```
EEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                 AAAAAAAAA    TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                 AAAAAAAAA    TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                 AAAAAAAAA    TTTTTTTTTTTTTTT
EEE              MMMMMM   MMMMMM  UUU         UUU  LLL              AAA         AAA       TTT
EEE              MMMMMM   MMMMMM  UUU         UUU  LLL              AAA         AAA       TTT
EEE              MMMMMM   MMMMMM  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
EEEEEEEEEEEEEEE  MMM         MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA       TTT
EEEEEEEEEEEEEEE  MMM         MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA       TTT
EEEEEEEEEEEEEEE  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    EEEEEEEE    MM      MM  UU      UU  LL          AA      AA      TT      EEEEEEEE
FFFFFFFF    PPPPPPPP    EEEEEEEE    MM      MM  UU      UU  LL          AA      AA      TT      EEEEEEEE
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
```

```
0000            .TITLE  VAX$EMULATE_FP - Emulate floating-point instructions
0000            .IDENT  /V04-000/
0000
0000      ;
0000      ;*****************************************************************************
0000      ;*                                                                         *
0000      ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                *
0000      ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                 *
0000      ;*  ALL RIGHTS RESERVED.                                                   *
0000      ;*                                                                         *
0000      ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
0000      ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
0000      ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER  *
0000      ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  *
0000      ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY  *
0000      ;*  TRANSFERRED.                                                           *
0000      ;*                                                                         *
0000      ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE  *
0000      ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
0000      ;*  CORPORATION.                                                           *
0000      ;*                                                                         *
0000      ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
0000      ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                *
0000      ;*                                                                         *
0000      ;*                                                                         *
0000      ;*****************************************************************************
0000      ;
0000
0000      ;++
0000      ; FACILITY: VMS Executive floating point emulation
0000      ;
0000      ; ABSTRACT:
0000      ;
0000      ;       Loadable code that emulates F, D, G and H floating instructions on
0000      ;       any processor.  Octaword integer emulation is included.
0000      ;
0000      ; ENVIRONMENT: Runs at any access mode, AST Reentrant
0000      ;
0000      ; AUTHOR: Steven B. Lionel, 22-March-1983
0000      ;
0000      ;       Emulation code based on LIB$EMULATE by Derek Zave.
0000      ;
0000      ; MODIFIED BY:
0000      ;
0000      ;       JCW1010         Jeffrey C. Wiener       21-Aug-1984
0000      ;       Added a missing CLRL REG_R1(FP) instruction to the POLYF emulation.
0000      ;
0000      ;       RNG0009         Rod N. Gamache          14-Aug-1984
0000      ;       Only require a P1 control region on exceptions if the exception
0000      ;       did not occur in kernel mode (i.e. we have to switch stacks), in
0000      ;       the routine VAX$OPCDEC.
0000      ;
0000      ;       JCW1008         Jeffrey C. Wiener       10-Aug-1984
0000      ;       Corrected a BBS instruction in MULTIPLY_FFLOAT. It had referred
0000      ;       to FRACTION1+8(FP). It should have been FRACTION1+12(FP)
0000      ;
0000      ;       LJK0025         Lawrence J. Kenah       8-Mar-1984
```

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

```
0000            .SBTTL  DECLARATIONS
0000    ;
0000    ; LIBRARY MACRO CALLS:
0000    ;
0000            $SSDEF                  ; System Status Codes
0000            $PSLDEF                 ; Processor Status Longword definitions
0000            $PRDEF                  ; Processor register definitions
0000            $ARCDEF                 ; Architecture support flag definitions
0000
0000    ;
0000    ; EXTERNAL DECLARATIONS:
0000    ;
0000            .DSABL  GBL             ; Force all external symbols to be declared
0000            .EXTRN  EXE$SRCHANDLER  ; System routine that looks for handlers
0000            .EXTRN  EXE$UNWIND      ; Unwind exception
0000            .EXTRN  EXE$OPCDEC      ; VMS entry to handle SS$_OPCDEC
0000            .EXTRN  EXE$ACVIOLAT    ; VMS entry to handle SS$_ACCVIO
0000            .EXTRN  EXE$GL_ARCHFLAG ; Architecture support flags
0000            .EXTRN  CTL$AL_STACK    ; Stack limit arrays
0000            .EXTRN  CTL$AL_STACKLIM ; Stack limit arrays
0000
0000    ;
0000    ; MACROS:
0000    ;
0000
0000    ;       Macro for Comparing Condition Codes
0000            .MACRO  CMPCOND COND,LOC
0000            CMPZV   #3,#26,LOC,#COND@-3
0000            .ENDM
0000
0000    ;       Macro for loading OP_TYPES with the types of the various operands,
0000            .MACRO  SET_OP_TYPES    OP1=F,OP2=F,OP3=F,OP4=F,OP5=F,OP6=F
0000            .NARG   N_ARGS
0000            .IF     EQ N_ARGS-6     ; 6 arguments?
0000            MOVW    #<TYP_'OP5'+<TYP_'OP6'@8>>, OP_TYPES4(FP)
0000            .ENDC
0000            .IF     EQ N_ARGS-5     ; 5 arguments?
0000            MOVB    #<TYP_'OP5'>, OP_TYPES4(FP)
0000            .ENDC
0000            .IF     GT N_ARGS-2     ; more than 2 arguments?
0000            MOVL    #<TYP_'OP1'+ -
0000                     <TYP_'OP2'@8>+ -
0000                     <TYP_'OP3'@16>+ -
0000                     <TYP_'OP4'@24>>, OP_TYPES(FP)
0000            .ENDC
0000            .IF     EQ N_ARGS-2     ; 2 arguments?
0000            MOVW    #<TYP_'OP1'+<TYP_'OP2'@8>>, OP_TYPES(FP)
0000            .ENDC
0000            .IF     EQ N_ARGS-1     ; 1 argument?
0000            MOVB    #TYP_'OP1', OP_TYPES(FP)
0000            .ENDC
0000            .ENDM
0000
0000    ;
0000    ; EQUATED SYMBOLS:
0000    ;
0000    ;       See body of routine
```

```
0000   144 ;
0000   145 ; OWN STORAGE:
0000   146 ;
0000   147 ;       NONE
0000   148 ;
```

```
          0000        ;       ****************************************************************
          0000        ;       *                                                              *
          0000        ;       *                                                              *
          0000        ;       *                    Assorted Definitions                      *
          0000        ;       *                                                              *
          0000        ;       *                                                              *
          0000        ;       ****************************************************************
          0000        ;
          0000        ;
          0000        ;
          0000        ;       Parameters
          0000        ;
00000028  0000  CALL_ARGS =    40                          ; flexible stack space (longwords)
          0000        ;
          0000        ;       Opcode range limits
          0000        ;
00000040  0000  LO_1BYTE =     ^X40
00000076  0000  HI_1BYTE =     ^X76
00000032  0000  LO_2BYTE =     ^X32
000000FF  0000  HI_2BYTE =     ^XFF
          0000        ;
          0000        ;       Operand Area Layout
          0000        ;
00000000  0000  ZERO     =     0                           ; zero indicator (byte)
00000001  0000  SIGN     =     1                           ; sign indicator (byte)
00000004  0000  POWER    =     4                           ; exponent (longword)
00000008  0000  FRACTION =     8                           ; fraction area (octaword)
00000018  0000  OPERAND_SIZE = 24                          ; operand area size (bytes)
          0000        ;
          0000        ;       Bits in the Processor Status Longword (PSL)
          0000        ;
00000000  0000  PSL_C    =     0                           ; carry indicator
00000001  0000  PSL_V    =     1                           ; overflow indicator
00000002  0000  PSL_Z    =     2                           ; zero indicator
00000003  0000  PSL_N    =     3                           ; negative indicator
00000004  0000  PSL_T    =     4                           ; trace enable indicator
00000005  0000  PSL_IV   =     5                           ; integer overflow trap enable
00000006  0000  PSL_FU   =     6                           ; floating underflow fault enable
00000018  0000  PSL_CAM  =     24                          ; low bit of current access mode
0000001B  0000  PSL_FPD  =     27                          ; instruction first part done
0000001E  0000  PSL_TP   =     30                          ; trace pending indicator
          0000        ;
          0000        ;       Masks for the Processor Status Longword
          0000        ;
00000001  0000  PSLM_C   =     1@PSL_C                     ; carry indicator
00000002  0000  PSLM_V   =     1@PSL_V                     ; overflow indicator
00000004  0000  PSLM_Z   =     1@PSL_Z                     ; zero indicator
00000008  0000  PSLM_N   =     1@PSL_N                     ; negative indicator
00000003  0000  PSLM_VC  =     PSLM_V+PSLM_C               ; overflow and carry indicators
0000000F  0000  PSLM_NZVC =    PSLM_VC+PSLM_N+PSLM_Z       ; condition code
0000000C  0000  PSLM_NZ  =     PSLM_N+PSLM_Z               ; comparison codes
0000000E  0000  PSLM_NZV =     PSLM_NZ+PSLM_V              ; bits other than carry
00000008  0000  PSLM_LSS =     PSLM_N                      ; less than condition code
00000004  0000  PSLM_EQL =     PSLM_Z                      ; equals condition code
00000000  0000  PSLM_GTR =     0                           ; greater than condition code
          0000        ;
          0000        ;       Call Frame Layout
```

```
         0000                      ;
00000000 0000   HANDLER =          0                          ; condition handler location
00000004 0000   SAVE_PSW =         4                          ; saved processor status word
00000006 0000   SAVE_MASK =        6                          ; register save mask
0000000E 0000   MASK_ALIGN =       14                         ; bit position of alignment bits
00000008 0000   SAVE_AP =          8                          ; user's argument pointer
0000000C 0000   SAVE_FP =          12                         ; user's frame pointer
00000010 0000   SAVE_PC =          16                         ; return point
00000014 0000   REG_R0  =          20                         ; user's R0
00000018 0000   REG_R1  =          24                         ; user's R1
0000001C 0000   REG_R2  =          28                         ; user's R2
00000020 0000   REG_R3  =          32                         ; user's R3
00000024 0000   REG_R4  =          36                         ; user's R4
00000028 0000   REG_R5  =          40                         ; user's R5
0000002C 0000   REG_R6  =          44                         ; user's R6
00000030 0000   REG_R7  =          48                         ; user's R7
00000034 0000   REG_R8  =          52                         ; user's R8
00000038 0000   REG_R9  =          56                         ; user's R9
0000003C 0000   REG_R10 =          60                         ; user's R10
00000040 0000   REG_R11 =          64                         ; user's R11
00000044 0000   FRAME_END =        68                         ; end of call frame
         0000              ;
         0000              ;       Call Frame Extension Layout
         0000              ;
00000044 0000   REG_AP  =          68                         ; user's AP
00000048 0000   REG_FP  =          72                         ; user's FP
0000004C 0000   REG_SP  =          76                         ; user's SP
00000050 0000   REG_PC  =          80                         ; user's PC
00000054 0000   PSL     =          84                         ; user's PSL
00000058 0000   LOCAL_END =        88                         ; end of Emulator local storage
00000058 0000   TEMP    =          88                         ; temporary area for arithmetic
         0000              ;
         0000              ;       Local Storage Layout
         0000              ;
FFFFFFFF 0000   SAVE_ALIGN =       HANDLER-1                  ; saved copy of alignment bits
FFFFFFFE 0000   SAVE_PARCNT =      SAVE_ALIGN-1               ; saved copy of parameter count
FFFFFFFD 0000   MODE    =          SAVE_PARCNT-1              ; access mode for probes
FFFFFFFC 0000   FLAGS   =          MODE-1                     ; indicator flag bits
FFFFFFFB 0000   SHORT_LOCAL =      FLAGS-1                    ; start of short local storage
FFFFFFFA 0000   REGMOD_PC =        SHORT_LOCAL-1              ; changes to user's PC
FFFFFFF9 0000   REGMOD_SP =        REGMOD_PC-1                ; changes to user's SP
FFFFFFF8 0000   REGMOD_FP =        REGMOD_SP-1                ; changes to user's FP
FFFFFFF7 0000   REGMOD_AP =        REGMOD_FP-1                ; changes to user's AP
FFFFFFF6 0000   REGMOD_R11 =       REGMOD_AP-1                ; changes to user's R11
FFFFFFF5 0000   REGMOD_R10 =       REGMOD_R11-1               ; changes to user's R10
FFFFFFF4 0000   REGMOD_R9 =        REGMOD_R10-1               ; changes to user's R9
FFFFFFF3 0000   REGMOD_R8 =        REGMOD_R9-1                ; changes to user's R8
FFFFFFF2 0000   REGMOD_R7 =        REGMOD_R8-1                ; changes to user's R7
FFFFFFF1 0000   REGMOD_R6 =        REGMOD_R7-1                ; changes to user's R6
FFFFFFF0 0000   REGMOD_R5 =        REGMOD_R6-1                ; changes to user's R5
FFFFFFEF 0000   REGMOD_R4 =        REGMOD_R5-1                ; changes to user's R4
FFFFFFEE 0000   REGMOD_R3 =        REGMOD_R4-1                ; changes to user's R3
FFFFFFED 0000   REGMOD_R2 =        REGMOD_R3-1                ; changes to user's R2
FFFFFFEC 0000   REGMOD_R1 =        REGMOD_R2-1                ; changes to user's R1
FFFFFFEB 0000   REGMOD_R0 =        REGMOD_R1-1                ; changes to user's R0
FFFFFFE7 0000   ADDRESS1 =         REGMOD_R0-4                ; temporary address area #1
FFFFFFE3 0000   ADDRESS2 =         ADDRESS1-4                 ; temporary address area #2
```

```
FFFFFFDF  0000  ADDRESS3 =       ADDRESS2-4              ; temporary address area #3
FFFFFFC7  0000  OPERAND1 =       ADDRESS3-OPERAND_SIZE   ; temporary operand area #1
FFFFFFAF  0000  OPERAND2 =       OPERAND1-OPERAND_SIZE   , temporary operand area #2
FFFFFF97  0000  OPERAND3 =       OPERAND2-OPERAND_SIZE   ; temporary operand area #3
FFFFFF93  0000  OP_TYPES4 =      OPERAND3-4              ; Four bytes for operand type codes
FFFFFF8F  0000  OP_TYPES=        OP_TYPES4-4             ; Four more bytes for operand type codes
FFFFFF8B  0000  OP_INDEX =       OP_TYPES-4              ; Pointer to current byte of OP_TYPES
FFFFFF8B  0000  LOCAL_START =    OP_INDEX                ; start of Emulator local storage
          0000          ;
          0000          ;        Indicator Bit Numbers
          0000          ;
00000000  0000  FLAG0     =      0                       ; inhibit local store check
00000001  0000  FLAG1     =      1                       ; register mode operand
00000002  0000  FLAG2     =      2                       ; (not assigned)
00000003  0000  FLAG3     =      3                       ; (not assigned)
00000004  0000  FLAG4     =      4                       ; (not assigned)
00000005  0000  FLAG5     =      5                       ; (not assigned)
00000006  0000  FLAG6     =      6                       ; (not assigned)
00000007  0000  FLAG7     =      7                       ; temporary use
          0000          ;
          0000          ;        Indicator Bit Masks
          0000          ;
00000001  0000  FLAG0M    =      1@0                     ; inhibit local store check
00000002  0000  FLAG1M    =      1@1                     ; register mode operand
00000004  0000  FLAG2M    =      1@2                     ; (not assigned)
00000008  0000  FLAG3M    =      1@3                     ; (not assigned)
00000010  0000  FLAG4M    =      1@4                     ; (not assigned)
00000020  0000  FLAG5M    =      1@5                     ; (not assigned)
00000040  0000  FLAG6M    =      1@6                     ; (not assigned)
00000080  0000  FLAG7M    =      1@7                     ; temporary use
          0000          ;
          0000          ;        Fields in the Operand Areas
          0000          ;
FFFFFFC7  0000  ZERO1     =      OPERAND1+ZERO           ; zero flag of OPERAND1
FFFFFFAF  0000  ZERO2     =      OPERAND2+ZERO           ; zero flag of OPERAND2
FFFFFF97  0000  ZERO3     =      OPERAND3+ZERO           ; zero flag of OPERAND3
FFFFFFC8  0000  SIGN1     =      OPERAND1+SIGN           ; sign of OPERAND1
FFFFFFB0  0000  SIGN2     =      OPERAND2+SIGN           ; sign of OPERAND2
FFFFFF98  0000  SIGN3     =      OPERAND3+SIGN           ; sign of OPERAND3
FFFFFFCB  0000  POWER1    =      OPERAND1+POWER          ; exponent of OPERAND1
FFFFFFB3  0000  POWER2    =      OPERAND2+POWER          ; exponent of OPERAND2
FFFFFF9B  0000  POWER3    =      OPERAND3+POWER          ; exponent of OPERAND3
FFFFFFCF  0000  FRACTION1 =      OPERAND1+FRACTION       ; fraction of OPERAND1
FFFFFFB7  0000  FRACTION2 =      OPERAND2+FRACTION       ; fraction of OPERAND2
FFFFFF9F  0000  FRACTION3 =      OPERAND3+FRACTION       ; fraction of OPERAND3
          0000          ;
          0000          ;        Access Type Code Definitions
          0000          ;
00000001  0000  TYPE_READ =      1                       ; read only access
00000002  0000  TYPE_WRITE =     2                       ; write only access
00000003  0000  TYPE_MODIFY =    3                       ; modify access
00000004  0000  TYPE_ADDRESS =   4                       ; address access
          0000          ;
          0000          ;        Data Type Code Definitions
          0000          ;
00000001  0000  TYP_B     =      1                       ; byte
00000002  0000  TYP_W     =      2                       ; word
```

```
00000003  0000      TYP_L    =      3                       ; longword
00000004  0000      TYP_Q    =      4                       ; quadword
00000005  0000      TYP_O    =      5                       ; octaword
00000006  0000      TYP_F    =      6                       ; F_floating
00000007  0000      TYP_D    =      7                       ; D_floating
00000008  0000      TYP_G    =      8                       ; G_floating
00000009  0000      TYP_H    =      9                       ; H_floating
          0000
          0000               ;
          0000               ; Instruction type definitions for optimization check
          0000               ;
00000001  0000      IT_D = ARC$M_DFLT_EMUL@-ARC$V_DFLT_EMUL
00000002  0000      IT_F = ARC$M_FFLT_EMUL@-ARC$V_DFLT_EMUL
00000004  0000      IT_G = ARC$M_GFLT_EMUL@-ARC$V_DFLT_EMUL
00000008  0000      IT_H = ARC$M_HFLT_EMUL@-ARC$V_DFLT_EMUL
00000000  0000      IT_X = 0        ; none
```

```
                      0000          .SUBTITLE       VAX$OPCDEC
                      0000  ;+
                      0000  ; Functional Description:
                      0000  ;
                      0000  ;       This routine is entered in kernel mode through the SCB vector for
                      0000  ;       the OPCDEC exception. It determines if the instruction that caused
                      0000  ;       the fault is one supported by this emulator. If so, the rest of the
                      0000  ;       emulation is carried out by an instruction-specific routine. If not,
                      0000  ;       control is transferred to the OPCDEC handler that is a part of the
                      0000  ;       VMS executive to allow normal exception dispatching to take place.
                      0000  ;
                      0000  ; Input Parameters:
                      0000  ;
                      0000  ;       0(SP) - PC of faulting instruction
                      0000  ;       4(SP) - PSL at the time of the fault
                      0000  ;
                      0000  ; Output Parameters:
                      0000  ;
                      0000  ;       The real output from this routine is the routine to which control
                      0000  ;       is passed, namely the routine that handles each separate instruction.
                      0000  ;-
                      0000
                  00000000          .PSECT  VAX$FPE_NONPAGED PIC, USR, CON, REL, LCL, SHR,-
                      0000                  EXE, RD, NOWRT, QUAD
                      0000          .ALIGN  QUAD
                      0000
                      0000  VAX$OPCDEC::
                      0000
         7E    50   7D  0000          MOVQ    R0, -(SP)               ; Save R0 and R1
                      0003
                      0003  ; It is not clear whether the following PROBER is necessary. One approach is
                      0003  ; that an inaccessible opcode would cause an access violation rather than
                      0003  ; a reserved instruction exception.
                      0003
      50    08 AE   D0  0003          MOVL    8(SP), R0               ; R0 contains the PC of the instruction
                      0007          IFNORD  #1, (R0), 15$           ; Insure that opcode can be read
         51    80   9A  000D          MOVZBL  (R0)+, R1               ; Get first opcode byte, increment "PC"
 74 00A5'CF    51   E1  0010          BBC     R1, W^OP_MASK_1BYTE, 90$; Test for opcode not emulated
      FD 8F    51   91  0016          CMPB    R1, #^XFD               ; Is it a two-byte opcode?
               0F   12  001A          BNEQ    10$                     ; Skip if not
                      001C          IFNORD  #1, (R0), 15$           ; Can second byte be read?
         51    60   9A  0022          MOVZBL  (R0), R1                ; Fetch second byte
 5F 00C5'CF    51   E1  0025          BBC     R1, W^OP_MASK_2BYTE, 90$; Test for opcode not emulated
                      002B
                      002B  ;+
                      002B  ; Now the kernel stack looks as follows:
                      002B  ;
                      002B  ;       00(SP)  - Saved R0
                      002B  ;       04(SP)  - Saved R1
                      002B  ;       08(SP)  - PC of instruction
                      002B  ;       12(SP)  - PSL of instruction
                      002B  ;
                      002B  ;
                      002B  ; Switch stacks to that of the exception, push onto that stack the PC/PSL.
                      002B  ; Push the PSL with T, TP and FPD bits clear, the address of
                      002B  ; EMULATE_FP, and then REI to EMULATE_FP.
                      002B  ;-
```

```
                                  002B
     51   0C AE   02   18   EF    002B   10$:    EXTZV    #PSL$V_CURMOD,#PSL$S_CURMOD,12(SP),R1 ; Get previous mode
                       51   13    0031           BEQL     20$                         ; Br if kernel, no stack change needed
                                  0033           IFNORD   #4,G^CTL$AL_STACK,90$       ; Is there a control region?
                                  003D                                                ; Br if not, handle as a real exception
                                  003D           ASSUME   PSL$C_KERNEL EQ 0
                                  003D           ASSUME   PR$_ESP EQ PSL$C_EXEC
                                  003D           ASSUME   PR$_SSP EQ PSL$C_SUPER
                                  003D           ASSUME   PR$_USP EQ PSL$C_USER
                  50   51   DB    003D           MFPR     R1, R0                      ; Get address of stack of excpt mode
                                  0040           IFNOWRT  #8, -8(R0), 40$             ; Br if cannot copy to excpt mode stk
      00000000'GF41    50   D1    0047           CMPL     R0,G^CTL$AL_STACK[R1]       ; Top address of stack in range?
                       42   1A    004F           BGTRU    40$                         ; If GTRU no.
                  50   08   C2    0051           SUBL2    #8, R0                      ; Get new low stack address
                  51   03   D1    0054           CMPL     #PSL$C_USER,R1              ; Previous mode user?
                       0A   13    0057           BEQL     12$                         ; If EQL yes.
      00000000'GF41    50   D1    0059           CMPL     R0,G^CTL$AL_STACKLIM[R1]; Bottom address of stack in range?
                       30   1F    0061           BLSSU    40$                         ; If LSSU no.
             60   08 AE     7D    0063   12$:    MOVQ     8(SP), (R0)                 ; Push PC/PSL
                  51   50   DA    0067           MTPR     R0, R1                      ; Set stack pointer to new value
   0C AE   48000010 8F      CA    006A           BICL2    #<PSL$M_TP!PSL$M_TBIT!PSL$M_FPD>,12(SP) ; Clear bits in PSL
 0C AE   02   16   51       F0    0072           INSV     R1,#PSL$V_PRVMOD,#PSL$S_PRVMOD,12(SP)   ; Set PRVMOD = CURMOD
      08 AE   0000'CF       9E    0078           MOVAB    W^VAX$$EMULATE_FP, 8(SP); Set entry address
                  50   8E   7D    007E           MOVQ     (SP)+, R0                   ; Pop saved R0 and R1
                       02         0081           REI                                  ; Jump to EMULATE_FP
                                  0082
                       0F   11    0082   15$:    BRB      40$                         ; Chain branch for ACCVIO test
                                  0084
                                  0084   ;+
                                  0084   ; We come here if the previous mode is kernel, since we don't need to
                                  0084   ; switch modes.
                                  0084   ;-
                                  0084
                  50   8E   7D    0084   20$:    MOVQ     (SP)+, R0                   ; Pop saved R0 and R1
                    FF76'   31    0087           BRW      VAX$$EMULATE_FP             ; Emulate the instruction
                                  008A
                                  008A   ;+
                                  008A   ; If we don't handle the instruction, remove the things we pushed on the
                                  008A   ; stack and jump to the EXECPTION code that handles SS$_OPCDEC
                                  008A   ;-
                                  008A
                  50   8E   7D    008A   90$:    MOVQ     (SP)+, R0                   ; Restore R0-R1
          00000000'GF       17    008D           JMP      G^EXE$OPCDEC                ; Reflect exception
                                  0093
                                  0093   ; If a PROBE of an outer access mode fails, then the exception is reported
                                  0093   ; as an access violation rather than as a reserved instruction. This allows
                                  0093   ; the stack expansion logic and other such things to be invoked without this
                                  0093   ; emulator worrying about such things.
                                  0093
                  7E   6E   7D    0093   40$:    MOVQ     (SP), -(SP)                 ; Move saved R0-R1
               0C AE   50   D0    0096           MOVL     R0,12(SP)                   ; Store inaccessible stack address
                  50   8E   7D    009A           MOVQ     (SP)+,R0                    ; Restore R0-R1
                                  009D
                                  009D   ; Note that the access violation mask generated here (always zero) is not
                                  009D   ; correct for the extremely rare case of an inaccessible exception stack.
                                  009D
                       6E   D4    009D           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          .SBTTL  Mask of emulated opcodes
                00A5
                00A5          ;
                00A5          ;       OP_MASK_1BYTE and OP_MASK_2BYTE are 256-bit bitmasks with
                00A5          ;       bits set corresponding to opcodes of instructions that
                00A5          ;       this emulator can handle.
                00A5          ;
                00A5
                00A5  OP_MASK_1BYTE:          ; Mask for 1-byte opcodes (including FD)
                00A5  ;         FEDCBA9876543210FEDCBA9876543210
       00000000 00A5          .LONG   ^B00000000000000000000000000000000      ; 00-1F
       00000000 00A9          .LONG   ^B00000000000000000000000000000000      ; 20-3F
       007FFFFF 00AD          .LONG   ^B00000000011111111111111111111111      ; 40-5F
       007FFFFF 00B1          .LONG   ^B00000000011111111111111111111111      ; 60-7F
       00000000 00B5          .LONG   ^B00000000000000000000000000000000      ; 80-9F
       00000000 00B9          .LONG   ^B00000000000000000000000000000000      ; A0-BF
       00000000 00BD          .LONG   ^B00000000000000000000000000000000      ; C0-DF
       20000000 00C1          .LONG   ^B00100000000000000000000000000000      ; E0-FF
                00C5
                00C5  OP_MASK_2BYTE:          ; Mask for 2-byte opcodes (second byte index)
                00C5  ;         FEDCBA9876543210FEDCBA9876543210
       00000000 00C5          .LONG   ^B00000000000000000000000000000000      ; 00-1F
       000C0000 00C9          .LONG   ^B00000000000011000000000000000000      ; 20-3F
       007FFFFF 00CD          .LONG   ^B00000000011111111111111111111111      ; 40-5F
       F07FFFFF 00D1          .LONG   ^B11110000011111111111111111111111      ; 60-7F
       03000000 00D5          .LONG   ^B00000011000000000000000000000000      ; 80-9F
       00000000 00D9          .LONG   ^B00000000000000000000000000000000      ; A0-BF
       00000000 00DD          .LONG   ^B00000000000000000000000000000000      ; C0-DF
       00C00000 00E1          .LONG   ^B00000000110000000000000000000000      ; E0-FF
```

```
                           00E5          .SBTTL  VAX$$EMULATE_FP
                           00E5
                           00E5          ;
                           00E5          ;       VAX$$EMULATE_FP - Emulator Entrance
                           00E5          ;
                           00E5          ;               entered by branching
                           00E5          ;
                           00E5          ;               parameters      0(SP)   Instruction PC
                           00E5          ;                               4(SP)   Instruction PSL
                           00E5          ;
                           00E5          ;       Discussion
                           00E5          ;
                           00E5          ;           This routine provides a simple method of activating the
                           00E5          ;       Emulator. The PC and PSL for the instruction being emulated
                           00E5          ;       are pushed onto the stack and the routine is entered. The
                           00E5          ;       routine simply allocates the simulated user stack space for
                           00E5          ;       the Emulator and calls the emulation procedure.
                           00E5          ;
                           00E5          ;       Notes:  1. The FPD bit may be set in the pushed PSL. This
                           00E5          ;                  bit will only be interpreted during the emulation
                           00E5          ;                  of the POLYx instructions where it is
                           00E5          ;                  interpreted as described in the VAX System
                           00E5          ;                  Reference Manual.
                           00E5          ;
                           00E5          ;
                           00E5
                       00000000          .PSECT  VAX$FPE_PAGED   PIC, USR, CON, REL, LCL, SHR,-
                           0000                  EXE, RD, NOWRT, QUAD
                           0000
                           0000 VAX$$EMULATE_FP::
5E   FF68 CE      9E       0000          MOVAB   -4*<CALL_ARGS-2>(SP),SP ; allocate the parameter block
 0F'AF    28      FB       0005          CALLS   #CALL_ARGS,B^EMULATOR   ; call the Emulator
 00000000'GF      17       0009          JMP     G^EXE$OPCDEC            ; shouldn't return here -
                           000F                                          ; indicate OPCDEC as fallback
                           000F          ;
```

```
000F        .SBTTL  EMULATOR
000F
000F    ;
000F    ;       EMULATOR - Start Instruction Emulation
000F    ;
000F    ;               parameters:     ( Described Below )
000F    ;
000F    ;       Discussion
000F    ;
000F    ;           This routine initializes the Emulator, processes the
000F    ;       instruction code, and passes control to the appropriate
000F    ;       instruction emulation routine. The parameter list consists of
000F    ;       CALL_ARGS longwords of which only the last two have any
000F    ;       meaning. These parameters are the PC and PSL for the emulation.
000F    ;       The position following the parameter list is the
000F    ;       user's stack pointer. The area covered by the parameter list
000F    ;       is used to emulate the top of the user's stack.
000F    ;
000F    ;           When the routine is entered the CALLS instruction saves
000F    ;       the user's registers R0 to R11 in order and saves AP and FP
000F    ;       elsewhere in the frame. The routine extends the saved
000F    ;       registers by saving the user's AP, FP, SP, PC, and PSL after
000F    ;       the saved register area (the last two are taken from the
000F    ;       parameter list).
000F    ;
000F    ;           The local storage is allocated by extending the stack and
000F    ;       the register modification bytes are cleared (these bytes
000F    ;       record small changes to the register values). The cell MODE
000F    ;       is set equal to the current access mode for use in probing
000F    ;       memory references. The alignment bits in the call frame and
000F    ;       the call parameter count are also saved so there is a safe
000F    ;       copies to use when processing unwinds.
000F    ;
000F    ;       Notes:  1. From the description of the way the simulated
000F    ;                  register area is constructed, it is clear that
000F    ;                  the length longword of the parameter list is
000F    ;                  overwritten. All of the methods of leaving the
000F    ;                  Emulator put this longword back together. The
000F    ;                  internal condition handler does this if it
000F    ;                  detects an unwind.
000F    ;
000F    ;               2. Here are some more details on the register change
000F    ;                  bytes: Until the very end of instruction processing
000F    ;                  when the results are output, all of the changes
000F    ;                  which occur to the registers are caused by adding
000F    ;                  or subtracting small values to or from a register.
000F    ;                  These changes are also recorded in a corresponding
000F    ;                  register change byte so the register may be
000F    ;                  restored to its original value if a fault occurs.
000F    ;                  Those instructions that save results in the
000F    ;                  registers in order to be interruptable, use the FPD
000F    ;                  bit in the PSL to indicate that this has been done
000F    ;                  so different logic should be used for restarting
000F    ;                  the instruction after a fault. In this case the
000F    ;                  change bytes should be cleared when FPD is set
000F    ;                  except for the one for PC.
000F    ;
```

```
                                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          .SBTTL  DISPATCH_1BYTE
                        0082
                        0082          ;
                        0082          ;       DISPATCH_1BYTE - Branch to emulation routine for 1-byte opcode
                        0082          ;
                        0082          ;               entered by branching
                        0082          ;
                        0082          ;               R0 contains opcode
                        0082          ;
                        0082
                        0082 DISPATCH_1BYTE:
  36   40 8F   50   8F  0082          CASEB   R0,#^X40,#<^X76-^X40>   ; from ADDF2 through CVTDF
                 037F'  0087 1$:      .WORD   INST_ADDF2-1$           ; 40    ADDF2
                 03B8'  0089          .WORD   INST_ADDF3-1$           ; 41    ADDF3
                 0D94'  008B          .WORD   INST_SUBF2-1$           ; 42    SUBF2
                 0DD0'  008D          .WORD   INST_SUBF3-1$           ; 43    SUBF3
                 097C'  008F          .WORD   INST_MULF2-1$           ; 44    MULF2
                 09B5'  0091          .WORD   INST_MULF3-1$           ; 45    MULF3
                 0606'  0093          .WORD   INST_DIVF2-1$           ; 46    DIVF2
                 063F'  0095          .WORD   INST_DIVF3-1$           ; 47    DIVF3
                 04BF'  0097          .WORD   INST_CVTFB-1$           ; 48    CVTFB
                 04DF'  0099          .WORD   INST_CVTFW-1$           ; 49    CVTFW
                 04FF'  009B          .WORD   INST_CVTFL-1$           ; 4A    CVTFL
                 05C7'  009D          .WORD   INST_CVTRFL-1$          ; 4B    CVTRFL
                 0441'  009F          .WORD   INST_CVTBF-1$           ; 4C    CVTBF
                 0461'  00A1          .WORD   INST_CVTWF-1$           ; 4D    CVTWF
                 0481'  00A3          .WORD   INST_CVTLF-1$           ; 4E    CVTLF
                 0301'  00A5          .WORD   INST_ACBF-1$            ; 4F    ACBF
                 090F'  00A7          .WORD   INST_MOVF-1$            ; 50    MOVF
                 040A'  00A9          .WORD   INST_CMPF-1$            ; 51    CMPF
                 08B8'  00AB          .WORD   INST_MNEGF-1$           ; 52    MNEGF
                 0E0F'  00AD          .WORD   INST_TSTF-1$            ; 53    TSTF
                 067B'  00AF          .WORD   INST_EMODF-1$           ; 54    EMODF
                 09F1'  00B1          .WORD   INST_POLYF-1$           ; 55    POLYF
                 0559'  00B3          .WORD   INST_CVTFD-1$           ; 56    CVTFD
                 006E'  00B5          .WORD   10$-1$                  ; 57
                 006E'  00B7          .WORD   10$-1$                  ; 58    ADAWI
                 006E'  00B9          .WORD   10$-1$                  ; 59
                 006E'  00BB          .WORD   10$-1$                  ; 5A
                 006E'  00BD          .WORD   10$-1$                  ; 5B
                 006E'  00BF          .WORD   10$-1$                  ; 5C    INSQHI
                 006E'  00C1          .WORD   10$-1$                  ; 5D    INSQTI
                 006E'  00C3          .WORD   10$-1$                  ; 5E    REMQHI
                 006E'  00C5          .WORD   10$-1$                  ; 5F    REMQTI
                 0385'  00C7          .WORD   INST_ADDD2-1$           ; 60    ADDD2
                 03BE'  00C9          .WORD   INST_ADDD3-1$           ; 61    ADDD3
                 0D9A'  00CB          .WORD   INST_SUBD2-1$           ; 62    SUBD2
                 0DD6'  00CD          .WORD   INST_SUBD3-1$           ; 63    SUBD3
                 0982'  00CF          .WORD   INST_MULD2-1$           ; 64    MULD2
                 09BB'  00D1          .WORD   INST_MULD3-1$           ; 65    MULD3
                 060C'  00D3          .WORD   INST_DIVD2-1$           ; 66    DIVD2
                 0645'  00D5          .WORD   INST_DIVD3-1$           ; 67    DIVD3
                 04C7'  00D7          .WORD   INST_CVTDB-1$           ; 68    CVTDB
                 04E7'  00D9          .WORD   INST_CVTDW-1$           ; 69    CVTDW
                 0507'  00DB          .WORD   INST_CVTDL-1$           ; 6A    CVTDL
                 05CF'  00DD          .WORD   INST_CVTRDL-1$          ; 6B    CVTRDL
                 0449'  00DF          .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         .SBTTL  DISPATCH_2BYTE
                             00F8
                             00F8         ;
                             00F8         ;       DISPATCH_2BYTE - Branch to emulation routine for 2-byte opcode
                             00F8         ;
                             00F8         ;               entered by branching
                             00F8         ;
                             00F8         ;               R0 contains second opcode byte
                             00F8         ;
                             00F8         ;
                             00F8   DISPATCH_2BYTE:
 4D 8F    32    50      8F   00F8         CASEB   R0,#^X32,#<^X7F-^X32>   ; Covers CVTDH through PUSHAO
                     0503'   00FD   1$:   .WORD   INST_CVTDH-1$           ; 32FD  CVTDH
                     050B'   00FF         .WORD   INST_CVTGF-1$           ; 33FD  CVTGF
                     009C'   0101         .WORD   10$-1$                  ; 34FD
                     009C'   0103         .WORD   10$-1$                  ; 35FD
                     009C'   0105         .WORD   10$-1$                  ; 36FD
                     009C'   0107         .WORD   10$-1$                  ; 37FD
                     009C'   0109         .WORD   10$-1$                  ; 38FD
                     009C'   010B         .WORD   10$-1$                  ; 39FD
                     009C'   010D         .WORD   10$-1$                  ; 3AFD
                     009C'   010F         .WORD   10$-1$                  ; 3BFD
                     009C'   0111         .WORD   10$-1$                  ; 3CFD
                     009C'   0113         .WORD   10$-1$                  ; 3DFD
                     009C'   0115         .WORD   10$-1$                  ; 3EFD
                     009C'   0117         .WORD   10$-1$                  ; 3FFD
                     0315'   0119         .WORD   INST_ADDG2-1$           ; 40FD  ADDG2
                     034E'   011B         .WORD   INST_ADDG3-1$           ; 41FD  ADDG3
                     0D2A'   011D         .WORD   INST_SUBG2-1$           ; 42FD  SUBG2
                     0D66'   011F         .WORD   INST_SUBG3-1$           ; 43FD  SUBG3
                     0912'   0121         .WORD   INST_MULG2-1$           ; 44FD  MULG2
                     094B'   0123         .WORD   INST_MULG3-1$           ; 45FD  MULG3
                     059C'   0125         .WORD   INST_DIVG2-1$           ; 46FD  DIVG2
                     05D5'   0127         .WORD   INST_DIVG3-1$           ; 47FD  DIVG3
                     0459'   0129         .WORD   INST_CVTGB-1$           ; 48FD  CVTGB
                     0479'   012B         .WORD   INST_CVTGW-1$           ; 49FD  CVTGW
                     0499'   012D         .WORD   INST_CVTGL-1$           ; 4AFD  CVTGL
                     0561'   012F         .WORD   INST_CVTRGL-1$          ; 4BFD  CVTRGL
                     03DB'   0131         .WORD   INST_CVTBG-1$           ; 4CFD  CVTBG
                     03FB'   0133         .WORD   INST_CVTWG-1$           ; 4DFD  CVTWG
                     041B'   0135         .WORD   INST_CVTLG-1$           ; 4EFD  CVTLG
                     0297'   0137         .WORD   INST_ACBG-1$            ; 4FFD  ACBG
                     08A5'   0139         .WORD   INST_MOVG-1$            ; 50FD  MOVG
                     03A0'   013B         .WORD   INST_CMPG-1$            ; 51FD  CMPG
                     084E'   013D         .WORD   INST_MNEGG-1$           ; 52FD  MNEGG
                     0DA5'   013F         .WORD   INST_TSTG-1$            ; 53FD  TSTG
                     06D1'   0141         .WORD   INST_EMODG-1$           ; 54FD  EMODG
                     0A89'   0143         .WORD   INST_POLYG-1$           ; 55FD  POLYG
                     0513'   0145         .WORD   INST_CVTGH-1$           ; 56FD  CVTGH
                     009C'   0147         .WORD   10$-1$                  ; 57FD
                     009C'   0149         .WORD   10$-1$                  ; 58FD
                     009C'   014B         .WORD   10$-1$                  ; 59FD
                     009C'   014D         .WORD   10$-1$                  ; 5AFD
                     009C'   014F         .WORD   10$-1$                  ; 5BFD
                     009C'   0151         .WORD   10$-1$                  ; 5CFD
                     009C'   0153         .WORD   10$-1$                  ; 5DFD
                     009C'   0155         .WORD   10$-1$                  ; 5EFD
```

