MICROCHIP

AN575

IEEE 754 Compliant Floating-Point Routines

Author: Frank Testa

INTRODUCTION

This application note presents an implementation of the following floating point math routines for the PIC16/17 microcontroller family:

INTxx(A)	float to integer conversion
FLOxx(A)	integer to float conversion
NRMxx(A)	normalize
FPAxx(A , B)	add/subtract
FPMxx(A , B)	multiply
FPDxx(A , B)	divide

Routines for the PIC16/17 families are provided in a modified IEEE754 32 bit format together with versions in 24-bit truncated format.

FLOATING POINT ARITHMETIC

Although fixed point arithmetic can usually be employed in many numerical problems through the use of proper scaling techniques, this approach can become complicated and sometimes result in less efficient code than is possible using floating point methods[1]. Floating point arithmetic is essentially equivalent to arithmetic in scientific notation relative to a particular base or radix. In the special case of base two or binary arithmetic, a number A has the floating point representation:

$$A = f * (2^e) \,,\ f = \sum_{k=1}^{n} \{ a(k) * (2^{-k}) \}$$

where f is the fraction or mantissa, e is the exponent or characteristic, n is the number of bits in f and a(k), k=1,..,n are the bit values of f with a(1)=MSB. Typically, the mantissa is in normalized sign-magnitude representation with implicit MSB equal to one, and e is stored in biased form, where the bias is the magnitude of the most negative possible exponent[1,2]. If m is the number of bits in the biased exponent e^b:

$$e^b = e + 2^{m-1}$$

Using biased exponents permits comparison of exponents through a simple unsigned comparator, and further results in a unique representation of zero given by f = eb = 0.

Algorithms for radix conversion are discussed in APPENDIX A, and can be used to produce the binary floating point representation of a given decimal number. Examples of sign-magnitude floating point representations of some decimal numbers are as follows:

Decimal	e	f
1.0	1	.10000000
0.15625	-2	.10100000
0.1	-3	.110011001100....
$1.23x10^3$	11	.10011001110

It is important to note that the only numbers that can be represented exactly in binary arithmetic are those which are sums of powers of two, resulting in non-terminating binary representations of some simple decimal numbers such as .1 as shown above, and leading to truncation errors regardless of the value of n. Floating point calculations, even involving numbers admitting an exact binary representation, usually lose information after truncation to an n bit result, and therefore require some rounding scheme to minimize such roundoff errors[1].

ROUNDING METHODS

Truncation of a binary representation to n bits is severely biased since it always leads to a number whose magnitude is less than or equal to that of the exact value, thereby possibly causing significant error buildup during a long sequence of calculations. Simple adder-based rounding by adding the NSB to the LSB is unbiased except when the value to be rounded is equidistant from the two nearest n bit values[1]. In this case, magnitudes are always rounded up thereby producing a small but still undesirable bias. This can be removed by stipulating that in the equidistant case, the n bit value with LSB=0 is selected, commonly referred to as the rounding to the nearest method, the default mode in the IEEE754 standard[4,5]. The number of guard bits or extra bits of precision, is related to the sensitivity of the rounding method since using more guard bits results in fewer equidistant cases to be resolved. Since more than one guard bit requires an extra byte in PIC16/17 arithmetic, only one guard bit, usually handled in the carry bit, is employed in this library of floating point routines. Nearest neighbor rounding with one guard bit leads to the following simple result:

n Bit Value	Guard Bit	Result
A	0	round to A

A	1	if A,LSB=0, round to A
		if A,LSB=1, round to A+1

A+1	0	round to A+1

Another interesting rounding method, is Von Neumann rounding or jamming, where the exact number is truncated to n bits and then set LSB=1. Although the errors can be twice as large as in round to the nearest, it is unbiased and requires little more effort than truncation[1].

FLOATING POINT FORMATS

In what follows, we use the following floating point formats:

	eb	radix point	f0	f1	f2
IEEE754 32-bit	xxxxxxxx	.	Sxxxxxxx	xxxxxxxx	xxxxxxxx
truncated 24-bit	xxxxxxxx	.	Sxxxxxxx	xxxxxxxx	

where eb is the biased 8-bit exponent, with bias=2^7=128= 0x80, S is the sign bit, and bytes f0, f1 and f2 constitute the mantissa with f0 the most significant byte with implicit MSB = 1. It is important to note that the IEEE754 standard format[4] places the sign bit as the MSB of eb with the LSB of the exponent as the MSB of f0. Because of the inherent byte structure of the PIC16/17 family of microcontrollers, more efficient code was possible by adopting the above formats rather than strictly adhering to the IEEE standard.

The limiting absolute values of the above floating point formats are given as follows:

	32-bit format	eb	|A| e	f	decimal
MAX	0xFF7FFFFF	FF	7F	7FFFFF	1.7014117E+38
MIN	0x01000000	01	81	000000	2.9387359E-39

where the MSB is implicitly equal to one, and its bit location is occupied by the sign bit. The 24-bit format has the same structure but with only a 16-bit mantissa. While 24- to 32-bit conversion is trivial, requiring only an additional zero byte in the mantissa, a 32- to 24-bit conversion routine would typically employ nearest neighbor rounding before truncation.

To produce the correct representation of a particular decimal number, a high-level language compiler and debugger could be used to display the internal binary representation on a host computer and make the appropriate conversion to the above format. If this approach is not feasible, algorithms for producing this representation are contained in Appendix A.

FLOATING POINT EXCEPTIONS

Although the dynamic range of mathematical calculations is increased through floating point arithmetic, overflow and underflow are both possible when the limiting values of the representation are exceeded, such as in multiplication requiring the addition of exponents, or in division with the difference of exponents[2]. In these operations, mantissa calculations followed by appropriate normalizing and exponent modification can also lead to overflow or underflow in special cases. Similarly, addition and subtraction after mantissa alignment, followed by normalization can also lead to such exceptions. Furthermore, if nearest neighbor rounding is enabled, floating point overflow can occur in the case of rounding 0xFF7FFFFF with NSB=1 since adding the NSB would be required, resulting in mantissa overflow followed by a right shift and increment of the exponent, thus producing exponent overflow.

DATA RAM REQUIREMENTS

The following contiguous data ram locations are used by the library:

EXP	=	AEXP	AARG and ACC exponent
ACC+B0	=	AARG+B0	AARG and ACC mantissa
ACC+B1	=	AARG+B1	
ACC+B2	=	AARG+B2	
ACC+B3			additional ACC storage
ACC+B4			
ACC+B5			
SIGN			sign in MSB
FPFLAGS			exception flags, option bits
BEXP			BARG exponent
BARG+B0			BARG mantissa
BARG+B1			
BARG+B2			
TEMP+B0			temporary storage
TEMP+B1			

where Bx = x.

The exception flags and option bits in FPFLAGS are defined as follows:

	SAT	RND			FDZ	FUN	FOV	IOV
FPFLAGS								
	7	6	5	4	3	2	1	0

IOV	integer overflow flag
FOV	floating point overflow flag
FUN	floating point underflow flag
FDZ	floating point divide by zero flag
RND	nearest neighbor rounding enable bit
SAT	saturate enable bit

USAGE

For the unary operations, input argument and result are in AARG. The binary operations require input arguments in AARG and BARG, and produces the result in AARG, thereby simplifying sequencing of operations.

EXCEPTION HANDLING

All routines return WREG = 0x00 upon successful completion and WREG = 0xFF, together with the appropriate flag bit set upon exception. If SAT = 0, saturation is disabled and spurious results are obtained in AARG upon an exception. If SAT = 1, saturation is enabled and only overflow or underflow exceptions produce saturated results in AARG. Saturation is not applied on a divide by zero exception.

ROUNDING

With RND = 0, rounding is disabled, and simple truncation is used, resulting in some speed enhancement. If RND = 1, nearest neighbor rounding is implemented.

INTEGER TO FLOAT CONVERSION

The routine FLOxx converts the two's complement integer in AARG to the above floating point representation, producing the result in AEXP, AARG. The routine initializes the exponent to move the radix point to the left of the MSB and then calls the normalize routine. An example is given by:

FLO24(12106) = FLO24(0x2F4A) = 0x8E3D28 = 12106.0

NORMALIZE

The routine NRMxx takes an unnormalized floating point number with all bits explicit in AEXP, AARG and left shifts the mantissa and adjusts the exponent until the result has an explicit MSB = 1, followed by application of the sign bit thereby making the MSB implicit. This routine is called by FLOxx, FPAxx and FPSxx, and is usually not needed explicitly by the user since all operations producing a floating point result are implicitly normalized.

FLOAT TO INTEGER CONVERSION

The routine INTxx converts the normalized floating point number in AEXP, AARG, to a two's complement integer in AARG. After removing the bias from AEXP and precluding a result of zero or integer overflow, the mantissa in AARG is left shifted by AEXP and converted to two's complement representation. As an example, consider:

INT24(123.45) = INT24(0x87F6E6) = 0x7B = 123

ADDITION/SUBTRACTION

The floating point add routine FPAxx, takes the arguments in AEXP, AARG and BEXP, BARG and returns the sum in AEXP, AARG. If necessary, the arguments are swapped to insure that AEXP >= BEXP, and then BARG is then aligned by right shifting by AEXP - BEXP. The mantissas are then added and the result is normalized by calling NRMxx. The subtract routine FPSxx simply toggles the sign bit in BARG and calls FPAxx. Several examples are as follows:

```
FPA24( -.32212E+5,.11120E+4 )
        = FPA24( 0x8FFBA8, 0x8B0B00 )
        = 0x8FF2F8 = -.31100E+5

FPS24( .89010E+4,-.17802E+5 )
        = FPS24( 0x8E0B14, 0x8F8B14 )
        = 0x8F509E = .26703E+5
```

MULTIPLICATION

The floating point multiply routine FPMxx, takes the arguments in AEXP, AARG and BEXP, BARG and returns the product in AEXP, AARG. After testing for a zero argument, the sign and exponent of the result are computed together with testing for overflow. The mantissas are then multiplied using a standard add-shift method, followed by postnormalization if necessary. For example, consider:

```
FPM24( -.32212E+5,.11120E+4 )
        = FPM24( 0x8FFBA8, 0x8B0B00 )
        = 0x9A88A4 = -.35820E+8
```

DIVISION

The floating point divide routine FPDxx, takes the numerator in AEXP, AARG and denominator in BEXP, BARG and returns the quotient in AEXP, AARG. After testing for a zero denominator, the sign and exponent of the result are computed together with testing for dividend alignment. If the argument mantissas satisfy the inequality AARG >= BARG, the dividend AARG is right shifted by one bit and the exponent is adjusted, thereby resulting in AARG < BARG and the dividend is aligned. Alignment permits a valid division sequence and eliminates the need for postnormalization. After testing for overflow or underflow as appropriate, the mantissas are then divided using a standard shift-subtract restoring method. A simple example is given by:

```
FPD24( -.16106E+5, .24715E+5 )
      = FPD24( 0x8EFBA8, 0x8F4116 )
      = 0x80A6D4 = -.65167E+0
```

APPENDIX A

Several algorithms for decimal to binary conversion are given below. The integer and fractional conversion algorithms are useful in both native assembly as well as high level languages, and comprise the basis for the floating point decimal to binary conversion algorithm A.3. Algorithm A.4 is a more brute force method easily implemented on a calculator or in a high-level language on a host computer.

A.1 Integer conversion algorithm[3]:

Given an integer N, where d(k) are the bit values of its n bit binary representation with d(0)=LSB,

$$N = \sum_{k=0}^{n-1} \{ d(k) * (2^{k}) \}$$

```
k=0
N(k) = N
while N(k) =! 0
d(k) = remainder of N(k)/2
N(k+1) = [ N(k)/2 ]
k = k + 1
endw
```

where [ ] denotes the greatest integer function.

A.2 Fractional conversion algorithm[3]:

Given a fraction N, where d(k) are the bit values of its n bit binary representation with d(1)=MSB:

$$N = \sum_{k=1}^{n} \{ d(k) * (2^{-k}) \}$$

```
k=0
N(k) = N
while k <= n
d(k) = [ N(k)*2 ]
N(k+1) = fractional part of N(k)*2
k = k + 1
endw
```

A.3 Floating point decimal to binary conversion algorithm[3]:

Given a decimal number $A = F * 10^{E}$, and the number of mantissa bits n, where F is not necessarily normalized, the binary floating point representation $A = f * 2^{e}$ can be found as follows:

```
normalize F and adjust E
convert $10^{E}$ to the normalized binary representation $f' * 2^{e}$
convert the fraction F to the binary fraction f using A.2
compute f = f * f'
normalize f and adjust e if necessary
```

The conversion of the integer 10^{E} can be done with algorithm A.1, or if space is available, can be performed by a table lookup. Furthermore, the conversion should be made using a number of guard bits and then rounded to an n bit result by the desired rounding method.

A.4 Floating point decimal to binary conversion algorithm:

Given a decimal number A, and the number of mantissa bits n, the bits in the mantissa of the above binary representation of A, a(k), k = 1,2,...,n, where a(1) = MSB, are given by the following algorithm:

```
z = ln A / ln 2
e = int ( z )
if e <= z
    e = e + 1
endif
x = A / (2**e)
k = 1
while k <= n
    if x >= 2**(-k)
        a(k) = 1
    else
        a(k) = 0
    endif
    x = x - a(k) * 2**(-k)
    k = k + 1
endw
```

Formally, the number A then has the floating point representation:

$$A = f * (2^{e}) \text{ , } f = \sum_{k=1}^{n} \{ a(k) * (2^{-k}) \}$$

5

A simple fortran implementation of algorithm A.4 is given as follows:

```
      PROGRAM FPREP
C
      IMPLICIT NONE
C
      REAL*8 X,Y,Z
C
      INTEGER*4 K,N,P,A(32),EB,F,BIAS
C
      PARAMETER (BIAS=128)
C
1     PRINT 200
200   FORMAT(1X,'ENTER N: ',$)
      READ 100,N
100   FORMAT(I5)
C
2     PRINT 201
201   FORMAT(1X,'ENTER Y: ',$)
      READ 101,Y
101   FORMAT(E32.15)
      IF(Y.LE.0.)GOTO 1
C
      F=0
      Z=DLOG(Y)/DLOG(2.D0)
      P=INT(Z)
      IF(DBLE(P).LE.Z)P=P+1
      EB=P+BIAS
      X=Y*2.**(-P)
      DO K=1,N
            IF(X.GE.2.**(-K))THEN
                  A(K)=1
            ELSE
                  A(K)=0
            ENDIF
            IF(A(K).NE.0)F=IBSET(F,31-K+1)
            X=X-DBLE(A(K))*2.**(-K)
      ENDDO
C
      PRINT 300,P,(A(K),K=1,N)
300   FORMAT(1X,I5,3X,8(I2,I2,I2,I2,1X))
C
      PRINT 400,EB,F
400   FORMAT(1X,Z4,2X,Z8)
C
      GOTO 2
C
      END
```

GLOSSARY

BIASED EXPONENTS - nonnegative representation of exponents produced by adding the absolute value of the most negative exponent to a two's complement exponent.

FLOATING POINT UNDERFLOW - occurs when the real number to be represented is smaller in absolute value than the smallest floating point number.

FLOATING POINT OVERFLOW - occurs when the real number to be represented is larger in absolute value than the largest floating point number.

GUARD BITS - additional bits of precision carried in a calculation for improved rounding sensitivity.

LSB - least significant bit

MSB - most significant bit

NEAREST NEIGHBOR ROUNDING - an unbiased rounding method where a number to be rounded is rounded to its nearest neighbor in the representation, with the stipulation that if equidistant from its nearest neighbors, the neighbor with LSB equal to zero is selected.

NORMALIZATION - the process of left shifting the mantissa of an unnormalized floating point number until the MSB equals one, while decreasing the exponent by the number of left shifts.

NSB - next significant bit just to the right of the LSB.

ONE'S COMPLEMENT - a special case of the diminished radix complement for radix two systems where the value of each bit is reversed. Although sometimes used in representing positive and negative numbers, it produces two representations of the number zero.

RADIX - the base of a given number system.

RADIX POINT - separates the integer and fractional parts of a number.

SATURATION - mode of operation where integer and floating point numbers are fixed at there limiting values when an underflow or overflow is detected.

SIGN MAGNITUDE - representation of positive and negative binary numbers where the absolute value is expressed together with the appropriate value of the sign bit.

TRUNCATION - discarding any bits to the right of a given bit location.

TWO'S COMPLEMENT - a special case of radix complement for radix two systems where the value of each bit is reversed and the result is incremented by one. Producing a unique representation of zero, and covering the range -2**(n-1) to 2**(n-1)-1, this is more easily applied in addition and subtraction operations and is therefore the most commonly used method of representing positive and negative numbers.

