CHAPTER 6

FLOATING POINT PACKAGE

The PDOS floating point package is a single accumulator, IBM format, multi-user floating point processor. It includes all the necessary routines to write assembly language floating point software, including addition, subtraction, multiplication, division, load, store, scale clear, float, normalize, negate, absolute value, multiplicative inverse, status, clock, and error handling. Input and output routines are also described in this chapter.

Single accumulator, IBM format

6.1 FLOATING POINT FORMAT

The PDOS floating point package is a single accumulator, IBM format, multi-user floating point processor. The IBM format consists of a sign bit, 7 bits of exponent or characteristic (excess 64), and 40 bits of fraction or mantissa. The resultant number is produced by taking 16 raised to the exponent, times the mantissa. This gives numbers in the range of 1E-79 to 1E75. Zero is represented by all 6 bytes being zero rather than just a zero mantissa.

Seeeeeeeecccccccc cccccccccccccccc cccccccccccccccc

>7FFF >FFFF >FFFF = 7.23700557730E75
>0010 >0000 >0000 = 5.39760534693E-79

True zero

All floating point numbers must be normalized for the floating point operations to work correctly. This means that the first hex digit of the mantissa must be nonzero. All floating point routines, with the exception of scale, return normalized numbers.

Normalization

The floating point processor is accessed via eight XOP vectors. Interrupts are disabled during all floating point operations. The Floating Point Accumulator (referred to as FPAC) is swapped in and out with the task, thus making the routines accessible to other tasks.

8 XOP vectors
Interrupts disabled

These XOP vectors are defined as follows:

```
DXOP LOADF,0    ;LOAD FPAC
DXOP STORE,1    ;STORE FPAC
DXOP FADD,2     ;ADD TO FPAC
DXOP FSUB,3     ;SUBTRACT FROM FPAC
DXOP FMUL,4     ;MULTIPLY FPAC
DXOP FDIV,5     ;DIVIDE FPAC
DXOP SCALE,6    ;SCALE FPAC
DXOP FXOPS,7    ;FP MISCELLANEOUS COMMANDS
```

6.2 FLOATING POINT COMMANDS

6.2.1 LOADF - LOAD FPAC

Format: LOADF <general address>

The LOAD FPAC routine loads the floating point accumulator with the six bytes pointed to by <general address>. No error checking is done by this operation.

```
LOAD1   LI RO,>4110       ;GET FP1
        CLR R1
        CLR R2
        LOADF RO          ;LOAD FPAC
        ....
```

6.2.2 STORE - STORE FPAC

Format: STORE <general address>

The STORE FPAC routine stores into user memory the six byte floating point accumulator. The address at which FPAC is stored is specified by <general address>.

```
        FMUL @FP10        ;MULTIPLY BY 10
        STORE @TEMP       ;SAVE IN TEMP
        ....

FP10    DATA >41A0,>0000,>0000
TEMP    BSS 6
```

6.2.3 FADD - ADD TO FPAC

Format: FADD <general address>

The ADD TO FPAC routine adds a six byte floating point number, pointed to by <general address>, to the contents of the floating point accumulator. Both the number and FPAC must be normalized floating point numbers.

The numbers are first shifted so that the exponents agree. Then the fractional parts are converted to 2's complement, 6 byte fractions and added together. Finally, the result is converted back to a 1's complement number, the corrected exponent and sign bit added, and the number is then normalized again.

```
INCRM   MPY @C6,R1        ;GET CORRECT INDEX
        FADD @TAB(2)      ;ADD CONSTANT
        STORE RO          ;RETRIEVE #
        ....

C6      DATA 6
TAB     DATA >4110,>0000,>0000
        DATA >4120,>0000,>0000
        DATA >4130,>0000,>0000
        DATA >4150,>0000,>0000
        DATA >4180,>0000,>0000
```

6.2.4 FSUB - SUBTRACT FROM FPAC

Format: FSUB <general address>

The SUBTRACT FROM FPAC routine subtracts a six byte floating point number pointed to by <general address> from the contents of the floating point accumulator. Both numbers need to be normalized floating point numbers.