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

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

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

```
                0297            .SBTTL  INSTRUCTION_TYPES
                0297
                0297    ;+
                0297    ; The following two tables provide information on what floating types
                0297    ; are manipulated by each instruction.  For each opcode there are
                0297    ; four bits which, if set, indicate that the instruction uses D, F, G
                0297    ; and H_floating data, respectively.  These four bits are compared against
                0297    ; the corresponding four bits in EXE$GL_ARCHFLAG to see if the next
                0297    ; instruction should be emulated; this test takes place in NORMAL_EXIT.
                0297    ;-
                0297
                0297
                0297            ;+
                0297            ; Macro INST_TYPE generates table entries
                0297            ;-
                0297
                0297            .MACRO  INST_TYPE       OPCODE1,OPCODE2
                0297            $$x=0
                0297            .IRPC   $$t,<OPCODE2>
                0297            $$x=$$x+IT_'$$t'
                0297            .ENDR
                0297            $$x=$$x@4
                0297            .IRPC   $$t,<OPCODE1>
                0297            $$x=$$x+IT_'$$t'
                0297            .ENDR
                0297            .BYTE   $$x
                0297            .ENDM
                0297
                0297    ;+
                0297    ; INST_TYPES_1BYTE - Masks for 1-byte instructions.
                0297    ; Covers opcodes 40 (ADDF2) through 76 (CVTDF)
                0297    ;-
                0297
00000277        0297    INST_TYPES_1BYTE=.-<^X40/2>     ; Offset for first opcode
                0297            INST_TYPE       F,F             ; ADDF2, ADDF3
                0298            INST_TYPE       F,F             ; SUBF2, SUBF3
                0299            INST_TYPE       F,F             ; MULF2, MULF3
                029A            INST_TYPE       F,F             ; DIVF2, DIVF3
                029B            INST_TYPE       F,F             ; CVTFB, CVTFW
                029C            INST_TYPE       F,F             ; CVTFL, CVTRFL
                029D            INST_TYPE       F,F             ; CVTBF, CVTWF
                029E            INST_TYPE       F,F             ; CVTLF, ACBF
                029F            INST_TYPE       F,F             ; MOVF, CMPF
                02A0            INST_TYPE       F,F             ; MNEGF, TSTF
                02A1            INST_TYPE       F,F             ; EMODF, POLYF
                02A2            INST_TYPE       FD,X            ; CVTFD, <57>
                02A3            INST_TYPE       X,X             ; ADAWI, <59>
                02A4            INST_TYPE       X,X             ; <5A>, <5B>
                02A5            INST_TYPE       X,X             ; INSQHI, INSQTI
                02A6            INST_TYPE       X,X             ; REMQHI, REMQTI
                02A7            INST_TYPE       D,D             ; ADDD2, ADDD3
                02A8            INST_TYPE       D,D             ; SUBD2, SUBD3
                02A9            INST_TYPE       D,D             ; MULD2, MULD3
                02AA            INST_TYPE       D,D             ; DIVD2, DIVD3
                02AB            INST_TYPE       D,D             ; CVTDB, CVTDW
                02AC            INST_TYPE       D,D             ; CVTDL, CVTRDL
                02AD            INST_TYPE       D,D             ; CVTBD, CVTWD
```

```
          02AE                 INST_TYPE       D,D             ; CVTLD, ACBD
          02AF                 INST_TYPE       D,D             ; MOVD, CMPD
          02B0                 INST_TYPE       D,D             ; MNEGD, TSTD
          02B1                 INST_TYPE       D,D             ; EMODD, POLYD
          02B2                 INST_TYPE       FD,X            ; CVTDF, <77>
          02B3
          02B3      ;+
          02B3      ; INST_TYPES_2BYTE - Masks for 2-byte instructions.
          02B3      ; Covers opcodes 32FD (CVTDH) through 7FFD (PUSHAO)
          02B3      ;-
          02B3
0000029A  02B3      INST_TYPES_2BYTE=.-<^X32/2>     ; Offset for first opcode
          02B3                 INST_TYPE       DH,GF           ; CVTDH, CVTGF
          02B4                 INST_TYPE       X,X             ; <34FD>, <35FD>
          02B5                 INST_TYPE       X,X             ; <36FD>, <37FD>
          02B6                 INST_TYPE       X,X             ; <38FD>, <39FD>
          02B7                 INST_TYPE       X,X             ; <3AFD>, <3BFD>
          02B8                 INST_TYPE       X,X             ; <3CFD>, <3DFD>
          02B9                 INST_TYPE       X,X             ; <3EFD>, <3FFD>
          02BA                 INST_TYPE       G,G             ; ADDG2, ADDG3
          02BB                 INST_TYPE       G,G             ; SUBG2, SUBG3
          02BC                 INST_TYPE       G,G             ; MULG2, MULG3
          02BD                 INST_TYPE       G,G             ; DIVG2, DIVG3
          02BE                 INST_TYPE       G,G             ; CVTGB, CVTGW
          02BF                 INST_TYPE       G,G             ; CVTGL, CVTRGL
          02C0                 INST_TYPE       G,G             ; CVTBG, CVTWG
          02C1                 INST_TYPE       G,G             ; CVTLG, ACBG
          02C2                 INST_TYPE       G,G             ; MOVG, CMPG
          02C3                 INST_TYPE       G,G             ; MNEGG, TSTG
          02C4                 INST_TYPE       G,G             ; EMODG, POLYG
          02C5                 INST_TYPE       GH,X            ; CVTGH, <57FD>
          02C6                 INST_TYPE       X,X             ; <58FD>, <59FD>
          02C7                 INST_TYPE       X,X             ; <5AFD>, <5BFD>
          02C8                 INST_TYPE       X,X             ; <5CFD>, <5DFD>
          02C9                 INST_TYPE       X,X             ; <5EFD>, <5FFD>
          02CA                 INST_TYPE       H,H             ; ADDH2, ADDH3
          02CB                 INST_TYPE       H,H             ; SUBH2, SUBH3
          02CC                 INST_TYPE       H,H             ; MULH2, MULH3
          02CD                 INST_TYPE       H,H             ; DIVH2, DIVH3
          02CE                 INST_TYPE       H,H             ; CVTHB, CVTHW
          02CF                 INST_TYPE       H,H             ; CVTHL, CVTRHL
          02D0                 INST_TYPE       H,H             ; CVTBH, CVTWH
          02D1                 INST_TYPE       H,H             ; CVTLH, ACBH
          02D2                 INST_TYPE       H,H             ; MOVH, CMPH
          02D3                 INST_TYPE       H,H             ; MNEGH, TSTH
          02D4                 INST_TYPE       H,H             ; EMODH, POLYH
          02D5                 INST_TYPE       HG,X            ; CVTHG, <77FD>
          02D6                 INST_TYPE       X,X             ; <78FD>, <79FD>
          02D7                 INST_TYPE       X,X             ; <7AFD>, <7BFD>
          02D8                 INST_TYPE       H,H             ; CLRO, MOVO
          02D9                 INST_TYPE       H,H             ; MOVAO, PUSHAO
```

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

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

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

```
0388          .SBTTL  Instruction Emulation Routines
0388
0388  ;       ****************************************************************
0388  ;       *                                                              *
0388  ;       *                                                              *
0388  ;       *              Instruction Emulation Routines                  *
0388  ;       *                                                              *
0388  ;       *                                                              *
0388  ;       ****************************************************************
0388  ;
0388  ;
0388  ;       Introduction
0388  ;       ------------
0388  ;
0388  ;           Following are the routines for emulating the individual
0388  ;       new instructions. There is one routine for each instruction
0388  ;       but the structure of most of these routines is extremely
0388  ;       simple so we have included all of the descriptive information
0388  ;       here rather than duplicating it for each routine. Special
0388  ;       discussions are given for those instructions or families of
0388  ;       instructions which do not quite fit into the general patterns.
0388  ;
0388  ;           The routines themselves have been written so that they
0388  ;       will be easy to follow. Because of this the temptation to
0388  ;       remove a considerable amount of duplicated code has been
0388  ;       staunchly resisted.
0388  ;
0388  ;           The emulation routines have names which are of the form
0388  ;       "INST_mnemonic", are entered by branching from the routine
0388  ;       EMULATOR, and when instruction is complete the routines branch
0388  ;       to NORMAL_EXIT. When exceptions occur, then flow of control is
0388  ;       somewhat more complicated and is described below.
0388  ;
0388  ;
0388  ;       Operand Processing
0388  ;       ------------------
0388  ;
0388  ;           The first step for each of the instructions is to process
0388  ;       the instruction operands (this process is inhibited only for
0388  ;       POLYx when FPD is set in the PSL). This is done by
0388  ;       calling the operand scanning routines for each operand and by
0388  ;       saving the operands values (for read and modify access
0388  ;       operands) and operand addresses (for write, modify, and
0388  ;       address, access operands) in the special areas of the frame
0388  ;       allocated for that purpose. Floating values of all types that
0388  ;       are input are converted to our internal floating format by the
0388  ;       UNPACK routines which are described below.
0388  ;
0388  ;
0388  ;       Instruction Emulation
0388  ;       ---------------------
0388  ;
0388  ;           After all of the operands have been processed, the action
0388  ;       of the instruction is performed usually by calling one of the
0388  ;       arithmetic routines. For many of the move and convert
0388  ;       instructions no special emulation action is necessary since
0388  ;       everything is done by the PACK and UNPACK routines. All of the
```

```
0388   ;       floating arithmetic is performed using our internal floating
0388   ;       representation.
0388   ;
0388   ;
0388   ;       Storing the Results
0388   ;       -------------------
0388   ;
0388   ;           The last step of instruction emulation is storing the
0388   ;       resulting values and updating the condition codes. The values
0388   ;       are stored in the order in which their corresponding operands
0388   ;       occur so the instruction is emulated properly when the write
0388   ;       or modify access operands overlap. Floating values are
0388   ;       converted from the internal floating representation to the
0388   ;       target representation by one of the PACK routines described
0388   ;       below. After the results are output, the condition codes in
0388   ;       the emulated PSL are updated. This order of operation is
0388   ;       important since the PACK routines may convert very small
0388   ;       nonzero values to zero values if an underflow occurs and FU is
0388   ;       clear in the PSL. When the instruction emulation is complete
0388   ;       the routine branches to NORMAL_EXIT.
0388   ;
0388   ;
0388   ;       Exceptions
0388   ;       ----------
0388   ;
0388   ;           Except for the integer overflow trap, all of the
0388   ;       exceptions that the Emulator checks for are faults. When
0388   ;       faults occur the Emulator restores everything to a state in
0388   ;       which the instruction can be restarted and signals the fault.
0388   ;       For the integer overflow trap, the instruction is run to
0388   ;       completion, possibly outputing a truncated result, and the
0388   ;       trap is signaled. Here we describe how instruction emulation
0388   ;       is organized so this can be accomplished. Further information
0388   ;       can be found in the essay on exception processing which
0388   ;       appears below.
0388   ;
0388   ;
0388   ;       Faults
0388   ;
0388   ;           Until the results of an instruction are output, all of the
0388   ;       changes which take place to any register are additions and
0388   ;       subtractions of small integers. These changes are also
0388   ;       recorded in the change bytes corresponding to the registers.
0388   ;       Therefore the only requirement necessary for correct fault
0388   ;       processing is to insure that all fault detection takes place
0388   ;       before any of the instruction results are output (actually,
0388   ;       things are a little more complicated for POLYx).
0388   ;       This is done by making the result outputing steps the last
0388   ;       part of instruction emulation. In this way fault detection
0388   ;       can be placed anywhere in the Emulator it is convenient rather
0388   ;       than requiring close synchronization with the instruction
0388   ;       emulation logic. For this reason direct processing of faults
0388   ;       (rather than, say, returning status codes) appears throughout
0388   ;       the common Emulator subroutines.
0388   ;
0388   ;           The architecture makes no special requirements on the
0388   ;       the condition codes generated by a fault except that they
```

```
0388    ;       be sufficient for correctly restarting the instruction. In
0388    ;       general, the V bit is cleared by EMULATOR on entry to an
0388    ;       instruction emulation routine since the bit is used for
0388    ;       detecting integer overflow traps (see below). None of the
0388    ;       other condition bits is altered until the results are output
0388    ;       at the end of instruction emulation. Consequently all of the
0388    ;       other condition bits are preserved into faults. (With the
0388    ;       present set of emulated instructions the only preservation
0388    ;       requirement is that the C bit be preserved on faults by the
0388    ;       ACBx instructions.)
0388    ;
0388    ;
0388    ;       Integer Overflow Traps
0388    ;
0388    ;           When a integer overflow is detected (either in a common
0388    ;       subroutine or in the instruction emulation routine) the V
0388    ;       bit is set in the emulated condition codes and the value is
0388    ;       truncated. Everything proceeds normally until control reaches
0388    ;       NORMAL_EXIT where the V and IV bits in the emulated PSL are
0388    ;       checked. If both are set then an integer overflow trap is
0388    ;       signaled.
0388    ;
0388    ;
0388    ;       Access Modes and Access Violations
0388    ;
0388    ;           This version of the Emulator has been designed to operate
0388    ;       at the same access mode as the instruction being emulated.
0388    ;       However most of the instruction emulation logic has been
0388    ;       designed to operate at any access mode that is not less
0388    ;       privleged than that of the instruction (exceptions: the logic
0388    ;       for getting in, the logic for getting out, and the exception
0388    ;       signaling logic). All attempts to access memory on behalf of
0388    ;       the instruction are probed using the access mode contained in
0388    ;       the local cell MODE. This is normally set to the current
0388    ;       access mode but may be set to some less privleged access mode.
0388    ;       Because this version of the Emulator operates on the user
0388    ;       stack it is not necessary to probe stack operations, however,
0388    ;       these probes are still done to insure that the code will
0388    ;       remain usable for different access modes.
0388    ;
0388    ;
0388    ;       Notes on the ACBx Instructions
0388    ;       --------------------------------------
0388    ;
0388    ;           For these instructions the operands are processed, the
0388    ;       step is added to the index, and the sum is packed and output.
0388    ;       Afterwards the sum is unpacked again and compared with the
0388    ;       limit, the type of comparison depending on the sign of the
0388    ;       step. The packing and unpacking is necessary to truncate the
0388    ;       sum to the correct number of bits, to perform any rounding,
0388    ;       and to check for underflow or overflow exceptions. If the
0388    ;       branch is to be taken, the branch destination is moved into
0388    ;       the emulated PC.
0388    ;
0388    ;
0388    ;       Notes on the EMODx Instructions
0388    ;       ----------------------------------------
```

```
0388    ;
0388    ;         For these instructions the first two operands are a
0388    ;     floating value of the appropriate type and a word containing
0388    ;     some extension bits. When the extension bits are picked up
0388    ;     they are stored in the correct position of the unpacked first
0388    ;     operand. The remaining operands are scanned and flag bits 1
0388    ;     and 7 are used to remember which output operands are in the
0388    ;     registers. The multiply is then performed using the proper
0388    ;     arithmetic routine and the product is truncated to the correct
0388    ;     number of bits. Next the integer part is extracted and saved
0388    ;     in an internal cell while the fraction part is extracted and
0388    ;     packed. Next any register output operands are output and
0388    ;     the routine TEST_FRAME is called to insure that the user's
0388    ;     stack pointer is within the parameter area. Next any output
0388    ;     operands in memory are output with stores into the Emulator's
0388    ;     working storage disabled .If both output operands are in the
0388    ;     same type of storage, then the integer part is output before
0388    ;     the fraction part. This complicated method of outputing the
0388    ;     results is necessary because one of the register operands may
0388    ;     contain SP and extend the user's stack into the Emulator's
0388    ;     working storage. The call to TEST_FRAME will move the frame
0388    ;     if this occurs so if the other operand will be stored into
0388    ;     memory properly if it is not below the user's stack pointer.
0388    ;
0388    ;
0388    ;     Notes on the POLYx Instructions
0388    ;     ------------------------------------------
0388    ;
0388    ;         These instructions have especially complicated emulation
0388    ;     routines because they do a lot, have a large number of
0388    ;     implicit inputs and outputs, and are intended to be resumed
0388    ;     instead of restarted when faults occur. For this reason we
0388    ;     describe the action of the instruction emulation routines in
0388    ;     some detail.
0388    ;
0388    ;         These instructions can be thought as having two states.
0388    ;     In the first of these states no unreversable changes have
0388    ;     been made to the registers so if a fault occurs the
0388    ;     instruction can be restarted. The instruction remains in this
0388    ;     state until all of the operands have been processed and the
0388    ;     first coefficient has been validated. After this point the
0388    ;     instruction saves various things in the registers R0-R5 (and
0388    ;     on the stack for POLYH) so the instruction can no longer be
0388    ;     restarted but the information saved is sufficient for resuming
0388    ;     the instruction after a fault. When this second state is
0388    ;     reached the FPD bit is set in the PSL so the when the
0388    ;     instruction emulation routine is entered for resuming the
0388    ;     instruction it will know enough to do so.
0388    ;
0388    ;         Following is an outline of the steps in the emulation of
0388    ;     the two instructions. The text generally describes POLYG and
0388    ;     the differences for POLYH are noted in parentheses.
0388    ;
0388    ;
0388    ;         1. This is the first step for the instruction emulation.
0388    ;            If the FPD bit is clear in the PSL then control passes
0388    ;            to Step 2. Otherwise the following actions which are
```

```
0388  ;        concerned with restarting the instruction take place:
0388  ;
0388  ;        The length of the instruction operands is retrieved
0388  ;        from the high order three bytes of the user's R2 (R4
0388  ;        for POLYH) and added to the current PC and to the
0388  ;        change byte for the PC. If the instruction terminates
0388  ;        the PC will be positioned following the instruction
0388  ;        and if the instruction faults it will be reset to the
0388  ;        location of the instruction.
0388  ;
0388  ;        The result so far is unpacked from R0-R1 (R0-R3 for
0388  ;        POLYH) and the unpacked value is stored in OPERAND2.
0388  ;        The condition codes are set to describe this value.
0388  ;
0388  ;        The argument is unpacked from R4-R5 (from a stacked
0388  ;        octaword for POLYH, which is probed) and the unpacked
0388  ;        value is stored in OPERAND3.
0388  ;
0388  ;        The remaining coefficient count in the low order byte
0388  ;        of R2 (R4 for POLYH) is checked for validity.
0388  ;
0388  ;        Finally control passes to Step 3.
0388  ;
0388  ;
0388  ;     2. This step performs all of the processing for initial
0388  ;        entry to the instruction.
0388  ;
0388  ;        The first operand is processed and the unpacked
0388  ;        argument is saved in OPERAND3.
0388  ;
0388  ;        The second operand is processed and the degree is
0388  ;        checked for validity and the value is saved
0388  ;        temporarily in ADDRESS1.
0388  ;
0388  ;        The third operand is processed and the address of the
0388  ;        coefficient table is saved. The first coefficient is
0388  ;        probed, unpacked, and saved in OPERAND2.
0388  ;
0388  ;        If the instruction is POLYH then the user's SP is
0388  ;        decremented by 16 (and so is the change byte) and
0388  ;        the argument value in OPERAND3 is packed and output
0388  ;        to the allocated stack area. (If a fault occurs before
0388  ;        FPD is set then SP will be reinitilized properly.)
0388  ;
0388  ;        The coefficient in OPERAND2 is packed and stored in
0388  ;        the user's registers R0-R1 (R0-R4 for POLYH), the
0388  ;        address of the remaining coefficients is stored in the
0388  ;        user's R4 (R6 for POLYH), the degree is stored in the
0388  ;        low order byte of the user's R2 (R4 for POLYH), and
0388  ;        the length of the instruction operands (the PC change
0388  ;        byte minus two) is saved in the upper three bytes of
0388  ;        that register. The condition codes are set based on
0388  ;        the value in OPERAND2.
0388  ;
0388  ;        All of the change by'es except that for PC are cleared
0388  ;        and the bit FPD is s t in the emulated PSL.
0388  ;
```

```
0388  ;              If the instruction is POLYG then the argument in
0388  ;              OPERAND3 is packed and saved in the user's R4-R5.
0388  ;
0388  ;              Control Passes to Step 3.
0388  ;
0388  ;
0388  ;           3. This step performs the basic polynomial evaluation
0388  ;              iteration.
0388  ;
0388  ;              If the remaining coefficient count in the low order
0388  ;              byte of the user's R2 (R4 for POLYH) is zero then
0388  ;              control passes to Step 4.
0388  ;
0388  ;              The value so far in OPERAND2 and the argument in
0388  ;              OPERAND3 are multiplied and the unnormalized product
0388  ;              is truncated to 63 (127 for POLYH) bits and stored
0388  ;              in OPERAND1. The truncation is performed by clearing
0388  ;              the low order bit of the 64 (128 for POLYH) bit
0388  ;              product returned by the multiply routine.
0388  ;
0388  ;              The coefficient addressed by the user's R3 (R5 for
0388  ;              POLYH) is probed and unpacked to OPERAND2. The result
0388  ;              is then added to OPERAND1.
0388  ;
0388  ;              The new value so far in OPERAND1 is packed and stored
0388  ;              in the user's R0-R1 (R0-R3 for POLYH). The condition
0388  ;              codes are set based on this value. The value is then
0388  ;              unpacked and stored in OPERAND2.
0388  ;
0388  ;              The remaining coefficient count in the low order byte
0388  ;              of R2 (R4 for POLYH) is decremented and the pointer to
0388  ;              the next coeffiecient in R3 (R5 for POLYH) is
0388  ;              incremented to point to the next coefficient.
0388  ;
0388  ;              Control then passes to the beginning of this step.
0388  ;
0388  ;
0388  ;           4. This step performs the termination of the instruction.
0388  ;
0388  ;              The register R2 (R4 for POLYH) which contains the
0388  ;              instruction length and the coefficient count is
0388  ;              cleared.
0388  ;
0388  ;              If the instruction was POLYG then the registers R4-R5
0388  ;              which contained the argument value are cleared.
0388  ;
0388  ;              If the instruction was POLYH then the user's SP is
0388  ;              incremented by 16 to unstack the argument.
0388  ;
0388  ;              The FPD bit in the PSL is cleared.
0388  ;
0388  ;              Instruction emulation is now complete.
0388  ;
0388  ;
0388  ;          Careful reading of the above outline will show that the
0388  ;       POLYG and POLYH instructions are correctly emulated even down
0388  ;       to the handling of faults. Faults may be detected at any of
```