FIGURE 1: INTEGER TO FLOAT CONVERSION

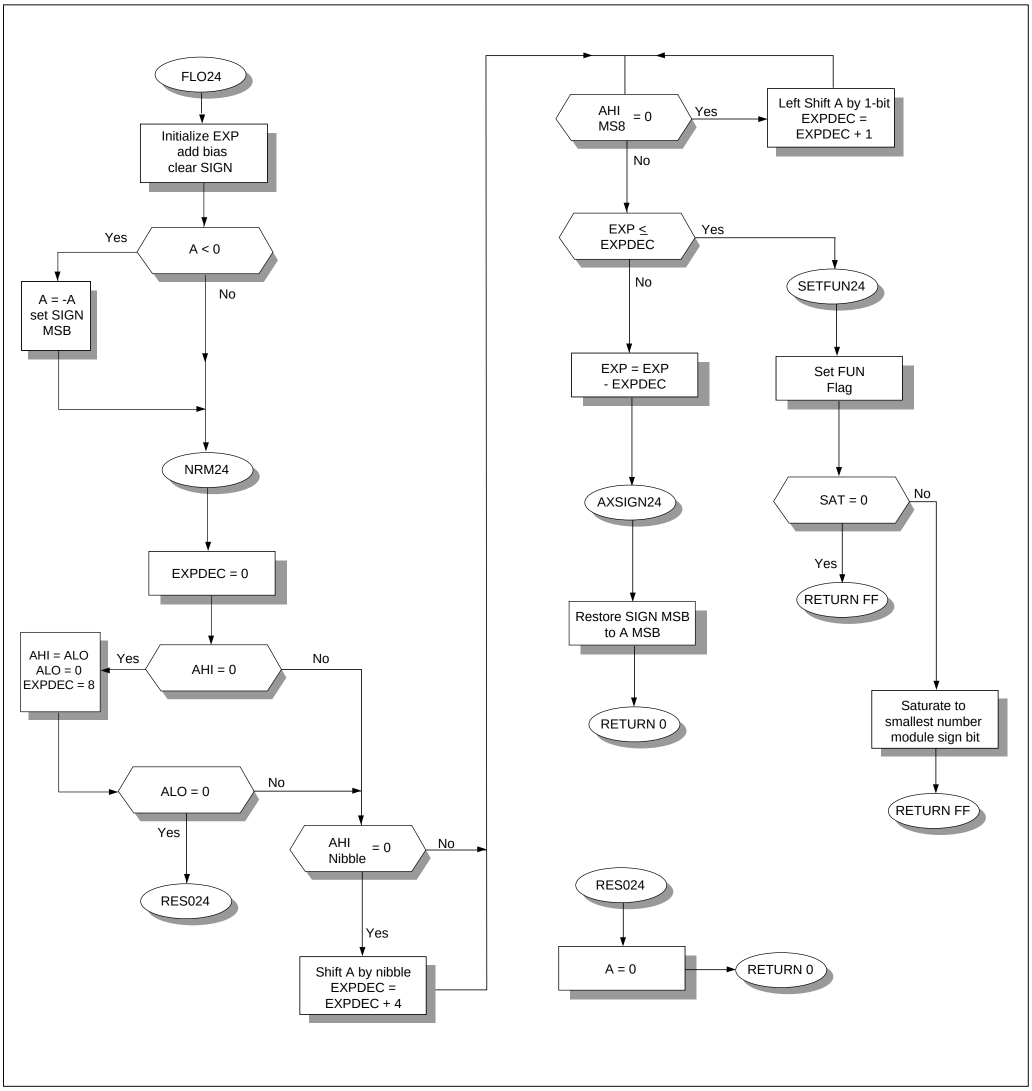


FIGURE 2: FLOAT TO INTEGER CONVERSION

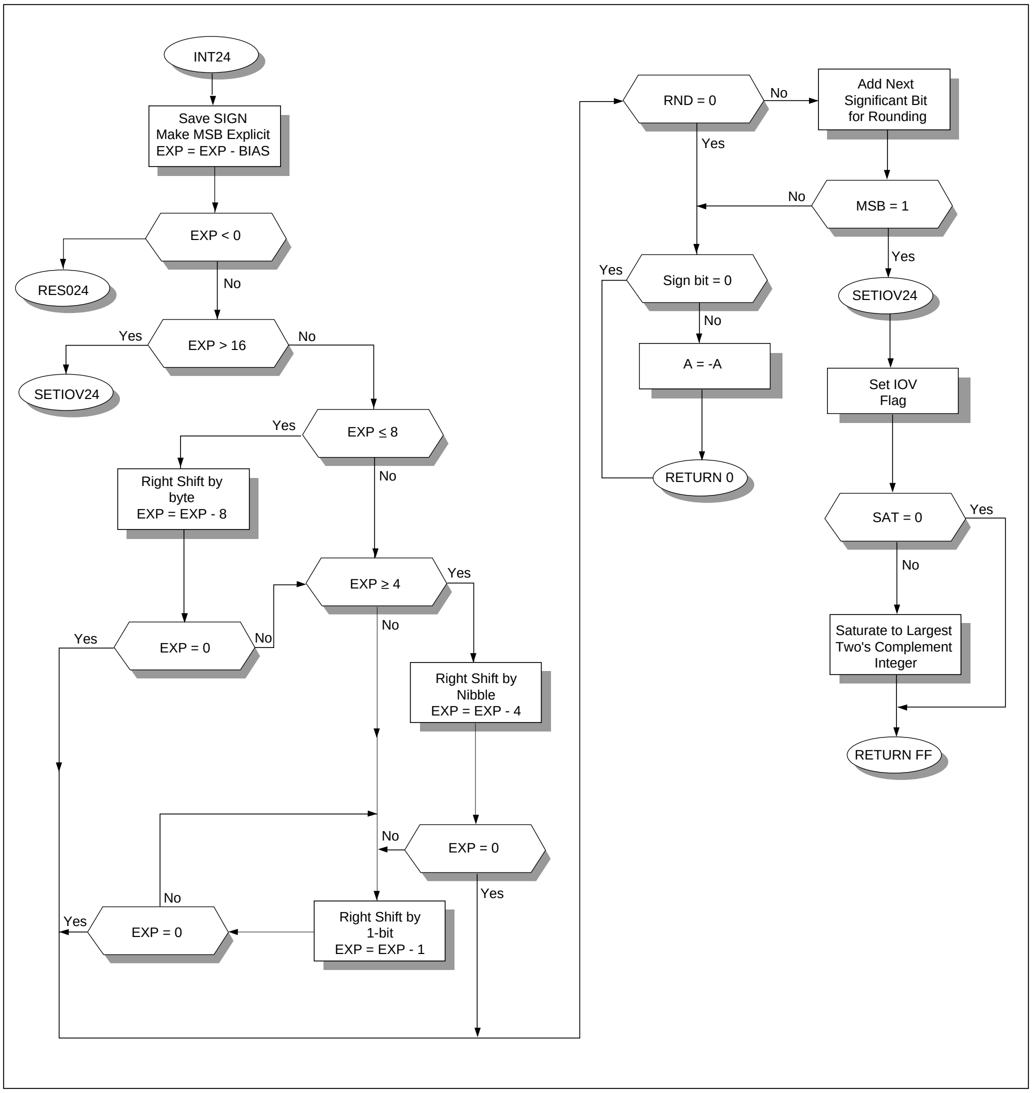


FIGURE 3: FLOATING POINT MULTIPLY

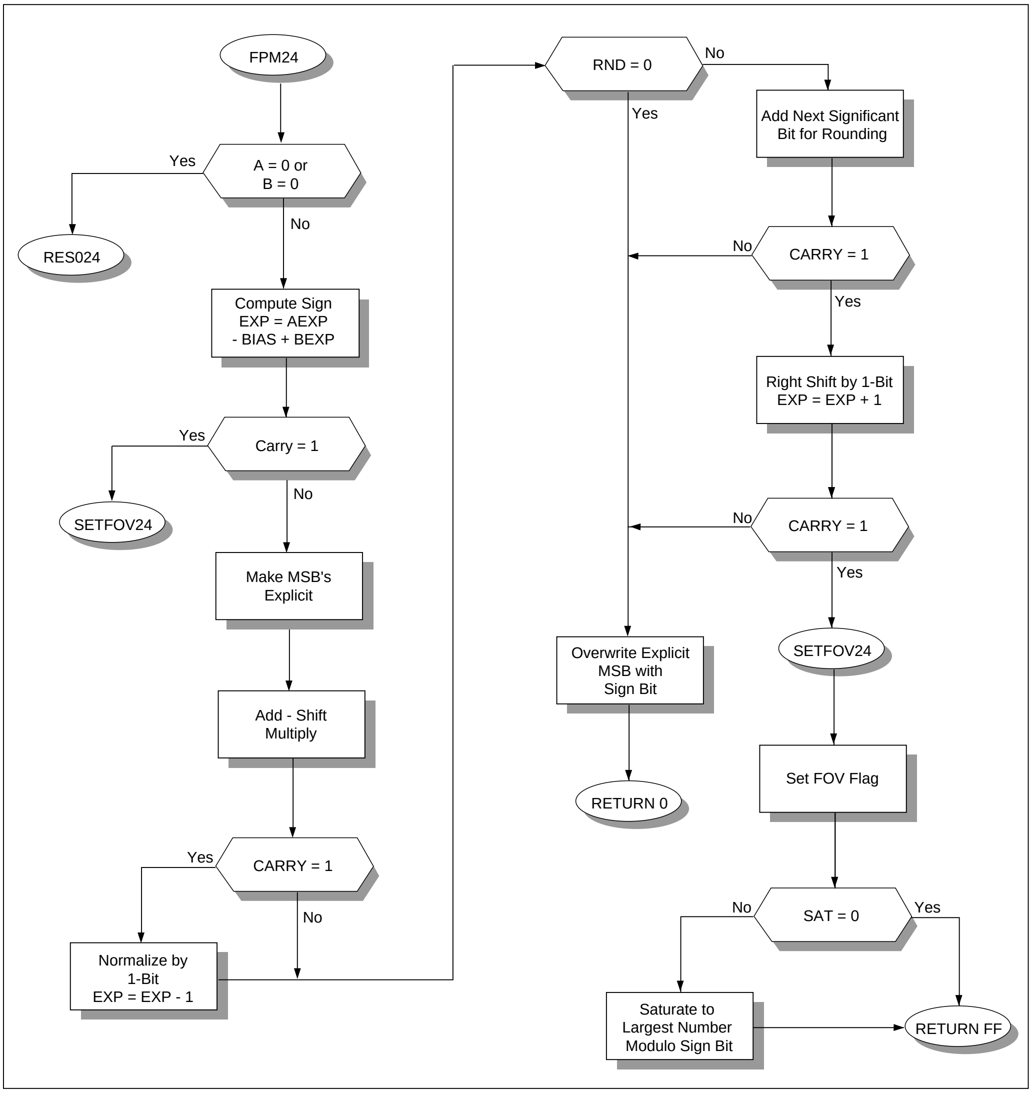

FPM24
A = 0 or B = 0
Yes
No
RES024
Compute Sign EXP = AEXP - BIAS + BEXP
Carry = 1
Yes
No
SETFOV24
Make MSB's Explicit
Add - Shift Multiply
CARRY = 1
Yes
No
Normalize by 1-Bit EXP = EXP - 1
RND = 0
No
Yes
Add Next Significant Bit for Rounding
CARRY = 1
No
Yes
Right Shift by 1-Bit EXP = EXP + 1
CARRY = 1
No
Yes
Overwrite Explicit MSB with Sign Bit
RETURN 0
SETFOV24
Set FOV Flag
SAT = 0
No
Yes
Saturate to Largest Number Modulo Sign Bit
RETURN FF

FIGURE 4: FLOATING POINT DIVIDE

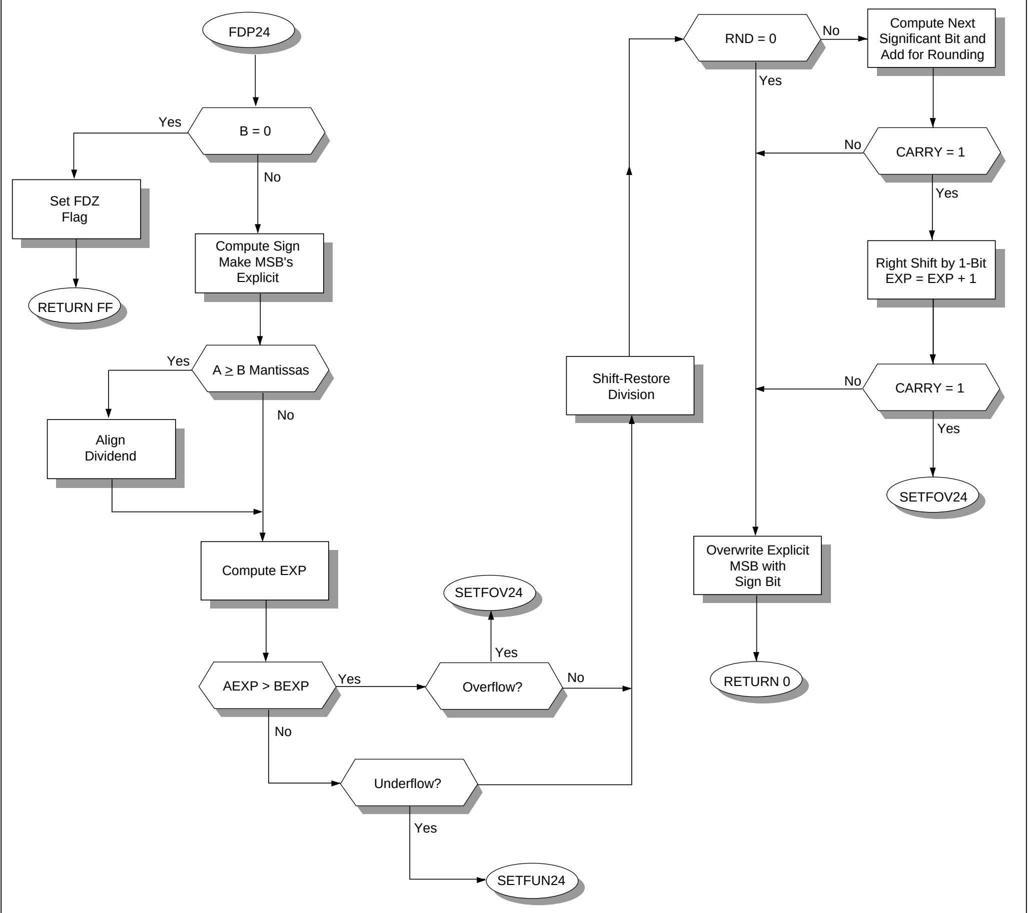


FIGURE 5: FLOATING POINT SUBTRACT

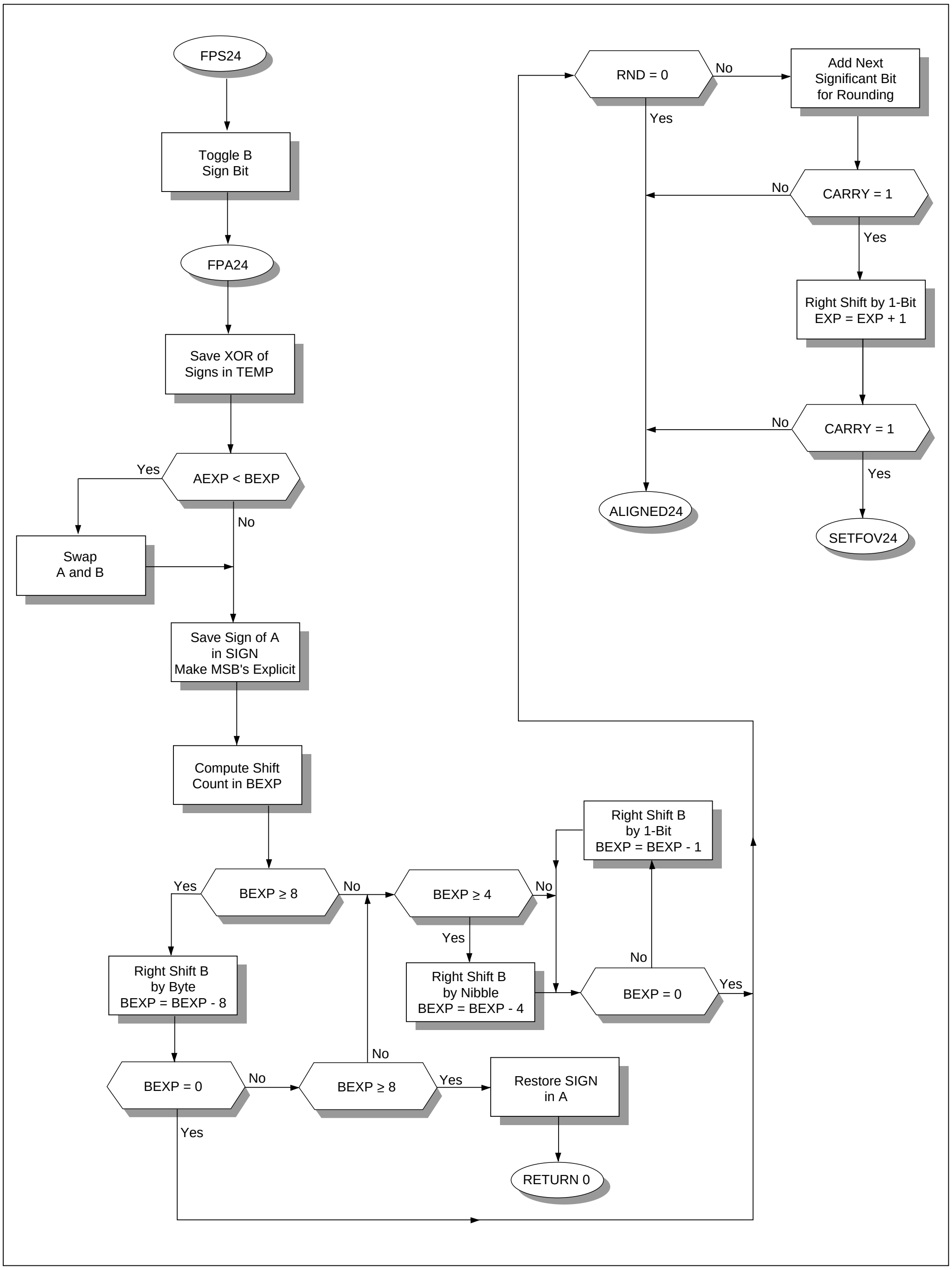
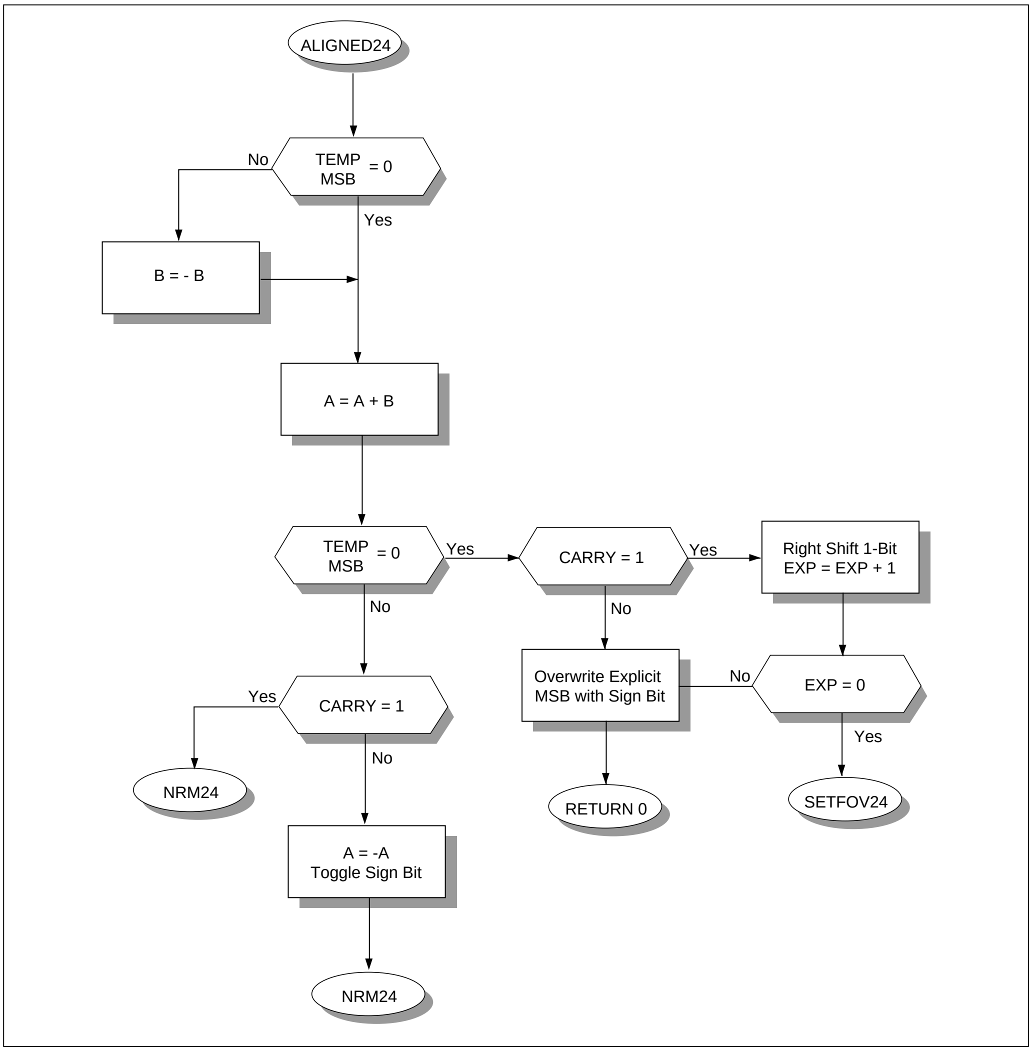

