4(FP)
0000    122          .ENDC
0000    123          .IF     EQ N_ARGS-5     ; 5 arguments?
0000    124          MOVB    #<TYP_'OP5'>, OP_TYPES4(FP)
0000    125          .ENDC
0000    126          .IF     GT N_ARGS-2     ; more than 2 arguments?
0000    127          MOVL    #<TYP_'OP1'+ -
0000    128                   <TYP_'OP2'@8>+ -
0000    129                   <TYP_'OP3'@16>+ -
0000    130                   <TYP_'OP4'@24>>, OP_TYPES(FP)
0000    131          .ENDC
0000    132          .IF     EQ N_ARGS-2     ; 2 arguments?
0000    133          MOVW    #<TYP_'OP1'+<TYP_'OP2'@8>>, OP_TYPES(FP)
0000    134          .ENDC
0000    135          .IF     EQ N_ARGS-1     ; 1 argument?
0000    136          MOVB    #TYP_'OP1', OP_TYPES(FP)
0000    137          .ENDC
0000    138          .ENDM
0000    139
0000    140 ;
0000    141 ; EQUATED SYMBOLS:
0000    142 ;
0000    143 ;        See body of routine
```

```
                        ;
                        ; OWN STORAGE:
                        ;
                        ;       NONE
                        ;
```

```
                     0000   150          ;       ************************************************************************
                     0000   151          ;       *                                                                      *
                     0000   152          ;       *                                                                      *
                     0000   153          ;       *                          Assorted Definitions                        *
                     0000   154          ;       *                                                                      *
                     0000   155          ;       *                                                                      *
                     0000   156          ;       ************************************************************************
                     0000   157          ;
                     0000   158          ;
                     0000   159          ;
                     0000   160          ;       Parameters
                     0000   161          ;
00000028             0000   162 CALL_ARGS =       40                          ; flexible stack space (longwords)
                     0000   163          ;
                     0000   164          ;       Opcode range limits
                     0000   165          ;
00000040             0000   166 LO_1BYTE =       ^X40
00000076             0000   167 HI_1BYTE =       ^X76
00000032             0000   168 LO_2BYTE =       ^X32
000000FF             0000   169 HI_2BYTE =       ^XFF
                     0000   170          ;
                     0000   171          ;       Operand Area Layout
                     0000   172          ;
00000000             0000   173 ZERO     =       0                           ; zero indicator (byte)
00000001             0000   174 SIGN     =       1                           ; sign indicator (byte)
00000004             0000   175 POWER    =       4                           ; exponent (longword)
00000008             0000   176 FRACTION =       8                           ; fraction area (octaword)
00000018             0000   177 OPERAND_SIZE =   24                          ; operand area size (bytes)
                     0000   178          ;
                     0000   179          ;       Bits in the Processor Status Longword (PSL)
                     0000   180          ;
00000000             0000   181 PSL_C    =       0                           ; carry indicator
00000001             0000   182 PSL_V    =       1                           ; overflow indicator
00000002             0000   183 PSL_Z    =       2                           ; zero indicator
00000003             0000   184 PSL_N    =       3                           ; negative indicator
00000004             0000   185 PSL_T    =       4                           ; trace enable indicator
00000005             0000   186 PSL_IV   =       5                           ; integer overflow trap enable
00000006             0000   187 PSL_FU   =       6                           ; floating underflow fault enable
00000018             0000   188 PSL_CAM  =       24                          ; low bit of current access mode
0000001B             0000   189 PSL_FPD  =       27                          ; instruction first part done
0000001E             0000   190 PSL_TP   =       30                          ; trace pending indicator
                     0000   191          ;
                     0000   192          ;       Masks for the Processor Status Longword
                     0000   193          ;
00000001             0000   194 PSLM_C    =       1@PSL_C                     ; carry indicator
00000002             0000   195 PSLM_V    =       1@PSL_V                     ; overflow indicator
00000004             0000   196 PSLM_Z    =       1@PSL_Z                     ; zero indicator
00000008             0000   197 PSLM_N    =       1@PSL_N                     ; negative indicator
00000003             0000   198 PSLM_VC   =       PSLM_V+PSLM_C               ; overflow and carry indicators
0000000F             0000   199 PSLM_NZVC =       PSLM_VC+PSLM_N+PSLM_Z       ; condition code
0000000C             0000   200 PSLM_NZ   =       PSLM_N+PSLM_Z               ; comparison codes
0000000E             0000   201 PSLM_NZV  =       PSLM_NZ+PSLM_V              ; bits other than carry
00000008             0000   202 PSLM_LSS  =       PSLM_N                      ; less than condition code
00000004             0000   203 PSLM_EQL  =       PSLM_Z                      ; equals condition code
00000000             0000   204 PSLM_GTR  =       0                           ; greater than condition code
                     0000   205          ;
                     0000   206          ;       Call Frame Layout
```

```
                                                G 14
                     0000   207                ;
00000000             0000   208 HANDLER =       0                       ; condition handler location
00000004             0000   209 SAVE_PSW =      4                       ; saved processor status word
00000006             0000   210 SAVE_MASK =     6                       ; register save mask
0000000E             0000   211 MASK_ALIGN =    14                      ; bit position of alignment bits
00000008             0000   212 SAVE_AP =       8                       ; user's argument pointer
0000000C             0000   213 SAVE_FP =       12                      ; user's frame pointer
00000010             0000   214 SAVE_PC =       16                      ; return point
00000014             0000   215 REG_R0  =       20                      ; user's R0
00000018             0000   216 REG_R1  =       24                      ; user's R1
0000001C             0000   217 REG_R2  =       28                      ; user's R2
00000020             0000   218 REG_R3  =       32                      ; user's R3
00000024             0000   219 REG_R4  =       36                      ; user's R4
00000028             0000   220 REG_R5  =       40                      ; user's R5
0000002C             0000   221 REG_R6  =       44                      ; user's R6
00000030             0000   222 REG_R7  =       48                      ; user's R7
00000034             0000   223 REG_R8  =       52                      ; user's R8
00000038             0000   224 REG_R9  =       56                      ; user's R9
0000003C             0000   225 REG_R10 =       60                      ; user's R10
00000040             0000   226 REG_R11 =       64                      ; user's R11
00000044             0000   227 FRAME_END =     68                      ; end of call frame
                     0000   228                ;
                     0000   229                ;       Call Frame Extension Layout
                     0000   230                ;
00000044             0000   231 REG_AP  =       68                      ; user's AP
00000048             0000   232 REG_FP  =       72                      ; user's FP
0000004C             0000   233 REG_SP  =       76                      ; user's SP
00000050             0000   234 REG_PC  =       80                      ; user's PC
00000054             0000   235 PSL     =       84                      ; user's PSL
00000058             0000   236 LOCAL_END =     88                      ; end of Emulator local storage
00000058             0000   237 TEMP    =       88                      ; temporary area for arithmetic
                     0000   238                ;
                     0000   239                ;       Local Storage Layout
                     0000   240                ;
FFFFFFFF             0000   241 SAVE_ALIGN =    HANDLER-1               ; saved copy of alignment bits
FFFFFFFE             0000   242 SAVE_PARCNT =   SAVE_ALIGN-1            ; saved copy of parameter count
FFFFFFFD             0000   243 MODE    =       SAVE_PARCNT-1           ; access mode for probes
FFFFFFFC             0000   244 FLAGS   =       MODE-1                  ; indicator flag bits
FFFFFFFB             0000   245 SHORT_LOCAL =   FLAGS-1                 ; start of short local storage
FFFFFFFA             0000   246 REGMOD_PC =     SHORT_LOCAL-1           ; changes to user's PC
FFFFFFF9             0000   247 REGMOD_SP =     REGMOD_PC-1             ; changes to user's SP
FFFFFFF8             0000   248 REGMOD_FP =     REGMOD_SP-1             ; changes to user's FP
FFFFFFF7             0000   249 REGMOD_AP =     REGMOD_FP-1             ; changes to user's AP
FFFFFFF6             0000   250 REGMOD_R11 =    REGMOD_AP-1             ; changes to user's R11
FFFFFFF5             0000   251 REGMOD_R10 =    REGMOD_R11-1            ; changes to user's R10
FFFFFFF4             0000   252 REGMOD_R9 =     REGMOD_R10-1            ; changes to user's R9
FFFFFFF3             0000   253 REGMOD_R8 =     REGMOD_R9-1             ; changes to user's R8
FFFFFFF2             0000   254 REGMOD_R7 =     REGMOD_R8-1             ; changes to user's R7
FFFFFFF1             0000   255 REGMOD_R6 =     REGMOD_R7-1             ; changes to user's R6
FFFFFFF0             0000   256 REGMOD_R5 =     REGMOD_R6-1             ; changes to user's R5
FFFFFFEF             0000   257 REGMOD_R4 =     REGMOD_R5-1             ; changes to user's R4
FFFFFFEE             0000   258 REGMOD_R3 =     REGMOD_R4-1             ; changes to user's R3
FFFFFFED             0000   259 REGMOD_R2 =     REGMOD_R3-1             ; changes to user's R2
FFFFFFEC             0000   260 REGMOD_R1 =     REGMOD_R2-1             ; changes to user's R1
FFFFFFEB             0000   261 REGMOD_R0 =     REGMOD_R1-1             ; changes to user's R0
FFFFFFE7             0000   262 ADDRESS1 =      REGMOD_R0-4             ; temporary address area #1
FFFFFFE3             0000   263 ADDRESS2 =      ADDRESS1-4              ; temporary address area #2
```

```
FFFFFFDF  0000   264 ADDRESS3 =      ADDRESS2-4                  ; temporary address area #3
FFFFFFC7  0000   265 OPERAND1 =      ADDRESS3-OPERAND_SIZE       ; temporary operand area #1
FFFFFFAF  0000   266 OPERAND2 =      OPERAND1-OPERAND_SIZE       ; temporary operand area #2
FFFFFF97  0000   267 OPERAND3 =      OPERAND2-OPERAND_SIZE       ; temporary operand area #3
FFFFFF93  0000   268 OP_TYPES4 =     OPERAND3-4                  ; Four bytes for operand type codes
FFFFFF8F  0000   269 OP_TYPES=       OP_TYPES4-4                 ; Four more bytes for operand type codes
FFFFFF8B  0000   270 OP_INDEX =      OP_TYPES-4                  ; Pointer to current byte of OP_TYPES
FFFFFF8B  0000   271 LOCAL_START =   OP_INDEX                    ; start of Emulator local storage
          0000   272                 ;
          0000   273                 ;       Indicator Bit Numbers
          0000   274                 ;
00000000  0000   275 FLAG0    =      0                           ; inhibit local store check
00000001  0000   276 FLAG1    =      1                           ; register mode operand
00000002  0000   277 FLAG2    =      2                           ; (not assigned)
00000003  0000   278 FLAG3    =      3                           ; (not assigned)
00000004  0000   279 FLAG4    =      4                           ; (not assigned)
00000005  0000   280 FLAG5    =      5                           ; (not assigned)
00000006  0000   281 FLAG6    =      6                           ; (not assigned)
00000007  0000   282 FLAG7    =      7                           ; temporary use
          0000   283                 ;
          0000   284                 ;       Indicator Bit Masks
          0000   285                 ;
00000001  0000   286 FLAG0M   =      1@0                         ; inhibit local store check
00000002  0000   287 FLAG1M   =      1@1                         ; register mode operand
00000004  0000   288 FLAG2M   =      1@2                         ; (not assigned)
00000008  0000   289 FLAG3M   =      1@3                         ; (not assigned)
00000010  0000   290 FLAG4M   =      1@4                         ; (not assigned)
00000020  0000   291 FLAG5M   =      1@5                         ; (not assigned)
00000040  0000   292 FLAG6M   =      1@6                         ; (not assigned)
00000080  0000   293 FLAG7M   =      1@7                         ; temporary use
          0000   294                 ;
          0000   295                 ;       Fields in the Operand Areas
          0000   296                 ;
FFFFFFC7  0000   297 ZERO1    =      OPERAND1+ZERO               ; zero flag of OPERAND1
FFFFFFAF  0000   298 ZERO2    =      OPERAND2+ZERO               ; zero flag of OPERAND2
FFFFFF97  0000   299 ZERO3    =      OPERAND3+ZERO               ; zero flag of OPERAND3
FFFFFFC8  0000   300 SIGN1    =      OPERAND1+SIGN               ; sign of OPERAND1
FFFFFFB0  0000   301 SIGN2    =      OPERAND2+SIGN               ; sign of OPERAND2
FFFFFF98  0000   302 SIGN3    =      OPERAND3+SIGN               ; sign of OPERAND3
FFFFFFCB  0000   303 POWER1   =      OPERAND1+POWER              ; exponent of OPERAND1
FFFFFFB3  0000   304 POWER2   =      OPERAND2+POWER              ; exponent of OPERAND2
FFFFFF9B  0000   305 POWER3   =      OPERAND3+POWER              ; exponent of OPERAND3
FFFFFFCF  0000   306 FRACTION1 =     OPERAND1+FRACTION           ; fraction of OPERAND1
FFFFFFB7  0000   307 FRACTION2 =     OPERAND2+FRACTION           ; fraction of OPERAND2
FFFFFF9F  0000   308 FRACTION3 =     OPERAND3+FRACTION           ; fraction of OPERAND3
          0000   309                 ;
          0000   310                 ;       Access Type Code Definitions
          0000   311                 ;
00000001  0000   312 TYPE_READ =     1                           ; read only access
00000002  0000   313 TYPE_WRITE =    2                           ; write only access
00000003  0000   314 TYPE_MODIFY =   3                           ; modify access
00000004  0000   315 TYPE_ADDRESS =  4                           ; address access
          0000   316                 ;
          0000   317                 ;       Data Type Code Definitions
          0000   318                 ;
00000001  0000   319 TYP_B    =      1                           ; byte
00000002  0000   320 TYP_W    =      2                           ; word
```

```
00000003  0000    321 TYP_L    =      3                       ; longword
00000004  0000    322 TYP_Q    =      4                       ; quadword
00000005  0000    323 TYP_O    =      5                       ; octaword
00000006  0000    324 TYP_F    =      6                       ; F_floating
00000007  0000    325 TYP_D    =      7                       ; D_floating
00000008  0000    326 TYP_G    =      8                       ; G_floating
00000009  0000    327 TYP_H    =      9                       ; H_floating
          0000    328
          0000    329          ;
          0000    330          ; Instruction type definitions for optimization check
          0000    331          ;
00000001  0000    332 IT_D = ARC$M_DFLT_EMUL@-ARC$V_DFLT_EMUL
00000002  0000    333 IT_F = ARC$M_FFLT_EMUL@-ARC$V_DFLT_EMUL
00000004  0000    334 IT_G = ARC$M_GFLT_EMUL@-ARC$V_DFLT_EMUL
00000008  0000    335 IT_H = ARC$M_HFLT_EMUL@-ARC$V_DFLT_EMUL
00000000  0000    336 IT_X = 0            ; none
```

```
                                0000   338          .SUBTITLE       VAX$OPCDEC
                                0000   339 ;+
                                0000   340 ; Functional Description:
                                0000   341 ;
                                0000   342 ;       This routine is entered in kernel mode through the SCB vector for
                                0000   343 ;       the OPCDEC exception. It determines if the instruction that caused
                                0000   344 ;       the fault is one supported by this emulator. If so, the rest of the
                                0000   345 ;       emulation is carried out by an instruction-specific routine. If not,
                                0000   346 ;       control is transferred to the OPCDEC handler that is a part of the
                                0000   347 ;       VMS executive to allow normal exception dispatching to take place.
                                0000   348 ;
                                0000   349 ; Input Parameters:
                                0000   350 ;
                                0000   351 ;       0(SP) - PC of faulting instruction
                                0000   352 ;       4(SP) - PSL at the time of the fault
                                0000   353 ;
                                0000   354 ; Output Parameters:
                                0000   355 ;
                                0000   356 ;       The real output from this routine is the routine to which control
                                0000   357 ;       is passed, namely the routine that handles each separate instruction.
                                0000   358 ;-
                                0000   359
                             00000000   360          .PSECT  VAX$FPE_NONPAGED PIC, USR, CON, REL, LCL, SHR,-
                                0000   361                  EXE, RD, NOWRT, QUAD
                                0000   362          .ALIGN  QUAD
                                0000   363
                                0000   364 VAX$OPCDEC::
                                0000   365
                        7E  50  7D  0000   366          MOVQ    R0, -(SP)               ; Save R0 and R1
                                0003   367
                                0003   368 ; It is not clear whether the following PROBER is necessary. One approach is
                                0003   369 ; that an inaccessible opcode would cause an access violation rather than
                                0003   370 ; a reserved instruction exception.
                                0003   371
                             50  08 AE  D0  0003   372          MOVL    8(SP), R0               ; R0 contains the PC of the instruction
                                0007   373          IFNORD  #1, (R0), 15$           ; Insure that opcode can be read
                                   51  80  9A  000D   374          MOVZBL  (R0)+, R1               ; Get first opcode byte, increment "PC"
                          74 00A5'CF   51  E1  0010   375          BBC     R1, W^OP_MASK_1BYTE, 90$; Test for opcode not emulated
                                FD 8F   51  91  0016   376          CMPB    R1, #^XFD               ; Is it a two-byte opcode?
                                   0F   12  001A   377          BNEQ    10$                     ; Skip if not
                                001C   378          IFNORD  #1, (R0), 15$           ; Can second byte be read?
                                   51  60  9A  0022   379          MOVZBL  (R0), R1                ; Fetch second byte
                          5F 00C5'CF   51  E1  0025   380          BBC     R1, W^OP_MASK_2BYTE, 90$; Test for opcode not emulated
                                002B   381
                                002B   382 ;+
                                002B   383 ; Now the kernel stack looks as follows:
                                002B   384 ;
                                002B   385 ;       00(SP)  - Saved R0
                                002B   386 ;       04(SP)  - Saved R1
                                002B   387 ;       08(SP)  - PC of instruction
                                002B   388 ;       12(SP)  - PSL of instruction
                                002B   389 ;
                                002B   390 ;
                                002B   391 ; Switch stacks to that of the exception, push onto that stack the PC/PSL.
                                002B   392 ; Push the PSL with T, TP and FPD bits clear, the address of
                                002B   393 ; EMULATE_FP, and then REI to EMULATE_FP.
                                002B   394 ;-
```

```
                                002B   395
      51   0C AE   02   18   EF  002B   396 10$:    EXTZV   #PSL$V_CURMOD,#PSL$S_CURMOD,12(SP),R1 ; Get previous mode
                        51   13  0031   397         BEQL    20$                     ; Br if kernel, no stack change needed
                                0033   398         IFNORD  #4,G^CTL$AL_STACK,90$   ; Is there a control region?
                                003D   399                                         ; Br if not, handle as a real exception
                                003D   400         ASSUME  PSL$C_KERNEL EQ 0
                                003D   401         ASSUME  PR$_ESP EQ PSL$C_EXEC
                                003D   402         ASSUME  PR$_SSP EQ PSL$C_SUPER
                                003D   403         ASSUME  PR$_USP EQ PSL$C_USER
                   50   51   DB  003D   404         MFPR    R1,R0                   ; Get address of stack of excpt mode
                                0040   405         IFNOWRT #8,-8(R0),40$           ; Br if cannot copy to excpt mode stk
      00000000'GF41   50   D1  0047   406         CMPL    R0,G^CTL$AL_STACK[R1]   ; Top address of stack in range?
                        42   1A  004F   407         BGTRU   40$                     ; If GTRU no.
                   50   08   C2  0051   408         SUBL2   #8,R0                   ; Get new low stack address
                   51   03   D1  0054   409         CMPL    #PSL$C_USER,R1          ; Previous mode user?
                        0A   13  0057   410         BEQL    12$                     ; If EQL yes.
      00000000'GF41   50   D1  0059   411         CMPL    R0,G^CTL$AL_STACKLIM[R1]; Bottom address of stack in range?
                        30   1F  0061   412         BLSSU   40$                     ; If LSSU no.
               60   08   AE   7D  0063   413 12$:    MOVQ    8(SP),(R0)              ; Push PC/PSL
                   51   50   DA  0067   414         MTPR    R0,R1                   ; Set stack pointer to new value
       0C AE   48000010   8F   CA  006A   415         BICL2   #<PSL$M_TP!PSL$M_TBIT!PSL$M_FPD>,12(SP) ; Clear bits in PSL
    0C AE   02   16   51   F0  0072   416         INSV    R1,#PSL$V_PRVMOD,#PSL$S_PRVMOD,12(SP)   ; Set PRVMOD = CURMOD
         08 AE   0000'CF   9E  0078   417         MOVAB   W^VAX$$EMULATE_FP,8(SP) ; Set entry address
                   50   8E   7D  007E   418         MOVQ    (SP)+,R0                ; Pop saved R0 and R1
                        02   0081   419         REI                             ; Jump to EMULATE_FP
                                0082   420
                        0F   11  0082   421 15$:    BRB     40$                     ; Chain branch for ACCVIO test
                                0084   422
                                0084   423 ;+
                                0084   424 ; We come here if the previous mode is kernel, since we don't need to
                                0084   425 ; switch modes.
                                0084   426 ;-
                                0084   427
         50   8E   7D  0084   428 20$:    MOVQ    (SP)+,R0                ; Pop saved R0 and R1
            FF76'   31  0087   429         BRW     VAX$$EMULATE_FP         ; Emulate the instruction
                                008A   430
                                008A   431 ;+
                                008A   432 ; If we don't handle the instruction, remove the things we pushed on the
                                008A   433 ; stack and jump to the EXECPTION code that handles SS$_OPCDEC
                                008A   434 ;-
                                008A   435
         50   8E   7D  008A   436 90$:    MOVQ    (SP)+,R0                ; Restore R0-R1
    00000000'GF   17  008D   437         JMP     G^EXE$OPCDEC            ; Reflect exception
                                0093   438
                                0093   439 ; If a PROBE of an outer access mode fails, then the exception is reported
                                0093   440 ; as an access violation rather than as a reserved instruction. This allows
                                0093   441 ; the stack expansion logic and other such things to be invoked without this
                                0093   442 ; emulator worrying about such things.
                                0093   443
         7E   6E   7D  0093   444 40$:    MOVQ    (SP),-(SP)              ; Move saved R0-R1
    0C AE   50   D0  0096   445         MOVL    R0,12(SP)               ; Store inaccessible stack address
         50   8E   7D  009A   446         MOVQ    (SP)+,R0                ; Restore R0-R1
                                009D   447
                                009D   448 ; Note that the access violation mask generated here (always zero) is not
                                009D   449 ; correct for the extremely rare case of an inaccessible exception stack.
                                009D   450
                   6E   D4  009D   451         CLRL    (SP)                    ; Set mask to indicate a read
```

```
                00000000'GF   17  009F   452          JMP     G^EXE$ACVIOLAT          ; Pass control to VMS handler
                              00A5   453
```

```
                        00A5   455          .SBTTL  Mask of emulated opcodes
                        00A5   456
                        00A5   457          ;
                        00A5   458          ;       OP_MASK_1BYTE and OP_MASK_2BYTE are 256-bit bitmasks with
                        00A5   459          ;       bits set corresponding to opcodes of instructions that
                        00A5   460          ;       this emulator can handle.
                        00A5   461          ;
                        00A5   462
                        00A5   463 OP_MASK_1BYTE:           ; Mask for 1-byte opcodes (including FD)
                        00A5   464 ;                          FEDCBA9876543210FEDCBA9876543210
00000000                00A5   465          .LONG   ^B00000000000000000000000000000000      ; 00-1F
00000000                00A9   466          .LONG   ^B00000000000000000000000000000000      ; 20-3F
007FFFFF                00AD   467          .LONG   ^B00000000011111111111111111111111      ; 40-5F
007FFFFF                00B1   468          .LONG   ^B00000000011111111111111111111111      ; 60-7F
00000000                00B5   469          .LONG   ^B00000000000000000000000000000000      ; 80-9F
00000000                00B9   470          .LONG   ^B00000000000000000000000000000000      ; A0-BF
00000000                00BD   471          .LONG   ^B00000000000000000000000000000000      ; C0-DF
20000000                00C1   472          .LONG   ^B00100000000000000000000000000000      ; E0-FF
                        00C5   473
                        00C5   474 OP_MASK_2BYTE:           ; Mask for 2-byte opcodes (second byte index)
                        00C5   475 ;                          FEDCBA9876543210FEDCBA9876543210
00000000                00C5   476          .LONG   ^B00000000000000000000000000000000      ; 00-1F
000C0000                00C9   477          .LONG   ^B00000000000011000000000000000000      ; 20-3F
007FFFFF                00CD   478          .LONG   ^B00000000011111111111111111111111      ; 40-5F
F07FFFFF                00D1   479          .LONG   ^B11110000011111111111111111111111      ; 60-7F
03000000                00D5   480          .LONG   ^B00000011000000000000000000000000      ; 80-9F
00000000                00D9   481          .LONG   ^B00000000000000000000000000000000      ; A0-BF
00000000                00DD   482          .LONG   ^B00000000000000000000000000000000      ; C0-DF
00C00000                00E1   483          .LONG   ^B00000000110000000000000000000000      ; E0-FF
```

```
                                00E5    485          .SBTTL  VAX$$EMULATE_FP
                                00E5    486
                                00E5    487          ;
                                00E5    488          ;       VAX$$EMULATE_FP - Emulator Entrance
                                00E5    489          ;
                                00E5    490          ;               entered by branching
                                00E5    491          ;
                                00E5    492          ;               parameters      0(SP)   Instruction PC
                                00E5    493          ;                               4(SP)   Instruction PSL
                                00E5    494          ;
                                00E5    495          ;       Discussion
                                00E5    496          ;
                                00E5    497          ;           This routine provides a simple method of activating the
                                00E5    498          ;       Emulator. The PC and PSL for the instruction being emulated
                                00E5    499          ;       are pushed onto the stack and the routine is entered. The
                                00E5    500          ;       routine simply allocates the simulated user stack space for
                                00E5    501          ;       the Emulator and calls the emulation procedure.
                                00E5    502          ;
                                00E5    503          ;       Notes:  1. The FPD bit may be set in the pushed PSL. This
                                00E5    504          ;                  bit will only be interpreted during the emulation
                                00E5    505          ;                  of the POLYx instructions where it is
                                00E5    506          ;                  interpreted as described in the VAX System
                                00E5    507          ;                  Reference Manual.
                                00E5    508          ;
                                00E5    509          ;
                                00E5    510
                            00000000    511          .PSECT  VAX$FPE_PAGED   PIC, USR, CON, REL, LCL, SHR,-
                                0000    512                  EXE, RD, NOWRT, QUAD
                                0000    513
                                0000    514  VAX$$EMULATE_FP::
5E   FF68 CE   9E               0000    515          MOVAB   -4*<CALL_ARGS-2>(SP),SP ; allocate the parameter block
  OF'AF   28   FB               0005    516          CALLS   #CALL_ARGS,B^EMULATOR   ; call the Emulator
 00000000'GF   17               0009    517          JMP     G^EXE$OPCDEC            ; shouldn't return here -
                                000F    518                                          ; indicate OPCDEC as fallback
                                000F    519          ;
```

```
                        .SBTTL  EMULATOR
                ;
                ;       EMULATOR - Start Instruction Emulation
                ;
                ;               parameters:     ( Described Below )
                ;
                ;       Discussion
                ;
                ;           This routine initializes the Emulator, processes the
                ;       instruction code, and passes control to the appropriate
                ;       instruction emulation routine. The parameter list consists of
                ;       CALL_ARGS longwords of which only the last two have any
                ;       meaning. These parameters are the PC and PSL for the emulation.
                ;       The position following the parameter list is the
                ;       user's stack pointer. The area covered by the parameter list
                ;       is used to emulate the top of the user's stack.
                ;
                ;           When the routine is entered the CALLS instruction saves
                ;       the user's registers R0 to R11 in order and saves AP and FP
                ;       elsewhere in the frame. The routine extends the saved
                ;       registers by saving the user's AP, FP, SP, PC, and PSL after
                ;       the saved register area (the last two are taken from the
                ;       parameter list).
                ;
                ;           The local storage is allocated by extending the stack and
                ;       the register modification bytes are cleared (these bytes
                ;       record small changes to the register values). The cell MODE
                ;       is set equal to the current access mode for use in probing
                ;       memory references. The alignment bits in the call frame and
                ;       the call parameter count are also saved so there is a safe
                ;       copies to use when processing unwinds.
                ;
                ;       Notes:  1. From the description of the way the simulated
                ;                  register area is constructed, it is clear that
                ;                  the length longword of the parameter list is
                ;                  overwritten. All of the methods of leaving the
                ;                  Emulator put this longword back together. The
                ;                  internal condition handler does this if it
                ;                  detects an unwind.
                ;
                ;               2. Here are some more details on the register change
                ;                  bytes: Until the very end of instruction processing
                ;                  when the results are output, all of the changes
                ;                  which occur to the registers are caused by adding
                ;                  or subtracting small values to or from a register.
                ;                  These changes are also recorded in a corresponding
                ;                  register change byte so the register may be
                ;                  restored to its original value if a fault occurs.
                ;                  Those instructions that save results in the
                ;                  registers in order to be interruptable, use the FPD
                ;                  bit in the PSL to indicate that this has been done
                ;                  so different logic should be used for restarting
                ;                  the instruction after a fault. In this case the
                ;                  change bytes should be cleared when FPD is set
                ;                  except for the one for PC.
```

```
                          000F           ;                        3. The location of the instruction being emulated is
                          000F           ;                           stored in the return PC for the Emulator's frame
                          000F           ;                           so it may be easily located from the traceback
                          000F           ;                           report if the Emulator blows up.
                          000F           ;
                          000F  EMULATOR:                                          ; entrance
                    0FFF  000F           .WORD   ^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; entry mask
       5E   8B AD    9E   0011           MOVAB   LOCAL_START(FP),SP              ; allocate the local storage
 50  06 AD  02  0E   EF   0015           EXTZV   #MASK_ALIGN,#2,SAVE_MASK(FP),R0 ; R0 = alignment bits
       FF AD    50   90   001B           MOVB    R0,SAVE_ALIGN(FP)               ; save them
       FE AD    28   90   001F           MOVB    #CALL_ARGS,SAVE_PARCNT(FP)      ; save the parameter count
            FC AD    94   0023           CLRB    FLAGS(FP)                       ; clear the flag bits
       6D  0357'CF   9E   0026           MOVAB   W^COND_HANDLER,HANDLER(FP)      ; set up the condition handler
            EB AD    7C   002B           CLRQ    REGMOD_R0(FP)                   ; clear register modification bytes
            F3 AD    7C   002E           CLRQ    REGMOD_R8(FP)                   ; clear register modification bytes
    44 AD   08 AD    7D   0031           MOVQ    SAVE_AP(FP),REG_AP(FP)          ; move user's AP and FP into place
    4C AD   00A4 CC  9E   0036           MOVAB   4*<CALL_ARGS+1>(AP),REG_SP(FP)  ; move user's SP into place
    50 AD   009C CC  7D   003C           MOVQ    4*<CALL_ARGS-1>(AP),REG_PC(FP)  ; move PC and PSL into place
 50  54 AD  02  18   EF   0042           EXTZV   #PSL_CAM,#2,PSL(FP),R0          ; R0 = user's access mode
       FD AD    50   90   0048           MOVB    R0,MODE(FP)                     ; save it for probes
       5B   50 AD    D0   004C           MOVL    REG_PC(FP),R11                  ; get instruction PC
       10 AD    5B   D0   0050           MOVL    R11,SAVE_PC(FP)                 ; save PC in the return PC
    8B AD   8F AD    9E   0054           MOVAB   OP_TYPES(FP), OP_INDEX(FP); Initialize ptr to operand type codes
         50     8B   9A   0059           MOVZBL  (R11)+,R0                       ; get first opcode byte
            50 AD    D6   005C           INCL    REG_PC(FP)                      ; increment emulated PC
            FA AD    D6   005F           INCL    REGMOD_PC(FP)                   ; remember the increment
    05 54 AD    04   E1   0062           BBC     #PSL_T,PSL(FP),10$              ; check for T-bit
    00 54 AD    1E   E2   0067           BBSS    #PSL_TP,PSL(FP),10$             ; set TP if T set
       54 AD    02   CA   006C  10$:     BICL2   #PSLM_V,PSL(FP)                 ; clear the V bit in the PSL
       FD 8F    50   91   0070           CMPB    R0,#^XFD                        ; two-byte instruction?
                0C   12   0074           BNEQ    DISPATCH_1BYTE                  ; skip if not
         50     8B   9A   0076           MOVZBL  (R11)+,R0                       ; get next opcode byte
            50 AD    D6   0079           INCL    REG_PC(FP)                      ; increment emulated PC
            FA AD    D6   007C           INCL    REGMOD_PC(FP)                   ; remember the increment
            0076     31   007F           BRW     DISPATCH_2BYTE                  ; execute 2-byte opcode
```

```
                                0082   614          .SBTTL  DISPATCH_1BYTE
                                0082   615
                                0082   616          ;+
                                0082   617          ;       DISPATCH_1BYTE - Branch to emulation routine for 1-byte opcode
                                0082   618          ;
                                0082   619          ;               entered by branching
                                0082   620          ;
                                0082   621          ;               R0 contains opcode
                                0082   622          ;
                                0082   623          ;-
                                0082   624  DISPATCH_1BYTE:
          36   40 8F    50   8F  0082   625          CASEB   R0,#^X40,#<^X76-^X40>   ; from ADDF2 through CVTDF
                        037F' 0087   626  1$:        .WORD   INST_ADDF2-1$           ;40     ADDF2
                        03B8' 0089   627          .WORD   INST_ADDF3-1$           ;41     ADDF3
                        0D94' 008B   628          .WORD   INST_SUBF2-1$           ;42     SUBF2
                        0DD0' 008D   629          .WORD   INST_SUBF3-1$           ;43     SUBF3
                        097C' 008F   630          .WORD   INST_MULF2-1$           ;44     MULF2
                        09B5' 0091   631          .WORD   INST_MULF3-1$           ;45     MULF3
                        0606' 0093   632          .WORD   INST_DIVF2-1$           ;46     DIVF2
                        063F' 0095   633          .WORD   INST_DIVF3-1$           ;47     DIVF3
                        04BF' 0097   634          .WORD   INST_CVTFB-1$           ;48     CVTFB
                        04DF' 0099   635          .WORD   INST_CVTFW-1$           ;49     CVTFW
                        04FF' 009B   636          .WORD   INST_CVTFL-1$           ;4A     CVTFL
                        05C7' 009D   637          .WORD   INST_CVTRFL-1$          ;4B     CVTRFL
                        0441' 009F   638          .WORD   INST_CVTBF-1$           ;4C     CVTBF
                        0461' 00A1   639          .WORD   INST_CVTWF-1$           ;4D     CVTWF
                        0481' 00A3   640          .WORD   INST_CVTLF-1$           ;4E     CVTLF
                        0301' 00A5   641          .WORD   INST_ACBF-1$            ;4F     ACBF
                        090F' 00A7   642          .WORD   INST_MOVF-1$            ;50     MOVF
                        040A' 00A9   643          .WORD   INST_CMPF-1$            ;51     CMPF
                        08B8' 00AB   644          .WORD   INST_MNEGF-1$           ;52     MNEGF
                        0E0F' 00AD   645          .WORD   INST_TSTF-1$            ;53     TSTF
                        067B' 00AF   646          .WORD   INST_EMODF-1$           ;54     EMODF
                        09F1' 00B1   647          .WORD   INST_POLYF-1$           ;55     POLYF
                        0559' 00B3   648          .WORD   INST_CVTFD-1$           ;56     CVTFD
                        006E' 00B5   649          .WORD   10$-T$                  ;57
                        006E' 00B7   650          .WORD   10$-1$                  ;58     ADAWI
                        006E' 00B9   651          .WORD   10$-1$                  ;59
                        006E' 00BB   652          .WORD   10$-1$                  ;5A
                        006E' 00BD   653          .WORD   10$-1$                  ;5B
                        006E' 00BF   654          .WORD   10$-1$                  ;5C     INSQHI
                        006E' 00C1   655          .WORD   10$-1$                  ;5D     INSQTI
                        006E' 00C3   656          .WORD   10$-1$                  ;5E     REMQHI
                        006E' 00C5   657          .WORD   10$-1$                  ;5F     REMQTI
                        0385' 00C7   658          .WORD   INST_ADDD2-1$           ;60     ADDD2
                        03BE' 00C9   659          .WORD   INST_ADDD3-1$           ;61     ADDD3
                        0D9A' 00CB   660          .WORD   INST_SUBD2-1$           ;62     SUBD2
                        0DD6' 00CD   661          .WORD   INST_SUBD3-1$           ;63     SUBD3
                        0982' 00CF   662          .WORD   INST_MULD2-1$           ;64     MULD2
                        09BB' 00D1   663          .WORD   INST_MULD3-1$           ;65     MULD3
                        060C' 00D3   664          .WORD   INST_DIVD2-1$           ;66     DIVD2
                        0645' 00D5   665          .WORD   INST_DIVD3-1$           ;67     DIVD3
                        04C7' 00D7   666          .WORD   INST_CVTDB-1$           ;68     CVTDB
                        04E7' 00D9   667          .WORD   INST_CVTDW-1$           ;69     CVTDW
                        0507' 00DB   668          .WORD   INST_CVTDL-1$           ;6A     CVTDL
                        05CF' 00DD   669          .WORD   INST_CVTRDL-1$          ;6B     CVTRDL
                        0449' 00DF   670          .WORD   INST_CVTBD-1$           ;6C     CVTBD
```

```
                0469' 00E1    671          .WORD   INST_CVTWD-1$           ; 6D    CVTWD
                0489' 00E3    672          .WORD   INST_CVTLD-1$           ; 6E    CVTLD
                0307' 00E5    673          .WORD   INST_ACBD-1$            ; 6F    ACBD
                0915' 00E7    674          .WORD   INST_MOVD-1$            ; 70    MOVD
                0410' 00E9    675          .WORD   INST_CMPD-1$            ; 71    CMPD
                08BE' 00EB    676          .WORD   INST_MNEGD-1$           ; 72    MNEGD
                0E15' 00ED    677          .WORD   INST_TSTD-1$            ; 73    TSTD
                0729' 00EF    678          .WORD   INST_EMODD-1$           ; 74    EMODD
                0AF5' 00F1    679          .WORD   INST_POLYD-1$           ; 75    POLYD
                0571' 00F3    680          .WORD   INST_CVTDF-1$           ; 76    CVTDF
                      00F5    681
      1B6C    31  00F5    682 10$:    BRW     OPCODE_FAULT                ; unrecognized opcode
```

```
                        00F8   684          .SBTTL  DISPATCH_2BYTE
                        00F8   685
                        00F8   686          ;
                        00F8   687          ;       DISPATCH_2BYTE - Branch to emulation routine for 2-byte opcode
                        00F8   688          ;
                        00F8   689          ;               entered by branching
                        00F8   690          ;
                        00F8   691          ;               R0 contains second opcode byte
                        00F8   692          ;
                        00F8   693
                        00F8   694 DISPATCH_2BYTE:
      4D 8F    32   50   8F   00F8   695          CASEB   R0,#^X32,#<^X7F-^X32>   ; Covers CVTDH through PUSHAO
                0503' 00FD   696 1$:       .WORD   INST_CVTDH-1$           ; 32FD  CVTDH
                050B' 00FF   697           .WORD   INST_CVTGF-1$           ; 33FD  CVTGF
                009C' 0101   698           .WORD   10$-1$                  ; 34FD
                009C' 0103   699           .WORD   10$-1$                  ; 35FD
                009C' 0105   700           .WORD   10$-1$                  ; 36FD
                009C' 0107   701           .WORD   10$-1$                  ; 37FD
                009C' 0109   702           .WORD   10$-1$                  ; 38FD
                009C' 010B   703           .WORD   10$-1$                  ; 39FD
                009C' 010D   704           .WORD   10$-1$                  ; 3AFD
                009C' 010F   705           .WORD   10$-1$                  ; 3BFD
                009C' 0111   706           .WORD   10$-1$                  ; 3CFD
                009C' 0113   707           .WORD   10$-1$                  ; 3DFD
                009C' 0115   708           .WORD   10$-1$                  ; 3EFD
                009C' 0117   709           .WORD   10$-1$                  ; 3FFD
                0315' 0119   710           .WORD   INST_ADDG2-1$           ; 40FD  ADDG2
                034E' 011B   711           .WORD   INST_ADDG3-1$           ; 41FD  ADDG3
                0D2A' 011D   712           .WORD   INST_SUBG2-1$           ; 42FD  SUBG2
                0D66' 011F   713           .WORD   INST_SUBG3-1$           ; 43FD  SUBG3
                0912' 0121   714           .WORD   INST_MULG2-1$           ; 44FD  MULG2
                094B' 0123   715           .WORD   INST_MULG3-1$           ; 45FD  MULG3
                059C' 0125   716           .WORD   INST_DIVG2-1$           ; 46FD  DIVG2
                05D5' 0127   717           .WORD   INST_DIVG3-1$           ; 47FD  DIVG3
                0459' 0129   718           .WORD   INST_CVTGB-1$           ; 48FD  CVTGB
                0479' 012B   719           .WORD   INST_CVTGW-1$           ; 49FD  CVTGW
                0499' 012D   720           .WORD   INST_CVTGL-1$           ; 4AFD  CVTGL
                0561' 012F   721           .WORD   INST_CVTRGL-1$          ; 4BFD  CVTRGL
                03DB' 0131   722           .WORD   INST_CVTBG-1$           ; 4CFD  CVTBG
                03FB' 0133   723           .WORD   INST_CVTWG-1$           ; 4DFD  CVTWG
                041B' 0135   724           .WORD   INST_CVTLG-1$           ; 4EFD  CVTLG
                0297' 0137   725           .WORD   INST_ACBG-1$            ; 4FFD  ACBG
                08A5' 0139   726           .WORD   INST_MOVG-1$            ; 50FD  MOVG
                03A0' 013B   727           .WORD   INST_CMPG-1$            ; 51FD  CMPG
                084E' 013D   728           .WORD   INST_MNEGG-1$           ; 52FD  MNEGG
                0DA5' 013F   729           .WORD   INST_TSTG-1$            ; 53FD  TSTG
                06D1' 0141   730           .WORD   INST_EMODG-1$           ; 54FD  EMODG
                0A89' 0143   731           .WORD   INST_POLYG-1$           ; 55FD  POLYG
                0513' 0145   732           .WORD   INST_CVTGH-1$           ; 56FD  CVTGH
                009C' 0147   733           .WORD   10$-1$                  ; 57FD
                009C' 0149   734           .WORD   10$-1$                  ; 58FD
                009C' 014B   735           .WORD   10$-1$                  ; 59FD
                009C' 014D   736           .WORD   10$-1$                  ; 5AFD
                009C' 014F   737           .WORD   10$-1$                  ; 5BFD
                009C' 0151   738           .WORD   10$-1$                  ; 5CFD
                009C' 0153   739           .WORD   10$-1$                  ; 5DFD
                009C' 0155   740           .WORD   10$-1$                  ; 5EFD
```

```
009C' 0157   741          .WORD   10$-1$                  ; 5FFD
031B' 0159   742          .WORD   INST_ADDH2-1$           ; 60FD  ADDH2
0354' 015B   743          .WORD   INST_ADDH3-1$           ; 61FD  ADDH3
0D30' 015D   744          .WORD   INST_SUBH2-1$           ; 62FD  SUBH2
0D6C' 015F   745          .WORD   INST_SUBH3-1$           ; 63FD  SUBH3
0918' 0161   746          .WORD   INST_MULH2-1$           ; 64FD  MULH2
0951' 0163   747          .WORD   INST_MULH3-1$           ; 65FD  MULH3
05A2' 0165   748          .WORD   INST_DIVH2-1$           ; 66FD  DIVH2
05DB' 0167   749          .WORD   INST_DIVH3-1$           ; 67FD  DIVH3
0461' 0169   750          .WORD   INST_CVTHB-1$           ; 68FD  CVTHB
0481' 016B   751          .WORD   INST_CVTHW-1$           ; 69FD  CVTHW
04A1' 016D   752          .WORD   INST_CVTHL-1$           ; 6AFD  CVTHL
0569' 016F   753          .WORD   INST_CVTRHL-1$          ; 6BFD  CVTRHL
03E3' 0171   754          .WORD   INST_CVTBH-1$           ; 6CFD  CVTBH
0403' 0173   755          .WORD   INST_CVTWH-1$           ; 6DFD  CVTWH
0423' 0175   756          .WORD   INST_CVTLH-1$           ; 6EFD  CVTLH
029D' 0177   757          .WORD   INST_ACBH-1$            ; 6FFD  ACBH
08AB' 0179   758          .WORD   INST_MOVH-1$            ; 70FD  MOVH
03A6' 017B   759          .WORD   INST_CMPH-1$            ; 71FD  CMPH
0854' 017D   760          .WORD   INST_MNEGH-1$           ; 72FD  MNEGH
0DAB' 017F   761          .WORD   INST_TSTH-1$            ; 73FD  TSTH
0785' 0181   762          .WORD   INST_EMODH-1$           ; 74FD  EMODH
0B98' 0183   763          .WORD   INST_POLYH-1$           ; 75FD  POLYH
052B' 0185   764          .WORD   INST_CVTHG-1$           ; 76FD  CVTHG
009C' 0187   765          .WORD   10$-T$                  ; 77FD
009C' 0189   766          .WORD   10$-1$                  ; 78FD
009C' 018B   767          .WORD   10$-1$                  ; 79FD
009C' 018D   768          .WORD   10$-1$                  ; 7AFD
009C' 018F   769          .WORD   10$-1$                  ; 7BFD
037E' 0191   770          .WORD   INST_CLRO-1$            ; 7CFD  CLRO,CLRH
08D7' 0193   771          .WORD   INST_MOVO-1$            ; 7DFD  MOVO
0878' 0195   772          .WORD   INST_MOVAO-1$           ; 7EFD  MOVAO,MOVAH
0CEB' 0197   773          .WORD   INST_PUSHAO-1$          ; 7FFD  PUSHAO,PUSHAH
             0199   774
99 8F  50    91 0199   775 10$:    CMPB    R0,#^X99        ; CVTFG?
       0E    1A 019D   776         BGTRU   12$             ; Skip if not
       09    13 019F   777         BEQL    11$             ; Skip if yes
98 8F  50    91 01A1   778         CMPB    R0,#^X98        ; CVTFH?
       1A    12 01A5   779         BNEQ    90$             ; Error if not
     0446    31 01A7   780         BRW     INST_CVTFH      ; Execute CVTFH
     043B    31 01AA   781 11$:    BRW     INST_CVTFG      ; Execute CVTFG
F6 8F  50    91 01AD   782 12$:    CMPB    R0,#^XF6        ; CVTHF?
       05    1A 01B1   783         BGTRU   13$             ; Skip if not
       0C    12 01B3   784         BNEQ    90$             ; Error if LSSU
     0460    31 01B5   785         BRW     INST_CVTHF      ; Execute CVTHF
F7 8F  50    91 01B8   786 13$:    CMPB    R0,#^XF7        ; CVTHD?
       03    12 01BC   787         BNEQ    90$             ; Error if not
     045F    31 01BE   788         BRW     INST_CVTHD      ; Execute CVTHD
             01C1   789
     1AA0    31 01C1   790 90$:    BRW     OPCODE_FAULT    ; Shouldn't happen
             01C4   791
```

```
                        01C4    793          .SBTTL  NORMAL_EXIT
                        01C4    794
                        01C4    795          ;
                        01C4    796          ;       NORMAL_EXIT - Normal End of Instruction Emulation
                        01C4    797          ;
                        01C4    798          ;               entered by branching
                        01C4    799          ;
                        01C4    800          ;               no parameters
                        01C4    801          ;
                        01C4    802          ;       Discussion
                        01C4    803          ;
                        01C4    804          ;           This routine restores control to the user program whenever
                        01C4    805          ;       the emulation ends without causing an exception. First the V
                        01C4    806          ;       and IV bits in the user's PSL are checked. If they are both
                        01C4    807          ;       set then an integer overflow trap is signaled. When the
                        01C4    808          ;       Emulator returns, all of the registers, PC, and the PSL are
                        01C4    809          ;       set to the emulated values.
                        01C4    810          ;
                        01C4    811          ;           The method of leaving the Emulator consists of pushing the
                        01C4    812          ;       user's PC and PSL onto the user's stack putting the saved AP
                        01C4    813          ;       and FP back in their proper places in the frame and performing
                        01C4    814          ;       the indicated adjustment so that when a RET instruction is
                        01C4    815          ;       executed all of the registers up to FP will be restored and
                        01C4    816          ;       the stack pointer will be positioned to the PC, PSL pair.
                        01C4    817          ;
                        01C4    818          ;           At this point, a speed optimization is performed.  If the
                        01C4    819          ;       next instruction is also one we emulate, then we can bypass the
                        01C4    820          ;       overhead for the reserved opcode fault and exception
                        01C4    821          ;       dispatching by merely looping back to the beginning of the
                        01C4    822          ;       emulator.  After the RET restores all registers, the stack
                        01C4    823          ;       contains the PC/PSL pair for the next instruction.  The PSL is
                        01C4    824          ;       examined to see if the T-bit is set.  If so, we can't do the
                        01C4    825          ;       optimization.  If T is clear, we then examine the next opcode.
                        01C4    826          ;       If it is one which we emulate, a branch is made to EMULATE$
                        01C4    827          ;       to begin emulation of the next instruction.  Note that the
                        01C4    828          ;       "arguments" to EMULATE$, a PC/PSL pair, are already in place!
                        01C4    829          ;
                        01C4    830          ;       If the optimization can not be performed, an REI is executed
                        01C4    831          ;       which restores the PC and PSL for the next instruction.
                        01C4    832          ;
                        01C4    833  NORMAL_EXIT:                                  ; entrance
        08 54 AD    01  E1  01C4    834          BBC     #PSL_V,PSL(FP),1$          ; no integer overflow - bypass
        03 54 AD    05  E1  01C9    835          BBC     #PSL_IV,PSL(FP),1$         ; no integer overflow trap - skip
              1AF9      31  01CE    836          BRW     INT_OVERFLOW               ; process the integer overflow trap
           5E   FB AD   9E  01D1    837  1$:     MOVAB   SHORT_LOCAL(FP),SP         ; shorten the frame
              50   08   D0  01D5    838          MOVL    #8,R0                      ; R0 = size of PC, PSL pair
                00FF    30  01D8    839          BSBW    TEST_FRAME                 ; make sure we have room to push it
           4C AD    08  C2  01DB    840          SUBL2   #8,REG_SP(FP)              ; allocate room on the user's stack
        4C BD   50 AD   7D  01DF    841          MOVQ    REG_PC(FP),@REG_SP(FP)     ; push the PC, PSL pair
        08 AD   44 AD   7D  01E4    842          MOVQ    REG_AP(FP),SAVE_AP(FP)     ; put the user's PC, PSL pair back
        10 AD   0C'AF   9E  01E9    843          MOVAB   B^2$,SAVE_PC(FP)           ; store our return point
           50   48 AD   9E  01EE    844          MOVAB   FRAME_END+4(FP),R0         ; R0 = location of end of frame
     51    4C AD    50  C3  01F2    845          SUBL3   R0,REG_SP(FP),R1           ; R1 = distance of user SP from it
  52    51    02    00  EF  01F7    846          EXTZV   #0,#2,R1,R2                ; R2 = stack alignment
06 AD   02    0E    52  F0  01FC    847          INSV    R2,#MASK_ALIGN,#2,SAVE_MASK(FP) ; store it into the frame
              50    52  C0  0202    848          ADDL2   R2,R0                      ; compute the parameter area location
  FC A0   51    FE 8F   78  0205    849          ASHL    #-2,R1,-4(R0)              ; store the parameter count
```