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

```
                    0388              ;
                    0388              ;       4F ACBF - Add Compare and Branch F_floating
                    0388              ;
                    0388   INST_ACBF:
                    0388              SET_OP_TYPES    F
          10   11   038C              BRB     INST_ACBx
                    038E
                    038E              ;
                    038E              ;       6F ACBD - Add Compare and Branch D_floating
                    038E              ;
                    038E   INST_ACBD:
                    038E              SET_OP_TYPES    D
          0A   11   0392              BRB     INST_ACBx
                    0394
                    0394              ;
                    0394              ;       4FFD ACBG - Add Compare and Branch G_floating
                    0394              ;
                    0394   INST_ACBG:
                    0394              SET_OP_TYPES    G
          04   11   0398              BRB     INST_ACBx
                    039A
                    039A              ;
                    039A              ;       6FFD ACBH - Add Compare and Branch H_floating
                    039A              ;
                    039A   INST_ACBH:
                    039A              SET_OP_TYPES    H
                    039E              ; BRB   INST_ACBx
                    039E
                    039E   INST_ACBx:
        0B22   30   039E              BSBW    READ_ACCESS             ; first operand is read only
        0E8C   30   03A1              BSBW    UNPACK_FLOAT3           ; unpack and save the value
        0B1C   30   03A4              BSBW    READ_ACCESS             ; second operand is read only
        0E80   30   03A7              BSBW    UNPACK_FLOAT2           ; unpack and save the value
        0B28   30   03AA              BSBW    MODIFY_ACCESS           ; third operand is modif'ed
  E7 AD   5B   D0   03AD              MOVL    R11,ADDRESS1(FP)        ; save the operand address
        0E70   30   03B1              BSBW    UNPACK_FLOAT1           ; unpack and save the value
        0E35   30   03B4              BSBW    BRANCH_WORD             ; fourth operand is branch destination
  E3 AD   5B   D0   03B7              MOVL    R11,ADDRESS2(FP)        ; save the branch destination
        1260   30   03BB              BSBW    ADD_REAL                ; add the step and index
        0F53   30   03BE              BSBW    PACK_FLOAT1             ; pack the index value
  5B   E7 AD   D0   03C1              MOVL    ADDRESS1(FP),R11        ; R11 = destination address
        10F0   30   03C5              BSBW    STORE_OPERAND           ; store into third operand
        0E59   30   03C8              BSBW    UNPACK_FLOAT1           ; put the value back
  51   C7 AD   9E   03CB              MOVAB   OPERAND1(FP),R1         ; R1 = location of index value
        177A   30   03CF              BSBW    TEST_REAL               ; test the value
        174B   30   03D2              BSBW    SET_CONDITION           ; set the condition codes
  54 AD   02   8A   03D5              BICB2   #PSL$M_V,PSL(FP)        ; clear V in the PSL
  51   AF AD   9E   03D9              MOVAB   OPERAND2(FP),R1         ; R1 = address of step value
        176C   30   03DD              BSBW    TEST_REAL               ; test the value
          14   19   03E0              BLSS    2$                      ; it's negative - bypass
  51   C7 AD   9E   03E2              MOVAB   OPERAND1(FP),R1         ; R1 = address of index value
  52   97 AD   9E   03E6              MOVAB   OPERAND3(FP),R2         ; R2 = address of limit value
        177D   30   03EA              BSBW    COMPARE_REAL            ; have we passed the limit ?
          14   14   03ED              BGTR    3$                      ; yes - bypass
50 AD  E3 AD   D0   03EF   1$:        MOVL    ADDRESS2(FP),REG_PC(FP) ; perform the branch
          0D   11   03F4              BRB     3$                      ; bypass
  51   C7 AD   9E   03F6   2$:        MOVAB   OPERAND1(FP),R1         ; R1 = address of index value
```

```
52   97 AD  9E  03FA         MOVAB   OPERAND3(FP),R2          ; R2 = address of limit value
      1769  30  03FE         BSBW    COMPARE_REAL             ; have we passed the limit ?
        EC  18  0401         BGEQ    1$                       ; no - perform the branch
      FDBE  31  0403  3$:    BRW     NORMAL_EXIT              ; done
                0406
                0406   ;
                0406   ;       40 ADDF2 - Add F_floating (Two Operands)
                0406   ;
                0406  INST_ADDF2:                             ; entrance
                0406         SET_OP_TYPES    F
        10  11  040A         BRB     INST_ADDx2
                040C
                040C   ;
                040C   ;       60 ADDD2 - Add D_floating (Two Operands)
                040C   ;
                040C  INST_ADDD2:                             ; entrance
                040C         SET_OP_TYPES    D
        0A  11  0410         BRB     INST_ADDx2
                0412
                0412   ;
                0412   ;       40FD ADDG2 - Add G_floating (Two Operands)
                0412   ;
                0412  INST_ADDG2:                             ; entrance
                0412         SET_OP_TYPES    G
        04  11  0416         BRB     INST_ADDx2
                0418
                0418   ;
                0418   ;       60FD ADDH2 - Add HFLOAT (Two Operands)
                0418   ;
                0418  INST_ADDH2:                             ; entrance
                0418         SET_OP_TYPES    H
                041C         ;BRB    INST_ADDx2
                041C
                041C  INST_ADDx2:
      0AA4  30  041C         BSBW    READ_ACCESS              ; first operand is read only
      0E02  30  041F         BSBW    UNPACK_FLOAT1            ; unpack and save the value
      0AB0  30  0422         BSBW    MODIFY_ACCESS            ; second operand is modified
E7 AD   5B  D0  0425         MOVL    R11,ADDRESS1(FP)         ; save the destination location
      0DFE  30  0429         BSBW    UNPACK_FLOAT2            ; unpack and save the value
      11EF  30  042C         BSBW    ADD_REAL                 ; compute the sum
      0EE2  30  042F         BSBW    PACK_FLOAT1              ; pack the sum
5B   E7 AD  D0  0432         MOVL    ADDRESS1(FP),R11         ; R11 = destination location
      107F  30  0436         BSBW    STORE_OPERAND            ; store the result
      1703  30  0439         BSBW    SET_CONDITION1           ; set the condition codes in the PSL
      FD85  31  043C         BRW     NORMAL_EXIT              ; done
                043F
                043F   ;
                043F   ;       41 ADDF3 - Add F_floating (Three Operands)
                043F   ;
                043F  INST_ADDF3:                             ; entrance
                043F         SET_OP_TYPES    F
        10  11  0443         BRB     INST_ADDx3
                0445
                0445   ;
                0445   ;       61 ADDD3 - Add D_floating (Three Operands)
                0445   ;
                0445  INST_ADDD3:                             ; entrance
```

```
                        0445          SET_OP_TYPES    D
             0A   11    0449          BRB     INST_ADDx3
                        044B
                        044B          ;
                        044B          ;       41FD ADDG3 - Add G_floating (Three Operands)
                        044B          ;
                        044B  INST_ADDG3:                                 ; entrance
                        044B          SET_OP_TYPES    G
             04   11    044F          BRB     INST_ADDx3
                        0451
                        0451          ;
                        0451          ;       61FD ADDH3 - Add HFLOAT (Three Operands)
                        0451          ;
                        0451  INST_ADDH3:                                 ; entrance
                        0451          SET_OP_TYPES    H
                        0455          ;BRB    INST_ADDx3
                        0455
                        0455  INST_ADDx3:
           0A6B   30    0455          BSBW    READ_ACCESS                 ; first operand is read only
           0DC9   30    0458          BSBW    UNPACK_FLOAT1               ; unpack and save the value
           0A65   30    045B          BSBW    READ_ACCESS                 ; second operand is read only
           0DC9   30    045E          BSBW    UNPACK_FLOAT2               ; unpack and save the value
           0A68   30    0461          BSBW    WRITE_ACCESS                ; third operand is write only
   E7 AD     5B   D0    0464          MOVL    R11,ADDRESS1(FP)            ; save the destination address
           11B3   30    0468          BSBW    ADD_REAL                    ; compute the sum
           0EA6   30    046B          BSBW    PACK_FLOAT1                 ; pack the sum
   5B     E7 AD   D0    046E          MOVL    ADDRESS1(FP),R11            ; R11 = destination location
           1043   30    0472          BSBW    STORE_OPERAND               ; store the result
           16C7   30    0475          BSBW    SET_CONDITION1              ; set the condition codes in the PSL
           FD49   31    0478          BRW     NORMAL_EXIT                 ; done
                        047B
                        047B          ;
                        047B          ;       7CFD CLRO - Clear Octaword
                        047B          ;
                        047B          ;       7CFD CLRH - Clear H_floating
                        047B          ;
                        047B  INST_CLRO:                                  ; entrance
                        047B          SET_OP_TYPES    O
           0A4A   30    047F          BSBW    WRITE_ACCESS                ; first operand is write only
             8B   7C    0482          CLRQ    (R11)+                      ; clear the first part of the value
             6B   7C    0484          CLRQ    (R11)                       ; clear the second part of the value
   54 AD     0E   CA    0486          BICL2   #PSLM_NZV,PSL(FP)           ; clear the condition codes except C
   54 AD     04   C8    048A          BISL2   #PSLM_Z,PSL(FP)             ; set the Z bit in the PSL
           FD33   31    048E          BRW     NORMAL_EXIT                 ; done
                        0491
                        0491          ;
                        0491          ;       51 CMPF - Compare F_floating
                        0491          ;
                        0491  INST_CMPF:                                  ; entrance
                        0491          SET_OP_TYPES    F
             10   11    0495          BRB     INST_CMPx
                        0497
                        0497          ;
                        0497          ;       71 CMPD - Compare D_floating
                        0497          ;
                        0497  INST_CMPD:                                  ; entrance
                        0497          SET_OP_TYPES    D
```

```
           0A   11  049B              BRB     INST_CMPx
                    049D
                    049D              ;
                    049D              ;       51FD CMPG - Compare G_floating
                    049D              ;
                    049D      INST_CMPG:                                ; entrance
                    049D              SET_OP_TYPES    G
           04   11  04A1              BRB     INST_CMPx
                    04A3
                    04A3              ;
                    04A3              ;       71FD CMPH - Compare H_floating
                    04A3              ;
                    04A3      INST_CMPH:                                ; entrance
                    04A3              SET_OP_TYPES    H
                    04A7              ;BRB    INST_CMPx
                    04A7
                    04A7      INST_CMPx:
         0A19   30  04A7              BSBW    READ_ACCESS               ; first operand is read only
         0D77   30  04AA              BSBW    UNPACK_FLOAT1             ; unpack and save the value
         0A13   30  04AD              BSBW    READ_ACCESS               ; second operand is read only
         0D77   30  04B0              BSBW    UNPACK_FLOAT2             ; unpack and save the value
   51   C7 AD   9E  04B3              MOVAB   OPERAND1(FP),R1           ; R1 = location of first value
   52   AF AD   9E  04B7              MOVAB   OPERAND2(FP),R2           ; R2 = location of second value
         16AC   30  04BB              BSBW    COMPARE_REAL              ; compare the values
         165F   30  04BE              BSBW    SET_CONDITION             ; set the condition codes
   54 AD   03   CA  04C1              BICL2   #PSL$M_VC,PSL(FP)         ; clear the V bit and C bit in the PSL
         FCFC   31  04C5              BRW     NORMAL_EXIT               ; done
                    04C8
                    04C8              ;
                    04C8              ;       4C CVTBF - Convert Byte to F_floating
                    04C8              ;
                    04C8      INST_CVTBF:                               ; entrance
                    04C8              SET_OP_TYPES    B,F
           56   11  04CE              BRB     INST_CVTBx
                    04D0
                    04D0              ;
                    04D0              ;       6C CVTBD - Convert Byte to D_floating
                    04D0              ;
                    04D0      INST_CVTBD:                               ; entrance
                    04D0              SET_OP_TYPES    B,D
           4E   11  04D6              BRB     INST_CVTBx
                    04D8
                    04D8              ;
                    04D8              ;       4CFD CVTBG - Convert Byte to G_floating
                    04D8              ;
                    04D8      INST_CVTBG:                               ; entrance
                    04D8              SET_OP_TYPES    B,G
           46   11  04DE              BRB     INST_CVTBx
                    04E0
                    04E0              ;
                    04E0              ;       6CFD CVTBH - Convert Byte to H_floating
                    04E0              ;
                    04E0      INST_CVTBH:                               ; entrance
                    04E0              SET_OP_TYPES    B,H
           3E   11  04E6              BRB     INST_CVTBx
                    04E8
                    04E8              ;
```

```
              04E8          ;       4D CVTWF - Convert Word to F_floating
              04E8          ;
              04E8  INST_CVTWF:
              04E8          SET_OP_TYPES    W,F
   36    11   04EE          BRB     INST_CVTWx
              04F0
              04F0          ;
              04F0          ;       6D CVTWD - Convert Word to D_floating
              04F0          ;
              04F0  INST_CVTWD:
              04F0          SET_OP_TYPES    W,D
   2E    11   04F6          BRB     INST_CVTWx
              04F8
              04F8          ;
              04F8          ;       4DFD CVTWG - Convert Word to G_floating
              04F8          ;
              04F8  INST_CVTWG:                             ; entrance
              04F8          SET_OP_TYPES    W,G
   26    11   04FE          BRB     INST_CVTWx
              0500
              0500          ;
              0500          ;       6DFD CVTWH - Convert Word to H_floating
              0500          ;
              0500  INST_CVTWH:                             ; entrance
              0500          SET_OP_TYPES    W,H
   1E    11   0506          BRB     INST_CVTWx
              0508
              0508          ;
              0508          ;       4E CVTLF - Convert Long to F_floating
              0508          ;
              0508  INST_CVTLF:                             ; entrance
              0508          SET_OP_TYPES    L,F
   16    11   050E          BRB     INST_CVTLx
              0510
              0510          ;
              0510          ;       6E CVTLD - Convert Long to D_floating
              0510          ;
              0510  INST_CVTLD:                             ; entrance
              0510          SET_OP_TYPES    L,D
   0E    11   0516          BRB     INST_CVTLx
              0518
              0518          ;
              0518          ;       4EFD CVTLG - Convert Long to G_floating
              0518          ;
              0518  INST_CVTLG:                             ; entrance
              0518          SET_OP_TYPES    L,G
   06    11   051E          BRB     INST_CVTLx
              0520
              0520          ;
              0520          ;       6EFD CVTLH - Convert Long to H_floating
              0520          ;
              0520  INST_CVTLH:                             ; entrance
              0520          SET_OP_TYPES    L,H
              0526          ;BRB    INST_CVTLx
              0526
              0526  INST_CVTBx:
              0526  INST_CVTWx:
```

```
                      0526  INST_CVTLx:
         099A   30    0526          BSBW    READ_ACCESS             ; first operand is read only
         0FC0   30    0529          BSBW    FLOAT_LONG              ; convert to floating and save value
       8B AD    D6    052C          INCL    OP_INDEX(FP)            ; move to second operand
         099A   30    052F          BSBW    WRITE_ACCESS            ; second operand is write only
  E7 AD   5B    D0    0532          MOVL    R11,ADDRESS1(FP)        ; save the destination address
         0DDB   30    0536          BSBW    PACK_FLOAT1             ; pack the value
  5B   E7 AD    D0    0539          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
         0F78   30    053D          BSBW    STORE_OPERAND           ; store the result
         15FC   30    0540          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
         FC7E   31    0543          BRW     NORMAL_EXIT             ; done
                      0546
                      0546          ;
                      0546          ;       48 CVTFB - Convert F_floating to Byte
                      0546          ;
                      0546  INST_CVTFB:
                      0546          SET_OP_TYPES    F,B
           58   11    054C          BRB     INST_CVTxB
                      054E
                      054E          ;
                      054E          ;       68 CVTDB - Convert D_floating to Byte
                      054E          ;
                      054E  INST_CVTDB:
                      054E          SET_OP_TYPES    D,B
           50   11    0554          BRB     INST_CVTxB
                      0556
                      0556          ;
                      0556          ;       48FD CVTGB - Convert G_floating to Byte
                      0556          ;
                      0556  INST_CVTGB:
                      0556          SET_OP_TYPES    G,B
           48   11    055C          BRB     INST_CVTxB
                      055E
                      055E          ;
                      055E          ;       68FD CVTHB - Convert H_floating to Byte
                      055E          ;
                      055E  INST_CVTHB:
                      055E          SET_OP_TYPES    H,B
           40   11    0564          BRB     INST_CVTxB
                      0566
                      0566          ;
                      0566          ;       49 CVTFW - Convert F_floating to Word
                      0566          ;
                      0566  INST_CVTFW:
                      0566          SET_OP_TYPES    F,W
           38   11    056C          BRB     INST_CVTxW
                      056E
                      056E          ;
                      056E          ;       69 CVTDW - Convert D_floating to Word
                      056E          ;
                      056E  INST_CVTDW:
                      056E          SET_OP_TYPES    D,W
           30   11    0574          BRB     INST_CVTxW
                      0576
                      0576          ;
                      0576          ;       49FD CVTGW - Convert G_floating to Word
                      0576          ;
```

```
                         0576  INST_CVTGW:
                         0576          SET_OP_TYPES    G,W
            28   11      057C          BRB     INST_CVTxW
                         057E
                         057E          ;
                         057E          ;       69FD CVTHW - Convert H_floating to Word
                         057E          ;
                         057E  INST_CVTHW:
                         057E          SET_OP_TYPES    H,W
            20   11      0584          BRB     INST_CVTxW
                         0586
                         0586          ;
                         0586          ;       4A CVTFL - Convert F_floating to Long
                         0586          ;
                         0586  INST_CVTFL:
                         0586          SET_OP_TYPES    F,L
            18   11      058C          BRB     INST_CVTxL
                         058E
                         058E          ;
                         058E          ;       6A CVTDL - Convert D_floating to Long
                         058E          ;
                         058E  INST_CVTDL:
                         058E          SET_OP_TYPES    D,L
            10   11      0594          BRB     INST_CVTxL
                         0596
                         0596          ;
                         0596          ;       4AFD CVTGL - Convert G_floating to Long
                         0596          ;
                         0596  INST_CVTGL:
                         0596          SET_OP_TYPES    G,L
            08   11      059C          BRB     INST_CVTxL
                         059E
                         059E          ;
                         059E          ;       6AFD CVTHL - Convert H_floating to Long
                         059E          ;
                         059E  INST_CVTHL:
                         059E          SET_OP_TYPES    H,L
            00   11      05A4          BRB     INST_CVTxL
                         05A6
                         05A6  INST_CVTxB:
                         05A6  INST_CVTxW:
                         05A6  INST_CVTxL:
          091A   30      05A6          BSBW    READ_ACCESS             ; first operand is read only
          0C78   30      05A9          BSBW    UNPACK_FLOAT1           ; unpack and save the value
         8B AD   D6      05AC          INCL    OP_INDEX(FP)            ; move to second operand
          091A   30      05AF          BSBW    WRITE_ACCESS            ; second operand is write only
   E7 AD    5B   D0      05B2          MOVL    R11,ADDRESS1(FP)        ; save the destination location
   54 AD    0F   CA      05B6          BICL2   #PSLM_NZVC,PSL(FP)      ; clear the condition codes
          0F67   30      05BA          BSBW    FIX_REAL                ; fix the value
   02    8B BD   91      05BD          CMPB    @OP_INDEX(FP),#TYP_W    ; case on datatype
            10   14      05C1          BGTR    30$                     ; skip if longword
            05   13      05C3          BEQL    10$                     ; skip if word
      50    50   F6      05C5          CVTLB   R0,R0                   ; convert long to byte
            03   11      05C8          BRB     20$                     ; continue
      50    50   F7      05CA  10$:    CVTLW   R0,R0                   ; convert long to word
            04   1C      05CD  20$:    BVC     30$                     ; no overflow - skip
   54 AD    02   C8      05CF          BISL2   #PSLM_V,PSL(FP)         ; indicate integer overflow
```

```
5B  E7 AD  D0 05D3   30$:    MOVL    ADDRESS1(FP),R11        ; get address of result
      0EDE 30 05D7           BSBW    STORE_OPERAND           ; store the result
      1543 30 05DA           BSBW    SET_CONDITION           ; set the condition codes
      FBE4 31 05DD           BRW     NORMAL_EXIT             ; done
              05E0
              05E0           ;
              05E0           ;       56 CVTFD - Convert F_floating to D_floating
              05E0           ;
              05E0   INST_CVTFD:
              05E0           SET_OP_TYPES    F,D
        46 11 05E6           BRB     INST_CVTxy
              05E8
              05E8           ;
              05E8           ;       99FD CVTFG - Convert Floating to G_floating
              05E8           ;
              05E8   INST_CVTFG:                             ; entrance
              05E8           SET_OP_TYPES    F,G
        3E 11 05EE           BRB     INST_CVTxy
              05F0
              05F0           ;
              05F0           ;       98FD CVTFH - Convert F_floating to H_floating
              05F0           ;
              05F0   INST_CVTFH:                             ; entrance
              05F0           SET_OP_TYPES    F,H
        36 11 05F6           BRB     INST_CVTxy
              05F8
              05F8           ;
              05F8           ;       76 CVTDF - Convert D_floating to H_floating
              05F8           ;
              05F8   INST_CVTDF:                             ; entrance
              05F8           SET_OP_TYPES    D,F
        2E 11 05FE           BRB     INST_CVTxy
              0600
              0600           ;
              0600           ;       32FD CVTDH - Convert D_floating to H_floating
              0600           ;
              0600   INST_CVTDH:
              0600           SET_OP_TYPES    D,H
        26 11 0606           BRB     INST_CVTxy
              0608
              0608           ;
              0608           ;       33FD CVTGF - Convert G_floating to F_floating
              0608           ;
              0608   INST_CVTGF:
              0608           SET_OP_TYPES    G,F
        1E 11 060E           BRB     INST_CVTxy
              0610
              0610           ;
              0610           ;       56FD CVTGH - Convert G_floating to H_floating
              0610           ;
              0610   INST_CVTGH:
              0610           SET_OP_TYPES    G,H
        16 11 0616           BRB     INST_CVTxy
              0618
              0618           ;
              0618           ;       F6FD CVTHF - Convert H_floating to F_floating
              0618           ;
```

```
                 0618  INST_CVTHF:
                 0618           SET_OP_TYPES    H,F
        0E   11  061E           BRB     INST_CVTxy
                 0620
                 0620           ;
                 0620           ;       F7FD CVTHD - Convert H_floating to D_floating
                 0620           ;
                 0620  INST_CVTHD:
                 0620           SET_OP_TYPES    H,D
        06   11  0626           BRB     INST_CVTxy
                 0628
                 0628           ;
                 0628           ;       76FD CVTHG - COnvert H_floating to G_floating
                 0628           ;
                 0628  INST_CVTHG:
                 0628           SET_OP_TYPES    H,G
                 062E           ;BRB    INST_CVTxy
                 062E
                 062E  INST_CVTxy:
      0892   30  062E           BSBW    READ_ACCESS             ; first operand is read only
      0BF0   30  0631           BSBW    UNPACK_FLOAT1           ; unpack and save the value
     8B AD   D6  0634           INCL    OP_INDEX(FP)            ; move to second operand
      0892   30  0637           BSBW    WRITE_ACCESS            ; second operand is write only
E7 AD   5B   D0  063A           MOVL    R11,ADDRESS1(FP)        ; save the destination location
      0CD3   30  063E           BSBW    PACK_FLOAT1             ; pack the value
5B   E7 AD   D0  0641           MOVL    ADDRESS1(FP),R11        ; R11 = destination location
      0E70   30  0645           BSBW    STORE_OPERAND           ; store the result
      14F4   30  0648           BSBW    SET_CONDITION1          ; set the condition codes in the PSL
      FB76   31  064B           BRW     NORMAL_EXIT             ; done
                 064E
                 064E           ;
                 064E           ;       4B CVTRFL - Convert Rounded F_floating to Long
                 064E           ;
                 064E  INST_CVTRFL:
                 064E           SET_OP_TYPES    F,L
        16   11  0654           BRB     INST_CVTRxL
                 0656
                 0656           ;
                 0656           ;       6B CVTRDL - Convert Rounded D_floating to Long
                 0656           ;
                 0656  INST_CVTRDL:
                 0656           SET_OP_TYPES    D,L
        0E   11  065C           BRB     INST_CVTRxL
                 065E
                 065E           ;
                 065E           ;       4BFD CVTRGL - Convert Rounded G_floating to Long
                 065E           ;
                 065E  INST_CVTRGL:                             ; entrance
                 065E           SET_OP_TYPES    G,L
        06   11  0664           BRB     INST_CVTRxL
                 0666
                 0666           ;
                 0666           ;       6BFD CVTRHL - Convert Rounded H_floating to Long
                 0666           ;
                 0666  INST_CVTRHL:                             ; entrance
                 0666           SET_OP_TYPES    H,L
                 066C           ;BRB    INST_CVTRxL
```

```
                    066C
                    066C  INST_CVTRxL:
         0854  30   066C            BSBW    READ_ACCESS             ; first operand is read only
         0BB2  30   066F            BSBW    UNPACK_FLOAT1           ; unpack the value
        8B AD  D6   0672            INCL    OP_INDEX(FP)            ; move to second operand
         0854  30   0675            BSBW    WRITE_ACCESS            ; second operand is write only
  E7 AD    5B  D0   0678            MOVL    R11,ADDRESS1(FP)        ; save the destination location
  54 AD    0F  CA   067C            BICL2   #PSLM_NZVC,PSL(FP)      ; clear the condition codes
         0EF5  30   0680            BSBW    ROUND_REAL              ; round the value
  E7 BD    50  D0   0683            MOVL    R0,@ADDRESS1(FP)        ; output the value
         1496  30   0687            BSBW    SET_CONDITION           ; set the condition codes
         FB37  31   068A            BRW     NORMAL_EXIT             ; done
                    068D
                    068D            ;
                    068D            ;       46 DIVF2 - Divide F_floating (Two Operands)
                    068D            ;
                    068D  INST_DIVF2:
                    068D            SET_OP_TYPES    F
           10  11   0691            BRB     INST_DIVx2
                    0693
                    0693            ;
                    0693            ;       66 DIVD2 - Divide D_floating (Two Operands)
                    0693            ;
                    0693  INST_DIVD2:
                    0693            SET_OP_TYPES    D
           0A  11   0697            BRB     INST_DIVx2
                    0699
                    0699            ;
                    0699            ;       46FD DIVG2 - Divide G_floating (Two Operands)
                    0699            ;
                    0699  INST_DIVG2:
                    0699            SET_OP_TYPES    G
           04  11   069D            BRB     INST_DIVx2
                    069F
                    069F            ;
                    069F            ;       66FD DIVH2 - Divide H_floating (Two Operands)
                    069F            ;
                    069F  INST_DIVH2:
                    069F            SET_OP_TYPES    H
                    06A3            ;BRB    INST_DIVx2
                    06A3
                    06A3  INST_DIVx2:                               ; entrance
         081D  30   06A3            BSBW    READ_ACCESS             ; first operand is read only
         0B87  30   06A6            BSBW    UNPACK_FLOAT3           ; unpack and save the value
         0829  30   06A9            BSBW    MODIFY_ACCESS           ; second operand is modified
  E7 AD    5B  D0   06AC            MOVL    R11,ADDRESS1(FP)        ; save the destination location
         0B77  30   06B0            BSBW    UNPACK_FLOAT2           ; unpack and save the value
         119C  30   06B3            BSBW    DIVIDE_FLOAT            ; compute the quotient
         0C5B  30   06B6            BSBW    PACK_FLOAT1             ; pack the result
  5B    E7 AD  D0   06B9            MOVL    ADDRESS1(FP),R11        ; R11 = destination location
         0DF8  30   06BD            BSBW    STORE_OPERAND           ; store the result
         147C  30   06C0            BSBW    SET_CONDITION1          ; set the condition codes in the PSL
         FAFE  31   06C3            BRW     NORMAL_EXIT             ; done
                    06C6
                    06C6            ;
                    06C6            ;       47 DIVF3 - Divide F_floating (Three Operands)
                    06C6            ;
```

```
                      06C6  INST_DIVF3:
                      06C6          SET_OP_TYPES    F
           10   11    06CA          BRB     INST_DIVx3
                      06CC
                      06CC          ;
                      06CC          ;       67 DIVD3 - Divide D_floating (Three Operands)
                      06CC          ;
                      06CC  INST_DIVD3:
                      06CC          SET_OP_TYPES    D
           0A   11    06D0          BRB     INST_DIVx3
                      06D2
                      06D2          ;
                      06D2          ;       47FD DIVG3 - Divide G_floating (Three Operands)
                      06D2          ;
                      06D2  INST_DIVG3:
                      06D2          SET_OP_TYPES    G
           04   11    06D6          BRB     INST_DIVx3
                      06D8
                      06D8          ;
                      06D8          ;       67FD DIVH3 - Divide HFLOAT (Three Operands)
                      06D8          ;
                      06D8  INST_DIVH3:                             ; entrance
                      06D8          SET_OP_TYPES    H
                      06DC          ;BRB    INST_DIVx3
                      06DC
                      06DC  INST_DIVx3:
         07E4   30    06DC          BSBW    READ_ACCESS             ; first operand is read only
         0B4E   30    06DF          BSBW    UNPACK_FLOAT3           ; unpack and save the value
         07DE   30    06E2          BSBW    READ_ACCESS             ; second operand is read only
         0B42   30    06E5          BSBW    UNPACK_FLOAT2           ; unpack and save the value
         07E1   30    06E8          BSBW    WRITE_ACCESS            ; third operand is write only
    E7 AD  5B   D0    06EB          MOVL    R11,ADDRESS1(FP)        ; save the destination location
         1160   30    06EF          BSBW    DIVIDE_FLOAT            ; compute the quotient
         0C1F   30    06F2          BSBW    PACK_FLOAT1             ; pack the result
    5B  E7 AD   D0    06F5          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
         0DBC   30    06F9          BSBW    STORE_OPERAND           ; store the result
         1440   30    06FC          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
         FAC2   31    06FF          BRW     NORMAL_EXIT             ; done
                      0702
                      0702          ;
                      0702          ;       54 EMODF - Extended Modulus F_floating
                      0702          ;
                      0702  INST_EMODF:
                      0702          SET_OP_TYPES    F,B,F,L,F
         07B2   30    070E          BSBW    READ_ACCESS             ; first operand is read only
         0B16   30    0711          BSBW    UNPACK_FLOAT2           ; unpack and save the operand
        8B AD   D6    0714          INCL    OP_INDEX(FP)            ; move to second operand
         07A9   30    0717          BSBW    READ_ACCESS             ; second operand is read only
    C3 AD  50   90    071A          MOVB    R0,FRACTION2+12(FP)     ; insert bits 0-7 of the fraction
        8B AD   D6    071E          INCL    OP_INDEX(FP)            ; move to third operand
         079F   30    0721          BSBW    READ_ACCESS             ; third operand is read only
         0B09   30    0724          BSBW    UNPACK_FLOAT3           ; unpack and save the value
    FC AD  01   90    0727          MOVB    #FLAG0M,FLAGS(FP)       ; inhibit checking for local store
        8B AD   D6    072B          INCL    OP_INDEX(FP)            ; move to fourth operand
         079B   30    072E          BSBW    WRITE_ACCESS            ; fourth operand is write only
 05 FC AD  01   E5    0731          BBCC    #FLAGT,FLAGS(FP),1$     ; not register mode - skip
 00 FC AD  07   E3    0736          BBCS    #FLAG7,FLAGS(FP),1$     ; make a note
```

```
         E7 AD   5B  D0 073B 1$:     MOVL    R11,ADDRESS1(FP)          ; save the first destination address
            8B AD    D6 073F         INCL    OP_INDEX(FP)              ; move to fifth operand
             0787    30 0742         BSBW    WRITE_ACCESS              ; fifth operand is write only
         E3 AD   5B  D0 0745         MOVL    R11,ADDRESS2(FP)          ; save the second destination address
             0FF3    30 0749         BSBW    MULTIPLY_FLOAT            ; compute the product
 DB AD   50   00  00 F0 074C         INSV    #0,#0,R0,FRACTION1+12(FP); truncate the product if necessary
         54 AD   0F  CA 0752         BICL2   #PSLM_NZVC,PSL(FP)        ; clear the condition codes
             0DCB    30 0756         BSBW    FIX_REAL                  ; fix the product
         DF AD   50  D0 0759         MOVL    R0,ADDRESS3(FP)           ; save the integer part
             0E87    30 075D         BSBW    FRACTION_REAL             ; form the fractional part
             0BB1    30 0760         BSBW    PACK_FLOAT1               ; pack the fraction value
               50    DD 0763         PUSHL   R0                        ; push the fraction
      05 FC AD   07  E1 0765         BBC     #FLAG7,FLAGS(FP),2$       ; fourth operand not register - skip
      E7 BD   DF AD  D0 076A         MOVL    ADDRESS3(FP),@ADDRESS1(FP) ; output the integer part
      04 FC AD   01  E1 076F 2$:     BBC     #FLAG1,FLAGS(FP),3$       ; fifth operand not register - skip
         E3 BD   8E  D0 0774         MOVL    (SP)+,@ADDRESS2(FP)       ; output the fraction
            50   04  D0 0778 3$:     MOVL    #4,R0                     ; allow room for dummy store
             FB5C    30 077B         BSBW    TEST_FRAME                ; move the frame if necessary
      0E FC AD   07  E0 077E         BBS     #FLAG7,FLAGS(FP),4$       ; fourth operand is register - bypass
            5A   04  D0 0783         MOVL    #4,R10                    ; R10 = size of integer part
         5B   E7 AD  D0 0786         MOVL    ADDRESS1(FP),R11          ; R11 = location of fourth operand
             0A80    30 078A         BSBW    LOCAL_TEST                ; check for a store into local storage
         6B   DF AD  D0 078D         MOVL    ADDRESS3(FP),(R11)        ; output the integer part
      0D FC AD   01  E0 0791 4$:     BBS     #FLAG1,FLAGS(FP),5$       ; fifth operand is register - bypass
            5A   04  D0 0796         MOVL    #4,R10                    ; R10 = size of the fraction
         5B   E3 AD  D0 0799         MOVL    ADDRESS2(FP),R11          ; R11 = location of second operand
             0A6D    30 079D         BSBW    LOCAL_TEST                ; check for a store into local storage
            6B   8E  D0 07A0         MOVL    (SP)+,(R11)               ; output the fraction
         51   C7 AD  9E 07A3 5$:     MOVAB   OPERAND1(FP),R1           ; R1 = location of the fraction
             13A2    30 07A7         BSBW    TEST_REAL                 ; test the fraction
             1373    30 07AA         BSBW    SET_CONDITION             ; set the condition codes
             FA14    31 07AD         BRW     NORMAL_EXIT               ; done
                        07B0
                        07B0         ;
                        07B0         ;       74 EMODD - Extended Modulus D_floating
                        07B0         ;
                        07B0 INST_EMODD:
                        07B0         SET_OP_TYPES    D,B,D,L,D
             0704    30 07BC         BSBW    READ_ACCESS               ; first operand is read only
             0A68    30 07BF         BSBW    UNPACK_FLOAT2             ; unpack and save the operand
            8B AD    D6 07C2         INCL    OP_INDEX(FP)              ; move to second operand
             06FB    30 07C5         BSBW    READ_ACCESS               ; second operand is read only
         BF AD   50  90 07C8         MOVB    R0,FRACTION2+8(FP)        ; insert bits 0-7 of the fraction
               22    11 07CC         BRB     COMMON_EMODdg             ; join common D/G code
                        07CE
                        07CE         ;
                        07CE         ;       54FD EMODG - Extended Modulus G_floating
                        07CE         ;
                        07CE INST_EMODG:                               ; entrance
                        07CE         SET_OP_TYPES    G,W,G,L,G
             06E6    30 07DA         BSBW    READ_ACCESS               ; first operand is read only
             0A4A    30 07DD         BSBW    UNPACK_FLOAT2             ; unpack and save the operand
            8B AD    D6 07E0         INCL    OP_INDEX(FP)              ; move to second operand
             06DD    30 07E3         BSBW    READ_ACCESS               ; second operand is read only
      50      50 1B  9C 07E6         ROTL    #27,R0,R0                 ; position the high-order 11 bits
 BF AD   0B   00  50 F0 07EA         INSV    R0,#0,#11,FRACTION2+8(FP) ; insert bits 0-10 of the fraction
                        07F0 COMMON_EMODdg:
```