FPS24
Toggle B Sign Bit
FPA24
Save XOR of Signs in TEMP
AEXP < BEXP
Yes
No
Swap A and B
Save Sign of A in SIGN Make MSB's Explicit
Compute Shift Count in BEXP
BEXP ≥ 8
Yes
No
Right Shift B by Byte BEXP = BEXP - 8
BEXP = 0
No
Yes
BEXP ≥ 8
No
Yes
Restore SIGN in A
RETURN 0
BEXP ≥ 4
No
Yes
Right Shift B by Nibble BEXP = BEXP - 4
BEXP = 0
No
Yes
Right Shift B by 1-Bit BEXP = BEXP - 1
RND = 0
No
Yes
Add Next Significant Bit for Rounding
CARRY = 1
No
Yes
Right Shift by 1-Bit EXP = EXP + 1
CARRY = 1
No
Yes
ALIGNED24
SETFOV24

FIGURE 6: NORMALIZATION



REFERENCES

1. Cavanagh, J.J.F., “Digital Computer Arithmetic,” McGraw-Hill,1984.

2. Hwang, K., “Computer Arithmetic,” John Wiley & Sons, 1979.

3. Scott, N.R., “Computer Number Systems & Arithmetic,” Prentice Hall, 1985.

4. IEEE Standards Board, “IEEE Standard for Floating-Point Arithmetic,” ANSI/IEEE Std 754-1985, IEEE, 1985.

5. Knuth, D.E., “The Art of Computer Programming, Volume 2,” Addison-Wesley, 1981.

TABLE 1: PIC17CXX FLOATING POINT PERFORMANCE DATA

Routine	Max Cycles	Min. Cycles	PM	DM	
FLO24	48 / 54	21 / 26	72	6	
INT24	57 / 65	11 / 28	101	6	
NRM24	39 / 46	15 / 20	63	6	
FPA24	116 / 131	31	212	10	
FPS24	117 / 132	32	213	10	
FPM24	175 / 189	10 / 70	224	10	
FPD24	328 / 352	11 / 277	377	11	
			1262	11	Total Memory
FLO32	60 / 67	26 / 30	120	7	
INT32	69 / 76	12 / 30	155	7	
NRM32	48 / 55	20 / 24	108	7	
FPA32	144 / 159	34	329	12	
FPS32	145 / 160	35	330	12	
FPM32	320 / 336	11 / 88	382	13	
FPD32	597 / 627	12 / 478	661	14	
			2085	14	Total Memory

TABLE 2: PIC16C5X / PIC16CXX FLOATING POINT PERFORMANCE DATA

Routine	Max Cycles	Min. Cycles	PM	DM	
FLO24	83 / 88	21 / 25	37	6	
INT24	80 / 89	14 / 31	64	6	
NRM24	72 / 77	14 / 18	26	6	
FPA24	194 / 210	44	102	11	
FPS24	196 / 212	46	104	11	
FPM24	298 / 313	11 / 218	80	11	
FPD24	469 / 495	348	117	11	
			530	11	Total Memory
FLO32	104 / 110	24 / 34	52	7	
INT32	90 / 100	15 / 32	83	6	
NRM32	90 / 96	18 / 28	38	7	
FPA32	248 / 268	50	136	14	
FPS32	250 / 270	52	138	14	
FPM32	574 / 591	12 / 315	94	14	
FPD32	929 / 968	704	152	14	
			693	14	Total Memory

APPENDIX A

```
;       PIC16 24 BIT FLOATING POINT LIBRARY     VERSION 1.14
;       Unary operations: both input and output are in AEXP,AARG
;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG
;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.
;       All timings are worst case cycle counts
;
;         Routine                Function
;
;       FLO24           16 bit integer to 24 bit floating point conversion
;
;               Timing:            RND
;                               0       1
;
;                       0       83      83
;                  SAT
;                       1       88      88

;       NRM24           normalization of unnormalized 24 bit floating point numbers

;               Timing:            RND
;                               0       1

;                       0       72      72
;                  SAT
;                       1       77      77


;       INT24           24 bit floating point to 16 bit integer conversion


;               Timing:            RND
;                               0       1

;                       0       80      86
;                  SAT
;                       1       80      89

;       FPA24           24 bit floating point add

;               Timing:            RND
;                               0       1

;                       0       194     205
;                  SAT
;                       1       194     210

;       FPS24           24 bit floating point subtract

;               Timing:            RND
;                               0       1

;                       0       196     207
;                  SAT
;                       1       196     212

;       FPM24           24 bit floating point multiply

;               Timing:            RND
;                               0       1

;                       0       298     309
;                  SAT
;                       1       298     313

;       FPD24           24 bit floating point divide

;               Timing:            RND
;                               0       1

;                       0       469     491
;                  SAT
;                       1       469     495

                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;**********************************************************************************************
;**********************************************************************************************

;       24 bit floating point representation

;       EXPONENT        8 bit biased exponent
;                       It is important to note that the use of biased exponents produces
;                       a unique representation of a floating point 0, given by
;                       EXP = HIGHBYTE = LOWBYTE = 0x00, with 0 being the only
;                       number with EXP = 0.

;       HIGHBYTE        8 bit most significant byte of sign-magnitude representation, with
;                       SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                       left of MSB

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                       RADIX
;       EXPONENT        POINT   HIGHBYTE        LOWBYTE

;       xxxxxxxx           .    Sxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS equ     128

;       Define library register variables

EXP             equ     0x0C    ; 8 bit biased exponent

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AEXP            equ     0x0C    ; 8 bit biased exponent for argument A

AARG            equ     0x0D    ; most significant byte of mantissa for argument A

BEXP            equ     0x15    ; 8 bit biased exponent for argument B

BARG            equ     0x16    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1 )  and  (EXP, ACC+B0, ACC+B1)
;               reference the same storage locations.
```

5

```
;       Define floating point library exception flags

FPFLAGS        equ     0x14   ; floating point library exception flags

IOV            equ     0      ; bit0 = integer overflow flag

FOV            equ     1      ; bit1 = floating point overflow flag

FUN            equ     2      ; bit2 = floating point underflow flag

FDZ            equ     3      ; bit3 = floating point divide by zero flag

RND            equ     6      ; bit6 = floating point rounding flag, 0 = truncation
                              ; 1 = rounding to nearest LSB

SAT            equ     7      ; bit7 = floating point saturate flag, 0 = terminate on
                              ; exception without saturation, 1 = terminate on
                              ; exception with saturation to appropriate value

;**********************************************************************************************
;**********************************************************************************************

;       Integer to float conversion

;       Input:  16 bit 2's complement integer right justified in AARG+B0, AARG+B1

;       Use:    CALL    FLO24

;       Output: 24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  FLOAT( AARG )

;       Max Timing:     11+72 = 83 clks         SAT = 0
;                                 11+77 = 88 clks         SAT = 1

;       Min Timing: 7+14 = 21 clks              AARG = 0
;                                 7+18 = 25 clks

;       PM: 11+26 = 37                    DM: 6

FLO24           MOVLW           16+EXPBIAS      ; initialize exponent and add bias
                MOVWF           EXP
                MOVF            ACC+B0,W
                MOVWF           SIGN
                BTFSS           ACC+B0,MSB      ; test sign
                GOTO            NRM24
                COMF            ACC+B1          ; if < 0, negate and set MSB in SIGN
                COMF            ACC+B0
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0

;——————————————————————————————————

;       Normalization routine

;       Input:  24 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               with sign in SIGN,MSB and other bits zero.

;       Use:    CALL    NRM24

;       Output: 24 bit normalized floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  NORMALIZE( AARG )

;       Max Timing:     10+6+7*7+7 = 72 clks            SAT = 0
;                                 10+6+7*7+1+11 = 77 clks SAT = 1

;       Min Timing: 14 clks                     AARG = 0
;                                       5+9+4 = 18 clks

;       PM: 26                                  DM: 6

NRM24           CLRF            TEMP                    ; clear exponent decrement
                MOVF            ACC+B0,W                ; test if highbyte=0
                BTFSS           _Z
                GOTO            NORM24
                MOVF            ACC+B1,W                ; if so, shift 8 bits by move
                MOVWF           ACC+B0
                BTFSC           _Z                      ; if highbyte=0, result=0
                GOTO            RES024
                CLRF            ACC+B1
                BSF             TEMP,3

NORM24          MOVF            TEMP,W
                SUBWF           EXP
                BTFSS           _Z
                BTFSS           _C
                GOTO            SETFUN24

                BCF             _C                      ; clear carry bit

NORM24A         BTFSC           ACC+B0,MSB              ; if MSB=1, normalization done
                GOTO            FIXSIGN24
                RLF             ACC+B1                  ; otherwise, shift left and
                RLF             ACC+B0                  ; decrement EXP
                DECFSZ  EXP
                GOTO            NORM24A

                GOTO            SETFUN24                ; underflow if EXP=0

FIXSIGN24       BTFSS SIGN,MSB
                BCF             ACC+B0,MSB              ; clear explicit MSB if positive
                RETLW           0

;**********************************************************************************************
;**********************************************************************************************

;       Float to integer conversion

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Use:    CALL    INT24

;       Output: 16 bit 2’s complement integer right justified in AARG+B0, AARG+B1

;       Result: AARG  <—  INT( AARG )

;       Max Timing:     26+6*6+5+13 = 80 clks           RND = 0
;                                       26+6*6+5+19 = 86 clks           RND = 1, SAT = 0
;                                       26+6*6+5+22 = 89 clks           RND = 1, SAT = 1

;       Min Timing: 9+5 = 14 clks               AARG = 0
;                                       19+5+7 = 31 clks

;       PM: 64                                  DM: 6

INT24           MOVF            ACC+B0,W                ; save sign in SIGN
                MOVWF           SIGN
                BSF             ACC+B0,MSB              ; make MSB explicit

                MOVLW           EXPBIAS                 ; remove bias from EXP
                SUBWF           EXP
                BTFSS           EXP,MSB                 ; if <= 0, result=0
                BTFSC           _Z
                GOTO            RES024

                MOVF            EXP,W
```

5

```
                SUBLW           16
                MOVWF           EXP
                BTFSS           _Z
                BTFSC           EXP,MSB
                GOTO            SETIOV24

                MOVLW           8                   ; do byte shift if EXP >= 8
                SUBWF           EXP,W
                BTFSS           _C
                GOTO            SHIFT24
                MOVWF           EXP
                RLF             ACC+B1              ; rotate next bit for rounding
                MOVF            ACC+B0,W
                MOVWF           ACC+B1
                CLRF            ACC+B0

                MOVF            EXP,W               ; shift completed if EXP = 0
                BTFSC           _Z
                GOTO            SHIFT24OK

SHIFT24         BCF             _C
                RRF             ACC+B0              ; right shift by EXP
                RRF             ACC+B1
                DECFSZ  EXP
                GOTO            SHIFT24

SHIFT24OK       BTFSC FPFLAGS,RND
                BTFSS           ACC+B1,LSB
                GOTO            INT24OK
                CLRF            ACC+B2
                RLF             ACC+B2,W
                ADDWF           ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                BTFSC           ACC+B0,MSB          ; test for overflow
                GOTO            SETIOV24

INT24OK         BTFSS           SIGN,MSB            ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                RETLW           0

RES024          CLRF            ACC+B0              ; integer result equals zero
                CLRF            ACC+B1
                CLRF            EXP                 ; clear EXP for other routines
                RETLW           0

SETIOV24        BSF             FPFLAGS,IOV         ; set integer overflow flag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                CLRF            ACC+B0              ; saturate to largest two’s
                BTFSS           SIGN,MSB            ; complement 16 bit integer
                MOVLW           0xFF
                MOVWF           ACC+B0              ; SIGN = 0, 0x 7F FF
                MOVWF           ACC+B1              ; SIGN = 1, 0x 80 00
                RLF             SIGN
                RRF             ACC+B0
                RETLW           0xFF                ; return error code in WREG


;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Multiply
;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPM24

;       Output: 24 bit floating point product in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG * BARG

;       Max Timing:     25+15*16+15+18 = 298 clks       RND = 0
;                               25+15*16+15+29 = 309 clks       RND = 1, SAT = 0
;                               25+15*16+15+33 = 313 clks       RND = 1, SAT = 1

;       Min Timing:     6+5 = 11 clks           AARG * BARG = 0
;                               24+15*11+14+15 = 218 clks

;       PM: 80                                  DM: 11

FPM24           MOVF            AEXP,W                  ; test for zero arguments
                BTFSS           _Z
                MOVF            BEXP,W
                BTFSC           _Z
                GOTO            RES024

M24BNE0         MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN                    ; save sign in SIGN

                MOVF            BEXP,W
                ADDWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            MTUN24

                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV24                ; set multiply overflow flag
                GOTO            MOK24

MTUN24          ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN24

MOK24           BSF             AARG+B0,MSB             ; make argument MSB’s explicit
                BSF             BARG+B0,MSB
                BCF             _C
                CLRF            ACC+B2                  ; clear initial partial product
                CLRF            ACC+B3
                MOVLW           16
                MOVWF           TEMP                    ; initialize counter

MLOOP24         BTFSS           AARG+B1,LSB             ; test high byte
                GOTO            MNOADD24

MADD24          MOVF            BARG+B1,W
                ADDWF           ACC+B3
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ  BARG+B0,W
                ADDWF           ACC+B2

MNOADD24        RRF        ACC+B2
                RRF             ACC+B3
                RRF             ACC+B0
                RRF             ACC+B1
                BCF             _C
                DECFSZ  TEMP
                GOTO            MLOOP24
                BTFSC           ACC+B2,MSB          ; check for postnormalization
                GOTO            MROUND24
                RLF             ACC+B0
                RLF             ACC+B3
                RLF             ACC+B2
                DECF            EXP

MROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            MUL24OK
                RLF             ACC+B0              ; rotate next significant bit into
                BTFSC           _C
                INCF            ACC+B3              ; carry for rounding
                BTFSC           _Z
                INCF            ACC+B2

                BTFSS           _Z                  ; has rounding caused carryout?
                GOTO            MUL24OK
                RRF             ACC+B2              ; if so, right shift
                RRF             ACC+B3
                INCF            EXP
                BTFSC           _C                  ; check for overflow
                GOTO            SETFOV24

MUL24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVF            ACC+B2,W
                MOVWF           AARG+B0             ; move result to AARG
                MOVF            ACC+B3,W
                MOVWF           AARG+B1
                RETLW           0

SETFOV24        BSF             FPFLAGS,FOV         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0xFF
                MOVWF           AEXP                ; saturate to largest floating
                MOVWF           AARG+B0             ; point number = 0x FF 7F FF
                MOVWF           AARG+B1             ; modulo the appropriate sign bit
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                ; return error code in WREG

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Divide

;       Input:  24 bit floating point dividend in AEXP, AARG+B0, AARG+B1
;               24 bit floating point divisor in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPD24

;       Output: 24 bit floating point quotient in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG / BARG

;       Max Timing:     29+13+15*26+25+12 = 469 clks    RND = 0
;                               29+13+15*26+25+34 = 491 clks    RND = 1, SAT = 0
;                               29+13+15*26+25+38 = 495 clks    RND = 1, SAT = 1

;       Min Timing:     24+12+15*20+12 = 348 clks

;       PM: 117                         DM: 11

FPD24           MOVF            BEXP,W              ; test for divide by zero
```

```
                BTFSC           _Z
                GOTO            SETFDZ24

D24BNE0         MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN                    ; save sign in SIGN
                BSF             AARG+B0,MSB             ; make argument MSB's explicit
                BSF             BARG+B0,MSB

TALIGN24        CLRF            TEMP                    ; clear align increment
                MOVF            AARG+B0,W
                MOVWF           ACC+B2                  ; test for alignment
                MOVF            AARG+B1,W
                MOVWF           ACC+B3

                MOVF            BARG+B1,W
                SUBWF           ACC+B3
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           ACC+B2

                CLRF            ACC+B2
                CLRF            ACC+B3

                BTFSS           _C
                GOTO            DALIGN24OK

                BCF             _C                      ; align if necessary
                RRF             ACC+B0
                RRF             ACC+B1
                RRF             ACC+B2
                MOVLW           0x01
                MOVWF           TEMP                    ; save align increment

DALIGN24OK      MOVF            BEXP,W                  ; compare AEXP and BEXP
                SUBWF           EXP
                BTFSS           _C
                GOTO            ALTB24

AGEB24          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV24
                GOTO            DARGOK24                ; set overflow flag

ALTB24          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN24                ; set underflow flag

DARGOK24        MOVLW           16                      ; initialize counter
                MOVWF           TEMP+B1

DLOOP24         RLF             ACC+B3                  ; left shift
                RLF             ACC+B2
                RLF             ACC+B1
                RLF             ACC+B0
                RLF             TEMP

                MOVF            BARG+B1,W               ; subtract
                SUBWF           ACC+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           ACC+B0
                RLF             BARG+B0,W
                IORWF           TEMP

                BTFSS           TEMP,LSB            ; test for restore
                GOTO            DREST24

                BSF             ACC+B3,LSB
                GOTO            DOK24

DREST24         MOVF            BARG+B1,W           ; restore if necessary
                ADDWF           ACC+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           ACC+B0
                BCF             ACC+B3,LSB

DOK24           DECFSZ          TEMP+B1
                GOTO            DLOOP24

DROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            DIV24OK
                BCF             _C
                RLF             ACC+B1              ; compute next significant bit
                RLF             ACC+B0              ; for rounding
                RLF             TEMP

                MOVF            BARG+B1,W           ; subtract
                SUBWF           ACC+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           ACC+B0

                RLF             BARG+B0,W
                IORWF           TEMP,W
                ANDLW           0x01

                ADDWF           ACC+B3
                BTFSC           _C
                INCF            ACC+B2

                BTFSS           _Z                  ; test if rounding caused carryout
                GOTO            DIV24OK
                RRF             ACC+B2
                RRF             ACC+B3
                INCF            EXP
                BTFSC           _Z                  ; test for overflow
                GOTO            SETFOV24

DIV24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVF            ACC+B2,W
                MOVWF           AARG+B0             ; move result to AARG
                MOVF            ACC+B3,W
                MOVWF           AARG+B1

                RETLW           0

SETFUN24        BSF             FPFLAGS,FUN         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0x01                ; saturate to smallest floating
                MOVWF           AEXP                ; point number = 0x 01 00 00
                CLRF            AARG+B0             ; modulo the appropriate sign bit
                CLRF            AARG+B1
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                ; return error code in WREG

SETFDZ24        BSF             FPFLAGS,FDZ         ; set divide by zero flag
                RETLW           0xFF

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Subtract

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPS24

;       Output: 24 bit floating point sum in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     2+194 = 196 clks                RND = 0
;                                       2+205 = 207 clks                RND = 1, SAT = 0
;                                       2+210 = 212 clks                RND = 1, SAT = 1

;       Min Timing:     2+44 = 46 clks

;       PM: 2+102 = 104         DM: 11

FPS24           MOVLW           0x80
                XORWF           BARG+B0

;————————————————————————————————

;       Floating Point Add

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPA24

;       Output: 24 bit floating point sum in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     25+25+6*6+5+31+72 = 194 clks    RND = 0
;                                       25+25+6*6+5+42+72 = 205 clks    RND = 1, SAT = 0
;                                       25+25+6*6+5+42+77 = 210 clks    RND = 1, SAT = 1

;       Min Timing:     22+18+4 = 44 clks

;       PM: 102                                 DM: 11

FPA24           MOVF            AARG+B0,W           ; exclusive or of signs in TEMP
                XORWF           BARG+B0,W
                MOVWF           TEMP

                MOVF            AEXP,W              ; use AARG if AEXP >= BEXP
                SUBWF           BEXP,W
                BTFSS           _C
                GOTO            USEA24

                MOVF            BEXP,W
                MOVWF           ACC+B2              ; otherwise, swap AARG and BARG
                MOVF            AEXP,W
                MOVWF           BEXP
                MOVF            ACC+B2,W
                MOVWF           AEXP
```