```
                    04  020B   850          RET                                 ; return (to next instruction)
                        020C   851
                        020C   852 ;+
                        020C   853 ; Perform instruction lookahead for speed optimization.
                        020C   854 ; At this point, 0(SP) contains the PC of the next user instruction, 4(SP)
                        020C   855 ; has the user PSL.
                        020C   856 ;-
                        020C   857
          51 04 AE  04  E0  020C   858 2$:      BBS     #PSL_T, 4(SP), 90$          ; If T-bit set, don't do lookahead
                7E  50  7D  0211   859          MOVQ    R0, =(SP)                   ; Save R0-R1 temporarily
             51   08 AE  D0  0214   860          MOVL    8(SP), R1                   ; Get PC in R1
     50   0C AE    02  18  EF  0218   861          EXTZV   #PSL_CAM, #2, 12(SP), R0; Get current access mode in R0
             61    02  50  0C  021E   862          PROBER  R0, #2, (R1)                ; Can we read the next TWO bytes?
                       3B  13  0222   863          BEQL    80$                         ; If not, just return
                   50  81  9A  0224   864          MOVZBL  (R1)+, R0                   ; Get next opcode byte in R0
                FD 8F  50  91  0227   865          CMPB    R0, #^XFD                   ; 2-byte opcode?
                       1A  13  022B   866          BEQL    20$                         ; Skip if yes
                40 8F  50  91  022D   867          CMPB    R0, #^X40                   ; Compare against ADDF2
                       2C  1F  0231   868          BLSSU   80$                         ; Not emulated if LSSU
                76 8F  50  91  0233   869          CMPB    R0, #^X76                   ; Compare against CVTDF
                       26  1A  0237   870          BGTRU   80$                         ; Not emulated if GTRU
                   50  04  C4  0239   871          MULL2   #4, R0                      ; Get bit offset of mask
 51    00000277'EF    04  50  EF  023C   872          EXTZV   R0, #4, INST_TYPES_1BYTE, R1 ; Get inst-types mask in R1
                       3D  11  0245   873          BRB     70$                         ; Join common code
                   50  61  98  0247   874 20$:     CVTBL   (R1), R0                    ; Get next opcode byte in R0
                       17  19  024A   875          BLSS    30$                         ; Skip if greater than PUSHAO (^X7F)
                   32  50  91  024C   876          CMPB    R0, #^X32                   ; Compare against CVTDH
                       0E  1F  024F   877          BLSSU   80$                         ; Not emulated if LSSU
                   50  04  C4  0251   878          MULL2   #4, R0                      ; Get bit offset of mask
 51    0000029A'EF    04  50  EF  0254   879          EXTZV   R0, #4, INST_TYPES_2BYTE, R1 ; Get inst-types mask in R1
                       25  11  025D   880          BRB     70$                         ; Join common code
                   50  8E  7D  025F   881 80$:     MOVQ    (SP)+, R0                   ; Restore saved R0
                       02  02  0262   882 90$:     REI                                 ; Return to the next user instruction
                   51  0A  9A  0263   883 30$:     MOVZBL  #<IT_F+IT_H>, R1            ; Assume CVTFH or CVTHF
                99 8F  50  91  0266   884          CMPB    R0,#^X99                    ; Compare against CVTFG
                       0D  1A  026A   885          BGTRU   40$                         ; Not CVTFG or CVTFH if GTRU
                98 8F  50  91  026C   886          CMPB    R0,#^X98                    ; Compare against CVTFH
                       ED  1F  0270   887          BLSSU   80$                         ; Not emulated if LSSU
                       10  13  0272   888          BEQL    70$                         ; R1 mask is correct for CVTFH
                   51  0C  8C  0274   889          XORB2   #<IT_G+IT_H>, R1            ; CVTFG - Switch to using F and G
                       0B  11  0277   890          BRB     70$                         ; Join common code
                F6 8F  50  91  0279   891 40$:     CMPB    R0, #^XF6                   ; Compare against CVTHF
                       E0  1F  027D   892          BLSSU   80$                         ; Not emulated if LSSU
                       03  13  027F   893          BEQL    70$                         ; Join common code if CVTHF
                   51  03  8C  0281   894          XORB2   #<IT_F+IT_D>, R1            ; CVTHD - Switch to using H and D
          51   51  08  78  0284   895 70$:     ASHL    #ARC$V_DFLT_EMUL, R1, R1; Align mask with EXE$GL_ARCHFLAG
00000000'GF   51  D3  0288   896          BITL    R1, G^EXE$GL_ARCHFLAG       ; Should we emulate this instruction?
                       CE  13  028F   897          BEQL    80$                         ; If all bits test zero, no
                   50  8E  7D  0291   898          MOVQ    (SP)+, R0                   ; Pop saved R0 and R1
                    FD69   31  0294   899          BRW     VAX$$EMULATE_FP             ; Emulate the next instruction
                           0297   900          ASSUME  ARC$V_FFLT_EMUL EQ ARC$V_DFLT_EMUL+1
                           0297   901          ASSUME  ARC$V_GFLT_EMUL EQ ARC$V_FFLT_EMUL+1
                           0297   902          ASSUME  ARC$V_HFLT_EMUL EQ ARC$V_GFLT_EMUL+1
                           0297   903          ;
```

```
                                J 15
0297   905          .SBTTL  INSTRUCTION_TYPES
0297   906
0297   907 ;+
0297   908 ; The following two tables provide information on what floating types
0297   909 ; are manipulated by each instruction.  For each opcode there are
0297   910 ; four bits which, if set, indicate that the instruction uses D, F, G
0297   911 ; and H_floating data, respectively.  These four bits are compared against
0297   912 ; the corresponding four bits in EXE$GL_ARCHFLAG to see if the next
0297   913 ; instruction should be emulated; this test takes place in NORMAL_EXIT.
0297   914 ;-
0297   915
0297   916
0297   917          ;+
0297   918          ; Macro INST_TYPE generates table entries
0297   919          ;-
0297   920
0297   921          .MACRO  INST_TYPE       OPCODE1,OPCODE2
0297   922          $$x=0
0297   923          .IRPC   $$t,<OPCODE2>
0297   924          $$x=$$x+IT_'$$t'
0297   925          .ENDR
0297   926          $$x=$$x@4
0297   927          .IRPC   $$t,<OPCODE1>
0297   928          $$x=$$x+IT_'$$t'
0297   929          .ENDR
0297   930          .BYTE   $$x
0297   931          .ENDM
0297   932
0297   933 ;+
0297   934 ; INST_TYPES_1BYTE - Masks for 1-byte instructions.
0297   935 ; Covers opcodes 40 (ADDF2) through 76 (CVTDF)
0297   936 ;-
0297   937
00000277   0297   938 INST_TYPES_1BYTE=.-<^X40/2>     ; Offset for first opcode
0297   939          INST_TYPE       F,F             ; ADDF2, ADDF3
0298   940          INST_TYPE       F,F             ; SUBF2, SUBF3
0299   941          INST_TYPE       F,F             ; MULF2, MULF3
029A   942          INST_TYPE       F,F             ; DIVF2, DIVF3
029B   943          INST_TYPE       F,F             ; CVTFB, CVTFW
029C   944          INST_TYPE       F,F             ; CVTFL, CVTRFL
029D   945          INST_TYPE       F,F             ; CVTBF, CVTWF
029E   946          INST_TYPE       F,F             ; CVTLF, ACBF
029F   947          INST_TYPE       F,F             ; MOVF, CMPF
02A0   948          INST_TYPE       F,F             ; MNEGF, TSTF
02A1   949          INST_TYPE       F,F             ; EMODF, POLYF
02A2   950          INST_TYPE       FD,X            ; CVTFD, <57>
02A3   951          INST_TYPE       X,X             ; ADAWI, <59>
02A4   952          INST_TYPE       X,X             ; <5A>, <5B>
02A5   953          INST_TYPE       X,X             ; INSQHI, INSQTI
02A6   954          INST_TYPE       X,X             ; REMQHI, REMQTI
02A7   955          INST_TYPE       D,D             ; ADDD2, ADDD3
02A8   956          INST_TYPE       D,D             ; SUBD2, SUBD3
02A9   957          INST_TYPE       D,D             ; MULD2, MULD3
02AA   958          INST_TYPE       D,D             ; DIVD2, DIVD3
02AB   959          INST_TYPE       D,D             ; CVTDB, CVTDW
02AC   960          INST_TYPE       D,D             ; CVTDL, CVTRDL
02AD   961          INST_TYPE       D,D             ; CVTBD, CVTWD
```

```
                02AE   962          INST_TYPE       D,D             ; CVTLD, ACBD
                02AF   963          INST_TYPE       D,D             ; MOVD, CMPD
                02B0   964          INST_TYPE       D,D             ; MNEGD, TSTD
                02B1   965          INST_TYPE       D,D             ; EMODD, POLYD
                02B2   966          INST_TYPE       FD,X            ; CVTDF, <77>
                02B3   967
                02B3   968 ;+
                02B3   969 ; INST_TYPES_2BYTE - Masks for 2-byte instructions.
                02B3   970 ; Covers opcodes 32FD (CVTDH) through 7FFD (PUSHAO)
                02B3   971 ;-
                02B3   972
0000029A        02B3   973 INST_TYPES_2BYTE=.-<^X32/2>     ; Offset for first opcode
                02B3   974          INST_TYPE       DH,GF           ; CVTDH, CVTGF
                02B4   975          INST_TYPE       X,X             ; <34FD>, <35FD>
                02B5   976          INST_TYPE       X,X             ; <36FD>, <37FD>
                02B6   977          INST_TYPE       X,X             ; <38FD>, <39FD>
                02B7   978          INST_TYPE       X,X             ; <3AFD>, <3BFD>
                02B8   979          INST_TYPE       X,X             ; <3CFD>, <3DFD>
                02B9   980          INST_TYPE       X,X             ; <3EFD>, <3FFD>
                02BA   981          INST_TYPE       G,G             ; ADDG2, ADDG3
                02BB   982          INST_TYPE       G,G             ; SUBG2, SUBG3
                02BC   983          INST_TYPE       G,G             ; MULG2, MULG3
                02BD   984          INST_TYPE       G,G             ; DIVG2, DIVG3
                02BE   985          INST_TYPE       G,G             ; CVTGB, CVTGW
                02BF   986          INST_TYPE       G,G             ; CVTGL, CVTRGL
                02C0   987          INST_TYPE       G,G             ; CVTBG, CVTWG
                02C1   988          INST_TYPE       G,G             ; CVTLG, ACBG
                02C2   989          INST_TYPE       G,G             ; MOVG, CMPG
                02C3   990          INST_TYPE       G,G             ; MNEGG, TSTG
                02C4   991          INST_TYPE       G,G             ; EMODG, POLYG
                02C5   992          INST_TYPE       GH,X            ; CVTGH, <57FD>
                02C6   993          INST_TYPE       X,X             ; <58FD>, <59FD>
                02C7   994          INST_TYPE       X,X             ; <5AFD>, <5BFD>
                02C8   995          INST_TYPE       X,X             ; <5CFD>, <5DFD>
                02C9   996          INST_TYPE       X,X             ; <5EFD>, <5FFD>
                02CA   997          INST_TYPE       H,H             ; ADDH2, ADDH3
                02CB   998          INST_TYPE       H,H             ; SUBH2, SUBH3
                02CC   999          INST_TYPE       H,H             ; MULH2, MULH3
                02CD  1000          INST_TYPE       H,H             ; DIVH2, DIVH3
                02CE  1001          INST_TYPE       H,H             ; CVTHB, CVTHW
                02CF  1002          INST_TYPE       H,H             ; CVTHL, CVTRHL
                02D0  1003          INST_TYPE       H,H             ; CVTBH, CVTWH
                02D1  1004          INST_TYPE       H,H             ; CVTLH, ACBH
                02D2  1005          INST_TYPE       H,H             ; MOVH, CMPH
                02D3  1006          INST_TYPE       H,H             ; MNEGH, TSTH
                02D4  1007          INST_TYPE       H,H             ; EMODH, POLYH
                02D5  1008          INST_TYPE       HG,X            ; CVTHG, <77FD>
                02D6  1009          INST_TYPE       X,X             ; <78FD>, <79FD>
                02D7  1010          INST_TYPE       X,X             ; <7AFD>, <7BFD>
                02D8  1011          INST_TYPE       H,H             ; CLRO, MOVO
                02D9  1012          INST_TYPE       H,H             ; MOVAO, PUSHAO
```

```
                                02DA  1014          .SBTTL  TEST_FRAME
                                02DA  1015
                                02DA  1016          ;
                                02DA  1017          ;       TEST_FRAME - Test Frame Location and Move If Necessary
                                02DA  1018          ;
                                02DA  1019          ;               entered by subroutine branching
                                02DA  1020          ;
                                02DA  1021          ;               parameter:      R0 = Size of Information to be Pushed
                                02DA  1022          ;
                                02DA  1023          ;               returns with    R0 = Distance Frame was Moved
                                02DA  1024          ;
                                02DA  1025          ;       Discussion
                                02DA  1026          ;
                                02DA  1027          ;           This routine determines whether or not the address given
                                02DA  1028          ;       by subtracting R0 from the user's stack pointer can be made
                                02DA  1029          ;       the location following a parameter list without the location
                                02DA  1030          ;       being within the local storage. If this cannot be done then
                                02DA  1031          ;       the entire procedure frame is moved so the condition can be
                                02DA  1032          ;       satisfied. The distance that the procedure frame was moved
                                02DA  1033          ;       is returned in R0. The value is zero if the frame is not
                                02DA  1034          ;       moved.
                                02DA  1035          ;
                                02DA  1036          ;       Note:   1. The switch from one frame location to another is
                                02DA  1037          ;                  performed by a single indivisible POPR instruction
                                02DA  1038          ;                  so the Emulator is never in an anomalous state.
                                02DA  1039          ;
                                02DA  1040          ;               2. If the frame is moved to a higher address, then
                                02DA  1041          ;                  the saved AP and FP are changed to the values of
                                02DA  1042          ;                  the emulated registers. The reason for this is
                                02DA  1043          ;                  that the move may overlay a valid frame so it is
                                02DA  1044          ;                  assumed that the user's AP and FP have been changed
                                02DA  1045          ;                  by the instruction to information about a new valid
                                02DA  1046          ;                  frame. The implementor thinks that all of this is
                                02DA  1047          ;                  pretty strange but if the program will work with
                                02DA  1048          ;                  the hardware it will still work with the Emulator.
                                02DA  1049          ;
                                02DA  1050  TEST_FRAME:                                     ; entrance
                          00    DD    02DA  1051            PUSHL   #0                      ; push a zero
50    4C AD     50      C3    02DC  1052            SUBL3   R0,REG_SP(FP),R0                ; compute pushed information address
          50    03      CA    02E1  1053            BICL2   #3,R0                           ; align the value
     51     58 AD      9E    02E4  1054            MOVAB   LOCAL_END(FP),R1                ; R1 = end of local storage
          51    50      D1    02E8  1055            CMPL    R0,R1                           ; does push extend below the frame ?
                14      1E    02EB  1056            BGEQU   2$                              ; no - bypass
   53     5E    03      CB    02ED  1057            BICL3   #3,SP,R3                        ; R3 = aligned stack pointer
          53    18      C2    02F1  1058            SUBL2   #24,R3                          ; adjust for additional pushes
          52    50      D0    02F4  1059            MOVL    R0,R2                           ; R2 = address following moved frame
          53    52      D1    02F7  1060            CMPL    R2,R3                           ; does it extend into the frame ?
                18      1B    02FA  1061            BLEQU   3$                              ; no - bypass
          52    53      D0    02FC  1062            MOVL    R3,R2                           ; yes - use address below the frame
                13      11    02FF  1063            BRB     3$                              ; bypass
   52     0447  CD      9E    0301  1064  2$:       MOVAB   FRAME_END+1027(FP),R2           ; R2 = last possible parameter end
          52    50      D1    0306  1065            CMPL    R0,R2                           ; does the push end above it ?
                49      1B    0309  1066            BLEQU   5$                              ; no - bypass
 08 AD     44 AD      7D    030B  1067            MOVQ    REG_AP(FP),SAVE_AP(FP)          ; change the saved AP and FP
   52    50      03      CB    0310  1068            BICL3   #3,R0,R2                        ; R2 = aligned user stack pointer
          52    51      C2    0314  1069  3$:       SUBL2   R1,R2                           ; R2 = distance of the move
                52      DD    0317  1070            PUSHL   R2                              ; push the quantity
```

```
                    6E42   9F   0319   1071          PUSHAB  (SP)[R2]                    ; push the modified SP
                    6D42   9F   031C   1072          PUSHAB  (FP)[R2]                    ; push the modified FP
                    6C42   9F   031F   1073          PUSHAB  (AP)[R2]                    ; push the modified AP
                FF  AD42   9F   0322   1074          PUSHAB  SAVE_ALIGN(FP)[R2]          ; push the new alignment bits location
                FE  AD42   9F   0326   1075          PUSHAB  SAVE_PARCNT(FP)[R2]         ; push the new parameter count address
            53  48  AD42   9E   032A   1076          MOVAB   FRAME_END+4(FP)[R2],R3      ; R3 = new frame end + 4 location
            53  50    53   C3   032F   1077          SUBL3   R3,R0,R3                    ; R3 = distance to user's SP
        7E  53    FE   8F   78   0333   1078          ASHL    #-2,R3,-(SP)                ; push the new parameter count
            50    51   5E   C3   0338   1079          SUBL3   SP,R1,R0                    ; R0 = number of bytes to move
                  51   5E   D0   033C   1080          MOVL    SP,R1                       ; R1 = location of bytes to move
                       52   D5   033F   1081          TSTL    R2                          ; will we extend the stack ?
                       03   18   0341   1082          BGEQ    4$                          ; no - skip
                  5E   52   C0   0343   1083          ADDL2   R2,SP                       ; yes - extend the stack pointer
                6142  61   50   28   0346   1084  4$:     MOVC3   R0,(R1),(R1)[R2]        ; move the frame
                  9E   8E   F6   034B   1085          CVTLB   (SP)+,@(SP)+                ; store the new parameter count
                       9E   94   034E   1086          CLRB    @(SP)+                      ; clear the new alignment bits
                  7000 8F   BA   0350   1087          POPR    #^M<AP,FP,SP>               ; switch to the new frame
                       01   BA   0354   1088  5$:     POPR    #^M<R0>                     ; R0 = distance frame was moved
                            05   0356   1089          RSB                                 ; return
                                 0357   1090          ;
```

```
                                0357  1092          .SBTTL  COND_HANDLER
                                0357  1093
                                0357  1094          ;
                                0357  1095          ;       COND_HANDLER - Internal Condition Handler
                                0357  1096          ;
                                0357  1097          ;               parameters:     P1 = Signal Array Location
                                0357  1098          ;                               P2 = Mechanism Array Location
                                0357  1099          ;
                                0357  1100          ;               returns with    R0 = Condition Response
                                0357  1101          ;
                                0357  1102          ;       Discussion
                                0357  1103          ;
                                0357  1104          ;           This routine is the internal condition handler for the
                                0357  1105          ;       Emulator. Since the Emulator does not make constructive use
                                0357  1106          ;       of exceptions in its main procedure, this routine requests
                                0357  1107          ;       resignaling of all conditions it intercepts.
                                0357  1108          ;
                                0357  1109          ;           If the condition is SS$_UNWIND which indicates that an
                                0357  1110          ;       unwind is about to take place, then it restores the argument
                                0357  1111          ;       count longword in the parameter list for the procedure so the
                                0357  1112          ;       unwind will work properly.
                                0357  1113          ;
                                0357  1114  COND_HANDLER:                                ; entrance
                           0000 0357  1115          .WORD   0                           ; entry mask
                     50  04 AC  7D   0359  1116          MOVQ    4(AP),R0                ; R0,R1 = condition array locations
                                 035D  1117          CMPCOND SS$_UNWIND,4(R0)            ; is this an unwind ?
                        19  12   0367  1118          BNEQ    1$                          ; no - bypass
                     50  04 A1  D0   0369  1119          MOVL    4(R1),R0                ; R0 = frame location
                     51  FF A0  90   036D  1120          MOVB    SAVE_ALIGN(R0),R1       ; R1 = safe copy of alignment bits
                     7E  FE AD  9A   0371  1121          MOVZBL  SAVE_PARCNT(FP),-(SP)   ; push the argument count
           06 A0  02    0E     51  F0   0375  1122          INSV    R1,#MASK_ALIGN,#2,SAVE_MASK(R0) ; store align bits in frame
                     50     51  C0   037B  1123          ADDL2   R1,R0                   ; add to the frame location
                  44 A0     8E  D0   037E  1124          MOVL    (SP)+,FRAME_END(R0)     ; store the argument count
                50   0918   8F  3C   0382  1125  1$:     MOVZWL  #SS$_RESIGNAL,R0        ; resignal the condition
                                04   0387  1126          RET                             ; return
                                     0388  1127          ;
```

```
0388  1129          .SBTTL  Instruction Emulation Routines
0388  1130
0388  1131          ;       ************************************************************************
0388  1132          ;       *                                                                      *
0388  1133          ;       *                                                                      *
0388  1134          ;       *                  Instruction Emulation Routines                      *
0388  1135          ;       *                                                                      *
0388  1136          ;       *                                                                      *
0388  1137          ;       ************************************************************************
0388  1138          ;
0388  1139          ;
0388  1140          ;       Introduction
0388  1141          ;       ------------
0388  1142          ;
0388  1143          ;           Following are the routines for emulating the individual
0388  1144          ;       new instructions. There is one routine for each instruction
0388  1145          ;       but the structure of most of these routines is extremely
0388  1146          ;       simple so we have included all of the descriptive information
0388  1147          ;       here rather than duplicating it for each routine. Special
0388  1148          ;       discussions are given for those instructions or families of
0388  1149          ;       instructions which do not quite fit into the general patterns.
0388  1150          ;
0388  1151          ;           The routines themselves have been written so that they
0388  1152          ;       will be easy to follow. Because of this the temptation to
0388  1153          ;       remove a considerable amount of duplicated code has been
0388  1154          ;       staunchly resisted.
0388  1155          ;
0388  1156          ;           The emulation routines have names which are of the form
0388  1157          ;       "INST_mnemonic", are entered by branching from the routine
0388  1158          ;       EMULATOR, and when instruction is complete the routines branch
0388  1159          ;       to NORMAL_EXIT. When exceptions occur, then flow of control is
0388  1160          ;       somewhat more complicated and is described below.
0388  1161          ;
0388  1162          ;
0388  1163          ;       Operand Processing
0388  1164          ;       ------------------
0388  1165          ;
0388  1166          ;           The first step for each of the instructions is to process
0388  1167          ;       the instruction operands (this process is inhibited only for
0388  1168          ;       POLYx when FPD is set in the PSL). This is done by
0388  1169          ;       calling the operand scanning routines for each operand and by
0388  1170          ;       saving the operands values (for read and modify access
0388  1171          ;       operands) and operand addresses (for write, modify, and
0388  1172          ;       address, access operands) in the special areas of the frame
0388  1173          ;       allocated for that purpose. Floating values of all types that
0388  1174          ;       are input are converted to our internal floating format by the
0388  1175          ;       UNPACK routines which are described below.
0388  1176          ;
0388  1177          ;
0388  1178          ;       Instruction Emulation
0388  1179          ;       ---------------------
0388  1180          ;
0388  1181          ;           After all of the operands have been processed, the action
0388  1182          ;       of the instruction is performed usually by calling one of the
0388  1183          ;       arithmetic routines. For many of the move and convert
0388  1184          ;       instructions no special emulation action is necessary since
0388  1185          ;       everything is done by the PACK and UNPACK routines. All of the
```

```
0388  1186          ;       floating arithmetic is performed using our internal floating
0388  1187          ;       representation.
0388  1188          ;
0388  1189          ;
0388  1190          ;       Storing the Results
0388  1191          ;       -------------------
0388  1192          ;
0388  1193          ;           The last step of instruction emulation is storing the
0388  1194          ;       resulting values and updating the condition codes. The values
0388  1195          ;       are stored in the order in which their corresponding operands
0388  1196          ;       occur so the instruction is emulated properly when the write
0388  1197          ;       or modify access operands overlap. Floating values are
0388  1198          ;       converted from the internal floating representation to the
0388  1199          ;       target representation by one of the PACK routines described
0388  1200          ;       below. After the results are output, the condition codes in
0388  1201          ;       the emulated PSL are updated. This order of operation is
0388  1202          ;       important since the PACK routines may convert very small
0388  1203          ;       nonzero values to zero values if an underflow occurs and FU is
0388  1204          ;       clear in the PSL. When the instruction emulation is complete
0388  1205          ;       the routine branches to NORMAL_EXIT.
0388  1206          ;
0388  1207          ;
0388  1208          ;       Exceptions
0388  1209          ;       ----------
0388  1210          ;
0388  1211          ;           Except for the integer overflow trap, all of the
0388  1212          ;       exceptions that the Emulator checks for are faults. When
0388  1213          ;       faults occur the Emulator restores everything to a state in
0388  1214          ;       which the instruction can be restarted and signals the fault.
0388  1215          ;       For the integer overflow trap, the instruction is run to
0388  1216          ;       completion, possibly outputing a truncated result, and the
0388  1217          ;       trap is signaled. Here we describe how instruction emulation
0388  1218          ;       is organized so this can be accomplished. Further information
0388  1219          ;       can be found in the essay on exception processing which
0388  1220          ;       appears below.
0388  1221          ;
0388  1222          ;
0388  1223          ;       Faults
0388  1224          ;
0388  1225          ;           Until the results of an instruction are output, all of the
0388  1226          ;       changes which take place to any register are additions and
0388  1227          ;       subtractions of small integers. These changes are also
0388  1228          ;       recorded in the change bytes corresponding to the registers.
0388  1229          ;       Therefore the only requirement necessary for correct fault
0388  1230          ;       processing is to insure that all fault detection takes place
0388  1231          ;       before any of the instruction results are output (actually,
0388  1232          ;       things are a little more complicated for POLYx).
0388  1233          ;       This is done by making the result outputing steps the last
0388  1234          ;       part of instruction emulation. In this way fault detection
0388  1235          ;       can be placed anywhere in the Emulator it is convenient rather
0388  1236          ;       than requiring close synchronization with the instruction
0388  1237          ;       emulation logic. For this reason direct processing of faults
0388  1238          ;       (rather than, say, returning status codes) appears throughout
0388  1239          ;       the common Emulator subroutines.
0388  1240          ;
0388  1241          ;           The architecture makes no special requirements on the
0388  1242          ;       the condition codes generated by a fault except that they
```

```
0388  1243          ;       be sufficient for correctly restarting the instruction. In
0388  1244          ;       general, the V bit is cleared by EMULATOR on entry to an
0388  1245          ;       instruction emulation routine since the bit is used for
0388  1246          ;       detecting integer overflow traps (see below). None of the
0388  1247          ;       other condition bits is altered until the results are output
0388  1248          ;       at the end of instruction emulation. Consequently all of the
0388  1249          ;       other condition bits are preserved into faults. (With the
0388  1250          ;       present set of emulated instructions the only preservation
0388  1251          ;       requirement is that the C bit be preserved on faults by the
0388  1252          ;       ACBx instructions.)
0388  1253          ;
0388  1254          ;
0388  1255          ;       Integer Overflow Traps
0388  1256          ;
0388  1257          ;           When a integer overflow is detected (either in a common
0388  1258          ;       subroutine or in the instruction emulation routine) the V
0388  1259          ;       bit is set in the emulated condition codes and the value is
0388  1260          ;       truncated. Everything proceeds normally until control reaches
0388  1261          ;       NORMAL_EXIT where the V and IV bits in the emulated PSL are
0388  1262          ;       checked. If both are set then an integer overflow trap is
0388  1263          ;       signaled.
0388  1264          ;
0388  1265          ;
0388  1266          ;       Access Modes and Access Violations
0388  1267          ;
0388  1268          ;           This version of the Emulator has been designed to operate
0388  1269          ;       at the same access mode as the instruction being emulated.
0388  1270          ;       However most of the instruction emulation logic has been
0388  1271          ;       designed to operate at any access mode that is not less
0388  1272          ;       privleged than that of the instruction (exceptions: the logic
0388  1273          ;       for getting in, the logic for getting out, and the exception
0388  1274          ;       signaling logic). All attempts to access memory on behalf of
0388  1275          ;       the instruction are probed using the access mode contained in
0388  1276          ;       the local cell MODE. This is normally set to the current
0388  1277          ;       access mode but may be set to some less privleged access mode.
0388  1278          ;       Because this version of the Emulator operates on the user
0388  1279          ;       stack it is not necessary to probe stack operations, however,
0388  1280          ;       these probes are still done to insure that the code will
0388  1281          ;       remain usable for different access modes.
0388  1282          ;
0388  1283          ;
0388  1284          ;       Notes on the ACBx Instructions
0388  1285          ;       ------------------------------
0388  1286          ;
0388  1287          ;           For these instructions the operands are processed, the
0388  1288          ;       step is added to the index, and the sum is packed and output.
0388  1289          ;       Afterwards the sum is unpacked again and compared with the
0388  1290          ;       limit, the type of comparison depending on the sign of the
0388  1291          ;       step. The packing and unpacking is necessary to truncate the
0388  1292          ;       sum to the correct number of bits, to perform any rounding,
0388  1293          ;       and to check for underflow or overflow exceptions. If the
0388  1294          ;       branch is to be taken, the branch destination is moved into
0388  1295          ;       the emulated PC.
0388  1296          ;
0388  1297          ;
0388  1298          ;       Notes on the EMODx Instructions
0388  1299          ;       -------------------------------
```

```
0388  1300          ;
0388  1301          ;       For these instructions the first two operands are a
0388  1302          ;   floating value of the appropriate type and a word containing
0388  1303          ;   some extension bits. When the extension bits are picked up
0388  1304          ;   they are stored in the correct position of the unpacked first
0388  1305          ;   operand. The remaining operands are scanned and flag bits 1
0388  1306          ;   and 7 are used to remember which output operands are in the
0388  1307          ;   registers. The multiply is then performed using the proper
0388  1308          ;   arithmetic routine and the product is truncated to the correct
0388  1309          ;   number of bits. Next the integer part is extracted and saved
0388  1310          ;   in an internal cell while the fraction part is extracted and
0388  1311          ;   packed. Next any register output operands are output and
0388  1312          ;   the routine TEST_FRAME is called to insure that the user's
0388  1313          ;   stack pointer is within the parameter area. Next any output
0388  1314          ;   operands in memory are output with stores into the Emulator's
0388  1315          ;   working storage disabled .If both output operands are in the
0388  1316          ;   same type of storage, then the integer part is output before
0388  1317          ;   the fraction part. This complicated method of outputing the
0388  1318          ;   results is necessary because one of the register operands may
0388  1319          ;   contain SP and extend the user's stack into the Emulator's
0388  1320          ;   working storage. The call to TEST_FRAME will move the frame
0388  1321          ;   if this occurs so if the other operand will be stored into
0388  1322          ;   memory properly if it is not below the user's stack pointer.
0388  1323          ;
0388  1324          ;
0388  1325          ;   Notes on the POLYx Instructions
0388  1326          ;   -------------------------------
0388  1327          ;
0388  1328          ;       These instructions have especially complicated emulation
0388  1329          ;   routines because they do a lot, have a large number of
0388  1330          ;   implicit inputs and outputs, and are intended to be resumed
0388  1331          ;   instead of restarted when faults occur. For this reason we
0388  1332          ;   describe the action of the instruction emulation routines in
0388  1333          ;   some detail.
0388  1334          ;
0388  1335          ;       These instructions can be thought as having two states.
0388  1336          ;   In the first of these states no unreversable changes have
0388  1337          ;   been made to the registers so if a fault occurs the
0388  1338          ;   instruction can be restarted. The instruction remains in this
0388  1339          ;   state until all of the operands have been processed and the
0388  1340          ;   first coefficient has been validated. After this point the
0388  1341          ;   instruction saves various things in the registers R0-R5 (and
0388  1342          ;   on the stack for POLYH) so the instruction can no longer be
0388  1343          ;   restarted but the information saved is sufficient for resuming
0388  1344          ;   the instruction after a fault. When this second state is
0388  1345          ;   reached the FPD bit is set in the PSL so the when the
0388  1346          ;   instruction emulation routine is entered for resuming the
0388  1347          ;   instruction it will know enough to do so.
0388  1348          ;
0388  1349          ;       Following is an outline of the steps in the emulation of
0388  1350          ;   the two instructions. The text generally describes POLYG and
0388  1351          ;   the differences for POLYH are noted in parentheses.
0388  1352          ;
0388  1353          ;
0388  1354          ;           1. This is the first step for the instruction emulation.
0388  1355          ;              If the FPD bit is clear in the PSL then control passes
0388  1356          ;              to Step 2. Otherwise the following actions which are
```

```
0388  1357          ;                concerned with restarting the instruction take place:
0388  1358          ;
0388  1359          ;                The length of the instruction operands is retrieved
0388  1360          ;                from the high order three bytes of the user's R2 (R4
0388  1361          ;                for POLYH) and added to the current PC and to the
0388  1362          ;                change byte for the PC. If the instruction terminates
0388  1363          ;                the PC will be positioned following the instruction
0388  1364          ;                and if the instruction faults it will be reset to the
0388  1365          ;                location of the instruction.
0388  1366          ;
0388  1367          ;                The result so far is unpacked from R0-R1 (R0-R3 for
0388  1368          ;                POLYH) and the unpacked value is stored in OPERAND2.
0388  1369          ;                The condition codes are set to describe this value.
0388  1370          ;
0388  1371          ;                The argument is unpacked from R4-R5 (from a stacked
0388  1372          ;                octaword for POLYH, which is probed) and the unpacked
0388  1373          ;                value is stored in OPERAND3.
0388  1374          ;
0388  1375          ;                The remaining coefficient count in the low order byte
0388  1376          ;                of R2 (R4 for POLYH) is checked for validity.
0388  1377          ;
0388  1378          ;                Finally control passes to Step 3.
0388  1379          ;
0388  1380          ;
0388  1381          ;            2. This step performs all of the processing for initial
0388  1382          ;                entry to the instruction.
0388  1383          ;
0388  1384          ;                The first operand is processed and the unpacked
0388  1385          ;                argument is saved in OPERAND3.
0388  1386          ;
0388  1387          ;                The second operand is processed and the degree is
0388  1388          ;                checked for validity and the value is saved
0388  1389          ;                temporarily in ADDRESS1.
0388  1390          ;
0388  1391          ;                The third operand is processed and the address of the
0388  1392          ;                coefficient table is saved. The first coefficient is
0388  1393          ;                probed, unpacked, and saved in OPERAND2.
0388  1394          ;
0388  1395          ;                If the instruction is POLYH then the user's SP is
0388  1396          ;                decremented by 16 (and so is the change byte) and
0388  1397          ;                the argument value in OPERAND3 is packed and output
0388  1398          ;                to the allocated stack area. (If a fault occurs before
0388  1399          ;                FPD is set then SP will be reinitilized properly.)
0388  1400          ;
0388  1401          ;                The coefficient in OPERAND2 is packed and stored in
0388  1402          ;                the user's registers R0-R1 (R0-R4 for POLYH), the
0388  1403          ;                address of the remaining coefficients is stored in the
0388  1404          ;                user's R4 (R6 for POLYH), the degree is stored in the
0388  1405          ;                low order byte of the user's R2 (R4 for POLYH), and
0388  1406          ;                the length of the instruction operands (the PC change
0388  1407          ;                byte minus two) is saved in the upper three bytes of
0388  1408          ;                that register. The condition codes are set based on
0388  1409          ;                the value in OPERAND2.
0388  1410          ;
0388  1411          ;                All of the change bytes except that for PC are cleared
0388  1412          ;                and the bit FPD is set in the emulated PSL.
0388  1413          ;
```

```
0388  1414          ;                   If the instruction is POLYG then the argument in
0388  1415          ;                   OPERAND3 is packed and saved in the user's R4-R5.
0388  1416          ;
0388  1417          ;                   Control Passes to Step 3.
0388  1418          ;
0388  1419          ;
0388  1420          ;               3.  This step performs the basic polynomial evaluation
0388  1421          ;                   iteration.
0388  1422          ;
0388  1423          ;                   If the remaining coefficient count in the low order
0388  1424          ;                   byte of the user's R2 (R4 for POLYH) is zero then
0388  1425          ;                   control passes to Step 4.
0388  1426          ;
0388  1427          ;                   The value so far in OPERAND2 and the argument in
0388  1428          ;                   OPERAND3 are multiplied and the unnormalized product
0388  1429          ;                   is truncated to 63 (127 for POLYH) bits and stored
0388  1430          ;                   in OPERAND1. The truncation is performed by clearing
0388  1431          ;                   the low order bit of the 64 (128 for POLYH) bit
0388  1432          ;                   product returned by the multiply routine.
0388  1433          ;
0388  1434          ;                   The coefficient addressed by the user's R3 (R5 for
0388  1435          ;                   POLYH) is probed and unpacked to OPERAND2. The result
0388  1436          ;                   is then added to OPERAND1.
0388  1437          ;
0388  1438          ;                   The new value so far in OPERAND1 is packed and stored
0388  1439          ;                   in the user's R0-R1 (R0-R3 for POLYH). The condition
0388  1440          ;                   codes are set based on this value. The value is then
0388  1441          ;                   unpacked and stored in OPERAND2.
0388  1442          ;
0388  1443          ;                   The remaining coefficient count in the low order byte
0388  1444          ;                   of R2 (R4 for POLYH) is decremented and the pointer to
0388  1445          ;                   the next coeffiecient in R3 (R5 for POLYH) is
0388  1446          ;                   incremented to point to the next coefficient.
0388  1447          ;
0388  1448          ;                   Control then passes to the beginning of this step.
0388  1449          ;
0388  1450          ;
0388  1451          ;               4.  This step performs the termination of the instruction.
0388  1452          ;
0388  1453          ;                   The register R2 (R4 for POLYH) which contains the
0388  1454          ;                   instruction length and the coefficient count is
0388  1455          ;                   cleared.
0388  1456          ;
0388  1457          ;                   If the instruction was POLYG then the registers R4-R5
0388  1458          ;                   which contained the argument value are cleared.
0388  1459          ;
0388  1460          ;                   If the instruction was POLYH then the user's SP is
0388  1461          ;                   incremented by 16 to unstack the argument.
0388  1462          ;
0388  1463          ;                   The FPD bit in the PSL is cleared.
0388  1464          ;
0388  1465          ;                   Instruction emulation is now complete.
0388  1466          ;
0388  1467          ;
0388  1468          ;           Careful reading of the above outline will show that the
0388  1469          ;       POLYG and POLYH instructions are correctly emulated even down
0388  1470          ;       to the handling of faults. Faults may be detected at any of
```

```
0388  1471          ;       the probes and at some of the pack and unpack operations. When
0388  1472          ;       faults are detected control immediatly leaves the outline so
0388  1473          ;       it is important that everything be correct at the time the
0388  1474          ;       check is made. The reader may notice that some results are
0388  1475          ;       packed and then immediatly unpacked. This is done to check for
0388  1476          ;       overflow and underflow and to perform any rounding specified
0388  1477          ;       by the architecture. This technique also converts nonstandard
0388  1478          ;       floating zero representations to standard ones.
0388  1479          ;
```

```
                     0388  1481          ;
                     0388  1482          ;       4F ACBF - Add Compare and Branch F_floating
                     0388  1483          ;
                     0388  1484 INST_ACBF:
                     0388  1485          SET_OP_TYPES    F
                10   11 038C  1486          BRB     INST_ACBx
                     038E  1487
                     038E  1488          ;
                     038E  1489          ;       6F ACBD - Add Compare and Branch D_floating
                     038E  1490          ;
                     038E  1491 INST_ACBD:
                     038E  1492          SET_OP_TYPES    D
                0A   11 0392  1493          BRB     INST_ACBx
                     0394  1494
                     0394  1495          ;
                     0394  1496          ;       4FFD ACBG - Add Compare and Branch G_floating
                     0394  1497          ;
                     0394  1498 INST_ACBG:
                     0394  1499          SET_OP_TYPES    G
                04   11 0398  1500          BRB     INST_ACBx
                     039A  1501
                     039A  1502          ;
                     039A  1503          ;       6FFD ACBH - Add Compare and Branch H_floating
                     039A  1504          ;
                     039A  1505 INST_ACBH:
                     039A  1506          SET_OP_TYPES    H
                     039E  1507          ; BRB   INST_ACBx
                     039E  1508
                     039E  1509 INST_ACBx:
                   0B22   30 039E  1510          BSBW    READ_ACCESS             ; first operand is read only
                   0E8C   30 03A1  1511          BSBW    UNPACK_FLOAT3           ; unpack and save the value
                   0B1C   30 03A4  1512          BSBW    READ_ACCESS             ; second operand is read only
                   0E80   30 03A7  1513          BSBW    UNPACK_FLOAT2           ; unpack and save the value
                   0B28   30 03AA  1514          BSBW    MODIFY_ACCESS           ; third operand is modified
             E7 AD    5B   D0 03AD  1515          MOVL    R11,ADDRESS1(FP)        ; save the operand address
                   0E70   30 03B1  1516          BSBW    UNPACK_FLOAT1           ; unpack and save the value
                   0E35   30 03B4  1517          BSBW    BRANCH_WORD             ; fourth operand is branch destination
             E3 AD    5B   D0 03B7  1518          MOVL    R11,ADDRESS2(FP)        ; save the branch destination
                   1260   30 03BB  1519          BSBW    ADD_REAL                ; add the step and index
                   0F53   30 03BE  1520          BSBW    PACK_FLOAT1             ; pack the index value
          5B    E7 AD    D0 03C1  1521          MOVL    ADDRESS1(FP),R11        ; R11 = destination address
                   10F0   30 03C5  1522          BSBW    STORE_OPERAND           ; store into third operand
                   0E59   30 03C8  1523          BSBW    UNPACK_FLOAT1           ; put the value back
          51    C7 AD    9E 03CB  1524          MOVAB   OPERAND1(FP),R1         ; R1 = location of index value
                   177A   30 03CF  1525          BSBW    TEST_REAL               ; test the value
                   174B   30 03D2  1526          BSBW    SET_CONDITION           ; set the condition codes
          54 AD    02   8A 03D5  1527          BICB2   #PSL$M_V,PSL(FP)        ; clear V in the PSL
          51    AF AD    9E 03D9  1528          MOVAB   OPERAND2(FP),R1         ; R1 = address of step value
                   176C   30 03DD  1529          BSBW    TEST_REAL               ; test the value
                     14   19 03E0  1530          BLSS    2$                      ; it's negative - bypass
          51    C7 AD    9E 03E2  1531          MOVAB   OPERAND1(FP),R1         ; R1 = address of index value
          52    97 AD    9E 03E6  1532          MOVAB   OPERAND3(FP),R2         ; R2 = address of limit value
                   177D   30 03EA  1533          BSBW    COMPARE_REAL            ; have we passed the limit ?
                     14   14 03ED  1534          BGTR    3$                      ; yes - bypass
50 AD    E3 AD    D0 03EF  1535 1$:       MOVL    ADDRESS2(FP),REG_PC(FP) ; perform the branch
                     0D   11 03F4  1536          BRB     3$                      ; bypass
          51    C7 AD    9E 03F6  1537 2$:       MOVAB   OPERAND1(FP),R1         ; R1 = address of index value
```

```
                52  97 AD   9E  03FA  1538          MOVAB   OPERAND3(FP),R2          ; R2 = address of limit value
                  1769   30  03FE  1539          BSBW    COMPARE_REAL             ; have we passed the limit ?
                    EC   18  0401  1540          BGEQ    1$                       ; no - perform the branch
                  FDBE   31  0403  1541 3$:      BRW     NORMAL_EXIT              ; done
                             0406  1542
                             0406  1543          ;
                             0406  1544          ;       40 ADDF2 - Add F_floating (Two Operands)
                             0406  1545          ;
                             0406  1546 INST_ADDF2:                               ; entrance
                             0406  1547          SET_OP_TYPES    F
                       10    11  040A  1548          BRB     INST_ADDx2
                             040C  1549
                             040C  1550          ;
                             040C  1551          ;       60 ADDD2 - Add D_floating (Two Operands)
                             040C  1552          ;
                             040C  1553 INST_ADDD2:                               ; entrance
                             040C  1554          SET_OP_TYPES    D
                       0A    11  0410  1555          BRB     INST_ADDx2
                             0412  1556
                             0412  1557          ;
                             0412  1558          ;       40FD ADDG2 - Add G_floating (Two Operands)
                             0412  1559          ;
                             0412  1560 INST_ADDG2:                               ; entrance
                             0412  1561          SET_OP_TYPES    G
                       04    11  0416  1562          BRB     INST_ADDx2
                             0418  1563
                             0418  1564          ;
                             0418  1565          ;       60FD ADDH2 - Add HFLOAT (Two Operands)
                             0418  1566          ;
                             0418  1567 INST_ADDH2:                               ; entrance
                             0418  1568          SET_OP_TYPES    H
                             041C  1569          ;BRB    INST_ADDx2
                             041C  1570
                             041C  1571 INST_ADDx2:
                     0AA4    30  041C  1572          BSBW    READ_ACCESS              ; first operand is read only
                     0E02    30  041F  1573          BSBW    UNPACK_FLOAT1            ; unpack and save the value
                     0AB0    30  0422  1574          BSBW    MODIFY_ACCESS            ; second operand is modified
               E7 AD   5B    D0  0425  1575          MOVL    R11,ADDRESS1(FP)         ; save the destination location
                     0DFE    30  0429  1576          BSBW    UNPACK_FLOAT2            ; unpack and save the value
                     11EF    30  042C  1577          BSBW    ADD_REAL                 ; compute the sum
                     0EE2    30  042F  1578          BSBW    PACK_FLOAT1              ; pack the sum
               5B    E7 AD    D0  0432  1579          MOVL    ADDRESS1(FP),R11         ; R11 = destination location
                     107F    30  0436  1580          BSBW    STORE_OPERAND            ; store the result
                     1703    30  0439  1581          BSBW    SET_CONDITION1           ; set the condition codes in the PSL
                     FD85    31  043C  1582          BRW     NORMAL_EXIT              ; done
                             043F  1583
                             043F  1584          ;
                             043F  1585          ;       41 ADDF3 - Add F_floating (Three Operands)
                             043F  1586          ;
                             043F  1587 INST_ADDF3:                               ; entrance
                             043F  1588          SET_OP_TYPES    F
                       10    11  0443  1589          BRB     INST_ADDx3
                             0445  1590
                             0445  1591          ;
                             0445  1592          ;       61 ADDD3 - Add D_floating (Three Operands)
                             0445  1593          ;
                             0445  1594 INST_ADDD3:                               ; entrance
```