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

```
              0615  30  08B4               BSBW    WRITE_ACCESS             ; fourth operand is write only
    05 FC AD    01  E5  08B7               BBCC    #FLAG1,FLAGS(FP),1$      ; not register mode - skip
    00 FC AD    07  E3  08BC               BBCS    #FLAG7,FLAGS(FP),1$      ; make a note
       E7 AD    5B  D0  08C1  1$:          MOVL    R11,ADDRESS1(FP)         ; save the first destination address
          8B AD     D6  08C5               INCL    OP_INDEX(FP)             ; move to fifth operand
              0601  30  08C8               BSBW    WRITE_ACCESS             ; fifth operand is write only
       E3 AD    5B  D0  08CB               MOVL    R11,ADDRESS2(FP)         ; save the second destination address
              0E6D  30  08CF               BSBW    MULTIPLY_FLOAT           ; compute the product
CF AD   50  00  00  F0  08D2               INSV    #0,#0,R0,FRACTION1(FP)   ; truncate the product if necessary
       54 AD    0F  CA  08D8               BICL2   #PSLM_NZVC,PSL(FP)       ; clear the condition codes
              0C45  30  08DC               BSBW    FIX_REAL                 ; fix the product
       DF AD    50  D0  08DF               MOVL    R0,ADDRESS3(FP)          ; save the integer part
              0D01  30  08E3               BSBW    FRACTION_REAL            ; form the fractional part
              0A2B  30  08E6               BSBW    PACK_FLOAT1              ; pack the fraction value
                0F  BB  08E9               PUSHR   #^M<R0,R1,R2,R3>         ; push the fraction
    05 FC AD    07  E1  08EB               BBC     #FLAG7,FLAGS(FP),2$      ; fourth operand not register - skip
    E7 BD    DF AD  D0  08F0               MOVL    ADDRESS3(FP),@ADDRESS1(FP) ; output the integer part
    0A FC AD    01  E1  08F5  2$:          BBC     #FLAG1,FLAGS(FP),3$      ; fifth operand not register - bypass
       5B    E3 AD  D0  08FA               MOVL    ADDRESS2(FP),R11         ; R11 = location of the register
          8B    8E  7D  08FE               MOVQ    (SP)+,(R11)+             ; output first part of the fraction
          6B    8E  7D  0901               MOVQ    (SP)+,(R11)              ; output second part of the fraction
          50    10  D0  0904  3$:          MOVL    #16,R0                   ; allow room for dummy store
              F9D0  30  0907               BSBW    TEST_FRAME               ; move the frame if necessary
    0E FC AD    07  E0  090A               BBS     #FLAG7,FLAGS(FP),4$      ; fourth operand is register - bypass
          5A    04  D0  090F               MOVL    #4,R10                   ; R10 = size of integer part
       5B    E7 AD  D0  0912               MOVL    ADDRESS1(FP),R11         ; R11 = location of fourth operand
              08F4  30  0916               BSBW    LOCAL_TEST               ; check for a store into local storage
       6B    DF AD  D0  0919               MOVL    ADDRESS3(FP),(R11)       ; output the integer part
    10 FC AD    01  E0  091D  4$:          BBS     #FLAG1,FLAGS(FP),5$      ; fifth operand is register - bypass
          5A    10  D0  0922               MOVL    #16,R10                  ; R10 = size of the fraction
       5B    E3 AD  D0  0925               MOVL    ADDRESS2(FP),R11         ; R11 = location of second operand
              08E1  30  0929               BSBW    LOCAL_TEST               ; check for a store into local storage
          8B    8E  7D  092C               MOVQ    (SP)+,(R11)+             ; output first part of the fraction
          6B    8E  7D  092F               MOVQ    (SP)+,(R11)              ; output second part of the fraction
       51    C7 AD  9E  0932  5$:          MOVAB   OPERAND1(FP),R1          ; R1 = location of the fraction
              1213  30  0936               BSBW    TEST_REAL                ; test the fraction
              11E4  30  0939               BSBW    SET_CONDITION            ; set the condition codes
              F885  31  093C               BRW     NORMAL_EXIT              ; done
                        093F
                        093F               ;
                        093F               ;       52 MNEGF - Move Negated F_floating
                        093F               ;
                        093F  INST_MNEGF:
                        093F               SET_OP_TYPES    F
                10  11  0943               BRB     INST_MNEGx
                        0945
                        0945               ;
                        0945               ;       72 MNEGD - Move Negated D_floating
                        0945               ;
                        0945  INST_MNEGD:
                        0945               SET_OP_TYPES    D
                0A  11  0949               BRB     INST_MNEGx
                        094B
                        094B               ;
                        094B               ;       52FD MNEGG - Move Negated G_floating
                        094B               ;
                        094B  INST_MNEGG:
```

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

```
                        0975          ;
                        0975          ;       7EFD MOVAO - Move Address of Octaword
                        0975          ;
                        0975          ;       7EFD MOVAH - Move Address of H_floating
                        0975          ;
                        0975  INST_MOVAO:                             ; entrance
                        0975          SET_OP_TYPES    O,L
           0560   30    097B          BSBW    ADDRESS_ACCESS          ; first operand is address only
       E7 AD 5B   D0    097E          MOVL    R11,ADDRESS1(FP)        ; save the operand address
          8B AD   D6    0982          INCL    OP_INDEX(FP)            ; move to second operand
           0544   30    0985          BSBW    WRITE_ACCESS            ; second operand is write only
       6B E7 AD   D0    0988          MOVL    ADDRESS1(FP),(R11)      ; output the operand address
           1191   30    098C          BSBW    SET_CONDITION           ; capture the condition code
       54 AD 02   CA    098F          BICL2   #PSL$M_V,PSL(FP)        ; clear the V bit in the PSL
           F82E   31    0993          BRW     NORMAL_EXIT             ; done
                        0996
                        0996          ;
                        0996          ;       50 MOVF - Move F_floating
                        0996          ;
                        0996  INST_MOVF:
                        0996          SET_OP_TYPES    F
             10   11    099A          BRB     INST_MOVx
                        099C
                        099C          ;
                        099C          ;       70 MOVD - Move D_floating
                        099C          ;
                        099C  INST_MOVD:
                        099C          SET_OP_TYPES    D
             0A   11    09A0          BRB     INST_MOVx
                        09A2
                        09A2          ;
                        09A2          ;       50FD MOVG - Move G_floating
                        09A2          ;
                        09A2  INST_MOVG:
                        09A2          SET_OP_TYPES    G
             04   11    09A6          BRB     INST_MOVx
                        09A8
                        09A8          ;
                        09A8          ;       70FD MOVH - Move HFLOAT
                        09A8          ;
                        09A8  INST_MOVH:
                        09A8          SET_OP_TYPES    H
                        09AC          ;BRB    INST_MOVx
                        09AC
                        09AC  INST_MOVx:
           0514   30    09AC          BSBW    READ_ACCESS             ; first operand is read only
           0872   30    09AF          BSBW    UNPACK_FLOAT1           ; unpack and save the value
           0517   30    09B2          BSBW    WRITE_ACCESS            ; second operand is write only
       E7 AD 5B   D0    09B5          MOVL    R11,ADDRESS1(FP)        ; save the destination location
           0958   30    09B9          BSBW    PACK_FLOAT1             ; pack the value
       5B E7 AD   D0    09BC          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
           0AF5   30    09C0          BSBW    STORE_OPERAND           ; store the result
       51 C7 AD   9E    09C3          MOVAB   OPERAND1(FP),R1         ; R1 = location of the value
           1182   30    09C7          BSBW    TEST_REAL               ; test the value
           1153   30    09CA          BSBW    SET_CONDITION           ; set the condition codes
       54 AD 02   CA    09CD          BICL2   #PSL$M_V,PSL(FP)        ; clear the V bit in the PSL
           F7F0   31    09D1          BRW     NORMAL_EXIT             ; done
```

```
                    09D4
                    09D4             ;
                    09D4             ;       7DFD MOVO - Move Octaword
                    09D4             ;
                    09D4    INST_MOVO:
                    09D4                     SET_OP_TYPES    O
         04E8   30  09D8                     BSBW    READ_ACCESS             ; first operand is read only
   C7 AD   50   7D  09DB                     MOVQ    R0,OPERAND1(FP)         ; save the first part of the value
   CF AD   52   7D  09DF                     MOVQ    R2,OPERAND1+8(FP)       ; save the second part of the value
         04E6   30  09E3                     BSBW    WRITE_ACCESS            ; second operand is write only
   E7 AD   5B   D0  09E6                     MOVL    R11,ADDRESS1(FP)        ; save destination location
         1168   30  09EA                     BSBW    TEST_OCTA               ; test the value
         1130   30  09ED                     BSBW    SET_CONDITION           ; set the condition codes
   54 AD   02   CA  09F0                     BICL2   #PSL$M_V,PSL(FP)        ; clear the V bit in the PSL
   5B   E7 AD   D0  09F4                     MOVL    ADDRESS1(FP),R11        ; R11 = destination location
   8B   C7 AD   7D  09F8                     MOVQ    OPERAND1(FP),(R11)+     ; output the first part of the result
   6B   CF AD   7D  09FC                     MOVQ    OPERAND1+8(FP),(R11)    ; output the second part of the result
         F7C1   31  0A00                     BRW     NORMAL_EXIT             ; done
                    0A03
                    0A03             ;
                    0A03             ;       44 MULF2 - Multiply F_floating (Two Operands)
                    0A03             ;
                    0A03    INST_MULF2:
                    0A03                     SET_OP_TYPES    F
           10   11  0A07                     BRB     INST_MULx2
                    0A09
                    0A09             ;
                    0A09             ;       64 MULD2 - Multiply D_floating (Two Operands)
                    0A09             ;
                    0A09    INST_MULD2:
                    0A09                     SET_OP_TYPES    D
           0A   11  0A0D                     BRB     INST_MULx2
                    0A0F
                    0A0F             ;
                    0A0F             ;       44FD MULG2 - Multiply G_floating (Two Operands)
                    0A0F             ;
                    0A0F    INST_MULG2:
                    0A0F                     SET_OP_TYPES    G
           04   11  0A13                     BRB     INST_MULx2
                    0A15
                    0A15             ;
                    0A15             ;       64FD MULH2 - Multiply HFLOAT (Two Operands)
                    0A15             ;
                    0A15    INST_MULH2:                                      ; entrance
                    0A15                     SET_OP_TYPES    H
                    0A19                     ;BRB    INST_MULx2
                    0A19
                    0A19    INST_MULx2:
         04A7   30  0A19                     BSBW    READ_ACCESS             ; first operand is read only
         080B   30  0A1C                     BSBW    UNPACK_FLOAT2           ; unpack and save the first operand
         04B3   30  0A1F                     BSBW    MODIFY_ACCESS           ; second operand is modified
   E7 AD   5B   D0  0A22                     MOVL    R11,ADDRESS1(FP)        ; save the destination location
         0807   30  0A26                     BSBW    UNPACK_FLOAT3           ; unpack and save the second operand
         0D13   30  0A29                     BSBW    MULTIPLY_FLOAT          ; compute the product
         08E5   30  0A2C                     BSBW    PACK_FLOAT1             ; pack the value
   5B   E7 AD   D0  0A2F                     MOVL    ADDRESS1(FP),R11        ; R11 = destination location
         0A82   30  0A33                     BSBW    STORE_OPERAND           ; store the result
```

```
              1106  30  0A36          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
              F788  31  0A39          BRW     NORMAL_EXIT             ; done
                        0A3C
                        0A3C          ;
                        0A3C          ;       45 MULF3 - Multiply F_floating (Three Operands)
                        0A3C          ;
                        0A3C  INST_MULF3:
                        0A3C          SET_OP_TYPES    F
                10  11  0A40          BRB     INST_MULx3
                        0A42
                        0A42          ;
                        0A42          ;       65 MULD3 - Multiply D_floating (Three Operands)
                        0A42          ;
                        0A42  INST_MULD3:
                        0A42          SET_OP_TYPES    D
                0A  11  0A46          BRB     INST_MULx3
                        0A48
                        0A48          ;
                        0A48          ;       45FD MULG3 - Multiply G_floating (Three Operands)
                        0A48          ;
                        0A48  INST_MULG3:
                        0A48          SET_OP_TYPES    G
                04  11  0A4C          BRB     INST_MULx3
                        0A4E
                        0A4E          ;
                        0A4E          ;       65FD MULH3 - Multiply HFLOAT (Three Operands)
                        0A4E          ;
                        0A4E  INST_MULH3:
                        0A4E          SET_OP_TYPES    H
                        0A52          ;BRB    INST_MULx3
                        0A52
                        0A52  INST_MULx3:
              046E  30  0A52          BSBW    READ_ACCESS             ; first operand is read only
              07D2  30  0A55          BSBW    UNPACK_FLOAT2           ; unpack and save the value
              0468  30  0A58          BSBW    READ_ACCESS             ; second operand is read only
              07D2  30  0A5B          BSBW    UNPACK_FLOAT3           ; unpack and save the value
              046B  30  0A5E          BSBW    WRITE_ACCESS            ; third operand is write only
         E7 AD  5B  D0  0A61          MOVL    R11,ADDRESS1(FP)        ; save the destination location
              0CD7  30  0A65          BSBW    MULTIPLY_FLOAT          ; compute the product
              08A9  30  0A68          BSBW    PACK_FLOAT1             ; pack the value
         5B  E7 AD  D0  0A6B          MOVL    ADDRESS1(FP),R11        ; R11 = destination location
              0A46  30  0A6F          BSBW    STORE_OPERAND           ; store the result
              10CA  30  0A72          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
              F74C  31  0A75          BRW     NORMAL_EXIT             ; done
                        0A78
                        0A78          ;
                        0A78          ;       55 POLYF - Evaluate Polynomial F_floating
                        0A78          ;
                        0A78  INST_POLYF:                             ; entrance
                        0A78          SET_OP_TYPES    F,W,B,F         ; phantom fourth operand
     2F 54 AD   1B  E1  0A80          BBC     #PSL_FPD,PSL(FP),1$     ; not instruction resumption - bypass
 50  1C AD  18  08  EF  0A85          EXTZV   #8,#24,REG_R2(FP),R0    ; R0 = saved instruction length
         50 AD  50  C0  0A8B          ADDL2   R0,REG_PC(FP)           ; position PC to end of instruction
         FA AD  50  80  0A8F          ADDB2   R0,REGMOD_PC(FP)        ; increment the PC modification count
         50  14 AD  D0  0A93          MOVL    REG_R0(FP),R0           ; R0 = result so far
              0790  30  0A97          BSBW    UNPACK_FLOAT2           ; unpack and save the value
         51  AF AD  9E  0A9A          MOVAB   OPERAND2(FP),R1         ; R1 = location of OPERAND2
```