5

```
                MOVF            BARG+B0,W
                MOVWF           ACC+B2
                MOVF            AARG+B0,W
                MOVWF           BARG+B0
                MOVF            ACC+B2,W
                MOVWF           AARG+B0

                MOVF            BARG+B1,W
                MOVWF           ACC+B2
                MOVF            AARG+B1,W
                MOVWF           BARG+B1
                MOVF            ACC+B2,W
                MOVWF           AARG+B1

USEA24          MOVF            AARG+B0,W
                MOVWF           SIGN                    ; save sign in SIGN
                BSF             AARG+B0,MSB             ; make MSB’s explicit
                BSF             BARG+B0,MSB

                MOVF            BARG+B0,W
                MOVWF           ACC+B2
                MOVF            BARG+B1,W
                MOVWF           ACC+B3

                MOVF            BEXP,W                  ; compute shift count in BEXP
                SUBWF           AEXP,W
                MOVWF           BEXP
                BTFSC           _Z
                GOTO            AROUND24

                MOVLW           8
                SUBWF           BEXP,W
                BTFSS           _C                      ; if BEXP >= 8, do byte shift
                GOTO            ALIGNB24
                MOVWF           BEXP
                RLF             ACC+B3                  ; rotate next bit for rounding
                MOVF            ACC+B2,W
                MOVWF           ACC+B3
                CLRF            ACC+B2

ALIGNB24        MOVF            BEXP,W                  ; already aligned if BEXP = 0
                BTFSC           _Z
                GOTO            AROUND24

ALOOPB24        BCF             _C                      ; right shift by BEXP
                RRF             ACC+B2
                RRF             ACC+B3
                DECFSZ          BEXP
                GOTO            ALOOPB24

AROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            ALIGNED24
                BTFSS           _C
                GOTO            ALIGNED24
                INCF            ACC+B3                  ; carry for rounding
                BTFSC           _Z
                INCF            ACC+B2
                BTFSS           _Z
                GOTO            ALIGNED24
                RRF             ACC+B2
                RRF             ACC+B3
                INCF            EXP
                BTFSC           _Z
                GOTO            SETFOV24

ALIGNED24       BTFSS           TEMP,MSB                ; negate if signs opposite
                GOTO            AOK24
                COMF            ACC+B3
                COMF            ACC+B2
                INCF            ACC+B3
                BTFSC           _Z
                INCF            ACC+B2

AOK24           MOVF            ACC+B3,W                ; add
                ADDWF           AARG+B1
                BTFSC           _C
                INCF            ACC+B0
                MOVF            ACC+B2,W
                ADDWF           AARG+B0

                BTFSC           TEMP,MSB
                GOTO            ACOMP24
                BTFSS           _C
                GOTO            FIXSIGN24

                RRF             AARG+B0                 ; shift right and increment EXP
                RRF             AARG+B1
                INCFSZ  AEXP
                GOTO            FIXSIGN24
                GOTO            SETFOV24

ACOMP24         BTFSC           _C
                GOTO            NRM24                   ; normalize and fix sign

                COMF            AARG+B1                 ; negate, toggle sign bit and
                COMF            AARG+B0                 ; then normalize
                INCF            AARG+B1
                BTFSC           _Z
                INCF            AARG+B0

                MOVLW           0x80
                XORWF           SIGN
                GOTO            NRM24

                END
```

APPENDIX B

```
;       PIC17 24 BIT FLOATING POINT LIBRARY     VERSION 1.14

;       Unary operations: both input and output are in AEXP,AARG

;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       Max timings are worst case cycle counts, while Min timings are non-exception
;       best case cycle counts.

;         Routine               Function

;       FLO24           16 bit integer to 24 bit floating point conversion

;       Max Timing:                 RND
;                               0       1

;                       0       48      48
;                  SAT
;                       1       55      55

;       NRM24           normalization of unnormalized 24 bit floating point numbers

;       Max Timing:                 RND
;                               0       1

;                       0       39      39
;                  SAT
;                       1       46      46


;       INT24           24 bit floating point to 16 bit integer conversion


;       Max Timing:                 RND
;                               0       1

;                       0       57      61
;                  SAT
;                       1       57      65

;       FPA24           24 bit floating point add

;       Max Timing:                 RND
;                               0       1

;                       0       116     131
;                  SAT
;                       1       116     131

;       FPS24           24 bit floating point subtract

;       Max Timing:                 RND
;                               0       1

;                       0       117     132
;                  SAT
;                       1       117     132

;       FPM24           24 bit floating point multiply

;       Max Timing:                 RND
;                               0       1

;                       0       175     184
;                   SAT
;                        1       175     189

;       FPD24           24 bit floating point divide

;       Max Timing:                RND
;                               0       1

;                        0       328     347
;                   SAT
;                        1       328     352


                list    r=dec,x=on,t=off,p=17C42

                include <PIC17.INC>

;***********************************************************************************************
;***********************************************************************************************

;       24 bit floating point representation

;       EXPONENT        8 bit biased exponent
;                       It is important to note that the use of biased exponents produces
;                       a unique representation of a floating point 0, given by
;                       EXP = HIGHBYTE = LOWBYTE = 0x00, with 0 being the only
;                       number with EXP = 0.

;       HIGHBYTE        8 bit most significant byte of sign-magnitude representation, with
;                       SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                       left of MSB

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                       RADIX
;       EXPONENT        POINT   HIGHBYTE        LOWBYTE

;       xxxxxxxx           .    Sxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS         equ     128

;       Define library register variables

EXP             equ     0x18    ; 8 bit biased exponent

ACC             equ     0x19    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x1F    ; save location for sign in MSB

TEMP            equ     0x25    ; temporary storage

;       Define binary operation arguments

AEXP            equ     0x18    ; 8 bit biased exponent for argument A

AARG            equ     0x19    ; most significant byte of mantissa for argument A

BEXP            equ     0x21    ; 8 bit biased exponent for argument B

BARG            equ     0x22    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1 )  and  (EXP, ACC+B0, ACC+B1)
;               reference the same storage locations.

;       Define floating point library exception flags
```

5

```
FPFLAGS         equ     0x20    ; floating point library exception flags

IOV             equ     0       ; bit0 = integer overflow flag

FOV             equ     1       ; bit1 = floating point overflow flag

FUN             equ     2       ; bit2 = floating point underflow flag

FDZ             equ     3       ; bit3 = floating point divide by zero flag

RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = unbiased rounding to nearest LSB

SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;**********************************************************************************************
;**********************************************************************************************

;       Integer to float conversion

;       Input:  16 bit 2’s complement integer right justified in AARG+B0, AARG+B1

;       Use:    CALL    FLO24

;       Output: 24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  FLOAT( AARG )

;       Max Timing:     9+39 = 48 clks          SAT = 0
;                                       9+45 = 54 clks          SAT = 1

;       Min Timing:     6+15 = 21 clks          AARG = 0
;                                       6+20 = 26 clks

;       PM: 9+63 = 72                   DM: 6

FLO24           MOVLW           16+EXPBIAS          ; initialize exponent and add bias
                MOVWF           EXP
                MOVFP           ACC+B0,SIGN         ; save sign in SIGN
                BTFSS           ACC+B0,MSB          ; test sign
                GOTO            NRM24
                COMF            ACC+B1              ; if < 0, negate, set MSB in SIGN
                COMF            ACC+B0
                INFSNZ  ACC+B1
                INCF            ACC+B0

;——————————————————————————————————

;       Normalization routine

;       Input:  24 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               with sign in SIGN,MSB.

;       Use:    CALL    NRM24

;       Output: 24 bit normalized floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  NORMALIZE( AARG )

;       Max Timing:     3+12+16+8 = 39 clks             SAT = 0
;                                       3+12+16+15 = 46 clks            SAT = 1

;       Min Timing:     10+5 = 15 clks          AARG = 0
;                                       3+5+4+8 = 20 clks

;       PM: 63                                  DM: 6
```

```
NRM24           CLRF            TEMP,W              ; clear exponent decrement
                CPFSGT          ACC+B0              ; test if highbyte=0
                GOTO            NRM24A

TNIB24          MOVLW           0xF0                ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM24
                SWAPF           ACC+B0              ; if so, shift 4 bits
                SWAPF           ACC+B1,W
                ANDLW           0x0F
                ADDWF           ACC+B0
                SWAPF           ACC+B1,W
                ANDLW           0xF0
                MOVPF           WREG,ACC+B1

                BSF             TEMP,2              ; increase decrement by 4

NORM24          BCF             _C                  ; clear carry bit

                BTFSC           ACC+B0,MSB          ; if MSB=1, normalization done
                GOTO            TNORMUN24
                RLCF            ACC+B1              ; otherwise, shift left and
                RLCF            ACC+B0              ; increment decrement
                INCF            TEMP
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN24
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB          ; since highnibble != 0, at most
                GOTO            TNORMUN24           ; 3 left shifts are required
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP

TNORMUN24       MOVFP           TEMP,WREG           ; if EXP <= decrement in TEMP,
                CPFSGT          EXP                 ; floating point underflow has
                GOTO            SETFUN24            ; occured
                SUBWF           EXP                 ; otherwise, compute EXP

FIXSIGN24       BTFSS           SIGN,MSB
                BCF             ACC+B0,MSB          ; clear explicit MSB if positive
                RETLW           0

NRM24A          MOVFP           ACC+B1,ACC+B0       ; if so, shift 8 bits by move
                CLRF            ACC+B1
                BSF             TEMP,3              ; increase decrement by 8
                CPFSGT          ACC+B0              ; if highbyte=0, result=0
                GOTO            RES024

                MOVLW           0xF0                ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM24A
                SWAPF           ACC+B0              ; if so, shift 4 bits

                BSF             TEMP,2              ; increase decrement by 4

NORM24A         BCF             _C                  ; clear carry bit

                BTFSC           ACC+B0,MSB          ; if MSB=1, normalization done
                GOTO            TNORMUN24
                RLCF            ACC+B0              ; otherwise, shift left and
                INCF            TEMP                ; increment decrement
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN24
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB          ; since highnibble != 0, at most
                GOTO            TNORMUN24           ; 3 left shifts are required
                RLCF            ACC+B0
                INCF            TEMP
                GOTO            TNORMUN24


;**********************************************************************************************
;**********************************************************************************************

;       Float to integer conversion

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Use:    CALL    INT24

;       Output: 16 bit 2’s complement integer right justified in AARG+B0, AARG+B1

;       Result: AARG  <—  INT( AARG )

;       Max Timing:     11+34+12 = 57 clks              RND = 0
;                               11+34+16 = 61 clks              RND = 1, SAT = 0
;                               11+34+20 = 65 clks              RND = 1, SAT = 1

;       Min Timing:     6+5 = 11 clks           AARG = 0
;                               11+10+7 = 28 clks

;       PM: 101                                 DM: 6

INT24           MOVFP           ACC+B0,SIGN         ; save sign in SIGN
                BSF             ACC+B0,MSB          ; make MSB explicit

                MOVLW           EXPBIAS             ; remove bias from EXP
                CPFSGT          EXP                 ; if <= 0, result=0
                GOTO            RES024
                SUBWF           EXP

                MOVLW           16
                CPFSLT          EXP                 ; if >= 16, integer overflow
                GOTO            SETIOV24            ; will occur
                SUBWF           EXP,W
                NEGW            EXP

                MOVLW           7                   ; do byte shift if EXP >= 8
                CPFSGT          EXP
                GOTO            SNIB24
                SUBWF           EXP                 ; EXP = EXP - 7
                RLCF            ACC+B1              ; rotate next bit for rounding
                MOVFP           ACC+B0,ACC+B1
                CLRF            ACC+B0
                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT24OK           ; shift completed if EXP = 0

SNIB24A         MOVLW           3                   ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT24A
                SUBWF           EXP                 ; EXP = EXP - 3
                SWAPF           ACC+B1,W
                MOVFP           WREG,ACC+B2         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B1
                RLCF            ACC+B2              ; rotate next bit for rounding
                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT24OK           ; shift completed if EXP = 0

SHIFT24A        BCF             _C                  ; at most 3 right shifts are required
                RRCF            ACC+B1              ; right shift by EXP
                DCFSNZ          EXP
                GOTO            SHIFT24OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B1
                DCFSNZ          EXP
                GOTO            SHIFT24OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B1
                GOTO            SHIFT24OK

SNIB24          MOVLW           3                   ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT24
                SUBWF           EXP                 ; EXP = EXP - 3
                SWAPF           ACC+B1,W
                MOVFP           WREG,ACC+B2         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B1
                SWAPF           ACC+B0,W
                ANDLW           0xF0
                ADDWF           ACC+B1
                SWAPF           ACC+B0,W
                ANDLW           0x0F
                MOVPF           WREG,ACC+B0
                RLCF            ACC+B2              ; rotate next bit for rounding
                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT24OK           ; shift completed if EXP = 0

SHIFT24         BCF             _C                  ; at most 3 right shifts are required
                RRCF            ACC+B0              ; right shift by EXP
                RRCF            ACC+B1
                DCFSNZ          EXP
                GOTO            SHIFT24OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1
                DCFSNZ          EXP
                GOTO            SHIFT24OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1

SHIFT24OK       BTFSC           FPFLAGS,RND
                BTFSS           ACC+B1,LSB
                GOTO            INT24OK
                CLRF            WREG
                ADDWFC          ACC+B1              ; add next bit for rounding
                ADDWFC          ACC+B0
                BTFSC           ACC+B0,MSB
                GOTO            SETIOV24

INT24OK         BTFSS           SIGN,MSB            ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B1
                COMF            ACC+B0
                INFSNZ          ACC+B1
                INCF            ACC+B0
                RETLW           0

RES024          CLRF            ACC+B0              ; integer result equals zero
                CLRF            ACC+B1
                CLRF            EXP                 ; clear EXP for other routines
                RETLW           0

SETIOV24        BSF             FPFLAGS,IOV         ; set integer overflow flag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                CLRF            ACC+B0              ; saturate to largest two's
                BTFSS           SIGN,MSB            ; complement 16 bit integer
                SETF            ACC+B0              ; SIGN = 0, 0x 7F FF
```