```
                        0445  1595          SET_OP_TYPES    D
                  0A    11  0449  1596          BRB     INST_ADDx3
                        044B  1597
                        044B  1598          ;
                        044B  1599          ;       41FD ADDG3 - Add G_floating (Three Operands)
                        044B  1600          ;
                        044B  1601  INST_ADDG3:                                 ; entrance
                        044B  1602          SET_OP_TYPES    G
                  04    11  044F  1603          BRB     INST_ADDx3
                        0451  1604
                        0451  1605          ;
                        0451  1606          ;       61FD ADDH3 - Add HFLOAT (Three Operands)
                        0451  1607          ;
                        0451  1608  INST_ADDH3:                                 ; entrance
                        0451  1609          SET_OP_TYPES    H
                        0455  1610          ;BRB    INST_ADDx3
                        0455  1611
                        0455  1612  INST_ADDx3:
              0A6B      30  0455  1613          BSBW    READ_ACCESS             ; first operand is read only
              0DC9      30  0458  1614          BSBW    UNPACK_FLOAT1           ; unpack and save the value
              0A65      30  045B  1615          BSBW    READ_ACCESS             ; second operand is read only
              0DC9      30  045E  1616          BSBW    UNPACK_FLOAT2           ; unpack and save the value
              0A68      30  0461  1617          BSBW    WRITE_ACCESS            ; third operand is write only
        E7 AD   5B      D0  0464  1618          MOVL    R11,ADDRESS1(FP)        ; save the destination address
              11B3      30  0468  1619          BSBW    ADD_REAL                ; compute the sum
              0EA6      30  046B  1620          BSBW    PACK_FLOAT1             ; pack the sum
     5B   E7 AD      D0  046E  1621          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
              1043      30  0472  1622          BSBW    STORE_OPERAND           ; store the result
              16C7      30  0475  1623          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
              FD49      31  0478  1624          BRW     NORMAL_EXIT             ; done
                        047B  1625
                        047B  1626          ;
                        047B  1627          ;       7CFD CLRO - Clear Octaword
                        047B  1628          ;
                        047B  1629          ;       7CFD CLRH - Clear H_floating
                        047B  1630          ;
                        047B  1631  INST_CLRO:                                  ; entrance
                        047B  1632          SET_OP_TYPES    O
              0A4A      30  047F  1633          BSBW    WRITE_ACCESS            ; first operand is write only
                8B      7C  0482  1634          CLRQ    (R11)+                  ; clear the first part of the value
                6B      7C  0484  1635          CLRQ    (R11)                   ; clear the second part of the value
        54 AD   0E      CA  0486  1636          BICL2   #PSLM_NZV,PSL(FP)       ; clear the condition codes except C
        54 AD   04      C8  048A  1637          BISL2   #PSLM_Z,PSL(FP)         ; set the Z bit in the PSL
              FD33      31  048E  1638          BRW     NORMAL_EXIT             ; done
                        0491  1639
                        0491  1640          ;
                        0491  1641          ;       51 CMPF - Compare F_floating
                        0491  1642          ;
                        0491  1643  INST_CMPF:                                 ; entrance
                        0491  1644          SET_OP_TYPES    F
                  10    11  0495  1645          BRB     INST_CMPx
                        0497  1646
                        0497  1647          ;
                        0497  1648          ;       71 CMPD - Compare D_floating
                        0497  1649          ;
                        0497  1650  INST_CMPD:                                 ; entrance
                        0497  1651          SET_OP_TYPES    D
```

```
                 0A   11   049B  1652          BRB     INST_CMPx
                           049D  1653
                           049D  1654          ;
                           049D  1655          ;       51FD CMPG - Compare G_floating
                           049D  1656          ;
                           049D  1657 INST_CMPG:                                       ; entrance
                           049D  1658          SET_OP_TYPES    G
                 04   11   04A1  1659          BRB     INST_CMPx
                           04A3  1660
                           04A3  1661          ;
                           04A3  1662          ;       71FD CMPH - Compare H_floating
                           04A3  1663          ;
                           04A3  1664 INST_CMPH:                                       ; entrance
                           04A3  1665          SET_OP_TYPES    H
                           04A7  1666          ;BRB    INST_CMPx
                           04A7  1667
                           04A7  1668 INST_CMPx:
               0A19   30   04A7  1669          BSBW    READ_ACCESS                     ; first operand is read only
               0D77   30   04AA  1670          BSBW    UNPACK_FLOAT1                   ; unpack and save the value
               0A13   30   04AD  1671          BSBW    READ_ACCESS                     ; second operand is read only
               0D77   30   04B0  1672          BSBW    UNPACK_FLOAT2                   ; unpack and save the value
            51   C7 AD   9E   04B3  1673          MOVAB   OPERAND1(FP),R1                 ; R1 = location of first value
            52   AF AD   9E   04B7  1674          MOVAB   OPERAND2(FP),R2                 ; R2 = location of second value
              16AC   30   04BB  1675          BSBW    COMPARE_REAL                    ; compare the values
              165F   30   04BE  1676          BSBW    SET_CONDITION                   ; set the condition codes
            54 AD    03   CA   04C1  1677          BICL2   #PSL$M_VC,PSL(FP)               ; clear the V bit and C bit in the PSL
              FCFC   31   04C5  1678          BRW     NORMAL_EXIT                     ; done
                           04C8  1679
                           04C8  1680          ;
                           04C8  1681          ;       4C CVTBF - Convert Byte to F_floating
                           04C8  1682          ;
                           04C8  1683 INST_CVTBF:                                      ; entrance
                           04C8  1684          SET_OP_TYPES    B,F
                 56   11   04CE  1685          BRB     INST_CVTBx
                           04D0  1686
                           04D0  1687          ;
                           04D0  1688          ;       6C CVTBD - Convert Byte to D_floating
                           04D0  1689          ;
                           04D0  1690 INST_CVTBD:                                      ; entrance
                           04D0  1691          SET_OP_TYPES    B,D
                 4E   11   04D6  1692          BRB     INST_CVTBx
                           04D8  1693
                           04D8  1694          ;
                           04D8  1695          ;       4CFD CVTBG - Convert Byte to G_floating
                           04D8  1696          ;
                           04D8  1697 INST_CVTBG:                                      ; entrance
                           04D8  1698          SET_OP_TYPES    B,G
                 46   11   04DE  1699          BRB     INST_CVTBx
                           04E0  1700
                           04E0  1701          ;
                           04E0  1702          ;       6CFD CVTBH - Convert Byte to H_floating
                           04E0  1703          ;
                           04E0  1704 INST_CVTBH:                                      ; entrance
                           04E0  1705          SET_OP_TYPES    B,H
                 3E   11   04E6  1706          BRB     INST_CVTBx
                           04E8  1707
                           04E8  1708          ;
```

```
                     04E8  1709          ;       4D CVTWF - Convert Word to F_floating
                     04E8  1710          ;
                     04E8  1711 INST_CVTWF:
                     04E8  1712          SET_OP_TYPES    W,F
36   11              04EE  1713          BRB     INST_CVTWx
                     04F0  1714
                     04F0  1715          ;
                     04F0  1716          ;       6D CVTWD - Convert Word to D_floating
                     04F0  1717          ;
                     04F0  1718 INST_CVTWD:
                     04F0  1719          SET_OP_TYPES    W,D
2E   11              04F6  1720          BRB     INST_CVTWx
                     04F8  1721
                     04F8  1722          ;
                     04F8  1723          ;       4DFD CVTWG - Convert Word to G_floating
                     04F8  1724          ;
                     04F8  1725 INST_CVTWG:                              ; entrance
                     04F8  1726          SET_OP_TYPES    W,G
26   11              04FE  1727          BRB     INST_CVTWx
                     0500  1728
                     0500  1729          ;
                     0500  1730          ;       6DFD CVTWH - Convert Word to H_floating
                     0500  1731          ;
                     0500  1732 INST_CVTWH:                              ; entrance
                     0500  1733          SET_OP_TYPES    W,H
1E   11              0506  1734          BRB     INST_CVTWx
                     0508  1735
                     0508  1736          ;
                     0508  1737          ;       4E CVTLF - Convert Long to F_floating
                     0508  1738          ;
                     0508  1739 INST_CVTLF:                              ; entrance
                     0508  1740          SET_OP_TYPES    L,F
16   11              050E  1741          BRB     INST_CVTLx
                     0510  1742
                     0510  1743          ;
                     0510  1744          ;       6E CVTLD - Convert Long to D_floating
                     0510  1745          ;
                     0510  1746 INST_CVTLD:                              ; entrance
                     0510  1747          SET_OP_TYPES    L,D
0E   11              0516  1748          BRB     INST_CVTLx
                     0518  1749
                     0518  1750          ;
                     0518  1751          ;       4EFD CVTLG - Convert Long to G_floating
                     0518  1752          ;
                     0518  1753 INST_CVTLG:                              ; entrance
                     0518  1754          SET_OP_TYPES    L,G
06   11              051E  1755          BRB     INST_CVTLx
                     0520  1756
                     0520  1757          ;
                     0520  1758          ;       6EFD CVTLH - Convert Long to H_floating
                     0520  1759          ;
                     0520  1760 INST_CVTLH:                              ; entrance
                     0520  1761          SET_OP_TYPES    L,H
                     0526  1762          ;BRB    INST_CVTLx
                     0526  1763
                     0526  1764 INST_CVTBx:
                     0526  1765 INST_CVTWx:
```

```
                                                                B 1
                     0526  1766 INST_CVTLx:
 099A   30   0526  1767          BSBW    READ_ACCESS                ; first operand is read only
 0FC0   30   0529  1768          BSBW    FLOAT_LONG                 ; convert to floating and save value
8B AD   D6   052C  1769          INCL    OP_INDEX(FP)               ; move to second operand
 099A   30   052F  1770          BSBW    WRITE_ACCESS               ; second operand is write only
E7 AD   5B   D0   0532  1771          MOVL    R11,ADDRESS1(FP)           ; save the destination address
 0DDB   30   0536  1772          BSBW    PACK_FLOAT1                ; pack the value
5B   E7 AD   D0   0539  1773          MOVL    ADDRESS1(FP),R11           ; R11 = destination location
 0F78   30   053D  1774          BSBW    STORE_OPERAND              ; store the result
 15FC   30   0540  1775          BSBW    SET_CONDITION1             ; set the condition codes in the PSL
 FC7E   31   0543  1776          BRW     NORMAL_EXIT                ; done
                     0546  1777
                     0546  1778          ;
                     0546  1779          ;       48 CVTFB - Convert F_floating to Byte
                     0546  1780          ;
                     0546  1781 INST_CVTFB:
                     0546  1782          SET_OP_TYPES    F,B
   58   11   054C  1783          BRB     INST_CVTxB
                     054E  1784
                     054E  1785          ;
                     054E  1786          ;       68 CVTDB - Convert D_floating to Byte
                     054E  1787          ;
                     054E  1788 INST_CVTDB:
                     054E  1789          SET_OP_TYPES    D,B
   50   11   0554  1790          BRB     INST_CVTxB
                     0556  1791
                     0556  1792          ;
                     0556  1793          ;       48FD CVTGB - Convert G_floating to Byte
                     0556  1794          ;
                     0556  1795 INST_CVTGB:
                     0556  1796          SET_OP_TYPES    G,B
   48   11   055C  1797          BRB     INST_CVTxB
                     055E  1798
                     055E  1799          ;
                     055E  1800          ;       68FD CVTHB - Convert H_floating to Byte
                     055E  1801          ;
                     055E  1802 INST_CVTHB:
                     055E  1803          SET_OP_TYPES    H,B
   40   11   0564  1804          BRB     INST_CVTxB
                     0566  1805
                     0566  1806          ;
                     0566  1807          ;       49 CVTFW - Convert F_floating to Word
                     0566  1808          ;
                     0566  1809 INST_CVTFW:
                     0566  1810          SET_OP_TYPES    F,W
   38   11   056C  1811          BRB     INST_CVTxW
                     056E  1812
                     056E  1813          ;
                     056E  1814          ;       69 CVTDW - Convert D_floating to Word
                     056E  1815          ;
                     056E  1816 INST_CVTDW:
                     056E  1817          SET_OP_TYPES    D,W
   30   11   0574  1818          BRB     INST_CVTxW
                     0576  1819
                     0576  1820          ;
                     0576  1821          ;       49FD CVTGW - Convert G_floating to Word
                     0576  1822          ;
```

```
                     0576  1823 INST_CVTGW:
                     0576  1824                 SET_OP_TYPES    G,W
28  11               057C  1825                 BRB     INST_CVTxW
                     057E  1826
                     057E  1827                 ;
                     057E  1828                 ;       69FD CVTHW - Convert H_floating to Word
                     057E  1829                 ;
                     057E  1830 INST_CVTHW:
                     057E  1831                 SET_OP_TYPES    H,W
20  11               0584  1832                 BRB     INST_CVTxW
                     0586  1833
                     0586  1834                 ;
                     0586  1835                 ;       4A CVTFL - Convert F_floating to Long
                     0586  1836                 ;
                     0586  1837 INST_CVTFL:
                     0586  1838                 SET_OP_TYPES    F,L
18  11               058C  1839                 BRB     INST_CVTxL
                     058E  1840
                     058E  1841                 ;
                     058E  1842                 ;       6A CVTDL - Convert D_floating to Long
                     058E  1843                 ;
                     058E  1844 INST_CVTDL:
                     058E  1845                 SET_OP_TYPES    D,L
10  11               0594  1846                 BRB     INST_CVTxL
                     0596  1847
                     0596  1848                 ;
                     0596  1849                 ;       4AFD CVTGL - Convert G_floating to Long
                     0596  1850                 ;
                     0596  1851 INST_CVTGL:
                     0596  1852                 SET_OP_TYPES    G,L
08  11               059C  1853                 BRB     INST_CVTxL
                     059E  1854
                     059E  1855                 ;
                     059E  1856                 ;       6AFD CVTHL - Convert H_floating to Long
                     059E  1857                 ;
                     059E  1858 INST_CVTHL:
                     059E  1859                 SET_OP_TYPES    H,L
        00   11      05A4  1860                 BRB     INST_CVTxL
                     05A6  1861
                     05A6  1862 INST_CVTxB:
                     05A6  1863 INST_CVTxW:
                     05A6  1864 INST_CVTxL:
      091A   30      05A6  1865                 BSBW    READ_ACCESS             ; first operand is read only
      0C78   30      05A9  1866                 BSBW    UNPACK_FLOAT1           ; unpack and save the value
    8B AD    D6      05AC  1867                 INCL    OP_INDEX(FP)            ; move to second operand
      091A   30      05AF  1868                 BSBW    WRITE_ACCESS            ; second operand is write only
E7 AD  5B    D0      05B2  1869                 MOVL    R11,ADDRESS1(FP)        ; save the destination location
54 AD  0F    CA      05B6  1870                 BICL2   #PSLM_NZVC,PSL(FP)      ; clear the condition codes
    0F67   30      05BA  1871                 BSBW    FIX_REAL                ; fix the value
02  8B BD    91      05BD  1872                 CMPB    @OP_INDEX(FP),#TYP_W    ; case on datatype
      10   14      05C1  1873                 BGTR    30$                     ; skip if longword
      05   13      05C3  1874                 BEQL    10$                     ; skip if word
   50 50   F6      05C5  1875                 CVTLB   R0,R0                   ; convert long to byte
      03   11      05C8  1876                 BRB     20$                     ; continue
   50 50   F7      05CA  1877 10$:    CVTLW   R0,R0                   ; convert long to word
      04   1C      05CD  1878 20$:    BVC     30$                     ; no overflow - skip
54 AD  02    C8      05CF  1879                 BISL2   #PSLM_V,PSL(FP)         ; indicate integer overflow
```

```
                5B   E7 AD   D0   05D3  1880 30$:     MOVL    ADDRESS1(FP),R11        ; get address of result
                      0EDE   30   05D7  1881          BSBW    STORE_OPERAND           ; store the result
                      1543   30   05DA  1882          BSBW    SET_CONDITION           ; set the condition codes
                      FBE4   31   05DD  1883          BRW     NORMAL_EXIT             ; done
                             05E0  1884
                             05E0  1885          ;
                             05E0  1886          ;       56 CVTFD - Convert F_floating to D_floating
                             05E0  1887          ;
                             05E0  1888 INST_CVTFD:
                             05E0  1889          SET_OP_TYPES    F,D
                        46   11   05E6  1890          BRB     INST_CVTxy
                             05E8  1891
                             05E8  1892          ;
                             05E8  1893          ;       99FD CVTFG - Convert Floating to G_floating
                             05E8  1894          ;
                             05E8  1895 INST_CVTFG:                              ; entrance
                             05E8  1896          SET_OP_TYPES    F,G
                        3E   11   05EE  1897          BRB     INST_CVTxy
                             05F0  1898
                             05F0  1899          ;
                             05F0  1900          ;       98FD CVTFH - Convert F_floating to H_floating
                             05F0  1901          ;
                             05F0  1902 INST_CVTFH:                              ; entrance
                             05F0  1903          SET_OP_TYPES    F,H
                        36   11   05F6  1904          BRB     INST_CVTxy
                             05F8  1905
                             05F8  1906          ;
                             05F8  1907          ;       76 CVTDF - Convert D_floating to H_floating
                             05F8  1908          ;
                             05F8  1909 INST_CVTDF:                              ; entrance
                             05F8  1910          SET_OP_TYPES    D,F
                        2E   11   05FE  1911          BRB     INST_CVTxy
                             0600  1912
                             0600  1913          ;
                             0600  1914          ;       32FD CVTDH - Convert D_floating to H_floating
                             0600  1915          ;
                             0600  1916 INST_CVTDH:
                             0600  1917          SET_OP_TYPES    D,H
                        26   11   0606  1918          BRB     INST_CVTxy
                             0608  1919
                             0608  1920          ;
                             0608  1921          ;       33FD CVTGF - Convert G_floating to F_floating
                             0608  1922          ;
                             0608  1923 INST_CVTGF:
                             0608  1924          SET_OP_TYPES    G,F
                        1E   11   060E  1925          BRB     INST_CVTxy
                             0610  1926
                             0610  1927          ;
                             0610  1928          ;       56FD CVTGH - Convert G_floating to H_floating
                             0610  1929          ;
                             0610  1930 INST_CVTGH:
                             0610  1931          SET_OP_TYPES    G,H
                        16   11   0616  1932          BRB     INST_CVTxy
                             0618  1933
                             0618  1934          ;
                             0618  1935          ;       F6FD CVTHF - Convert H_floating to F_floating
                             0618  1936          ;
```

```
                                                                E 1
                     0618  1937 INST_CVTHF:
                     0618  1938         SET_OP_TYPES    H,F
                0E   11 061E  1939         BRB     INST_CVTxy
                     0620  1940
                     0620  1941         ;
                     0620  1942         ;       F7FD CVTHD - Convert H_floating to D_floating
                     0620  1943         ;
                     0620  1944 INST_CVTHD:
                     0620  1945         SET_OP_TYPES    H,D
                06   11 0626  1946         BRB     INST_CVTxy
                     0628  1947
                     0628  1948         ;
                     0628  1949         ;       76FD CVTHG - COnvert H_floating to G_floating
                     0628  1950         ;
                     0628  1951 INST_CVTHG:
                     0628  1952         SET_OP_TYPES    H,G
                     062E  1953         ;BRB    INST_CVTxy
                     062E  1954
                     062E  1955 INST_CVTxy:
              0892   30 062E  1956         BSBW    READ_ACCESS             ; first operand is read only
              0BF0   30 0631  1957         BSBW    UNPACK_FLOAT1           ; unpack and save the value
            8B AD    D6 0634  1958         INCL    OP_INDEX(FP)            ; move to second operand
              0892   30 0637  1959         BSBW    WRITE_ACCESS            ; second operand is write only
E7 AD    5B    D0 063A  1960         MOVL    R11,ADDRESS1(FP)        ; save the destination location
              0CD3   30 063E  1961         BSBW    PACK_FLOAT1             ; pack the value
5B    E7 AD    D0 0641  1962         MOVL    ADDRESS1(FP),R11        ; R11 = destination location
              0E70   30 0645  1963         BSBW    STORE_OPERAND           ; store the result
              14F4   30 0648  1964         BSBW    SET_CONDITION1          ; set the condition codes in the PSL
              FB76   31 064B  1965         BRW     NORMAL_EXIT             ; done
                     064E  1966
                     064E  1967         ;
                     064E  1968         ;       4B CVTRFL - Convert Rounded F_floating to Long
                     064E  1969         ;
                     064E  1970 INST_CVTRFL:
                     064E  1971         SET_OP_TYPES    F,L
                16   11 0654  1972         BRB     INST_CVTRxL
                     0656  1973
                     0656  1974         ;
                     0656  1975         ;       6B CVTRDL - Convert Rounded D_floating to Long
                     0656  1976         ;
                     0656  1977 INST_CVTRDL:
                     0656  1978         SET_OP_TYPES    D,L
                0E   11 065C  1979         BRB     INST_CVTRxL
                     065E  1980
                     065E  1981         ;
                     065E  1982         ;       4BFD CVTRGL - Convert Rounded G_floating to Long
                     065E  1983         ;
                     065E  1984 INST_CVTRGL:                             ; entrance
                     065E  1985         SET_OP_TYPES    G,L
                06   11 0664  1986         BRB     INST_CVTRxL
                     0666  1987
                     0666  1988         ;
                     0666  1989         ;       6BFD CVTRHL - Convert Rounded H_floating to Long
                     0666  1990         ;
                     0666  1991 INST_CVTRHL:                             ; entrance
                     0666  1992         SET_OP_TYPES    H,L
                     066C  1993         ;BRB    INST_CVTRxL
```

```
                                                F 1
                     066C  1994
                     066C  1995 INST_CVTRxL:
          0854   30  066C  1996         BSBW    READ_ACCESS                 ; first operand is read only
          0BB2   30  066F  1997         BSBW    UNPACK_FLOAT1               ; unpack the value
        8B AD    D6  0672  1998         INCL    OP_INDEX(FP)                ; move to second operand
          0854   30  0675  1999         BSBW    WRITE_ACCESS                ; second operand is write only
E7 AD     5B   D0  0678  2000         MOVL    R11,ADDRESS1(FP)            ; save the destination location
54 AD     0F   CA  067C  2001         BICL2   #PSLM_NZVC,PSL(FP)          ; clear the condition codes
          0EF5   30  0680  2002         BSBW    ROUND_REAL                  ; round the value
E7 BD     50   D0  0683  2003         MOVL    R0,@ADDRESS1(FP)            ; output the value
          1496   30  0687  2004         BSBW    SET_CONDITION               ; set the condition codes
          FB37   31  068A  2005         BRW     NORMAL_EXIT                 ; done
                     068D  2006
                     068D  2007         ;
                     068D  2008         ;       46 DIVF2 - Divide F_floating (Two Operands)
                     068D  2009         ;
                     068D  2010 INST_DIVF2:
                     068D  2011         SET_OP_TYPES    F
            10   11  0691  2012         BRB     INST_DIVx2
                     0693  2013
                     0693  2014         ;
                     0693  2015         ;       66 DIVD2 - Divide D_floating (Two Operands)
                     0693  2016         ;
                     0693  2017 INST_DIVD2:
                     0693  2018         SET_OP_TYPES    D
            0A   11  0697  2019         BRB     INST_DIVx2
                     0699  2020
                     0699  2021         ;
                     0699  2022         ;       46FD DIVG2 - Divide G_floating (Two Operands)
                     0699  2023         ;
                     0699  2024 INST_DIVG2:
                     0699  2025         SET_OP_TYPES    G
            04   11  069D  2026         BRB     INST_DIVx2
                     069F  2027
                     069F  2028         ;
                     069F  2029         ;       66FD DIVH2 - Divide H_floating (Two Operands)
                     069F  2030         ;
                     069F  2031 INST_DIVH2:
                     069F  2032         SET_OP_TYPES    H
                     06A3  2033         ;BRB    INST_DIVx2
                     06A3  2034
                     06A3  2035 INST_DIVx2:                                 ; entrance
          081D   30  06A3  2036         BSBW    READ_ACCESS                 ; first operand is read only
          0B87   30  06A6  2037         BSBW    UNPACK_FLOAT3               ; unpack and save the value
          0829   30  06A9  2038         BSBW    MODIFY_ACCESS               ; second operand is modified
E7 AD     5B   D0  06AC  2039         MOVL    R11,ADDRESS1(FP)            ; save the destination location
          0B77   30  06B0  2040         BSBW    UNPACK_FLOAT2               ; unpack and save the value
          119C   30  06B3  2041         BSBW    DIVIDE_FLOAT                ; compute the quotient
          0C5B   30  06B6  2042         BSBW    PACK_FLOAT1                 ; pack the result
5B    E7 AD    D0  06B9  2043         MOVL    ADDRESS1(FP),R11            ; R11 = destination location
          0DF8   30  06BD  2044         BSBW    STORE_OPERAND               ; store the result
          147C   30  06C0  2045         BSBW    SET_CONDITION1              ; set the condition codes in the PSL
          FAFE   31  06C3  2046         BRW     NORMAL_EXIT                 ; done
                     06C6  2047
                     06C6  2048         ;
                     06C6  2049         ;       47 DIVF3 - Divide F_floating (Three Operands)
                     06C6  2050         ;
```

```
                                06C6  2051 INST_DIVF3:
                                06C6  2052         SET_OP_TYPES     F
                     10   11    06CA  2053         BRB      INST_DIVx3
                                06CC  2054
                                06CC  2055         ;
                                06CC  2056         ;       67 DIVD3 - Divide D_floating (Three Operands)
                                06CC  2057         ;
                                06CC  2058 INST_DIVD3:
                                06CC  2059         SET_OP_TYPES     D
                     0A   11    06D0  2060         BRB      INST_DIVx3
                                06D2  2061
                                06D2  2062         ;
                                06D2  2063         ;       47FD DIVG3 - Divide G_floating (Three Operands)
                                06D2  2064         ;
                                06D2  2065 INST_DIVG3:
                                06D2  2066         SET_OP_TYPES     G
                     04   11    06D6  2067         BRB      INST_DIVx3
                                06D8  2068
                                06D8  2069         ;
                                06D8  2070         ;       67FD DIVH3 - Divide HFLOAT (Three Operands)
                                06D8  2071         ;
                                06D8  2072 INST_DIVH3:                          ; entrance
                                06D8  2073         SET_OP_TYPES     H
                                06DC  2074         ;BRB     INST_DIVx3
                                06DC  2075
                                06DC  2076 INST_DIVx3:
                07E4    30      06DC  2077         BSBW     READ_ACCESS         ; first operand is read only
                0B4E    30      06DF  2078         BSBW     UNPACK_FLOAT3       ; unpack and save the value
                07DE    30      06E2  2079         BSBW     READ_ACCESS         ; second operand is read only
                0B42    30      06E5  2080         BSBW     UNPACK_FLOAT2       ; unpack and save the value
                07E1    30      06E8  2081         BSBW     WRITE_ACCESS        ; third operand is write only
             E7 AD   5B     D0  06EB  2082         MOVL     R11,ADDRESS1(FP)    ; save the destination location
                1160    30      06EF  2083         BSBW     DIVIDE_FLOAT        ; compute the quotient
                0C1F    30      06F2  2084         BSBW     PACK_FLOAT1         ; pack the result
          5B    E7 AD   D0      06F5  2085         MOVL     ADDRESS1(FP),R11    ; R11 = destination location
                0DBC    30      06F9  2086         BSBW     STORE_OPERAND       ; store the result
                1440    30      06FC  2087         BSBW     SET_CONDITION1      ; set the condition codes in the PSL
                FAC2    31      06FF  2088         BRW      NORMAL_EXIT         ; done
                                0702  2089
                                0702  2090         ;
                                0702  2091         ;       54 EMODF - Extended Modulus F_floating
                                0702  2092         ;
                                0702  2093 INST_EMODF:
                                0702  2094         SET_OP_TYPES     F,B,F,L,F
                07B2    30      070E  2095         BSBW     READ_ACCESS         ; first operand is read only
                0B16    30      0711  2096         BSBW     UNPACK_FLOAT2       ; unpack and save the operand
             8B AD   D6      0714  2097         INCL     OP_INDEX(FP)        ; move to second operand
                07A9    30      0717  2098         BSBW     READ_ACCESS         ; second operand is read only
          C3 AD   50     90  071A  2099         MOVB     R0,FRACTION2+12(FP) ; insert bits 0-7 of the fraction
             8B AD   D6      071E  2100         INCL     OP_INDEX(FP)        ; move to third operand
                079F    30      0721  2101         BSBW     READ_ACCESS         ; third operand is read only
                0B09    30      0724  2102         BSBW     UNPACK_FLOAT3       ; unpack and save the value
          FC AD   01     90  0727  2103         MOVB     #FLAGOM,FLAGS(FP)   ; inhibit checking for local store
             8B AD   D6      072B  2104         INCL     OP_INDEX(FP)        ; move to fourth operand
                079B    30      072E  2105         BSBW     WRITE_ACCESS        ; fourth operand is write only
       05 FC AD   01     E5  0731  2106         BBCC     #FLAGT,FLAGS(FP),1$ ; not register mode - skip
       00 FC AD   07     E3  0736  2107         BBCS     #FLAG7,FLAGS(FP),1$ ; make a note
```

```
                                                                H 1
                E7 AD    5B   D0  073B  2108 1$:     MOVL    R11,ADDRESS1(FP)          ; save the first destination address
                   8B AD      D6  073F  2109         INCL    OP_INDEX(FP)              ; move to fifth operand
                     0787     30  0742  2110         BSBW    WRITE_ACCESS              ; fifth operand is write only
                E3 AD    5B   D0  0745  2111         MOVL    R11,ADDRESS2(FP)          ; save the second destination address
                     0FF3     30  0749  2112         BSBW    MULTIPLY_FLOAT            ; compute the product
          DB AD, 50  00    00   F0  074C  2113         INSV    #0,#0,R0,FRACTION1+12(FP); truncate the product if necessary
                54 AD    0F   CA  0752  2114         BICL2   #PSL$M_NZVC,PSL(FP)       ; clear the condition codes
                     0DCB     30  0756  2115         BSBW    FIX_REAL                  ; fix the product
                DF AD    50   D0  0759  2116         MOVL    R0,ADDRESS3(FP)           ; save the integer part
                     0E87     30  075D  2117         BSBW    FRACTION_REAL             ; form the fractional part
                     0BB1     30  0760  2118         BSBW    PACK_FLOAT1               ; pack the fraction value
                        50    DD  0763  2119         PUSHL   R0                        ; push the fraction
             05 FC AD    07   E1  0765  2120         BBC     #FLAG7,FLAGS(FP),2$       ; fourth operand not register - skip
             E7 BD   DF AD    D0  076A  2121         MOVL    ADDRESS3(FP),@ADDRESS1(FP) ; output the integer part
             04 FC AD    01   E1  076F  2122 2$:     BBC     #FLAG1,FLAGS(FP),3$       ; fifth operand not register - skip
                E3 BD    8E   D0  0774  2123         MOVL    (SP)+,@ADDRESS2(FP)       ; output the fraction
                50   04       D0  0778  2124 3$:     MOVL    #4,R0                     ; allow room for dummy store
                   FB5C     30  077B  2125         BSBW    TEST_FRAME                ; move the frame if necessary
             0E FC AD    07   E0  077E  2126         BBS     #FLAG7,FLAGS(FP),4$       ; fourth operand is register - bypass
                5A   04       D0  0783  2127         MOVL    #4,R10                    ; R10 = size of integer part
                5B   E7 AD    D0  0786  2128         MOVL    ADDRESS1(FP),R11          ; R11 = location of fourth operand
                     0A80     30  078A  2129         BSBW    LOCAL_TEST                ; check for a store into local storage
                6B   DF AD    D0  078D  2130         MOVL    ADDRESS3(FP),(R11)        ; output the integer part
             0D FC AD    01   E0  0791  2131 4$:     BBS     #FLAG1,FLAGS(FP),5$       ; fifth operand is register - bypass
                5A   04       D0  0796  2132         MOVL    #4,R10                    ; R10 = size of the fraction
                5B   E3 AD    D0  0799  2133         MOVL    ADDRESS2(FP),R11          ; R11 = location of second operand
                     0A6D     30  079D  2134         BSBW    LOCAL_TEST                ; check for a store into local storage
                6B   8E       D0  07A0  2135         MOVL    (SP)+,(R11)               ; output the fraction
                51   C7 AD    9E  07A3  2136 5$:     MOVAB   OPERAND1(FP),R1           ; R1 = location of the fraction
                     13A2     30  07A7  2137         BSBW    TEST_REAL                 ; test the fraction
                     1373     30  07AA  2138         BSBW    SET_CONDITION             ; set the condition codes
                     FA14     31  07AD  2139         BRW     NORMAL_EXIT               ; done
                                  07B0  2140
                                  07B0  2141 ;
                                  07B0  2142 ;       74 EMODD - Extended Modulus D_floating
                                  07B0  2143 ;
                                  07B0  2144 INST_EMODD:
                                  07B0  2145         SET_OP_TYPES    D,B,D,L,D
                     0704     30  07BC  2146         BSBW    READ_ACCESS               ; first operand is read only
                     0A68     30  07BF  2147         BSBW    UNPACK_FLOAT2             ; unpack and save the operand
                   8B AD      D6  07C2  2148         INCL    OP_INDEX(FP)              ; move to second operand
                     06FB     30  07C5  2149         BSBW    READ_ACCESS               ; second operand is read only
                BF AD    50   90  07C8  2150         MOVB    R0,FRACTION2+8(FP)        ; insert bits 0-7 of the fraction
                     22       11  07CC  2151         BRB     COMMON_EMODdg             ; join common D/G code
                                  07CE  2152
                                  07CE  2153 ;
                                  07CE  2154 ;       54FD EMODG - Extended Modulus G_floating
                                  07CE  2155 ;
                                  07CE  2156 INST_EMODG:                               ; entrance
                                  07CE  2157         SET_OP_TYPES    G,W,G,L,G
                     06E6     30  07DA  2158         BSBW    READ_ACCESS               ; first operand is read only
                     0A4A     30  07DD  2159         BSBW    UNPACK_FLOAT2             ; unpack and save the operand
                   8B AD      D6  07E0  2160         INCL    OP_INDEX(FP)              ; move to second operand
                     06DD     30  07E3  2161         BSBW    READ_ACCESS               ; second operand is read only
                50   50   1B   9C  07E6  2162         ROTL    #27,R0,R0                 ; position the high-order 11 bits
          BF AD 0B   00   50   F0  07EA  2163         INSV    R0,#0,#11,FRACTION2+8(FP) ; insert bits 0-10 of the fraction
                                  07F0  2164 COMMON_EMODdg:
```

```
                8B AD    D6  07F0  2165          INCL    OP_INDEX(FP)                ; move to third operand
                  06CD   30  07F3  2166          BSBW    READ_ACCESS                 ; third operand is read only
                  0A37   30  07F6  2167          BSBW    UNPACK_FLOAT3               ; unpack and save the value
             FC AD   01   90  07F9  2168          MOVB    #FLAGOM,FLAGS(FP)           ; inhibit checking for local store
                8B AD    D6  07FD  2169          INCL    OP_INDEX(FP)                ; move to fourth operand
                  06C9   30  0800  2170          BSBW    WRITE_ACCESS                ; fourth operand is write only
          05 FC AD   01   E5  0803  2171          BBCC    #FLAG1,FLAGS(FP),1$         ; not register mode - skip
          00 FC AD   07   E3  0808  2172          BBCS    #FLAG7,FLAGS(FP),1$         ; make a note
             E7 AD   5B   D0  080D  2173  1$:     MOVL    R11,ADDRESS1(FP)            ; save the first destination address
                8B AD    D6  0811  2174          INCL    OP_INDEX(FP)                ; move to fifth operand
                  06B5   30  0814  2175          BSBW    WRITE_ACCESS                ; fifth operand is write only
             E3 AD   5B   D0  0817  2176          MOVL    R11,ADDRESS2(FP)            ; save the second destination address
                  0F21   30  081B  2177          BSBW    MULTIPLY_FLOAT              ; compute the product
       D7 AD  50   00   00   F0  081E  2178          INSV    #0,#0,R0,FRACTION1+8(FP); truncate the product if necessary
             54 AD   0F   CA  0824  2179          BICL2   #PSLM_NZVC,PSL(FP)          ; clear the condition codes
                  0CF9   30  0828  2180          BSBW    FIX_REAL                    ; fix the product
             DF AD   50   D0  082B  2181          MOVL    R0,ADDRESS3(FP)             ; save the integer part
                  0DB5   30  082F  2182          BSBW    FRACTION_REAL               ; form the fractional part
                  0ADF   30  0832  2183          BSBW    PACK_FLOAT1                 ; pack the fraction value
                    03   BB  0835  2184          PUSHR   #^M<R0,R1>                  ; push the fraction
          05 FC AD   07   E1  0837  2185          BBC     #FLAG7,FLAGS(FP),2$         ; fourth operand not register - skip
          E7 BD   DF AD   D0  083C  2186          MOVL    ADDRESS3(FP),@ADDRESS1(FP) ; output the integer part
          04 FC AD   01   E1  0841  2187  2$:     BBC     #FLAG1,FLAGS(FP),3$         ; fifth operand not register - skip
             E3 BD   8E   7D  0846  2188          MOVQ    (SP)+,@ADDRESS2(FP)         ; output the fraction
                50   08   D0  084A  2189  3$:     MOVL    #8,R0                       ; allow room for dummy store
                  FA8A   30  084D  2190          BSBW    TEST_FRAME                  ; move the frame if necessary
          0E FC AD   07   E0  0850  2191          BBS     #FLAG7,FLAGS(FP),4$         ; fourth operand is register - bypass
                5A   04   D0  0855  2192          MOVL    #4,R10                      ; R10 = size of integer part
             5B   E7 AD   D0  0858  2193          MOVL    ADDRESS1(FP),R11            ; R11 = location of fourth operand
                  09AE   30  085C  2194          BSBW    LOCAL_TEST                  ; check for a store into local storage
                6B   DF AD   D0  085F  2195          MOVL    ADDRESS3(FP),(R11)          ; output the integer part
          0D FC AD   01   E0  0863  2196  4$:     BBS     #FLAG1,FLAGS(FP),5$         ; fifth operand is register - bypass
                5A   08   D0  0868  2197          MOVL    #8,R10                      ; R10 = size of the fraction
             5B   E3 AD   D0  086B  2198          MOVL    ADDRESS2(FP),R11            ; R11 = location of second operand
                  099B   30  086F  2199          BSBW    LOCAL_TEST                  ; check for a store into local storage
                6B   8E   7D  0872  2200          MOVQ    (SP)+,(R11)                 ; output the fraction
             51   C7 AD   9E  0875  2201  5$:     MOVAB   OPERAND1(FP),R1             ; R1 = location of the fraction
                  12D0   30  0879  2202          BSBW    TEST_REAL                   ; test the fraction
                  12A1   30  087C  2203          BSBW    SET_CONDITION               ; set the condition codes
                  F942   31  087F  2204          BRW     NORMAL_EXIT                 ; done
                         0882  2205
                         0882  2206          ;
                         0882  2207          ;       74FD EMODH - Extended Modulus HFLOAT
                         0882  2208          ;
                         0882  2209  INST_EMODH:                                      ; entrance
                         0882  2210          SET_OP_TYPES    H,W,H,L,H
                  0632   30  088E  2211          BSBW    READ_ACCESS                 ; first operand is read only
                  0996   30  0891  2212          BSBW    UNPACK_FLOAT2               ; unpack and save the operand
                8B AD    D6  0894  2213          INCL    OP_INDEX(FP)                ; move to second operand
                  0629   30  0897  2214          BSBW    READ_ACCESS                 ; second operand is read only
          50    50   1F   9C  089A  2215          ROTL    #31,R0,R0                   ; position the high-order 15 bits
       B7 AD    0F   00   50   F0  089E  2216          INSV    R0,#0,#15,FRACTION2(FP) ; insert bits 0-14 of the fraction
                8B AD    D6  08A4  2217          INCL    OP_INDEX(FP)                ; move to third operand
                  0619   30  08A7  2218          BSBW    READ_ACCESS                 ; third operand is read only
                  0983   30  08AA  2219          BSBW    UNPACK_FLOAT3               ; unpack and save the value
             FC AD   01   90  08AD  2220          MOVB    #FLAGOM,FLAGS(FP)           ; inhibit checking for local store
                8B AD    D6  08B1  2221          INCL    OP_INDEX(FP)                ; move to fourth operand
```

```
                                                                J 1
                0615   30   08B4  2222          BSBW    WRITE_ACCESS                    ; fourth operand is write only
          05 FC AD 01   E5   08B7  2223          BBCC    #FLAG1,FLAGS(FP),1$             ; not register mode - skip
          00 FC AD 07   E3   08BC  2224          BBCS    #FLAG7,FLAGS(FP),1$             ; make a note
             E7 AD 5B   D0   08C1  2225 1$:      MOVL    R11,ADDRESS1(FP)                ; save the first destination address
                8B AD   D6   08C5  2226          INCL    OP_INDEX(FP)                    ; move to fifth operand
                0601    30   08C8  2227          BSBW    WRITE_ACCESS                    ; fifth operand is write only
             E3 AD 5B   D0   08CB  2228          MOVL    R11,ADDRESS2(FP)                ; save the second destination address
                0E6D    30   08CF  2229          BSBW    MULTIPLY_FLOAT                  ; compute the product
       CF AD 50 00 00   F0   08D2  2230          INSV    #0,#0,R0,FRACTION1(FP)          ; truncate the product if necessary
             54 AD 0F   CA   08D8  2231          BICL2   #PSLM_NZVC,PSL(FP)              ; clear the condition codes
                0C45    30   08DC  2232          BSBW    FIX_REAL                        ; fix the product
             DF AD 50   D0   08DF  2233          MOVL    R0,ADDRESS3(FP)                 ; save the integer part
                0D01    30   08E3  2234          BSBW    FRACTION_REAL                   ; form the fractional part
                0A2B    30   08E6  2235          BSBW    PACK_FLOAT1                     ; pack the fraction value
                  0F    BB   08E9  2236          PUSHR   #^M<R0,R1,R2,R3>                ; push the fraction
          05 FC AD 07   E1   08EB  2237          BBC     #FLAG7,FLAGS(FP),2$             ; fourth operand not register - skip
          E7 BD DF AD   D0   08F0  2238          MOVL    ADDRESS3(FP),@ADDRESS1(FP)      ; output the integer part
          0A FC AD 01   E1   08F5  2239 2$:      BBC     #FLAG1,FLAGS(FP),3$             ; fifth operand not register - bypass
             5B E3 AD   D0   08FA  2240          MOVL    ADDRESS2(FP),R11                ; R11 = location of the register
                8B 8E   7D   08FE  2241          MOVQ    (SP)+,(R11)+                    ; output first part of the fraction
                6B 8E   7D   0901  2242          MOVQ    (SP)+,(R11)                     ; output second part of the fraction
                50 10   D0   0904  2243 3$:      MOVL    #16,R0                          ; allow room for dummy store
                F9D0    30   0907  2244          BSBW    TEST_FRAME                      ; move the frame if necessary
          0E FC AD 07   E0   090A  2245          BBS     #FLAG7,FLAGS(FP),4$             ; fourth operand is register - bypass
             5A 04   D0   090F  2246          MOVL    #4,R10                          ; R10 = size of integer part
             5B E7 AD   D0   0912  2247          MOVL    ADDRESS1(FP),R11                ; R11 = location of fourth operand
                08F4    30   0916  2248          BSBW    LOCAL_TEST                      ; check for a store into local storage
             6B DF AD   D0   0919  2249          MOVL    ADDRESS3(FP),(R11)              ; output the integer part
          10 FC AD 01   E0   091D  2250 4$:      BBS     #FLAG1,FLAGS(FP),5$             ; fifth operand is register - bypass
             5A 10   D0   0922  2251          MOVL    #16,R10                         ; R10 = size of the fraction
             5B E3 AD   D0   0925  2252          MOVL    ADDRESS2(FP),R11                ; R11 = location of second operand
                08E1    30   0929  2253          BSBW    LOCAL_TEST                      ; check for a store into local storage
                8B 8E   7D   092C  2254          MOVQ    (SP)+,(R11)+                    ; output first part of the fraction
                6B 8E   7D   092F  2255          MOVQ    (SP)+,(R11)                     ; output second part of the fraction
       51 C7 AD 9E   0932  2256 5$:      MOVAB   OPERAND1(FP),R1                 ; R1 = location of the fraction
                1213    30   0936  2257          BSBW    TEST_REAL                       ; test the fraction
                11E4    30   0939  2258          BSBW    SET_CONDITION                   ; set the condition codes
                F885    31   093C  2259          BRW     NORMAL_EXIT                     ; done
                             093F  2260
                             093F  2261          ;
                             093F  2262          ;       52 MNEGF - Move Negated F_floating
                             093F  2263          ;
                             093F  2264 INST_MNEGF:
                             093F  2265          SET_OP_TYPES    F
                10   11      0943  2266          BRB     INST_MNEGx
                             0945  2267
                             0945  2268          ;
                             0945  2269          ;       72 MNEGD - Move Negated D_floating
                             0945  2270          ;
                             0945  2271 INST_MNEGD:
                             0945  2272          SET_OP_TYPES    D
                0A   11      0949  2273          BRB     INST_MNEGx
                             094B  2274
                             094B  2275          ;
                             094B  2276          ;       52FD MNEGG - Move Negated G_floating
                             094B  2277          ;
                             094B  2278 INST_MNEGG:
```

```
                     094B  2279          SET_OP_TYPES    G
          04   11    094F  2280          BRB     INST_MNEGx
                     0951  2281
                     0951  2282          ;
                     0951  2283          ;       72FD MNEGH - Move Negated HFLOAT
                     0951  2284          ;
                     0951  2285  INST_MNEGH:
                     0951  2286          SET_OP_TYPES    H
                     0955  2287          ;BRB    INST_MNEGx
                     0955  2288
                     0955  2289  INST_MNEGx:
          056B   30  0955  2290          BSBW    READ_ACCESS             ; first operand is read only
          08C9   30  0958  2291          BSBW    UNPACK_FLOAT1           ; unpack and save the value
          0B85   30  095B  2292          BSBW    NEGATE_REAL             ; negate the value
          056B   30  095E  2293          BSBW    WRITE_ACCESS            ; second operand is write only
E7 AD     5B   D0  0961  2294          MOVL    R11,ADDRESS1(FP)        ; save the destination location
          09AC   30  0965  2295          BSBW    PACK_FLOAT1             ; pack the value
5B   E7 AD     D0  0968  2296          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
          0B49   30  096C  2297          BSBW    STORE_OPERAND           ; store the result
          11CD   30  096F  2298          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
          F84F   31  0972  2299          BRW     NORMAL_EXIT             ; done
```

```
                                                                L 1
                     0975  2301          ;
                     0975  2302          ;       7EFD MOVAO - Move Address of Octaword
                     0975  2303          ;
                     0975  2304          ;       7EFD MOVAH - Move Address of H_floating
                     0975  2305          ;
                     0975  2306 INST_MOVAO:                                  ; entrance
                     0975  2307          SET_OP_TYPES    O,L
           0560   30 097B  2308          BSBW    ADDRESS_ACCESS              ; first operand is address only
E7 AD      5B   D0 097E  2309          MOVL    R11,ADDRESS1(FP)            ; save the operand address
      8B AD   D6 0982  2310          INCL    OP_INDEX(FP)                ; move to second operand
           0544   30 0985  2311          BSBW    WRITE_ACCESS                ; second operand is write only
6B    E7 AD   D0 0988  2312          MOVL    ADDRESS1(FP),(R11)          ; output the operand address
           1191   30 098C  2313          BSBW    SET_CONDITION               ; capture the condition code
54 AD      02   CA 098F  2314          BICL2   #PSL$M_V,PSL(FP)            ; clear the V bit in the PSL
           F82E   31 0993  2315          BRW     NORMAL_EXIT                 ; done
                     0996  2316
                     0996  2317          ;
                     0996  2318          ;       50 MOVF - Move F_floating
                     0996  2319          ;
                     0996  2320 INST_MOVF:
                     0996  2321          SET_OP_TYPES    F
                10   11 099A  2322          BRB     INST_MOVx
                     099C  2323
                     099C  2324          ;
                     099C  2325          ;       70 MOVD - Move D_floating
                     099C  2326          ;
                     099C  2327 INST_MOVD:
                     099C  2328          SET_OP_TYPES    D
                0A   11 09A0  2329          BRB     INST_MOVx
                     09A2  2330
                     09A2  2331          ;
                     09A2  2332          ;       50FD MOVG - Move G_floating
                     09A2  2333          ;
                     09A2  2334 INST_MOVG:
                     09A2  2335          SET_OP_TYPES    G
                04   11 09A6  2336          BRB     INST_MOVx
                     09A8  2337
                     09A8  2338          ;
                     09A8  2339          ;       70FD MOVH - Move HFLOAT
                     09A8  2340          ;
                     09A8  2341 INST_MOVH:
                     09A8  2342          SET_OP_TYPES    H
                     09AC  2343          ;BRB    INST_MOVx
                     09AC  2344
                     09AC  2345 INST_MOVx:
           0514   30 09AC  2346          BSBW    READ_ACCESS                 ; first operand is read only
           0872   30 09AF  2347          BSBW    UNPACK_FLOAT1               ; unpack and save the value
           0517   30 09B2  2348          BSBW    WRITE_ACCESS                ; second operand is write only
E7 AD      5B   D0 09B5  2349          MOVL    R11,ADDRESS1(FP)            ; save the destination location
           0958   30 09B9  2350          BSBW    PACK_FLOAT1                 ; pack the value
5B    E7 AD   D0 09BC  2351          MOVL    ADDRESS1(FP),R11            ; R11 = destination location
           0AF5   30 09C0  2352          BSBW    STORE_OPERAND               ; store the result
51    C7 AD   9E 09C3  2353          MOVAB   OPERAND1(FP),R1             ; R1 = location of the value
           1182   30 09C7  2354          BSBW    TEST_REAL                   ; test the value
           1153   30 09CA  2355          BSBW    SET_CONDITION               ; set the condition codes
54 AD      02   CA 09CD  2356          BICL2   #PSL$M_V,PSL(FP)            ; clear the V bit in the PSL
           F7F0   31 09D1  2357          BRW     NORMAL_EXIT                 ; done
```