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

```
                06C8  30  0B5F          BSBW    UNPACK_FLOAT2           ; unpack the result for next iteration
                0FDA  30  0B62          BSBW    SET_CONDITION1          ; set the condition codes in the PSL
          20 AD   04  C0  0B65          ADDL2   #4,REG_R3(FP)           ; update the table pointer
             1C AD    97  0B69          DECB    REG_R2(FP)              ; decrement the iteration counter
                  BF  11  0B6C          BRB     4$                      ; start the next iteration
             18 AD    D4  0B6E  6$:     CLRL    REG_R1(FP)              ; clear the user's R1
             1C AD    D4  0B71          CLRL    REG_R2(FP)              ; clear the user's R2
       00 54 AD   1B  E5  0B74          BBCC    #PSL_FPD,PSL(FP),7$     ; indicate instruction complete
                F648  31  0B79  7$:     BRW     NORMAL_EXIT             ; done
                          0B7C
                          0B7C          ;
                          0B7C          ;       75 POLYD - Evaluate Polynomial D_floating
                          0B7C          ;
                          0B7C  INST_POLYD:                             ; entrance
                          0B7C          SET_OP_TYPES    D,W,B,D         ; phantom fourth operand
                  08  11  0B84          BRB     INST_POLYdg
                          0B86
                          0B86          ;
                          0B86          ;       55FD POLYG - Evaluate Polynomial G_floating
                          0B86          ;
                          0B86  INST_POLYG:                             ; entrance
                          0B86          SET_OP_TYPES    G,W,B,G         ; phantom fourth operand
                          0B8E          ;BRB    INST_POLYdg
                          0B8E
                          0B8E  INST_POLYdg:
       32 54 AD   1B  E1  0B8E          BBC     #PSL_FPD,PSL(FP),1$     ; not instruction resumption - bypass
  50   1C AD   18     08  EF  0B93      EXTZV   #8,#24,REG_R2(FP),R0    ; R0 = saved instruction length
          50 AD   50  C0  0B99          ADDL2   R0,REG_PC(FP)           ; position PC to end of instruction
          FA AD   50  80  0B9D          ADDB2   R0,REGMOD_PC(FP)        ; increment the PC modification count
          50   14 AD  7D  0BA1          MOVQ    REG_R0(FP),R0           ; R0,R1 = result so far
                0682  30  0BA5          BSBW    UNPACK_FLOAT2           ; unpack and save the value
          51   AF AD  9E  0BA8          MOVAB   OPERAND2(FP),R1         ; R1 = location of OPERAND2
                0F9D  30  0BAC          BSBW    TEST_REAL               ; test the value
                0F6E  30  0BAF          BSBW    SET_CONDITION           ; set the condition codes
          50   24 AD  7D  0BB2          MOVQ    REG_R4(FP),R0           ; R0,R1 = argument value
                0677  30  0BB6          BSBW    UNPACK_FLOAT3           ; unpack and save the value
          1C AD   1F  91  0BB9          CMPB    #31,REG_R2(FP)          ; is the iteration count too large ?
                  03  1E  0BBD          BGEQU   14$                     ; no - resume the instruction
                10C4  31  0BBF          BRW     OPERAND_FAULT           ; process the reserved operand
                0081  31  0BC2  14$:    BRW     4$
                02FB  30  0BC5  1$:     BSBW    READ_ACCESS             ; first operand is read only
                0665  30  0BC8          BSBW    UNPACK_FLOAT3           ; unpack and save the value
             8B AD    D6  0BCB          INCL    OP_INDEX(FP)            ; look at second operand
                02F2  30  0BCE          BSBW    READ_ACCESS             ; second operand is read only
             50   1F  D1  0BD1          CMPL    #31,R0                  ; is the value reserved ?
                  03  1E  0BD4          BGEQU   2$                      ; no - skip
                10AD  31  0BD6          BRW     OPERAND_FAULT           ; process the reserved operand
          E7 AD   50  D0  0BD9  2$:     MOVL    R0,ADDRESS1(FP)         ; save the operand value
             8B AD    D6  0BDD          INCL    OP_INDEX(FP)            ; look at third operand
                02FB  30  0BE0          BSBW    ADDRESS_ACCESS          ; third operand is address only
  6B    08     FD AD  0C  0BE3          PROBER  MODE(FP),#8,(R11)       ; can we read the first quadword ?
                  06  12  0BE8          BNEQ    3$                      ; yes - skip
             5A   08  D0  0BEA          MOVL    #8,R10                  ; R10 = size of probe
                0FBB  30  0BED          BSBW    READ_FAULT              ; process the access violation
             50   8B  7D  0BF0  3$:     MOVQ    (R11)+,R0               ; R0,R1 = the first table entry
          E3 AD   5B  D0  0BF3          MOVL    R11,ADDRESS2(FP)        ; save the following address
             8B AD    D6  0BF7          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          BSBW    UNPACK_FLOAT2               ; unpack and save the value
        51   AF AD   9E  0CBB          MOVAB   OPERAND2(FP),R1             ; R1 = location of OPERAND2
              0E8A   30  0CBF          BSBW    TEST_REAL                   ; test the value
              0E5B   30  0CC2          BSBW    SET_CONDITION               ; set the condition codes
        5B   4C AD   D0  0CC5          MOVL    REG_SP(FP),R11              ; R11 = user's stack pointer
   6B   10   FD AD   0C  0CC9          PROBER  MODE(FP),#16,(R11)          ; can we read a stacked octaword ?
                06   12  0CCE          BNEQ    1$                          ; yes - skip
        5A      10   D0  0CD0          MOVL    #16,R10                     ; R10 = size of probe
              0ED5   30  0CD3          BSBW    READ_FAULT                  ; process the access violation
        50      8B   7D  0CD6  1$:     MOVQ    (R11)+,R0                   ; R0,R1 = first part of argument
        52      6B   7D  0CD9          MOVQ    (R11),R2                    ; R2,R3 = second part of argument
              0551   30  0CDC          BSBW    UNPACK_FLOAT3               ; unpack and save the value
        54      1F   91  0CDF          CMPB    #31,R4                      ; is the iteration count too large ?
                03   1F  0CE2          BLSSU   2$                          ; no - skip
              0F9F   31  0CE4          BRW     OPERAND_FAULT               ; processed the reserved operand
              00A4   31  0CE7  2$:     BRW     7$                          ; resume the instruction
              01D6   30  0CEA  3$:     BSBW    READ_ACCESS                 ; first operand is read only
              0540   30  0CED          BSBW    UNPACK_FLOAT3               ; unpack and save the value
             8B AD   D6  0CF0          INCL    OP_INDEX(FP)                ; look at second operand
              01CD   30  0CF3          BSBW    READ_ACCESS                 ; second operand is read only
        50      1F   D1  0CF6          CMPL    #31,R0                      ; is the value reserved ?
                03   1E  0CF9          BGEQU   4$                          ; no - skip
              0F88   31  0CFB          BRW     OPERAND_FAULT               ; process the reserved operand
     E7 AD      50   D0  0CFE  4$:     MOVL    R0,ADDRESS1(FP)             ; save the operand value
             8B AD   D6  0D02          INCL    OP_INDEX(FP)                ; look at third operand
              01D6   30  0D05          BSBW    ADDRESS_ACCESS              ; third operand is address only
     E3 AD      5B   D0  0D08          MOVL    R11,ADDRESS2(FP)            ; save the table address
             8B AD   D6  0D0C          INCL    OP_INDEX(FP)                ; use dummy fourth operand for datatype
              060E   30  0D0F          BSBW    PACK_FLOAT3                 ; pack the argument value
     4C AD      10   C2  0D12          SUBL2   #16,REG_SP(FP)              ; decrement the user SP
     F9 AD      10   82  0D16          SUBB2   #16,REGMOD_SP(FP)           ; remember the decrement
        5B   4C AD   D0  0D1A          MOVL    REG_SP(FP),R11              ; R11 = user's stack pointer
   6B   10   FD AD   0D  0D1F          PROBEW  MODE(FP),#16,(R11)          ; can we stack the argument ?
                06   12  0D23          BNEQ    5$                          ; yes - skip
        5A      10   D0  0D25          MOVL    #16,R10                     ; R10 = size of probe
              0EF1   30  0D28          BSBW    WRITE_FAULT                 ; process the access violation
        8B      50   7D  0D2B  5$:     MOVQ    R0,(R11)+                   ; save first part of argument
        6B      52   7D  0D2E          MOVQ    R2,(R11)                    ; save second part of argument
        5B   E3 AD   D0  0D31          MOVL    ADDRESS2(FP),R11            ; R11 = coefficient table address
   6B   10   FD AD   0C  0D35          PROBER  MODE(FP),#16,(R11)          ; can we read the first octaword ?
                06   12  0D3A          BNEQ    6$                          ; yes - skip
        5A      10   D0  0D3C          MOVL    #16,R10                     ; R10 = size of probe
              0E69   30  0D3F          BSBW    READ_FAULT                  ; process the access violation
        50      8B   7D  0D42  6$:     MOVQ    (R11)+,R0                   ; R0,R1 = first part of coefficient
        52      8B   7D  0D45          MOVQ    (R11)+,R2                   ; R2,R3 = second part of coefficient
     E3 AD      5B   D0  0D48          MOVL    R11,ADDRESS2(FP)            ; save the following address
              04DB   30  0D4C          BSBW    UNPACK_FLOAT2               ; unpack and save the value
              05C8   30  0D4F          BSBW    PACK_FLOAT2                 ; pack the value again
     14 AD      50   7D  0D52          MOVQ    R0,REG_R0(FP)               ; save first part of value so far
     1C AD      52   7D  0D56          MOVQ    R2,REG_R2(FP)               ; save second part of value so far
        51   AF AD   9E  0D5A          MOVAB   OPERAND2(FP),R1             ; R1 = location of OPERAND2
              0DEB   30  0D5E          BSBW    TEST_REAL                   ; test the value
              0DBC   30  0D61          BSBW    SET_CONDITION               ; set the condition codes
     54 AD      03   CA  0D64          BICL2   #PSL$M_VC,PSL(FP)           ; clear the V bit and C bit in the PSL
  24 AD      E7 AD   D0  0D68          MOVL    ADDRESS1(FP),REG_R4(FP)     ; save the remaining iteration count
  28 AD      E3 AD   D0  0D6D          MOVL    ADDRESS2(FP),REG_R5(FP)     ; save the next coefficient address
     50      FA AD   98  0D72          CVTBL   REGMOD_PC(FP),R0            ; R0 = PC register modifications
```

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

```
                  0E1B         ;       42 SUBF2 - Subtract F_floating (Two Operands)
                  0E1B         ;
                  0E1B  INST_SUBF2:
                  0E1B                 SET_OP_TYPES    F
          10  11  0E1F                 BRB     INST_SUBx2
                  0E21
                  0E21         ;
                  0E21         ;       62 SUBD2 - Subtract D_floating (Two Operands)
                  0E21         ;
                  0E21  INST_SUBD2:
                  0E21                 SET_OP_TYPES    D
          0A  11  0E25                 BRB     INST_SUBx2
                  0E27
                  0E27         ;
                  0E27         ;       42FD SUBG2 - Subtract G_floating (Two Operands)
                  0E27         ;
                  0E27  INST_SUBG2:
                  0E27                 SET_OP_TYPES    G
          04  11  0E2B                 BRB     INST_SUBx2
                  0E2D
                  0E2D         ;
                  0E2D         ;       62FD SUBH2 - Subtract HFLOAT (Two Operands)
                  0E2D         ;
                  0E2D  INST_SUBH2:
                  0E2D                 SET_OP_TYPES    H
                  0E31                 ;BRB    INST_SUBx2
                  0E31
                  0E31  INST_SUBx2:
        008F  30  0E31                 BSBW    READ_ACCESS             ; first operand is read only
        03ED  30  0E34                 BSBW    UNPACK_FLOAT1           ; unpack and save the value
        06A9  30  0E37                 BSBW    NEGATE_REAL             ; negate the value
        0098  30  0E3A                 BSBW    MODIFY_ACCESS           ; second operand is modified
  E7 AD   5B  D0  0E3D                 MOVL    R11,ADDRESS1(FP)        ; save the destination location
        03E6  30  0E41                 BSBW    UNPACK_FLOAT2           ; unpack and save the value
        07D7  30  0E44                 BSBW    ADD_REAL                ; compute the sum
        04CA  30  0E47                 BSBW    PACK_FLOAT1             ; pack the value
  5B   E7 AD  D0  0E4A                 MOVL    ADDRESS1(FP),R11        ; R11 = destination location
        0667  30  0E4E                 BSBW    STORE_OPERAND           ; store result
        0CEB  30  0E51                 BSBW    SET_CONDITION1          ; set the condition codes in the PSL
        F36D  31  0E54                 BRW     NORMAL_EXIT             ; done
                  0E57
                  0E57         ;
                  0E57         ;       43 SUBF3 - Subtract F_floating (Three Operands)
                  0E57         ;
                  0E57  INST_SUBF3:
                  0E57                 SET_OP_TYPES    F
          10  11  0E5B                 BRB     INST_SUBx3
                  0E5D
                  0E5D         ;
                  0E5D         ;       63 SUBD3 - Subtract D_floating (Three Operands)
                  0E5D         ;
                  0E5D  INST_SUBD3:
                  0E5D                 SET_OP_TYPES    D
          0A  11  0E61                 BRB     INST_SUBx3
                  0E63
                  0E63         ;
                  0E63         ;       43FD SUBG3 - Subtract G_floating (Three Operands)
```

```
                    0E63         ;
                    0E63   INST_SUBG3:
                    0E63           SET_OP_TYPES    G
          04    11  0E67           BRB     INST_SUBx3
                    0E69
                    0E69           ;
                    0E69           ;       63FD SUBH3 - Subtract HFLOAT (Three Operands)
                    0E69           ;
                    0E69   INST_SUBH3:
                    0E69           SET_OP_TYPES    H
                    0E6D           ;BRB    INST_SUBx3
                    0E6D
                    0E6D   INST_SUBx3:
        0053    30  0E6D           BSBW    READ_ACCESS             ; first operand is read only
        03B1    30  0E70           BSBW    UNPACK_FLOAT1           ; unpack and save the value
        066D    30  0E73           BSBW    NEGATE_REAL             ; negate the value
        004A    30  0E76           BSBW    READ_ACCESS             ; second operand is read only
        03AE    30  0E79           BSBW    UNPACK_FLOAT2           ; unpack and save the value
        004D    30  0E7C           BSBW    WRITE_ACCESS            ; third operand is write only
  E7 AD   5B    D0  0E7F           MOVL    R11,ADDRESS1(FP)        ; save the destination location
        0798    30  0E83           BSBW    ADD_REAL                ; compute the sum
        048B    30  0E86           BSBW    PACK_FLOAT1             ; pack the value
  5B   E7 AD    D0  0E89           MOVL    ADDRESS1(FP),R11        ; R11 = destination location
        0628    30  0E8D           BSBW    STORE_OPERAND           ; store result
        0CAC    30  0E90           BSBW    SET_CONDITION1          ; set the condition codes in the PSL
        F32E    31  0E93           BRW     NORMAL_EXIT             ; done
                    0E96
                    0E96           ;
                    0E96           ;       53 TSTF - Test F_floating
                    0E96           ;
                    0E96   INST_TSTF:
                    0E96           SET_OP_TYPES    F
          10    11  0E9A           BRB     INST_TSTx
                    0E9C
                    0E9C           ;
                    0E9C           ;       73 TSTD - Test D_floating
                    0E9C           ;
                    0E9C   INST_TSTD:
                    0E9C           SET_OP_TYPES    D
          0A    11  0EA0           BRB     INST_TSTx
                    0EA2
                    0EA2           ;
                    0EA2           ;       53FD TSTG - Test G_floating
                    0EA2           ;
                    0EA2   INST_TSTG:
                    0EA2           SET_OP_TYPES    G
          04    11  0EA6           BRB     INST_TSTx
                    0EA8
                    0EA8           ;
                    0EA8           ;       73FD TSTH - Test HFLOAT
                    0EA8           ;
                    0EA8   INST_TSTH:
                    0EA8           SET_OP_TYPES    H
                    0EAC           ;BRB    INST_TSTx
                    0EAC
                    0EAC   INST_TSTx:
        0014    30  0EAC           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.
```

```
0EC3      ;
0EC3      ;          When the routines are entered they scan the next
0EC3      ;      instruction operand starting at the value of the user's PC
0EC3      ;      and check the operand for validity. If any exceptions are
0EC3      ;      detected during operand scanning they are processed immediatly
0EC3      ;      and the routines do not return. Any changes that are made to
0EC3      ;      any of the registers (including PC) are recorded in the change
0EC3      ;      bytes so faults will be handled properly.
0EC3      ;
0EC3      ;          When the routines return the following rules describe the
0EC3      ;      contents of the registers:
0EC3      ;
0EC3      ;             o  If the access type is READ or MODIFY and the data type
0EC3      ;                is BYTE, WORD, LONG, or FLOAT, then R0 is the value
0EC3      ;                of the operand. If the data type is BYTE or WORD then
0EC3      ;                the value is sign extended to a longword.
0EC3      ;
0EC3      ;             o  If the access type is READ or MODIFY and the data type
0EC3      ;                is QUAD, DFLOAT, or GFLOAT, then R0-R1 contains the
0EC3      ;                value of the operand.
0EC3      ;
0EC3      ;             o  If the access type is READ or MODIFY and the data type
0EC3      ;                is OCTA or HFLOAT then R0-R3 contains the value of the
0EC3      ;                operand.
0EC3      ;
0EC3      ;             o  If the access type is WRITE, MODIFY, ADDRESS, or
0EC3      ;                BRANCH, then R11 is the address of the operand or the
0EC3      ;                branch destination.
0EC3      ;
0EC3      ;      If the instruction operand specifies register mode, then the
0EC3      ;      address associated with the operand is the location of the
0EC3      ;      emulated register. If an instruction operand with WRITE or
0EC3      ;      MODIFY access addresses the Emulator's local storage then it
0EC3      ;      is changed to an address that won't do any harm. This is
0EC3      ;      consistant with the notion that the area below the user's
0EC3      ;      stack pointer is being continually garbaged. This check is not
0EC3      ;      performed if flag bit 0 is set. The routines set flag bit 1 if
0EC3      ;      the operand is a register mode operand.
0EC3      ;
0EC3      ;          The subroutine LOCAL_TEST is available for checking for
0EC3      ;      stores into the Emulator's local storage in places where this
0EC3      ;      is not done automatically.
0EC3      ;
0EC3      ;
0EC3      ;      Exceptions
0EC3      ;      ----------
0EC3      ;
0EC3      ;          The instruction operand scanning routines perform complete
0EC3      ;      error checking and immediatly signal any exceptions detected.
0EC3      ;      All of these exceptions are faults. The change bytes are
0EC3      ;      constantly kept up to date so the instruction will be left in
0EC3      ;      a consistant state for restarting if a fault occurs.
0EC3      ;
0EC3      ;          All fetches from memory done in scanning the instruction
0EC3      ;      operand or in fetching the operand or operand address are
0EC3      ;      probed and access violations are signaled if the probes fail.
0EC3      ;      All of the addressing modes specified by the architecture to
```

```
0EC3    ;       be reserved addressing modes or unpredictable are checked for
0EC3    ;       and are signaled as reserved addressing modes if they are
0EC3    ;       detected.
0EC3    ;
0EC3    ;
0EC3    ;       Routine Organization
0EC3    ;       --------------------
0EC3    ;
0EC3    ;           Except for BRANCH access mode for which there are only
0EC3    ;       isolated routines, the code at the individual routine
0EC3    ;       entrances simply loads the data type code into R9 and branches
0EC3    ;       to a routine for the access type. This routine in turn loads
0EC3    ;       the access type code into R8 and branches to the routine
0EC3    ;       GET_SPECIFIER which process the operand specifier byte.
0EC3    ;
0EC3    ;           GET_SPECIFIER loads the length of the data type into R10
0EC3    ;       and the operand specifier byte into R0. The high and low
0EC3    ;       order nibbles of this byte are stored in R1 and R2. The
0EC3    ;       register R7 which is reserved for the index modification is
0EC3    ;       cleared. The routine now branches on the high order nibble
0EC3    ;       to a routine which will handle the specific kind of operand.
0EC3    ;
0EC3    ;           For literals the values are expanded immediatly and the
0EC3    ;       routine returns.
0EC3    ;
0EC3    ;           For index mode operand specifiers, the index modification
0EC3    ;       is computed and the next operand specifier byte is loaded into
0EC3    ;       R0 and decomposed as before. Again we branch on the high order
0EC3    ;       nibble but this time certain addressing modes which are
0EC3    ;       illegal with indexing are checked for. Also for those
0EC3    ;       addressing modes which change register values a check is made
0EC3    ;       that the register is not the same as the index register.
0EC3    ;
0EC3    ;           For register mode operands the address of the emulated
0EC3    ;       register is loaded into R11. A check is made that the operand
0EC3    ;       does not contain PC. Then flag bit 1 is set and control passes
0EC3    ;       to the operand reading routine if the access type is READ or
0EC3    ;       MODIFY and the routine returns otherwise.
0EC3    ;
0EC3    ;           For the remaining addressing modes the operand addresses
0EC3    ;       are computed in a straightforward manner and loaded into R11.
0EC3    ;       for some of these addressing modes tne values of registers may
0EC3    ;       be changed. These changes are reflected in the change bytes.
0EC3    ;       When the operand address is computed control passes to the
0EC3    ;       operand accessing routine.
0EC3    ;
0EC3    ;           For ADDRESS access mode operands the operand accessing
0EC3    ;       routine returns but for all others it probes the operand
0EC3    ;       address and also checks for writes into the Emulator's local
0EC3    ;       storage unless flag bit 0 is set. If the operand is READ or
0EC3    ;       MODIFY access control passes to the operand reading routine.
0EC3    ;
0EC3    ;           The operand reading routine simply reads the operand
0EC3    ;       value into the registers starting at R0 and then returns.
0EC3    ;       Bytes and Words are sign extended into longwords. However
0EC3    ;       this routine does not check for reserved floating values
0EC3    ;       since this is done by the unpack routines.
```

```
0EC3  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
```