5

```
                MOVPF           ACC+B0,ACC+B1   ; SIGN = 1, 0x 80 00
                RLCF            SIGN
                RRCF            ACC+B0
                RETLW           0xFF            ; return error code in WREG

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Multiply

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPM24

;       Output: 24 bit floating point product in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG * BARG

;       Max Timing:     19+140+16 = 175 clks            RND = 0
;                                       19+140+25 = 184 clks            RND = 1, SAT = 0
;                                       19+140+30 = 189 clks            RND = 1, SAT = 1

;       Min Timing:     5+5 = 10 clks           AARG * BARG = 0
;                                       13+5+37+15 = 70 clks

;       PM: 21+169+34 = 224                     DM: 10

MUL16           macro

;       Max Timing:     10+14*9+4 = 140 clks
;       Min Timing:     33+4 = 37 clks


;       PM: 3+16*2+4+14*9+4 = 169       DM: 6

                variable i = 0
                variable j = 1

                BCF             _C                  ; clear carry bit

                CLRF            ACC+B2              ; clear initial partial product
                CLRF            ACC+B3

                while   i < 16

                if      i < 8

                BTFSC           AARG+B1,i           ; test low byte

                else

                BTFSC           AARG+B0,i-8         ; test high byte

                endif

                GOTO            Madd24#v( i )

                i += 1

                endw

;       we cannot get here since MSB’s are explicit

Madd240         MOVFP           BARG+B0,ACC+B2 ; if we get here, ACC=0 so move
                MOVFP           BARG+B1,ACC+B3  ; BARG for add

                RRCF            ACC+B2
                RRCF            ACC+B3
                while j < 15

                if j < 8

                BTFSS           AARG+B1,j           ; test low byte

                else

                BTFSS           AARG+B0,j-8         ; test high byte

                endif

                GOTO            Mnoadd24#v( j )

Madd24#v( j )   MOVFP           BARG+B1,WREG
                ADDWF           ACC+B3              ; add lsb
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B2              ; add msb

Mnoadd24#v( j ) RRCF            ACC+B2
                RRCF            ACC+B3

                if j > 13

                RRCF            ACC+B1              ; save bits for rounding

                endif

                if j < 14

                BCF             _C                  ; clear carry when rotate possible

                endif

                j += 1

                endw

Madd2415        MOVFP           BARG+B1,WREG
                ADDWF           ACC+B3              ; add lsb
                MOVFP           BARG+B0,WREG
                ADDWFC           ACC+B2             ; add msb

                endm

FPM24           CLRF            WREG                ; test for zero arguments
                CPFSEQ          BEXP
                CPFSGT          AEXP
                GOTO            RES024

M24BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVPF           WREG,SIGN           ; save sign in SIGN

                MOVFP           BEXP,WREG
                ADDWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            MTUN24

                ADDWF           EXP                 ; remove bias and overflow test
                BTFSC           _C
                GOTO            SETFOV24
                GOTO            MOK24

MTUN24          ADDWF           EXP                 ; remove bias and underflow test
                BTFSS           _C
```

5

```
                GOTO            SETFUN24

MOK24           BSF             AARG+B0,MSB         ; make argument MSB’s explicit
                BSF             BARG+B0,MSB

                MUL16                               ; use macro for multiplication

                BTFSC           _C                  ; check for postnormalization
                GOTO            MRIGHT24
                DECF            EXP
                GOTO            MROUND24
MRIGHT24        RRCF            ACC+B2
                RRCF            ACC+B3
                RRCF            ACC+B1

MROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            MUL24OK
                CLRF            WREG
                RLCF            ACC+B1              ; rotate next significant bit into
                ADDWFC          ACC+B3              ; carry for rounding
                ADDWFC          ACC+B2

                BTFSS           _C                  ; has rounding caused carryout?
                GOTO            MUL24OK
                RRCF            ACC+B2              ; if so, right shift
                RRCF            ACC+B3
                INFSNZ          EXP                 ; test for floating point overflow
                GOTO            SETFOV24

MUL24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVFP           ACC+B2,AARG+B0      ; move result to AARG
                MOVFP           ACC+B3,AARG+B1
                RETLW           0

SETFOV24        BSF             FPFLAGS,FOV         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                SETF            AEXP                ; saturate to largest floating
                SETF            AARG+B0             ; point number = 0x FF 7F FF
                SETF            AARG+B1             ; modulo the appropriate sign bit
                RLCF            SIGN
                RRCF            AARG+B0
                RETLW           0xFF                ; return error code in WREG


;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Divide

;       Input:  24 bit floating point dividend in AEXP, AARG+B0, AARG+B1
;               24 bit floating point divisor in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPD24

;       Output: 24 bit floating point quotient in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG / BARG

;       Max Timing:     22+11+285+10 = 328 clks RND = 0
;                                   22+11+285+29 = 347 clks RND = 1, SAT = 0
;                                   22+11+285+34 = 352 clks RND = 1, SAT = 1

;       Min Timing: 6+5 = 11 clks               AARG = 0
;                                   30+237+10 = 277 clks
;       PM: 36+301+40 = 377                     DM: 11

DIV16           macro

;       Timing: 1+16*20 = 321 clks

                variable i = 0

                BCF             _C

                while i < 16

                CLRF            TEMP                    ; left shift
                RLCF            ACC+B3
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP

                MOVFP           BARG+B1,WREG            ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                CLRF            WREG
                SUBWFB          TEMP

                BTFSS           _C                      ; test for restore
                GOTO            Drest#v( i )

                BSF             ACC+B3,LSB
                GOTO            Dok#v( i )

Drest#v( i )    MOVFP           BARG+B1,WREG                    ; restore if necessary
                ADDWF           ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B3,LSB

Dok#v( i )

                i += 1

                endw

                endm


DIV16A          macro

;       Max Timing:     15+15*18 = 285 clks
;       Min Timing:     12+15*15 = 237 clks

;       PM: 16+15*19 = 301                      DM: 7

                BTG             TEMP,LSB                ; save carry

                MOVFP           BARG+B1,WREG            ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB                ; test for restore
                GOTO            DArest0

                BSF             ACC+B2,MSB
                GOTO            DAok0
```

```
DArest0         MOVFP           BARG+B1,WREG         ; restore if necessary
                ADDWF           ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B2,MSB

DAok0


                variable i = 1

                while i < 16

                BCF             _C
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP                 ; save carry

                MOVFP           BARG+B1,WREG         ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB             ; test for restore
                GOTO            DArest#v( i )

                if i < 8

                BSF             ACC+B2,MSB-i         ; set bit in quotient

                else

                BSF             ACC+B3,15-i          ; set bit in quotient

                endif

                GOTO            DAok#v( i )

DArest#v( i )   MOVFP           BARG+B1,WREG         ; restore if necessary
                ADDWF           ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0


                if i < 8

                BCF             ACC+B2,MSB-i         ; clear bit in quotient

                else

                BCF             ACC+B3,15-i          ; clear bit in quotient

                endif

DAok#v( i )

                i += 1

                endw

                endm

FPD24           CLRF            TEMP,W               ; test for divide by zero
                CPFSGT          BEXP
                GOTO            SETFDZ24
                CPFSGT          AEXP
                GOTO            RES024

D24BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVPF           WREG,SIGN           ; save sign in SIGN
                BSF             AARG+B0,MSB         ; make argument MSB's explicit
                BSF             BARG+B0,MSB

TALIGN24        INCF            TEMP                ; set align increment
                MOVFP           AARG+B0,ACC+B2      ; test for alignment
                MOVFP           AARG+B1,ACC+B3
                MOVFP           BARG+B1,WREG
                SUBWF           ACC+B3
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B2
                BTFSC           _C
                GOTO            DALIGN24OK

                RLCF            ACC+B1              ; align if necessary
                RLCF            ACC+B0
                DECF            TEMP                ; fix align increment

DALIGN24OK      MOVFP           BEXP,WREG           ; compute AEXP - BEXP
                SUBWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            ALTB24

AGEB24          ADDWF           TEMP,W              ; if AEXP > BEXP, test for
                ADDWF           EXP                 ; overflow
                BTFSC           _C
                GOTO            SETFOV24
                GOTO            DARGOK24

ALTB24          ADDWF           TEMP,W              ; if AEXP < BEXP, test for
                ADDWF           EXP                 ; underflow
                BTFSS           _C
                GOTO            SETFUN24

DARGOK24        DIV16A                              ; macro for division

DROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            DIV24OK
                BCF             _C                  ; compute next significant bit
                RLCF            ACC+B1              ; for rounding
                RLCF            ACC+B0
                RLCF            TEMP                ; save carry

                MOVFP           BARG+B1,WREG        ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                RRCF            TEMP                ; rotate next significant bit into
                CLRF            WREG                ; carry and add for rounding
                ADDWFC          ACC+B3
                ADDWFC          ACC+B2

                BTFSS           _C                  ; test if rounding caused carryout
                GOTO            DIV24OK
                RRCF            ACC+B2
                RRCF            ACC+B3
                INFSNZ          EXP                 ; test for overflow
                GOTO            SETFOV24
```

```
DIV24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVFP           ACC+B2,AARG+B0      ; move result to AARG
                MOVFP           ACC+B3,AARG+B1

                RETLW           0

SETFUN24        BSF             FPFLAGS,FUN         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0x01                ; saturate to smallest floating
                MOVPF           WREG,AEXP           ; point number = 0x 01 00 00
                CLRF            AARG+B0             ; modulo the appropriate sign bit
                CLRF            AARG+B1
                RLCF            SIGN
                RRCF            AARG+B0
                RETLW           0xFF                ; return error code in WREG

SETFDZ24        BSF             FPFLAGS,FDZ         ; set floating point divide by zero
                RETLW           0xFF                ; flag and return error code in
        ; WREG
;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Subtract

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPS24

;       Output: 24 bit floating point difference in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     1+116 = 117 clks                RND = 0
;                                       1+131 = 132 clks                RND = 1

;       Min Timing:     1+31 = 32 clks

;       PM: 1+212 = 213         DM: 10

FPS24           BTG             BARG+B0,MSB         ; toggle sign bit for subtraction

;—————————————————————————————————

;       Floating Point Add

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPA24

;       Output: 24 bit floating point sum in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     60+17+39 = 116 clks             RND = 0
;                                       60+25+46 = 131 clks             RND = 1

;       Min Timing:     27+4 = 31 clks

;       PM: 212                                 DM: 10

FPA24           MOVFP           AARG+B0,WREG        ; exclusive or of signs in TEMP
                XORWF           BARG+B0,W
                MOVPF           WREG,TEMP
                MOVFP           AEXP,WREG           ; use AARG if AEXP >= BEXP
                CPFSGT          BEXP
                GOTO            USEA24

USEB24          MOVFP           BARG+B0,SIGN        ; save sign in SIGN
                BSF             BARG+B0,MSB         ; make MSB’s explicit
                BSF             AARG+B0,MSB

                MOVFP           AEXP,WREG           ; compute shift count in BEXP
                MOVFP           BEXP,AEXP
                SUBWF           BEXP
                BTFSC           _Z
                GOTO            BLIGNED24

                MOVLW           7
                CPFSGT          BEXP                ; do byte shift if BEXP >= 8
                GOTO            BNIB24

                SUBWF           BEXP                ; BEXP = BEXP - 7
                RLCF            AARG+B1             ; rotate next bit for rounding
                MOVFP           AARG+B0,AARG+B1
                CLRF            AARG+B0
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            BROUND24

                CPFSGT          BEXP                ; if BEXP still >= 8, then
                GOTO            BNIB24A             ; AARG = 0 relative to BARG

                MOVFP           SIGN,AARG+B0        ; return BARG
                MOVFP           BARG+B1,AARG+B1
                RETLW           0x00

BNIB24A         MOVLW           3                   ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            BLOOP24A
                SUBWF           BEXP                ; BEXP = BEXP -3
                SWAPF           AARG+B1,W
                MOVPF           WREG,ACC+B4         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,AARG+B1
                RLCF            ACC+B4              ; rotate next bit for rounding
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            BROUND24            ; aligned if BEXP = 0

BLOOP24A        BCF             _C                  ; right shift by BEXP
                RRCF            AARG+B1
                DCFSNZ          BEXP
                GOTO            BROUND24            ; aligned if BEXP = 0
                BCF             _C
                RRCF            AARG+B1
                DCFSNZ          BEXP
                GOTO            BROUND24            ; aligned if BEXP = 0
                BCF             _C                  ; at most 3 right shifts are
                RRCF            AARG+B1             ; possible
                GOTO            BROUND24

BNIB24          MOVLW           3                   ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            BLOOP24
                SUBWF           BEXP                ; BEXP = BEXP -3
                SWAPF           AARG+B1,W
                MOVPF           WREG,ACC+B4         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,AARG+B1
                SWAPF           AARG+B0,W
                ANDLW           0xF0
                ADDWF           AARG+B1
                SWAPF           AARG+B0,W
                ANDLW           0x0F
                MOVPF           WREG,AARG+B0
                RLCF            ACC+B4                  ; rotate next bit for rounding
                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            BROUND24                ; aligned if BEXP = 0

BLOOP24         BCF             _C                      ; right shift by BEXP
                RRCF            AARG+B0
                RRCF            AARG+B1
                DCFSNZ          BEXP
                GOTO            BROUND24                ; aligned if BEXP = 0
                BCF             _C
                RRCF            AARG+B0
                RRCF            AARG+B1
                DCFSNZ          BEXP
                GOTO            BROUND24                ; aligned if BEXP = 0
                BCF             _C                      ; at most 3 right shifts are
                RRCF            AARG+B0                 ; possible
                RRCF            AARG+B1

BROUND24        BTFSC           FPFLAGS,RND
                BTFSS           AARG+B1,LSB
                GOTO            BLIGNED24
                CLRF            WREG
                ADDWFC          AARG+B1                 ; add carry for rounding
                ADDWFC          AARG+B0

                BTFSS           _C                      ; has rounding caused carryout?
                GOTO            BLIGNED24
                RRCF            AARG+B0                 ; if so, right shift
                RRCF            AARG+B1
                INFSNZ          EXP                     ; test for floating point overflow
                GOTO            SETFOV24

BLIGNED24       BTFSS           TEMP,MSB                ; negate if signs opposite
                GOTO            AOK24
                COMF            AARG+B1
                COMF            AARG+B0
                INFSNZ          AARG+B1
                INCF            AARG+B0
                GOTO            AOK24


USEA24          MOVFP           AARG+B0,SIGN            ; save sign in SIGN
                BSF             AARG+B0,MSB             ; make MSB’s explicit
                BSF             BARG+B0,MSB

                MOVFP           BEXP,WREG               ; compute shift count in BEXP
                SUBWF           AEXP,W
                MOVPF           WREG,BEXP
                BTFSC           _Z
                GOTO            ALIGNED24

                MOVLW           7
                CPFSGT          BEXP                    ; do byte shift if BEXP >= 8
                GOTO            ANIB24

                SUBWF           BEXP                    ; BEXP = BEXP - 7
                RLCF            BARG+B1                 ; rotate next bit for rounding
                MOVFP           BARG+B0,WREG
                MOVPF           WREG,BARG+B1
                CLRF            BARG+B0
                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            AROUND24

                CPFSGT          BEXP                    ; if BEXP still >= 8, then
                GOTO            ANIB24A                 ; BARG = 0 relative to AARG

                MOVFP           SIGN,AARG+B0            ; return AARG
                RETLW           0x00

ANIB24A         MOVLW           3                   ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP24A
                SUBWF           BEXP                ; BEXP = BEXP -3
                SWAPF           BARG+B1,W
                MOVPF           WREG,ACC+B4         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B1
                RLCF            ACC+B4              ; rotate next bit for rounding
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            AROUND24            ; aligned if BEXP = 0

ALOOP24A        BCF             _C                  ; right shift by BEXP
                RRCF            BARG+B1
                DCFSNZ          BEXP
                GOTO            AROUND24            ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B1
                DCFSNZ          BEXP
                GOTO            AROUND24            ; aligned if BEXP = 0
                BCF             _C                  ; at most 3 right shifts are
                RRCF            BARG+B1             ; possible
                GOTO            AROUND24

ANIB24          MOVLW           3                   ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP24
                SUBWF           BEXP                ; BEXP = BEXP -3
                SWAPF           BARG+B1,W
                MOVPF           WREG,ACC+B4         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B1
                SWAPF           BARG+B0,W
                ANDLW           0xF0
                ADDWF           BARG+B1
                SWAPF           BARG+B0,W
                ANDLW           0x0F
                MOVPF           WREG,BARG+B0
                RLCF            ACC+B4              ; rotate next bit for rounding
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            AROUND24            ; aligned if BEXP = 0

ALOOP24         BCF             _C                  ; right shift by BEXP
                RRCF            BARG+B0
                RRCF            BARG+B1
                DCFSNZ          BEXP
                GOTO            AROUND24            ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B0
                RRCF            BARG+B1
                DCFSNZ          BEXP
                GOTO            AROUND24            ; aligned if BEXP = 0
                BCF             _C                  ; at most 3 right shifts are
                RRCF            BARG+B0             ; possible
                RRCF            BARG+B1

AROUND24        BTFSC           FPFLAGS,RND
                BTFSS           BARG+B1,LSB
                GOTO            ALIGNED24
                CLRF            WREG
                ADDWFC          BARG+B1             ; add carry for rounding
                ADDWFC          BARG+B0

                BTFSS           _C                  ; has rounding caused carryout?
                GOTO            ALIGNED24
                RRCF            BARG+B0             ; if so, right shift
                RRCF            BARG+B1
```