```
                                                     M 1
                     09D4  2358
                     09D4  2359          ;
                     09D4  2360          ;       7DFD MOVO - Move Octaword
                     09D4  2361          ;
                     09D4  2362 INST_MOVO:
                     09D4  2363          SET_OP_TYPES    O
        04E8   30    09D8  2364          BSBW    READ_ACCESS             ; first operand is read only
C7 AD    50   7D     09DB  2365          MOVQ    R0,OPERAND1(FP)         ; save the first part of the value
CF AD    52   7D     09DF  2366          MOVQ    R2,OPERAND1+8(FP)       ; save the second part of the value
        04E6   30    09E3  2367          BSBW    WRITE_ACCESS            ; second operand is write only
E7 AD    5B   D0     09E6  2368          MOVL    R11,ADDRESS1(FP)        ; save destination location
        1168   30    09EA  2369          BSBW    TEST_OCTA               ; test the value
        1130   30    09ED  2370          BSBW    SET_CONDITION           ; set the condition codes
54 AD    02   CA     09F0  2371          BICL2   #PSL$M_V,PSL(FP)        ; clear the V bit in the PSL
5B   E7 AD   D0     09F4  2372          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
8B   C7 AD   7D     09F8  2373          MOVQ    OPERAND1(FP),(R11)+     ; output the first part of the result
6B   CF AD   7D     09FC  2374          MOVQ    OPERAND1+8(FP),(R11)    ; output the second part of the result
        F7C1   31    0A00  2375          BRW     NORMAL_EXIT             ; done
                     0A03  2376
                     0A03  2377          ;
                     0A03  2378          ;       44 MULF2 - Multiply F_floating (Two Operands)
                     0A03  2379          ;
                     0A03  2380 INST_MULF2:
                     0A03  2381          SET_OP_TYPES    F
              10   11    0A07  2382          BRB     INST_MULx2
                     0A09  2383
                     0A09  2384          ;
                     0A09  2385          ;       64 MULD2 - Multiply D_floating (Two Operands)
                     0A09  2386          ;
                     0A09  2387 INST_MULD2:
                     0A09  2388          SET_OP_TYPES    D
              0A   11    0A0D  2389          BRB     INST_MULx2
                     0A0F  2390
                     0A0F  2391          ;
                     0A0F  2392          ;       44FD MULG2 - Multiply G_floating (Two Operands)
                     0A0F  2393          ;
                     0A0F  2394 INST_MULG2:
                     0A0F  2395          SET_OP_TYPES    G
              04   11    0A13  2396          BRB     INST_MULx2
                     0A15  2397
                     0A15  2398          ;
                     0A15  2399          ;       64FD MULH2 - Multiply HFLOAT (Two Operands)
                     0A15  2400          ;
                     0A15  2401 INST_MULH2:                              ; entrance
                     0A15  2402          SET_OP_TYPES    H
                     0A19  2403          ;BRB    INST_MULx2
                     0A19  2404
                     0A19  2405 INST_MULx2:
        04A7   30    0A19  2406          BSBW    READ_ACCESS             ; first operand is read only
        080B   30    0A1C  2407          BSBW    UNPACK_FLOAT2           ; unpack and save the first operand
        04B3   30    0A1F  2408          BSBW    MODIFY_ACCESS           ; second operand is modified
E7 AD    5B   D0     0A22  2409          MOVL    R11,ADDRESS1(FP)        ; save the destination location
        0807   30    0A26  2410          BSBW    UNPACK_FLOAT3           ; unpack and save the second operand
        0D13   30    0A29  2411          BSBW    MULTIPLY_FLOAT          ; compute the product
        08E5   30    0A2C  2412          BSBW    PACK_FLOAT1             ; pack the value
5B   E7 AD   D0     0A2F  2413          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
        0A82   30    0A33  2414          BSBW    STORE_OPERAND           ; store the result
```

```
                     1106   30  0A36  2415          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
                     F788   31  0A39  2416          BRW     NORMAL_EXIT             ; done
                            0A3C  2417
                            0A3C  2418          ;
                            0A3C  2419          ;       45 MULF3 - Multiply F_floating (Three Operands)
                            0A3C  2420          ;
                            0A3C  2421  INST_MULF3:
                            0A3C  2422          SET_OP_TYPES    F
                       10   11  0A40  2423          BRB     INST_MULx3
                            0A42  2424
                            0A42  2425          ;
                            0A42  2426          ;       65 MULD3 - Multiply D_floating (Three Operands)
                            0A42  2427          ;
                            0A42  2428  INST_MULD3:
                            0A42  2429          SET_OP_TYPES    D
                       0A   11  0A46  2430          BRB     INST_MULx3
                            0A48  2431
                            0A48  2432          ;
                            0A48  2433          ;       45FD MULG3 - Multiply G_floating (Three Operands)
                            0A48  2434          ;
                            0A48  2435  INST_MULG3:
                            0A48  2436          SET_OP_TYPES    G
                       04   11  0A4C  2437          BRB     INST_MULx3
                            0A4E  2438
                            0A4E  2439          ;
                            0A4E  2440          ;       65FD MULH3 - Multiply HFLOAT (Three Operands)
                            0A4E  2441          ;
                            0A4E  2442  INST_MULH3:
                            0A4E  2443          SET_OP_TYPES    H
                            0A52  2444          ;BRB    INST_MULx3
                            0A52  2445
                            0A52  2446  INST_MULx3:
                     046E   30  0A52  2447          BSBW    READ_ACCESS             ; first operand is read only
                     07D2   30  0A55  2448          BSBW    UNPACK_FLOAT2           ; unpack and save the value
                     0468   30  0A58  2449          BSBW    READ_ACCESS             ; second operand is read only
                     07D2   30  0A5B  2450          BSBW    UNPACK_FLOAT3           ; unpack and save the value
                     046B   30  0A5E  2451          BSBW    WRITE_ACCESS            ; third operand is write only
                  E7 AD   5B   D0  0A61  2452          MOVL    R11,ADDRESS1(FP)        ; save the destination location
                     0CD7   30  0A65  2453          BSBW    MULTIPLY_FLOAT          ; compute the product
                     08A9   30  0A68  2454          BSBW    PACK_FLOAT1             ; pack the value
                  5B   E7 AD   D0  0A6B  2455          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
                     0A46   30  0A6F  2456          BSBW    STORE_OPERAND           ; store the result
                     10CA   30  0A72  2457          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
                     F74C   31  0A75  2458          BRW     NORMAL_EXIT             ; done
                            0A78  2459
                            0A78  2460          ;
                            0A78  2461          ;       55 POLYF - Evaluate Polynomial F_floating
                            0A78  2462          ;
                            0A78  2463  INST_POLYF:                             ; entrance
                            0A78  2464          SET_OP_TYPES    F,W,B,F         ; phantom fourth operand
                  2F 54 AD   1B   E1  0A80  2465          BBC     #PSL_FPD,PSL(FP),1$     ; not instruction resumption - bypass
               50 1C AD   18   08   EF  0A85  2466          EXTZV   #8,#24,REG_R2(FP),R0    ; R0 = saved instruction length
                     50 AD   50   C0  0A8B  2467          ADDL2   R0,REG_PC(FP)           ; position PC to end of instruction
                     FA AD   50   80  0A8F  2468          ADDB2   R0,REGMOD_PC(FP)        ; increment the PC modification count
                     50   14 AD   D0  0A93  2469          MOVL    REG_R0(FP),R0           ; R0 = result so far
                        0790   30  0A97  2470          BSBW    UNPACK_FLOAT2           ; unpack and save the value
                     51   AF AD   9E  0A9A  2471          MOVAB   OPERAND2(FP),R1         ; R1 = location of OPERAND2
```

```
                10AB   30   0A9E  2472          BSBW    TEST_REAL                       ; test the value
                107C   30   0AA1  2473          BSBW    SET_CONDITION                   ; set the condition codes
           50    18 AD  D0   0AA4  2474          MOVL    REG_R1(FP),R0                   ; R0 = argument value
                 0785   30   0AA8  2475          BSBW    UNPACK_FLOAT3                   ; unpack and save the value
           1C AD    1F  91   0AAB  2476          CMPB    #31,REG_R2(FP)                  ; is the iteration count too large ?
                    7C  1E   0AAF  2477          BGEQU   4$                              ; no - resume the instruction
                 11D2   31   0AB1  2478          BRW     OPERAND_FAULT                   ; process the reserved operand
                 040C   30   0AB4  2479 1$:      BSBW    READ_ACCESS                     ; first operand is read only
                 0776   30   0AB7  2480          BSBW    UNPACK_FLOAT3                   ; unpack and save the value
                8B AD   D6   0ABA  2481          INCL    OP_INDEX(FP)                    ; look at second operand
                 0403   30   0ABD  2482          BSBW    READ_ACCESS                     ; second operand is read only
              50    1F   D1   0AC0  2483          CMPL    #31,R0                          ; is the value reserved ?
                    03  1E   0AC3  2484          BGEQU   2$                              ; no - skip
                 11BE   31   0AC5  2485          BRW     OPERAND_FAULT                   ; process the reserved operand
           E7 AD    50   D0   0AC8  2486 2$:      MOVL    R0,ADDRESS1(FP)                 ; save the operand value
                8B AD   D6   0ACC  2487          INCL    OP_INDEX(FP)                    ; look at third operand
                 040C   30   0ACF  2488          BSBW    ADDRESS_ACCESS                  ; third operand is address only
        6B 04    FD AD  0C   0AD2  2489          PROBER  MODE(FP),#4,(R11)               ; can we read the first longword?
                    06  12   0AD7  2490          BNEQ    3$                              ; yes - skip
              5A    04   D0   0AD9  2491          MOVL    #4,R10                          ; R10 = size of probe
                 10CC   30   0ADC  2492          BSBW    READ_FAULT                      ; process the access violation
              50    8B   D0   0ADF  2493 3$:      MOVL    (R11)+,R0                       ; R0 = the first table entry
           E3 AD    5B   D0   0AE2  2494          MOVL    R11,ADDRESS2(FP)                ; save the following address
                8B AD   D6   0AE6  2495          INCL    OP_INDEX(FP)                    ; use "fourth operand" type for rest
                 073E   30   0AE9  2496          BSBW    UNPACK_FLOAT2                   ; unpack and save the value
                 082B   30   0AEC  2497          BSBW    PACK_FLOAT2                     ; pack the value
           14 AD    50   D0   0AEF  2498          MOVL    R0,REG_R0(FP)                   ; save the value in user's R0
           51    AF AD  9E   0AF3  2499          MOVAB   OPERAND2(FP),R1                 ; R1 = location of OPERAND2
                 1052   30   0AF7  2500          BSBW    TEST_REAL                       ; test the value
                 1023   30   0AFA  2501          BSBW    SET_CONDITION                   ; set the condition codes
              54 AD    03  CA   0AFD  2502          BICL2   #PSL$M_V!C,PSL(FP)              ; clear the V bit and C bit in the PSL
           20 AD    E3 AD  D0   0B01  2503          MOVL    ADDRESS2(FP),REG_R3(FP)         ; store next location in user's R3
           1C AD    E7 AD  D0   0B06  2504          MOVL    ADDRESS1(FP),REG_R2(FP)         ; save the remaining iteration count
              50    FA AD  98   0B0B  2505          CVTBL   REGMOD_PC(FP),R0                ; R0 = PC register modifications
                    EB AD  7C   0B0F  2506          CLRQ    REGMOD_R0(FP)                   ; clear modifications for R0 to R7
                    F3 AD  7C   0B12  2507          CLRQ    REGMOD_R8(FP)                   ; clear modifications for R8 to PC
              FA AD    50   90   0B15  2508          MOVB    R0,REGMOD_PC(FP)                ; put the PC modifications back
                    50  D7   0B19  2509          DECL    R0                              ; subtract the operation code length
     1C AD    18    08   50   F0   0B1B  2510          INSV    R0,#8,#24,REG_R2(FP)            ; save the value for resumption
                 07FC   30   0B21  2511          BSBW    PACK_FLOAT3                     ; pack the argument value
           18 AD    50   D0   0B24  2512          MOVL    R0,REG_R1(FP)                   ; save it for resumption
        00 54 AD    1B  E2   0B28  2513          BBSS    #PSL_FPD,PSL(FP),4$             ; indicate first part done in PSL
                1C AD   95   0B2D  2514 4$:      TSTB    REG_R2(FP)                      ; more iterations to perform ?
                    3C  13   0B30  2515          BEQL    6$                              ; no - finish up
                 0C0A   30   0B32  2516          BSBW    MULTIPLY_FLOAT                  ; multiply the result and argument
                    50  D6   0B35  2517          INCL    R0                              ; increment the shift count
     DB AD    50    00   00   F0   0B37  2518          INSV    #0,#0,R0,FRACTION1+12(FP)       ; truncate the product
              5B    20 AD  D0   0B3D  2519          MOVL    REG_R3(FP),R11                  ; R11 = location of next table entry
        6B    04    FD AD  0C   0B41  2520          PROBER  MODE(FP),#4,(R11)               ; can we read the next coefficient ?
                    06  12   0B46  2521          BNEQ    5$                              ; yes - skip
                 5A   04   D0   0B48  2522          MOVL    #4,R10                          ; R10 = size of probe
                 105D   30   0B4B  2523          BSBW    READ_FAULT                      ; process the access violation
              50    20 BD  D0   0B4E  2524 5$:      MOVL    @REG_R3(FP),R0                  ; R0 = next coefficient
                 06D5   30   0B52  2525          BSBW    UNPACK_FLOAT2                   ; unpack and save the value
                 0AC6   30   0B55  2526          BSBW    ADD_REAL                        ; add the coefficient to the product
                 07B9   30   0B58  2527          BSBW    PACK_FLOAT1                     ; pack the result so far
           14 AD    50   D0   0B5B  2528          MOVL    R0,REG_R0(FP)                   ; save the value
```

```
                06C8   30  0B5F  2529          BSBW    UNPACK_FLOAT2           ; unpack the result for next iteration
                0FDA   30  0B62  2530          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
          20 AD   04   C0  0B65  2531          ADDL2   #4,REG_R3(FP)           ; update the table pointer
             1C AD     97  0B69  2532          DECB    REG_R2(FP)              ; decrement the iteration counter
                BF     11  0B6C  2533          BRB     4$                      ; start the next iteration
             18 AD     D4  0B6E  2534 6$:      CLRL    REG_R1(FP)              ; clear the user's R1
             1C AD     D4  0B71  2535          CLRL    REG_R2(FP)              ; clear the user's R2
       00 54 AD   1B   E5  0B74  2536          BBCC    #PSL_FPD,PSL(FP),7$     ; indicate instruction complete
                F648   31  0B79  2537 7$:      BRW     NORMAL_EXIT             ; done
                       0B7C  2538
                       0B7C  2539          ;
                       0B7C  2540          ;       75 POLYD - Evaluate Polynomial D_floating
                       0B7C  2541          ;
                       0B7C  2542 INST_POLYD:                                 ; entrance
                       0B7C  2543          SET_OP_TYPES    D,W,B,D             ; phantom fourth operand
                   08  11  0B84  2544          BRB     INST_POLYdg
                       0B86  2545
                       0B86  2546          ;
                       0B86  2547          ;       55FD POLYG - Evaluate Polynomial G_floating
                       0B86  2548          ;
                       0B86  2549 INST_POLYG:                                 ; entrance
                       0B86  2550          SET_OP_TYPES    G,W,B,G             ; phantom fourth operand
                       0B8E  2551          ;BRB    INST_POLYdg
                       0B8E  2552
                       0B8E  2553 INST_POLYdg:
       32 54 AD   1B   E1  0B8E  2554          BBC     #PSL_FPD,PSL(FP),1$     ; not instruction resumption - bypass
    50    1C AD   18   08  EF  0B93  2555          EXTZV   #8,#24,REG_R2(FP),R0    ; R0 = saved instruction length
          50 AD     50   C0  0B99  2556          ADDL2   R0,REG_PC(FP)           ; position PC to end of instruction
          FA AD     50   80  0B9D  2557          ADDB2   R0,REGMOD_PC(FP)        ; increment the PC modification count
          50   14 AD     7D  0BA1  2558          MOVQ    REG_R0(FP),R0           ; R0,R1 = result so far
                0682     30  0BA5  2559          BSBW    UNPACK_FLOAT2           ; unpack and save the value
          51   AF AD     9E  0BA8  2560          MOVAB   OPERAND2(FP),R1         ; R1 = location of OPERAND2
                0F9D     30  0BAC  2561          BSBW    TEST_REAL               ; test the value
                0F6E     30  0BAF  2562          BSBW    SET_CONDITION           ; set the condition codes
          50   24 AD     7D  0BB2  2563          MOVQ    REG_R4(FP),R0           ; R0,R1 = argument value
                0677     30  0BB6  2564          BSBW    UNPACK_FLOAT3           ; unpack and save the value
          1C AD   1F     91  0BB9  2565          CMPB    #31,REG_R2(FP)          ; is the iteration count too large ?
                  03     1E  0BBD  2566          BGEQU   14$                     ; no - resume the instruction
                10C4     31  0BBF  2567          BRW     OPERAND_FAULT           ; process the reserved operand
                0081     31  0BC2  2568 14$:     BRW     4$
                02FB     30  0BC5  2569 1$:      BSBW    READ_ACCESS             ; first operand is read only
                0665     30  0BC8  2570          BSBW    UNPACK_FLOAT3           ; unpack and save the value
                8B AD    D6  0BCB  2571          INCL    OP_INDEX(FP)            ; look at second operand
                02F2     30  0BCE  2572          BSBW    READ_ACCESS             ; second operand is read only
             50   1F     D1  0BD1  2573          CMPL    #31,R0                  ; is the value reserved ?
                  03     1E  0BD4  2574          BGEQU   2$                      ; no - skip
                10AD     31  0BD6  2575          BRW     OPERAND_FAULT           ; process the reserved operand
          E7 AD     50   D0  0BD9  2576 2$:     MOVL    R0,ADDRESS1(FP)         ; save the operand value
                8B AD    D6  0BDD  2577          INCL    OP_INDEX(FP)            ; look at third operand
                02FB     30  0BE0  2578          BSBW    ADDRESS_ACCESS          ; third operand is address only
 6B     08   FD AD     0C  0BE3  2579          PROBER  MODE(FP),#8,(R11)       ; can we read the first quadword ?
                  06     12  0BE8  2580          BNEQ    3$                      ; yes - skip
             5A   08     D0  0BEA  2581          MOVL    #8,R10                  ; R10 = size of probe
                0FBB     30  0BED  2582          BSBW    READ_FAULT              ; process the access violation
             50   8B     7D  0BF0  2583 3$:     MOVQ    (R11)+,R0               ; R0,R1 = the first table entry
          E3 AD   5B     D0  0BF3  2584          MOVL    R11,ADDRESS2(FP)        ; save the following address
                8B AD    D6  0BF7  2585          INCL    OP_INDEX(FP)            ; use "fourth operand" type for rest
```

```
                062D   30   0BFA  2586          BSBW    UNPACK_FLOAT2           ; unpack and save the value
                071A   30   0BFD  2587          BSBW    PACK_FLOAT2             ; pack the value
           14 AD  50   7D   0C00  2588          MOVQ    R0,REG_R0(FP)           ; save the value in user's R0-R1
           51  AF AD   9E   0C04  2589          MOVAB   OPERAND2(FP),R1         ; R1 = location of OPERAND2
                0F41   30   0C08  2590          BSBW    TEST_REAL               ; test the value
                0F12   30   0C0B  2591          BSBW    SET_CONDITION           ; set the condition codes
           54 AD  03   CA   0C0E  2592          BICL2   #PSL$M_VC,PSL(FP)       ; clear the V bit and C bit in the PSL
        20 AD  E3 AD   D0   0C12  2593          MOVL    ADDRESS2(FP),REG_R3(FP) ; store next location in user's R3
        1C AD  E7 AD   D0   0C17  2594          MOVL    ADDRESS1(FP),REG_R2(FP) ; save the remaining iteration count
           50  FA AD   98   0C1C  2595          CVTBL   REGMOD_PC(FP),R0        ; R0 = PC register modifications
               EB AD   7C   0C20  2596          CLRQ    REGMOD_R0(FP)           ; clear modifications for R0 to R7
               F3 AD   7C   0C23  2597          CLRQ    REGMOD_R8(FP)           ; clear modifications for R8 to PC
           FA AD  50   90   0C26  2598          MOVB    R0,REGMOD_PC(FP)        ; put the PC modifications back
                50   D7   0C2A  2599          DECL    R0                      ; subtract the operation code length
           08   8B BD   91   0C2C  2600          CMPB    @OP_INDEX(FP),#TYP_G    ; POLYG?
                02   12   0C30  2601          BNEQ    31$                     ; Skip if not
                50   D7   0C32  2602          DECL    R0                      ; Subtract another opcode byte length
1C AD   18   08  50   F0   0C34  2603 31$:     INSV    R0,#8,#24,REG_R2(FP)    ; save the value for resumption
                06E3   30   0C3A  2604          BSBW    PACK_FLOAT3             ; pack the argument value
           24 AD  50   7D   0C3D  2605          MOVQ    R0,REG_R4(FP)           ; save it for resumption
        00 54 AD  1B   E2   0C41  2606          BBSS    #PSL_FPD,PSL(FP),4$     ; indicate first part done in PSL
              1C AD  95   0C46  2607 4$:        TSTB    REG_R2(FP)              ; more iterations to perform ?
                3C   13   0C49  2608          BEQL    6$                      ; no - finish up
                0AF1   30   0C4B  2609          BSBW    MULTIPLY_FLOAT          ; multiply the result and argument
                50   D6   0C4E  2610          INCL    R0                      ; increment the shift count
D7 AD   50   00   00   F0   0C50  2611          INSV    #0,#0,R0,FRACTION1:8(FP); truncate the product
           5B   20 AD  D0   0C56  2612          MOVL    REG_R3(FP),R11          ; R11 = location of next table entry
        6B   08   FD AD  0C   0C5A  2613          PROBER  MODE(FP),#8,(R11)       ; can we read the next coefficient ?
                06   12   0C5F  2614          BNEQ    5$                      ; yes - skip
              5A   08   D0   0C61  2615          MOVL    #8,R10                  ; R10 = size of probe
                0F44   30   0C64  2616          BSBW    READ_FAULT              ; process the access violation
           50   20 BD  7D   0C67  2617 5$:       MOVQ    @REG_R3(FP),R0          ; R0 = next coefficient
                05BC   30   0C6B  2618          BSBW    UNPACK_FLOAT2           ; unpack and save the value
                09AD   30   0C6E  2619          BSBW    ADD_REAL                ; add the coefficient to the product
                06A0   30   0C71  2620          BSBW    PACK_FLOAT1             ; pack the result so far
           14 AD  50   7D   0C74  2621          MOVQ    R0,REG_R0(FP)           ; save the value
                05AF   30   0C78  2622          BSBW    UNPACK_FLOAT2           ; unpack the result for next iteration
                0EC1   30   0C7B  2623          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
           20 AD  08   C0   0C7E  2624          ADDL2   #8,REG_R3(FP)           ; update the table pointer
              1C AD  97   0C82  2625          DECB    REG_R2(FP)              ; decrement the iteration counter
                BF   11   0C85  2626          BRB     4$                      ; start the next iteration
              1C AD  D4   0C87  2627 6$:        CLRL    REG_R2(FP)              ; clear the user's R2
              24 AD  7C   0C8A  2628          CLRQ    REG_R4(FP)              ; clear the user's R4 and R5
        00 54 AD  1B   E5   0C8D  2629          BBCC    #PSL_FPD,PSL(FP),7$     ; indicate instruction complete
                F52F   31   0C92  2630 7$:        BRW     NORMAL_EXIT             ; done
                            0C95  2631
                            0C95  2632          ;
                            0C95  2633          ;       75FD POLYH - Evaluate Polynomial HFLOAT
                            0C95  2634          ;
                            0C95  2635 INST_POLYH:                              ; entrance
                            0C95  2636          SET_OP_TYPES    H,W,B,H         ; use "phantom" fourth argument
     48 54 AD  1B   E1   0C9D  2637          BBC     #PSL_FPD,PSL(FP),3$     ; not instruction resumption - bypass
50   24 AD  18   08   EF   0CA2  2638          EXTZV   #8,#24,REG_R4(FP),R0    ; R0 = saved instruction length
           50 AD  50   C0   0CA8  2639          ADDL2   R0,REG_PC(FP)           ; position PC to end of instruction
           FA AD  50   80   0CAC  2640          ADDB2   R0,REGMOD_PC(FP)        ; increment the PC modification count
           50   14 AD  7D   0CB0  2641          MOVQ    REG_R0(FP),R0           ; R0,R1 = first part of result so far
           52   1C AD  7D   0CB4  2642          MOVQ    REG_R2(FP),R2           ; R2,R3 = second part of result so far
```

```
                 056F   30   0CB8  2643          BSBW    UNPACK_FLOAT2           ; unpack and save the value
           51    AF AD   9E   0CBB  2644          MOVAB   OPERAND2(FP),R1         ; R1 = location of OPERAND2
             0E8A   30   0CBF  2645          BSBW    TEST_REAL               ; test the value
             0E5B   30   0CC2  2646          BSBW    SET_CONDITION           ; set the condition codes
        5B    4C AD   D0   0CC5  2647          MOVL    REG_SP(FP),R11          ; R11 = user's stack pointer
   6B   10    FD AD   0C   0CC9  2648          PROBER  MODE(FP),#16,(R11)      ; can we read a stacked octaword ?
                06   12   0CCE  2649          BNEQ    1$                      ; yes - skip
           5A   10   D0   0CD0  2650          MOVL    #16,R10                 ; R10 = size of probe
             0ED5   30   0CD3  2651          BSBW    READ_FAULT              ; process the access violation
        50    8B   7D   0CD6  2652  1$:      MOVQ    (R11)+,R0               ; R0,R1 = first part of argument
        52    6B   7D   0CD9  2653          MOVQ    (R11),R2                ; R2,R3 = second part of argument
             0551   30   0CDC  2654          BSBW    UNPACK_FLOAT3           ; unpack and save the value
        54    1F   91   0CDF  2655          CMPB    #31,R4                  ; is the iteration count too large ?
                03   1F   0CE2  2656          BLSSU   2$                      ; no - skip
             0F9F   31   0CE4  2657          BRW     OPERAND_FAULT           ; processed the reserved operand
             00A4   31   0CE7  2658  2$:      BRW     7$                      ; resume the instruction
             01D6   30   0CEA  2659  3$:      BSBW    READ_ACCESS             ; first operand is read only
             0540   30   0CED  2660          BSBW    UNPACK_FLOAT3           ; unpack and save the value
          8B AD   D6   0CF0  2661          INCL    OP_INDEX(FP)            ; look at second operand
             01CD   30   0CF3  2662          BSBW    READ_ACCESS             ; second operand is read only
        50    1F   D1   0CF6  2663          CMPL    #31,R0                  ; is the value reserved ?
                03   1E   0CF9  2664          BGEQU   4$                      ; no - skip
             0F88   31   0CFB  2665          BRW     OPERAND_FAULT           ; process the reserved operand
     E7 AD    50   D0   0CFE  2666  4$:      MOVL    R0,ADDRESS1(FP)         ; save the operand value
          8B AD   D6   0D02  2667          INCL    OP_INDEX(FP)            ; look at third operand
             01D6   30   0D05  2668          BSBW    ADDRESS_ACCESS          ; third operand is address only
     E3 AD    5B   D0   0D08  2669          MOVL    R11,ADDRESS2(FP)        ; save the table address
          8B AD   D6   0D0C  2670          INCL    OP_INDEX(FP)            ; use dummy fourth operand for datatype
             060E   30   0D0F  2671          BSBW    PACK_FLOAT3             ; pack the argument value
     4C AD    10   C2   0D12  2672          SUBL2   #16,REG_SP(FP)          ; decrement the user SP
     F9 AD    10   82   0D16  2673          SUBB2   #16,REGMOD_SP(FP)       ; remember the decrement
        5B    4C AD   D0   0D1A  2674          MOVL    REG_SP(FP),R11          ; R11 = user's stack pointer
   6B   10    FD AD   0D   0D1E  2675          PROBEW  MODE(FP),#16,(R11)      ; can we stack the argument ?
                06   12   0D23  2676          BNEQ    5$                      ; yes - skip
           5A   10   D0   0D25  2677          MOVL    #16,R10                 ; R10 = size of probe
             0EF1   30   0D28  2678          BSBW    WRITE_FAULT             ; process the access violation
        8B    50   7D   0D2B  2679  5$:      MOVQ    R0,(R1)+                ; save first part of argument
        6B    52   7D   0D2E  2680          MOVQ    R2,(R11)                ; save second part of argument
        5B    E3 AD   D0   0D31  2681          MOVL    ADDRESS2(FP),R11        ; R11 = coefficient table address
   6B   10    FD AD   0C   0D35  2682          PROBER  MODE(FP),#16,(R11)      ; can we read the first octaword ?
                06   12   0D3A  2683          BNEQ    6$                      ; yes - skip
           5A   10   D0   0D3C  2684          MOVL    #16,R10                 ; R10 = size of probe
             0E69   30   0D3F  2685          BSBW    READ_FAULT              ; process the access violation
        50    8B   7D   0D42  2686  6$:      MOVQ    (R11)+,R0               ; R0,R1 = first part of coefficient
        52    8B   7D   0D45  2687          MOVQ    (R11)+,R2               ; R2,R3 = second part of coefficient
        E3 AD    5B   D0   0D48  2688          MOVL    R11,ADDRESS2(FP)        ; save the following address
             04DB   30   0D4C  2689          BSBW    UNPACK_FLOAT2           ; unpack and save the value
             05C8   30   0D4F  2690          BSBW    PACK_FLOAT2             ; pack the value again
        14 AD    50   7D   0D52  2691          MOVQ    R0,REG_R0(FP)           ; save first part of value so far
        1C AD    52   7D   0D56  2692          MOVQ    R2,REG_R2(FP)           ; save second part of value so far
        51    AF AD   9E   0D5A  2693          MOVAB   OPERAND2(FP),R1         ; R1 = location of OPERAND2
             0DEB   30   0D5E  2694          BSBW    TEST_REAL               ; test the value
             0DBC   30   0D61  2695          BSBW    SET_CONDITION           ; set the condition codes
        54 AD    03   CA   0D64  2696          BICL2   #PSL$M_V,PSL(FP)        ; clear the V bit and C bit in the PSL
     24 AD    E7 AD   D0   0D68  2697          MOVL    ADDRESS1(FP),REG_R4(FP) ; save the remaining iteration count
     28 AD    E3 AD   D0   0D6D  2698          MOVL    ADDRESS2(FP),REG_R5(FP) ; save the next coefficient address
        50    FA AD   98   0D72  2699          CVTBL   REGMOD_PC(FP),R0        ; R0 = PC register modifications
```

```
                                                F 2
                EB AD    7C  0D76  2700          CLRQ    REGMOD_R0(FP)           ; clear modifications for R0 to R7
                F3 AD    7C  0D79  2701          CLRQ    REGMOD_R8(FP)           ; clear modifications for R8 to PC
             FA AD   50  90  0D7C  2702          MOVB    R0,REGMOD_PC(FP)        ; put the PC modifications back
                50   02  C2  0D80  2703          SUBL2   #2,R0                   ; subtract the operation code length
          24 AD   18   08  50  F0  0D83  2704          INSV    R0,#8,#24,REG_R4(FP)    ; save the value for resumption
             00 54 AD   1B  E2  0D89  2705          BBSS    #PSL_FPD,PSL(FP),7$     ; indicate first part done in PSL
                24 AD   95  0D8E  2706  7$:     TSTB    REG_R4(FP)              ; more iterations to perform ?
                   46   13  0D91  2707          BEQL    9$                      ; no - finish up
                 09A9   30  0D93  2708          BSBW    MULTIPLY_FLOAT          ; multiply the result and argument
                   50   D6  0D96  2709          INCL    R0                      ; increment the shift count
          CF AD   50   00   00  F0  0D98  2710          INSV    #0,#0,R0,FRACTION1(FP)  ; truncate the product
                5B   28 AD   D0  0D9E  2711          MOVL    REG_R5(FP),R11          ; R11 = location of next table entry
             6B   10   FD AD   0C  0DA2  2712          PROBER  MODE(FP),#16,(R11)      ; can we read the next coefficient ?
                   06   12  0DA7  2713          BNEQ    8$                      ; yes - skip
                5A   10   D0  0DA9  2714          MOVL    #16,R10                 ; R10 = size of probe
                 0DFC   30  0DAC  2715          BSBW    READ_FAULT              ; process the access violation
                5B   28 AD   D0  0DAF  2716  8$:     MOVL    REG_R5(FP),R11          ; R11 = location of next table entry
                50   8B   7D  0DB3  2717          MOVQ    (R11)+,R0               ; R0,R1 = first part of next entry
                52   6B   7D  0DB6  2718          MOVQ    (R11),R2                ; R2,R3 = second part of next entry
                   046E   30  0DB9  2719          BSBW    UNPACK_FLOAT2           ; unpack and save the value
                   085F   30  0DBC  2720          BSBW    ADD_REAL                ; add the coefficient to the product
                   0552   30  0DBF  2721          BSBW    PACK_FLOAT1             ; pack the result so far
                14 AD   50   7D  0DC2  2722          MOVQ    R0,REG_R0(FP)           ; save the first part of the value
                1C AD   52   7D  0DC6  2723          MOVQ    R2,REG_R2(FP)           ; save the second part of the value
                   045D   30  0DCA  2724          BSBW    UNPACK_FLOAT2           ; unpack the result for next iteration
                   0D6F   30  0DCD  2725          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
                28 AD   10   C0  0DD0  2726          ADDL2   #16,REG_R5(FP)          ; update the table pointer
                24 AD   97  0DD4  2727          DECB    REG_R4(FP)              ; decrement the iteration counter
                   B5   11  0DD7  2728          BRB     7$                      ; start the next iteration
                24 AD   D4  0DD9  2729  9$:     CLRL    REG_R4(FP)              ; clear the user's R4
             4C AD   10   C0  0DDC  2730          ADDL2   #16,REG_SP(FP)          ; unstack the argument value
          00 54 AD   1B   E5  0DE0  2731          BBCC    #PSL_FPD,PSL(FP),10$    ; indicate instruction complete
                 F3DC   31  0DE5  2732  10$:     BRW     NORMAL_EXIT             ; done
                        0DE8  2733
                        0DE8  2734                  ;
                        0DE8  2735                  ;       7FFD PUSHAO - Push Address of Octaword
                        0DE8  2736                  ;
                        0DE8  2737                  ;       7FFD PUSHAH - Push Address of H_floating
                        0DE8  2738                  ;
                        0DE8  2739  INST_PUSHAO:
                        0DE8  2740          SET_OP_TYPES    O
                 00EF   30  0DEC  2741          BSBW    ADDRESS_ACCESS          ; first operand is address only
             4C AD   04   C2  0DEF  2742          SUBL2   #4,REG_SP(FP)           ; decrement the user SP
             F9 AD   04   82  0DF3  2743          SUBB2   #4,REGMOD_SP(FP)        ; remember the decrement
                50   5B   D0  0DF7  2744          MOVL    R11,R0                  ; R0 = the operand address
             5B   4C AD   D0  0DFA  2745          MOVL    REG_SP(FP),R11          ; R11 = user's stack pointer
          6B   04   FD AD   0D  0DFE  2746          PROBEW  MODE(FP),#4,(R11)       ; can we write the operand address ?
                   06   12  0E03  2747          BNEQ    1$                      ; yes - skip
                5A   04   D0  0E05  2748          MOVL    #4,R10                  ; R10 = size of probe
                 0E11   30  0E08  2749          BSBW    WRITE_FAULT             ; process the access violation
             4C BD   50   D0  0E0B  2750  1$:     MOVL    R0,@REG_SP(FP)          ; stack the operand address
                50   D5  0E0F  2751          TSTL    R0                      ; test the operand address
                 0D0C   30  0E11  2752          BSBW    SET_CONDITION           ; capture the condition code
             54 AD   02   CA  0E14  2753          BICL2   #PSL_M_V,PSL(FP)        ; clear the V bit in the PSL
                 F3A9   31  0E18  2754          BRW     NORMAL_EXIT             ; done
                        0E1B  2755
                        0E1B  2756                  ;
```

```
                                                                G 2
                     0E1B  2757          ;       42 SUBF2 - Subtract F_floating (Two Operands)
                     0E1B  2758          ;
                     0E1B  2759 INST_SUBF2:
                     0E1B  2760          SET_OP_TYPES    F
          10   11    0E1F  2761          BRB     INST_SUBx2
                     0E21  2762
                     0E21  2763          ;
                     0E21  2764          ;       62 SUBD2 - Subtract D_floating (Two Operands)
                     0E21  2765          ;
                     0E21  2766 INST_SUBD2:
                     0E21  2767          SET_OP_TYPES    D
          0A   11    0E25  2768          BRB     INST_SUBx2
                     0E27  2769
                     0E27  2770          ;
                     0E27  2771          ;       42FD SUBG2 - Subtract G_floating (Two Operands)
                     0E27  2772          ;
                     0E27  2773 INST_SUBG2:
                     0E27  2774          SET_OP_TYPES    G
          04   11    0E2B  2775          BRB     INST_SUBx2
                     0E2D  2776
                     0E2D  2777          ;
                     0E2D  2778          ;       62FD SUBH2 - Subtract HFLOAT (Two Operands)
                     0E2D  2779          ;
                     0E2D  2780 INST_SUBH2:
                     0E2D  2781          SET_OP_TYPES    H
                     0E31  2782          ;BRB    INST_SUBx2
                     0E31  2783
                     0E31  2784 INST_SUBx2:
        008F   30    0E31  2785          BSBW    READ_ACCESS             ; first operand is read only
        03ED   30    0E34  2786          BSBW    UNPACK_FLOAT1           ; unpack and save the value
        06A9   30    0E37  2787          BSBW    NEGATE_REAL             ; negate the value
        0098   30    0E3A  2788          BSBW    MODIFY_ACCESS           ; second operand is modified
  E7 AD   5B   D0    0E3D  2789          MOVL    R11,ADDRESS1(FP)        ; save the destination location
        03E6   30    0E41  2790          BSBW    UNPACK_FLOAT2           ; unpack and save the value
        07D7   30    0E44  2791          BSBW    ADD_REAL                ; compute the sum
        04CA   30    0E47  2792          BSBW    PACK_FLOAT1             ; pack the value
  5B    E7 AD  D0    0E4A  2793          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
        0667   30    0E4E  2794          BSBW    STORE_OPERAND           ; store result
        0CEB   30    0E51  2795          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
        F36D   31    0E54  2796          BRW     NORMAL_EXIT             ; done
                     0E57  2797
                     0E57  2798          ;
                     0E57  2799          ;       43 SUBF3 - Subtract F_floating (Three Operands)
                     0E57  2800          ;
                     0E57  2801 INST_SUBF3:
                     0E57  2802          SET_OP_TYPES    F
          10   11    0E5B  2803          BRB     INST_SUBx3
                     0E5D  2804
                     0E5D  2805          ;
                     0E5D  2806          ;       63 SUBD3 - Subtract D_floating (Three Operands)
                     0E5D  2807          ;
                     0E5D  2808 INST_SUBD3:
                     0E5D  2809          SET_OP_TYPES    D
          0A   11    0E61  2810          BRB     INST_SUBx3
                     0E63  2811
                     0E63  2812          ;
                     0E63  2813          ;       43FD SUBG3 - Subtract G_floating (Three Operands)
```

```
                                                H  2
                     0E63  2814                ;
                     0E63  2815 INST_SUBG3:
                     0E63  2816                SET_OP_TYPES    G
           04   11   0E67  2817                BRB     INST_SUBx3
                     0E69  2818
                     0E69  2819                ;
                     0E69  2820                ;       63FD SUBH3 - Subtract HFLOAT (Three Operands)
                     0E69  2821                ;
                     0E69  2822 INST_SUBH3:
                     0E69  2823                SET_OP_TYPES    H
                     0E6D  2824                ;BRB    INST_SUBx3
                     0E6D  2825
                     0E6D  2826 INST_SUBx3:
           0053   30 0E6D  2827                BSBW    READ_ACCESS             ; first operand is read only
           03B1   30 0E70  2828                BSBW    UNPACK_FLOAT1           ; unpack and save the value
           066D   30 0E73  2829                BSBW    NEGATE_REAL             ; negate the value
           004A   30 0E76  2830                BSBW    READ_ACCESS             ; second operand is read only
           03AE   30 0E79  2831                BSBW    UNPACK_FLOAT2           ; unpack and save the value
           004D   30 0E7C  2832                BSBW    WRITE_ACCESS            ; third operand is write only
     E7 AD   5B   D0 0E7F  2833                MOVL    R11,ADDRESS1(FP)        ; save the destination location
           0798   30 0E83  2834                BSBW    ADD_REAL                ; compute the sum
           048B   30 0E86  2835                BSBW    PACK_FLOAT1             ; pack the value
     5B    E7 AD   D0 0E89  2836                MOVL    ADDRESS1(FP),R11        ; R11 = destination location
           0628   30 0E8D  2837                BSBW    STORE_OPERAND           ; store result
           0CAC   30 0E90  2838                BSBW    SET_CONDITION1          ; set the condition codes in the PSL
           F32E   31 0E93  2839                BRW     NORMAL_EXIT             ; done
                     0E96  2840
                     0E96  2841                ;
                     0E96  2842                ;       53 TSTF - Test F_floating
                     0E96  2843                ;
                     0E96  2844 INST_TSTF:
                     0E96  2845                SET_OP_TYPES    F
           10   11   0E9A  2846                BRB     INST_TSTx
                     0E9C  2847
                     0E9C  2848                ;
                     0E9C  2849                ;       73 TSTD - Test D_floating
                     0E9C  2850                ;
                     0E9C  2851 INST_TSTD:
                     0E9C  2852                SET_OP_TYPES    D
           0A   11   0EA0  2853                BRB     INST_TSTx
                     0EA2  2854
                     0EA2  2855                ;
                     0EA2  2856                ;       53FD TSTG - Test G_floating
                     0EA2  2857                ;
                     0EA2  2858 INST_TSTG:
                     0EA2  2859                SET_OP_TYPES    G
           04   11   0EA6  2860                BRB     INST_TSTx
                     0EA8  2861
                     0EA8  2862                ;
                     0EA8  2863                ;       73FD TSTH - Test HFLOAT
                     0EA8  2864                ;
                     0EA8  2865 INST_TSTH:
                     0EA8  2866                SET_OP_TYPES    H
                     0EAC  2867                ;BRB    INST_TSTx
                     0EAC  2868
                     0EAC  2869 INST_TSTx:
         0014   30   0EAC  2870                BSBW    READ_ACCESS             ; first operand is read only
```

```
                0372   30  0EAF  2871          BSBW    UNPACK_FLOAT1                   ; unpack and save the value
51    C7 AD     9E   0EB2  2872          MOVAB   OPERAND1(FP),R1                 ; R1 = location of the value
                0C93   30  0EB6  2873          BSBW    TEST_REAL                       ; test the value
                0C64   30  0EB9  2874          BSBW    SET_CONDITION                   ; set the condition codes
54 AD             03   CA  0EBC  2875          BICL2   #PSL$M_VC,PSL(FP)               ; clear the V bit and C bit in the PSL
                F301   31  0EC0  2876          BRW     NORMAL_EXIT                     ; done
                       0EC3  2877          ;
```

```
                        ;*****************************************************************************
                        ;*                                                                           *
                        ;*                                                                           *
                        ;*          Routines for Scanning Instruction Operands                       *
                        ;*                                                                           *
                        ;*                                                                           *
                        ;*****************************************************************************
                        ;
                        ;Introduction
                        ;------------
                        ;
                        ;    The following section contains a complex of routines for
                        ;scanning the operands of an instruction and determining the
                        ;values and locations of operands. The code contains full error
                        ;checking and also checks for the situations that the
                        ;architecture considers to be unpredictable. In order to make
                        ;the operand scanning in the instruction emulation routines
                        ;clear, separate entrys appear for each possible kind of
                        ;operand.
                        ;
                        ;Operand Scanning Routines
                        ;------------------------
                        ;
                        ;    The operand scanning routines are entered by subroutine
                        ;branching and have entry names of the form
                        ;
                        ;        <access type>_<data type>
                        ;
                        ;in which <access type> is any one of the following
                        ;
                        ;        READ            operand is read only
                        ;        WRITE           operand is write only
                        ;        MODIFY          operand is both read and write
                        ;        ADDRESS         only the operand address is required
                        ;        BRANCH          relative branch destintion
                        ;
                        ;and in which <data type> is any one of the following
                        ;
                        ;        BYTE            byte
                        ;        WORD            word
                        ;        LONG            longword
                        ;        QUAD            quadword
                        ;        OCTA            octaword
                        ;        FLOAT           F-format floating
                        ;        DFLOAT          D-format floating
                        ;        GFLOAT          G-format floating
                        ;        HFLOAT          H-format floating
                        ;
                        ;For an access type of BRANCH the data type refers to the
                        ;size of the relative address rather than the properties of
                        ;any addressed information.
                        ;
                        ;    Entrys only exist for those types of operands which appear
                        ;in the emulated instructions. If additional entrys are
                        ;required they can be added easily enough.
```

```
                                                ;
                                                ;       When the routines are entered they scan the next
                                                ; instruction operand starting at the value of the user's PC
                                                ; and check the operand for validity. If any exceptions are
                                                ; detected during operand scanning they are processed immediatly
                                                ; and the routines do not return. Any changes that are made to
                                                ; any of the registers (including PC) are recorded in the change
                                                ; bytes so faults will be handled properly.
                                                ;
                                                ;       When the routines return the following rules describe the
                                                ; contents of the registers:
                                                ;
                                                ;       o  If the access type is READ or MODIFY and the data type
                                                ;          is BYTE, WORD, LONG, or FLOAT, then R0 is the value
                                                ;          of the operand. If the data type is BYTE or WORD then
                                                ;          the value is sign extended to a longword.
                                                ;
                                                ;       o  If the access type is READ or MODIFY and the data type
                                                ;          is QUAD, DFLOAT, or GFLOAT, then R0-R1 contains the
                                                ;          value of the operand.
                                                ;
                                                ;       o  If the access type is READ or MODIFY and the data type
                                                ;          is OCTA or HFLOAT then R0-R3 contains the value of the
                                                ;          operand.
                                                ;
                                                ;       o  If the access type is WRITE, MODIFY, ADDRESS, or
                                                ;          BRANCH, then R11 is the address of the operand or the
                                                ;          branch destination.
                                                ;
                                                ; If the instruction operand specifies register mode, then the
                                                ; address associated with the operand is the location of the
                                                ; emulated register. If an instruction operand with WRITE or
                                                ; MODIFY access addresses the Emulator's local storage then it
                                                ; is changed to an address that won't do any harm. This is
                                                ; consistant with the notion that the area below the user's
                                                ; stack pointer is being continually garbaged. This check is not
                                                ; performed if flag bit 0 is set. The routines set flag bit 1 if
                                                ; the operand is a register mode operand.
                                                ;
                                                ;       The subroutine LOCAL_TEST is available for checking for
                                                ; stores into the Emulator's local storage in places where this
                                                ; is not done automatically.
                                                ;
                                                ;
                                                ; Exceptions
                                                ; ----------
                                                ;
                                                ;       The instruction operand scanning routines perform complete
                                                ; error checking and immediatly signal any exceptions detected.
                                                ; All of these exceptions are faults. The change bytes are
                                                ; constantly kept up to date so the instruction will be left in
                                                ; a consistant state for restarting if a fault occurs.
                                                ;
                                                ;       All fetches from memory done in scanning the instruction
                                                ; operand or in fetching the operand or operand address are
                                                ; probed and access violations are signaled if the probes fail.
                                                ; All of the addressing modes specified by the architecture to
```

```
                                ;
                                ;       be reserved addressing modes or unpredictable are checked for
                                ;       and are signaled as reserved addressing modes if they are
                                ;       detected.
                                ;
                                ;
                                ;       Routine Organization
                                ;       --------------------
                                ;
                                ;           Except for BRANCH access mode for which there are only
                                ;       isolated routines, the code at the individual routine
                                ;       entrances simply loads the data type code into R9 and branches
                                ;       to a routine for the access type. This routine in turn loads
                                ;       the access type code into R8 and branches to the routine
                                ;       GET_SPECIFIER which process the operand specifier byte.
                                ;
                                ;           GET_SPECIFIER loads the length of the data type into R10
                                ;       and the operand specifier byte into R0. The high and low
                                ;       order nibbles of this byte are stored in R1 and R2. The
                                ;       register R7 which is reserved for the index modification is
                                ;       cleared. The routine now branches on the high order nibble
                                ;       to a routine which will handle the specific kind of operand.
                                ;
                                ;           For literals the values are expanded immediatly and the
                                ;       routine returns.
                                ;
                                ;           For index mode operand specifiers, the index modification
                                ;       is computed and the next operand specifier byte is loaded into
                                ;       R0 and decomposed as before. Again we branch on the high order
                                ;       nibble but this time certain addressing modes which are
                                ;       illegal with indexing are checked for. Also for those
                                ;       addressing modes which change register values a check is made
                                ;       that the register is not the same as the index register.
                                ;
                                ;           For register mode operands the address of the emulated
                                ;       register is loaded into R11. A check is made that the operand
                                ;       does not contain PC. Then flag bit 1 is set and control passes
                                ;       to the operand reading routine if the access type is READ or
                                ;       MODIFY and the routine returns otherwise.
                                ;
                                ;           For the remaining addressing modes the operand addresses
                                ;       are computed in a straightforward manner and loaded into R11.
                                ;       for some of these addressing modes the values of registers may
                                ;       be changed. These changes are reflected in the change bytes.
                                ;       When the operand address is computed control passes to the
                                ;       operand accessing routine.
                                ;
                                ;           For ADDRESS access mode operands the operand accessing
                                ;       routine returns but for all others it probes the operand
                                ;       address and also checks for writes into the Emulator's local
                                ;       storage unless flag bit 0 is set. If the operand is READ or
                                ;       MODIFY access control passes to the operand reading routine.
                                ;
                                ;           The operand reading routine simply reads the operand
                                ;       value into the registers starting at R0 and then returns.
                                ;       Bytes and Words are sign extended into longwords. However
                                ;       this routine does not check for reserved floating values
                                ;       since this is done by the unpack routines.
```