The sign of the operand is toggled and then the two numbers are added. This is done by shifting the fractional parts until the exponents agree. Then the fractional parts are converted to 2's complement, 6 byte fractions and added together. Finally, the result is converted back to a 1's complement number, the corrected exponent and sign bit added, and the number is then normalized again.

```
        LOADF @B        ;A=B-C
        FSUB @C         ;SUBTRACT C
        STORE @A        ;STORE
        ....

A       BSS 6
B       DATA >4210,>0000,>0000
C       DATA >C120,>0000,>0000
```

6.2.5 FMUL - MULTIPLY FPAC

Format: FMUL <general address>

The MULTIPLY FPAC routine multiplies the contents of the floating point accumulator by the 6 byte number pointed to by <general address>. The product is obtained by adding exponents and doing a three word unsigned multiply. The product is then normalized.

```
        LOADF @A        ;A=A*10
        FMUL @FP10
        STORE @A
        ....

A       BSS 6
FP10    DATA >41A0,>0000,>0000
```

6.2.6 FDIV - DIVIDE FPAC

Format: FDIV <general address>

The DIVIDE FPAC routine divides the contents of the floating point accumulator by the 6 byte number pointed to by <general address>. The quotient is obtained by subtracting exponents and doing a three word unsigned divide. The quotient is then normalized.

```
        LOADF @A        ;A=A/10+5
        FDIV @FP10
        FADD @FP5
        STORE @A
        ....

A       BSS 6
FP5     DATA >4150,>0000,>0000
FP10    DATA >41A0,>0000,>0000
```

6.2.7 SCALE - SCALE FPAC

Format: SCALE <general address>

The SCALE FPAC routine adjusts the floating point accumulator so that the exponent matches the left byte of the word pointed to by <general address>. If the exponent of FPAC is greater than the scale exponent, a floating point error occurs.

The SCALE FPAC routine is useful in changing floating point to fixed point. With a normalized floating point number, the mantissa is a positive fraction less than 1. By scaling FPAC to a known exponent, the decimal point is set anywhere within the number.

All of the following floating point numbers are equivalent to the number 1, although not necessarily normalized:

```
>4110 >0000 >0000
>4201 >0000 >0000
>4300 >1000 >0000
>4400 >0100 >0000
>4500 >0010 >0000
>4600 >0001 >0000
>4700 >0000 >1000
>4800 >0000 >0100
>4900 >0000 >0010
>4A00 >0000 >0001
```

Notice that when scaling to exponent >4A, the number becomes an integer as the fractional part is lost to the right.

```
* RETURN 16-BIT 2'S COMPLEMENT INTEGER
*
FIX     LOADF *R2       ;LOAD FPAC
        SCALE @H4600    ;SCALE
        STORE R0        ;GET RESULT
        SLA R0,1        ;NEGATIVE?
         JNC FIX2       ;N
        NEG R1          ;Y, NEGATE #
*
FIX2    RT              ;R1=INTEGER PART
*
H4600   DATA >4600      ;SCALE FACTOR
```

6.2.8 FXOPS 0 - CLEAR FPAC

Format: FXOPS 0

The CLEAR FPAC routine sets the floating point accumulator to all zeros.

```
CLRFP   FXOPS 0         ;CLEAR FPAC
```

6.2.9 FXOPS 1 - FLOAT FPAC

Format: FXOPS 1

The FLOAT FPAC routine converts a 2's complement, 16-bit integer to a 48-bit floating point number. The first word of FPAC must be zero; the second word is loaded with the 16-bit number.

```
FLOAT   CLR RO          ;CLEAR HIGH WORD
        MOV @NUM,R1     ;GET NUMBER
        LOADF RO        ;LOAD FPAC
        FXOPS 1         ;FLOAT
        STORE @FPNUM    ;STORE
        ....

NUM     DATA 100
FPNUM   BSS 6           ;FLOATING POINT RESULT
```

6.2.10 FXOPS 2 - NORMALIZE FPAC

Format: FXOPS 2

The NORMALIZE FPAC routine shifts the fractional part of FPAC left and decrements the exponent until the first hex digit of the fraction is nonzero. This constitutes a normalized floating point number.

```
INTFP   LOADF @NUM      ;LOAD NUMBER
        SCALE @H4A00    ;REMOVE ANY FRACTION
        FXOPS 2         ;NORMALIZE AGAIN
        ....

NUM     BSS 6
H4A00   DATA >4A00
```

6.2.11 FXOPS 3 - NEGATE FPAC

Format: FXOPS 3

If FPAC is nonzero, NEGATE FPAC toggles the sign bit.

```
* FRACTION = NUM - INT[NUM]
*
FRAF    LOADF @NUM      ;LOAD NUMBER
        SCALE @H4A00    ;REMOVE FRACTION
        FXOPS 2         ;NORMALIZE AGAIN
        FXOPS 3         ;NEGATE
        FADD @NUM       ;ADD NUMBER
        ....

NUM     BSS 6
H4A00   DATA >4A00
```

6.2.12 FXOPS 4 - ABSOLUTE VALUE

Format: FXOPS 4

The ABSOLUTE VALUE function takes the absolute value of FPAC. If FPAC is negative (sign bit=1), then FPAC is negated.

```
        LOADF @NUM      ;LOAD NUMBER
        FXOPS 4         ;ABSOLUTE VALUE
        SCALE @H4600    ;SCALE
        STORE R0        ;GET #
        MOV R1,@FNUM    ;SAVE
        ....

NUM     BSS 6
FNUM    BSS 2

H4600   DATA >6400
```