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

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

```
                               1020         ;
                               1020         ;      Process an Autodecrement Mode Operand Specifier
                               1020         ;
                               1020  DECR_MODE:                                    ; entrance
         52    0F     D1      1020         CMPL    #15,R2                         ; is the register PC ?
               03     14      1023         BGTR    1$                             ; no - skip
             0C4D     31      1025         BRW     ADDRESS_FAULT                  ; process the reserved addressing mode
      14 AD42  5A     C2      1028  1$:    SUBL2   R10,REG_RO(FP)[R2]             ; subtract data size from register
      EB AD42  5A     82      102D         SUBB2   R10,REGMOD_RO(FP)[R2]          ; remember the subtraction
 5B   14 AD42  57     C1      1032         ADDL3   R7,REG_RO(FP)[R2],R11          ; form the operand address
             0157     31      1038         BRW     ACCESS_VALUE                   ; finish establishing the access
                               103B         ;
                               103B         ;      Process an Autoincrement Mode Operand Specifier
                               103B         ;
                               103B  INCR_MODE:                                    ; entrance
         52    0F     D1      103B         CMPL    #15,R2                         ; is the register PC ?
               0C     14      103E         BGTR    2$                             ; no - bypass
      03   01  58     CF      1040         CASEL   R8,#1,#3                       ; branch on the access type
                    0008'     1044  1$:    .WORD   2$-1$                          ; 1 - read only access
                    0C31'     1046         .WORD   ADDRESS_FAULT-1$               ; 2 - write only access
                    0C31'     1048         .WORD   ADDRESS_FAULT-1$               ; 3 - modify access
                    0008'     104A         .WORD   2$-1$                          ; 4 - address access
 5B   14 AD42  57     C1      104C  2$:    ADDL3   R7,REG_RO(FP)[R2],R11          ; form the operand address
      14 AD42  5A     C0      1052         ADDL2   R10,REG_RO(FP)[R2]             ; add the data size to the register
      EB AD42  5A     80      1057         ADDB2   R10,REGMOD_RO(FP)[R2]          ; remember the addition
             0133     31      105C         BRW     ACCESS_VALUE                   ; finish establishing the access
                               105F         ;
                               105F         ;      Process an Autoincrement Deferred Mode Operand Specifier
                               105F         ;
                               105F  INCR_DEF_MODE:                                ; entrance
      5B   14 AD42    D0      105F         MOVL    REG_RO(FP)[R2],R11             ; R11 = register value
   6B   04    FD AD   0C      1064         PROBER  MODE(FP),#4,(R11)              ; can we read longword it addresses ?
               06     12      1069         BNEQ    1$                             ; yes - skip
         5A    04     D0      106B         MOVL    #4,R10                         ; R10 = size of probe
             0B3A     30      106E         BSBW    READ_FAULT                     ; process the access violation
      5B   6B  57     C1      1071  1$:    ADDL3   R7,(R11),R11                   ; form the operand address
      14 AD42  04     C0      1075         ADDL2   #4,REG_RO(FP)[R2]              ; add longword size to the register
      EB AD42  04     80      107A         ADDB2   #4,REGMOD_RO(FP)[R2]           ; remember the incrementation
             0110     31      107F         BRW     ACCESS_VALUE                   ; finish establishing the access
                               1082         ;
                               1082         ;      Process a Byte Displacement Mode Operand Specifier
                               1082         ;
                               1082  BYTE_DISP_MODE:                               ; entrance
         5B    50 AD  D0      1082         MOVL    REG_PC(FP),R11                 ; R11 = location of displacement
   6B   01    FD AD   0C      1086         PROBER  MODE(FP),#1,(R11)              ; can we read the displacement ?
               06     12      108B         BNEQ    1$                             ; yes - skip
         5A    01     D0      108D         MOVL    #1,R10                         ; R10 = size of probe
             0B18     30      1090         BSBW    READ_FAULT                     ; process the access violation
         5B    6B     98      1093  1$:    CVTBL   (R11),R11                      ; R11 = displacement value
            50 AD     D6      1096         INCL    REG_PC(FP)                     ; increment PC
            FA AD     D6      1099         INCL    REGMOD_PC(FP)                  ; remember the increment
         5B    57     C0      109C         ADDL2   R7,R11                         ; add the displacement to the index
      5B   14 AD42    C0      109F         ADDL2   REG_RO(FP)[R2],R11             ; add the register to the result
             00EB     31      10A4         BRW     ACCESS_VALUE                   ; finish establishing the access
                               10A7         ;
                               10A7         ;      Process a Byte Displacement Deferred Mode Operand Specifier
                               10A7         ;
```

```
                         10A7  BYTE_DEF_MODE:                                       ; entrance
      5B    50 AD   D0   10A7              MOVL    REG_PC(FP),R11                   ; R11 = location of displacement
   6B    01  FD AD   0C   10AB              PROBER  MODE(FP),#1,(R11)                ; can we read the displacement ?
               06   12   10B0              BNEQ    1$                               ; yes - skip
         5A    01   D0   10B2              MOVL    #1,R10                           ; R10 = size of probe
             0AF3   30   10B5              BSBW    READ_FAULT                       ; process the access violation
         5B    6B   98   10B8  1$:         CVTBL   (R11),R11                        ; R11 = displacement value
            50 AD   D6   10BB              INCL    REG_PC(FP)                       ; increment PC
            FA AD   96   10BE              INCB    REGMOD_PC(FP)                    ; remember the increment
      5B  14 AD42   C0   10C1              ADDL2   REG_R0(FP)[R2],R11               ; add the register to the displacement
   6B    04  FD AD   0C   10C6              PROBER  MODE(FP),#4,(R11)                ; can we read longword it addresses ?
               06   12   10CB              BNEQ    2$                               ; yes - skip
         5A    04   D0   10CD              MOVL    #4,R10                           ; R10 = size of probe
             0AD8   30   10D0              BSBW    READ_FAULT                       ; process the access fault
      5B    6B 57   C1   10D3  2$:         ADDL3   R7,(R11),R11                     ; form the operand address
             00B8   31   10D7              BRW     ACCESS_VALUE                     ; finish establishing the access
                         10DA              ;
                         10DA              ;       Process a Word Displacement Mode Operand Specifier
                         10DA              ;
                         10DA  WORD_DISP_MODE:                                      ; entrance
      5B    50 AD   D0   10DA              MOVL    REG_PC(FP),R11                   ; R11 = location of the displacement
   6B    02  FD AD   0C   10DE              PROBER  MODE(FP),#2,(R11)                ; can we read the displacement
               06   12   10E3              BNEQ    1$                               ; yes - skip
         5A    02   D0   10E5              MOVL    #2,R10                           ; R10 = size of probe
             0AC0   30   10E8              BSBW    READ_FAULT                       ; process the access violation
         5B    6B   32   10EB  1$:         CVTWL   (R11),R11                        ; R11 = displacement value
      50 AD    02   C0   10EE              ADDL2   #2,REG_PC(FP)                    ; increment PC
      FA AD    02   80   10F2              ADDB2   #2,REGMOD_PC(FP)                 ; remember the increment
         5B    57   C0   10F6              ADDL2   R7,R11                           ; add the index to the displacement
      5B  14 AD42   C0   10F9              ADDL2   REG_R0(FP)[R2],R11               ; add the register to the result
             0091   31   10FE              BRW     ACCESS_VALUE                     ; finish establishing the access
                         1101              ;
                         1101              ;       Process a Word Displacement Deferred Mode Operand Specifier
                         1101              ;
                         1101  WORD_DEF_MODE:                                       ; entrance
      5B    50 AD   D0   1101              MOVL    REG_PC(FP),R11                   ; R11 = location of the displacement
   6B    02  FD AD   0C   1105              PROBER  MODE(FP),#2,(R11)                ; can we read the displacement ?
               06   12   110A              BNEQ    1$                               ; yes - skip
         5A    02   D0   110C              MOVL    #2,R10                           ; R10 = size of probe
             0A99   30   110F              BSBW    READ_FAULT                       ; process the access violation
         5B    6B   32   1112  1$:         CVTWL   (R11),R11                        ; R11 = displacement value
      50 AD    02   C0   1115              ADDL2   #2,REG_PC(FP)                    ; increment PC
      FA AD    02   80   1119              ADDB2   #2,REGMOD_PC(FP)                 ; remember the increment
      5B  14 AD42   C0   111D              ADDL2   REG_R0(FP)[R2],R11               ; add the register to the displacement
   6B    04  FD AD   0C   1122              PROBER  MODE(FP),#4,(R11)                ; can we read longword it addresses ?
               06   12   1127              BNEQ    2$                               ; yes - skip
         5A    04   D0   1129              MOVL    #4,R10                           ; R10 = size of probe
             0A7C   30   112C              BSBW    READ_FAULT                       ; process the access violation
      5B    6B 57   C1   112F  2$:         ADDL3   R7,(R11),R11                     ; form the operand address
             005C   31   1133              BRW     ACCESS_VALUE                     ; finish establishing the access
                         1136              ;
                         1136              ;       Process a Long Displacement Mode Operand Specifier
                         1136              ;
                         1136  LONG_DISP_MODE:                                      ; entrance
      5B    50 AD   D0   1136              MOVL    REG_PC(FP),R11                   ; R11 = location o' the displacement
   6B    04  FD AD   0C   113A              PROBER  MODE(FP),#4,(R11)                ; can we read the displacement ?
               06   12   113F              BNEQ    1$                               ; yes - skip
```

```
        5A   04  D0 1141           MOVL    #4,R10                  ; R10 = size of probe
           0A64  30 1144           BSBW    READ_FAULT              ; process the access violation
        5B   6B  D0 1147  1$:      MOVL    (R11),R11               ; R11 = displacement value
     50 AD   04  C0 114A           ADDL2   #4,REG_PC(FP)           ; increment PC
     FA AD   04  80 114E           ADDB2   #4,REGMOD_PC(FP)        ; remember the increment
        5B   57  C0 1152           ADDL2   R7,R11                  ; add the index to the displacement
  5B    14 AD42  C0 1155           ADDL2   REG_R0(FP)[R2],R11      ; add the register to the address
           0035  31 115A           BRW     ACCESS_VALUE            ; finish establishing the access
                    115D   ;
                    115D   ;       Process a Long Displacement Deferred Mode Operand Specifier
                    115D   ;
                    115D   LONG_DEF_MODE:                          ; entrance
     5B   50 AD  D0 115D           MOVL    REG_PC(FP),R11          ; R11 = location of the displacement
 6B   04  FD AD  0C 1161           PROBER  MODE(FP),#4,(R11)       ; can we read the displacement ?
             06  12 1166           BNEQ    1$                      ; yes - skip
        5A   04  D0 1168           MOVL    #4,R10                  ; R10 = size of probe
           0A3D  30 116B           BSBW    READ_FAULT              ; process the access violation
        5B   6B  D0 116E  1$:      MOVL    (R11),R11               ; R11 = displacement value
     50 AD   04  C0 1171           ADDL2   #4,REG_PC(FP)           ; increment PC
     FA AD   04  80 1175           ADDB2   #4,REGMOD_PC(FP)        ; remember the increment
  5B    14 AD42  C0 1179           ADDL2   REG_R0(FP)[R2],R11      ; add the register to the displacement
 6B   04  FD AD  0C 117E           PROBER  MODE(FP),#4,(R11)       ; can we read longword it addresses ?
             06  12 1183           BNEQ    2$                      ; yes - skip
        5A   04  D0 1185           MOVL    #4,R10                  ; R10 = size of probe
           0A20  30 1188           BSBW    READ_FAULT              ; process the access violation
  5B    6B   57  C1 118B  2$:      ADDL3   R7,(R11),R11            ; form the operand address
           0000  31 118F           BRW     ACCESS_VALUE            ; finish establishing the access
                    1192   ;
                    1192   ;       Set Up the Type of Access Requested
                    1192   ;
                    1192   ACCESS_VALUE:                           ; entrance
  03    01   58  CF 1192           CASEL   R8,#1,#3                ; branch on the access type
           000D' 1196  1$:         .WORD   READ_CHECK-1$           ; 1 - read only access
           0019' 1198              .WORD   WRITE_CHECK-1$          ; 2 - write only access
           0009' 119A              .WORD   MODIFY_CHECK-1$         ; 3 - modify access
           0008' 119C              .WORD   2$-1$                   ; 4 - address access
             05  119E  2$:         RSB                             ; return with the operand address
                    119F   ;
                    11)F   ;       Perform Error Checking for Modify Access Operands
                    119F   ;
                    119F   MODIFY_CHECK:                           ; entrance
             0E  10 119F           BSBB    WRITE_CHECK             ; check write (and read) access
             1F  11 11A1           BRB     READ_VALUE              ; load the value
                    11A3   ;
                    11A3   ;       Perform Error Checking for Read Only Access Operands
                    11A3   ;
                    11A3   READ_CHECK:                             ; entrance
 6B   5A  FD AD  0C 11A3           PROBER  MODE(FP),R10,(R11)      ; can we read the operand ?
             03  12 11A8           BNEQ    1$                      ; yes - load the value
           09FE  30 11AA           BSBW    READ_FAULT              ; process the access violation
             13  11 11AD  1$:      BRB     READ_VALUE              ; load the value
                    11AF   ;
                    11AF   ;       Perform Error Checking for Write Only Access Operands
                    11AF   ;
                    11AF   WRITE_CHECK:                            ; entrance
 6B   5A  FD AD  0D 11AF           PROBEW  MODE(FP),R10,(R11)      ; can we write the operand ?
             0B  12 11B4           BNEQ    1$                      ; yes - bypass
```

```
            0A63   30  11B6          BSBW    WRITE_FAULT             ; process the access violation
   03 FC AD   00   E0  11B9          BBS     #FLAG0,FLAGS(FP),1$     ; no local store checking - skip
            004C   30  11BE          BSBW    LOCAL_TEST              ; test for a write into local storage
                   05  11C1  1$:     RSB                             ; return
                       11C2          ;
                       11C2          ;       Load a Read Operand into the Registers
                       11C2          ;
                       11C2  READ_VALUE:                             ; entrance
     08    01   59   CF  11C2        CASEL   R9,#1,#8                ; branch on the data type
               0012'   11C6  1$:     .WORD   2$-1$                   ; 1 - byte
               0016'   11C8          .WORD   3$-1$                   ; 2 - word
               001A'   11CA          .WORD   4$-1$                   ; 3 - longword
               0022'   11CC          .WORD   6$-1$                   ; 4 - quadword
               001E'   11CE          .WORD   5$-1$                   ; 5 - octaword
               001A'   11D0          .WORD   4$-1$                   ; 6 - F_floating
               0022'   11D2          .WORD   6$-1$                   ; 7 - D_floating
               0022'   11D4          .WORD   6$-1$                   ; 8 - G_floating
               001E'   11D6          .WORD   5$-1$                   ; 9 - H_floating
          50    6B   98  11D8  2$:   CVTBL   (R11),R0                ; R0 = operand value
                     05  11DB        RSB                             ; return
          50    6B   32  11DC  3$:   CVTWL   (R11),R0                ; R0 = operand value
                     05  11DF        RSB                             ; return
          50    6B   D0  11E0  4$:   MOVL    (R11),R0                ; R0 = operand value
                     05  11E3        RSB                             ; return
       52    08 AB   7D  11E4  5$:   MOVQ    8(R11),R2               ; R2,R3 = high order quadword of value
          50    6B   7D  11E8  6$:   MOVQ    (R11),R0                ; R0,R1 = low order quadword of value
                     05  11EB        RSB                             ; return
                         11EC        ;
                         11EC        ;       Process a Word Branch Displacement Operand
                         11EC        ;
                         11EC  BRANCH_WORD:                          ; entrance
       5B    50 AD   D0  11EC        MOVL    REG_PC(FP),R11          ; R11 = location of the displacement
  6B   02    FD AD   0C  11F0        PROBER  MODE(FP),#2,(R11)       ; can we read the displacement ?
             06      12  11F5        BNEQ    1$                      ; yes - skip
       5A    02      D0  11F7        MOVL    #2,R10                  ; R10 = size of probe
           09AE      30  11FA        BSBW    READ_FAULT              ; process the access violation
       5B    6B      32  11FD  1$:   CVTWL   (R11),R11               ; R11 = branch displacement
    50 AD    02      C0  1200        ADDL2   #2,REG_PC(FP)           ; increment PC
    FA AD    02      80  1204        ADDB2   #2,REGMOD_PC(FP)        ; remember the increment
       5B    50 AD   C0  1208        ADDL2   REG_PC(FP),R11          ; compute the branch destination
                     05  120C        RSB                             ; return
                         120D        ;
                         120D        ;       Test for a Write into Local Storage
                         120D        ;
                         120D        ;               entered by subroutine branching
                         120D        ;
                         120D        ;               parameters:     R10 = Number of Bytes to be Written
                         120D        ;                               R11 = Distination Address
                         120D        ;
                         120D        ;               returns with    R11 = Corrected Destination Address
                         120D        ;
                         120D        ;       Discussion
                         120D        ;
                         120D        ;           This routine checks the write operation described by
                         120D        ;       the parameters in R10 and R11 for a write into the Emulator's
                         120D        ;       working storage. If such a write is about to take place, R11
                         120D        ;       is changed to an address where the write will not do any harm.
```

```
                      120D  3451               ;
                      120D  3452 LOCAL_TEST:                                  ; entrance
  53    58 AD    9E   120D  3453         MOVAB   LOCAL_END(FP),R3             ; R3 = byte following local storage
     53    5B    D1   1211  3454         CMPL    R11,R3                       ; is the write above the frame ?
           0D    1E   1214  3455         BGEQU   1$                           ; yes - bypass
 53    5B  5A    C1   1216  3456         ADDL3   R10,R11,R3                   ; R3 = byte following operand
       5E  53    D1   121A  3457         CMPL    R3,SP                        ; is it above the stack pointer ?
           04    1B   121D  3458         BLEQU   1$                           ; no - operand is not in local storage
  5B    58 AD    9E   121F  3459         MOVAB   TEMP(FP),R11                 ; redirect the write to TEMP
                 05   1223  3460 1$:     RSB                                  ; return with the operand address
                      1224  3461         ;
```

```
1224  ;       ****************************************************************
1224  ;       *                                                              *
1224  ;       *                                                              *
1224  ;       *     Routines for Unpacking and Packing Floating Values       *
1224  ;       *                                                              *
1224  ;       *                                                              *
1224  ;       ****************************************************************
1224  ;
1224  ;
1224  ;       Introduction
1224  ;       ------------
1224  ;
1224  ;           The following routines perform all of the conversions
1224  ;       between the VAX floating representations and the internal
1224  ;       representation used by the Emulator. The unpack routines
1224  ;       convert from the VAX representation to the internal
1224  ;       representation and the pack routines perform the opposite
1224  ;       conversion. These routines perform all of the necessary
1224  ;       rounding and check for reserved values, underflow, and
1224  ;       overflow.
1224  ;
1224  ;
1224  ;       The Unpack Routines
1224  ;       -------------------
1224  ;
1224  ;           The unpack routines convert a value in one of the VAX
1224  ;       floating representations to our internal representation.
1224  ;       The value to be converted is assumed to be contained in
1224  ;       the registers starting at R0. For floating and double
1224  ;       floating values the available unpack routines only place
1224  ;       the converted value in OPERAND1. For GFLOAT and HFLOAT routines
1224  ;       are available which place the result in all of the operand
1224  ;       areas.
1224  ;
1224  ;           The unpack routines all check for a reserved floating
1224  ;       value (sign bit set and biased exponent equal to zero) and
1224  ;       signal a reserved operand exception if one is found.
1224  ;
1224  ;
1224  ;       The Pack Routines
1224  ;       -----------------
1224  ;
1224  ;           The pack routines convert a value in the internal
1224  ;       representation to one of the VAX floating representations.
1224  ;       The value to be converted is assumed to be in one of the
1224  ;       operand areas and the value must be in OPERAND1 if the value
1224  ;       is to be converted to floating or double floating. For GFLOAT
1224  ;       and HFLOAT routines are available to convert from each of the
1224  ;       operand areas. The routines always leave the result in the
1224  ;       registers starting at R0.
1224  ;
1224  ;           Before the value is converted the rounding bit (the first
1224  ;       bit of the fraction which will not appear in the converted
1224  ;       result) is tested and if it is set the value is rounded by
1224  ;       adding one to the next higher bit (the lowest one that will
1224  ;       appear in the converted result) and processing any carries
1224  ;       that occur. When the conversion is performed the biased
```

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

```
                          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           ;
             1241           ;       UNPACK_FFLOAT - Unpack a Floating Value
             1241           ;
             1241           ;               entered by subroutine branching
             1241           ;
             1241           ;               parameter:      R0 = Input Floating Value
             1241           ;
             1241           ;               returns with    (R4) = Converted Value
             1241           ;
             1241           ;               uses R0-R5
             1241           ;
             1241 UNPACK_FFLOAT:                            ; entrance
   51    D4  1241           CLRL    R1                      ; make F look like D
             1243           ;BRB    UNPACK_DFLOAT           ; use D_floating unpack
             1243           ;
```

```
                                1243       ;
                                1243       ;       UNPACK_DFLOAT - Unpack a Double Floating Value
                                1243       ;
                                1243       ;               entered by subroutine branching
                                1243       ;
                                1243       ;               parameter:      R0,R1 = Input Double Floating Value
                                1243       ;
                                1243       ;               returns with    (R4) = Converted Value
                                1243       ;
                                1243       ;               uses R0-R5
                                1243       ;
                                1243  UNPACK_DFLOAT:                                    ; entrance
                   64      D4   1243       CLRL    (R4)                              ; clear the zero and sign flags
   55    50    08  07      EF   1245       EXTZV   #7,#8,R0,R5                       ; R5 = biased exponent
                   0A      12   124A       BNEQ    2$                                ; it's not zero - bypass
         03 50     0F      E1   124C       BBC     #15,R0,1$                         ; no sign bit - skip
                 0A33      31   1250       BRW     OPERAND_FAULT                     ; double floating value is reserved
                   64      96   1253  1$:  INCB    ZERO(R4)                          ; indicate a zero value
                           05   1255       RSB                                       ; return
         03 50     0F      E1   1256  2$:  BBC     #15,R0,3$                         ; no sign bit - skip
               C1  A4      96   125A       INCB    SIGN(R4)                          ; indicate a negative value
      04 A4    80  A5      9E   125D  3$:  MOVAB   -128(R5),POWER(R4)                ; store the unbiased exponent
      50       50  10      9C   1262       ROTL    #16,R0,R0                         ; R0 = leading bits of the fraction
         00 50     17      E2   1266       BBSS    #23,R0,4$                         ; set the hidden bit
 14 A4   18    08  50      F0   126A  4$:  INSV    R0,#8,#24,FRACTION+12(R4); store bits 40-63 of the fraction
      51       51  10      9C   1270       ROTL    #16,R1,R1                         ; R1 = trailing bits of the fraction
         11 A4     51      D0   1274       MOVL    R1,FRACTION+9(R4)                 ; store bits 8-39 of the fraction
            10 A4          94   1278       CLRB    FRACTION+8(R4)                    ; clear bits 0-7 of the fraction
            08 A4          7C   127B       CLRQ    FRACTION(R4)                      ; extend the fraction to 128 bits
                           05   127E       RSB                                       ; return
                                127F       ;
```

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

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

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

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

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

```
                              143B       ;
                              143B       ;       PACK_HFLOAT - Pack an H_floating Value
                              143B       ;
                              143B       ;               entered by subroutine branching
                              143B       ;
                              143B       ;               parameter:      R4 = Location of Source Value
                              143B       ;
                              143B       ;               returns with    R0,R1,R2,R3 = Converted HFLOAT Value
                              143B       ;
                              143B PACK_HFLOAT:                          ; entrance
                  05 64   E9  143B       BLBC    ZERO(R4),2$             ; value is not zero - bypass
                     50   7C  143E 1$:   CLRQ    R0                      ; clear the first part of the value
                     52   7C  1440       CLRQ    R2                      ; clear the second part of the value
                          05  1442       RSB                             ; return
            1F 08 A4  0E  E1  1443 2$:   BBC     #14,FRACTION(R4),3$     ; rounding bit is zero - bypass
    08 A4   00008000  8F  C0  1448       ADDL2   #1@15,FRACTION(R4)      ; round the value
               0C A4  00  D8  1450       ADWC    #0,FRACTION+4(R4)       ; propagate carry into third part
               10 A4  00  D8  1454       ADWC    #0,FRACTION+8(R4)       ; propagate carry into second part
               14 A4  00  D8  1458       ADWC    #0,FRACTION+12(R4)      ; propagate carry into first part
                      09  1E  145C       BCC     3$                      ; no carry out of fraction - bypass
                  04 A4   D6  145E       INCL    POWER(R4)               ; increment the exponent
                     50   7C  1461       CLRQ    R0                      ; clear first part of the fraction
                     52   7C  1463       CLRQ    R2                      ; clear second part of the fraction
                     23   11  1465       BRB     4$                      ; bypass
       50   14 A4     01  78  1467 3$:   ASHL    #1,FRACTION+12(R4),R0   ; load first part of the fraction
   51  10 A4  20      0F  EE  146C       EXTV    #15,#32,FRACTION+8(R4),R1 ; load second part of the fraction
          51  51      10  9C  1472       ROTL    #16,R1,R1               ; reverse the words
   52  0C A4  20      0F  EE  1476       EXTV    #15,#32,FRACTION+4(R4),R2 ; load third part of the fraction
          52  52      10  9C  147C       ROTL    #16,R2,R2               ; reverse the words
   53  08 A4  20      0F  EE  1480       EXTV    #15,#32,FRACTION(R4),R3 ; load third part of the fraction
          53  53      10  9C  1486       ROTL    #16,R3,R3               ; reverse the words
55   04 A4  00004000  8F  C1  148A 4$:   ADDL3   #16384,POWER(R4),R5     ; R5 = biased exponent
                      0B  14  1493       BGTR    5$                      ; it's greater than zero - bypass
                  64  01  D0  1495       MOVL    #1,(R4)                 ; mark the original value as zero
            A1 54 AD  06  E1  1498       BBC     #PSL_FU,PSL(FP),1$      ; no fault enabled - return zero
                    07F7  31  149D       BRW     UNDERFLOW               ; process the floating underflow
        55  00007FFF  8F  D1  14A0 5$:   CMPL    #32767,R5               ; is the exponent too large ?
                      03  18  14A7       BGEQ    6$                      ; no - skip
                    07FC  31  14A9       BRW     OVERFLOW                ; process the floating overflow
                  50  55  B0  14AC 6$:   MOVW    R5,R0                   ; insert exponent and clear sign
             04 01 A4     E9  14AF       BLBC    SIGN(R4),7$             ; is the value negative ?
             00 50    0F  E2  14B3       BBSS    #15,R0,7$               ; yes - set the sign bit
                          05  14B7 7$:   RSB                             ; return
                              14B8       ;
```

```
                        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
```

```
14E3    ;       ****************************************************************
14E3    ;       *                                                              *
14E3    ;       *                                                              *
14E3    ;       *                      Arithmetic Routines                     *
14E3    ;       *                                                              *
14E3    ;       *                                                              *
14E3    ;       ****************************************************************
14E3    ;
14E3    ;
14E3    ;       Introduction
14E3    ;       ------------
14E3    ;
14E3    ;           The routines which follow perform the actual floating
14E3    ;       arithmetic operations of the Emulator. These routines all
14E3    ;       work with the internal floating representation so only one
14E3    ;       routine is needed for each type of operation. However, since
14E3    ;       multiplication and division are comparatively slow operations
14E3    ;       separate routines have been included for the GFLOAT and HFLOAT
14E3    ;       versions of these operations so the GFLOAT operation will not
14E3    ;       be slowed to the speed of the HFLOAT operation.
14E3    ;
14E3    ;           The algorithms for the individual routines will be
14E3    ;       described in the routines themselves. The following discussion
14E3    ;       will be limited to a description of our internal floating
14E3    ;       format.
14E3    ;
14E3    ;
14E3    ;       Internal Floating Representation
14E3    ;       --------------------------------
14E3    ;
14E3    ;           All of the floating arithmetic operations used by the
14E3    ;       Emulator are performed using an internal floating format
14E3    ;       which is much easier to work with than any of the hardware
14E3    ;       floating representations and which is sufficiently accurate
14E3    ;       to represent all of the hardware representations. The format
14E3    ;       is also used as an intermediate representation for emulating
14E3    ;       the conversion instructions.
14E3    ;
14E3    ;           The internal representation has the following four fields:
14E3    ;
14E3    ;
14E3    ;               ZERO            is a one byte field whose low order
14E3    ;                               bit indicates that the represented
14E3    ;                               value is nonzero.
14E3    ;
14E3    ;               SIGN            is a one byte field whose low order
14E3    ;                               bit indicates that the represented
14E3    ;                               value is negative. If the low order
14E3    ;                               bit of ZERO is set then this field
14E3    ;                               must be zero.
14E3    ;
14E3    ;               POWER           is a longword field which contains the
14E3    ;                               exponent of the power of two which
14E3    ;                               is used to scale the fraction
14E3    ;
14E3    ;               FRACTION        is a 128 bit field (four longwords)
14E3    ;                               which hold the fraction as a single
```

```
14E3      ;                               128 bit value. The decimal point is
14E3      ;                               assumed to be at the end of the
14E3      ;                               fraction next to the high order bit.
14E3      ;                               The fraction is considered to be a
14E3      ;                               a positive value and is normalized
14E3      ;                               if the high order bit is one.
14E3      ;
14E3      ;
14E3      ;       Three areas OPERAND1, OPERAND2, OPERAND3 are available for
14E3      ;       holding values in the internal representation. The name of
14E3      ;       of a field of one of these areas is found by appending the
14E3      ;       trailing digit of the area name to the field name. Thus
14E3      ;       POWER2 is the POWER field of OPERAND2.
14E3      ;
```

```
                        14E3              ;
                        14E3              ;       NEGATE_REAL - Negate a Real Value
                        14E3              ;
                        14E3              ;               entered by subroutine branching
                        14E3              ;
                        14E3              ;               parameter:      OPERAND1 = The Floating Value
                        14E3              ;
                        14E3              ;               returns with    OPERAND1 = The Negated Result
                        14E3              ;
                        14E3              ;       Discussion
                        14E3              ;
                        14E3              ;           If the value is nonzero, then the sign of the value
                        14E3              ;       is complemented.
                        14E3              ;
                        14E3   NEGATE_REAL:                                    ; entrance
        04 C7 AD   E8   14E3              BLBS    ZERO1(FP),1$                 ; don't negate zero
      C8 AD   01   8C   14E7              XORB2   #1,SIGN1                     ; complement the sign
                   05   14EB   1$:        RSB                                  ; return
                        14EC              ;
                        14EC              ;       FLOAT_LONG - Convert a Longword to a Floating Value
                        14EC              ;
                        14EC              ;               entered by subroutine branching
                        14EC              ;
                        14EC              ;               parameter:      R0 = The Longword Value
                        14EC              ;
                        14EC              ;               returns with    OPERAND1 = The Converted Value
                        14EC              ;
                        14EC              ;       Discussion
                        14EC              ;
                        14EC              ;           The longword is converted to double floating using the
                        14EC              ;       hardware and then to the internal representation by one of
                        14EC              ;       the unpack routines.
                        14EC              ;
                        14EC   FLOAT_LONG:                                     ; entrance
                        14EC   ;*
                        14EC   ; The original code used CVTLD, which we can no longer assume is supported
                        14EC   ; in hardware.  The action of a CVTLD R0,R0 is therefore emulated using
                        14EC   ; non-floating instructions.
                        14EC   ;
                        14EC   ;          CVTLD   R0,R0                        ; convert the value to double floating
                        14EC   ;*
52     000000A0 8F   D0   14EC            MOVL    #<128+32>,R2                 ; set initial biased exponent
              51   D4   14F3              CLRL    R1                           ; zero "low" longword
              50   D5   14F5              TSTL    R0                           ; test sign of value
              23   13   14F7              BEQL    30$                          ; skip if zero
              0A   14   14F9              BGTR    20$                          ; skip if positive
      52  0100 8F   A8   14FB             BISW2   #^X100,R2                    ; remember that it is negative
         50   50   CE   1500              MNEGL   R0,R0                        ; get |R0|
              08   1D   1503              BVS     25$                          ; skip shift loop if -2**32
              52   D7   1505   20$:       DECL    R2                           ; decrement binary exponent
      50   50   01   78   1507            ASHL    #1,R0,R0                     ; shift value left one bit
              F8   1C   150B              BVC     20$                          ; repeat until bit 31 set
      50   50   08   9C   150D   25$:     ROTL    #8,R0,R0                     ; rearrange into floating format
51   50  00FF 8F   AB   1511              BICW3   #^X00FF,R0,R1                ; move low 8 bits into place in R1
50   09   07   52   F0   1517             INSV    R2,#7,#9,R0                  ; move sign+exponent into place
      54   C7 AD   9E   151C   30$:       MOVAB   OPERAND1(FP),R4              ; set operand address
           FD20   30   1520               BSBW    UNPACK_DFLOAT                ; unpack the value
```