5

```
                INFSNZ          EXP                     ; test for floating point overflow
                GOTO            SETFOV24

ALIGNED24       BTFSS           TEMP,MSB                ; negate if signs opposite
                GOTO            AOK24
                COMF            BARG+B1
                COMF            BARG+B0
                INFSNZ          BARG+B1
                INCF            BARG+B0

AOK24           MOVFP           BARG+B1,WREG            ; add
                ADDWF           AARG+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          AARG+B0

                BTFSC           TEMP,MSB
                GOTO            ACOMP24
                BTFSS           _C
                GOTO            FIXSIGN24

                RRCF            AARG+B0                 ; shift right and increment EXP
                RRCF            AARG+B1
                INCFSZ          AEXP
                GOTO            FIXSIGN24
                GOTO            SETFOV24                ; set floating point overflow flag

ACOMP24         BTFSC           _C
                GOTO            NRM24                   ; normalize and fix sign

                COMF            AARG+B1                 ; negate, toggle sign bit and
                COMF            AARG+B0                 ; then normalize
                INFSNZ          AARG+B1
                INCF            AARG+B0
                BTG             SIGN,MSB
                GOTO            NRM24

                END
```

APPENDIX C

```
;       PIC16 32 BIT FLOATING POINT LIBRARY     VERSION 1.14

;       Unary operations: both input and output are in AEXP,AARG

;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       All timings are worst case cycle counts

;         Routine                Function

;       FLO32           24 bit integer to 32 bit floating point conversion

;               Timing:            RND
;                               0       1

;                       0       104     104
;                  SAT
;                       1       110     110

;       NRM32           normalization of unnormalized 32 bit floating point numbers

;               Timing:            RND
;                               0       1

;                       0       90      90
;                  SAT
;                       1       96      96


;       INT32           32 bit floating point to 24 bit integer conversion


;               Timing:            RND
;                               0       1

;                       0       90      98
;                  SAT
;                       1       90      100

;       FPA32           32 bit floating point add

;               Timing:            RND
;                               0       1

;                       0       248     262
;                  SAT
;                       1       248     268

;       FPS32           32 bit floating point subtract

;               Timing:            RND
;                               0       1

;                       0       250     264
;                  SAT
;                       1       250     270

;       FPM32           32 bit floating point multiply

;               Timing:            RND
;                               0       1

;                       0       574     588
;                 SAT
;                      1       574     591

;       FPD32          32 bit floating point divide

;               Timing:            RND
;                              0       1

;                      0       929     965
;                 SAT
;                      1       929     968

                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;**********************************************************************************************
;**********************************************************************************************

;       32 bit floating point representation

;       EXPONENT        8 bit biased exponent
;                       It is important to note that the use of biased exponents produces
;                       a unique representation of a floating point 0, given by
;                       EXP = HIGHBYTE = MIDBYTE = LOWBYTE = 0x00, with 0 being
;                       the only number with EXP = 0.

;       HIGHBYTE        8 bit most significant byte of sign-magnitude representation, with
;                       SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                       left of MSB

;       MIDBYTE 8 bit middle significant byte of sign-magnitude matissa

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                       RADIX
;       EXPONENT        POINT   HIGHBYTE        MIDBYTE LOWBYTE

;       xxxxxxxx           .    Sxxxxxxx        xxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS         equ     128

;       Define library register variables

EXP             equ     0x0C    ; 8 bit biased exponent

ACC             equ     0x0D    ; most significant byte of contiguous 6 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AEXP            equ     0x0C    ; 8 bit biased exponent for argument A

AARG            equ     0x0D    ; most significant byte of mantissa for argument A

BEXP            equ     0x15    ; 8 bit biased exponent for argument B

BARG            equ     0x16    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1, AARG+B2 )  and
;               (EXP, ACC+B0, ACC+B1, ACC+B2 ) reference the same storage locations.
;       Define floating point library exception flags

FPFLAGS         equ     0x14    ; floating point library exception flags

IOV             equ     0       ; bit0 = integer overflow flag

FOV             equ     1       ; bit1 = floating point overflow flag

FUN             equ     2       ; bit2 = floating point underflow flag

FDZ             equ     3       ; bit3 = floating point divide by zero flag

RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = rounding to nearest LSB

SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value


;**********************************************************************************************
;**********************************************************************************************

;       Integer to float conversion

;       Input:  24 bit 2’s complement integer right justified in AARG+B0, AARG+B1,
;               AARG+B2

;       Use:    CALL    FLO32

;       Output: 32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  FLOAT( AARG )

;       Max Timing:     14+90 = 104 clks                SAT = 0
;                                       14+96 = 110 clks                SAT = 1

;       Min Timing:     6+28 = 34 clks          AARG = 0
;                                       6+18 = 24 clks

;       PM: 14+38 = 52                  DM: 7

FLO32           MOVLW           24+EXPBIAS          ; initialize exponent and add bias
                MOVWF           EXP
                CLRF            SIGN
                BTFSS           ACC+B0,MSB          ; test sign
                GOTO            NRM32
                COMF            ACC+B2              ; if < 0, negate and set MSB in SIGN
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B2
                BTFSC           _Z
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                BSF             SIGN,MSB

;—————————————————————————————————

;       Normalization routine

;       Input:  32 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2, with sign in SIGN,MSB

;       Use:    CALL    NRM32

;       Output: 32 bit normalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2
```

5

```
;       Result: AARG  <—  NORMALIZE( AARG )

;       Max Timing:     21+6+7*8+7 = 90 clks            SAT = 0
;                                       21+6+7*8+1+12 = 96 clks SAT = 1

;       Min Timing: 22+6 = 28 clks              AARG = 0
;                                       5+9+4 = 18 clks

;       PM: 38                                  DM: 7

NRM32           CLRF            TEMP                    ; clear exponent decrement
                MOVF            ACC+B0,W                ; test if highbyte=0
                BTFSS           _Z
                GOTO            NORM32
                MOVF            ACC+B1,W                ; if so, shift 8 bits by move
                MOVWF           ACC+B0
                MOVF            ACC+B2,W
                MOVWF           ACC+B1
                CLRF            ACC+B2
                BSF             TEMP,3                  ; increase decrement by 8

                MOVF            ACC+B0,W                ; test if highbyte=0
                BTFSS           _Z
                GOTO            NORM32
                MOVF            ACC+B1,W                ; if so, shift 8 bits by move
                MOVWF           ACC+B0
                CLRF            ACC+B1
                BCF             TEMP,3                  ; increase decrement by 8
                BSF             TEMP,4

                MOVF            ACC+B0,W                ; if highbyte=0, result=0
                BTFSC           _Z
                GOTO            RES032

NORM32          MOVF            TEMP,W
                SUBWF           EXP
                BTFSS           _Z
                BTFSS           _C
                GOTO            SETFUN32

                BCF             _C                      ; clear carry bit

NORM32A         BTFSC           ACC+B0,MSB              ; if MSB=1, normalization done
                GOTO            FIXSIGN32
                RLF             ACC+B2                  ; otherwise, shift left and
                RLF             ACC+B1                  ; decrement EXP
                RLF             ACC+B0
                DECFSZ          EXP
                GOTO            NORM32A

                GOTO            SETFUN32                ; underflow if EXP=0

FIXSIGN32       BTFSS           SIGN,MSB
                BCF             ACC+B0,MSB              ; clear explicit MSB if positive
                RETLW           0

;**********************************************************************************************
;**********************************************************************************************

;       Float to integer conversion

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Use:    CALL    INT32

;       Output: 24 bit 2’s complement integer right justified in AARG+B0, AARG+B1,
;               AARG+B2
;       Result: AARG  <—  INT( AARG )

;       Max Timing:     26+6*7+6+16 = 90 clks           RND = 0
;                                   26+6*7+6+24 = 98 clks           RND = 1, SAT = 0
;                                   26+6*7+6+26 = 100 clks          RND = 1, SAT = 1

;       Min Timing:     9+6 = 15 clks           AARG = 0
;                                   19+6+7 = 32 clks

;       PM: 83                                  DM: 6

INT32           MOVF            ACC+B0,W                ; save sign in SIGN
                MOVWF           SIGN
                BSF             ACC+B0,MSB              ; make MSB explicit

                MOVLW           EXPBIAS                 ; remove bias from EXP
                SUBWF           EXP
                BTFSS           EXP,MSB                 ; if <= 0, result=0
                BTFSC           _Z
                GOTO            RES032

                MOVF            EXP,W
                SUBLW           24
                MOVWF           EXP
                BTFSS           _Z
                BTFSC           EXP,MSB
                GOTO            SETIOV32

                MOVLW           8                       ; do byte shift if EXP >= 8
                SUBWF           EXP,W
                BTFSS           _C
                GOTO            SHIFT32
                MOVWF           EXP
                RLF             ACC+B2                  ; rotate next bit for rounding
                MOVF            ACC+B1,W
                MOVWF           ACC+B2
                MOVF            ACC+B0,W
                MOVWF           ACC+B1
                CLRF            ACC+B0

                MOVLW           8                       ; do another byte shift if EXP >= 8
                SUBWF           EXP,W
                BTFSS           _C
                GOTO            SHIFT32
                MOVWF           EXP
                RLF             ACC+B2                  ; rotate next bit for rounding
                MOVF            ACC+B1,W
                MOVWF           ACC+B2
                CLRF            ACC+B1

                MOVF            EXP,W                   ; shift completed if EXP = 0
                BTFSC           _Z
                GOTO            SHIFT32OK

SHIFT32         BCF             _C
                RRF             ACC+B0                  ; right shift by EXP
                RRF             ACC+B1
                RRF             ACC+B2
                DECFSZ          EXP
                GOTO            SHIFT32

SHIFT32OK       BTFSC           FPFLAGS,RND
                BTFSS           ACC+B2,LSB
                GOTO            INT32OK
                BTFSS           _C
                GOTO            INT32OK
                INCF            ACC+B2
                BTFSC           _Z
                INCF            ACC+B1
```

5

```
                BTFSC           _Z
                INCF            ACC+B0
                BTFSC           ACC+B0,MSB          ; test for overflow
                GOTO            SETIOV32

INT32OK         BTFSS           SIGN,MSB            ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B0
                COMF            ACC+B1
                COMF            ACC+B2
                INCF            ACC+B2
                BTFSC           _Z
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                RETLW           0

RES032          CLRF            ACC+B0              ; integer result equals zero
                CLRF            ACC+B1
                CLRF            ACC+B2
                CLRF            EXP                 ; clear EXP for other routines
                RETLW           0

SETIOV32        BSF             FPFLAGS,IOV         ; set integer overflow flag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                CLRF            ACC+B0              ; saturate to largest two’s
                BTFSS           SIGN,MSB            ; complement 16 bit integer
                MOVLW           0xFF
                MOVWF           ACC+B0              ; SIGN = 0, 0x 7F FF FF
                MOVWF           ACC+B1              ; SIGN = 1, 0x 80 00 00
                MOVWF           ACC+B2
                RLF             SIGN
                RRF             ACC+B0
                RETLW           0xFF                ; return error code in WREG


;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Multiply

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPM32

;       Output: 32 bit floating point product in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG * BARG

;       Max Timing:     26+23*22+21+21 = 574 clks       RND = 0
;                               26+23*22+21+35 = 588 clks       RND = 1, SAT = 0
;                               26+23*22+21+38 = 591 clks       RND = 1, SAT = 1

;       Min Timing:     6+6 = 12 clks           AARG * BARG = 0
;                               24+23*11+21+17 = 315 clks

;       PM: 94                          DM: 14

FPM32           MOVF            AEXP,W              ; test for zero arguments
                BTFSS           _Z
                MOVF            BEXP,W
                BTFSC           _Z
                GOTO            RES032

M32BNE0         MOVF            AARG+B0,W
```

```
                XORWF           BARG+B0,W
                MOVWF           SIGN                    ; save sign in SIGN

                MOVF            BEXP,W
                ADDWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            MTUN32

                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV32                ; set multiply overflow flag
                GOTO            MOK32

MTUN32          ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN32

MOK32           BSF             AARG+B0,MSB             ; make argument MSB's explicit
                BSF             BARG+B0,MSB
                BCF             _C
                CLRF            ACC+B3                  ; clear initial partial product
                CLRF            ACC+B4
                CLRF            ACC+B5
                MOVLW           24
                MOVWF           TEMP                    ; initialize counter

MLOOP32         BTFSS           AARG+B2,LSB             ; test high byte
                GOTO            MNOADD32

MADD32          MOVF            BARG+B2,W
                ADDWF           ACC+B5
                MOVF            BARG+B1,W
                BTFSC           _C
                INCFSZ          BARG+B1,W
                ADDWF           ACC+B4

                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           ACC+B3

MNOADD32        RRF             ACC+B3
                RRF             ACC+B4
                RRF             ACC+B5
                RRF             ACC+B0
                RRF             ACC+B1
                RRF             ACC+B2
                BCF             _C
                DECFSZ  TEMP
                GOTO            MLOOP32

                BTFSC           ACC+B3,MSB              ; check for postnormalization
                GOTO            MROUND32
                RLF             ACC+B0
                RLF             ACC+B5
                RLF             ACC+B4
                RLF             ACC+B3
                DECF            EXP

MROUND32        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B5,LSB
                GOTO            MUL32OK
                RLF             ACC+B0                  ; rotate next significant bit into
                BTFSC           _C
                INCF            ACC+B5                  ; carry for rounding
                BTFSC           _Z
                INCF            ACC+B4
                BTFSC           _Z
                INCF            ACC+B3

                BTFSS           _C                  ; has rounding caused carryout?
                GOTO            MUL32OK
                RRF             ACC+B3              ; if so, right shift
                RRF             ACC+B4
                RRF             ACC+B5
                INCF            EXP
                BTFSC           _C                  ; check for overflow
                GOTO            SETFOV32

MUL32OK         BTFSS           SIGN,MSB
                BCF             ACC+B3,MSB          ; clear explicit MSB if positive

                MOVF            ACC+B3,W
                MOVWF           AARG+B0             ; move result to AARG
                MOVF            ACC+B4,W
                MOVWF           AARG+B1
                MOVF            ACC+B5,W
                MOVWF           AARG+B2
                RETLW           0

SETFOV32        BSF             FPFLAGS,FOV         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0xFF
                MOVWF           AEXP                ; saturate to largest floating
                MOVWF           AARG+B0             ; point number = 0x FF 7F FF FF
                MOVWF           AARG+B1             ; modulo the appropriate sign bit
                MOVWF           AARG+B2
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                ; return error code in WREG

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Divide

;       Input:  32 bit floating point dividend in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point divisor in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPD32

;       Output: 32 bit floating point quotient in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG / BARG

;       Max Timing:     40+12+23*36+35+14 = 929 clks    RND = 0
;                                40+12+23*36+35+50 = 965 clks    RND = 1, SAT = 0
;                                40+12+23*36+35+53 = 968 clks    RND = 1, SAT = 1

;       Min Timing:     31+12+23*27+26+14 = 704 clks

;       PM: 152                         DM: 14

FPD32           MOVF            BEXP,W              ; test for divide by zero
                BTFSC           _Z
                GOTO            SETFDZ32

D32BNE0         MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN                ; save sign in SIGN
                BSF             AARG+B0,MSB         ; make argument MSB’s explicit
                BSF             BARG+B0,MSB

TALIGN32        CLRF            TEMP                ; clear align increment
                MOVF            AARG+B0,W
                MOVWF           ACC+B3                  ; test for alignment
                MOVF            AARG+B1,W
                MOVWF           ACC+B4
                MOVF            AARG+B2,W
                MOVWF           ACC+B5

                MOVF            BARG+B2,W
                SUBWF           ACC+B5
                MOVF            BARG+B1,W
                BTFSS           _C
                INCFSZ          BARG+B1,W

TS1ALIGN32      SUBWF           ACC+B4
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W

TS2ALIGN32      SUBWF           ACC+B3

                CLRF            ACC+B3
                CLRF            ACC+B4
                CLRF            ACC+B5

                BTFSS           _C
                GOTO            DALIGN32OK

                BCF             _C                      ; align if necessary
                RRF             ACC+B0
                RRF             ACC+B1
                RRF             ACC+B2
                RRF             ACC+B3
                MOVLW           0x01
                MOVWF           TEMP                    ; save align increment

DALIGN32OK      MOVF            BEXP,W                  ; compare AEXP and BEXP
                SUBWF           EXP
                BTFSS           _C
                GOTO            ALTB32

AGEB32          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV32
                GOTO            DARGOK32                ; set overflow flag

ALTB32          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN32                ; set underflow flag

DARGOK32        MOVLW           24                      ; initialize counter
                MOVWF           TEMP+B1

DLOOP32         RLF             ACC+B5                  ; left shift
                RLF             ACC+B4
                RLF             ACC+B3
                RLF             ACC+B2
                RLF             ACC+B1
                RLF             ACC+B0
                RLF             TEMP

                MOVF            BARG+B2,W               ; subtract
                SUBWF           ACC+B2
                MOVF            BARG+B1,W
                BTFSS           _C
                INCFSZ          BARG+B1,W
DS132           SUBWF           ACC+B1
```

```
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
DS232           SUBWF           ACC+B0

                RLF             BARG+B0,W
                IORWF           TEMP

                BTFSS           TEMP,LSB            ; test for restore
                GOTO            DREST32

                BSF             ACC+B5,LSB
                GOTO            DOK32

DREST32         MOVF            BARG+B2,W           ; restore if necessary
                ADDWF           ACC+B2
                MOVF            BARG+B1,W
                BTFSC           _C
                INCFSZ          BARG+B1,W
DAREST32        ADDWF           ACC+B1

                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           ACC+B0

                BCF             ACC+B5,LSB

DOK32           DECFSZ          TEMP+B1
                GOTO            DLOOP32

DROUND32        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B5,LSB
                GOTO            DIV32OK
                BCF             _C
                RLF             ACC+B2              ; compute next significant bit
                RLF             ACC+B1              ; for rounding
                RLF             ACC+B0
                RLF             TEMP

                MOVF            BARG+B2,W           ; subtract
                SUBWF           ACC+B2
                MOVF            BARG+B1,W
                BTFSS           _C
                INCFSZ          BARG+B1,W
                GOTO            DS1ROUND32
                BSF             _C
                BTFSS           _C
DS1ROUND32      SUBWF           ACC+B1

                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                GOTO            DS2ROUND32
                BSF             _C
                BTFSS           _C
DS2ROUND32      SUBWF           ACC+B0

                RLF             BARG+B0,W
                IORWF           TEMP,W
                ANDLW           0x01

                ADDWF           ACC+B5
                BTFSC           _C
                INCF            ACC+B4
                BTFSC           _Z
                INCF            ACC+B3
                BTFSS           _Z                  ; test if rounding caused carryout
                GOTO            DIV32OK
                RRF             ACC+B3
                RRF             ACC+B4
                RRF             ACC+B5
                INCF            EXP
                BTFSC           _Z                  ; test for overflow
                GOTO            SETFOV32


DIV32OK         BTFSS           SIGN,MSB
                BCF             ACC+B3,MSB          ; clear explicit MSB if positive

                MOVF            ACC+B3,W
                MOVWF           AARG+B0             ; move result to AARG
                MOVF            ACC+B4,W
                MOVWF           AARG+B1
                MOVF            ACC+B5,W
                MOVWF           AARG+B2

                RETLW           0

SETFUN32        BSF             FPFLAGS,FUN         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0x01                ; saturate to smallest floating
                MOVWF           AEXP                ; point number = 0x 01 00 00 00
                CLRF            AARG+B0             ; modulo the appropriate sign bit
                CLRF            AARG+B1
                CLRF            AARG+B2
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                ; return error code in WREG

SETFDZ32        BSF             FPFLAGS,FDZ         ; set divide by zero flag
                RETLW           0xFF

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Subtract

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL FPS32

;       Output: 32 bit floating point sum in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     2+248 = 250 clks                RND = 0
;                       2+262 = 264 clks                RND = 1, SAT = 0
;                       2+268 = 270 clks                RND = 1, SAT = 1

;       Min Timing:     2+50 = 52 clks

;       PM: 2+136 = 138         DM: 14

FPS32           MOVLW           0x80
                XORWF           BARG+B0

;—————————————————————————————————

;       Floating Point Add

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2
```