```
                OEC3  3050          ;
```

```
                                0EC3  3052          ;
                                0EC3  3053          ;       Process a Read Only Operand
                                0EC3  3054          ;
                                0EC3  3055 READ_ACCESS:                             ; entrance
                58    01   D0   0EC3  3056          MOVL    #TYPE_READ,R8          ; R8 = access type
            59    8B BD   9A   0EC6  3057          MOVZBL  @OP_INDEX(FP),R9       ; R9 = data type
                    24    11   0ECA  3058          BRB     GET_SPECIFIER          ; scan the operand
                                0ECC  3059          ;
                                0ECC  3060          ;       Process a Write Only Operand
                                0ECC  3061          ;
                                0ECC  3062 WRITE_ACCESS:                            ; entrance
                58    02   D0   0ECC  3063          MOVL    #TYPE_WRITE,R8         ; R8 = access type
            59    8B BD   9A   0ECF  3064          MOVZBL  @OP_INDEX(FP),R9       ; R9 = data type
                    1B    11   0ED3  3065          BRB     GET_SPECIFIER          ; scan the operand
                                0ED5  3066          ;
                                0ED5  3067          ;       Process a Modified Operand
                                0ED5  3068          ;
                                0ED5  3069 MODIFY_ACCESS:                           ; entrance
                58    03   D0   0ED5  3070          MOVL    #TYPE_MODIFY,R8        ; R8 = access type
            59    8B BD   9A   0ED8  3071          MOVZBL  @OP_INDEX(FP),R9       ; R9 = data type
                    12    11   0EDC  3072          BRB     GET_SPECIFIER          ; scan the operand
                                0EDE  3073          ;
                                0EDE  3074          ;       Process an Addressed Operand
                                0EDE  3075          ;
                                0EDE  3076 ADDRESS_ACCESS:                          ; entrance
                58    04   D0   0EDE  3077          MOVL    #TYPE_ADDRESS,R8       ; R8 = access type
            59    8B BD   9A   0EE1  3078          MOVZBL  @OP_INDEX(FP),R9       ; R9 = data type
                    09    11   0EE5  3079          BRB     GET_SPECIFIER          ; scan the operand
                                0EE7  3080          ;
                                0EE7  3081          ;       Table of Data Type Lengths
                                0EE7  3082          ;
                                0EE7  3083 LENGTHS:                                 ; table origin
                          01    0EE7  3084          .BYTE   1                      ; 1 - byte
                          02    0EE8  3085          .BYTE   2                      ; 2 - word
                          04    0EE9  3086          .BYTE   4                      ; 3 - longword
                          08    0EEA  3087          .BYTE   8                      ; 4 - quadword
                          10    0EEB  3088          .BYTE   16                     ; 5 - octaword
                          04    0EEC  3089          .BYTE   4                      ; 6 - F_floating
                          08    0EED  3090          .BYTE   8                      ; 7 - D_floating
                          08    0EEE  3091          .BYTE   8                      ; 8 - G_floating
                          10    0EEF  3092          .BYTE   16                     ; 9 - H_floating
                                0EF0  3093          ;
                                0EF0  3094          ;       Process the Next Operand Specifier
                                0EF0  3095          ;
                                0EF0  3096 GET_SPECIFIER:                           ; entrance
          5A    F2 AF49   9A    0EF0  3097          MOVZBL  LENGTHS-1[R9],R10      ; R10 = data type length
                      57    D4  0EF5  3098          CLRL    R7                     ; clear the index value
            5B    50 AD   D0    0EF7  3099          MOVL    REG_PC(FP),R11         ; R11 = specifier byte location
      6B    01    FD AD   0C    0EFB  3100          PROBER  MODE(FP),#1,(R11)      ; can we read the specifier byte ?
                      06    12  0F00  3101          BNEQ    1$                     ; yes - skip
                5A    01    D0  0F02  3102          MOVL    #1,R10                 ; R10 = size of probe
                    0CA3    30  0F05  3103          BSBW    READ_FAULT             ; process an access violation
            50    50 BD   9A    0F08  3104 1$:      MOVZBL  @REG_PC(FP),R0         ; R0 = specifier byte
                  50 AD   D6    0F0C  3105          INCL    REG_PC(FP)             ; increment PC
                  FA AD   96    0F0F  3106          INCB    REGMOD_PC(FP)          ; remember the incrementation
      51    50    04    04    EF  0F12  3107          EXTZV   #4,#4,R0,R1            ; R1 = high order nibble of specifier
      52    50    04    00    EF  0F17  3108          EXTZV   #0,#4,R0,R2            ; R2 = low order nibble of specifier
```

```
                                                B 3
                     0F   00   51   CF   0F1C  3109                CASEL   R1,#0,#15                  ; branch on the high order nibble
                                0020' 0F20  3110 2$:               .WORD   LITERAL_MODE-2$             ; 0 - literal mode
                                0020' 0F22  3111                   .WORD   LITERAL_MODE-2$             ; 1 - literal mode
                                0020' 0F24  3112                   .WORD   LITERAL_MODE-2$             ; 2 - literal mode
                                0020' 0F26  3113                   .WORD   LITERAL_MODE-2$             ; 3 - literal mode
                                0061' 0F28  3114                   .WORD   INDEX_MODE-2$               ; 4 - index mode
                                00CD' 0F2A  3115                   .WORD   REGISTER_MODE-2$            ; 5 - register mode
                                00EF' 0F2C  3116                   .WORD   REG_DEF_MODE-2$             ; 6 - register deferred mode
                                0100' 0F2E  3117                   .WORD   DECR_MODE-2$                ; 7 - autodecrement mode
                                011B' 0F30  3118                   .WORD   INCR_MODE-2$                ; 8 - autoincrement mode
                                013F' 0F32  3119                   .WORD   INCR_DEF_MODE-2$            ; 9 - autoincrement deferred mode
                                0162' 0F34  3120                   .WORD   BYTE_DISP_MODE-2$           ; A - byte displacement mode
                                0187' 0F36  3121                   .WORD   BYTE_DEF_MODE-2$            ; B - byte displacement deferred mode
                                01BA' 0F38  3122                   .WORD   WORD_DISP_MODE-2$           ; C - word displacement mode
                                01E1' 0F3A  3123                   .WORD   WORD_DEF_MODE-2$            ; D - word displacement deferred mode
                                0216' 0F3C  3124                   .WORD   LONG_DISP_MODE-2$           ; E - long displacement mode
                                023D' 0F3E  3125                   .WORD   LONG_DEF_MODE-2$            ; F - long displacement deferred mode
                                      0F40  3126                   ;
                                      0F40  3127                   ;       Process a Literal Mode Operand Specifier
                                      0F40  3128                   ;
                                      0F40  3129 LITERAL_MODE:                                         ; entrance
                     03   01   58   CF   0F40  3130                CASEL   R8,#1,#3                    ; branch on the access type
                                0008' 0F44  3131 1$:               .WORD   2$-1$                       ; 1 - read only access
                                0D31' 0F46  3132                   .WORD   ADDRESS_FAULT-1$            ; 2 - write only access
                                0D31' 0F48  3133                   .WORD   ADDRESS_FAULT-1$            ; 3 - modify access
                                0D31' 0F4A  3134                   .WORD   ADDRESS_FAULT-1$            ; 4 - address access
                     08   01   59   CF   0F4C  3135 2$:            CASEL   R9,#1,#8                    ; branch on the data type
                                0016' 0F50  3136 3$:               .WORD   6$-3$                       ; 1 - byte
                                0016' 0F52  3137                   .WORD   6$-3$                       ; 2 - word
                                0016' 0F54  3138                   .WORD   6$-3$                       ; 3 - longword
                                0014' 0F56  3139                   .WORD   5$-3$                       ; 4 - quadword
                                0012' 0F58  3140                   .WORD   4$-3$                       ; 5 - octaword
                                0019' 0F5A  3141                   .WORD   8$-3$                       ; 6 - F_floating
                                0017' 0F5C  3142                   .WORD   7$-3$                       ; 7 - D_floating
                                0021' 0F5E  3143                   .WORD   9$-3$                       ; 8 - G_floating
                                0029' 0F60  3144                   .WORD   10$-3$                      ; 9 - H_floating
                           52   7C   0F62  3145 4$:                 CLRQ    R2                          ; clear second quadword of value
                           51   D4   0F64  3146 5$:                 CLRL    R1                          ; clear second longword of value
                                05   0F66  3147 6$:                 RSB                                 ; return with the literal value
                           51   D4   0F67  3148 7$:                 CLRL    R1                          ; clear second longword of value
                     50    50   04   78   0F69  3149 8$:           ASHL    #4,R0,R0                    ; position the literal bits
                       F5  50   0E   E3   0F6D  3150                BBCS    #14,R0,6$                   ; include exponent bias and return
                     50    50   01   78   0F71  3151 9$:           ASHL    #1,R0,R0                    ; position the literal bits
                       EB  50   0E   E3   0F75  3152                BBCS    #14,R0,5$                   ; include exponent bias and finish up
                     50    50   1D   9C   0F79  3153 10$:          ROTL    #29,R0,R0                   ; position the literal bits
                       E1  50   0E   E3   0F7D  3154                BBCS    #14,R0,4$                   ; include exponent bias and finish up
                                      0F81  3155                   ;
                                      0F81  3156                   ;       Process an Index Mode Operand Specifier
                                      0F81  3157                   ;
                                      0F81  3158 INDEX_MODE:                                           ; entrance
                           52   0F   D1   0F81  3159                CMPL    #15,R2                      ; is the register PC ?
                                03   12   0F84  3160                BNEQ    1$                          ; no - skip
                                0CEC 31   0F86  3161                BRW     ADDRESS_FAULT               ; process the reserved addressing mode
                  57   14 AD42    5A   C5   0F89  3162 1$:         MULL3   R10,REG_R0(FP)[R2],R7       ; R7 = index address modification
                           53   52   D0   0F8F  3163                MOVL    R2,R3                       ; save the register number
                        5B    50 AD   D0   0F92  3164                MOVL    REG_PC(FP),R11              ; R11 = location of next byte
                     6B   01    FD AD   0C   0F96  3165                PROBER  MODE(FP),#1,(R11)           ; can we read the next byte ?
```

```
                     06   12   0F9B  3166          BNEQ    2$                      ; yes - skip
                  5A 01   D0   0F9D  3167          MOVL    #1,R10                  ; R10 = size of probe
                  0C08    30   0FA0  3168          BSBW    READ_FAULT              ; process the access violation
               50 50 BD   9A   0FA3  3169 2$:      MOVZBL  @REG_PC(FP),R0          ; R0 = next operand specifier
                  50 AD   D6   0FA7  3170          INCL    REG_PC(FP)              ; increment PC
                  FA AD   96   0FAA  3171          INCB    REGMOD_PC(FP)           ; remember the incrementation
            51 50 04 04   EF   0FAD  3172          EXTZV   #4,#4,R0,R1             ; R1 = high order nibble of specifier
            52 50 04 00   EF   0FB2  3173          EXTZV   #0,#4,R0,R2             ; R2 = low order nibble of specifier
            0F 00 51      CF   0FB7  3174          CASEL   R1,#0,#15               ; branch on the low order nibble
                  0CBA' 0FBB  3175 3$:      .WORD   ADDRESS_FAULT-3$        ; 0 - literal mode
                  0CBA' 0FBD  3176          .WORD   ADDRESS_FAULT-3$        ; 1 - literal mode
                  0CBA' 0FBF  3177          .WORD   ADDRESS_FAULT-3$        ; 2 - literal mode
                  0CBA' 0FC1  3178          .WORD   ADDRESS_FAULT-3$        ; 3 - literal mode
                  0CBA' 0FC3  3179          .WORD   ADDRESS_FAULT-3$        ; 4 - index mode
                  0CBA' 0FC5  3180          .WORD   ADDRESS_FAULT-3$        ; 5 - register mode
                  0054' 0FC7  3181          .WORD   REG_DEF_MODE-3$         ; 6 - register deferred mode
                  0020' 0FC9  3182          .WORD   4$-3$                   ; 7 - autodecrement mode
                  0020' 0FCB  3183          .WORD   4$-3$                   ; 8 - autoincrement mode
                  0020' 0FCD  3184          .WORD   4$-3$                   ; 9 - autoincrement deferred mode
                  00C7' 0FCF  3185          .WORD   BYTE_DISP_MODE-3$       ; A - byte displacement mode
                  00EC' 0FD1  3186          .WORD   BYTE_DEF_MODE-3$        ; B - byte displacement deferred mode
                  011F' 0FD3  3187          .WORD   WORD_DISP_MODE-3$       ; C - word displacement mode
                  0146' 0FD5  3188          .WORD   WORD_DEF_MODE-3$        ; D - word displacement deferred mode
                  017B' 0FD7  3189          .WORD   LONG_DISP_MODE-3$       ; E - long displacement mode
                  01A2' 0FD9  3190          .WORD   LONG_DEF_MODE-3$        ; F - long displacement deferred mode
                  53 52 D1    0FDB  3191 4$:      CMPL    R2,R3                   ; is register the same as index ?
                     03 12    0FDE  3192          BNEQ    5$                      ; no - skip
                     0C92 31  0FE0  3193          BRW     ADDRESS_FAULT           ; process the reserved addressing mode
                  02 07 51    CF   0FE3  3194 5$:      CASEL   R1,#7,#2                ; branch on the high order nibble
                  0039' 0FE7  3195 6$:      .WORD   DECR_MODE-6$            ; 7 - autodecrement mode
                  0054' 0FE9  3196          .WORD   INCR_MODE-6$            ; 8 - autoincrement mode
                  0078' 0FEB  3197          .WORD   INCR_DEF_MODE-6$        ; 9 - autoincrement deferred mode
                        0FED  3198          ;
                        0FED  3199          ;       Process a Register Mode Operand Specifier
                        0FED  3200          ;
                        0FED  3201 REGISTER_MODE:                           ; entrance
                FC AD   02    88    0FED  3202          BISB2   #FLAG1M,FLAGS(FP)       ; indicate a register mode operand
                  53 6A42     DE    0FF1  3203          MOVAL   (R10)[R2],R3            ; byte position following operand
                     53 3C    D1    0FF5  3204          CMPL    #60,R3                  ; does the operand overlap PC ?
                        03    18    0FF8  3205          BGEQ    1$                      ; no - skip
                     0C78 31  0FFA  3206          BRW     ADDRESS_FAULT           ; process the reserved addressing mode
               5B 14 AD42    DE    0FFD  3207 1$:      MOVAL   REG_R0(FP)[R2],R11      ; R11 = location of user register
               03 01 58      CF    1002  3208          CASEL   R8,#1,#3                ; branch on the access type
                  01BC' 1006  3209 2$:      .WORD   READ_VALUE-2$           ; 1 - read only access
                  0008' 1008  3210          .WORD   3$-2$                   ; 2 - write access
                  01BC' 100A  3211          .WORD   READ_VALUE-2$           ; 3 - modify access
                  0C6F' 100C  3212          .WORD   ADDRESS_FAULT-2$        ; 4 - address access
                     05 100E  3213 3$:      RSB                             ; return with the register address
                        100F  3214          ;
                        100F  3215          ;       Process a Register Deferred Mode Operand Specifier
                        100F  3216          ;
                        100F  3217 REG_DEF_MODE:                            ; entrance
                  52 0F D1    100F  3218          CMPL    #15,R2                  ; is the register PC ?
                     03 14    1012  3219          BGTR    1$                      ; no - skip
                     0C5E 31  1014  3220          BRW     ADDRESS_FAULT           ; process the reserved addressing mode
            5B 14 AD42 57 C1    1017  3221 1$:      ADDL3   R7,REG_R0(FP)[R2],R11   ; form the operand address
                     0172 31    101D  3222          BRW     ACCESS_VALUE            ; finish establishing the access
```

```
                        1020  3223          ;
                        1020  3224          ;       Process an Autodecrement Mode Operand Specifier
                        1020  3225          ;
                        1020  3226 DECR_MODE:                                    ; entrance
                52   0F   D1  1020  3227          CMPL    #15,R2                 ; is the register PC ?
                     03   14  1023  3228          BGTR    1$                     ; no - skip
                   0C4D   31  1025  3229          BRW     ADDRESS_FAULT          ; process the reserved addressing mode
             14 AD42    5A   C2  1028  3230 1$:    SUBL2   R10,REG_R0(FP)[R2]     ; subtract data size from register
             EB AD42    5A   82  102D  3231          SUBB2   R10,REGMOD_R0(FP)[R2]  ; remember the subtraction
          5B 14 AD42    57   C1  1032  3232          ADDL3   R7,REG_R0(FP)[R2],R11  ; form the operand address
                   0157   31   1038  3233          BRW     ACCESS_VALUE           ; finish establishing the access
                        103B  3234          ;
                        103B  3235          ;       Process an Autoincrement Mode Operand Specifier
                        103B  3236          ;
                        103B  3237 INCR_MODE:                                    ; entrance
                52   0F   D1  103B  3238          CMPL    #15,R2                 ; is the register PC ?
                     0C   14  103E  3239          BGTR    2$                     ; no - bypass
             03   01   58   CF  1040  3240          CASEL   R8,#1,#3               ; branch on the access type
                   0008' 1044  3241 1$:    .WORD   2$-1$                  ; 1 - read only access
                   0C31' 1046  3242          .WORD   ADDRESS_FAULT-1$       ; 2 - write only access
                   0C31' 1048  3243          .WORD   ADDRESS_FAULT-1$       ; 3 - modify access
                   0008' 104A  3244          .WORD   2$-1$                  ; 4 - address access
          5B 14 AD42    57   C1  104C  3245 2$:    ADDL3   R7,REG_R0(FP)[R2],R11  ; form the operand address
             14 AD42    5A   C0  1052  3246          ADDL2   R10,REG_R0(FP)[R2]     ; add the data size to the register
             EB AD42    5A   80  1057  3247          ADDB2   R10,REGMOD_R0(FP)[R2]  ; remember the addition
                   0133   31   105C  3248          BRW     ACCESS_VALUE           ; finish establishing the access
                        105F  3249          ;
                        105F  3250          ;       Process an Autoincrement Deferred Mode Operand Specifier
                        105F  3251          ;
                        105F  3252 INCR_DEF_MODE:                                ; entrance
          5B 14 AD42    D0   105F  3253          MOVL    REG_R0(FP)[R2],R11     ; R11 = register value
       6B   04   FD AD    0C  1064  3254          PROBER  MODE(FP),#4,(R11)      ; can we read longword it addresses ?
                     06   12  1069  3255          BNEQ    1$                     ; yes - skip
                5A   04   D0  106B  3256          MOVL    #4,R10                 ; R10 = size of probe
                   0B3A   30  106E  3257          BSBW    READ_FAULT             ; process the access violation
          5B   6B   57   C1  1071  3258 1$:    ADDL3   R7,(R11),R11           ; form the operand address
          14 AD42   04   C0  1075  3259          ADDL2   #4,REG_R0(FP)[R2]      ; add longword size to the register
          EB AD42   04   80  107A  3260          ADDB2   #4,REGMOD_R0(FP)[R2]   ; remember the incrementation
                   0110   31  107F  3261          BRW     ACCESS_VALUE           ; finish establishing the access
                        1082  3262          ;
                        1082  3263          ;       Process a Byte Displacement Mode Operand Specifier
                        1082  3264          ;
                        1082  3265 BYTE_DISP_MODE:                               ; entrance
             5B   50 AD    D0  1082  3266          MOVL    REG_PC(FP),R11         ; R11 = location of displacement
       6B   01   FD AD    0C  1086  3267          PROBER  MODE(FP),#1,(R11)      ; can we read the displacement ?
                     06   12  108B  3268          BNEQ    1$                     ; yes - skip
                5A   01   D0  108D  3269          MOVL    #1,R10                 ; R10 = size of probe
                   0B18   30  1090  3270          BSBW    READ_FAULT             ; process the access violation
                5B   6B   98  1093  3271 1$:    CVTBL   (R11),R11              ; R11 = displacement value
                  50 AD   D6  1096  3272          INCL    REG_PC(FP)             ; increment PC
                  FA AD   D6  1099  3273          INCL    REGMOD_PC(FP)          ; remember the increment
                5B   57   C0  109C  3274          ADDL2   R7,R11                 ; add the displacement to the index
          5B   14 AD42    C0  109F  3275          ADDL2   REG_R0(FP)[R2],R11     ; add the register to the result
                   00EB   31  10A4  3276          BRW     ACCESS_VALUE           ; finish establishing the access
                        10A7  3277          ;
                        10A7  3278          ;       Process a Byte Displacement Deferred Mode Operand Specifier
                        10A7  3279          ;
```

```
                        10A7  3280 BYTE_DEF_MODE:                                ; entrance
      5B   50 AD   D0   10A7  3281          MOVL    REG_PC(FP),R11              ; R11 = location of displacement
6B    01   FD AD   0C   10AB  3282          PROBER  MODE(FP),#1,(R11)           ; can we read the displacement ?
             06   12   10B0  3283          BNEQ    1$                          ; yes - skip
         5A  01   D0   10B2  3284          MOVL    #1,R10                      ; R10 = size of probe
           0AF3   30   10B5  3285          BSBW    READ_FAULT                  ; process the access violation
         5B  6B   98   10B8  3286 1$:      CVTBL   (R11),R11                   ; R11 = displacement value
           50 AD  D6   10BB  3287          INCL    REG_PC(FP)                  ; increment PC
           FA AD  96   10BE  3288          INCB    REGMOD_PC(FP)               ; remember the increment
   5B   14 AD42   C0   10C1  3289          ADDL2   REG_R0(FP)[R2],R11          ; add the register to the displacement
6B   04   FD AD   0C   10C6  3290          PROBER  MODE(FP),#4,(R11)           ; can we read longword it addresses ?
             06   12   10CB  3291          BNEQ    2$                          ; yes - skip
         5A  04   D0   10CD  3292          MOVL    #4,R10                      ; R10 = size of probe
           0AD8   30   10D0  3293          BSBW    READ_FAULT                  ; process the access fault
   5B   6B   57   C1   10D3  3294 2$:      ADDL3   R7,(R11),R11                ; form the operand address
           00B8   31   10D7  3295          BRW     ACCESS_VALUE                ; finish establishing the access
                       10DA  3296                  ;
                       10DA  3297                  ;       Process a Word Displacement Mode Operand Specifier
                       10DA  3298                  ;
                       10DA  3299 WORD_DISP_MODE:                              ; entrance
      5B   50 AD   D0   10DA  3300          MOVL    REG_PC(FP),R11              ; R11 = location of the displacement
6B    02   FD AD   0C   10DE  3301          PROBER  MODE(FP),#2,(R11)           ; can we read the displacement
             06   12   10E3  3302          BNEQ    1$                          ; yes - skip
         5A  02   D0   10E5  3303          MOVL    #2,R10                      ; R10 = size of probe
           0AC0   30   10E8  3304          BSBW    READ_FAULT                  ; process the access violation
         5B  6B   32   10EB  3305 1$:      CVTWL   (R11),R11                   ; R11 = displacement value
      50 AD   02   C0   10EE  3306          ADDL2   #2,REG_PC(FP)               ; increment PC
      FA AD   02   80   10F2  3307          ADDB2   #2,REGMOD_PC(FP)            ; remember the increment
         5B  57   C0   10F6  3308          ADDL2   R7,R11                      ; add the index to the displacement
   5B   14 AD42   C0   10F9  3309          ADDL2   REG_R0(FP)[R2],R11          ; add the register to the result
           0091   31   10FE  3310          BRW     ACCESS_VALUE                ; finish establishing the access
                       1101  3311                  ;
                       1101  3312                  ;       Process a Word Displacement Deferred Mode Operand Specifier
                       1101  3313                  ;
                       1101  3314 WORD_DEF_MODE:                               ; entrance
      5B   50 AD   D0   1101  3315          MOVL    REG_PC(FP),R11              ; R11 = location of the displacement
6B    02   FD AD   0C   1105  3316          PROBER  MODE(FP),#2,(R11)           ; can we read the displacement ?
             06   12   110A  3317          BNEQ    1$                          ; yes - skip
         5A  02   D0   110C  3318          MOVL    #2,R10                      ; R10 = size of probe
           0A99   30   110F  3319          BSBW    READ_FAULT                  ; process the access violation
         5B  6B   32   1112  3320 1$:      CVTWL   (R11),R11                   ; R11 = displacement value
      50 AD   02   C0   1115  3321          ADDL2   #2,REG_PC(FP)               ; increment PC
      FA AD   02   80   1119  3322          ADDB2   #2,REGMOD_PC(FP)            ; remember the increment
   5B   14 AD42   C0   111D  3323          ADDL2   REG_R0(FP)[R2],R11          ; add the register to the displacement
6B   04   FD AD   0C   1122  3324          PROBER  MODE(FP),#4,(R11)           ; can we read longword it addresses ?
             06   12   1127  3325          BNEQ    2$                          ; yes - skip
         5A  04   D0   1129  3326          MOVL    #4,R10                      ; R10 = size of probe
           0A7C   30   112C  3327          BSBW    READ_FAULT                  ; process the access violation
   5B   6B   57   C1   112F  3328 2$:      ADDL3   R7,(R11),R11                ; form the operand address
           005C   31   1133  3329          BRW     ACCESS_VALUE                ; finish establishing the access
                       1136  3330                  ;
                       1136  3331                  ;       Process a Long Displacement Mode Operand Specifier
                       1136  3332                  ;
                       1136  3333 LONG_DISP_MODE:                              ; entrance
      5B   50 AD   D0   1136  3334          MOVL    REG_PC(FP),R11              ; R11 = location of the displacement
6B    04   FD AD   0C   113A  3335          PROBER  MODE(FP),#4,(R11)           ; can we read the displacement ?
             06   12   113F  3336          BNEQ    1$                          ; yes - skip
```

```
                5A    04    D0  1141  3337          MOVL    #4,R10                  ; R10 = size of probe
                    0A64    30  1144  3338          BSBW    READ_FAULT              ; process the access violation
                5B    6B    D0  1147  3339 1$:      MOVL    (R11),R11               ; R11 = displacement value
              50 AD    04    C0  114A  3340          ADDL2   #4,REG_PC(FP)           ; increment PC
              FA AD    04    80  114E  3341          ADDB2   #4,REGMOD_PC(FP)        ; remember the increment
                5B    57    C0  1152  3342          ADDL2   R7,R11                  ; add the index to the displacement
           5B   14 AD42    C0  1155  3343          ADDL2   REG_R0(FP)[R2],R11      ; add the register to the address
                0035    31  115A  3344          BRW     ACCESS_VALUE            ; finish establishing the access
                            115D  3345          ;
                            115D  3346          ;       Process a Long Displacement Deferred Mode Operand Specifier
                            115D  3347          ;
                            115D  3348 LONG_DEF_MODE:                           ; entrance
                5B    50 AD   D0  115D  3349          MOVL    REG_PC(FP),R11          ; R11 = location of the displacement
           6B   04    FD AD   0C  1161  3350          PROBER  MODE(FP),#4,(R11)       ; can we read the displacement ?
                      06    12  1166  3351          BNEQ    1$                      ; yes - skip
                5A    04    D0  1168  3352          MOVL    #4,R10                  ; R10 = size of probe
                    0A3D    30  116B  3353          BSBW    READ_FAULT              ; process the access violation
                5B    6B    D0  116E  3354 1$:      MOVL    (R11),R11               ; R11 = displacement value
              50 AD    04    C0  1171  3355          ADDL2   #4,REG_PC(FP)           ; increment PC
              FA AD    04    80  1175  3356          ADDB2   #4,REGMOD_PC(FP)        ; remember the increment
           5B   14 AD42    C0  1179  3357          ADDL2   REG_R0(FP)[R2],R11      ; add the register to the displacement
           6B   04    FD AD   0C  117E  3358          PROBER  MODE(FP),#4,(R11)       ; can we read longword it addresses ?
                      06    12  1183  3359          BNEQ    2$                      ; yes - skip
                5A    04    D0  1185  3360          MOVL    #4,R10                  ; R10 = size of probe
                    0A20    30  1188  3361          BSBW    READ_FAULT              ; process the access violation
           5B   6B    57    C1  118B  3362 2$:      ADDL3   R7,(R11),R11            ; form the operand address
                0000    31  118F  3363          BRW     ACCESS_VALUE            ; finish establishing the access
                            1192  3364          ;
                            1192  3365          ;       Set Up the Type of Access Requested
                            1192  3366          ;
                            1192  3367 ACCESS_VALUE:                           ; entrance
     03    01    58    CF  1192  3368          CASEL   R8,#1,#3                ; branch on the access type
                0000D' 1196  3369 1$:      .WORD   READ_CHECK-1$           ; 1 - read only access
                0019' 1198  3370          .WORD   WRITE_CHECK-1$          ; 2 - write only access
                0009' 119A  3371          .WORD   MODIFY_CHECK-1$         ; 3 - modify access
                0008' 119C  3372          .WORD   2$-1$                   ; 4 - address access
                  05  119E  3373 2$:      RSB                             ; return with the operand address
                      119F  3374          ;
                      119F  3375          ;       Perform Error Checking for Modify Access Operands
                      119F  3376          ;
                      119F  3377 MODIFY_CHECK:                            ; entrance
                   0E   10  119F  3378          BSBB    WRITE_CHECK             ; check write (and read) access
                   1F   11  11A1  3379          BRB     READ_VALUE              ; load the value
                      11A3  3380          ;
                      11A3  3381          ;       Perform Error Checking for Read Only Access Operands
                      11A3  3382          ;
                      11A3  3383 READ_CHECK:                              ; entrance
     6B    5A    FD AD   0C  11A3  3384          PROBER  MODE(FP),R10,(R11)      ; can we read the operand ?
                   03   12  11A8  3385          BNEQ    1$                      ; yes - load the value
                 09FE   30  11AA  3386          BSBW    READ_FAULT              ; process the access violation
                   13   11  11AD  3387 1$:      BRB     READ_VALUE              ; load the value
                      11AF  3388          ;
                      11AF  3389          ;       Perform Error Checking for Write Only Access Operands
                      11AF  3390          ;
                      11AF  3391 WRITE_CHECK:                             ; entrance
     6B    5A    FD AD   0D  11AF  3392          PROBEW  MODE(FP),R10,(R11)      ; can we write the operand ?
                   0B   12  11B4  3393          BNEQ    1$                      ; yes - bypass
```

```
                    0A63   30  11B6  3394          BSBW    WRITE_FAULT                     ; process the access violation
          03 FC AD    00   E0  11B9  3395          BBS     #FLAGO,FLAGS(FP),1$             ; no local store checking - skip
                      004C 30  11BE  3396          BSBW    LOCAL_TEST                      ; test for a write into local storage
                           05  11C1  3397 1$:       RSB                                     ; return
                               11C2  3398          ;
                               11C2  3399          ;       Load a Read Operand into the Registers
                               11C2  3400          ;
                               11C2  3401 READ_VALUE:                                      ; entrance
                08    01   59   CF  11C2  3402          CASEL   R9,#1,#8                    ; branch on the data type
                           0012' 11C6  3403 1$:       .WORD   2$-1$                       ; 1 - byte
                           0016' 11C8  3404           .WORD   3$-1$                       ; 2 - word
                           001A' 11CA  3405           .WORD   4$-1$                       ; 3 - longword
                           0022' 11CC  3406           .WORD   6$-1$                       ; 4 - quadword
                           001E' 11CE  3407           .WORD   5$-1$                       ; 5 - octaword
                           001A' 11D0  3408           .WORD   4$-1$                       ; 6 - F_floating
                           0022' 11D2  3409           .WORD   6$-1$                       ; 7 - D_floating
                           0022' 11D4  3410           .WORD   6$-1$                       ; 8 - G_floating
                           001E' 11D6  3411           .WORD   5$-1$                       ; 9 - H_floating
                      50   6B   98  11D8  3412 2$:       CVTBL   (R11),R0                ; R0 = operand value
                           05   11DB  3413           RSB                                 ; return
                      50   6B   32  11DC  3414 3$:       CVTWL   (R11),R0                ; R0 = operand value
                           05   11DF  3415           RSB                                 ; return
                      50   6B   D0  11E0  3416 4$:       MOVL    (R11),R0                ; R0 = operand value
                           05   11E3  3417           RSB                                 ; return
                52    08 AB   7D  11E4  3418 5$:       MOVQ    8(R11),R2               ; R2,R3 = high order quadword of value
                      50   6B   7D  11E8  3419 6$:       MOVQ    (R11),R0                ; R0,R1 = low order quadword of value
                           05   11EB  3420           RSB                                 ; return
                               11EC  3421          ;
                               11EC  3422          ;       Process a Word Branch Displacement Operand
                               11EC  3423          ;
                               11EC  3424 BRANCH_WORD:                                     ; entrance
                5B    50 AD   D0  11EC  3425          MOVL    REG_PC(FP),R11              ; R11 = location of the displacement
          6B    02    FD AD   0C  11F0  3426          PROBER  MODE(FP),#2,(R11)           ; can we read the displacement ?
                         06   12  11F5  3427          BNEQ    1$                          ; yes - skip
                   5A    02   D0  11F7  3428          MOVL    #2,R10                      ; R10 = size of probe
                      09AE      30  11FA  3429          BSBW    READ_FAULT                  ; process the access violation
                5B     6B      32  11FD  3430 1$:       CVTWL   (R11),R11               ; R11 = branch displacement
                50 AD    02    C0  1200  3431          ADDL2   #2,REG_PC(FP)               ; increment PC
                FA AD    02    80  1204  3432          ADDB2   #2,REGMOD_PC(FP)            ; remember the increment
                5B    50 AD    C0  1208  3433          ADDL2   REG_PC(FP),R11              ; compute the branch destination
                           05   120C  3434          RSB                                 ; return
                               120D  3435          ;
                               120D  3436          ;       Test for a Write into Local Storage
                               120D  3437          ;
                               120D  3438          ;               entered by subroutine branching
                               120D  3439          ;
                               120D  3440          ;               parameters:     R10 = Number of Bytes to be Written
                               120D  3441          ;                               R11 = Distination Address
                               120D  3442          ;
                               120D  3443          ;               returns with    R11 = Corrected Destination Address
                               120D  3444          ;
                               120D  3445          ;       Discussion
                               120D  3446          ;
                               120D  3447          ;           This routine checks the write operation described by
                               120D  3448          ;       the parameters in R10 and R11 for a write into the Emulator's
                               120D  3449          ;       working storage. If such a write is about to take place, R11
                               120D  3450          ;       is changed to an address where the write will not do any harm.
```

```
                                                H 3
                        120D  3451                ;
                        120D  3452 LOCAL_TEST:
                        120D  3453          MOVAB   LOCAL_END(FP),R3        ; entrance
                                                                            ; R3 = byte following local storage
                        1211  3454          CMPL    R11,R3                  ; is the write above the frame ?
                        1214  3455          BGEQU   1$                      ; yes - bypass
                        1216  3456          ADDL3   R10,R11,R3              ; R3 = byte following operand
                        121A  3457          CMPL    R3,SP                   ; is it above the stack pointer ?
                        121D  3458          BLEQU   1$                      ; no - operand is not in local storage
                        121F  3459          MOVAB   TEMP(FP),R11            ; redirect the write to TEMP
                        1223  3460 1$:      RSB                             ; return with the operand address
                        1224  3461          ;
```

```
;*****************************************************************************
;*                                                                           *
;*                                                                           *
;*      Routines for Unpacking and Packing Floating Values                   *
;*                                                                           *
;*                                                                           *
;*****************************************************************************

;Introduction
;------------

;    The following routines perform all of the conversions
;between the VAX floating representations and the internal
;representation used by the Emulator. The unpack routines
;convert from the VAX representation to the internal
;representation and the pack routines perform the opposite
;conversion. These routines perform all of the necessary
;rounding and check for reserved values, underflow, and
;overflow.


;The Unpack Routines
;-------------------

;    The unpack routines convert a value in one of the VAX
;floating representations to our internal representation.
;The value to be converted is assumed to be contained in
;the registers starting at R0. For floating and double
;floating values the available unpack routines only place
;the converted value in OPERAND1. For GFLOAT and HFLOAT routines
;are available which place the result in all of the operand
;areas.

;    The unpack routines all check for a reserved floating
;value (sign bit set and biased exponent equal to zero) and
;signal a reserved operand exception if one is found.


;The Pack Routines
;-----------------

;    The pack routines convert a value in the internal
;representation to one of the VAX floating representations.
;The value to be converted is assumed to be in one of the
;operand areas and the value must be in OPERAND1 if the value
;is to be converted to floating or double floating. For GFLOAT
;and HFLOAT routines are available to convert from each of the
;operand areas. The routines always leave the result in the
;registers starting at R0.

;    Before the value is converted the rounding bit (the first
;bit of the fraction which will not appear in the converted
;result) is tested and if it is set the value is rounded by
;adding one to the next higher bit (the lowest one that will
;appear in the converted result) and processing any carries
;that occur. When the conversion is performed the biased
```

```
1224  3520          ;       exponent is checked for possible overflow or underflow. If an
1224  3521          ;       overflow condition is detected, then the condition is
1224  3522          ;       signaled. If an underflow condition is detected, then the
1224  3523          ;       condition is signaled only if the FU bit is set in the user's
1224  3524          ;       PSL. If the bit is not set then a value of zero is returned.
1224  3525          ;       If a nonzero value is converted to zero because of underflow,
1224  3526          ;       the source value in operand area will be marked as zero so
1224  3527          ;       condition code determination will still work properly.
1224  3528          ;
```

```
                                1224  3530          ;
                                1224  3531          ;       UNPACK_FLOAT1 - Unpack a floating value to OPERAND1
                                1224  3532          ;
                                1224  3533 UNPACK_FLOAT1:
        54    C7 AD    9E       1224  3534          MOVAB   OPERAND1(FP), R4        ; operand area address
                 0A    11       1228  3535          BRB     UNPACK_FLOAT
                                122A  3536
                                122A  3537          ;
                                122A  3538          ;       UNPACK_FLOAT2 - Unpack a floating value to OPERAND2
                                122A  3539          ;
                                122A  3540 UNPACK_FLOAT2:
        54    AF AD    9E       122A  3541          MOVAB   OPERAND2(FP), R4        ; operand area address
                 04    11       122E  3542          BRB     UNPACK_FLOAT
                                1230  3543
                                1230  3544          ;
                                1230  3545          ;       UNPACK_FLOAT3 - Unpack a floating value to OPERAND3
                                1230  3546          ;
                                1230  3547 UNPACK_FLOAT3:
        54    97 AD    9E       1230  3548          MOVAB   OPERAND3(FP), R4        ; operand area address
                                1234  3549          ;BRB    UNPACK_FLOAT
                                1234  3550
                                1234  3551 UNPACK_FLOAT:
     03 06    8B BD    8F       1234  3552          CASEB   @OP_INDEX(FP),#TYP_F,#<TYP_H-TYP_F>
                      0008'     1239  3553 1$:       .WORD   UNPACK_FFLOAT-1$        ; 6 - F_floating
                      000A'     123B  3554          .WORD   UNPACK_DFLOAT-1$        ; 7 - D_floating
                      0046'     123D  3555          .WORD   UNPACK_GFLOAT-1$        ; 8 - G_floating
                      0088'     123F  3556          .WORD   UNPACK_HFLOAT-1$        ; 9 - H_floating
```

```
                     1241  3558          ;
                     1241  3559          ;       UNPACK_FFLOAT - Unpack a Floating Value
                     1241  3560          ;
                     1241  3561          ;               entered by subroutine branching
                     1241  3562          ;
                     1241  3563          ;               parameter:      R0 = Input Floating Value
                     1241  3564          ;
                     1241  3565          ;               returns with    (R4) = Converted Value
                     1241  3566          ;
                     1241  3567          ;               uses R0-R5
                     1241  3568          ;
                     1241  3569 UNPACK_FFLOAT:                              ; entrance
51   D4              1241  3570          CLRL    R1                          ; make F look like D
                     1243  3571          ;BRB    UNPACK_DFLOAT               ; use D_floating unpack
                     1243  3572          ;
```

```
                                1243  3574          ;
                                1243  3575          ;       UNPACK_DFLOAT - Unpack a Double Floating Value
                                1243  3576          ;
                                1243  3577          ;               entered by subroutine branching
                                1243  3578          ;
                                1243  3579          ;               parameter:      R0,R1 = Input Double Floating Value
                                1243  3580          ;
                                1243  3581          ;               returns with    (R4) = Converted Value
                                1243  3582          ;
                                1243  3583          ;               uses R0-R5
                                1243  3584          ;
                                1243  3585 UNPACK_DFLOAT:                            ; entrance
                          64    1243  3586          CLRL    (R4)                    ; clear the zero and sign flags
     55   50   08    07    EF  1245  3587          EXTZV   #7,#8,R0,R5             ; R5 = biased exponent
                     0A    12  124A  3588          BNEQ    2$                      ; it's not zero - bypass
          03 50    0F    E1  124C  3589          BBC     #15,R0,1$               ; no sign bit - skip
                 0A33    31  1250  3590          BRW     OPERAND_FAULT           ; double floating value is reserved
                   64    96  1253  3591 1$:     INCB    ZERO(R4)                ; indicate a zero value
                         05  1255  3592          RSB                             ; return
          03 50    0F    E1  1256  3593 2$:     BBC     #15,R0,3$               ; no sign bit - skip
                C1 A4    96  125A  3594          INCB    SIGN(R4)                ; indicate a negative value
        04 A4    80 A5    9E  125D  3595 3$:     MOVAB   -128(R5),POWER(R4)      ; store the unbiased exponent
         50   50    10    9C  1262  3596          ROTL    #16,R0,R0               ; R0 = leading bits of the fraction
          00 50    17    E2  1266  3597          BBSS    #23,R0,4$               ; set the hidden bit
14 A4    18   08    50    F0  126A  3598 4$:     INSV    R0,#8,#24,FRACTION+12(R4); store bits 40-63 of the fraction
         51   51    10    9C  1270  3599          ROTL    #16,R1,R1               ; R1 = trailing bits of the fraction
           11 A4    51    D0  1274  3600          MOVL    R1,FRACTION+9(R4)       ; store bits 8-39 of the fraction
              10 A4  94    94  1278  3601          CLRB    FRACTION+8(R4)          ; clear bits 0-7 of the fraction
              08 A4  7C    7C  127B  3602          CLRQ    FRACTION(R4)            ; extend the fraction to 128 bits
                     05    05  127E  3603          RSB                             ; return
                           127F  3604          ;
```

```
                                127F  3606          ;
                                127F  3607          ;       UNPACK_GFLOAT- Unpack a G_floating Floating Value
                                127F  3608          ;
                                127F  3609          ;              entered by subroutine branching
                                127F  3610          ;
                                127F  3611          ;              parameter:      R0,R1 = Input G_floating Floating Value
                                127F  3612          ;
                                127F  3613          ;              returns with    (R4) = Converted Value
                                127F  3614          ;
                                127F  3615          ;              uses R0-R5
                                127F  3616          ;
                                127F  3617  UNPACK_GFLOAT:                          ; entrance
                          64    D4    127F  3618          CLRL    (R4)              ; clear the zero and sign flags
     55   50   0B      04    EF    1281  3619          EXTZV   #4,#11,R0,R5      ; R5 = biased exponent
                        0A    12    1286  3620          BNEQ    2$                ; it's not zero - bypass
          03 50       0F    E1    1288  3621          BBC     #15,R0,1$         ; no sign bit - skip
                    09F7    31    128C  3622          BRW     OPERAND_FAULT     ; G_floating floating value is reserved
                          64    96    128F  3623  1$:    INCB    ZERO(R4)          ; indicate a zero value
                          05    1291  3624          RSB                       ; return
          03 50       0F    E1    1292  3625  2$:    BBC     #15,R0,3$         ; no sign bit - skip
                01    A4    96    1296  3626          INCB    SIGN(R4)          ; indicate a negative value
     04   A4    FC00  C5    9E    1299  3627  3$:    MOVAB   -1024(R5),POWER(R4) ; store the unbiased exponent
          50    50    10    9C    129F  3628          ROTL    #16,R0,R0         ; R0 = leading bits of the fraction
          00 50       14    E2    12A3  3629          BBSS    #20,R0,4$         ; set the hidden bit
14   A4   15    0B    50    F0    12A7  3630  4$:    INSV    R0,#11,#21,FRACTION+12(R4); store bits 43-63 of the fraction
          51    51    10    9C    12AD  3631          ROTL    #16,R1,R1         ; R1 = trailing bits of the fraction
10   A4   20    0B    51    F0    12B1  3632          INSV    R1,#11,#32,FRACTION+8(R4); store bits 11-42 of the fraction
10   A4   0B    00    00    F0    12B7  3633          INSV    #0,#0,#11,FRACTION+8(R4); clear bits 0-10 of the fraction
                08    A4    7C    12BD  3634          CLRQ    FRACTION(R4)      ; extend the fraction to 128 bits
                          05    12C0  3635          RSB                       ; return
                                12C1  3636          ;
```

```
                                                B 4
                                12C1  3638          ;
                                12C1  3639          ;       UNPACK_HFLOAT - Unpack a HFLOAT Floating Value
                                12C1  3640          ;
                                12C1  3641          ;               entered by subroutine branching
                                12C1  3642          ;
                                12C1  3643          ;               parameter:      R0,R1,R2,R3 = Input HFLOAT Value
                                12C1  3644          ;
                                12C1  3645          ;               returns with    (R4) = Converted Value
                                12C1  3646          ;
                                12C1  3647          ;               uses R0-R5
                                12C1  3648          ;
                                12C1  3649          ;
                                12C1  3650 UNPACK_HFLOAT:                               ; entrance
                          64    D4    12C1  3651          CLRL    (R4)                  ; clear the zero and sign flags
     55   50   0F      00    EF    12C3  3652          EXTZV   #0,#15,R0,R5          ; R5 = biased exponent
                       0A    12    12C8  3653          BNEQ    2$                    ; it's not zero - bypass
               03 50   0F    E1    12CA  3654          BBC     #15,R0,1$             ; no sign bit - skip
                     09B5    31    12CE  3655          BRW     OPERAND_FAULT         ; HFLOAT floating value is reserved
                       64    96    12D1  3656 1$:      INCB    ZERO(R4)              ; indicate a zero value
                       05    12D3  3657          RSB                                 ; return
               03 50   0F    E1    12D4  3658 2$:      BBC     #15,R0,3$             ; no sign bit - skip
                     01 A4   96    12D8  3659          INCB    SIGN(R4)              ; indicate a negative value
          04 A4    C000    C5    9E    12DB  3660 3$:      MOVAB   -16384(R5),POWER(R4)  ; store the unbiased exponent
               50   50      10    9C    12E1  3661          ROTL    #16,R0,R0             ; R0 = leading bits of the fraction
                  00 50      10    E2    12E5  3662          BBSS    #16,R0,4$             ; set the hidden bit
     14 A4    11    0F      50    F0    12E9  3663 4$:      INSV    R0,#15,#17,FRACTION+12(R4); store bits 111-127 of the fraction
               51   51      10    9C    12EF  3664          ROTL    #16,R1,R1             ; R1 = next bits of the fraction
     10 A4    20    0F      51    F0    12F3  3665          INSV    R1,#15,#32,FRACTION+8(R4); store bits 79-110 of the fraction
               52   52      10    9C    12F9  3666          ROTL    #16,R2,R2             ; R2 = next bits of the fraction
     0C A4    20    0F      52    F0    12FD  3667          INSV    R2,#15,#32,FRACTION+4(R4); store bits 47-78 of the fraction
               53   53      10    9C    1303  3668          ROTL    #16,R3,R3             ; R3 = next bits of the fraction
     08 A4    20    0F      53    F0    1307  3669          INSV    R3,#15,#32,FRACTION(R4) ; store bits 15-46 of the fraction
     08 A4    0F    00      00    F0    130D  3670          INSV    #0,#0,#15,FRACTION(R4)  ; clear bits 0-14 of the fraction
                       05    1313  3671          RSB                                 ; return
                             1314  3672          ;
```

```
                                1314  3674          ;
                                1314  3675          ;       PACK_FLOAT1 - Pack a floating value to OPERAND1
                                1314  3676          ;
                                1314  3677  PACK_FLOAT1:
        54    C7 AD    9E       1314  3678          MOVAB   OPERAND1(FP), R4        ; operand area address
              0A       11       1318  3679          BRB     PACK_FLOAT
                                131A  3680
                                131A  3681          ;
                                131A  3682          ;       PACK_FLOAT2 - Pack a floating value to OPERAND2
                                131A  3683          ;
                                131A  3684  PACK_FLOAT2:
        54    AF AD    9E       131A  3685          MOVAB   OPERAND2(FP), R4        ; operand area address
              04       11       131E  3686          BRB     PACK_FLOAT
                                1320  3687
                                1320  3688          ;
                                1320  3689          ;       PACK_FLOAT3 - Pack a floating value to OPERAND3
                                1320  3690          ;
                                1320  3691  PACK_FLOAT3:
        54    97 AD    9E       1320  3692          MOVAB   OPERAND3(FP), R4        ; operand area address
                                1324  3693          ;BRB    PACK_FLOAT
                                1324  3694
                                1324  3695  PACK_FLOAT:
  03    06    8B BD    8F       1324  3696          CASEB   @OP_INDEX(FP),#TYP_F,#<TYP_H-TYP_F>
                    0008'       1329  3697  1$:     .WORD   PACK_FFLOAT-1$          ; 6 - F_floating
                    0059'       132B  3698          .WORD   PACK_DFLOAT-1$          ; 7 - D_floating
                    00B3'       132D  3699          .WORD   PACK_GFLOAT-1$          ; 8 - G_floating
                    0112'       132F  3700          .WORD   PACK_HFLOAT-1$          ; 9 - H_floating
                                1331  3701
```