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

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

```
                                1578        ;
                                1578        ;       ROUND_REAL - Convert a Floating Value to a Longword (Rounded)
                                1578        ;
                                1578        ;               entered by subroutine branching
                                1578        ;
                                1578        ;               parameter:      OPERAND1 = The Floating Value
                                1578        ;
                                1578        ;               returns with    R0 = The Longword Result
                                1578        ;
                                1578        ;       Discussion
                                1578        ;
                                1578        ;               The exponent of the floating value is used to determine
                                1578        ;       where the decimal point goes within the signed fraction and
                                1578        ;       whatever part of the integer part exists within the fraction
                                1578        ;       is extracted. If the bit immediatly below the decimal point is
                                1578        ;       nonzero, then the integer part is rounded by adding a value
                                1578        ;       of one with the same sign as the floating value. If the value
                                1578        ;       contains significant bits of higher order than those in the
                                1578        ;       fraction or if overflow occured during the rounding operation
                                1578        ;       then the V bit is set in the user's PSL to indicate an integer
                                1578        ;       overflow.
                                1578        ;
                                1578        ROUND_REAL:                                             ; entrance
                  50    D4      1578                CLRL    R0                              ; clear the returned value
            39 C7 AD    E8      157A                BLBS    ZERO1(FP),3$                    ; the value is zero - return
         51    CB AD    CE      157E                MNEGL   POWER1(FP),R1                   ; R1 = negated exponent
                  33    14      1582                BGTR    3$                              ; the value is less than 0.5 - return
            CB AD 20    D1      1584                CMPL    #32,POWER1(FP)                  ; is the decimal point deep inside ?
                  2E    19      1588                BLSS    4$                              ; yes - bypass
 50   DF AD  CB AD 51   EF      158A                EXTZV   R1,POWER1(FP),FRACTION1+16(FP),R0 ; extract some leading bits
               51 3F    C0      1591                ADDL2   #63,R1                          ; *** equivalent sequence to ***
         08 D7 AD 51    E1      1594                BBC     R1,FRACTION1+8(FP),1$           ; *** get around hardware problem ***
                                1599        ;       DECL    R1                              ; compute the rounding position
                                1599        ;       BBC     R1,FRACTION1+16(FP),1$          ; rounding bit is clear - skip
                  50    D6      1599                INCL    R0                              ; round the extracted bits
                  04    1E      159B                BCC     1$                              ; no carry - skip
            54 AD 02    C8      159D                BISL2   #PSLM_V,PSL(FP)                 ; indicate an integer overflow
            03 C8 AD    E9      15A1        1$:     BLBC    SIGN1(FP),2$                    ; is the floating value negative
               50 50    CE      15A5                MNEGL   R0,R0                           ; yes - complement the value
            51 50 01    9C      15A8        2$:     ROTL    #1,R0,R1                        ; position the sign bit
         51    C8 AD    8C      15AC                XORB2   SIGN1(FP),R1                    ; complment it if negative value
               04 51    E9      15B0                BLBC    R1,3$                           ; the sign is correct - skip
            54 AD 02    C8      15B3                BISL2   #PSLM_V,PSL(FP)                 ; indicate an integer overflow
                        05      15B7        3$:     RSB                                     ; return
            54 AD 02    C8      15B8        4$:     BISL2   #PSLM_V,PSL(FP)                 ; indicate an integer overflow
         51    00A0 C1  9E      15BC                MOVAB   160(R1),R1                      ; switch origin to previous longword
                  F4    15      15C1                BLEQ    3$                              ; no nonzero bits - return
   50    CB AD  20 51   EF      15C3                EXTZV   R1,#32,FRACTION1-4(FP),R0 ; extract 32 bits from the fraction
         52    20 51    C3      15C9                SUBL3   R1,#32,R2                       ; compute the bits to clear
                  07    19      15CD                BLSS    5$                              ; possible rounding - skip
   50    52    00 00    F0      15CF                INSV    #0,#0,R2,R0                     ; clear some low order bits
                  09    11      15D4                BRB     6$                              ; bypass
                  51    D7      15D6        5$:     DECL    R1                              ; compute the rounding bit
         02 CB AD 51    E1      15D8                BBC     R1,FRACTION1-4(FP),6$           ; the rounding bit is clear - skip
                  50    D6      15DD                INCL    R0                              ; round the extracted bits
            D4 C8 AD    E9      15DF        6$:     BLBC    SIGN1(FP),3$                    ; the value is positive - return
               50 50    CE      15E3                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           ;
```

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

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

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

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

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

```
                          1852        .SBTTL  DIVIDE_FLOAT
                          1852
                          1852        ;
                          1852        ;       DIVIDE_FLOAT - Divide Two Floating Values
                          1852        ;
                          1852        ;               entered by subroutine branching
                          1852        ;
                          1852        ;               parameters:     OPERAND2 = The Floating Dividend
                          1852        ;                               OPERAND3 = The Floating Divisor
                          1852        ;
                          1852        ;               returns with    OPERAND1 = Floating Quotient
                          1852        ;
                          1852        ;       See DIVIDE_xFLOAT routines for more information.
                          1852        ;
                          1852
                          1852 DIVIDE_FLOAT:
03    06    8B BD   8F    1852        CASEB   @OP_INDEX(FP),#TYP_F,#<TYP_H-TYP_F>
                0008'     1857 1$:    .WORD   DIVIDE_FFLOAT-1$        ; F_floating
                005B'     1859        .WORD   DIVIDE_DGFLOAT-1$       ; D_floating
                005B'     185B        .WORD   DIVIDE_DGFLOAT-1$       ; G_floating
                00AE'     185D        .WORD   DIVIDE_HFLOAT-1$        ; H_floating
```

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

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

```
                          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          ;
```

```
                                197F            ;
                                197F            ;       NORMALIZE - Normalize a Floating Value
                                197F            ;
                                197F            ;               entered by subroutine branching
                                197F            ;
                                197F            ;               parameter:      OPERAND1 = The Unnormalized Value
                                197F            ;
                                197F            ;               returns with    OPERAND1 = The Normalized Result
                                197F            ;
                                197F            ;       Discussion
                                197F            ;
                                197F            ;           This routine normalizes a floating value in the internal
                                197F            ;       representation so that the high order bit of the fraction is
                                197F            ;       one. This is done by locating the high order significant bit
                                197F            ;       of the fraction and then by shifting the fraction so that the
                                197F            ;       bit appears in the proper position. The shift count is
                                197F            ;       subtracted from the exponent so the value does not change.
                                197F            ;
                                197F    NORMALIZE:                                      ; entrance
              3A C7 AD    E8    197F            BLBS    ZERO1(FP),2$                    ; the value is zero - return
                 DB AD    D5    1983    1$:     TSTL    FRACTION1+12(FP)                ; test the first longword
                    35    19    1986            BLSS    2$                              ; value is already normalized - return
                    34    13    1988            BEQL    3$                              ; first longword is zero - bypass
                    51    D4    198A            CLRL    R1                              ; set intitial shoft count
           50    DB AD    D0    198C            MOVL    FRACTION1+12(FP),R0             ; get high longword of fraction
                    51    D6    1990    11$:    INCL    R1                              ; increment shift count
        50    50    01    78    1992            ASHL    #1,R0,R0                        ; shift one more bit
                    F8    1C    1996            BVC     11$                             ; repeat until high bit set
              50    51    CE    1998            MNEGL   R1,R0                           ; R0 = -shift count
                                199B    ;*
                                199B    ; The previous six instructions are a replacement for the following four
                                199B    ; instructions.  We can no longer guarantee that CVTLD is supported.
                                199B    ;
                                199B    ;       CVTLD   FRACTION1+12(FP),R0             ; use hardware to find shift count
                                199B    ;       EXTZV   #7,#6,R0,R0                     ; R0 = high order bit position + 1
                                199B    ;       SUBL2   #32,R0                          ; R0 = - shift count
                                199B    ;       MNEGL   R0,R1                           ; R1 = shift count
                                199B    ;*
 D7 AD    D7 AD     51    79    199B            ASHQ    R1,FRACTION1+8(FP),FRACTION1+8(FP) ; position first longword
 D7 AD    D7 AD     50    78    19A1            ASHL    R0,FRACTION1+8(FP),FRACTION1+8(FP) ; get second parts together
 D3 AD    D3 AD     51    79    19A7            ASHQ    R1,FRACTION1+4(FP),FRACTION1+4(FP) ; position second longword
 D3 AD    D3 AD     50    78    19AD            ASHL    R0,FRACTION1+4(FP),FRACTION1+4(FP) ; get other parts together
 CF AD    CF AD     51    79    19B3            ASHQ    R1,FRACTION1(FP),FRACTION1(FP) ; position last two longwords
          CB AD     51    C2    19B9            SUBL2   R1,POWER1(FP)                   ; increment the exponent
                    05          19BD    2$:     RSB                                     ; return
          CB AD     20    C2    19BE    3$:     SUBL2   #32,POWER1(FP)                  ; decrement the exponent
                 D7 AD    D5    19C2            TSTL    FRACTION1+8(FP)                 ; test the second longword
                    0F    13    19C5            BEQL    4$                              ; it's zero - bypass
       D7 AD     D3 AD    7D    19C7            MOVQ    FRACTION1+4(FP),FRACTION1+8(FP) ; shift first two longwords
       D3 AD     CF AD    D0    19CC            MOVL    FRACTION1(FP),FRACTION1+4(FP) ; shift third longword
                 CF AD    D4    19D1            CLRL    FRACTION1(FP)                   ; clear the final longword
                    AD    11    19D4            BRB     1$                              ; finish up
          CB AD     20    C2    19D6    4$:     SUBL2   #32,POWER1(FP)                  ; decrement the exponent
                 D3 AD    D5    19DA            TSTL    FRACTION1+4(FP)                 ; test the third longword
                    0A    13    19DD            BEQL    5$                              ; it's zero - bypass
       D7 AD     CF AD    7D    19DF            MOVQ    FRACTION1(FP),FRACTION1+8(FP)   ; position first two longwords
                 CF AD    7C    19E4            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    ;
1A05    ;       MULTIPLY - Unsigned Multiple Length Integer Multiply
1A05    ;
1A05    ;               entered by subroutine branching
1A05    ;
1A05    ;               parameters:     R0 = Size of Inputs in Longwords
1A05    ;                               R1 = Location of First Factor
1A05    ;                               R2 = Location of Second Factor
1A05    ;                               R3 = Location of Destination Area
1A05    ;
1A05    ;       Discussion
1A05    ;
1A05    ;           This routine computes the product of two multiple length
1A05    ;       unsigned integers and stores the unsigned product in a
1A05    ;       specified area. The parameter R0 contains the number of
1A05    ;       longwords in each of the input factors and the number of
1A05    ;       longwords in the product is twice the value of R0. The
1A05    ;       parameters R1 and R2 contain the locations of the input
1A05    ;       factors and the parameter R3 contains the location of the
1A05    ;       area for the product.
1A05    ;
1A05    ;
1A05    ;       The Algorithm
1A05    ;
1A05    ;           The algorithm used is perfectly straightforward. The
1A05    ;       second factor is multiplied by each of the unsigned longwords
1A05    ;       of the first factor and the aligned products are added to the
1A05    ;       destination area. The multiplications and additions are
1A05    ;       performed by a series of EMUL instructions in which the two
1A05    ;       factors are longwords from each of the factors and the added
1A05    ;       operand is formed by adding target longword of the result area
1A05    ;       and from carryover information from the previous iteration
1A05    ;       of the EMUL loop. The carryover information is formed by
1A05    ;       adding the high order bits of the previous product, the
1A05    ;       opposite factor for each factor longword which is negative
1A05    ;       from the previous multiplication, a one if the previous
1A05    ;       carryover longword was negative, and a one if the previous
1A05    ;       additive operand was negative. Most of these contributions are
1A05    ;       compensations for the fact that the EMUL instruction assumes
1A05    ;       that the operands are signed.
1A05    ;
1A05    ;           To show that the algorithm is correct the major step is
1A05    ;       to show that the EMUL operations do not loose information
1A05    ;       because of overflow. To show this we make use of the fact that
1A05    ;       for two's complemented addition and multiplication information
1A05    ;       only moves in the direction of the high order bits. Therefore,
1A05    ;       if the product fits into two unsigned longwords it is correct.
1A05    ;       The unsigned information input to each EMUL step is
1A05    ;
1A05    ;               first factor <= 2^32-1
1A05    ;               second factor <= 2^32-1
1A05    ;               longword from result area <= 2^32-1
1A05    ;               unsigned carryover information <= 2^32-1
1A05    ;
1A05    ;       The output of the step is the product of the first two values
1A05    ;       plus the second two values. This does not exceed
1A05    ;
```

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

```
1A4F    ;
1A4F    ;       DIVIDE - Unsigned Multiple Length Integer Divide
1A4F    ;
1A4F    ;               entered by subroutine branching
1A4F    ;
1A4F    ;               parameters:     R0 = Size of the Divisor in Longwords
1A4F    ;                               R1 = Location of the Dividend
1A4F    ;                               R2 = Location of the Divisor
1A4F    ;                               R3 = Location of the Quotient Area
1A4F    ;
1A4F    ;       Discussion
1A4F    ;
1A4F    ;           This routine performs unsigned multiple length division
1A4F    ;       and develops a quotient and a remainder. The number of
1A4F    ;       longwords in the divisor and in the quotient area is given
1A4F    ;       by the parameter R0. The number of longwords in the dividend
1A4F    ;       area is twice the value of R0. The parameter R1 is the
1A4F    ;       location of the dividend, R2 is the location of the divisor,
1A4F    ;       and R3 is the location for the quotient. The remainder is
1A4F    ;       formed in the low order R0 longwords of the dividend area.
1A4F    ;       It is assumed that the high order bit of the divisor is one
1A4F    ;       and that the high order bit of the dividend is zero. These
1A4F    ;       conditions insure that the quotient will fit into the quotient
1A4F    ;       area.
1A4F    ;
1A4F    ;
1A4F    ;       The Algorithm
1A4F    ;
1A4F    ;           The algorithm used is a variation of the classical divide
1A4F    ;       and correct method which has been adapted for the above
1A4F    ;       specifications. The algorithm has been implemented using words
1A4F    ;       rather than longwords because the word version is much easier
1A4F    ;       to verify for correctness since there are no problems with
1A4F    ;       using signed arithmetic operations to perform unsigned
1A4F    ;       arithmetic.
1A4F    ;
1A4F    ;           We let A[0..2N-1] denote the dividend as an array of words
1A4F    ;       with A[0] being the low order word, and N being twice the
1A4F    ;       value of R0. Similarly we let B[0..N-1] denote the divisor and
1A4F    ;       C[0..N-1] the quotient. The remainder will be developed in the
1A4F    ;       array A[0..N-1]. The algorithm may be given as follows:
1A4F    ;
1A4F    ;
1A4F    ;               Step 0. Let I = N.
1A4F    ;
1A4F    ;               Step 1. Let I = I-1 and A[2N] = 0.
1A4F    ;
1A4F    ;                       { The use of A[2N] is only a simplification,
1A4F    ;                         it does not really appear in the program. }
1A4F    ;
1A4F    ;               Step 2. Let Q = entier(A[I+N-1..I+N+1]/(B[N-1]+1)) and
1A4F    ;                       let R be the remainder from the division.
1A4F    ;
1A4F    ;                       { This operation is performed by a single
1A4F    ;                         EDIV instruction. }
1A4F    ;
1A4F    ;               Step 3. Let X = b*B[N-1]+B[N-2]+1 and
```

```
                                1A4F            ;               Y = -Q*B[N-2]+b*R+A[I+N-2]
                                1A4F            ;                 =  Q*(b-B[N-2]-1)+b*R+A[I+N-2].
                                1A4F            ;
                                1A4F            ;               { Here b denotes 2^16 which is the "base" of
                                1A4F            ;                 the number system we are using and b-B[N-2]-1
                                1A4F            ;                 is the ones complement of -B[N-2]. These
                                1A4F            ;                 values are used to correct the quotient. }
                                1A4F            ;
                                1A4F            ;       Step 4. Let Y = Y-X. If  Y >= 0 then let Q = Q+1 and
                                1A4F            ;               repeat this step.
                                1A4F            ;
                                1A4F            ;               { When this step is complete the value of Q is
                                1A4F            ;
                                1A4F            ;                  entier(A[I+N-2..I+N+1]/(B[N-2..N-1]+1) }
                                1A4F            ;
                                1A4F            ;       Step 5. Let A[I..I+N] = A[I..I+N]-Q*B[0..N-1].
                                1A4F            ;
                                1A4F            ;       Step 6. Let C[I] = Q and if Q is too large then add
                                1A4F            ;               the overflow to C[I+1..N-1]. If I > 2 then
                                1A4F            ;               go to Step 2.
                                1A4F            ;
                                1A4F            ;       Step 7. If B[0..N-1] <= A[0..N] then let
                                1A4F            ;               C[0..N-1] = C[0..N-1]+1 and
                                1A4F            ;               A[0..N-1] = A[0..N-1]-B[0..N-1].
                                1A4F            ;               The division is complete.
                                1A4F            ;
                                1A4F            ;
                                1A4F DIVIDE:                                    ; entrance
         50    50    01   78    1A4F            ASHL    #1,R0,R0                ; convert the longword count to words
               51  6140   3E    1A53            MOVAW   (R1)[R0],R1             ; R1 = current position in dividend
               54  6240   3E    1A57            MOVAW   (R2)[R0],R4             ; R4 = position above divisor
               53  6340   3E    1A5B            MOVAW   (R3)[R0],R3             ; R3 = position above quotient area
            59    FE A4   3C    1A5F            MOVZWL  -2(R4),R9               ; R9 = leading word of divisor
                     59   D6    1A63            INCL    R9                      ; form the trial divisor
            5A    FC A4   D0    1A65            MOVL    -4(R4),R10              ; R10 = leading longword of divisor
                     5A   D6    1A69            INCL    R10                     ; form the correction divisor
            5B    FC A4   B2    1A6B            MCOMW   -4(R4),R11              ; R11 = complemented second word
               5B    5B   3C    1A6F            MOVZWL  R11,R11                 ; form the correction multiplier
               54    50   D0    1A72            MOVL    R0,R4                   ; R4 = loop counter
                     56   D4    1A75            CLRL    R6                      ; clear the carryover information
               51    02   C2    1A77 ONE$:      SUBL2   #2,R1                   ; R1 = current position in dividend
               53    02   C2    1A7A            SUBL2   #2,R3                   ; R3 = current position in quotient
               55  6140   3E    1A7D            MOVAW   (R1)[R0],R5             ; R5 = location of next dividend word
            55    FE A5   D0    1A81            MOVL    -2(R5),R5               ; R5 = leading longword of dividend
      56    55    55 59   7B    1A85            EDIV    R9,R5,R5,R6             ; R5 = quotient, R6 = remainder
                     58   D4    1A8A            CLRL    R8                      ; clear the double length adjustment
         56    56    10   78    1A8C            ASHL    #16,R6,R6               ; position the remainder
                     02   18    1A90            BGEQ    2$                      ; is the value negative ?
                     58   D6    1A92            INCL    R8                      ; yes - the sum will need adjusting
         56    FC A140    B0    1A94 2$:        MOVW    -4(R1)[R0],R6           ; include next word from dividend
      56    56    5B 55   7A    1A99            EMUL    R5,R11,R6,R6            ; form the correction remainder
               57    58   C0    1A9E            ADDL2   R8,R7                   ; adjust the second longword
                     58   D4    1AA1 3$:        CLRL    R8                      ; clear the subtraction adjustment
         02    5A    1F   E0    1AA3            BBS     #31,R10,4$              ; is the correction divisor negative ?
                     58   D6    1AA7            INCL    R8                      ; yes - difference will need adjusting
               56    5A   C2    1AA9 4$:        SUBL2   R10,R6                  ; subtract the first longwords
               57    58   D9    1AAC            SBWC    R8,R7                   ; subtract the second longwords
```

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

```
1B20  ;    ****************************************************************
1B20  ;    *                                                              *
1B20  ;    *                                                              *
1B20  ;    *              Condition Code Processing Routines              *
1B20  ;    *                                                              *
1B20  ;    *                                                              *
1B20  ;    ****************************************************************
1B20  ;
1B20  ;
1B20  ;    Introduction
1B20  ;    ------------
1B20  ;
1B20  ;        In order that condition code information be usable
1B20  ;    directly by the code of the Emulator as well as be available
1B20  ;    for use in the emulated PSL, the routines which perform tests
1B20  ;    and compares set the hardware condition codes. The routine
1B20  ;    SET_CONDITION is used to move the hardware condition codes
1B20  ;    to the emulated PSL.
1B20  ;
1B20  ;        The routines of this portion of the Emulator are extremely
1B20  ;    simple so the descriptions of routines are included with the
1B20  ;    routines. Here we will just discuss the general policy on
1B20  ;    condition codes within the Emulator.
1B20  ;
1B20  ;
1B20  ;    General Policy on Condition Codes
1B20  ;    ---------------------------------
1B20  ;
1B20  ;        In general it is the responsibility of each of the
1B20  ;    instruction emulation routines to insure that the condition
1B20  ;    codes are correct. Since for most of the instructions
1B20  ;    presently emulated the condition codes reflect the floating
1B20  ;    value that appears in OPERAND1, the routine SET_CONDITION1
1B20  ;    is available for checking the value and setting the condition
1B20  ;    codes in the emulated PSL. In other cases these operations
1B20  ;    must be done in line.
1B20  ;
1B20  ;        For those instructions which return an integer result
1B20  ;    the V bit is used to indicate whether or not an integer
1B20  ;    overflow took place. Because of this it is also checked in
1B20  ;    NORMAL_EXIT in order to determine if a integer overflow trap
1B20  ;    should be signaled. For this reason it is cleared when the
1B20  ;    instruction emulation is started in EMULATOR.
1B20  ;
```

```
                     1B20             ;
                     1B20             ;       SET_CONDITION - Capture the Condition Codes N and Z in the PSL
                     1B20             ;
                     1B20             ;               entered by subroutine branching
                     1B20             ;
                     1B20             ;               no parameters
                     1B20             ;
                     1B20             ;       Discussion
                     1B20             ;
                     1B20             ;           This routine sets the N and Z bits in the emulated PSL
                     1B20             ;       equal to those bits in PSL available on entry to the routine.
                     1B20             ;
                     1B20  SET_CONDITION:                                  ; entrance
           0B  19    1B20             BLSS    1$                        ; less than - bypass
           12  14    1B22             BGTR    2$                        ; greater than - bypass
 54 AD     0C  CA    1B24             BICL2   #PSLM_NZ,PSL(FP)          ; clear the N bit and Z bit in the PSL
 54 AD     04  C8    1B28             BISL    #PSLM_EQL,PSL(FP)         ; specify equals in the PSL
               05    1B2C             RSB                               ; return
 54 AD     0C  CA    1B2D  1$:        BICL2   #PSLM_NZ,PSL(FP)          ; clear the N bit and Z bit in the PSL
 54 AD     08  C8    1B31             BISL    #PSLM_LSS,PSL(FP)         ; specify less than in the PSL
               05    1B35             RSB                               ; return
 54 AD     0C  CA    1B36  2$:        BICL2   #PSLM_NZ,PSL(FP)          ; clear the N bit and Z bit in the PSL
 54 AD     00  C8    1B3A             BISL2   #PSLM_GTR,PSL(FP)         ; specify greater than in the PSL
               05    1B3E             RSB                               ; return
                     1B3F             ;
                     1B3F             ;       SET_CONDITION1 - Test a Floating Value Setting the PSL
                     1B3F             ;
                     1B3F             ;               entered by subroutine branching
                     1B3F             ;
                     1B3F             ;               parameter:      OPERAND1 = The Floating Value
                     1B3F             ;
                     1B3F             ;       Discussion
                     1B3F             ;
                     1B3F             ;           This routine tests the floating value in OPERAND1 and
                     1B3F             ;       sets the N and Z bits in the emulated PSL according to the
                     1B3F             ;       outcome of the test. The V and C bits in the emulated PSL
                     1B3F             ;       are cleared.
                     1B3F             ;
                     1B3F  SET_CONDITION1:                                 ; entrance
 51    C7 AD   9E    1B3F             MOVAB   OPERAND1(FP),R1           ; R1 = location of OPERAND1
           07  10    1B43             BSBB    TEST_REAL                 ; test the value
           D9  10    1B45             BSBB    SET_CONDITION             ; set the condition codes
 54 AD     03  CA    1B47             BICL2   #PSLM_VC,PSL(FP)          ; clear the V bit and C bit in the PSL
               05    1B4B             RSB
                     1B4C             ;
```

```
                1B4C              ;
                1B4C              ;       TEST_REAL - Test all Floating Types
                1B4C              ;
                1B4C              ;               entered by subroutine branching
                1B4C              ;
                1B4C              ;               parameter:      R1 = Floating Value Location
                1B4C              ;
                1B4C              ;       Discussion
                1B4C              ;
                1B4C              ;           This routine tests the floating value in the internal
                1B4C              ;       representation that is addressed by R1 and sets the hardware
                1B4C              ;       condition codes accordingly. These settings are available
                1B4C              ;       when the routine returns.
                1B4C              ;
                1B4C      TEST_REAL:                                      ; entrance
     56 61   E8 1B4C              BLBS    ZERO(R1),TEST_EQL               ; value is zero - bypass
  55 01 A1   E9 1B4F              BLBC    SIGN(R1),TEST_GTR               ; value is positive - bypass
        4C   11 1B53              BRB     TEST_LSS                        ; value is negative - bypass
                1B55              ;
                1B55              ;       TEST_OCTA - Test an Octaword in OPERAND1
                1B55              ;
                1B55              ;               entered by subroutine branching
                1B55              ;
                1B55              ;               parameter:      OPERAND1 = 128 Bit Octaword Value
                1B55              ;
                1B55              ;       Discussion
                1B55              ;
                1B55              ;           This routine tests the octaword value which occupies
                1B55              ;       128 bits and starts at OPERAND1 (it is not in the internal
                1B55              ;       floating representation) and sets the hardware condition
                1B55              ;       codes according to the outcome of the test. These settings
                1B55              ;       are available when the routine returns.
                1B55              ;
                1B55      TEST_OCTA:                                      ; entrance
  50    03   D0 1B55              MOVL    #3,R0                           ; initialize the index
  C7 AD40    D5 1B58              TSTL    OPERAND1(FP)[R0]                ; test the first longword
        05   18 1B5C              BGEQ    2$                              ; it's not negative - bypass
             05 1B5E              RSB                                     ; return indicating negative value
  C7 AD40    D5 1B5F      1$:     TSTL    OPERAND1(FP)[R0]                ; test the next longword
        43   12 1B63      2$:     BNEQ    TEST_GTR                        ; value is positive - bypass
     F7 50   F4 1B65              SOBGEQ  R0,1$                           ; more longwords to examine - loop
        3B   11 1B68              BRB     TEST_EQL                        ; value is zero - bypass
                1B6A              ;
```

```
                         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           ;
             1BA1           ;       TEST_LSS - Set the Condition Codes to Specify Less Than
             1BA1           ;
             1BA1           ;               entered by subroutine branching
             1BA1           ;
             1BA1           ;               no parameters
             1BA1           ;
             1BA1           ;       Discussion
             1BA1           ;
             1BA1           ;          This routine sets the hardware condition codes to specify
             1BA1           ;       a "less than" outcome. This setting is available when the
             1BA1           ;       routine returns.
             1BA1           ;
             1BA1   TEST_LSS:                               ; entrance
 FF 8F   95  1BA1           TSTB    #-1                     ; set the condition codes
         05  1BA4           RSB                             ; return
             1BA5           ;
             1BA5           ;       TEST_EQL - Set the Condition Codes to Specify Equals
             1BA5           ;
             1BA5           ;               entered by subroutine branching
             1BA5           ;
             1BA5           ;               no parameters
             1BA5           ;
             1BA5           ;       Discussion
             1BA5           ;
             1BA5           ;          This routine sets the hardware condition codes to specify
             1BA5           ;       an "equals" outcome. This setting is available when the
             1BA5           ;       routine returns.
             1BA5           ;
             1BA5   TEST_EQL:                               ; entrance
    00   95  1BA5           TSTB    #0                      ; set the condition codes
         05  1BA7           RSB                             ; return
             1BA8           ;
             1BA8           ;       TEST_GTR - Set the Condition Codes to Specify Greater Than
             1BA8           ;
             1BA8           ;               entered by subroutine branching
             1BA8           ;
             1BA8           ;               no parameters
             1BA8           ;
             1BA8           ;       Discussion
             1BA8           ;
             1BA8           ;          This routine sets the hardware condition codes to specify
             1BA8           ;       a "greater than" outcome. This setting is available when the
             1BA8           ;       routine returns.
             1BA8           ;
             1BA8   TEST_GTR:                               ; entrance
    01   95  1BA8           TSTB    #1                      ; set the condition codes
         05  1BAA           RSB                             ; return
             1BAB           ;
```

```
1BAB    ;       ****************************************************************
1BAB    ;       *                                                              *
1BAB    ;       *                                                              *
1BAB    ;       *               Exception Processing Routir s                  *
1BAB    ;       *                                                              *
1BAB    ;       *                                                              *
1BAB    ;       ****************************************************************
1BAB    ;
1BAB    ;
1BAB    ;       Introduction
1BAB    ;       ------------
1BAB    ;
1BAB    ;           In order to simplify the design of the Emulator, it was
1BAB    ;       decided that whenever fault conditions were detected during
1BAB    ;       instruction emulation, that control would branch immediatly
1BAB    ;       to the signaling routine rather than using status codes to
1BAB    ;       inform some outer routine of the condition. Because of this
1BAB    ;       some care was necessary in the ordering of operations so that
1BAB    ;       the Emulator is always in the correct state when faults are
1BAB    ;       detected. The only trap supported is the integer overflow
1BAB    ;       trap and since this condition is only signaled when the
1BAB    ;       instruction emulation is complete, there is no problem with
1BAB    ;       the flow of control.
1BAB    ;
1BAB    ;           For each of the exceptions recognized, there is a routine
1BAB    ;       which is branched to (except for access violations in which
1BAB    ;       a subroutine branch is used instead) as soon as the condition
1BAB    ;       is detected. This routine pushes a shortened version of the
1BAB    ;       signal array onto the stack and branches to SIGNAL_START which
1BAB    ;       builds the signal and mechanism arrays in the proper place in
1BAB    ;       memory and enters the signal dispatcher to search for handlers
1BAB    ;       to process the condition. If the exception was a fault, the
1BAB    ;       routine FAULT_RESET is called to restore the registers to
1BAB    ;       their values when the instruction was started.
1BAB    ;
1BAB    ;
1BAB    ;       Access Violations
1BAB    ;       -----------------
1BAB    ;
1BAB    ;           The routines READ_FAULT and WRITE_FAULT are called by
1BAB    ;       subroutine branching when memory probes of read and write
1BAB    ;       access fail during instruction emulat'  . The register R11 is
1BAB    ;       assumed to contain the location of the area being probed
1BAB    ;       and the register R10 is assumed to contain its length. The
1BAB    ;       routine tries to produce the fault under controlled conditions
1BAB    ;       and returns if it can not produce the fault. If it can produce
1BAB    ;       the fault the the fault is signaled with the reason mask being
1BAB    ;       the reason mask from the attempt to produce the fault and with
1BAB    ;       the violation address as the address of the first byte of the
1BAB    ;       area for which the access violation occurs.
1BAB    ;
1BAB    ;
1BAB    ;       The Signal Dispatcher
1BAB    ;       ---------------------
1BAB    ;
1BAB    ;           The Emulator presently contains all of the code necessary
1BAB    ;       for signaling the condition since it is necessary that the
```