5

```
;       Use:    CALL FPA32

;       Output: 32 bit floating point sum in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     31+38+6*7+6+41+90 = 248 clks    RND = 0
;                                   31+38+6*7+6+55+90 = 262 clks    RND = 1, SAT = 0
;                                   31+38+6*7+6+55+96 = 268 clks    RND = 1, SAT = 1

;       Min Timing:     8+38+4 = 50 clks

;       PM: 136                                 DM: 14

FPA32           MOVF            AARG+B0,W               ; exclusive or of signs in TEMP
                XORWF           BARG+B0,W
                MOVWF           TEMP

                MOVF            AEXP,W                  ; use AARG if AEXP >= BEXP
                SUBWF           BEXP,W
                BTFSS           _C
                GOTO            USEA32

                MOVF            BEXP,W
                MOVWF           ACC+B5                  ; otherwise, swap AARG and BARG
                MOVF            AEXP,W
                MOVWF           BEXP
                MOVF            ACC+B5,W
                MOVWF           AEXP

                MOVF            BARG+B0,W
                MOVWF           ACC+B5
                MOVF            AARG+B0,W
                MOVWF           BARG+B0
                MOVF            ACC+B5,W
                MOVWF           AARG+B0

                MOVF            BARG+B1,W
                MOVWF           ACC+B5
                MOVF            AARG+B1,W
                MOVWF           BARG+B1
                MOVF            ACC+B5,W
                MOVWF           AARG+B1

                MOVF            BARG+B2,W
                MOVWF           ACC+B5
                MOVF            AARG+B2,W
                MOVWF           BARG+B2
                MOVF            ACC+B5,W
                MOVWF           AARG+B2

USEA32          MOVF            AARG+B0,W
                MOVWF           SIGN                    ; save sign in SIGN
                BSF             AARG+B0,MSB             ; make MSB’s explicit
                BSF             BARG+B0,MSB

                MOVF            BARG+B0,W
                MOVWF           ACC+B3
                MOVF            BARG+B1,W
                MOVWF           ACC+B4
                MOVF            BARG+B2,W
                MOVWF           ACC+B5

                MOVF            BEXP,W                  ; compute shift count in BEXP
                SUBWF           AEXP,W
                MOVWF           BEXP
                BTFSC           _Z
                GOTO            AROUND32
                MOVLW           8
                SUBWF           BEXP,W
                BTFSS           _C                    ; if BEXP >= 8, do byte shift
                GOTO            ALIGNB32
                MOVWF           BEXP
                RLF             ACC+B5                ; rotate next bit for rounding
                MOVF            ACC+B4,W
                MOVWF           ACC+B5
                MOVF            ACC+B3,W
                MOVWF           ACC+B4
                CLRF            ACC+B3

                MOVLW           8
                SUBWF           BEXP,W
                BTFSS           _C                    ; if BEXP >= 8, do byte shift
                GOTO            ALIGNB32
                MOVWF           BEXP
                RLF             ACC+B5                ; rotate next bit for rounding
                MOVF            ACC+B4,W
                MOVWF           ACC+B5
                CLRF            ACC+B4


ALIGNB32        MOVF            BEXP,W                ; already aligned if BEXP = 0
                BTFSC           _Z
                GOTO            AROUND32

ALOOPB32        BCF             _C                    ; right shift by BEXP
                RRF             ACC+B3
                RRF             ACC+B4
                RRF             ACC+B5
                DECFSZ          BEXP
                GOTO            ALOOPB32

AROUND32        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B5,LSB
                GOTO            ALIGNED32

                BTFSS           _C
                GOTO            ALIGNED32
                INCF            ACC+B5
                BTFSC           _Z
                INCF            ACC+B4
                BTFSC           _Z
                INCF            ACC+B3

                BTFSS           _Z
                GOTO            ALIGNED32
                RRF             ACC+B3
                RRF             ACC+B4
                RRF             ACC+B5
                INCF            EXP
                BTFSC           _Z
                GOTO            SETFOV32

ALIGNED32       BTFSS           TEMP,MSB              ; negate if signs opposite
                GOTO            AOK32

                COMF            ACC+B3
                COMF            ACC+B4
                COMF            ACC+B5
                INCF            ACC+B5
                BTFSC           _Z
                INCF            ACC+B4
                BTFSC           _Z
                INCF            ACC+B3

AOK32           MOVF            ACC+B5,W              ; add
```

```
                ADDWF           AARG+B2
                MOVF            ACC+B4,W
                BTFSC           _C
                INCFSZ          ACC+B4,W
                ADDWF           AARG+B1

                MOVF            ACC+B3,W
                BTFSC           _C
                INCFSZ          ACC+B3,W
                ADDWF           AARG+B0

                BTFSC           TEMP,MSB
                GOTO            ACOMP32
                BTFSS           _C
                GOTO            FIXSIGN32

                RRF             AARG+B0             ; shift right and increment EXP
                RRF             AARG+B1
                RRF             AARG+B2
                INCFSZ           AEXP
                GOTO            FIXSIGN32
                GOTO            SETFOV32

ACOMP32         BTFSC           _C
                GOTO            NRM32               ; normalize and fix sign

                COMF            AARG+B0             ; negate, toggle sign bit and
                COMF            AARG+B1             ; then normalize
                COMF            AARG+B2
                INCF            AARG+B2
                BTFSC           _Z
                INCF            AARG+B1
                BTFSC           _Z
                INCF            AARG+B0

                MOVLW           0x80
                XORWF           SIGN
                GOTO            NRM32

                END
```

APPENDIX D

```
;       PIC17 32 BIT FLOATING POINT LIBRARY     VERSION 1.14

;       Unary operations: both input and output are in AEXP,AARG

;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       Max timings are worst case cycle counts, while Min timings are non-exception
;       best case cycle counts.

;         Routine               Function

;       FLO24           16 bit integer to 24 bit floating point conversion

;               Timing:            RND
;                               0       1

;                       0       60      60
;                  SAT
;                       1       67      67

;       NRM24           normalization of unnormalized 24 bit floating point numbers

;               Timing:            RND
;                               0       1

;                       0       48      48
;                  SAT
;                       1       55      55


;       INT24           24 bit floating point to 16 bit integer conversion


;               Timing:            RND
;                               0       1

;                       0       69      73
;                  SAT
;                       1       69      76

;       FPA24           24 bit floating point add

;               Timing:            RND
;                               0       1

;                       0       144     159
;                  SAT
;                       1       144     159

;       FPS24           24 bit floating point subtract

;               Timing:            RND
;                               0       1

;                       0       145     160
;                  SAT
;                       1       145     160

;       FPM24           24 bit floating point multiply

;               Timing:            RND
;                               0       1

;                       0       320     331
;                  SAT
;                       1       320     336

;       FPD24           24 bit floating point divide

;               Timing:            RND
;                               0       1

;                       0       597     621
;                  SAT
;                       1       597     627


                list    r=dec,x=on,t=off,p=17C42

                include <PIC17.INC>

;**********************************************************************************************
;**********************************************************************************************

;       32 bit floating point representation

;       EXPONENT        8 bit biased exponent
;                       It is important to note that the use of biased exponents produces
;                       a unique representation of a floating point 0, given by
;                       EXP = HIGHBYTE = MIDBYTE = LOWBYTE = 0x00, with 0 being
;                       the only number with EXP = 0.

;       HIGHBYTE        8 bit most significant byte of sign-magnitude representation, with
;                       SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                       left of MSB

;       MIDBYTE 8 bit middle significant byte of sign-magnitude matissa

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                       RADIX
;       EXPONENT        POINT   HIGHBYTE        MIDBYTE LOWBYTE

;       xxxxxxxx           .    Sxxxxxxx        xxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS         equ     128

;       Define library register variables

EXP             equ     0x18    ; 8 bit biased exponent

ACC             equ     0x19    ; most significant byte of contiguous 6 byte accumulator

SIGN            equ     0x1F    ; save location for sign in MSB

TEMP            equ     0x25    ; temporary storage

;       Define binary operation arguments

AEXP            equ     0x18    ; 8 bit biased exponent for argument A

AARG            equ     0x19    ; most significant byte of mantissa for argument A

BEXP            equ     0x21    ; 8 bit biased exponent for argument B

BARG            equ     0x22    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1 , AARG+B2)  and
;               (EXP, ACC+B0, ACC+B1 , ACC+B2) reference the same storage locations.
;       Define floating point library exception flags

FPFLAGS         equ     0x20    ; floating point library exception flags

IOV             equ     0       ; bit0 = integer overflow flag

FOV             equ     1       ; bit1 = floating point overflow flag

FUN             equ     2       ; bit2 = floating point underflow flag

FDZ             equ     3       ; bit3 = floating point divide by zero flag

RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = rounding to nearest LSB

SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;**********************************************************************************************
;**********************************************************************************************

;       Integer to float conversion

;       Input:  24 bit 2's complement integer right justified in AARG+B0,AARG+B1,
;                               AARG+B2

;       Use:    CALL    FLO32

;       Output: 32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  FLOAT( AARG )

;       Max Timing:     12+48 = 60 clks         SAT = 0
;                                       12+55 = 67 clks         SAT = 1

;       Min Timing:     6+24 = 30 clks          AARG = 0
;                                       6+20 = 26 clks

;       PM: 12+108 = 120                DM: 7

FLO32           MOVLW           24+EXPBIAS              ; initialize exponent and add bias
                MOVWF           EXP
                MOVFP           ACC+B0,SIGN             ; save sign in SIGN
                BTFSS           ACC+B0,MSB              ; test sign
                GOTO            NRM32
                CLRF            WREG                    ; if < 0, negate, set MSB in SIGN
                COMF            ACC+B2
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B2
                ADDWFC          ACC+B1
                ADDWFC          ACC+B0

;————————————————————————————————————

;       Normalization routine

;       Input:  32 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2 with sign in SIGN,MSB.

;       Use:    CALL    NRM32

;       Output: 32 bit normalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2

;       Result: AARG  <—  NORMALIZE( AARG )

;       Max Timing:     21+19+8 = 48 clks              SAT = 0
;                                       21+19+15 = 55 clks              SAT = 1

;       Min Timing:     4+7+7+6 = 24 clks               AARG = 0
;                                       3+5+4+8 = 20 clks

;       PM: 108                                 DM: 7

NRM32           CLRF            TEMP,W                  ; clear exponent decrement
                CPFSGT          ACC+B0                  ; test if highbyte=0
                GOTO            NRM32A

TNIB32          MOVLW           0xF0                    ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM32
                SWAPF           ACC+B0                  ; if so, shift 4 bits
                SWAPF           ACC+B1,W
                ANDLW           0x0F
                ADDWF           ACC+B0

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                MOVPF           WREG,ACC+B1
                SWAPF           ACC+B2,W
                ANDLW           0x0F
                ADDWF           ACC+B1

                SWAPF           ACC+B2,W
                ANDLW           0xF0
                MOVPF           WREG,ACC+B2

                BSF             TEMP,2                  ; increase decrement by 4

NORM32          BCF             _C                      ; clear carry bit

                BTFSC           ACC+B0,MSB              ; if MSB=1, normalization done
                GOTO            TNORMUN32
                RLCF            ACC+B2                  ; otherwise, shift left and
                RLCF            ACC+B1                  ; increment decrement
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN32
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB              ; since highnibble != 0, at most
                GOTO            TNORMUN32               ; 3 left shifts are required
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP

TNORMUN32       MOVFP           TEMP,WREG               ; if EXP <= decrement in TEMP,
                CPFSGT          EXP                     ; floating point underflow has
                GOTO            SETFUN32                ; occured
                SUBWF           EXP                     ; otherwise, compute EXP

FIXSIGN32       BTFSS           SIGN,MSB
                BCF             ACC+B0,MSB              ; clear explicit MSB if positive
                RETLW           0

NRM32A          MOVFP           ACC+B1,ACC+B0           ; shift 8 bits by move
                MOVFP           ACC+B2,ACC+B1
                CLRF            ACC+B2,W
                BSF             TEMP,3                  ; increase decrement by 8
                CPFSGT          ACC+B0                  ; test if highbyte=0
                GOTO            NRM32B
```

```
TNIB32A         MOVLW           0xF0                ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM32A
                SWAPF           ACC+B0              ; if so, shift 4 bits
                SWAPF           ACC+B1,W
                ANDLW           0x0F
                ADDWF           ACC+B0

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                MOVPF           WREG,ACC+B1

                BSF             TEMP,2              ; increase decrement by 4

NORM32A         BCF             _C                  ; clear carry bit

                BTFSC           ACC+B0,MSB          ; if MSB=1, normalization done
                GOTO            TNORMUN32
                RLCF            ACC+B1              ; otherwise, shift left and
                RLCF            ACC+B0              ; increment decrement
                INCF            TEMP
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN32
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB          ; since highnibble != 0, at most
                GOTO            TNORMUN32           ; 3 left shifts are required
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP
                GOTO            TNORMUN32

NRM32B          MOVFP           ACC+B1,ACC+B0       ; shift 8 bits by move
                CLRF            ACC+B1,W
                BCF             TEMP,3              ; increase decrement by 8
                BSF             TEMP,4
                CPFSGT          ACC+B0              ; if highbyte=0, result=0
                GOTO            RES032

TNIB32B         MOVLW           0xF0                ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM32B
                SWAPF           ACC+B0              ; if so, shift 4 bits

                BSF             TEMP,2              ; increase decrement by 4

NORM32B         BCF             _C                  ; clear carry bit

                BTFSC           ACC+B0,MSB          ; if MSB=1, normalization done
                GOTO            TNORMUN32
                RLCF            ACC+B0              ; otherwise, shift left and
                INCF            TEMP                ; increment decrement
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN32
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB          ; since highnibble != 0, at most
                GOTO            TNORMUN32           ; 3 left shifts are required
                RLCF            ACC+B0
                INCF            TEMP
                GOTO            TNORMUN32

;**********************************************************************************************
;**********************************************************************************************
```

5

```
;       Float to integer conversion

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Use:    CALL    INT32

;       Output: 24 bit 2's complement integer right justified in AARG+B0, AARG+B1,
;               AARG+B2

;       Result: AARG  <—  INT( AARG )

;       Max Timing:     11+43+15 = 69 clks             RND = 0
;                                       11+43+19 = 73 clks              RND = 1, SAT = 0
;                                       11+43+22 = 76 clks              RND = 1, SAT = 1

;       Min Timing:     6+6 = 12 clks           AARG = 0
;                                       11+11+8 = 30 clks

;       PM: 155                                 DM: 7

INT32           MOVFP           ACC+B0,SIGN         ; save sign in SIGN
                BSF             ACC+B0,MSB          ; make MSB explicit

                MOVLW           EXPBIAS             ; remove bias from EXP
                CPFSGT          EXP                 ; if <= 0, result=0
                GOTO            RES032
                SUBWF           EXP

                MOVLW           24
                CPFSLT          EXP                 ; if >= 16, integer overflow
                GOTO            SETIOV32            ; will occur
                SUBWF           EXP,W
                NEGW            EXP

                MOVLW           7                   ; do byte shift if EXP >= 8
                CPFSGT          EXP
                GOTO            SNIB32
                SUBWF           EXP                 ; EXP = EXP - 7
                RLCF            ACC+B2              ; rotate next bit for rounding
                MOVFP           ACC+B1,ACC+B2
                MOVFP           ACC+B0,ACC+B1
                CLRF            ACC+B0
                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT32OK           ; shift completed if EXP = 0

                CPFSGT          EXP                 ; do another byte shift if EXP >= 8
                GOTO            SNIB32A
                SUBWF           EXP                 ; EXP = EXP - 7
                RLCF            ACC+B2              ; rotate next bit for rounding
                MOVFP           ACC+B1,ACC+B2
                CLRF            ACC+B1
                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT32OK           ; shift completed if EXP = 0

SNIB32B         MOVLW           3                   ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT32B
                SUBWF           EXP                 ; EXP = EXP - 3
                SWAPF           ACC+B2,W
                MOVPF           WREG,ACC+B3         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B2
                RLCF            ACC+B3              ; rotate next bit for rounding

                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT32OK           ; shift completed if EXP = 0

SHIFT32B        BCF             _C                  ; at most 3 right shifts are required
```

```
                RRCF            ACC+B2                  ; right shift by EXP
                DCFSNZ          EXP
                GOTO            SHIFT32OK               ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK               ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B2
                GOTO            SHIFT32OK

SNIB32A         MOVLW           3                       ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT32A
                SUBWF           EXP                     ; EXP = EXP - 3
                SWAPF           ACC+B2,W
                MOVPF           WREG,ACC+B3             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                ADDWF           ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0x0F
                MOVPF           WREG,ACC+B1
                RLCF            ACC+B3                  ; rotate next bit for rounding

                DCFSNZ          EXP                     ; EXP = EXP - 1
                GOTO            SHIFT32OK               ; shift completed if EXP = 0

SHIFT32A        BCF             _C                      ; at most 3 right shifts are required
                RRCF            ACC+B1                  ; right shift by EXP
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK               ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B1
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK               ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B1
                RRCF            ACC+B2
                GOTO            SHIFT32OK

SNIB32          MOVLW           3                       ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT32
                SUBWF           EXP                     ; EXP = EXP - 3
                SWAPF           ACC+B2,W
                MOVPF           WREG,ACC+B3             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                ADDWF           ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0x0F
                MOVPF           WREG,ACC+B1

                SWAPF           ACC+B0,W
                ANDLW           0xF0
                ADDWF           ACC+B1

                SWAPF           ACC+B0,W
```