6.2.13 FXOPS 5 - READ FPAC STATUS

Format: FXOPS 5

OUT: (R2) = FPAC
Status = LT, EQ, GT

The READ STATUS routine returns in the user status register the sign of FPAC. An EQUAL status is returned if FPAC is zero, GREATER THAN if FPAC is positive, and LESS THAN if FPAC is negative. Register R2 is returned with the address of FPAC.

```
        LOADF @FA       ;IF A<B: GOTO 100
        FSUB @FB
        FXOPS 5         ;A<B?
          JLT L100      ;Y
        ....

L100    ....

FA      BSS 6
FB      BSS 6
```

6.2.14 FXOPS 6 - READ CLOCK TICS

Format: FXOPS 6

The READ CLOCK TICS routine loads FPAC with the 2 word tic counter and converts it to a floating point number. The tic counter is incremented every 1/125th of a second.

```
        FXOPS 6         ;READ CLOCK TICS
        STORE @TEMP     ;SAVE
        ....

        FXOPS 6         ;READ CLOCK AGAIN
        FSUB @TEMP      ;GET ELAPSED TIME
        ....

TEMP    BSS 6
```

6.2.15 FXOPS 7 - INVERSE OF FPAC

Format: FXOPS 7

The INVERSE OF FPAC routine takes the multiplicative inverse of FPAC. This is equivalent to dividing one by FPAC and putting the result back in FPAC.

```
        LOADF @NUM      ;LOAD NUM
        FXOPS 7         ;TAKE INVERSE
        ....
```

6.2.16 FXOPS 8 - LOAD ERROR RETURN ADDRESS

Format: FXOPS 8

IN: R0 = Error trap address

The SET ERROR RETURN ADDRESS routine sets the error trap address for all floating point errors. This is initially set to zero, which causes the floating point processor to ignore errors. The error address is passed in register R0. If an error occurs during a floating point operation, control is passed to the error trap address.

The error trap address is swapped with the task and thus each task has its own error trap routine.

```
        LI R0,ERTRP     ;GET ERROR TRAP ADDRESS
        FXOPS 8         ;SET IN FP PROCESSOR
        FMUL @FPN       ;CONTINUE
        ....

ERTRP   XPMC            ;FP ERROR
          DATA ERM1
        MOV R0,R1
        XCBD            ;CONVERT
        XPLC            ;PRINT
        ....

FPN     BSS 6
ERM1    BYTE >0A,>0D
        TEXT 'FLOATING POINT ERROR='
        BYTE 0
        EVEN
```

6.3 CONVERT DECIMAL TO FLOATING POINT

```
Module: FPINP:OBJ

Format: BLWP @FPINZ
          JL = No number
          JH = Number
          JEQ = Number w/o null delimiter

Registers:  IN R1  = Pointer to string

           OUT R0  = Delimiter
              (R2) = Updated pointer
              FPAC = Number
```

Included with a PDOS system is the object file 'FPINP:OBJ'. This relocatable module is linked with your floating point routines and used to convert an ASCII string of characters to a floating point number. The converted number is returned in the floating point accumulator.

The entry vector is the external definition (DEF) label 'FPINZ'. Register R1 passes the address of the ASCII string to the module. Register R0 is returned with the number delimiter and Register R1 is updated.

The status register reflects the success of the conversion. If it is low, then no number conversion was possible. If it is equal, than a number was converted to floating point but was not terminated with a null. The offending character is returned in register R0. If it is high, then a successful conversion was completed and register R0 is returned with a zero.