```
                                1331  3703          ;
                                1331  3704          ;       PACK_FFLOAT - Pack an F_floating Value
                                1331  3705          ;
                                1331  3706          ;               entered by subroutine branching
                                1331  3707          ;
                                1331  3708          ;               parameter:      (R4) = Source Value
                                1331  3709          ;
                                1331  3710          ;               returns with    R0 = Converted Floating Value
                                1331  3711          ;
                                1331  3712  PACK_FFLOAT:                            ; entrance
                         03 64   E9     1331  3713          BLBC    ZERO(R4),2$     ; value is not zero - bypass
                            50   D4     1334  3714  1$:     CLRL    R0              ; clear the value
                                 05     1336  3715          RSB                     ; return
                   11 14 A4     07     E1     1337  3716  2$:     BBC     #7,FRACTION+12(R4),3$   ; rounding bit is zero - bypass
                14 A4   00000100 8F     C0     133C  3717          ADDL2   #100,FRACTION+12(R4)    ; round the value
                                 07     1E     1344  3718          BCC     3$              ; no carry out of fraction - bypass
                            04 A4     D6     1346  3719          INCL    POWER(R4)       ; increment the exponent
                                50     D4     1349  3720          CLRL    R0              ; clear the fraction bits
                                 05     11     134B  3721          BRB     4$              ; bypass
                      50    14 A4     08     9C     134D  3722  3$:     ROTL    #8,FRACTION+12(R4),R0   ; load the fraction
           51   04 A4   00000080 8F     C1     1352  3723  4$:     ADDL3   #128,POWER(R4),R1       ; R1 = biased exponent
                                 0B     14     135B  3724          BGTR    5$              ; it's greater than zero - bypass
                            64   01     D0     135D  3725          MOVL    #1,(R4)         ; mark the original value as zero
                      CF 54 AD   06     E1     1360  3726          BBC     #PSL_FU,PSL(FP),1$      ; no fault enabled - return zero
                             092F 31     1365  3727          BRW     UNDERFLOW       ; process the floating underflow
                   51   000000FF 8F     D1     1368  3728  5$:     CMPL    #255,R1         ; is the exponent too large ?
                                 03     18     136F  3729          BGEQ    6$              ; no - skip
                             0934 31     1371  3730          BRW     OVERFLOW        ; process the floating overflow
                50   09   07   51     F0     1374  3731  6$:     INSV    R1,#7,#9,R0     ; insert exponent and clear sign
                          04 01 A4     E9     1379  3732          BLBC    SIGN(R4),7$     ; is the value negative ?
                       00 50   0F     E2     137D  3733          BBSS    #15,R0,7$       ; yes - set the sign bit
                                 05     1381  3734  7$:     RSB                     ; return
                                        1382  3735          ;
```

```
                                1382  3737          ;
                                1382  3738          ;       PACK_DFLOAT - Pack a D_floating Value
                                1382  3739          ;
                                1382  3740          ;               entered by subroutine branching
                                1382  3741          ;
                                1382  3742          ;               parameter:      (R4)= Source Value
                                1382  3743          ;
                                1382  3744          ;               returns with    R0,R1 = Converted Double Value
                                1382  3745          ;
                                1382  3746 PACK_DFLOAT:                             ; entrance
                     03 64   E9 1382  3747          BLBC    ZERO(R4),2$             ; value is not zero - bypass
                        50   7C 1385  3748 1$:       CLRQ    R0                      ; clear the value
                             05 1387  3749          RSB                             ; return
                 15 10 A4    07 E1    1388  3750 2$:       BBC     #7,FRACTION+8(R4),3$    ; rounding bit is zero - bypass
             10 A4 00000100  8F C0    138D  3751          ADDL2   #128,FRACTION+8(R4)     ; round the value
                   14 A4     00 D8    1395  3752          ADWC    #0,FRACTION+12(R4)      ; propagate any carry
                             07 1E    1399  3753          BCC     3$                      ; no carry out of fraction - bypass
                     04 A4   D6 139B  3754          INCL    POWER(R4)               ; increment the exponent
                        50   7C 139E  3755          CLRQ    R0                      ; clear the fraction bits
                             0A 11    13A0  3756          BRB     4$                      ; bypass
                50   14 A4    08 9C    13A2  3757 3$:       ROTL    #8,FRACTION+12(R4),R0   ; load the first part of the fraction
                51   11 A4    10 9C    13A7  3758          ROTL    #16,FRACTION+9(R4),R1   ; load the second part of the fraction
        52   04 A4 00000080  8F C1    13AC  3759 4$:       ADDL3   #128,POWER(R4),R2       ; R2 = biased exponent
                             0B 14    13B5  3760          BGTR    5$                      ; it's greater than zero - bypass
                        64   01 D0    13B7  3761          MOVL    #1,(R4)                 ; mark the original value as zero
                   C6 54 AD    06 E1    13BA  3762          BBC     #PSL_FU,PSL(FP),1$      ; no fault enabled - return zero
                         08D5  31 13BF  3763          BRW     UNDERFLOW               ; process the floating underflow
                52   000000FF  8F D1    13C2  3764 5$:       CMPL    #255,R2                 ; is the exponent too large ?
                             03 18    13C9  3765          BGEQ    6$                      ; no - skip
                         08DA  31 13CB  3766          BRW     OVERFLOW                ; process the floating overflow
           50   09    07    52 F0    13CE  3767 6$:       INSV    R2,#7,#9,R0             ; insert exponent and clear sign
                   04 01 A4    E9 13D3  3768          BLBC    SIGN(R4),7$             ; is the value negative ?
                00 50    0F    E2 13D7  3769          BBSS    #15,R0,7$               ; yes - set the sign bit
                             05 13DB  3770 7$:       RSB                             ; return
                                13DC  3771          ;
```

```
                                13DC  3773          ;
                                13DC  3774          ;       PACK_GFLOAT - Pack a G_floating Value
                                13DC  3775          ;
                                13DC  3776          ;               entered by subroutine branching
                                13DC  3777          ;
                                13DC  3778          ;               parameter:      R4 = Location of Source Value
                                13DC  3779          ;
                                13DC  3780          ;               returns with    R0,R1 = Converted G_floating Value
                                13DC  3781          ;
                                13DC  3782 PACK_GFLOAT:                             ; entrance
                      03 64     E9  13DC  3783          BLBC    ZERO(R4),2$         ; value is not zero - bypass
                         50     7C  13DF  3784 1$:      CLRQ    R0                  ; clear the value
                         05     13  13E1  3785          RSB                         ; return
                 15 10 A4       0A  E1  13E2  3786 2$:      BBC     #10,FRACTION+8(R4),3$   ; rounding bit is zero - bypass
              10 A4   00000800  8F  C0  13E7  3787          ADDL2   #1@11,FRACTION+8(R4)    ; round the value
                   14 A4        00  D8  13EF  3788          ADWC    #0,FRACTION+12(R4)      ; propagate any carry
                         07     1E  13F3  3789          BCC     3$                  ; no carry out of fraction - bypass
                      04 A4     D6  13F5  3790          INCL    POWER(R4)           ; increment the exponent
                         50     7C  13F8  3791          CLRQ    R0                  ; clear the fraction bits
                         0F     11  13FA  3792          BRB     4$                  ; bypass
                50      14 A4    05  9C  13FC  3793 3$:      ROTL    #5,FRACTION+12(R4),R0   ; load the first part of the fraction
             51   10 A4    20    0B  EE  1401  3794          EXTV    #11,#32,FRACTION+8(R4),R1 ; load second part of the fraction
                   51     51    10  9C  1407  3795          ROTL    #16,R1,R1           ; unscramble the bits
          53   04 A4   00000400  8F  C1  140B  3796 4$:      ADDL3   #1024,POWER(R4),R3  ; R3 = biased exponent
                         0B     14  1414  3797          BGTR    5$                  ; it's greater than zero - bypass
                      64     01  D0  1416  3798          MOVL    #1,(R4)             ; mark the original value as zero
                   C1 54 AD     06  E1  1419  3799          BBC     #PSL_FU,PSL(FP),1$  ; no fault enabled - return zero
                         0876   31  141E  3800          BRW     UNDERFLOW           ; process the floating underflow
                53    000007FF  8F  D1  1421  3801 5$:      CMPL    #2047,R3            ; is the exponent too large ?
                         03     18  1428  3802          BGEQ    6$                  ; no - skip
                         087B   31  142A  3803          BRW     OVERFLOW            ; process the floating overflow
          50    0C     04     53  F0  142D  3804 6$:      INSV    R3,#4,#12,R0        ; insert exponent and clear sign
                   04 01 A4     E9  1432  3805          BLBC    SIGN(R4),7$         ; is the value negative ?
                00 50     0F    E2  1436  3806          BBSS    #15,R0,7$           ; yes - set the sign bit
                         05     13A   3807 7$:      RSB                         ; return
                                143B  3808          ;
```

```
                                143B  3810          ;
                                143B  3811          ;       PACK_HFLOAT - Pack an H_floating Value
                                143B  3812          ;
                                143B  3813          ;               entered by subroutine branching
                                143B  3814          ;
                                143B  3815          ;               parameter:      R4 = Location of Source Value
                                143B  3816          ;
                                143B  3817          ;               returns with    R0,R1,R2,R3 = Converted HFLOAT Value
                                143B  3818          ;
                                143B  3819 PACK_HFLOAT:                             ; entrance
                     05 64   E9 143B  3820          BLBC    ZERO(R4),2$             ; value is not zero - bypass
                        50   7C 143E  3821 1$:       CLRQ    R0                      ; clear the first part of the value
                        52   7C 1440  3822          CLRQ    R2                      ; clear the second part of the value
                        05   1442  3823          RSB                             ; return
                  1F 08 A4   0E E1  1443  3824 2$:       BBC     #14,FRACTION(R4),3$     ; rounding bit is zero - bypass
               08 A4  00008000 8F CO  1448  3825          ADDL2   #1@15,FRACTION(R4)      ; round the value
                     0C A4   00 D8  1450  3826          ADWC    #0,FRACTION+4(R4)       ; propagate carry into third part
                     10 A4   00 D8  1454  3827          ADWC    #0,FRACTION+8(R4)       ; propagate carry into second part
                     14 A4   00 D8  1458  3828          ADWC    #0,FRACTION+12(R4)      ; propagate carry into first part
                        09   1E 145C  3829          BCC     3$                      ; no carry out of fraction - bypass
                     04 A4   D6 145E  3830          INCL    POWER(R4)               ; increment the exponent
                        50   7C 1461  3831          CLRQ    R0                      ; clear first part of the fraction
                        52   7C 1463  3832          CLRQ    R2                      ; clear second part of the fraction
                        23   11 1465  3833          BRB     4$                      ; bypass
                  50    14 A4   01 78  1467  3834 3$:       ASHL    #1,FRACTION+12(R4),R0   ; load first part of the fraction
               51 10 A4    20   0F EE  146C  3835          EXTV    #15,#32,FRACTION+8(R4),R1 ; load second part of the fraction
                     51    51   10 9C  1472  3836          ROTL    #16,R1,R1               ; reverse the words
               52 0C A4    20   0F EE  1476  3837          EXTV    #15,#32,FRACTION+4(R4),R2 ; load third part of the fraction
                     52    52   10 9C  147C  3838          ROTL    #16,R2,R2               ; reverse the words
               53 08 A4    20   0F EE  1480  3839          EXTV    #15,#32,FRACTION(R4),R3 ; load  third part of the fraction
                     53    53   10 9C  1486  3840          ROTL    #16,R3,R3               ; reverse the words
            55 04 A4   00004000 8F C1  148A  3841 4$:       ADDL3   #16384,POWER(R4),R5     ; R5 = biased exponent
                        0B   14 1493  3842          BGTR    5$                      ; it's greater than zero - bypass
                     64    01 D0  1495  3843          MOVL    #1,(R4)                 ; mark the original value as zero
                  A1 54 AD    06 E1  1498  3844          BBC     #PSL_FU,PSL(FP),1$      ; no fault enabled - return zero
                        07F7   31 149D  3845          BRW     UNDERFLOW               ; process the floating underflow
                  55   00007FFF 8F D1  14A0  3846 5$:       CMPL    #32767,R5               ; is the exponent too large ?
                        03   18 14A7  3847          BGEQ    6$                      ; no - skip
                        07FC   31 14A9  3848          BRW     OVERFLOW                ; process the floating overflow
                     50    55   B0 14AC  3849 6$:       MOVW    R5,R0                   ; insert exponent and clear sign
                  04 01 A4   E9 14AF  3850          BLBC    SIGN(R4),7$             ; is the value negative ?
               00 50    0F   E2 14B3  3851          BBSS    #15,R0,7$               ; yes - set the sign bit
                        05   14B7  3852 7$:       RSB                             ; return
                             14B8  3853          ;
```

```
                                14B8  3855          .SBTTL  STORE_OPERAND
                                14B8  3856
                                14B8  3857          ;
                                14B8  3858          ;       STORE_OPERAND - Move result to destination
                                14B8  3859          ;
                                14B8  3860          ;       entered by BSBW
                                14B8  3861          ;
                                14B8  3862          ;       arguments:      R0-R3 contain operand to store
                                14B8  3863          ;                       R11 contains address of destination
                                14B8  3864          ;                       OP_INDEX(FP) contains pointer to type code
                                14B8  3865          ;
                                14B8  3866
                                14B8  3867  STORE_OPERAND:
        08    01    8B BD   8F  14B8  3868          CASEB   @OP_INDEX(FP),#TYP_B,#<TYP_H-TYP_B>
                        0012'   14BD  3869  1$:      .WORD   2$-1$                   ; 1 - byte
                        0016'   14BF  3870          .WORD   3$-1$                   ; 2 - word
                        001A'   14C1  3871          .WORD   4$-1$                   ; 3 - longword
                        0022'   14C3  3872          .WORD   6$-1$                   ; 4 - quadword
                        001E'   14C5  3873          .WORD   5$-1$                   ; 5 - octaword
                        001A'   14C7  3874          .WORD   4$-1$                   ; 6 - F_floating
                        0022'   14C9  3875          .WORD   6$-1$                   ; 7 - D_floating
                        0022'   14CB  3876          .WORD   6$-1$                   ; 8 - G_floating
                        001E'   14CD  3877          .WORD   5$-1$                   ; 9 - H_floating
                                14CF  3878
            6B    50    90    14CF  3879  2$:      MOVB    R0,(R11)                ; byte
                        05    14D2  3880          RSB                             ;
            6B    50    B0    14D3  3881  3$:      MOVW    R0,(R11)                ; word
                        05    14D6  3882          RSB                             ;
            6B    50    D0    14D7  3883  4$:      MOVL    R0,(R11)                ; longword
                        05    14DA  3884          RSB                             ;
        08  AB    52    7D    14DB  3885  5$:      MOVQ    R2,8(R11)               ; octaword
            6B    50    7D    14DF  3886  6$:      MOVQ    R0,(R11)                ; quadword
                        05    14E2  3887          RSB                             ;
                              14E3  3888
```

```
                        ;*****************************************************************************
                        ;*                                                                           *
                        ;*                                                                           *
                        ;*                          Arithmetic Routines                              *
                        ;*                                                                           *
                        ;*                                                                           *
                        ;*****************************************************************************
                        ;
                        ;       Introduction
                        ;       ------------
                        ;
                        ;           The routines which follow perform the actual floating
                        ;       arithmetic operations of the Emulator. These routines all
                        ;       work with the internal floating representation so only one
                        ;       routine is needed for each type of operation. However, since
                        ;       multiplication and division are comparatively slow operations
                        ;       separate routines have been included for the GFLOAT and HFLOAT
                        ;       versions of these operations so the GFLOAT operation will not
                        ;       be slowed to the speed of the HFLOAT operation.
                        ;
                        ;           The algorithms for the individual routines will be
                        ;       described in the routines themselves. The following discussion
                        ;       will be limited to a description of our internal floating
                        ;       format.
                        ;
                        ;       Internal Floating Representation
                        ;       --------------------------------
                        ;
                        ;           All of the floating arithmetic operations used by the
                        ;       Emulator are performed using an internal floating format
                        ;       which is much easier to work with than any of the hardware
                        ;       floating representations and which is sufficiently accurate
                        ;       to represent all of the hardware representations. The format
                        ;       is also used as an intermediate representation for emulating
                        ;       the conversion instructions.
                        ;
                        ;           The internal representation has the following four fields:
                        ;
                        ;               ZERO            is a one byte field whose low order
                        ;                               bit indicates that the represented
                        ;                               value is nonzero.
                        ;
                        ;               SIGN            is a one byte field whose low order
                        ;                               bit indicates that the represented
                        ;                               value is negative. If the low order
                        ;                               bit of ZERO is set then this field
                        ;                               must be zero.
                        ;
                        ;               POWER           is a longword field which contains the
                        ;                               exponent of the power of two which
                        ;                               is used to scale the fraction
                        ;
                        ;               FRACTION        is a 128 bit field (four longwords)
                        ;                               which hold the fraction as a single
```

```
                        ;                       128 bit value. The decimal point is
                        ;                       assumed to be at the end of the
                        ;                       fraction next to the high order bit.
                        ;                       The fraction is considered to be a
                        ;                       a positive value and is normalized
                        ;                       if the high order bit is one.
                        ;
                        ;   Three areas OPERAND1, OPERAND2, OPERAND3 are available for
                        ;   holding values in the internal representation. The name of
                        ;   of a field of one of these areas is found by appending the
                        ;   trailing digit of the area name to the field name. Thus
                        ;   POWER2 is the POWER field of OPERAND2.
                        ;
```

```
                                14E3  3962          ;
                                14E3  3963          ;       NEGATE_REAL - Negate a Real Value
                                14E3  3964          ;
                                14E3  3965          ;               entered by subroutine branching
                                14E3  3966          ;
                                14E3  3967          ;               parameter:      OPERAND1 = The Floating Value
                                14E3  3968          ;
                                14E3  3969          ;               returns with    OPERAND1 = The Negated Result
                                14E3  3970          ;
                                14E3  3971          ;       Discussion
                                14E3  3972          ;
                                14E3  3973          ;           If the value is nonzero, then the sign of the value
                                14E3  3974          ;       is complemented.
                                14E3  3975          ;
                                14E3  3976 NEGATE_REAL:                                 ; entrance
                   04 C7 AD     E8  14E3  3977              BLBS    ZERO1(FP),1$        ; don't negate zero
                 C8 AD    01    8C  14E7  3978              XORB2   #1,SIGN1(FP)        ; complement the sign
                               05  14EB  3979 1$:          RSB                         ; return
                                14EC  3980          ;
                                14EC  3981          ;       FLOAT_LONG - Convert a Longword to a Floating Value
                                14EC  3982          ;
                                14EC  3983          ;               entered by subroutine branching
                                14EC  3984          ;
                                14EC  3985          ;               parameter:      R0 = The Longword Value
                                14EC  3986          ;
                                14EC  3987          ;               returns with    OPERAND1 = The Converted Value
                                14EC  3988          ;
                                14EC  3989          ;       Discussion
                                14EC  3990          ;
                                14EC  3991          ;           The longword is converted to double floating using the
                                14EC  3992          ;       hardware and then to the internal representation by one of
                                14EC  3993          ;       the unpack routines.
                                14EC  3994          ;
                                14EC  3995 FLOAT_LONG:                                  ; entrance
                                14EC  3996 ;*
                                14EC  3997 ; The original code used CVTLD, which we can no longer assume is supported
                                14EC  3998 ; in hardware.  The action of a CVTLD R0,R0 is therefore emulated using
                                14EC  3999 ; non-floating instructions.
                                14EC  4000 ;
                                14EC  4001 ;            CVTLD   R0,R0                   ; convert the value to double floating
                                14EC  4002 ;*
                52   000000A0   8F    D0  14EC  4003              MOVL    #<128+32>,R2        ; set initial biased exponent
                                 51    D4  14F3  4004              CLRL    R1                  ; zero "low" longword
                                 50    D5  14F5  4005              TSTL    R0                  ; test sign of value
                                 23    13  14F7  4006              BEQL    30$                 ; skip if zero
                                 0A    14  14F9  4007              BGTR    20$                 ; skip if positive
                     52    0100  8F    A8  14FB  4008              BISW2   #^X100,R2           ; remember that it is negative
                           50    50    CE  1500  4009              MNEGL   R0,R0               ; get |R0|
                                 08    1D  1503  4010              BVS     25$                 ; skip shift loop if -2**32
                                 52    D7  1505  4011 20$:         DECL    R2                  ; decrement binary exponent
                     50    50    01    78  1507  4012              ASHL    #1,R0,R0            ; shift value left one bit
                                 F8    1C  150B  4013              BVC     20$                 ; repeat until bit 31 set
                     50    50    08    9C  150D  4014 25$:         ROTL    #8,R0,R0            ; rearrange into floating format
               51    50    00FF  8F    AB  1511  4015              BICW3   #^X00FF,R0,R1       ; move low 8 bits into place in R1
               50    09    07    52    F0  1517  4016              INSV    R2,#7,#9,R0         ; move sign+exponent into place
                       54    C7 AD    9E  151C  4017 30$:         MOVAB   OPERAND1(FP),R4     ; set operand address
                             FD20     30  1520  4018              BSBW    UNPACK_DFLOAT       ; unpack the value
```

```
                                                                L 4
 05  1523  4019          RSB                         ; return
     1524  4020          ;
     1524  4021
```

```
                                1524  4023          ;
                                1524  4024          ;       FIX_REAL - Convert a Floating Value to a Longword (Truncated)
                                1524  4025          ;
                                1524  4026          ;               entered by subroutine branching
                                1524  4027          ;
                                1524  4028          ;               parameter:      OPERAND1 = The Floating Value
                                1524  4029          ;
                                1524  4030          ;               returns with    R0 = Longword Result
                                1524  4031          ;
                                1524  4032          ;       Discussion
                                1524  4033          ;
                                1524  4034          ;           The exponent is used to determine where in the fraction
                                1524  4035          ;       the decimal point lies and any part of the integer part that
                                1524  4036          ;       exists within the signed fraction is extracted. If the
                                1524  4037          ;       fraction contains bits of higher order than those extracted,
                                1524  4038          ;       then the V bit is set in the user's PSL to indicate an integer
                                1524  4039          ;       overflow.
                                1524  4040          ;
                                1524  4041 FIX_REAL:                                  ; entrance
                     50   D4    1524  4042          CLRL    R0                        ; clear the returned value
                  29 C7 AD   E8 1526  4043          BLBS    ZERO1(FP),2$              ; the value is zero - return
                51    CB AD   CE 152A  4044          MNEGL   POWER1(FP),R1             ; R1 = negated exponent
                        23   18 152E  4045          BGEQ    2$                        ; value is less than one - return
                   CB AD  20   D1 1530  4046          CMPL    #32,POWER1(FP)            ; is the decimal point deep inside ?
                        1E   19 1534  4047          BLSS    3$                        ; yes - bypass
50   DF AD    CB AD   51   EF 1536  4048          EXTZV   R1,POWER1(FP),FRACTION1+16(FP),R0 ; extract leading bits
                  03 C8 AD   E9 153D  4049          BLBC    SIGN1(FP),1$              ; the value is positive - skip
                     50   50   CE 1541  4050          MNEGL   R0,R0                     ; negate the value
             51      50   01   9C 1544  4051 1$:      ROTL    #1,R0,R1                  ; position the sign bit
                51    C8 AD   8C 1548  4052          XORB2   SIGN1(FP),R1              ; complement it if negative value
                     04 51   E9 154C  4053          BLBC    R1,2$                     ; the sign is correct - skip
                54 AD    02   C8 154F  4054          BISL2   #PSLM_V,PSL(FP)           ; indicate an integer overflow
                         05   1553  4055 2$:         RSB                               ; return
                54 AD    02   C8 1554  4056 3$:      BISL2   #PSLM_V,PSL(FP)           ; indicate an integer overflow
             51      00A0 C1   9E 1558  4057          MOVAB   160(R1),R1                ; switch origin to previous longword
                          F4   15 155D  4058          BLEQ    2$                        ; no nonzero bits - return
50   CB AD    20      51   EF 155F  4059          EXTZV   R1,#32,FRACTION1-4(FP),R0 ; extract 32 bits from the fraction
             51   20    51   C3 1565  4060          SUBL3   R1,#32,R1                 ; compute the bits to clear
                          05   15 1569  4061          BLEQ    4$                        ; no bits to clear - bypass
   50        51   00    00   F0 156B  4062          INSV    #0,#0,R1,R0               ; clear some low order bits
                  DF C8 AD   E9 1570  4063 4$:      BLBC    SIGN1(FP),2$              ; the value is positive - return
                   50   50   CE 1574  4064          MNEGL   R0,R0                     ; complement the extracted bits
                         05   1577  4065          RSB                               ; return
                              1578  4066          ;
```

```
                                1578  4068          ;
                                1578  4069          ;       ROUND_REAL - Convert a Floating Value to a Longword (Rounded)
                                1578  4070          ;
                                1578  4071          ;               entered by subroutine branching
                                1578  4072          ;
                                1578  4073          ;               parameter:      OPERAND1 = The Floating Value
                                1578  4074          ;
                                1578  4075          ;               returns with    R0 = The Longword Result
                                1578  4076          ;
                                1578  4077          ;       Discussion
                                1578  4078          ;
                                1578  4079          ;           The exponent of the floating value is used to determine
                                1578  4080          ; where the decimal point goes within the signed fraction and
                                1578  4081          ; whatever part of the integer part exists within the fraction
                                1578  4082          ; is extracted. If the bit immediatly below the decimal point is
                                1578  4083          ; nonzero, then the integer part is rounded by adding a value
                                1578  4084          ; of one with the same sign as the floating value. If the value
                                1578  4085          ; contains significant bits of higher order than those in the
                                1578  4086          ; fraction or if overflow occured during the rounding operation
                                1578  4087          ; then the V bit is set in the user's PSL to indicate an integer
                                1578  4088          ; overflow.
                                1578  4089          ;
                                1578  4090 ROUND_REAL:                                ; entrance
              50    D4          1578  4091          CLRL    R0                        ; clear the returned value
           39 C7 AD E8          157A  4092          BLBS    ZERO1(FP),3$              ; the value is zero - return
         51    CB AD CE          157E  4093          MNEGL   POWER1(FP),R1             ; R1 = negated exponent
              33    14          1582  4094          BGTR    3$                        ; the value is less than 0.5 - return
         CB AD      20 D1       1584  4095          CMPL    #32,POWER1(FP)            ; is the decimal point deep inside ?
              2E    19          1588  4096          BLSS    4$                        ; yes - bypass
50    DF AD    CB AD  51 EF     158A  4097          EXTZV   R1,POWER1(FP),FRACTION1+16(FP),R0 ; extract some leading bits
                 51    3F C0     1591  4098          ADDL2   #63,R1                    ; *** equivalent sequence to ***
           08 D7 AD    51 E1     1594  4099          BBC     R1,FRACTION1+8(FP),1$     ; *** get around hardware problem ***
                             1599  4100 ;            DECL    R1                        ; compute the rounding position
                             1599  4101 ;            BBC     R1,FRACTION1+16(FP),1$    ; rounding bit is clear - skip
                 50    D6     1599  4102          INCL    R0                        ; round the extracted bits
                 04    1E     159B  4103          BCC     1$                        ; no carry - skip
              54 AD    02 C8  159D  4104          BISL2   #PSLM_V,PSL(FP)           ; indicate an integer overflow
                03 C8 AD E9     15A1  4105 1$:      BLBC    SIGN1(FP),2$              ; is the floating value negative
                 50    50 CE     15A5  4106          MNEGL   R0,R0                     ; yes - complement the value
              51    50    01 9C  15A8  4107 2$:      ROTL    #1,R0,R1                  ; position the sign bit
                51    C8 AD 8C     15AC  4108          XORB2   SIGN1(FP),R1              ; complment it if negative value
                   04 51 E9        15B0  4109          BLBC    R1,3$                     ; the sign is correct - skip
              54 AD    02 C8  15B3  4110          BISL2   #PSLM_V,PSL(FP)           ; indicate an integer overflow
                             15B7  4111 3$:       RSB                               ; return
              54 AD    02 C8  15B8  4112 4$:      BISL2   #PSLM_V,PSL(FP)           ; indicate an integer overflow
              51    00A0 C1  15BC  4113          MOVAB   160(R1),R1                ; switch origin to previous longword
                       F4 15  15C1  4114          BLEQ    3$                        ; no nonzero bits - return
50    CB AD    20    51 EF     15C3  4115          EXTZV   R1,#32,FRACTION1-4(FP),R0 ; extract 32 bits from the fraction
              52    20 51 C3     15C9  4116          SUBL3   R1,#32,R2                 ; compute the bits to clear
                       07 19     15CD  4117          BLSS    5$                        ; possible rounding - skip
   50    52    00    00 F0     15CF  4118          INSV    #0,#0,R2,R0               ; clear some low order bits
                       09 11     15D4  4119          BRB     6$                        ; bypass
                       51 D7     15D6  4120 5$:      DECL    R1                        ; compute the rounding bit
           02 CB AD    51 E1     15D8  4121          BBC     R1,FRACTION1-4(FP),6$     ; the rounding bit is clear - skip
                       50 D6     15DD  4122          INCL    R0                        ; round the extracted bits
              D4 C8 AD E9     15DF  4123 6$:      BLBC    SIGN1(FP),3$              ; the value is positive - return
                 50    50 CE     15E3  4124          MNEGL   R0,R0                     ; negate the value
```

```
05  15E6  4125          RSB                             ; return
      15E7  4126          ;
```

```
                                15E7  4128          ;
                                15E7  4129          ;       FRACTION_REAL - Isolate the Fraction Part of a Floating Value
                                15E7  4130          ;
                                15E7  4131          ;               entered by subroutine branching
                                15E7  4132          ;
                                15E7  4133          ;               parameter:      OPERAND1 = The Floating Value
                                15E7  4134          ;
                                15E7  4135          ;               returns with    OPERAND1 = The Fractional Part
                                15E7  4136          ;
                                15E7  4137          ;       Discussion
                                15E7  4138          ;
                                15E7  4139          ;           The exponent is used to determine the position of the
                                15E7  4140          ;       decimal point within the fraction and all of the bits of
                                15E7  4141          ;       the fraction above the decimal point are cleared. The result
                                15E7  4142          ;       is then normalized.
                                15E7  4143          ;
                                15E7  4144  FRACTION_REAL:                              ; entrance
          19 C7 AD    E8        15E7  4145          BLBS    ZERO1(FP),1$                ; the value is zero - return
       50    CB AD    D0        15EB  4146          MOVL    POWER1(FP),R0               ; R0 = the exponent
                13    15        15EF  4147          BLEQ    1$                          ; the value is all fraction - return
    51  50    05    00  EF      15F1  4148          EXTZV   #0,#5,R0,R1                 ; R1 = bits to clear in last longword
  50    50    FB    8F  78      15F6  4149          ASHL    #-5,R0,R0                   ; R0 = number of longwords to clear
          50    04    D1        15FB  4150          CMPL    #4,R0                       ; will we clear the whole value ?
                05    14        15FE  4151          BGTR    2$                          ; no - bypass
       C7 AD    01    D0        1600  4152          MOVL    #1,OPERAND1(FP)             ; mark the value as zero
                05    1604  4153  1$:   RSB                                           ; return
       52  20    51    C3        1605  4154  2$:    SUBL3   R1,#32,R2                   ; compute the last clear position
         53    DF AD    DE      1609  4155          MOVAL   FRACTION1+16(FP),R3         ; R3 = clear index
                02    11        160D  4156          BRB     4$                          ; enter the clear loop
                73    D4        160F  4157  3$:     CLRL    -(R3)                       ; clear a longword of the fraction
             FB 50    F4        1611  4158  4$:     SOBGEQ  R0,3$                       ; more longwords to clear - loop
FC A3   51   52    00  F0        1614  4159          INSV    #0,R2,R1,-4(R3)             ; perform the last clear
              0362   30          161A  4160          BSBW    NORMALIZE                   ; normalize the result
                05    161D  4161          RSB                                         ; return
                      161E  4162          ;
```

```
                                                D 5
                     161E  4164 ;
                     161E  4165 ;       ADD_REAL - Add Floating Values
                     161E  4166 ;
                     161E  4167 ;              entered by subroutine branching
                     161E  4168 ;
                     161E  4169 ;              parameters:     OPERAND1 = First Floating Operand
                     161E  4170 ;                              OPERAND2 = Second Floating Operand
                     161E  4171 ;
                     161E  4172 ;              returns with    OPERAND1 = The Floating Result
                     161E  4173 ;
                     161E  4174 ;       Discussion
                     161E  4175 ;
                     161E  4176 ;           This routine performs the addition of floating values
                     161E  4177 ;       in the internal representation in such a way that the sum is
                     161E  4178 ;       the exact sum truncated to 127 or 128 significant bits. This
                     161E  4179 ;       precision is sufficient for performing G_floating and HFLOAT additio
                     161E  4180 ;       since these operations are based on truncating the exact sum
                     161E  4181 ;       to smaller numbers of significant digits.
                     161E  4182 ;
                     161E  4183 ;           After preliminary checks for zero operands, the general
                     161E  4184 ;       addition is performed by first identifying a primary and a
                     161E  4185 ;       secondary operand with the primary operand being sufficiently
                     161E  4186 ;       large that it will not need to be shifted to align the values.
                     161E  4187 ;       This choice is made by examining the exponents. If the
                     161E  4188 ;       operands have opposite signs, the fractions are further
                     161E  4189 ;       compared so the magnitude of the primary operand is larger
                     161E  4190 ;       than the magnitude of the secondary operand. The primary and
                     161E  4191 ;       secondary operand fractions are loaded into groups of
                     161E  4192 ;       registers and the primary operand is shifted to the right one
                     161E  4193 ;       bit to allow for overflows. The secondary operand fraction is
                     161E  4194 ;       shifted to line it up with the primary operand fraction. If
                     161E  4195 ;       the signs of the operands are not the same then a record is
                     161E  4196 ;       made if any significant bits are lost during the alignment of
                     161E  4197 ;       the secondary operand fraction. The resulting fractions are
                     161E  4198 ;       added or subtracted and an additional one is subtracted if
                     161E  4199 ;       lost bits were detected in the test made above. The result
                     161E  4200 ;       is then normalized.
                     161E  4201 ;
                     161E  4202 ADD_REAL:                                    ; entrance
     50    C7 AD    9E  161E  4203         MOVAB   OPERAND1(FP),R0          ; R0 = location of first operand
     51    AF AD    9E  1622  4204         MOVAB   OPERAND2(FP),R1          ; R1 = location of second operand
           41 61    E8  1626  4205         BLBS    ZERO(R1),3$              ; second operand is zero - bypass
           15 60    E9  1629  4206         BLBC    ZERO(R0),1$              ; first operand is not zero - bypass
C7 AD      AF AD    D0  162C  4207         MOVL    OPERAND2(FP),OPERAND1(FP) ; copy the sign and zero flags
CB AD      B3 AD    D0  1631  4208         MOVL    POWER2(FP),POWER1(FP)    ; copy the exponent
CF AD      B7 AD    7D  1636  4209         MOVQ    FRACTION2(FP),FRACTION1(FP) ; copy second half of fraction
D7 AD      BF AD    7D  163B  4210         MOVQ    FRACTION2+8(FP),FRACTION1+8(FP) ; copy first half of fraction
                    05  1640  4211         RSB                              ; done
04 A1      04 A0    D1  1641  4212 1$:     CMPL    POWER(R0),POWER(R1)      ; compare the exponents
              2A    14  1646  4213         BGTR    5$                       ; first is greater - bypass
              21    19  1648  4214         BLSS    4$                       ; second is greater - bypass
01 A1      01 A0    91  164A  4215         CMPB    SIGN(R0),SIGN(R1)        ; compare the signs
              21    13  164F  4216         BEQL    5$                       ; they're equal - bypass
     52    18 A0    9E  1651  4217         MOVAB   FRACTION+16(R0),R2       ; R2 = position past first fraction
     53    18 A1    9E  1655  4218         MOVAB   FRACTION+16(R1),R3       ; R3 = position past second fraction
        54    04    D0  1659  4219         MOVL    #4,R4                    ; R4 = loop counter
        73    72    D1  165C  4220 2$:     CMPL    -(R2),-(R3)              ; compare two fraction longwords
```

```
                                                E 5
                    11   1A  165F  4221          BGTRU   5$                      ; first is greater - bypass
                    08   1F  1661  4222          BLSSU   4$                      ; second is greater - bypass
                  F6 54  F5  1663  4223          SOBGTR  R4,2$                   ; more longwords to compare - loop
                C7 AD    01  D0  1666  4224          MOVL    #1,OPERAND1(FP)         ; mark the result as zero
                         05  166A  4225  3$:        RSB                             ; return
                   51    50  D0  166B  4226  4$:    MOVL    R0,R1                   ; R1 = secondary operand location
                50   AF  AD  9E  166E  4227          MOVAB   OPERAND2(FP),R0         ; R0 = primary operand location
          52    04 A0    04 A1  C3  1672  4228  5$:    SUBL3   POWER(R1),POWER(R0),R2  ; R2 = exponent difference
          CB AD    04 A0    01  C1  1678  4229          ADDL3   #1,POWER(R0),POWER1(FP) ; store the result exponent
          53   01 A1    01 A0  8D  167E  4230          XORB3   SIGN(R0),SIGN(R1),R3    ; R3 = subtraction flag
                C7 AD    60  D0  1684  4231          MOVL    (R0),OPERAND1(FP)       ; store the result sign and zero flag
          54   08 A1    FF 8F  79  1688  4232          ASHQ    #-1,FRACTION(R1),R4     ; R4 = last shifted secondary longword
          55   0C A1    FF 8F  79  168E  4233          ASHQ    #-1,FRACTION+4(R1),R5   ; R5 = previous shifted longword
          56   10 A1    FF 8F  79  1694  4234          ASHQ    #-1,FRACTION+8(R1),R6   ; R6,R7 = first two shifted longwords
                00 57    1F  E5  169A  4235          BBCC    #31,R7,6$               ; clear the high order bit
       CF AD    08 A0    FF 8F  79  169E  4236  6$:    ASHQ    #-1,FRACTION(R0),FRACTION1(FP) ; shift fourth result longword
          D3 AD    D3 AD    01  78  16A5  4237          ASHL    #1,FRACTION1+4(FP),FRACTION1+4(FP) ; position third longword
       D3 AD    0C A0    FF 8F  79  16AB  4238          ASHQ    #-1,FRACTION+4(R0),FRACTION1+4(FP) ; shift third longword
          D7 AD    D7 AD    01  78  16B2  4239          ASHL    #1,FRACTION1+8(FP),FRACTION1+8(FP) ; position second longword
       D7 AD    10 A0    FF 8F  79  16B8  4240          ASHQ    #-1,FRACTION+8(R0),FRACTION1+8(FP) ; shift first two longwords
                00 DB AD    1F  E5  16BF  4241          BBCC    #31,FRACTION1+12(FP),7$ ; clear the high order bit
                   58    01  D0  16C4  4242  7$:    MOVL    #1,R8                   ; R8 = negation adjustment
             52    0000007F 8F  D1  16C7  4243          CMPL    #127,R2                 ; is the shift count large ?
                         04  18  16CE  4244          BGEQ    8$                      ; no - skip
                   52    7F 8F  9A  16D0  4245          MOVZBL  #127,R2                 ; yes - use a smaller one
                         52  D5  16D4  4246  8$:    TSTL    R2                      ; is the shift count zero ?
                         38  13  16D6  4247          BEQL    12$                     ; yes - bypass
                      52    20  D1  16D8  4248          CMPL    #32,R2                  ; is the shift more than a longword ?
                         13  18  16DB  4249          BGEQ    10$                     ; no - bypass
                         54  D5  16DD  4250          TSTL    R4                      ; is the last longword zero ?
                         02  13  16DF  4251          BEQL    9$                      ; no - skip
                         58  D4  16E1  4252          CLRL    R8                      ; clear the negation adjustment
                      52    20  C2  16E3  4253  9$:    SUBL2   #32,R2                  ; decrement the shift count
                      54    55  7D  16E6  4254          MOVQ    R5,R4                   ; shift the last two longwords
                      56    57  D0  16E9  4255          MOVL    R7,R6                   ; shift the previous longword
                         57  D4  16EC  4256          CLRL    R7                      ; clear the leading longword
                         E4  11  16EE  4257          BRB     8$                      ; try again
                      59    52  CE  16F0  4258  10$:    MNEGL   R2,R9                   ; R9 = - shift count
          00    54    52    00  ED  16F3  4259          CMPZV   #0,R2,R4,#0             ; are the low order bits zero ?
                         02  13  16F8  4260          BEQL    11$                     ; yes - skip
                         58  D4  16FA  4261          CLRL    R8                      ; clear the negation adjustment
                54    54    59  79  16FC  4262  11$:    ASHQ    R9,R4,R4                ; shift the last longword
                55    55    52  78  1700  4263          ASHL    R2,R5,R5                ; position the previous longword
                55    55    59  79  1704  4264          ASHQ    R9,R5,R5                ; shift the previous longword
                56    56    52  78  1708  4265          ASHL    R2,R6,R6                ; position the previous longword
                56    56    59  79  170C  4266          ASHQ    R9,R6,R6                ; shift the first two longwords
                   18 53  E9  1710  4267  12$:    BLBC    R3,13$                  ; not subtraction - bypass
                      57    57  D2  1713  4268          MCOML   R7,R7                   ; complement first longword
                      56    56  D2  1716  4269          MCOML   R6,R6                   ; complement second longword
                      55    55  D2  1719  4270          MCOML   R5,R5                   ; complement third longword
                      54    54  D2  171C  4271          MCOML   R4,R4                   ; complement last longword
                      54    58  C0  171F  4272          ADDL2   R8,R4                   ; add the negation adjustment
                      55    00  D8  1722  4273          ADWC    #0,R5                   ; propagate any carry
                      56    00  D8  1725  4274          ADWC    #0,R6                   ; propagate any carry
                      57    00  D8  1728  4275          ADWC    #0,R7                   ; propagate any carry
                CF AD    54  C0  172B  4276  13$:    ADDL2   R4,FRACTION1(FP)        ; add the fourth longwords
                D3 AD    55  D8  172F  4277          ADWC    R5,FRACTION1+4(FP)      ; add the third longwords
```

```
                                                F 5
D7 AD   56   D8  1733  4278          ADWC    R6,FRACTION1+8(FP)      ; add the second longwords
DB AD   57   D8  1737  4279          ADWC    R7,FRACTION1+12(FP)     ; add the leading longwords
       0241  30  173B  4280          BSBW    NORMALIZE               ; normalize the result
             05  173E  4281          RSB                             ; return
                 173F  4282          ;
```

```
                     173F  4284          .SBTTL  MULTIPLY_FLOAT
                     173F  4285
                     173F  4286          ;
                     173F  4287          ;       MULTIPLY_FLOAT - Multiply Two Floating Values
                     173F  4288          ;
                     173F  4289          ;               entered by subroutine branching
                     173F  4290          ;
                     173F  4291          ;               parameters:     OPERAND2 = First Floating Factor
                     173F  4292          ;                               OPERAND3 = Second Floating Factor
                     173F  4293          ;
                     173F  4294          ;               returns with    OPERAND1 = The Floating Product
                     173F  4295          ;                               R0 = Normalization Shift Count
                     173F  4296          ;
                     173F  4297          ;               See MULYIPLY_xFLOAT routines for more information.
                     173F  4298          ;
                     173F  4299
                     173F  4300 MULTIPLY_FLOAT:
03   06   8B BD   8F  173F  4301          CASEB   @OP_INDEX(FP),#TYP_F,#<TYP_H-TYP_F>
                0008' 1744  4302 1$:      .WORD   MULTIPLY_FFLOAT-1$      ; F_floating
                005B' 1746  4303          .WORD   MULTIPLY_DGFLOAT-1$     ; D_floating
                005B' 1748  4304          .WORD   MULTIPLY_DGFLOAT-1$     ; G_floating
                00AB' 174A  4305          .WORD   MULTIPLY_HFLOAT-1$      ; H_floating
```

```
                                 174C  4307          .SBTTL  MULTIPLY_FFLOAT
                                 174C  4308
                                 174C  4309          ;       Discussion
                                 174C  4310          ;
                                 174C  4311          ;           This routine forms the product of two floating values
                                 174C  4312          ;       in the internal representation and deliberately limits the
                                 174C  4313          ;       accuracy to 32 bits. Only the high order 32 bits of each of
                                 174C  4314          ;       the operand fractions is used and the result is the high
                                 174C  4315          ;       order 32 bits of the exact product. The remaining bits of the
                                 174C  4316          ;       fraction are zero. Upon return the register R0 contains the
                                 174C  4317          ;       distance the product was shifted during normalization.
                                 174C  4318          ;
                                 174C  4319          ;
                                 174C  4320  MULTIPLY_FFLOAT:
                C7 AD    01  D0  174C  4321          MOVL    #1,OPERAND1(FP)         ; mark the result as zero
                      50  D4  1750  4322          CLRL    R0                      ; clear the shift count
                  48 AF AD  E8  1752  4323          BLBS    ZERO2(FP),2$            ; first operand is zero - return
                  44 97 AD  E8  1756  4324          BLBS    ZERO3(FP),2$            ; second operand is zero - return
                     C7 AD  94  175A  4325          CLRB    ZERO1(FP)               ; clear the zero flag
                   50    01  D0  175D  4326          MOVL    #1,R0                   ; R0 = number of longwords to multiply
                51    C3 AD  9E  1760  4327          MOVAB   FRACTION2+12(FP),R1     ; R1 = location of first factor
                52    AB AD  9E  1764  4328          MOVAB   FRACTION3+12(FP),R2     ; R2 = location of second factor
                53    D7 AD  9E  1768  4329          MOVAB   FRACTION1+8(FP),R3      ; R3 = location for product
                     0296    30  176C  4330          BSBW    MULTIPLY                ; multiply the quadwords
                      50  D4  176F  4331          CLRL    R0                      ; clear the shift count
          CB AD    9B AD    B3 AD  C1  1771  4332          ADDL3   POWER2(FP),POWER3(FP),POWER1(FP) ; compute the exponent
          C8 AD    98 AD    B0 AD  8D  1778  4333          XORB3   SIGN2(FP),SIGN3(FP),SIGN1(FP) ; compute the sign
                14 DB AD    1F  E0  177F  4334          BBS     #31,FRACTION1+12(FP),1$ ; result is normalized - bypass
                      50  D6  1784  4335          INCL    R0                      ; set the shift count to one
                   CB AD  D7  1786  4336          DECL    POWER1(FP)              ; decrement the exponent
             DB AD    DB AD    01  78  1789  4337          ASHL    #1,FRACTION1+12(FP),FRACTION1+12(FP) ; normalize the fraction
                04 D7 AD    1F  E1  178F  4338          BBC     #31,FRACTION1+8(FP),1$  ; low order bit should be clear - skip
                   DB AD    01  C8  1794  4339          BISL2   #1,FRACTION1+12(FP)     ; set the low order bit
                      CF AD  7C  1798  4340  1$:       CLRQ    FRACTION1(FP)           ; extend the fraction to an octaword
                      D7 AD  D4  179B  4341                  CLRL    FRACTION1+8(FP)         ;
                             05  179E  4342  2$:       RSB                             ; return
                                 179F  4343          ;
```

```
                                                I 5
                                179F  4345          .SBTTL  MULTIPLY_DGFLOAT
                                179F  4346
                                179F  4347          ;       Discussion
                                179F  4348          ;
                                179F  4349          ;           This routine forms the product of two floating values
                                179F  4350          ;       in the internal representation and deliberately limits the
                                179F  4351          ;       accuracy to 64 bits. Only the high order 64 bits of each of
                                179F  4352          ;       the operand fractions is used and the result is the high
                                179F  4353          ;       order 64 bits of the exact product. The remaining bits of the
                                179F  4354          ;       fraction are zero. Upon return the register R0 contains the
                                179F  4355          ;       distance the product was shifted during normalization.
                                179F  4356          ;
                                179F  4357 MULTIPLY_DGFLOAT:                            ; entrance
            C7 AD    01    D0   179F  4358          MOVL    #1,OPERAND1(FP)            ; mark the result as zero
                     50    D4   17A3  4359          CLRL    R0                         ; clear the shift count
              45 AF  AD    E8   17A5  4360          BLBS    ZERO2(FP),2$               ; first operand is zero - return
              41 97  AD    E8   17A9  4361          BLBS    ZERO3(FP),2$               ; second operand is zero - return
                 C7  AD    94   17AD  4362          CLRB    ZERO1(FP)                  ; clear the zero flag
                 50    02    D0   17B0  4363          MOVL    #2,R0                      ; R0 = number of longwords to multiply
           51    BF  AD    9E   17B3  4364          MOVAB   FRACTION2+8(FP),R1         ; R1 = location of first factor
           52    A7  AD    9E   17B7  4365          MOVAB   FRACTION3+8(FP),R2         ; R2 = location of second factor
           53    CF  AD    9E   17BB  4366          MOVAB   FRACTION1(FP),R3           ; R3 = location for product
                   0243    30   17BF  4367          BSBW    MULTIPLY                   ; multiply the quadwords
                     50    D4   17C2  4368          CLRL    R0                         ; clear the shift count
CB AD    9B AD    B3 AD    C1   17C4  4369          ADDL3   POWER2(FP),POWER3(FP),POWER1(FP) ; compute the exponent
C8 AD    98 AD    B0 AD    8D   17CB  4370          XORB3   SIGN2(FP),SIGN3(FP),SIGN1(FP) ; compute the sign
         14 DB AD    1F    E0   17D2  4371          BBS     #31,FRACTION1+12(FP),1$ ; result is normalized - bypass
                     50    D6   17D7  4372          INCL    R0                         ; set the shift count to one
                 CB  AD    D7   17D9  4373          DECL    POWER1(FP)                 ; decrement the exponent
   D7 AD    D7 AD    01    79   17DC  4374          ASHQ    #1,FRACTION1+8(FP),FRACTION1+8(FP) ; normalize the fraction
         04 D3 AD    1F    E1   17E2  4375          BBC     #31,FRACTION1+4(FP),1$     ; low order bit should be clear - skip
            D7 AD    01    C8   17E7  4376          BISL2   #1,FRACTION1+8(FP)         ; set the low order bit
                 CF  AD    7C   17EB  4377 1$:      CLRQ    FRACTION1(FP)              ; extend the fraction to an octaword
                     05    05   17EE  4378 2$:      RSB                                ; return
                           17EF  4379              ;
```