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

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

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

```
                   1C64           ;
                   1C64           ;       OPCODE_FAULT - Process an Opcode Reserved to Digital Fault
                   1C64           ;
                   1C64           ;               entered by branching
                   1C64           ;
                   1C64           ;               no parameters
                   1C64           ;
                   1C64  OPCODE_FAULT:                               ; entrance
        0071  30   1C64           BSBW    FAULT_RESET                 ; reinitialize registers and clear TP
     5E  FB AD  9E   1C67         MOVAB   SHORT_LOCAL(FP),SP          ; shorten the frame
  7E   043C 8F  3C   1C6B         MOVZWL  #SS$_OPCDEC,-(SP)           ; push the condition code
          01  DD   1C70           PUSHL   #1                          ; push the number of arguments
        0079  31   1C72           BRW     SIGNAL_START                ; signal the condition
                   1C75           ;
                   1C75           ;       ADDRESS_FAULT - Process an Invalid Addressing Mode Fault
                   1C75           ;
                   1C75           ;               entered by branching
                   1C75           ;
                   1C75           ;               no parameters
                   1C75           ;
                   1C75  ADDRESS_FAULT:                              ; entrance
        0060  30   1C75           BSBW    FAULT_RESET                 ; reinitialize registers and clear TP
     5E  FB AD  9E   1C78         MOVAB   SHORT_LOCAL(FP),SP          ; shorten the frame
  7E   044C 8F  3C   1C7C         MOVZWL  #SS$_RADRMOD,-(SP)          ; push the condition code
          01  DD   1C81           PUSHL   #1                          ; push the number of arguments
        0068  31   1C83           BRW     SIGNAL_START                ; signal the condition
                   1C86           ;
                   1C86           ;       OPERAND_FAULT - Processed a Reserved Operand Fault
                   1C86           ;
                   1C86           ;               entered by branching
                   1C86           ;
                   1C86           ;               no parameters
                   1C86           ;
                   1C86  OPERAND_FAULT:                              ; entrance
        004F  30   1C86           BSBW    FAULT_RESET                 ; reinitialize registers and clear TP
     5E  FB AD  9E   1C89         MOVAB   SHORT_LOCAL(FP),SP          ; shorten the frame
  7E   0454 8F  3C   1C8D         MOVZWL  #SS$_ROPRAND,-(SP)          ; push the condition code
          01  DD   1C92           PUSHL   #1                          ; push the number of arguments
        0057  31   1C94           BRW     SIGNAL_START                ; signal the condition
                   1C97           ;
                   1C97           ;       UNDERFLOW - Process a Floating Underflow Fault
                   1C97           ;
                   1C97           ;               entered by branching
                   1C97           ;
                   1C97           ;               no parameters
                   1C97           ;
                   1C97  UNDERFLOW:                                  ; entrance
        003E  30   1C97           BSBW    FAULT_RESET                 ; reinitialize registers and clear TP
     5E  FB AD  9E   1C9A         MOVAB   SHORT_LOCAL(FP),SP          ; shorten the frame
  7E   04C4 8F  3C   1C9E         MOVZWL  #SS$_FLTUND_F,-(SP)         ; push the condition code
          01  DD   1CA3           PUSHL   #1                          ; push the number of arguments
        0046  31   1CA5           BRW     SIGNAL_START                ; signal the condition
                   1CA8           ;
                   1CA8           ;       OVERFLOW - Process a Floating Overflow Fault
                   1CA8           ;
                   1CA8           ;               entered by branching
                   1CA8           ;
```

FP
Ta

```
                        1CA8        ;               no parameters
                        1CA8        ;
                        1CA8   OVERFLOW:                                      ; entrance
          002D   30     1CA8             BSBW    FAULT_RESET                  ; reinitialize registers and clear TP
     5E  FB AD   9E     1CAB             MOVAB   SHORT_LOCAL(FP),SP           ; shorten the frame
  7E  04B4 8F    3C     1CAF             MOVZWL  #SS$_FLTOVF_F,-(SP)          ; push the condition code
            01   DD     1CB4             PUSHL   #1                           ; push the number of arguments
          0035   31     1CB6             BRW     SIGNAL_START                 ; signal the condition
                        1CB9        ;
                        1CB9        ;       DIVIDE_FAULT - Process a Floating Divide by Zero Fault
                        1CB9        ;
                        1CB9        ;               entered by branching
                        1CB9        ;
                        1CB9        ;               no parameters
                        1CB9        ;
                        1CB9   DIVIDE_FAULT:                                  ; entrance
          001C   30     1CB9             BSBW    FAULT_RESET                  ; reinitialize registers and clear TP
     5E  FB AD   9E     1CBC             MOVAB   SHORT_LOCAL(FP),SP           ; shorten the frame
  7E  04BC 8F    3C     1CC0             MOVZWL  #SS$_FLTDIV_F,-(SP)          ; push the condition code
            01   DD     1CC5             PUSHL   #1                           ; push the number of arguments
          0024   31     1CC7             BRW     SIGNAL_START                 ; signal the condition
                        1CCA        ;
                        1CCA        ;       INT_OVERFLOW - Process an Integer Overflow Trap
                        1CCA        ;
                        1CCA        ;               entered by branching
                        1CCA        ;
                        1CCA        ;               no parameters
                        1CCA        ;
                        1CCA   INT_OVERFLOW:                                  ; entrance
     5E  FB AD   9E     1CCA             MOVAB   SHORT_LOCAL(FP),SP           ; shorten the frame
  7E  047C 8F    3C     1CCE             MOVZWL  #SS$_INTOVF,-(SP)            ; push the condition code
            01   DD     1CD3             PUSHL   #1                           ; push the number of arguments
          0016   31     1CD5             BRW     SIGNAL_START                 ; signal the condition
                        1CD8        ;
```

```
                            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          ;
```

```
                           1CEE         ;
                           1CEE         ;       SIGNAL_START - Build the Parameter Blocks for SIGNAL
                           1CEE         ;
                           1CEE         ;               entered by branching
                           1CEE         ;
                           1CEE         ;               parameters:     (SP) = Truncated Signal Array Size (M)
                           1CEE         ;                               4(SP) = Condition Code
                           1CEE         ;                               8(SP) = First Signal Argument
                           1CEE         ;                                 .
                           1CEE         ;                                 .
                           1CEE         ;                                 .
                           1CEE         ;                               4*<M-1>(SP) = Last Signal Argument
                           1CEE         ;
                           1CEE         ;       Discussion
                           1CEE         ;
                           1CEE         ;           This routine builds the signal and mechanism arrays
                           1CEE         ;       for a condition generated by the Emulator. It is entered
                           1CEE         ;       with the signal array for the condition except for the
                           1CEE         ;       PC and PSL pair pushed onto the Emulator's stack (with the
                           1CEE         ;       pushed array length correspondingly shortened). The signal
                           1CEE         ;       array, mechanism array, and the handler parameter block
                           1CEE         ;       are then constructed on the user's emulated stack. The routine
                           1CEF         ;       then removes the Emulator frame from the stack and enters
                           1CEE         ;       the signal dispatching loop at SIGNAL.
                           1CEE         ;
                           1CEE         ;       Notes:  1. The precise format of the information pushed onto
                           1CEE         ;                  the user's stack is given in the description of
                           1CEE         ;                  SIGNAL below.
                           1CEE         ;
                           1CEE         ;               2. The method of getting out of the Emulator used in
                           1CEE         ;                  this routine is essentially the same as that used
                           1CEE         ;                  in NORMAL_EXIT.
                           1CEE         ;
                           1CEE SIGNAL_START:                                   ; entrance
           57    8E   D0   1CEE         MOVL    (SP)+,R7                        ; R7 = number of signal parameters
      58   57    02   78   1CF1         ASHL    #2,R7,R8                        ; R8 = size of the signal parameters
      50   58    34   C1   1CF5         ADDL3   #52,R8,R0                       ; R0 = size of signal information
               E5DE   30   1CF9         BSBW    TEST_FRAME                      ; make sure we have room for it
      56    4C   AD   D0   1CFC         MOVL    REG_SP(FP),R6                   ; R6 = user's stack pointer
      76    50   AD   7D   1D00         MOVQ    REG_PC(FP),-(R6)                ; push the PC, PSL pair
           56    58   C2   1D04         SUBL2   R8,R6                           ; make room for the signal parameters
      66   6E    58   28   1D07         MOVC3   R8,(SP),(R6)                    ; push the signal parameters
      76   57    02   C1   1D0B         ADDL3   #2,R7,-(R6)                     ; push the signal array length
           76    01   D0   1D0F         MOVL    #1,-(R6)                        ; push code for SIGNAL (vs. STOP)
      76    14   AD   7D   1D12         MOVQ    REG_R0(FP),-(R6)                ; push user's R0 and R1
           76    03   CE   1D16         MNEGL   #3,-(R6)                        ; push -3 (depth number)
      76    48   AD   D0   1D19         MOVL    REG_FP(FP),-(R6)                ; push the user's FP
           76    04   D0   1D1D         MOVL    #4,-(R6)                        ; push the mechanism array length
           76    56   D0   1D20         MOVL    R6,-(R6)                        ; push the mechanism array location
      76    1C   A6   9E   1D23         MOVAB   28(R6),-(R6)                    ; push the signal array location
           76    02   D0   1D27         MOVL    #2,-(R6)                        ; push the handler parameter count
   08 AD    44   AD   7D   1D2A         MOVQ    REG_AP(FP),SAVE_AP(FP)          ; put the user's PC, PSL pair back
   10 AD    50'  AF   9E   1D2F         MOVAB   B^SIGNAL,SAVE_PC(FP)            ; store the return point
      50    48   AD   9E   1D34         MOVAB   FRAME_END+4(FP),R0              ; R0 = location of end of frame
           56    50   C2   1D38         SUBL2   R0,R6                           ; R6 = distance of user SP from it
   51   56   02   00   EF   1D3B         EXTZV   #0,#2,R6,R1                     ; R1 = stack alignment
06 AD   02   0E   51   F0   1D40         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          ;
                      1D50          ;       SIGNAL - Signal the Condition
                      1D50          ;
                      1D50          ;               entered by branching
                      1D50          ;
                      1D50          ;               parameters:     ( Described in Note 3 )
                      1D50          ;
                      1D50          ;       Discussion
                      1D50          ;
                      1D50          ;               Following is a description of the information which
                      1D50          ;               is assumed to be pushed onto the stack when the
                      1D50          ;               routine SIGNAL is entered. The values are all
                      1D50          ;               longwords.
                      1D50          ;
                      1D50          ;               Handler Parameter Block:
                      1D50          ;
                      1D50          ;                       (SP)    2 (handler parameter block length)
                      1D50          ;                       4(SP)   signal array location
                      1D50          ;                       8(SP)   mechanism ar~ay location
                      1D50          ;
                      1D50          ;               Mechanism Array:
                      1D50          ;
                      1D50          ;                       12(SP)  4 (mechanism array length)
                      1D50          ;                       16(SP)  user's FP (establisher frame)
                      1D50          ;                       20(SP)  -3 (establisher depth)
                      1D50          ;                       24(SP)  user's R0
                      1D50          ;                       28(SP)  user's R1
                      1D50          ;
                      1D50          ;               Information Not Part of any Array:
                      1D50          ;
                      1D50          ;                       32(SP)  1 (code for SIGNAL rather than STOP)
                      1D50          ;
                      1D50          ;               Signal Array:
                      1D50          ;
                      1D50          ;                       36(SP)  signal array length M
                      1D50          ;                       40(SP)  condition code
                      1D50          ;                       44(SP)  first signal argument
                      1D50          ;                         .
                      1D50          ;                         .
                      1D50          ;                         .
                      1D50          ;                       <4*M>+28(SP) last signal argument
                      1D50          ;                       <4*M>+32(SP) user's PC
                      1D50          ;                       <4*M>+36(SP) user's PSL
                      1D50          ;
                      1D50          ;               The user's stack pointer should coincide with the
                      1D50          ;               address <4*M>+40(SP).
                      1D50          ;
                      1D50          ;       We now jump to the special entry point in VMS to start
                      1D50          ;       the search for handlers.  Execution will not return to us.
                      1D50          ;-
                      1D50
                      1D50  SIGNAL:                                 ; entrance
00000000'GF     17    1D50          JMP     G^EXE$SRCHANDLER
                      1D56
```

```
1D56  : ***********************************************************************
1D56  : *                                                                     *
1D56  : *                                                                     *
1D56  : *          End of EMULATE$ VAX Instruction Subset Emulator            *
1D56  : *                                                                     *
1D56  : *                                                                     *
1D56  : ***********************************************************************
1D56
1D56          .END
```

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

INST_CVTGL		00000596	R	03
INST_CVTGW		00000576	R	03
INST_CVTHB		0000055E	R	03
INST_CVTHD		00000620	R	03
INST_CVTHF		00000618	R	03
INST_CVTHG		00000628	R	03
INST_CVTHL		0000059E	R	03
INST_CVTHW		0000057E	R	03
INST_CVTLD		00000510	R	03
INST_CVTLF		00000508	R	03
INST_CVTLG		00000518	R	03
INST_CVTLH		00000520	R	03
INST_CVTLX		00000526	R	03
INST_CVTRDL		00000656	R	03
INST_CVTRFL		0000064E	R	03
INST_CVTRGL		0000065E	R	03
INST_CVTRHL		00000666	R	03
INST_CVTRXL		0000066C	R	03
INST_CVTWD		000004F0	R	03
INST_CVTWF		000004E8	R	03
INST_CVTWG		000004F8	R	03
INST_CVTWH		00000500	R	03
INST_CVTWX		00000526	R	03
INST_CVTXB		000005A6	R	03
INST_CVTXL		000005A6	R	03
INST_CVTXW		000005A6	R	03
INST_CVTXY		0000062E	R	03
INST_DIVD2		00000693	R	03
INST_DIVD3		000006CC	R	03
INST_DIVF2		0000068D	R	03
INST_DIVF3		000006C6	R	03
INST_DIVG2		00000699	R	03
INST_DIVG3		000006D2	R	03
INST_DIVH2		0000069F	R	03
INST_DIVH3		000006D8	R	03
INST_DIVX2		000006A3	R	03
INST_DIVX3		000006DC	R	03
INST_EMODD		000007B0	R	03
INST_EMODF		00000702	R	03
INST_EMODG		000007CE	R	03
INST_EMODH		00000882	R	03
INST_MNEGD		00000945	R	03
INST_MNEGF		0000093F	R	03
INST_MNEGG		0000094B	R	03
INST_MNEGH		00000951	R	03
INST_MNEGX		00000955	R	03
INST_MOVAO		00000975	R	03
INST_MOVD		0000099C	R	03
INST_MOVF		00000996	R	03
INST_MOVG		000009A2	R	03
INST_MOVH		000009A8	R	03
INST_MOVO		000009D4	R	03
INST_MOVX		000009AC	R	03
INST_MULD2		00000A09	R	03
INST_MULD3		00000A42	R	03
INST_MULF2		00000A03	R	03
INST_MULF3		00000A3C	R	03
INST_MULG2		00000A0F	R	03
INST_MULG3		00000A48	R	03
INST_MULH2		00000A15	R	03
INST_MULH3		00000A4E	R	03
INST_MULX2		00000A19	R	03
INST_MULX3		00000A52	R	03
INST_POLYD		00000B7C	R	03
INST_POLYDG		00000B8E	R	03
INST_POLYF		00000A78	R	03
INST_POLYG		00000B86	R	03
INST_POLYH		00000C95	R	03
INST_PUSHAO		00000DE8	R	03
INST_SUBD2		00000E21	R	03
INST_SUBD3		00000E5D	R	03
INST_SUBF2		00000E1B	R	03
INST_SUBF3		00000E57	R	03
INST_SUBG2		00000E27	R	03
INST_SUBG3		00000E63	R	03
INST_SUBH2		00000E2D	R	03
INST_SUBH3		00000E69	R	03
INST_SUBX2		00000E31	R	03
INST_SUBX3		00000E6D	R	03
INST_TSTD		00000E9C	R	03
INST_TSTF		00000E96	R	03
INST_TSTG		00000EA2	R	03
INST_TSTH		00000EA8	R	03
INST_TSTX		00000EAC	R	03
INST_TYPES_1BYTE	=	00000277	R	03
INST_TYPES_2BYTE	=	0000029A	R	03
INT_OVERFLOW		00001CCA	R	03
IT_D	=	00000001		
IT_F	=	00000002		
IT_G	=	00000004		
IT_H	=	00000008		
IT_X	=	00000000		
LENGTHS		00000EE7	R	03
LITERAL_MODE		00000F40	R	03
LOCAL_END	=	00000058		
LOCAL_START	=	FFFFFF8B		
LOCAL_TEST		0000120D	R	03
LONG_DEF_MODE		0000115D	R	03
LONG_DISP_MODE		00001136	R	03
LO_1BYTE	=	00000040		
LO_2BYTE	=	00000032		
MASK_ALIGN	=	0000000E		
MODE	=	FFFFFFFD		
MODIFY_ACCESS		00000ED5	R	03
MODIFY_CHECK		0000119F	R	03
MULTIPLY		00001A05	R	03
MULTIPLY_DGFLOAT		0000179F	R	03
MULTIPLY_FFLOAT		0000174C	R	03
MULTIPLY_FLOAT		0000173F	R	03
MULTIPLY_HFLOAT		000017EF	R	03
NEGATE_REAL		000014E3	R	03
NORMALIZE		0000197F	R	03
NORMAL_EXIT		000001C4	R	03
N_ARGS	=	00000001		

```
ONES$                           00001A77 R      03      PSL_T                           = 00000004
OPCODE_FAULT                    00001C64 R      03      PSL_TP                          = 0000001E
OPERAND1                      = FFFFFFC7                PSL_V                           = 00000001
OPERAND2                      = FFFFFFAF                PSL_Z                           = 00000002
OPERAND3                      = FFFFFF97                READ_ACCESS                       00000EC3 R      03
OPERAND_FAULT                   00001C86 R      03      READ_CHECK                        000011A3 R      03
OPERAND_SIZE                  = 00000018                READ_FAULT                        00001BAB R      03
OP_INDEX                      = FFFFFF8B                READ_REASON                       00001BE6 R      03
OP_MASK_1BYTE                   000000A5 R      02      READ_VALUE                        000011C2 R      03
OP_MASK_2BYTE                   000000C5 R      02      REASON_HANDLER                    00001BF2 R      03
OP_TYPES                      = FFFFFF8F                REGISTER_MODE                     00000FED R      03
OP_TYPES4                     = FFFFFF93                REGMOD_AP                       = FFFFFFF7
OVERFLOW                        00001CA8 R      03      REGMOD_FP                       = FFFFFFF8
PACK_DFLOAT                     00001382 R      03      REGMOD_PC                       = FFFFFFFA
PACK_FFLOAT                     00001331 R      03      REGMOD_R0                       = FFFFFFEB
PACK_FLOAT                      00001324 R      03      REGMOD_R1                       = FFFFFFEC
PACK_FLOAT1                     00001314 R      03      REGMOD_R10                      = FFFFFFF5
PACK_FLOAT2                     0000131A R      03      REGMOD_R11                      = FFFFFFF6
PACK_FLOAT3                     00001320 R      03      REGMOD_R2                       = FFFFFFED
PACK_GFLOAT                     000013DC R      03      REGMOD_R3                       = FFFFFFEE
PACK_HFLOAT                     0000143B R      03      REGMOD_R4                       = FFFFFFEF
POWER                         = 00000004                REGMOD_R5                       = FFFFFFF0
POWER1                        = FFFFFFCB                REGMOD_R6                       = FFFFFFF1
POWER2                        = FFFFFFB3                REGMOD_R7                       = FFFFFFF2
POWER3                        = FFFFFF9B                REGMOD_R8                       = FFFFFFF3
PR$_ESP                       = 00000001                REGMOD_R9                       = FFFFFFF4
PR$_SSP                       = 00000002                REGMOD_SP                       = FFFFFFF9
PR$_USP                       = 00000003                REG_AP                          = 00000044
PSL                           = 00000054                REG_DEF_MODE                      0000100F R      03
PSL$C_EXEC                    = 00000001                REG_FP                          = 00000048
PSL$C_KERNEL                  = 00000000                REG_PC                          = 00000050
PSL$C_SUPER                   = 00000002                REG_R0                          = 00000014
PSL$C_USER                    = 00000003                REG_R1                          = 00000018
PSL$M_FPD                     = 08000000                REG_R10                         = 0000003C
PSL$M_TBIT                    = 00000010                REG_R11                         = 00000040
PSL$M_TP                      = 40000000                REG_R2                          = 0000001C
PSL$S_CURMOD                  = 00000002                REG_R3                          = 00000020
PSL$S_PRVMOD                  = 00000002                REG_R4                          = 00000024
PSL$V_CURMOD                  = 00000018                REG_R5                          = 00000028
PSL$V_PRVMOD                  = 00000016                REG_R6                          = 0000002C
PSLM_C                        = 00000001                REG_R7                          = 00000030
PSLM_EQL                      = 00000004                REG_R8                          = 00000034
PSLM_GTR                      - 00000000                REG_R9                          = 00000038
PSLM_LSS                        00000008                REG_SP                          = 0000004C
PSLM_N                          00000008                ROUND_REAL                        00001578 R      03
PSLM_NZ                         0000000C                SAVE_ALIGN                      = FFFFFFFF
PSLM_NZV                        0000000E                SAVE_AP                         = 00000008
PSLM_NZVC                       0000000F                SAVE_FP                         = 0000000C
PSLM_V                        - 00000002                SAVE_MASK                       = 00000006
PSLM_VC                       - 00000003                SAVE_PARCNT                     = FFFFFFFE
PSLM_Z                          00000004                SAVE_PC                         = 00000010
PSL_C                           00000000                SAVE_PSW                        = 00000004
PSL_CAM                         00000018                SET_CONDITION                     00001B20 R      03
PSL_FPD                       = 0000001B                SET_CONDITION1                    00001B3F R      03
PSL_FU                        = 00000006                SHORT_LOCAL                     = FFFFFFFB
PSL_IV                        = 00000005                SIGN                            = 00000001
PSL_N                         = 00000003                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
------------------                           --------------
_$255$DUA28:[SYS.OBJ]LIB.MLB;1                          3
_$255$DUA28:[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

0143 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

FPLOAD
LIS

VAXASHP
LIS

VAXCONVRT
LIS

VAXARITH
LIS