The module is called via a 'BLWP @FPINZ'. An internal workspace is defined as a 32 byte data section (DSEG) area. The following is an example using the program created at the right:

```
.TEMP2
ENTER NUMBER:100
BINARY=426400000000
ENTER NUMBER:3.1415926
BINARY=413243F69A25
ENTER NUMBER:1.23E10
BINARY=492DD231B000
ENTER NUMBER:123AC
CONVERSION ERROR!
ENTER NUMBER:
```

```
.SF TEMP
        REF FPINZ       ;DEFINE ENTRY
        DXOP FXOPS,7    ;MISCELLANEOUS
*
START   XPMC            ;OUTPUT PROMPT
          DATA MES01
        XGLU            ;GET LINE
        BLWP @FPINZ     ;CONVERT TO FP
          JH NUMBOK     ;OK
        XPMC            ;ERROR
          DATA MES02
        JMP START       ;TRY AGAIN
*
NUMBOK  XPMC            ;OUTPUT 'BINARY='
          DATA MES03
        FXOPS 5         ;GET FPAC ADDRESS
        MOV *R2+,R1
        XCBH            ;CONVERT TO HEX
        XPLC            ;OUTPUT 1ST WORD
        MOV *R2+,R1
        XCBH
        XPLC            ;OUTPUT 2ND WORD
        MOV *R2,R1
        XCBH
        XPLC            ;OUTPUT 3RD WORD
        JMP START
*
MES01   BYTE >0A,>0D
        TEXT 'ENTER NUMBER:'
        BYTE 0
MES02   BYTE >0A,>0D
        TEXT 'CONVERSION ERROR!'
        BYTE 0
MES03   BYTE >0A,>0D
        TEXT 'BINARY='
        BYTE 0
        END START
.CT (ASM TEMP,TEMP1),10
.LINK
LINKER R2.4
*12,2
WAS >0000
*0,TEMP2
*1,TEMP1
*1,FPINP:OBJ
*6
START TAG = >0000
*7
.
```

6.4 CONVERT FLOATING POINT TO DECIMAL

Module: FPOUT:OBJ

Format: BLWP @FPOINZ

```
Registers:  IN R0,R1 = 32 bit 2's complement number
               (R2) = Output mask  (0=format free)

           OUT (R2) = ASCII converted string
```

Format: BLWP @FPOFPZ

```
Registers:  IN (R0) = 48 bit floating point number
               (R2) = Output mask  (0=format free)

           OUT (R2) = ASCII converted string
```

A floating point number or a two word 2's complement fixed point number is converted to an ASCII string by the relocatable object module 'FPOUT:OBJ'. The output is format free or formatted, according to a conversion mask.

The relocatable module has two entry vectors 'FPOINZ' and 'FPOFPZ'. An 80 byte data segment (DSEG) workspace area holds the internal registers and character buffer.

A two word 2's complement number in registers R0 and R1 is converted to an ASCII string by the entry vector 'FPOINZ'. If register R2 is zero, then a format free string pointer is returned in R2. If R2 is nonzero, then the conversion mask pointed to by R2 is used in formatting the number and R2 is returned with a pointer to the string.

A three word floating point number pointed to by register R0 is converted to an ASCII string by the entry vector 'FPOFPZ'. If R2 is nonzero, then the conversion mask pointed to by R2 is used in formatting the number. Otherwise, a format free conversion is done. In either case, register R2 is returned with a pointer to the converted string.

Formatting allows numbers to be right justified, have a floating sign, dollar sign, or angle brackets, or commas and periods inserted. Numbers are rounded on the last converted digit.

```
.SF TEMP
*       FPOFP EXAMPLE
        REF FPOFPZ
        DXOP LOADF,0     ;LOAD FPAC
        DXOP FMUL,4      ;MULTIPLE #
        DXOP FXOPS,7     ;MISCELLANEOUS
*
START   LOADF @FP1       ;LOAD 4.0
        FMUL @FP2        ;X ATN 1
        FXOPS 5          ;GET ADDRESS
        MOV R2,R0        ;(R0)=RESULT
        LI R2,MASK       ;POINT TO MASK
        BLWP @FPOFPZ     ;CONVERT
        MOV R2,R1
        XPLC             ;PRINT LINE
        XEXT             ;RETURN
*
FP1     DATA >4140,>0000,>0000
FP2     DATA >40C9,>0FDA,>A220
MASK    TEXT 'SSS.999 999 999 999'
        BYTE 0
        END START
.ASM TEMP,TEMP1
ASM R2.4
SRCE=TEMP
OBJ=TEMP1
LIST=
ERR=
XREF=

END OF PASS 1
0 DIAGNOSTICS
END OF PASS 2
0 DIAGNOSTICS
.LINK
LINKER R2.4
*12,2
WAS >0000
*0,TEMP2
*1,TEMP1
*1,FPOUT:OBJ
*6
START TAG = >0000
*7
.TEMP2  3.141 592 653 590
.
```

(6.4 CONVERT FLOATING POINT TO DECIMAL continued)

Format characters are defined as follows:

Character	Digit holder	No digit
9	Yes	Space
0	Yes	0
$	Yes	Floats $
S	Yes	Floats sign
<	Yes	Floats < on negative
>	No	> on negative
E	No	Print sign
.	Decimal point	
,	Prints only if preceded by digit	
^	Replaced with period	

A digit holder is defined as a position where a digit can be stored. A floater appears only once and to the left of the first digit. If there are not enough digit holders to handle the edited number, the format is replaced with asterisks. All non-formatting characters are transferred to their corresponding positions in the output string.