```
                                17EF  4381          .SBTTL  MULTIPLY_HFLOAT
                                17EF  4382
                                17EF  4383          ;
                                17EF  4384          ;       Discussion
                                17EF  4385          ;
                                17EF  4386          ;           This routine computes the product of two floating values
                                17EF  4387          ;       in the internal representation. The fraction of the result
                                17EF  4388          ;       consists of the high order 128 bits of the exact product.
                                17EF  4389          ;       Upon return the register R0 contains the distance the product
                                17EF  4390          ;       was shifted during normalization.
                                17EF  4391          ;
                                17EF  4392  MULTIPLY_HFLOAT:
                C7 AD    01     D0  17EF  4393          MOVL    #1,OPERAND1(FP)             ; mark the result as zero
                         50     D4  17F3  4394          CLRL    R0                          ; clear the shift count
                   58 AF  AD     E8  17F5  4395          BLBS    ZERO2(FP),2$                ; first operand is zero - return
                   54 97  AD     E8  17F9  4396          BLBS    ZERO3(FP),2$                ; second operand is zero - return
                      C7  AD     94  17FD  4397          CLRB    ZERO1(FP)                   ; clear the zero flag
                   50     04     D0  1800  4398          MOVL    #4,R0                       ; R0 = number of longwords to multiply
                51    B7  AD     9E  1803  4399          MOVAB   FRACTION2(FP),R1            ; R1 = location of first factor
                52    9F  AD     9E  1807  4400          MOVAB   FRACTION3(FP),R2            ; R2 = location of second factor
                53    58  AD     9E  180B  4401          MOVAB   TEMP(FP),R3                 ; use temporary area for result
                       01F3      30  180F  4402          BSBW    MULTIPLY                    ; perform the multiplication
                         50     D4  1812  4403          CLRL    R0                          ; clear the shift count
          CB AD    9B AD    B3 AD     C1  1814  4404          ADDL3   POWER2(FP),POWER3(FP),POWER1(FP) ; compute the exponent
          C8 AD    98 AD    B0 AD     8D  181B  4405          XORB3   SIGN2(FP),SIGN3(FP),SIGN1(FP) ; compute the sign
                CF AD    68 AD     7D  1822  4406          MOVQ    TEMP+16(FP),FRACTION1(FP) ; insert second part of fraction
                D7 AD    70 AD     7D  1827  4407          MOVQ    TEMP+24(FP),FRACTION1+8(FP) ; insert first part of fraction
                            23     19  182C  4408          BLSS    2$                          ; fraction is normalized - bypass
                         50     D6  182E  4409          INCL    R0                          ; set the shift count to one
                      CB AD     D7  1830  4410          DECL    POWER1(FP)                  ; decrement the exponent
          D7 AD    D7 AD    01     79  1833  4411          ASHQ    #1,FRACTION1+8(FP),FRACTION1+8(FP) ; normalize the first part
                04 D3  AD    1F     E1  1839  4412          BBC     #31,FRACTION1+4(FP),1$      ; low order bit should be clear - skip
                   D7  AD    01     C8  183E  4413          BISL2   #1,FRACTION1+8(FP)          ; set the low bit in the first part
          CF AD    CF AD    01     79  1842  4414  1$:     ASHQ    #1,FRACTION1(FP),FRACTION1(FP) ; normalize the second part
                04 64  AD    1F     E1  1848  4415          BBC     #31,TEMP+12(FP),2$          ; low order bit should be clear - skip
                   CF  AD    01     C8  184D  4416          BISL2   #1,FRACTION1(FP)            ; set the low bit in the second part
                             05     1851  4417  2$:     RSB                                 ; return
                                    1852  4418          ;
```

```
                        1852  4420          .SBTTL  DIVIDE_FLOAT
                        1852  4421
                        1852  4422          ;
                        1852  4423          ;       DIVIDE_FLOAT - Divide Two Floating Values
                        1852  4424          ;
                        1852  4425          ;               entered by subroutine branching
                        1852  4426          ;
                        1852  4427          ;               parameters:     OPERAND2 = The Floating Dividend
                        1852  4428          ;                               OPERAND3 = The Floating Divisor
                        1852  4429          ;
                        1852  4430          ;               returns with    OPERAND1 = Floating Quotient
                        1852  4431          ;
                        1852  4432          ;       See DIVIDE_xFLOAT routines for more information.
                        1852  4433          ;
                        1852  4434
                        1852  4435 DIVIDE_FLOAT:
03   06   8B BD   8F    1852  4436          CASEB   @OP_INDEX(FP),#TYP_F,#<TYP_H-TYP_F>
                0008'   1857  4437 1$:      .WORD   DIVIDE_FFLOAT-1$        ; F_floating
                005B'   1859  4438          .WORD   DIVIDE_DGFLOAT-1$       ; D_floating
                005B'   185B  4439          .WORD   DIVIDE_DGFLOAT-1$       ; G_floating
                00AE'   185D  4440          .WORD   DIVIDE_HFLOAT-1$        ; H_floating
```

```
                                185F  4442          .SBTTL  DIVIDE_FFLOAT
                                185F  4443
                                185F  4444          ;       Discussion
                                185F  4445          ;
                                185F  4446          ;           This routine computes the quotient of two floating values
                                185F  4447          ;       in the internal representation. The fractions of the two input
                                185F  4448          ;       values consist of the high order 32 bits of the two operand
                                185F  4449          ;       fractions and the fraction of the quotient consists of the
                                185F  4450          ;       high order 31 or 32 bits of the exact quotient. The remaining
                                185F  4451          ;       bits of the fraction are set to zero. If the divisor is zero
                                185F  4452          ;       then a floating division by zero fault is signaled.
                                185F  4453          ;
                                185F  4454 DIVIDE_FFLOAT:                               ; entrance
                     03 97 AD   E9  185F  4455          BLBC    ZERO3(FP),1$            ; divisor is not zero - skip
                        0453    31  1863  4456          BRW     DIVIDE_FAULT            ; process the floating divide fault
                   C7 AD  01    D0  1866  4457 1$:      MOVL    #1,OPERAND1(FP)         ; mark the result as zero
                  43 AF AD      E8  186A  4458          BLBS    ZERO2(FP),3$            ; dividend is zero - bypass
          C3 AD    C3 AD  FF 8F  78  186E  4459          ASHL    #-1,FRACTION2+12(FP),FRACTION2+12(FP) ; normalize for division
                00 C3 AD   1F    E5  1875  4460          BBCC    #31,FRACTION2+12(FP),2$ ; clear the high order bit
                   B3 AD         D6  187A  4461 2$:      INCL    POWER2(FP)              ; increment the exponent
                      50  01     D0  187D  4462          MOVL    #1,R0                   ; R0 = number of longwords in divisor
                   51  BF AD     9E  1880  4463          MOVAB   FRACTION2+8(FP),R1      ; R1 = dividend location
                   52  AB AD     9E  1884  4464          MOVAB   FRACTION3+12(FP),R2     ; R2 = divisor location
                   53  DB AD     9E  1888  4465          MOVAB   FRACTION1+12(FP),R3     ; R3 = quotient area location
                     01C0        30  188C  4466          BSBW    DIVIDE                  ; perform the division
                   C7 AD         94  188F  4467          CLRB    ZERO1(FP)               ; indicate a nonzero result
          CB AD    B3 AD  9B AD  C3  1892  4468          SUBL3   POWER3(FP),POWER2(FP),POWER1(FP) ; compute the exponent
          C8 AD    B0 AD  98 AD  8D  1899  4469          XORB3   SIGN3(FP),SIGN2(FP),SIGN1(FP) ; compute the sign
                   CF AD         7C  18A0  4470          CLRQ    FRACTION1(FP)           ; extend the quotient to 128 bits
                09 DB AD   1F    E0  18A3  4471          BBS     #31,FRACTION1+12(FP),3$ ; the result is normalized - bypass
             DB AD  DB AD  01    78  18A8  4472          ASHL    #1,FRACTION1+12(FP),FRACTION1+12(FP) ; normalize the quotient
                   CB AD         D7  18AE  4473          DECL    POWER1(FP)              ; increment the exponent
                05              05  18B1  4474 3$:      RSB                             ; return
                                18B2  4475          ;
```

```
                                                M 5
                                18B2  4477          .SBTTL  DIVIDE_DGFLOAT
                                18B2  4478
                                18B2  4479          ;       Discussion
                                18B2  4480          ;
                                18B2  4481          ;           This routine computes the quotient of two floating values
                                18B2  4482          ;       in the internal representation. The fractions of the two input
                                18B2  4483          ;       values consist of the high order 64 bits of the two operand
                                18B2  4484          ;       fractions and the fraction of the quotient consists of the
                                18B2  4485          ;       high order 63 or 64 bits of the exact quotient. The remaining
                                18B2  4486          ;       bits of the fraction are set to zero. If the divisor is zero
                                18B2  4487          ;       then a floating division by zero fault is signaled.
                                18B2  4488          ;
                                18B2  4489 DIVIDE_DGFLOAT:
                   03 97 AD     E9  18B2  4490          BLBC    ZERO3(FP),1$                    ; divisor is not zero - skip
                        0400    31  18B6  4491          BRW     DIVIDE_FAULT                    ; process the floating divide fault
                C7 AD    01     D0  18B9  4492 1$:      MOVL    #1,OPERAND1(FP)                 ; mark the result as zero
                43 AF AD        E8  18BD  4493          BLBS    ZERO2(FP),3$                    ; dividend is zero - bypass
BF AD   BF AD    FF 8F          79  18C1  4494          ASHQ    #-1,FRACTION2+8(FP),FRACTION2+8(FP) ; normalize for division
        00 C3 AD  1F            E5  18C8  4495          BBCC    #31,FRACTION2+12(FP),2$         ; clear the high order bit
                  B3 AD         D6  18CD  4496 2$:      INCL    POWER2(FP)                      ; increment the exponent
                50    02        D0  18D0  4497          MOVL    #2,R0                           ; R0 = number of longwords in divisor
              51    B7 AD       9E  18D3  4498          MOVAB   FRACTION2(FP),R1                ; R1 = dividend location
              52    A7 AD       9E  18D7  4499          MOVAB   FRACTION3+8(FP),R2              ; R2 = divisor location
              53    D7 AD       9E  18DB  4500          MOVAB   FRACTION1+8(FP),R3              ; R3 = quotient area location
                    016D        30  18DF  4501          BSBW    DIVIDE                          ; perform the division
                  C7 AD         94  18E2  4502          CLRB    ZERO1(FP)                       ; indicate a nonzero result
CB AD   B3 AD    9B AD          C3  18E5  4503          SUBL3   POWER3(FP),POWER2(FP),POWER1(FP) ; compute the exponent
C8 AD   B0 AD    98 AD          8D  18EC  4504          XORB3   SIGN3(FP),SIGN2(FP),SIGN1(FP) ; compute the sign
                 CF AD          7C  18F3  4505          CLRQ    FRACTION1(FP)                   ; extend the quotient to 128 bits
        09 DB AD  1F            E0  18F6  4506          BBS     #31,FRACTION1+12(FP),3$         ; the result is normalized - bypass
   D7 AD   D7 AD    01          79  18FB  4507          ASHQ    #1,FRACTION1+8(FP),FRACTION1+8(FP) ; normalize the quotient
                 CB AD          D7  1901  4508          DECL    POWER1(FP)                      ; increment the exponent
                                05  1904  4509 3$:      RSB                                     ; return
                                    1905  4510          ;
```

```
                                1905  4512          .SBTTL  DIVIDE_HFLOAT
                                1905  4513
                                1905  4514          ;       Discussion
                                1905  4515          ;
                                1905  4516          ;           This routine computes the quotient of two floating values
                                1905  4517          ;       in the internal representation. The fraction of the quotient
                                1905  4518          ;       consists of the high order 127 or 128 bits of the exact
                                1905  4519          ;       quotient. If the divisor is zero, then a floating divide by
                                1905  4520          ;       zero fault is signaled.
                                1905  4521          ;
                                1905  4522  DIVIDE_HFLOAT:
          03 97 AD      E9      1905  4523          BLBC    ZERO3(FP),1$                    ; the divisor is not zero - skip
             03AD       31      1909  4524          BRW     DIVIDE_FAULT                    ; process the floating divide fault
       C7 AD    01      D0      190C  4525  1$:     MOVL    #1,OPERAND1(FP)                 ; mark the result as zero
         6A AF AD      E8      1910  4526          BLBS    ZERO2(FP),6$                    ; dividend is zero - bypass
            58 AD      7C      1914  4527          CLRQ    TEMP(FP)                        ; clear the last octaword of dividend
            60 AD      7C      1917  4528          CLRQ    TEMP+8(FP)                      ; clear the last octaword of dividend
68 AD   B7 AD   FF 8F  79      191A  4529          ASHQ    #-1,FRACTION2(FP),TEMP+16(FP) ; move last part of dividend
70 AD   BF AD   FF 8F  79      1921  4530          ASHQ    #-1,FRACTION2+8(FP),TEMP+24(FP) ; move first part of dividend
            B3 AD      D6      1928  4531          INCL    POWER2(FP)                      ; increment the exponent
       00 6C AD    1F  E5      192B  4532          BBCC    #31,TEMP+20(FP),2$              ; clear a sign extension bit
          05 BF AD      E9      1930  4533  2$:     BLBC    FRACTION2+8(FP),3$              ; should the bit have been set ?
       00 6C AD    1F  E2      1934  4534          BBSS    #31,TEMP+20(FP),3$              ; yes - set it
       00 74 AD    1F  E5      1939  4535  3$:     BBCC    #31,TEMP+28(FP),4$              ; clear the high order bit of dividend
            50    04   D0      193E  4536  4$:     MOVL    #4,R0                           ; R0 = number of longwords in divisor
          51    58 AD   9E      1941  4537          MOVAB   TEMP(FP),R1                     ; R1 = location of dividend
          52    9F AD   9E      1945  4538          MOVAB   FRACTION3(FP),R2                ; R2 = location of divisor
          53    CF AD   9E      1949  4539          MOVAB   FRACTION1(FP),R3                ; R3 = location for quotient
                00FF    30      194D  4540          BSBW    DIVIDE                          ; perform the division
                C7 AD   94      1950  4541          CLRB    ZERO1(FP)                       ; mark the result as nonzero
CB AD   B3 AD   9B AD   C3      1953  4542          SUBL3   POWER3(FP),POWER2(FP),POWER1(FP) ; compute the exponent
C8 AD   B0 AD   98 AD   8D      195A  4543          XORB3   SIGN3(FP),SIGN2(FP),SIGN1(FP) ; compute the sign
      18 DB AD    1F    E0      1961  4544          BBS     #31,FRACTION1+12(FP),6$ ; the result is normalized - bypass
   D7 AD   D7 AD    01  79      1966  4545          ASHQ    #1,FRACTION1+8(FP),FRACTION1+8(FP) ; normalize the first part
         04 D3 AD    1F  E1      196C  4546          BBC     #31,FRACTION1+4(FP),5$  ; should the low order bit be set ?
            D7 AD    01  C8      1971  4547          BISL2   #1,FRACTION1+8(FP)      ; yes - set the bit
   CF AD   CF AD    01  79      1975  4548  5$:     ASHQ    #1,FRACTION1(FP),FRACTION1(FP) ; normalize the second part
                CB AD   D7      197B  4549          DECL    POWER1(FP)                      ; decrement the exponent
                        05      197E  4550  6$:     RSB                                     ; return
                                197F  4551          ;
```

```
                                                B 6
                     197F  4553          ;
                     197F  4554          ;       NORMALIZE - Normalize a Floating Value
                     197F  4555          ;
                     197F  4556          ;               entered by subroutine branching
                     197F  4557          ;
                     197F  4558          ;               parameter:      OPERAND1 = The Unnormalized Value
                     197F  4559          ;
                     197F  4560          ;               returns with    OPERAND1 = The Normalized Result
                     197F  4561          ;
                     197F  4562          ;       Discussion
                     197F  4563          ;
                     197F  4564          ;           This routine normalizes a floating value in the internal
                     197F  4565          ;       representation so that the high order bit of the fraction is
                     197F  4566          ;       one. This is done by locating the high order significant bit
                     197F  4567          ;       of the fraction and then by shifting the fraction so that the
                     197F  4568          ;       bit appears in the proper position. The shift count is
                     197F  4569          ;       subtracted from the exponent so the value does not change.
                     197F  4570          ;
                     197F  4571  NORMALIZE:                                  ; entrance
        3A C7 AD  E8 197F  4572          BLBS    ZERO1(FP),2$               ; the value is zero - return
           DB AD  D5 1983  4573  1$:      TSTL    FRACTION1+12(FP)           ; test the first longword
              35  19 1986  4574          BLSS    2$                         ; value is already normalized - return
              34  13 1988  4575          BEQL    3$                         ; first longword is zero - bypass
              51  D4 198A  4576          CLRL    R1                         ; set intitial shoft count
     50       DB AD  D0 198C  4577          MOVL    FRACTION1+12(FP),R0        ; get high longword of fraction
              51  D6 1990  4578  11$:     INCL    R1                         ; increment shift count
  50  50          01  78 1992  4579          ASHL    #1,R0,R0                   ; shift one more bit
              F8  1C 1996  4580          BVC     11$                        ; repeat until high bit set
        50    51  CE 1998  4581          MNEGL   R1,R0                      ; R0 = -shift count
                     199B  4582  ;*
                     199B  4583  ; The previous six instructions are a replacement for the following four
                     199B  4584  ; instructions.  We can no longer guarantee that CVTLD is supported.
                     199B  4585  ;
                     199B  4586  ;           CVTLD   FRACTION1+12(FP),R0        ; use hardware to find shift count
                     199B  4587  ;           EXTZV   #7,#6,R0,R0                ; R0 = high order bit position + 1
                     199B  4588  ;           SUBL2   #32,R0                     ; R0 = - shift count
                     199B  4589  ;           MNEGL   R0,R1                      ; R1 = shift count
                     199B  4590  ;*
D7 AD    D7 AD    51  79 199B  4591          ASHQ    R1,FRACTION1+8(FP),FRACTION1+8(FP) ; position first longword
D7 AD    D7 AD    50  78 19A1  4592          ASHL    R0,FRACTION1+8(FP),FRACTION1+8(FP) ; get second parts together
D3 AD    D3 AD    51  79 19A7  4593          ASHQ    R1,FRACTION1+4(FP),FRACTION1+4(FP) ; position second longword
D3 AD    D3 AD    50  78 19AD  4594          ASHL    R0,FRACTION1+4(FP),FRACTION1+4(FP) ; get other parts together
CF AD    CF AD    51  79 19B3  4595          ASHQ    R1,FRACTION1(FP),FRACTION1(FP) ; position last two longwords
         CB AD    51  C2 19B9  4596          SUBL2   R1,POWER1(FP)              ; increment the exponent
                  05  19BD  4597  2$:      RSB                                ; return
         CB AD    20  C2 19BE  4598  3$:      SUBL2   #32,POWER1(FP)             ; decrement the exponent
              D7  AD  D5 19C2  4599          TSTL    FRACTION1+8(FP)            ; test the second longword
                  0F  13 19C5  4600          BEQL    4$                         ; it's zero - bypass
      D7 AD    D3 AD  7D 19C7  4601          MOVQ    FRACTION1+4(FP),FRACTION1+8(FP) ; shift first two longwords
      D3 AD    CF AD  D0 19CC  4602          MOVL    FRACTION1(FP),FRACTION1+4(FP) ; shift third longword
               CF AD  D4 19D1  4603          CLRL    FRACTION1(FP)              ; clear the final longword
                  AD  11 19D4  4604          BRB     1$                         ; finish up
         CB AD    20  C2 19D6  4605  4$:      SUBL2   #32,POWER1(FP)             ; decrement the exponent
              D3  AD  D5 19DA  4606          TSTL    FRACTION1+4(FP)            ; test the third longword
                  0A  13 19DD  4607          BEQL    5$                         ; it's zero - bypass
      D7 AD    CF AD  7D 19DF  4608          MOVQ    FRACTION1(FP),FRACTION1+8(FP) ; position first two longwords
               CF AD  7C 19E4  4609          CLRQ    FRACTION1(FP)              ; clear the last two longwords
```

```
                9A   11  19E7  4610                BRB     1$                              ; finish up
          CB AD 20   C2  19E9  4611  5$:           SUBL2   #32,POWER1(FP)                  ; decrement the exponent
             CF AD   D5  19ED  4612                TSTL    FRACTION1(FP)                   ; test the fourth longword
                0E   13  19F0  4613                BEQL    6$                              ; it's zero - bypass
       DB AD    CF AD   D0  19F2  4614                MOVL    FRACTION1(FP),FRACTION1+12(FP) ; position the first longword
             D7 AD   D4  19F7  4615                CLRL    FRACTION1+8(FP)                 ; clear the second longword
             CF AD   7C  19FA  4616                CLRQ    FRACTION1(FP)                   ; clear the last two longwords
              FF83   31  19FD  4617                BRW     1$                              ; finish up
          C7 AD 01   D0  1A00  4618  6$:           MOVL    #1,OPERAND1(FP)                 ; mark the value as zero
                05   1A04  4619                RSB                                     ; return
                     1A05  4620                ;
```

```
1A05  4622
1A05  4623          ;       MULTIPLY - Unsigned Multiple Length Integer Multiply
1A05  4624          ;
1A05  4625          ;               entered by subroutine branching
1A05  4626          ;
1A05  4627          ;               parameters:     R0 = Size of Inputs in Longwords
1A05  4628          ;                               R1 = Location of First Factor
1A05  4629          ;                               R2 = Location of Second Factor
1A05  4630          ;                               R3 = Location of Destination Area
1A05  4631          ;
1A05  4632          ;       Discussion
1A05  4633          ;
1A05  4634          ;           This routine computes the product of two multiple length
1A05  4635          ;       unsigned integers and stores the unsigned product in a
1A05  4636          ;       specified area. The parameter R0 contains the number of
1A05  4637          ;       longwords in each of the input factors and the number of
1A05  4638          ;       longwords in the product is twice the value of R0. The
1A05  4639          ;       parameters R1 and R2 contain the locations of the input
1A05  4640          ;       factors and the parameter R3 contains the location of the
1A05  4641          ;       area for the product.
1A05  4642          ;
1A05  4643          ;
1A05  4644          ;       The Algorithm
1A05  4645          ;
1A05  4646          ;           The algorithm used is perfectly straightforward. The
1A05  4647          ;       second factor is multiplied by each of the unsigned longwords
1A05  4648          ;       of the first factor and the aligned products are added to the
1A05  4649          ;       destination area. The multiplications and additions are
1A05  4650          ;       performed by a series of EMUL instructions in which the two
1A05  4651          ;       factors are longwords from each of the factors and the added
1A05  4652          ;       operand is formed by adding target longword of the result area
1A05  4653          ;       and from carryover information from the previous iteration
1A05  4654          ;       of the EMUL loop. The carryover information is formed by
1A05  4655          ;       adding the high order bits of the previous product, the
1A05  4656          ;       opposite factor for each factor longword which is negative
1A05  4657          ;       from the previous multiplication, a one if the previous
1A05  4658          ;       carryover longword was negative, and a one if the previous
1A05  4659          ;       additive operand was negative. Most of these contributions are
1A05  4660          ;       compensations for the fact that the EMUL instruction assumes
1A05  4661          ;       that the operands are signed.
1A05  4662          ;
1A05  4663          ;           To show that the algorithm is correct the major step is
1A05  4664          ;       to show that the EMUL operations do not loose information
1A05  4665          ;       because of overflow. To show this we make use of the fact that
1A05  4666          ;       for two's complemented addition and multiplication information
1A05  4667          ;       only moves in the direction of the high order bits. Therefore,
1A05  4668          ;       if the product fits into two unsigned longwords it is correct.
1A05  4669          ;       The unsigned information input to each EMUL step is
1A05  4670          ;
1A05  4671          ;               first factor <= 2^32-1
1A05  4672          ;               second factor <= 2^32-1
1A05  4673          ;               longword from result area <= 2^32-1
1A05  4674          ;               unsigned carryover information <= 2^32-1
1A05  4675          ;
1A05  4676          ;       The output of the step is the product of the first two values
1A05  4677          ;       plus the second two values. This does not exceed
1A05  4678          ;
```

```
                                1A05  4679          ;               (2^32-1)^2+2*(2^32-1) = 2^64-1
                                1A05  4680          ;
                                1A05  4681          ;       which is representable in two unsigned longwords so the
                                1A05  4682          ;       output carryover information fits into an unsigned longword.
                                1A05  4683          ;       Consequently no information can be lost during the EMUL steps.
                                1A05  4684          ;
                                1A05  4685          ;
                                1A05  4686 MULTIPLY:;                                       ; entrance
                          54    D4    1A05  4687          CLRL    R4                        ; clear the loop index
                        6344    D4    1A07  4688 1$:      CLRL    (R3)[R4]                  ; clear a longword of the result
                   F9   54 50   F2    1A0A  4689          AOBLSS  R0,R4,1$                  ; more longwords to clear - loop
                          54    D4    1A0E  4690          CLRL    R4                        ; clear the loop index
                          55    D4    1A10  4691 2$:      CLRL    R5                        ; clear the inner loop index
                       56 81    D0    1A12  4692          MOVL    (R1)+,R6                  ; R6 = next value from first factor
                       57 83    DE    1A15  4693          MOVAL   (R3)+,R7                  ; R7 = area of product being affected
                          59    D4    1A18  4694          CLRL    R9                        ; clear the carryover information
                          5A    D4    1A1A  4695 3$:      CLRL    R10                       ; clear the cell to take the carry
                       59 67    C0    1A1C  4696          ADDL2   (R7),R9                   ; include value so far in carryover
                       5A 00    D8    1A1F  4697          ADWC    #0,R10                    ; remember the carry
                    58   6245    D0    1A22  4698          MOVL    (R2)[R5],R8               ; R8 = next value from second factor
                          03    18    1A26  4699          BGEQ    4$                        ; it's not negative - skip
                       5A 56    C0    1A28  4700          ADDL2   R6,R10                    ; make unsigned by adding other factor
                   03  56 1F    E1    1A2B  4701 4$:      BBC     #31,R6,5$                 ; first factor is not negative - skip
                       5A 58    C0    1A2F  4702          ADDL2   R8,R10                    ; make unsigned by adding other factor
                   02  59 1F    E1    1A32  4703 5$:      BBC     #31,R9,6$                 ; carryover is not negative - skip
                          5A    D6    1A36  4704          INCL    R10                       ; make unsigned by adding one
             58    59    58 56  7A    1A38  4705 6$:      EMUL    R6,R8,R9,R8               ; multiply factors including carryover
                       87 58    D0    1A3D  4706          MOVL    R8,(R7)+                  ; accumulate the product
                       59 5A    C0    1A40  4707          ADDL2   R10,R9                    ; form the new carryover
                   D3  55 50    F2    1A43  4708          AOBLSS  R0,R5,3$                  ; partial product not complete - loop
                       67 59    D0    1A47  4709          MOVL    R9,(R7)                   ; store the last part of the product
                   C2  54 50    F2    1A4A  4710          AOBLSS  R0,R4,2$                  ; full product not complete - loop
                                05    1A4E  4711          RSB                               ; return
                                      1A4F  4712          ;
```

```
                                                F 6
1A4F  4714          ;
1A4F  4715          ;       DIVIDE - Unsigned Multiple Length Integer Divide
1A4F  4716          ;
1A4F  4717          ;               entered by subroutine branching
1A4F  4718          ;
1A4F  4719          ;               parameters:     R0 = Size of the Divisor in Longwords
1A4F  4720          ;                               R1 = Location of the Dividend
1A4F  4721          ;                               R2 = Location of the Divisor
1A4F  4722          ;                               R3 = Location of the Quotient Area
1A4F  4723          ;
1A4F  4724          ;       Discussion
1A4F  4725          ;
1A4F  4726          ;           This routine performs unsigned multiple length division
1A4F  4727          ;       and develops a quotient and a remainder. The number of
1A4F  4728          ;       longwords in the divisor and in the quotient area is given
1A4F  4729          ;       by the parameter R0. The number of longwords in the dividend
1A4F  4730          ;       area is twice the value of R0. The parameter R1 is the
1A4F  4731          ;       location of the dividend, R2 is the location of the divisor,
1A4F  4732          ;       and R3 is the location for the quotient. The remainder is
1A4F  4733          ;       formed in the low order R0 longwords of the dividend area.
1A4F  4734          ;       It is assumed that the high order bit of the divisor is one
1A4F  4735          ;       and that the high order bit of the dividend is zero. These
1A4F  4736          ;       conditions insure that the quotient will fit into the quotient
1A4F  4737          ;       area.
1A4F  4738          ;
1A4F  4739          ;
1A4F  4740          ;       The Algorithm
1A4F  4741          ;
1A4F  4742          ;           The algorithm used is a variation of the classical divide
1A4F  4743          ;       and correct method which has been adapted for the above
1A4F  4744          ;       specifications. The algorithm has been implemented using words
1A4F  4745          ;       rather than longwords because the word version is much easier
1A4F  4746          ;       to verify for correctness since there are no problems with
1A4F  4747          ;       using signed arithmetic operations to perform unsigned
1A4F  4748          ;       arithmetic.
1A4F  4749          ;
1A4F  4750          ;           We let A[0..2N-1] denote the dividend as an array of words
1A4F  4751          ;       with A[0] being the low order word, and N being twice the
1A4F  4752          ;       value of R0. Similarly we let B[0..N-1] denote the divisor and
1A4F  4753          ;       C[0..N-1] the quotient. The remainder will be developed in the
1A4F  4754          ;       array A[0..N-1]. The algorithm may be given as follows:
1A4F  4755          ;
1A4F  4756          ;
1A4F  4757          ;               Step 0. Let I = N.
1A4F  4758          ;
1A4F  4759          ;               Step 1. Let I = I-1 and A[2N] = 0.
1A4F  4760          ;
1A4F  4761          ;                       { The use of A[2N] is only a simplification,
1A4F  4762          ;                         it does not really appear in the program. }
1A4F  4763          ;
1A4F  4764          ;               Step 2. Let Q = entier(A[I+N-1..I+N+1]/(B[N-1]+1)) and
1A4F  4765          ;                       let R be the remainder from the division.
1A4F  4766          ;
1A4F  4767          ;                       { This operation is performed by a single
1A4F  4768          ;                         EDIV instruction. }
1A4F  4769          ;
1A4F  4770          ;               Step 3. Let X = b*B[N-1]+B[N-2]+1 and
```

```
                                1A4F  4771          ;
                                1A4F  4772          ;                       Y = -Q*B[N-2]+b*R+A[I+N-2]
                                1A4F  4773          ;                         =  Q*(b-B[N-2]-1)+b*R+A[I+N-2].
                                1A4F  4774          ;
                                1A4F  4775          ;                       { Here b denotes 2^16 which is the "base" of
                                1A4F  4776          ;                         the number system we are using and b-B[N-2]-1
                                1A4F  4777          ;                         is the ones complement of -B[N-2]. These
                                1A4F  4778          ;                         values are used to correct the quotient. }
                                1A4F  4779          ;
                                1A4F  4780          ;                   Step 4. Let Y = Y-X. If  Y >= 0 then let Q = Q+1 and
                                1A4F  4781          ;                           repeat this step.
                                1A4F  4782          ;
                                1A4F  4783          ;                           { When this step is complete the value of Q is
                                1A4F  4784          ;
                                1A4F  4785          ;                             entier(A[I+N-2..I+N+1]/(B[N-2..N-1]+1) }
                                1A4F  4786          ;
                                1A4F  4787          ;                   Step 5. Let A[I..I+N] = A[I..I+N]-Q*B[0..N-1].
                                1A4F  4788          ;
                                1A4F  4789          ;                   Step 6. Let C[I] = Q and if Q is too large then add
                                1A4F  4790          ;                           the overflow to C[I+1..N-1]. If I > 2 then
                                1A4F  4791          ;                           go to Step 2.
                                1A4F  4792          ;
                                1A4F  4793          ;                   Step 7. If B[0..N-1] <= A[0..N] then let
                                1A4F  4794          ;                           C[0..N-1] = C[0..N-1]+1 and
                                1A4F  4795          ;                           A[0..N-1] = A[0..N-1]-B[0..N-1].
                                1A4F  4796          ;                           The division is complete.
                                1A4F  4797          ;
                                1A4F  4798 DIVIDE:  ;
     50   50   01   78  1A4F  4799          ASHL    #1,R0,R0            ; entrance
                                                                        ; convert the longword count to words
        51   6140   3E  1A53  4800          MOVAW   (R1)[R0],R1         ; R1 = current position in dividend
        54   6240   3E  1A57  4801          MOVAW   (R2)[R0],R4         ; R4 = position above divisor
        53   6340   3E  1A5B  4802          MOVAW   (R3)[R0],R3         ; R3 = position above quotient area
        59    FE A4   3C  1A5F  4803          MOVZWL  -2(R4),R9           ; R9 = leading word of divisor
                 59   D6  1A63  4804          INCL    R9                  ; form the trial divisor
        5A    FC A4   D0  1A65  4805          MOVL    -4(R4),R10          ; R10 = leading longword of divisor
                 5A   D6  1A69  4806          INCL    R10                 ; form the correction divisor
        5B    FC A4   B2  1A6B  4807          MCOMW   -4(R4),R11          ; R11 = complemented second word
           5B    5B   3C  1A6F  4808          MOVZWL  R11,R11             ; form the correction multiplier
           54    50   D0  1A72  4809          MOVL    R0,R4               ; R4 = loop counter
                 56   D4  1A75  4810          CLRL    R6                  ; clear the carryover information
           51    02   C2  1A77  4811 ONE$:    SUBL2   #2,R1               ; R1 = current position in dividend
           53    02   C2  1A7A  4812          SUBL2   #2,R3               ; R3 = current position in quotient
        55   6140   3E  1A7D  4813          MOVAW   (R1)[R0],R5         ; R5 = location of next dividend word
        55    FE A5   D0  1A81  4814          MOVL    -2(R5),R5           ; R5 = leading longword of dividend
56   55   55   59   7B  1A85  4815          EDIV    R9,R5,R5,R6         ; R5 = quotient, R6 = remainder
                 58   D4  1A8A  4816          CLRL    R8                  ; clear the double length adjustment
     56   56   10   78  1A8C  4817          ASHL    #16,R6,R6           ; position the remainder
                 02   18  1A90  4818          BGEQ    2$                  ; is the value negative ?
                 58   D6  1A92  4819          INCL    R8                  ; yes - the sum will need adjusting
     56    FC A140  B0  1A94  4820 2$:      MOVW    -4(R1)[R0],R6       ; include next word from dividend
56   56   5B   55   7A  1A99  4821          EMUL    R5,R11,R6,R6        ; form the correction remainder
        57   58   C0  1A9E  4822          ADDL2   R8,R7               ; adjust the second longword
                 58   D4  1AA1  4823 3$:      CLRL    R8                  ; clear the subtraction adjustment
        02 5A   1F   E0  1AA3  4824          BBS     #31,R10,4$          ; is the correction divisor negative ?
                 58   D6  1AA7  4825          INCL    R8                  ; yes - difference will need adjusting
           56   5A   C2  1AA9  4826 4$:      SUBL2   R10,R6              ; subtract the first longwords
           57   58   D9  1AAC  4827          SBWC    R8,R7               ; subtract the second longwords
```

```
                                                H 6
                     04   19  1AAF  4828          BLSS    5$                      ; result is negative - bypass
                     55   D6  1AB1  4829          INCL    R5                      ; correct the quotient
                     EC   11  1AB3  4830          BRB     3$                      ; try for another correction
                  56 53   D0  1AB5  4831  5$:     MOVL    R3,R6                   ; R6 = current word in quotient
                     66   B4  1AB8  4832          CLRW    (R6)                    ; clear the word
                  86 55   C0  1ABA  4833          ADDL2   R5,(R6)+                ; add the current quotient into it
                  86 00   D8  1ABD  4834  6$:     ADWC    #0,(R6)+                ; propagate any carry
                     FB   1F  1AC0  4835          BCS     6$                      ; another carry - loop
                  55 55   CE  1AC2  4836          MNEGL   R5,R5                   ; negate the quotient
                     56   D4  1AC5  4837          CLRL    R6                      ; clear the carryover information
                     58   D4  1AC7  4838          CLRL    R8                      ; clear the loop index
               57  6148   3C  1AC9  4839  7$:     MOVZWL  (R1)[R8],R7             ; R7 = next word from dividend
                  56 57   C0  1ACD  4840          ADDL2   R7,R6                   ; add it to the carryover
               57  6248   3C  1AD0  4841          MOVZWL  (R2)[R8],R7             ; R7 = next word from divisor
            56 56 57 55   7A  1AD4  4842          EMUL    R5,R7,R6,R6             ; form next word of dividend
              6148 56     B0  1AD9  4843          MOVW    R6,(R1)[R8]             ; store it
         56 56    F0 8F   79  1ADD  4844          ASHQ    #-16,R6,R6              ; position the carryover information
              E3 58 50    F2  1AE2  4845          AOBLSS  R0,R8,7$                ; remainder is not complete - loop
               57  6140   3C  1AE6  4846          MOVZWL  (R1)[R0],R7             ; R7 = high order word of remainder
                  56 57   C0  1AEA  4847          ADDL2   R7,R6                   ; add it to the carryover
                     02   18  1AED  4848          BGEQ    GO_ON
                     56   D4  1AEF  4849          CLRL    R6                      ; the carry can not be negative
                          1AF1  4850  GO_ON:
                  83 54   F5  1AF1  4851          SOBGTR  R4,ONE$                 ; division is not complete - loop
            50 50 FF 8F   78  1AF4  4852          ASHL    #-1,R0,R0               ; restore the count to longwords
                     56   D5  1AF9  4853          TSTL    R6                      ; is the carryover nonzero ?
                     10   12  1AFB  4854          BNEQ    9$                      ; yes - bypass
                  54 50   01  C3  1AFD  4855          SUBL3   #1,R0,R4                ; R4 = loop index
                6244  6144   D1  1B01  4856  8$:     CMPL    (R1)[R4],(R2)[R4]       ; compare remainder and divisor
                     17   1F  1B06  4857          BLSSU   12$                     ; less than - bypass
                     03   1A  1B08  4858          BGTRU   9$                      ; greater than - skip
                  F4 54   F4  1B0A  4859          SOBGEQ  R4,8$                   ; more longwords to compare - loop
                     54   D4  1B0D  4860  9$:     CLRL    R4                      ; clear the loop index
                     01   B9  1B0F  4861          BICPSW  #PSLM_C                 ; clear the carry bit
                  81 82   D9  1B11  4862  10$:    SBWC    (R2)+,(R1)+             ; subtract corresponding longwords
               F9 54 50   F2  1B14  4863          AOBLSS  R0,R4,10$               ; more lonwords to subtract - loop
                     83   D6  1B18  4864          INCL    (R3)+                   ; increment the quotient
                  83 00   D8  1B1A  4865  11$:    ADWC    #0,(R3)+                ; propagate any carry
                     FB   1F  1B1D  4866          BCS     11$                     ; another carry to propagate - loop
                     05   1B1F  4867  12$:    RSB                             ; return
                          1B20  4868          ;
```

```
1B20  4870          ;*****************************************************************************
1B20  4871          ;*                                                                           *
1B20  4872          ;*                                                                           *
1B20  4873          ;*                    Condition Code Processing Routines                     *
1B20  4874          ;*                                                                           *
1B20  4875          ;*                                                                           *
1B20  4876          ;*****************************************************************************
1B20  4877          ;
1B20  4878          ;
1B20  4879          ;       Introduction
1B20  4880          ;       ------------
1B20  4881          ;
1B20  4882          ;           In order that condition code information be usable
1B20  4883          ;       directly by the code of the Emulator as well as be available
1B20  4884          ;       for use in the emulated PSL, the routines which perform tests
1B20  4885          ;       and compares set the hardware condition codes. The routine
1B20  4886          ;       SET_CONDITION is used to move the hardware condition codes
1B20  4887          ;       to the emulated PSL.
1B20  4888          ;
1B20  4889          ;           The routines of this portion of the Emulator are extremely
1B20  4890          ;       simple so the descriptions of routines are included with the
1B20  4891          ;       routines. Here we will just discuss the general policy on
1B20  4892          ;       condition codes within the Emulator.
1B20  4893          ;
1B20  4894          ;
1B20  4895          ;       General Policy on Condition Codes
1B20  4896          ;       ---------------------------------
1B20  4897          ;
1B20  4898          ;           In general it is the responsibility of each of the
1B20  4899          ;       instruction emulation routines to insure that the condition
1B20  4900          ;       codes are correct. Since for most of the instructions
1B20  4901          ;       presently emulated the condition codes reflect the floating
1B20  4902          ;       value that appears in OPERAND1, the routine SET_CONDITION1
1B20  4903          ;       is available for checking the value and setting the condition
1B20  4904          ;       codes in the emulated PSL. In other cases these operations
1B20  4905          ;       must be done in line.
1B20  4906          ;
1B20  4907          ;           For those instructions which return an integer result
1B20  4908          ;       the V bit is used to indicate whether or not an integer
1B20  4909          ;       overflow took place. Because of this it is also checked in
1B20  4910          ;       NORMAL_EXIT in order to determine if a integer overflow trap
1B20  4911          ;       should be signaled. For this reason it is cleared when the
1B20  4912          ;       instruction emulation is started in EMULATOR.
1B20  4913          ;
```

```
                     1B20  4915          ;
                     1B20  4916          ;       SET_CONDITION - Capture the Condition Codes N and Z in the PSL
                     1B20  4917          ;
                     1B20  4918          ;               entered by subroutine branching
                     1B20  4919          ;
                     1B20  4920          ;               no parameters
                     1B20  4921          ;
                     1B20  4922          ;       Discussion
                     1B20  4923          ;
                     1B20  4924          ;           This routine sets the N and Z bits in the emulated PSL
                     1B20  4925          ;       equal to those bits in PSL available on entry to the routine.
                     1B20  4926          ;
                     1B20  4927 SET_CONDITION:                               ; entrance
        0B   19      1B20  4928          BLSS    1$                          ; less than - bypass
        12   14      1B22  4929          BGTR    2$                          ; greater than - bypass
54 AD   0C   CA      1B24  4930          BICL2   #PSLM_NZ,PSL(FP)            ; clear the N bit and Z bit in the PSL
54 AD   04   C8      1B28  4931          BISL    #PSLM_EQL,PSL(FP)           ; specify equals in the PSL
             05      1B2C  4932          RSB                                 ; return
54 AD   0C   CA      1B2D  4933 1$:      BICL2   #PSLM_NZ,PSL(FP)            ; clear the N bit and Z bit in the PSL
54 AD   08   C8      1B31  4934          BISL    #PSLM_LSS,PSL(FP)           ; specify less than in the PSL
             05      1B35  4935          RSB                                 ; return
54 AD   0C   CA      1B36  4936 2$:      BICL2   #PSLM_NZ,PSL(FP)            ; clear the N bit and Z bit in the PSL
54 AD   00   C8      1B3A  4937          BISL2   #PSLM_GTR,PSL(FP)           ; specify greater than in the PSL
             05      1B3E  4938          RSB                                 ; return
                     1B3F  4939          ;
                     1B3F  4940          ;       SET_CONDITION1 - Test a Floating Value Setting the PSL
                     1B3F  4941          ;
                     1B3F  4942          ;               entered by subroutine branching
                     1B3F  4943          ;
                     1B3F  4944          ;               parameter:      OPERAND1 = The Floating Value
                     1B3F  4945          ;
                     1B3F  4946          ;       Discussion
                     1B3F  4947          ;
                     1B3F  4948          ;           This routine tests the floating value in OPERAND1 and
                     1B3F  4949          ;       sets the N and Z bits in the emulated PSL according to the
                     1B3F  4950          ;       outcome of the test. The V and C bits in the emulated PSL
                     1B3F  4951          ;       are cleared.
                     1B3F  4952          ;
                     1B3F  4953 SET_CONDITION1:                              ; entrance
51   C7 AD   9E      1B3F  4954          MOVAB   OPERAND1(FP),R1             ; R1 = location of OPERAND1
             07      1B43  4955          BSBB    TEST_REAL                   ; test the value
             D9      1B45  4956          BSBB    SET_CONDITION               ; set the condition codes
54 AD   03   CA      1B47  4957          BICL2   #PSLM_VC,PSL(FP)            ; clear the V bit and C bit in the PSL
             05      1B4B  4958          RSB
                     1B4C  4959          ;
```

```
                                K 6
                        1B4C  4961          ;
                        1B4C  4962          ;       TEST_REAL - Test all Floating Types
                        1B4C  4963          ;
                        1B4C  4964          ;               entered by subroutine branching
                        1B4C  4965          ;
                        1B4C  4966          ;               parameter:      R1 = Floating Value Location
                        1B4C  4967          ;
                        1B4C  4968          ;       Discussion
                        1B4C  4969          ;
                        1B4C  4970          ;           This routine tests the floating value in the internal
                        1B4C  4971          ;       representation that is addressed by R1 and sets the hardware
                        1B4C  4972          ;       condition codes accordingly. These settings are available
                        1B4C  4973          ;       when the routine returns.
                        1B4C  4974          ;
                        1B4C  4975 TEST_REAL:                           ; entrance
    56 61    E8         1B4C  4976          BLBS    ZERO(R1),TEST_EQL   ; value is zero - bypass
55 01 A1     E9         1B4F  4977          BLBC    SIGN(R1),TEST_GTR   ; value is positive - bypass
      4C     11         1B53  4978          BRB     TEST_LSS            ; value is negative - bypass
                        1B55  4979          ;
                        1B55  4980          ;       TEST_OCTA - Test an Octaword in OPERAND1
                        1B55  4981          ;
                        1B55  4982          ;               entered by subroutine branching
                        1B55  4983          ;
                        1B55  4984          ;               parameter:      OPERAND1 = 128 Bit Octaword Value
                        1B55  4985          ;
                        1B55  4986          ;       Discussion
                        1B55  4987          ;
                        1B55  4988          ;           This routine tests the octaword value which occupies
                        1B55  4989          ;       128 bits and starts at OPERAND1 (it is not in the internal
                        1B55  4990          ;       floating representation) and sets the hardware condition
                        1B55  4991          ;       codes according to the outcome of the test. These settings
                        1B55  4992          ;       are available when the routine returns.
                        1B55  4993          ;
                        1B55  4994 TEST_OCTA:                           ; entrance
50   03    D0           1B55  4995          MOVL    #3,R0               ; initialize the index
C7 AD40    D5           1B58  4996          TSTL    OPERAND1(FP)[R0]    ; test the first longword
      05   18           1B5C  4997          BGEQ    2$                  ; it's not negative - bypass
      05   1B5E  4998          RSB                                     ; return indicating negative value
C7 AD40    D5           1B5F  4999 1$:      TSTL    OPERAND1(FP)[R0]    ; test the next longword
      43   12           1B63  5000 2$:      BNEQ    TEST_GTR            ; value is positive - bypass
  F7 50    F4           1B65  5001          SOBGEQ  R0,1$               ; more longwords to examine - loop
      3B   11           1B68  5002          BRB     TEST_EQL            ; value is zero - bypass
                        1B6A  5003          ;
```

```
                                1B6A  5005          ;
                                1B6A  5006          ;       COMPARE_REAL - Compare Internal Format Floating Values
                                1B6A  5007          ;
                                1B6A  5008          ;               entered by subroutine branching
                                1B6A  5009          ;
                                1B6A  5010          ;               parameters:     R1 = Location of First Floating Value
                                1B6A  5011          ;                               R2 = Locaiton of Second Floating Value
                                1B6A  5012          ;
                                1B6A  5013          ;       Discussion
                                1B6A  5014          ;
                                1B6A  5015          ;           This routine compares the two floating values addressed
                                1B6A  5016          ;       by R1 and R2 and sets the hardware condition codes according
                                1B6A  5017          ;       to the outcome of the comparison. These settings are available
                                1B6A  5018          ;       when the routine returns.
                                1B6A  5019          ;
                                1B6A  5020 COMPARE_REAL:                        ; entrance
      01 A2    01 A1    91      1B6A  5021          CMPB    SIGN(R1),SIGN(R2)   ; compare the sign indicators
                  24    12      1B6F  5022          BNEQ    3$                  ; not equal - bypass
            61    62    91      1B71  5023          CMPB    ZERO(R2),ZERO(R1)   ; compare the zero indicators
                  1D    12      1B74  5024          BNEQ    2$                  ; not equal - bypass
               2C 61    E8      1B76  5025          BLBS    ZERO(R1),TEST_EQL   ; both are zero and equal - bypass
      04 A2    04 A1    D1      1B79  5026          CMPL    POWER(R1),POWER(R2) ; compare the exponents
                  1B    19      1B7E  5027          BLSS    4$                  ; condition was less than - bypass
                  13    14      1B80  5028          BGTR    3$                  ; condition was greater - bypass
            50    03    D0      1B82  5029          MOVL    #3,R0               ; R0 = loop index
   08 A240    08 A140    D1      1B85  5030 1$:      CMPL    FRACTION(R1)[R0],-  ; compare corresponding longwords
                                1B8C  5031                  FRACTION(R2)[R0]    ; from the fractions
                  05    12      1B8C  5032          BNEQ    2$                  ; not equal - bypass
               F4 50    F4      1B8E  5033          SOBGEQ  R0,1$               ; more longwords to examine - loop
                  12    11      1B91  5034          BRB     TEST_EQL            ; arrange an equals return
                  06    1F      1B93  5035 2$:      BLSSU   4$                  ; condition was less than - bypass
   0F 01    A1    E9      1B95  5036 3$:      BLBC    SIGN(R1),TEST_GTR   ; condition is greater than - bypass
                  06    11      1B99  5037          BRB     TEST_LSS            ; condition is less than - bypass
   02 01    A1    E9      1B9B  5038 4$:      BLBC    SIGN(R1),TEST_LSS   ; condition is less than - bypass
                  07    11      1B9F  5039          BRB     TEST_GTR            ; condition is greater than - bypass
                                1BA1  5040          ;
```

```
                                1BA1  5042          ;
                                1BA1  5043          ;       TEST_LSS - Set the Condition Codes to Specify Less Than
                                1BA1  5044          ;
                                1BA1  5045          ;               entered by subroutine branching
                                1BA1  5046          ;
                                1BA1  5047          ;               no parameters
                                1BA1  5048          ;
                                1BA1  5049          ;       Discussion
                                1BA1  5050          ;
                                1BA1  5051          ;           This routine sets the hardware condition codes to specify
                                1BA1  5052          ;       a "less than" outcome. This setting is available when the
                                1BA1  5053          ;       routine returns.
                                1BA1  5054          ;
                                1BA1  5055  TEST_LSS:
                      FF 8F  95 1BA1  5056          TSTB    #-1                     ; entrance
                                                                                    ; set the condition codes
                         05     1BA4  5057          RSB                             ; return
                                1BA5  5058          ;
                                1BA5  5059          ;       TEST_EQL - Set the Condition Codes to Specify Equals
                                1BA5  5060          ;
                                1BA5  5061          ;               entered by subroutine branching
                                1BA5  5062          ;
                                1BA5  5063          ;               no parameters
                                1BA5  5064          ;
                                1BA5  5065          ;       Discussion
                                1BA5  5066          ;
                                1BA5  5067          ;           This routine sets the hardware condition codes to specify
                                1BA5  5068          ;       an "equals" outcome. This setting is available when the
                                1BA5  5069          ;       routine returns.
                                1BA5  5070          ;
                                1BA5  5071  TEST_EQL:                               ; entrance
                      00     95 1BA5  5072          TSTB    #0                      ; set the condition codes
                         05     1BA7  5073          RSB                             ; return
                                1BA8  5074          ;
                                1BA8  5075          ;       TEST_GTR - Set the Condition Codes to Specify Greater Than
                                1BA8  5076          ;
                                1BA8  5077          ;               entered by subroutine branching
                                1BA8  5078          ;
                                1BA8  5079          ;               no parameters
                                1BA8  5080          ;
                                1BA8  5081          ;       Discussion
                                1BA8  5082          ;
                                1BA8  5083          ;           This routine sets the hardware condition codes to specify
                                1BA8  5084          ;       a "greater than" outcome. This setting is available when the
                                1BA8  5085          ;       routine returns.
                                1BA8  5086          ;
                                1BA8  5087  TEST_GTR:                               ; entrance
                      01     95 1BA8  5088          TSTB    #1                      ; set the condition codes
                         05     1BAA  5089          RSB                             ; return
                                1BAB  5090          ;
```

```
1BAB  5092 ;
1BAB  5093 ;       ************************************************************************
1BAB  5094 ;       *                                                                      *
1BAB  5095 ;       *                                                                      *
1BAB  5096 ;       *                    Exception Processing Routines                     *
1BAB  5097 ;       *                                                                      *
1BAB  5098 ;       *                                                                      *
1BAB  5099 ;       ************************************************************************
1BAB  5100 ;
1BAB  5101 ;       Introduction
1BAB  5102 ;       ------------
1BAB  5103 ;
1BAB  5104 ;           In order to simplify the design of the Emulator, it was
1BAB  5105 ;       decided that whenever fault conditions were detected during
1BAB  5106 ;       instruction emulation, that control would branch immediatly
1BAB  5107 ;       to the signaling routine rather than using status codes to
1BAB  5108 ;       inform some outer routine of the condition. Because of this
1BAB  5109 ;       some care was necessary in the ordering of operations so that
1BAB  5110 ;       the Emulator is always in the correct state when faults are
1BAB  5111 ;       detected. The only trap supported is the integer overflow
1BAB  5112 ;       trap and since this condition is only signaled when the
1BAB  5113 ;       instruction emulation is complete, there is no problem with
1BAB  5114 ;       the flow of control.
1BAB  5115 ;
1BAB  5116 ;           For each of the exceptions recognized, there is a routine
1BAB  5117 ;       which is branched to (except for access violations in which
1BAB  5118 ;       a subroutine branch is used instead) as soon as the condition
1BAB  5119 ;       is detected. This routine pushes a shortened version of the
1BAB  5120 ;       signal array onto the stack and branches to SIGNAL_START which
1BAB  5121 ;       builds the signal and mechanism arrays in the proper place in
1BAB  5122 ;       memory and enters the signal dispatcher to search for handlers
1BAB  5123 ;       to process the condition. If the exception was a fault, the
1BAB  5124 ;       routine FAULT_RESET is called to restore the registers to
1BAB  5125 ;       their values when the instruction was started.
1BAB  5126 ;
1BAB  5127 ;
1BAB  5128 ;       Access Violations
1BAB  5129 ;       -----------------
1BAB  5130 ;
1BAB  5131 ;           The routines READ_FAULT and WRITE_FAULT are called by
1BAB  5132 ;       subroutine branching when memory probes of read and write
1BAB  5133 ;       access fail during instruction emulat'   . The register R11 is
1BAB  5134 ;       assumed to contain the location of the area being probed
1BAB  5135 ;       and the register R10 is assumed to contain its length. The
1BAB  5136 ;       routine tries to produce the fault under controlled conditions
1BAB  5137 ;       and returns if it can not produce the fault. If it can produce
1BAB  5138 ;       the fault the the fault is signaled with the reason mask being
1BAB  5139 ;       the reason mask from the attempt to produce the fault and with
1BAB  5140 ;       the violation address as the address of the first byte of the
1BAB  5141 ;       area for which the access violation occurs.
1BAB  5142 ;
1BAB  5143 ;
1BAB  5144 ;       The Signal Dispatcher
1BAB  5145 ;       ---------------------
1BAB  5146 ;
1BAB  5147 ;           The Emulator presently contains all of the code necessary
1BAB  5148 ;       for signaling the condition since it is necessary that the
```

```
                                                B 7
1BAB  5149          ;       emulator build the signal and mechanism arrays itself and
1BAB  5150          ;       since there is no "back door" to either the signal dispatcher
1BAB  5151          ;       in the common RTL or the one in VMS. The version contained in
1BAB  5152          ;       the emulator is a copy with minor modifications of the one in
1BAB  5153          ;       LIB$SIGNAL.
1BAB  5154          ;
```

```
                                                C 7
                                1BAB  5156          ;
                                1BAB  5157          ;       READ_FAULT - Process a Read Access Violation Fault
                                1BAB  5158          ;
                                1BAB  5159          ;               entered by subroutine branching
                                1BAB  5160          ;
                                1BAB  5161          ;               parameters:     R10 = Size of Area being Read
                                1BAB  5162          ;                               R11 = Location of Area being Read
                                1BAB  5163          ;
                                1BAB  5164 READ_FAULT:                              ; entrance
                          07    BB    1BAB  5165          PUSHR   #^M<R0,R1,R2>     ; save R0,R1,R2
                       52 5B    D0    1BAD  5166          MOVL    R11,R2            ; R2 = probed address
                    62 01 FD AD 0C    1BB0  5167          PROBER  MODE(FP),#1,(R2)  ; is the first byte readable ?
                          13    13    1BB5  5168          BEQL    1$                ; no - bypass
                FF A24A 01 FD AD 0C    1BB7  5169          PROBER  MODE(FP),#1,-1(R2)[R10] ; is the last byte readable ?
                          0A    12    1BBE  5170          BNEQ    1$                ; yes - bypass
                52 FF A24A    9E    1BC0  5171          MOVAB   -1(R2)[R10],R2    ; R2 = address of last byte
                52 01FF 8F    AA    1BC5  5172          BICW2   #511,R2           ; compute address of first bad byte
                E6'AF 00      FB    1BCA  5173 1$:      CALLS   #0,B^READ_REASON  ; get the reason mask
                      12 50   E8    1BCE  5174          BLBS    R0,2$             ; the read went all right - bypass
                        0104  30    1BD1  5175          BSBW    FAULT_RESET       ; reinitialize registers and clear TP
                5E FB AD      9E    1BD4  5176          MOVAB   SHORT_LOCAL(FP),SP ; shorten the frame
                          52  DD    1BD8  5177          PUSHL   R2                ; push the bad address
                          51  DD    1BDA  5178          PUSHL   R1                ; push the reason mask
                          0C  DD    1BDC  5179          PUSHL   #SS$_ACCVIO       ; push the condition code
                          03  DD    1BDE  5180          PUSHL   #3                ; push the number of arguments
                        010B  31    1BE0  5181          BRW     SIGNAL_START      ; signal the condition
                          07  BA    1BE3  5182 2$:      POPR    #^M<R0,R1,R2>     ; restore R0,R1,R2
                          05  1BE5  5183                RSB                       ; get back
                              1BE6  5184          ;
                              1BE6  5185          ;       READ_REASON - Get the Reason Mask for a Read Access Violation
                              1BE6  5186          ;
                              1BE6  5187          ;               parameter:      R2 = Address for which Probe Failed
                              1BE6  5188          ;
                              1BE6  5189          ;               returns with    R0 = Status of Access Attempt
                              1BE6  5190          ;                               R1 = Reason Mask if Unsuccessful
                              1BE6  5191          ;
                              1BE6  5192 READ_REASON:                             ; entrance
                        0000  1BE6  5193                .WORD   0                 ; entry mask
                6D F2'AF      9E    1BE8  5194          MOVAB   B^REASON_HANDLER,(FP) ; set up the condition handler
                          62  95    1BEC  5195          TSTB    (R2)              ; touch the location
                       50 01  D0    1BEE  5196          MOVL    #1,R0             ; indicate a successful read
                              04    1BF1  5197          RET                       ; return
                              1BF2  5198          ;
                              1BF2  5199          ;       REASON_HANDLER - Condition Handler for Reason Routines
                              1BF2  5200          ;
                              1BF2  5201          ;               parameters:     P1 = Signal Array Location
                              1BF2  5202          ;                               P2 = Mechanism Array Location
                              1BF2  5203          ;
                              1BF2  5204          ;               returns with    R0 = Condition Response
                              1BF2  5205          ;
                              1BF2  5206 REASON_HANDLER:                          ; entrance
                        0000  1BF2  5207                .WORD   0                 ; entry mask
                50 04 AC      7D    1BF4  5208          MOVQ    4(AP),R0          ; R0 and R1 = location of arrays
                   08 A1      D5    1BF8  5209          TSTL    8(R1)             ; condition from establisher frame ?
                          19  12    1BFB  5210          BNEQ    1$                ; no - bypass
                              1BFD  5211          CMPCOND SS$_ACCVIO,4(R0)          ; access violation condition ?
                          11  12    1C03  5212          BNEQ    1$                ; no - bypass
```

```
                                                D 7
                 0C A1   D4   1C05  5213          CLRL    12(R1)                  ; return zero status in R0
           10 A1 08 A0   D0   1C08  5214          MOVL    8(R0),16(R1)            ; return the reason mask in R1
                 7E      7C   1C0D  5215          CLRQ    -(SP)                   ; default PC and level for unwind
     00000000'GF   02      FB   1C0F  5216          CALLS   #2,G^EXE$UNWIND         ; unwind the reason routine frame
          50 0918  8F      32   1C16  5217 1$:      CVTWL   #SS$_RESIGNAL,R0        ; specify condition not handled
                           04   1C1B  5218          RET                             ; return
                                1C1C  5219          ;
```

```
                                1C1C  5221          ;
                                1C1C  5222          ;       WRITE_FAULT - Process a Write Access Violation Fault
                                1C1C  5223          ;
                                1C1C  5224          ;               entered by subroutine branching
                                1C1C  5225          ;
                                1C1C  5226          ;               parameters:     R10 = Size of Area being Written
                                1C1C  5227          ;                               R11 = Location of Area being Written
                                1C1C  5228          ;
                                1C1C  5229 WRITE_FAULT:                                 ; entrance
                          07    BB    1C1C  5230          PUSHR   #^M<R0,R1,R2>         ; save R0,R1,R2
                       52 5B    D0    1C1E  5231          MOVL    R11,R2                ; R2 = probed address
                    62 01 FD    AD    0D    1C21  5232    PROBEW  MODE(FP),#1,(R2)      ; is the first byte writeable ?
                          13    13    1C26  5233          BEQL    1$                    ; no - bypass
                FF A24A 01 FD    AD    0D    1C28  5234    PROBEW  MODE(FP),#1,-1(R2)[R10] ; is the last byte writeable ?
                          0A    12    1C2F  5235          BNEQ    1$                    ; yes - bypass
                    52 FF A24A    9E    1C31  5236          MOVAB   -1(R2)[R10],R2        ; R2 = address of last byte
                    52 01FF 8F    AA    1C36  5237          BICW2   #511,R2               ; compute address of first bad byte
                  57'AF 00        FB    1C3B  5238 1$:      CALLS   #0,B^WRITE_REASON     ; get the reason mask
                     12 50        E8    1C3F  5239          BLBS    R0,2$                 ; the write went all right - bypass
                        0093      30    1C42  5240          BSBW    FAULT_RESET           ; reinitialize registers and clear TP
                     5E FB AD     9E    1C45  5241          MOVAB   SHORT_LOCAL(FP),SP    ; shorten the frame
                          52      DD    1C49  5242          PUSHL   R2                    ; push the bad address
                          51      DD    1C4B  5243          PUSHL   R1                    ; push the reason mask
                          0C      DD    1C4D  5244          PUSHL   #SS$_ACCVIO           ; push the condition code
                          03      DD    1C4F  5245          PUSHL   #3                    ; push the number of arguments
                        009A      31    1C51  5246          BRW     SIGNAL_START          ; signal the condition
                          07      BA    1C54  5247 2$:      POPR    #^M<R0,R1,R2>         ; restore R0,R1,R2
                          05      05    1C56  5248          RSB                           ; get back
                                1C57  5249          ;
                                1C57  5250          ;       WRITE_REASON - Get the Reason Mask for Write Access Violation
                                1C57  5251          ;
                                1C57  5252          ;               parameter:      R2 = Address for which Probe Failed
                                1C57  5253          ;
                                1C57  5254          ;               returns with    R0 = Status of Access Attempt
                                1C57  5255          ;                               R1 = Reason Mask if Unsuccessful
                                1C57  5256          ;
                                1C57  5257 WRITE_REASON:                                ; entrance
                        0000    1C57  5258          .WORD   0                     ; entry mask
              6D 96 AF    9E    1C59  5259          MOVAB   B^REASON_HANDLER,(FP) ; set up the condition handler
                 62 00    80    1C5D  5260          ADDB2   #0,(R2)               ; try to change the location
                 50 01    D0    1C60  5261          MOVL    #1,R0                 ; indicate a successful write
                          04    1C63  5262          RET                           ; return
                                1C64  5263          ;
```

```
                     1C64  5265          ;
                     1C64  5266          ;       OPCODE_FAULT - Process an Opcode Reserved to Digital Fault
                     1C64  5267          ;
                     1C64  5268          ;               entered by branching
                     1C64  5269          ;
                     1C64  5270          ;               no parameters
                     1C64  5271          ;
                     1C64  5272 OPCODE_FAULT:                            ; entrance
        0071   30    1C64  5273          BSBW    FAULT_RESET             ; reinitialize registers and clear TP
   5E    FB AD   9E  1C67  5274          MOVAB   SHORT_LOCAL(FP),SP      ; shorten the frame
7E  043C 8F   3C    1C6B  5275          MOVZWL  #SS$_OPCDEC,-(SP)       ; push the condition code
          01   DD    1C70  5276          PUSHL   #1                      ; push the number of arguments
        0079   31    1C72  5277          BRW     SIGNAL_START            ; signal the condition
                     1C75  5278          ;
                     1C75  5279          ;       ADDRESS_FAULT - Process an Invalid Addressing Mode Fault
                     1C75  5280          ;
                     1C75  5281          ;               entered by branching
                     1C75  5282          ;
                     1C75  5283          ;               no parameters
                     1C75  5284          ;
                     1C75  5285 ADDRESS_FAULT:                           ; entrance
        0060   30    1C75  5286          BSBW    FAULT_RESET             ; reinitialize registers and clear TP
   5E    FB AD   9E  1C78  5287          MOVAB   SHORT_LOCAL(FP),SP      ; shorten the frame
7E  044C 8F   3C    1C7C  5288          MOVZWL  #SS$_RADRMOD,-(SP)      ; push the condition code
          01   DD    1C81  5289          PUSHL   #1                      ; push the number of arguments
        0068   31    1C83  5290          BRW     SIGNAL_START            ; signal the condition
                     1C86  5291          ;
                     1C86  5292          ;       OPERAND_FAULT - Processed a Reserved Operand Fault
                     1C86  5293          ;
                     1C86  5294          ;               entered by branching
                     1C86  5295          ;
                     1C86  5296          ;               no parameters
                     1C86  5297          ;
                     1C86  5298 OPERAND_FAULT:                           ; entrance
        004F   30    1C86  5299          BSBW    FAULT_RESET             ; reinitialize registers and clear TP
   5E    FB AD   9E  1C89  5300          MOVAB   SHORT_LOCAL(FP),SP      ; shorten the frame
7E  0454 8F   3C    1C8D  5301          MOVZWL  #SS$_ROPRAND,-(SP)      ; push the condition code
          01   DD    1C92  5302          PUSHL   #1                      ; push the number of arguments
        0057   31    1C94  5303          BRW     SIGNAL_START            ; signal the condition
                     1C97  5304          ;
                     1C97  5305          ;       UNDERFLOW - Process a Floating Underflow Fault
                     1C97  5306          ;
                     1C97  5307          ;               entered by branching
                     1C97  5308          ;
                     1C97  5309          ;               no parameters
                     1C97  5310          ;
                     1C97  5311 UNDERFLOW:                               ; entrance
        003E   30    1C97  5312          BSBW    FAULT_RESET             ; reinitialize registers and clear TP
   5E    FB AD   9E  1C9A  5313          MOVAB   SHORT_LOCAL(FP),SP      ; shorten the frame
7E  04C4 8F   3C    1C9E  5314          MOVZWL  #SS$_FLTUND_F,-(SP)     ; push the condition code
          01   DD    1CA3  5315          PUSHL   #1                      ; push the number of arguments
        0046   31    1CA5  5316          BRW     SIGNAL_START            ; signal the condition
                     1CA8  5317          ;
                     1CA8  5318          ;       OVERFLOW - Process a Floating Overflow Fault
                     1CA8  5319          ;
                     1CA8  5320          ;               entered by branching
                     1CA8  5321          ;
```

```
                                1CA8  5322          ;               no parameters
                                1CA8  5323          ;
                                1CA8  5324 OVERFLOW:                                ; entrance
                      002D   30 1CA8  5325          BSBW    FAULT_RESET           ; reinitialize registers and clear TP
                   5E   FB AD   9E 1CAB  5326          MOVAB   SHORT_LOCAL(FP),SP    ; shorten the frame
                7E   04B4 8F   3C 1CAF  5327          MOVZWL  #SS$_FLTOVF_F,-(SP)   ; push the condition code
                        01     DD 1CB4  5328          PUSHL   #1                    ; push the number of arguments
                      0035   31 1CB6  5329          BRW     SIGNAL_START          ; signal the condition
                                1CB9  5330          ;
                                1CB9  5331          ;       DIVIDE_FAULT - Process a Floating Divide by Zero Fault
                                1CB9  5332          ;
                                1CB9  5333          ;               entered by branching
                                1CB9  5334          ;
                                1CB9  5335          ;               no parameters
                                1CB9  5336          ;
                                1CB9  5337 DIVIDE_FAULT:                            ; entrance
                      001C   30 1CB9  5338          BSBW    FAULT_RESET           ; reinitialize registers and clear TP
                   5E   FB AD   9E 1CBC  5339          MOVAB   SHORT_LOCAL(FP),SP    ; shorten the frame
                7E   04BC 8F   3C 1CC0  5340          MOVZWL  #SS$_FLTDIV_F,-(SP)   ; push the condition code
                        01     DD 1CC5  5341          PUSHL   #1                    ; push the number of arguments
                      0024   31 1CC7  5342          BRW     SIGNAL_START          ; signal the condition
                                1CCA  5343          ;
                                1CCA  5344          ;       INT_OVERFLOW - Process an Integer Overflow Trap
                                1CCA  5345          ;
                                1CCA  5346          ;               entered by branching
                                1CCA  5347          ;
                                1CCA  5348          ;               no parameters
                                1CCA  5349          ;
                                1CCA  5350 INT_OVERFLOW:                            ; entrance
                   5E   FB AD   9E 1CCA  5351          MOVAB   SHORT_LOCAL(FP),SP    ; shorten the frame
                7E   047C 8F   3C 1CCE  5352          MOVZWL  #SS$_INTOVF,-(SP)     ; push the condition code
                        01     DD 1CD3  5353          PUSHL   #1                    ; push the number of arguments
                      0016   31 1CD5  5354          BRW     SIGNAL_START          ; signal the condition
                                1CD8  5355          ;
```

```
                                                H 7
                     1CD8  5357          ;
                     1CD8  5358          ;       FAULT_RESET - Perform Reinitialization Operations for a Fault
                     1CD8  5359          ;
                     1CD8  5360          ;               entered by subroutine branching
                     1CD8  5361          ;
                     1CD8  5362          ;               no parameters
                     1CD8  5363          ;
                     1CD8  5364          ;       Discussion
                     1CD8  5365          ;
                     1CD8  5366          ;           This routine subtracts the sign-extended value of each
                     1CD8  5367          ;       of the register modification bytes from the corresponding
                     1CD8  5368          ;       emulated register and clears the trace pending bit in the
                     1CD8  5369          ;       PSL.
                     1CD8  5370          ;
                     1CD8  5371 FAULT_RESET:                                 ; entrance
          50   D4    1CD8  5372          CLRL    R0                          ; clear the index
51   EB AD40   98    1CDA  5373 1$:      CVTBL   REGMOD_R0(FP)[R0],R1        ; R1 = modifications to R[R0]
14 AD40   51   C2    1CDF  5374          SUBL2   R1,REG_R0(FP)[R0]           ; unmodify the register
  F2 50   0F   F3    1CE4  5375          AOBLEQ  #15,R0,1$                   ; more registers to reset - loop
00 54 AD   1E   E5    1CE8  5376          BBCC    #PSL_TP,PSL(FP),2$          ; clear the trace pending bit
                05    1CED  5377 2$:      RSB                                 ; return
                      1CEE  5378          ;
```

```
                                                .SUBTITLE SIGNAL_START - Build the Parameter Blocks for SIGNAL
                                                ;++
                                                ; SIGNAL_START - Build the Parameter Blocks for SIGNAL
                                                ;
                                                ;       entered by branching
                                                ;
                                                ;       parameters:     (SP) = Truncated Signal Array Size (M)
                                                ;                       4(SP) = Condition Code
                                                ;                       8(SP) = First Signal Argument
                                                ;                         .
                                                ;                         .
                                                ;                         .
                                                ;                       4*<M-1>(SP) = Last Signal Argument
                                                ;
                                                ; Discussion
                                                ;
                                                ;    This routine builds the signal and mechanism arrays
                                                ; for a condition generated by the Emulator. It is entered
                                                ; with the signal array for the condition except for the
                                                ; PC and PSL pair pushed onto the Emulator's stack (with the
                                                ; pushed array length correspondingly shortened). The signal
                                                ; array, mechanism array, and the handler parameter block
                                                ; are then constructed on the user's emulated stack. The routine
                                                ; then removes the Emulator frame from the stack and enters
                                                ; the signal dispatching loop at SIGNAL.
                                                ;
                                                ; Notes:  1. The precise format of the information pushed onto
                                                ;           the user's stack is given in the description of
                                                ;           SIGNAL below.
                                                ;
                                                ;        2. The method of getting out of the Emulator used in
                                                ;           this routine is essentially the same as that used
                                                ;           in NORMAL_EXIT.
                                                ;--
                                                SIGNAL_START:                                   ; entrance
                57    8E    D0                  MOVL    (SP)+,R7                                ; R7 = number of signal parameters
             58 57    02    78                  ASHL    #2,R7,R8                                ; R8 = size of the signal parameters
             50 58    34    C1                  ADDL3   #52,R8,R0                               ; R0 = size of signal information
                   E5DE    30                  BSBW    TEST_FRAME                              ; make sure we have room for it
                56   4C AD  D0                  MOVL    REG_SP(FP),R6                           ; R6 = user's stack pointer
                76   50 AD  7D                  MOVQ    REG_PC(FP),-(R6)                        ; push the PC, PSL pair
                56    58    C2                  SUBL2   R8,R6                                   ; make room for the signal parameters
             66 6E    58    28                  MOVC3   R8,(SP),(R6)                            ; push the signal parameters
             76 57    02    C1                  ADDL3   #2,R7,-(R6)                             ; push the signal array length
                76    01    D0                  MOVL    #1,-(R6)                                ; push code for SIGNAL (vs. STOP)
                   76 14 AD  7D                  MOVQ    REG_R0(FP),-(R6)                        ; push user's R0 and R1
                76    03    CE                  MNEGL   #3,-(R6)                                ; push -3 (depth number)
                   76 48 AD  D0                  MOVL    REG_FP(FP),-(R6)                        ; push the user's FP
                76    04    D0                  MOVL    #4,-(R6)                                ; push the mechanism array length
                76    56    D0                  MOVL    R6,-(R6)                                ; push the mechanism array location
                   76 1C A6  9E                  MOVAB   28(R6),-(R6)                            ; push the signal array location
                76    02    D0                  MOVL    #2,-(R6)                                ; push the handler parameter count
          08 AD    44 AD  7D                  MOVQ    REG_AP(FP),SAVE_AP(FP)                  ; put the user's PC, PSL pair back
          10 AD    50'AF  9E                  MOVAB   B^SIGNAL,SAVE_PC(FP)                    ; store the return point
             50    48 AD  9E                  MOVAB   FRAME_END+4(FP),R0                      ; R0 = location of end of frame
                56    50    C2                  SUBL2   R0,R6                                   ; R6 = distance of user SP from it
          51    56    02    00    EF            EXTZV   #0,#2,R6,R1                             ; R1 = stack alignment
    06 AD 02    0E    51    F0                  INSV    R1,#MASK_ALIGN,#2,SAVE_MASK(FP) ; store it into the frame
```

```
                50 51   C0  1D46  5437          ADDL2   R1,R0                   ; compute the parameter area location
          FC A0 56 FE 8F  78  1D49  5438          ASHL    #-2,R6,-4(R0)           ; store the parameter count
                          04  1D4F  5439          RET                             ; return (to SIGNAL)
                              1D50  5440          ;
```

```
                        1D50  5442 ;
                        1D50  5443 ;       SIGNAL - Signal the Condition
                        1D50  5444 ;
                        1D50  5445 ;              entered by branching
                        1D50  5446 ;
                        1D50  5447 ;              parameters:     ( Described in Note 3 )
                        1D50  5448 ;
                        1D50  5449 ;       Discussion
                        1D50  5450 ;
                        1D50  5451 ;                    Following is a description of the information which
                        1D50  5452 ;                    is assumed to be pushed onto the stack when the
                        1D50  5453 ;                    routine SIGNAL is entered. The values are all
                        1D50  5454 ;                    longwords.
                        1D50  5455 ;
                        1D50  5456 ;                    Handler Parameter Block:
                        1D50  5457 ;
                        1D50  5458 ;                         (SP)    2 (handler parameter block length)
                        1D50  5459 ;                         4(SP)   signal array location
                        1D50  5460 ;                         8(SP)   mechanism array location
                        1D50  5461 ;
                        1D50  5462 ;                    Mechanism Array:
                        1D50  5463 ;
                        1D50  5464 ;                         12(SP)  4 (mechanism array length)
                        1D50  5465 ;                         16(SP)  user's FP (establisher frame)
                        1D50  5466 ;                         20(SP)  -3 (establisher depth)
                        1D50  5467 ;                         24(SP)  user's R0
                        1D50  5468 ;                         28(SP)  user's R1
                        1D50  5469 ;
                        1D50  5470 ;                    Information Not Part of any Array:
                        1D50  5471 ;
                        1D50  5472 ;                         32(SP)  1 (code for SIGNAL rather than STOP)
                        1D50  5473 ;
                        1D50  5474 ;                    Signal Array:
                        1D50  5475 ;
                        1D50  5476 ;                         36(SP)  signal array length M
                        1D50  5477 ;                         40(SP)  condition code
                        1D50  5478 ;                         44(SP)  first signal argument
                        1D50  5479 ;                           .
                        1D50  5480 ;                           .
                        1D50  5481 ;                           .
                        1D50  5482 ;                         <4*M>+28(SP) last signal argument
                        1D50  5483 ;                         <4*M>+32(SP) user's PC
                        1D50  5484 ;                         <4*M>+36(SP) user's PSL
                        1D50  5485 ;
                        1D50  5486 ;                    The user's stack pointer should coincide with the
                        1D50  5487 ;                    address <4*M>+40(SP).
                        1D50  5488 ;
                        1D50  5489 ;       We now jump to the special entry point in VMS to start
                        1D50  5490 ;       the search for handlers.  Execution will not return to us.
                        1D50  5491 ;-
                        1D50  5492 ;
                        1D50  5493 SIGNAL:                                      ; entrance
00000000'GF     17      1D50  5494          JMP     G^EXE$SRCHANDLER
                        1D56  5495
```

```
1D56  5497 ;       ********************************************************************************
1D56  5498 ;       *                                                                              *
1D56  5499 ;       *                                                                              *
1D56  5500 ;       *          End of EMULATE$ VAX Instruction Subset Emulator                     *
1D56  5501 ;       *                                                                              *
1D56  5502 ;       *                                                                              *
1D56  5503 ;       ********************************************************************************
1D56  5504
1D56  5505         .END
```

```
$$X                             =  00000088
ACCESS_VALUE                       00001192  R      03
ADDRESS1                        =  FFFFFFE7
ADDRESS2                        =  FFFFFFE3
ADDRESS3                        =  FFFFFFDF
ADDRESS_ACCESS                     00000EDE  R      03
ADDRESS_FAULT                      00001C75  R      03
ADD_REAL                           0000161E  R      03
ARC$M_DFLT_EMUL                 =  00000100
ARC$M_FFLT_EMUL                 =  00000200
ARC$M_GFLT_EMUL                 =  00000400
ARC$M_HFLT_EMUL                 =  00000800
ARC$V_DFLT_EMUL                 =  00000008
ARC$V_FFLT_EMUL                 =  00000009
ARC$V_GFLT_EMUL                 =  0000000A
ARC$V_HFLT_EMUL                 =  0000000B
BRANCH_WORD                        000011EC  R      03
BYTE_DEF_MODE                      000010A7  R      03
BYTE_DISP_MODE                     00001082  R      03
CALL_ARGS                       =  00000028
COMMON_EMODDG                      000007F0  R      03
COMPARE_REAL                       00001B6A  R      03
COND_HANDLER                       00000357  R      03
CTL$AL_STACK                       ********    X    00
CTL$AL_STACKLIM                    ********    X    00
DECR_MODE                          00001020  R      03
DISPATCH_1BYTE                     00000082  R      03
DISPATCH_2BYTE                     000000F8  R      03
DIVIDE                             00001A4F  R      03
DIVIDE_DGFLOAT                     000018B2  R      03
DIVIDE_FAULT                       00001CB9  R      03
DIVIDE_FFLOAT                      0000185F  R      03
DIVIDE_FLOAT                       00001852  R      03
DIVIDE_HFLOAT                      00001905  R      03
EMULATOR                           0000000F  R      03
EXE$ACVIOLAT                       ********    X    00
EXE$GL_ARCHFLAG                    ********    X    00
EXE$OPCDEC                         ********    X    00
EXE$SRCHANDLER                     ********    X    00
EXE$UNWIND                         ********    X    00
FAULT_RESET                        00001CD8  R      03
FIX_REAL                           00001524  R      03
FLAG0                           =  00000000
FLAG0M                          =  00000001
FLAG1                           =  00000001
FLAG1M                          =  00000002
FLAG2                           =  00000002
FLAG2M                          =  00000004
FLAG3                           =  00000003
FLAG3M                          =  00000008
FLAG4                           =  00000004
FLAG4M                          =  00000010
FLAG5                           =  00000005
FLAG5M                          =  00000020
FLAG6                           =  00000006
FLAG6M                          =  00000040
FLAG7                           =  00000007
FLAG7M                          =  00000080
FLAGS                           =  FFFFFFFC
FLOAT_LONG                         000014EC  R      03
FRACTION                        =  00000008
FRACTION1                       =  FFFFFFCF
FRACTION2                       =  FFFFFFB7
FRACTION3                       =  FFFFFF9F
FRACTION_REAL                      000015E7  R      03
FRAME_END                       =  00000044
GET_SPECIFIER                      00000EF0  R      03
GO_ON                              00001AF1  R      03
HANDLER                         =  00000000
HI_1BYTE                        =  00000076
HI_2BYTE                        =  000000FF
INCR_DEF_MODE                      0000105F  R      03
INCR_MODE                          0000103B  R      03
INDEX_MODE                         00000F81  R      03
INST_ACBD                          0000038E  R      03
INST_ACBF                          00000388  R      03
INST_ACBG                          00000394  R      03
INST_ACBH                          0000039A  R      03
INST_ACBX                          0000039E  R      03
INST_ADDD2                         0000040C  R      03
INST_ADDD3                         00000445  R      03
INST_ADDF2                         00000406  R      03
INST_ADDF3                         0000043F  R      03
INST_ADDG2                         00000412  R      03
INST_ADDG3                         0000044B  R      03
INST_ADDH2                         00000418  R      03
INST_ADDH3                         00000451  R      03
INST_ADDX2                         0000041C  R      03
INST_ADDX3                         00000455  R      03
INST_CLRO                          0000047B  R      03
INST_CMPD                          00000497  R      03
INST_CMPF                          00000491  R      03
INST_CMPG                          0000049D  R      03
INST_CMPH                          000004A3  R      03
INST_CMPX                          000004A7  R      03
INST_CVTBD                         000004D0  R      03
INST_CVTBF                         000004C8  R      03
INST_CVTBG                         000004D8  R      03
INST_CVTBH                         000004E0  R      03
INST_CVTBX                         00000526  R      03
INST_CVTDB                         0000054E  R      03
INST_CVTDF                         000005F8  R      03
INST_CVTDH                         00000600  R      03
INST_CVTDL                         0000058E  R      03
INST_CVTDW                         0000056E  R      03
INST_CVTFB                         00000546  R      03
INST_CVTFD                         000005E0  R      03
INST_CVTFG                         000005E8  R      03
INST_CVTFH                         000005F0  R      03
INST_CVTFL                         00000586  R      03
INST_CVTFW                         00000566  R      03
INST_CVTGB                         00000556  R      03
INST_CVTGF                         00000608  R      03
INST_CVTGH                         00000610  R      03
```

INST_CVTGL	00000596	R	03	INST_MULG2		00000A0F	R	03		
INST_CVTGW	00000576	R	03	INST_MULG3		00000A48	R	03		
INST_CVTHB	0000055E	R	03	INST_MULH2		00000A15	R	03		
INST_CVTHD	00000620	R	03	INST_MULH3		00000A4E	R	03		
INST_CVTHF	00000618	R	03	INST_MULX2		00000A19	R	03		
INST_CVTHG	00000628	R	03	INST_MULX3		00000A52	R	03		
INST_CVTHL	0000059E	R	03	INST_POLYD		00000B7C	R	03		
INST_CVTHW	0000057E	R	03	INST_POLYDG		00000B8E	R	03		
INST_CVTLD	00000510	R	03	INST_POLYF		00000A78	R	03		
INST_CVTLF	00000508	R	03	INST_POLYG		00000B86	R	03		
INST_CVTLG	00000518	R	03	INST_POLYH		00000C95	R	03		
INST_CVTLH	00000520	R	03	INST_PUSHAO		00000DE8	R	03		
INST_CVTLX	00000526	R	03	INST_SUBD2		00000E21	R	03		
INST_CVTRDL	00000656	R	03	INST_SUBD3		000C0E5D	R	03		
INST_CVTRFL	0000064E	R	03	INST_SUBF2		00000E1B	R	03		
INST_CVTRGL	0000065E	R	03	INST_SUBF3		00000E57	R	03		
INST_CVTRHL	00000666	R	03	INST_SUBG2		00000E27	R	03		
INST_CVTRXL	0000066C	R	03	INST_SUBG3		00000E63	R	03		
INST_CVTWD	000004F0	R	03	INST_SUBH2		00000E2D	R	03		
INST_CVTWF	000004E8	R	03	INST_SUBH3		00000E69	R	03		
INST_CVTWG	000004F8	R	03	INST_SUBX2		00000E31	R	03		
INST_CVTWH	00000500	R	03	INST_SUBX3		00000E6D	R	03		
INST_CVTWX	00000526	R	03	INST_TSTD		00000E9C	R	03		
INST_CVTXB	000005A6	R	03	INST_TSTF		00000E96	R	03		
INST_CVTXL	000005A6	R	03	INST_TSTG		00000EA2	R	03		
INST_CVTXW	000005A6	R	03	INST_TSTH		00000EA8	R	03		
INST_CVTXY	0000062E	R	03	INST_TSTX		00000EAC	R	03		
INST_DIVD2	00000693	R	03	INST_TYPES_1BYTE	=	00000277	R	03		
INST_DIVD3	000006CC	R	03	INST_TYPES_2BYTE	=	0000029A	R	03		
INST_DIVF2	0000068D	R	03	INT_OVERFLOW		00001CCA	R	03		
INST_DIVF3	000006C6	R	03	IT_D	=	00000001				
INST_DIVG2	00000699	R	03	IT_F	=	00000002				
INST_DIVG3	000006D2	R	03	IT_G	=	00000004				
INST_DIVH2	0000069F	R	03	IT_H	=	00000008				
INST_DIVH3	000006D8	R	03	IT_X	=	00000000				
INST_DIVX2	000006A3	R	03	LENGTHS		00000EE7	R	03		
INST_DIVX3	000006DC	R	03	LITERAL_MODE		00000F40	R	03		
INST_EMODD	000007B0	R	03	LOCAL_END	=	00000058				
INST_EMODF	00000702	R	03	LOCAL_START	=	FFFFFF8B				
INST_EMODG	000007CE	R	03	LOCAL_TEST		0000120D	R	03		
INST_EMODH	00000882	R	03	LONG_DEF_MODE		0000115D	R	03		
INST_MNEGD	00000945	R	03	LONG_DISP_MODE		00001136	R	03		
INST_MNEGF	0000093F	R	03	LO_1BYTE	=	00000040				
INST_MNEGG	0000094B	R	03	LO_2BYTE	=	00000032				
INST_MNEGH	00000951	R	03	MASK_ALIGN	=	0000000E				
INST_MNEGX	00000955	R	03	MODE	=	FFFFFFFD				
INST_MOVAO	00000975	R	03	MODIFY_ACCESS		00000ED5	R	03		
INST_MOVD	0000099C	R	03	MODIFY_CHECK		0000119F	R	03		
INST_MOVF	00000996	R	03	MULTIPLY		00001A05	R	03		
INST_MOVG	000009A2	R	03	MULTIPLY_DGFLOAT		0000179F	R	03		
INST_MOVH	000009A8	R	03	MULTIPLY_FFLOAT		0000174C	R	03		
INST_MOVO	000009D4	R	03	MULTIPLY_FLOAT		0000173F	R	03		
INST_MOVX	000009AC	R	03	MULTIPLY_HFLOAT		000017EF	R	03		
INST_MULD2	00000A09	R	03	NEGATE_REAL		000014E3	R	03		
INST_MULD3	00000A42	R	03	NORMALIZE		0000197F	R	03		
INST_MULF2	00000A03	R	03	NORMAL_EXIT		000001C4	R	03		
INST_MULF3	00000A3C	R	03	N_ARGS	=	00000001				

```
ONE$                    00001A77 R      03
OPCODE_FAULT            00001C64 R      03
OPERAND1              = FFFFFFC7
OPERAND2              = FFFFFFAF
OPERAND3              = FFFFFF97
OPERAND_FAULT           00001C86 R      03
OPERAND_SIZE          = 00000018
OP_INDEX              = FFFFFF8B
OP_MASK_1BYTE           000000A5 R      02
OP_MASK_2BYTE           000000C5 R      02
OP_TYPES              = FFFFFF8F
OP_TYPES4             = FFFFFF93
OVERFLOW                00001CA8 R      03
PACK_DFLOAT             00001382 R      03
PACK_FFLOAT             00001331 R      03
PACK_FLOAT              00001324 R      03
PACK_FLOAT1             00001314 R      03
PACK_FLOAT2             0000131A R      03
PACK_FLOAT3             00001320 R      03
PACK_GFLOAT             000013DC R      03
PACK_HFLOAT             0000143B R      03
POWER                 = 00000004
POWER1                = FFFFFFCB
POWER2                = FFFFFFB3
POWER3                = FFFFFF9B
PR$_ESP               = 00000001
PR$_SSP               = 00000002
PR$_USP               = 00000003
PSL                   = 00000054
PSL$C_EXEC            = 00000001
PSL$C_KERNEL          = 00000000
PSL$C_SUPER           = 00000002
PSL$C_USER            = 00000003
PSL$M_FPD             = 08000000
PSL$M_TBIT            = 00000010
PSL$M_TP              = 40000000
PSL$S_CURMOD          = 00000002
PSL$S_PRVMOD          = 00000002
PSL$V_CURMOD          = 00000018
PSL$V_PRVMOD          = 00000016
PSLM_C                = 00000001
PSLM_EQL              = 00000004
PSLM_GTR              = 00000000
PSLM_LSS              = 00000008
PSLM_N                = 00000008
PSLM_NZ               = 0000000C
PSLM_NZV              = 0000000E
PSLM_NZVC             = 0000000F
PSLM_V                = 00000002
PSLM_VC               = 00000003
PSLM_Z                = 00000004
PSL_C                 = 00000000
PSL_CAM               = 00000018
PSL_FPD               = 0000001B
PSL_FU                = 00000006
PSL_IV                = 00000005
PSL_N                 = 00000003
PSL_T                 = 00000004
PSL_TP                = 0000001E
PSL_V                 = 00000001
PSL_Z                 = 00000002
READ_ACCESS             00000EC3 R      03
READ_CHECK              000011A3 R      03
READ_FAULT              00001BAB R      03
READ_REASON             00001BE6 R      03
READ_VALUE              000011C2 R      03
REASON_HANDLER          00001BF2 R      03
REGISTER_MODE           00000FED R      03
REGMOD_AP             = FFFFFFF7
REGMOD_FP             = FFFFFFF8
REGMOD_PC             = FFFFFFFA
REGMOD_R0             = FFFFFFEB
REGMOD_R1             = FFFFFFEC
REGMOD_R10            = FFFFFFF5
REGMOD_R11            = FFFFFFF6
REGMOD_R2             = FFFFFFED
REGMOD_R3             = FFFFFFEE
REGMOD_R4             = FFFFFFEF
REGMOD_R5             = FFFFFFF0
REGMOD_R6             = FFFFFFF1
REGMOD_R7             = FFFFFFF2
REGMOD_R8             = FFFFFFF3
REGMOD_R9             = FFFFFFF4
REGMOD_SP             = FFFFFFF9
REG_AP                = 00000044
REG_DEF_MODE            0000100F R      03
REG_FP                = 00000048
REG_PC                = 00000050
REG_R0                = 00000014
REG_R1                = 00000018
REG_R10               = 0000003C
REG_R11               = 00000040
REG_R2                = 0000001C
REG_R3                = 00000020
REG_R4                = 00000024
REG_R5                = 00000028
REG_R6                = 0000002C
REG_R7                = 00000030
REG_R8                = 00000034
REG_R9                = 00000038
REG_SP                = 0000004C
ROUND_REAL              00001578 R      03
SAVE_ALIGN            = FFFFFFFF
SAVE_AP               = 00000008
SAVE_FP               = 0000000C
SAVE_MASK             = 00000006
SAVE_PARCNT           = FFFFFFFE
SAVE_PC               = 00000010
SAVE_PSW              = 00000004
SET_CONDITION           00001B20 R      03
SET_CONDITION1          00001B3F R      03
SHORT_LOCAL           = FFFFFFFB
SIGN                  = 00000001
SIGN1                 = FFFFFFC8
```

```
SIGN2                           = FFFFFFB0
SIGN3                           = FFFFFF98
SIGNAL                            00001D50 R      03
SIGNAL_START                      00001CEE R      03
SS$_ACCVIO                      = 0000000C
SS$_FLTDIV_F                    = 000004BC
SS$_FLTOVF_F                    = 000004B4
SS$_FLTUND_F                    = 000004C4
SS$_INTOVF                      = 0000047C
SS$_OPCDEC                      = 0000043C
SS$_RADRMOD                     = 0000044C
SS$_RESIGNAL                    = 00000918
SS$_ROPRAND                     = 00000454
SS$_UNWIND                      = 00000920
STORE_OPERAND                     000014B8 R      03
TEMP                            = 00000058
TEST_EQL                          00001BA5 R      03
TEST_FRAME                        000002DA R      03
TEST_GTR                          00001BA8 R      03
TEST_LSS                          00001BA1 R      03
TEST_OCTA                         00001B55 R      03
TEST_REAL                         00001B4C R      03
TYPE_ADDRESS                    = 00000004
TYPE_MODIFY                     = 00000003
TYPE_READ                       = 00000001
TYPE_WRITE                      = 00000002
TYP_B                           = 00000001
TYP_D                           = 00000007
TYP_F                           = 00000006
TYP_G                           = 00000008
TYP_H                           = 00000009
TYP_L                           = 00000003
TYP_O                           = 00000005
TYP_Q                           = 00000004
TYP_W                           = 00000002
UNDERFLOW                         00001C97 R      03
UNPACK_DFLOAT                     00001243 R      03
UNPACK_FFLOAT                     00001241 R      03
UNPACK_FLOAT                      00001234 R      03
UNPACK_FLOAT1                     00001224 R      03
UNPACK_FLOAT2                     0000122A R      03
UNPACK_FLOAT3                     00001230 R      03
UNPACK_GFLOAT                     0000127F R      03
UNPACK_HFLOAT                     000012C1 R      03
VAX$$EMULATE_FP                   00000000 RG     03
VAX$OPCDEC                        00000000 RG     02
WORD_DEF_MODE                     00001101 R      03
WORD_DISP_MODE                    000010DA R      03
WRITE_ACCESS                      00000ECC R      03
WRITE_CHECK                       000011AF R      03
WRITE_FAULT                       00001C1C R      03
WRITE_REASON                      00001C57 R      03
ZERO                            = 00000000
ZERO1                           = FFFFFFC7
ZERO2                           = FFFFFFAF
ZERO3                           = FFFFFF97
```

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
VAX$FPE_NONPAGED	000000E5 (229.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC QUAD
VAX$FPE_PAGED	00001D56 (7510.)	03 (3.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC QUAD

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	9	00:00:00.03	00:00:01.94
Command processing	78	00:00:00.55	00:00:04.93
Pass 1	837	00:00:24.47	00:01:17.29
Symbol table sort	0	00:00:01.60	00:00:03.69
Pass 2	445	00:00:11.18	00:00:36.86
Symbol table output	0	00:00:00.31	00:00:00.63
Psect synopsis output	2	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1371	00:00:38.17	00:02:05.37

The working set limit was 2100 pages.
153491 bytes (300 pages) of virtual memory were used to buffer the intermediate code.
There were 60 pages of symbol table space allocated to hold 910 non-local and 261 local symbols.
5505 source lines were read in Pass 1, producing 35 object records in Pass 2.
18 pages of virtual memory were used to define 16 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYS.OBJ]LIB.MLB;1	3
_255DUA28:[SYSLIB]STARLET.MLB;2	7
TOTALS (all libraries)	10

632 GETS were required to define 10 macros.

There were no errors, warnings or information messages.

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

XLATE
LIS

VAXEMUL
MAP

WIOHNDLR
LIS

WORKIO
LIS

BOOSTRING
LIS

EMULAT

FPEMUL
MAP

BOOEMULAT
LIS

WORDWRAP
LIS

VAXMACROS
MAR

WRIEDTMSG
LIS

FPEMULATE
LIS

VAXREGDEF
SDL

FPLOAD
LIS

VAXASHP
LIS

VAXCONVRT
LIS

VAXARITH
LIS