5

```
                ANDLW           0x0F
                MOVPF           WREG,ACC+B0
                RLCF            ACC+B3                  ; rotate next bit for rounding

                DCFSNZ          EXP                     ; EXP = EXP - 1
                GOTO            SHIFT32OK               ; shift completed if EXP = 0

SHIFT32         BCF             _C                      ; at most 3 right shifts are required
                RRCF            ACC+B0                  ; right shift by EXP
                RRCF            ACC+B1
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK               ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK               ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1
                RRCF            ACC+B2

SHIFT32OK       CLRF            WREG
                BTFSC           FPFLAGS,RND
                BTFSS           ACC+B2,LSB
                GOTO            INT32OK
                ADDWFC          ACC+B2                  ; add next bit for rounding
                ADDWFC          ACC+B1
                ADDWFC          ACC+B0
                BTFSC           ACC+B0,MSB
                GOTO            SETIOV32

INT32OK         BTFSS           SIGN,MSB                ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B2
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B2
                ADDWFC          ACC+B1
                ADDWFC          ACC+B0
                RETLW           0

RES032          CLRF            ACC+B0                  ; integer result equals zero
                CLRF            ACC+B1
                CLRF            ACC+B2
                CLRF            EXP                     ; clear exponent for other routines
                RETLW           0

SETIOV32        BSF             FPFLAGS,IOV             ; set integer overflow flag
                BTFSS           FPFLAGS,SAT             ; test for saturation
                RETLW           0xFF                    ; return error code in WREG

                CLRF            ACC+B0                  ; saturate to largest two's
                BTFSS           SIGN,MSB                ; complement 24 bit integer
                SETF            ACC+B0                  ; SIGN = 0, 0x 7F FF FF
                MOVPF           ACC+B0,ACC+B1           ; SIGN = 1, 0x 80 00 00
                MOVPF           ACC+B0,ACC+B2
                RLCF            SIGN
                RRCF            ACC+B0
                RETLW           0xFF                    ; return error code in WREG

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Multiply

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPM32

;       Output: 32 bit floating point product in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG * BARG

;       Max Timing:     19+283+18 = 320 clks            RND = 0, SAT = 0
;                               19+283+29 = 331 clks            RND = 1, SAT = 0
;                               19+283+34 = 336 clks            RND = 1, SAT = 1

;       Min Timing:     5+6 = 11 clks           AARG * BARG = 0
;                               18+54+16 = 88 clks

;       PM: 21+322+39 = 382                     DM: 13

MUL32           macro

;       Max Timing:     13+22*12+6 = 283 clks

;       Min Timing:     4+22*2+6 = 54 clks

;       PM: 4+24*2+6+22*12 = 322                        DM: 9

                variable i = 0
                variable j = 1

                BCF             _C                  ; clear carry bit

                CLRF            ACC+B3              ; clear initial partial product
                CLRF            ACC+B4
                CLRF            ACC+B5

                while   i < 24

                if      i < 8

                BTFSC           AARG+B2,i           ; test low byte

                endif

                if      i >= 8 && i < 16

                BTFSC           AARG+B1,i-8         ; test middle byte

                endif

                GOTO            Madd32#v( i )

                if      i >= 16

                BTFSC           AARG+B0,i-16        ; test high byte

                endif

                GOTO            Madd32#v( i )

                i += 1

                endw

;       we can not get here since the MSB’s are explicit

Madd320         MOVFP           BARG+B0,ACC+B3      ; if we get here, ACC=0 so move
                MOVFP           BARG+B1,ACC+B4      ; BARG for add
                MOVFP           BARG+B2,ACC+B5

                RRCF            ACC+B3
```

5

```
                RRCF            ACC+B4
                RRCF            ACC+B5

                while j < 23

                if      j < 8

                BTFSS           AARG+B2,j               ; test low byte

                endif

                if      j >= 8 && j < 16

                BTFSS           AARG+B1,j-8             ; test middle byte

                endif

                if      j >= 16

                BTFSS           AARG+B0,j-16            ; test middle byte

                endif

                GOTO            Mnoadd32#v( j )

Madd32#v( j )   MOVFP           BARG+B2,WREG
                ADDWF           ACC+B5                  ; add low byte
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B4                  ; add middle byte
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B3                  ; add high byte

Mnoadd32#v( j ) RRCF            ACC+B3
                RRCF            ACC+B4
                RRCF            ACC+B5

                if j > 21

                RRCF            ACC+B1                  ; save bits for rounding

                endif

                if j < 22

                BCF             _C                      ; clear carry when rotate possible

                endif

                j += 1

                endw


Madd3223        MOVFP           BARG+B2,WREG
                ADDWF           ACC+B5                  ; add low byte
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B4                  ; add middle byte
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B3                  ; add high byte

                endm

FPM32           CLRF            WREG                    ; test for zero arguments
                CPFSEQ          BEXP
                CPFSGT          AEXP
                GOTO            RES032

M24BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVPF           WREG,SIGN           ; save sign in SIGN

                MOVFP           BEXP,WREG
                ADDWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            MTUN32

                ADDWF           EXP                 ; remove bias and overflow test
                BTFSC           _C
                GOTO            SETFOV32
                GOTO            MOK32

MTUN32          ADDWF           EXP                 ; remove bias and underflow test
                BTFSS           _C
                GOTO            SETFUN32

MOK32           BSF             AARG+B0,MSB         ; make argument MSB’s explicit
                BSF             BARG+B0,MSB

                MUL32                               ; use macro for multiplication

                BTFSC           _C                  ; check for postnormalization
                GOTO            MRIGHT32
                DECF            EXP
                GOTO            MROUND32
MRIGHT32        RRCF            ACC+B3
                RRCF            ACC+B4
                RRCF            ACC+B5
                RRCF            ACC+B1

MROUND32        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B5,LSB
                GOTO            MUL32OK
                CLRF            WREG
                RLCF            ACC+B1              ; rotate next significant bit into
                ADDWFC          ACC+B5              ; carry for rounding
                ADDWFC          ACC+B4
                ADDWFC          ACC+B3

                BTFSS           _C                  ; has rounding caused carryout?
                GOTO            MUL32OK
                RRCF            ACC+B3              ; if so, right shift
                RRCF            ACC+B4
                RRCF            ACC+B5
                INFSNZ          EXP                 ; test for floating point overflow
                GOTO            SETFOV32

MUL32OK         BTFSS           SIGN,MSB
                BCF             ACC+B3,MSB          ; clear explicit MSB if positive

                MOVFP           ACC+B3,AARG+B0      ; move result to AARG
                MOVFP           ACC+B4,AARG+B1
                MOVFP           ACC+B5,AARG+B2
                RETLW           0

SETFOV32        BSF             FPFLAGS,FOV         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                SETF            AEXP                ; saturate to largest floating
                SETF            AARG+B0             ; point number = 0x FF 7F FF FF
                SETF            AARG+B1             ; modulo the appropriate sign bit
                SETF            AARG+B2
                RLCF            SIGN
                RRCF            AARG+B0
                RETLW           0xFF                ; return error code in WREG
```

5

```
;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Divide

;       Input:  32 bit floating point dividend in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point divisor in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPD32

;       Output: 32 bit floating point quotient in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG / BARG

;       Max Timing:     27+11+548+11 = 597 clks RND = 0
;                       27+11+548+35 = 621 clks RND = 1, SAT = 0
;                       27+11+548+41 = 627 clks RND = 1, SAT = 1

;       Min Timing:     6+6 = 12 clks           AARG = 0
;                       33+428+11 = 472 clks

;       PM: 42+572+47 = 661                     DM: 14

DIV24           macro

;       Timing: 1+24*26 = 625 clks

                variable i = 0

                BCF             _C

                while i < 24

                CLRF            TEMP                    ; left shift
                RLCF            ACC+B5
                RLCF            ACC+B4
                RLCF            ACC+B3
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP

                MOVFP           BARG+B2,WREG            ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                CLRF            WREG
                SUBWFB          TEMP

                BTFSS           _C                      ; test for restore
                GOTO            Drest#v( i )

                BSF             ACC+B5,LSB
                GOTO            Dok#v( i )

Drest#v( i )    MOVFP           BARG+B2,WREG            ; restore if necessary
                ADDWF           ACC+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B5,LSB

Dok#v( i )

                i += 1
                endw

                endm


DIV24A          macro

;       Min Timing:     19+23*23 = 548 clks

;       Min Timing:     14+23*18 = 428 clks

;       PM: 20+23*24 = 572                      DM: 9

                BTG             TEMP,LSB            ; save carry

                MOVFP           BARG+B2,WREG        ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB            ; test for restore
                GOTO            DArest0

                BSF             ACC+B3,MSB
                GOTO            DAok0

DArest0         MOVFP           BARG+B2,WREG        ; restore if necessary
                ADDWF           ACC+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B3,MSB

DAok0


                variable i = 1

                while i < 24

                BCF             _C
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP                ; save carry

                MOVFP           BARG+B2,WREG        ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB            ; test for restore
                GOTO            DArest#v( i )

                if i < 8

                BSF             ACC+B3,MSB-i        ; set bit in quotient

                endif
                if i >= 8 && i < 16

                BSF             ACC+B4,15-i         ; set bit in quotient

                endif

                if i >= 16

                BSF             ACC+B5,23-i         ; set bit in quotient

                endif

                GOTO            DAok#v( i )

DArest#v( i )   MOVFP           BARG+B2,WREG        ; restore if necessary
                ADDWF           ACC+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0


                if i < 8

                BCF             ACC+B3,MSB-i        ; clear bit in quotient

                endif

                if i >= 8 && i < 16

                BCF             ACC+B4,15-i         ; clear bit in quotient

                endif

                if i >= 16

                BCF             ACC+B5,23-i         ; clear bit in quotient

                endif

DAok#v( i )

                i += 1

                endw

                endm

FPD32           CLRF            TEMP,W
                CPFSGT  BEXP                        ; test for divide by zero
                GOTO            SETFDZ32

                CPFSGT  AEXP
                GOTO            RES032

D32BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVPF           WREG,SIGN           ; save sign in SIGN
                BSF             AARG+B0,MSB         ; make argument MSB’s explicit
                BSF             BARG+B0,MSB

TALIGN32        INCF            TEMP                ; set align increment
                MOVFP           AARG+B0,ACC+B3      ; test for alignment
                MOVFP           AARG+B1,ACC+B4
                MOVFP           AARG+B2,ACC+B5
                MOVFP           BARG+B2,WREG
                SUBWF           ACC+B5
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B4
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B3
                BTFSC           _C
                GOTO            DALIGN32OK

                BCF             _C                      ; align if necessary
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                DECF            TEMP                    ; fix align increment

DALIGN32OK      MOVFP           BEXP,WREG               ; compute AEXP - BEXP
                SUBWF           EXP
                BTFSS           _C
                GOTO            ALTB32

AGEB32          MOVLW           EXPBIAS                 ; if AEXP > BEXP, test for
                ADDWF           TEMP,W                  ; overflow
                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV32
                GOTO            DARGOK32

ALTB32          MOVLW           EXPBIAS                 ; if AEXP < BEXP, test for
                ADDWF           TEMP,W                  ; underflow
                ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN32

DARGOK32        DIV24A                                  ; macro for division

DROUND32        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B5,LSB
                GOTO            DIV32OK
                BCF             _C                      ; compute next significant bit
                RLCF            ACC+B2                  ; for rounding
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP                    ; save carry

                MOVFP           BARG+B2,WREG            ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                RRCF            TEMP                    ; rotate next significant bit into
                CLRF            WREG                    ; carry and add for rounding
                ADDWFC          ACC+B5
                ADDWFC          ACC+B4
                ADDWFC          ACC+B3

                BTFSS           _C                      ; test if rounding caused carryout
                GOTO            DIV32OK
                RRCF            ACC+B3
                RRCF            ACC+B4
                RRCF            ACC+B5
                INFSNZ          EXP                     ; test for overflow
                GOTO            SETFOV32

DIV32OK         BTFSS           SIGN,MSB
                BCF             ACC+B3,MSB              ; clear explicit MSB if positive

                MOVFP           ACC+B3,AARG+B0          ; move result to AARG
                MOVFP           ACC+B4,AARG+B1
                MOVFP           ACC+B5,AARG+B2
                RETLW           0

SETFUN32        BSF             FPFLAGS,FUN          ; set floating point underflag
                BTFSS           FPFLAGS,SAT          ; test for saturation
                RETLW           0xFF                 ; return error code in WREG

                MOVLW           0x01                 ; saturate to smallest floating
                MOVPF           WREG,AEXP            ; point number = 0x 01 00 00 00
                CLRF            AARG+B0              ; modulo the appropriate sign bit
                CLRF            AARG+B1
                CLRF            AARG+B2
                RLCF            SIGN
                RRCF            AARG+B0
                RETLW           0xFF                 ; return error code in WREG

SETFDZ32        BSF             FPFLAGS,FDZ          ; set floating point divide by zero
                RETLW           0xFF                 ; flag and return error code in WREG

;**********************************************************************************************
;**********************************************************************************************

;       Floating Point Subtract

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL FPS32

;       Output: 32 bit floating point difference in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     1+144 = 145 clks                RND = 0
;                       1+159 = 160 clks                RND = 1

;       Min Timing:     1+34 = 35 clks

;       PM: 330                                 DM: 14

FPS32           BTG             BARG+B0,MSB          ; toggle sign bit for subtraction

;——————————————————————————————————

;       Floating Point Add

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL FPA32

;       Output: 32 bit floating point sum in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <—  AARG - BARG

;       Max Timing:     7+67+22+48 = 144 clks           RND = 0
;                       7+75+22+55 = 159 clks           RND = 1

;       Min Timing:     6+23+5 = 34 clks

;       PM: 329                                 DM: 14

FPA32           MOVFP           AARG+B0,WREG         ; exclusive or of signs in TEMP
                XORWF           BARG+B0,W
                MOVPF           WREG,TEMP

                MOVFP           AEXP,WREG            ; use AARG if AEXP >= BEXP
                CPFSGT          BEXP
                GOTO            USEA32
```

```
USEB32          MOVFP           BARG+B0,SIGN        ; save sign in SIGN
                BSF             BARG+B0,MSB         ; make MSB's explicit
                BSF             AARG+B0,MSB

                MOVFP           AEXP,WREG           ; compute shift count in BEXP
                MOVFP           BEXP,AEXP
                SUBWF           BEXP
                BTFSC           _Z
                GOTO            BLIGNED32

                MOVLW           7
                CPFSGT          BEXP                ; do byte shift if BEXP >= 8
                GOTO            BNIB32
                SUBWF           BEXP                ; BEXP = BEXP - 7
                RLCF            AARG+B2             ; rotate next bit for rounding
                MOVFP           AARG+B1,AARG+B2
                MOVFP           AARG+B0,AARG+B1
                CLRF            AARG+B0
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            BROUND32

                CPFSGT          BEXP                ; do another byte shift if BEXP >= 8
                GOTO            BNIB32A
                SUBWF           AEXP                ; BEXP = BEXP - 7
                RLCF            AARG+B2             ; rotate next bit for rounding
                MOVFP           AARG+B1,AARG+B2
                CLRF            AARG+B1
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            BROUND32

                CPFSGT          BEXP                ; if BEXP still >= 8, then
                GOTO            BNIB32B             ; AARG = 0 relative to BARG

                MOVFP           SIGN,AARG+B0        ; return BARG
                MOVFP           BARG+B1,AARG+B1
                MOVFP           BARG+B2,AARG+B2
                RETLW           0x00

BNIB32B         MOVLW           3                   ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            BLOOP32B
                SUBWF           BEXP                ; BEXP = BEXP -3

                SWAPF           AARG+B2,W
                MOVPF           WREG,ACC+B5         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,AARG+B2
                RLCF            ACC+B5              ; rotate next bit for rounding

                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            BROUND32            ; aligned if BEXP = 0

BLOOP32B        BCF             _C                  ; right shift by BEXP
                RRCF            AARG+B2
                DCFSNZ          BEXP
                GOTO            BROUND32            ; aligned if BEXP = 0
                BCF             _C
                RRCF            AARG+B2
                DCFSNZ          BEXP
                GOTO            BROUND32            ; aligned if BEXP = 0
                BCF             _C                  ; at most 3 right shifts are
                RRCF            AARG+B2             ; possible
                GOTO            BROUND32

BNIB32A         MOVLW           3                   ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            BLOOP32A
                SUBWF           BEXP                ; BEXP = BEXP -3
                SWAPF           AARG+B2,W
                MOVPF           WREG,ACC+B5             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,AARG+B2

                SWAPF           AARG+B1,W
                ANDLW           0xF0
                ADDWF           AARG+B2

                SWAPF           AARG+B1,W
                ANDLW           0x0F
                MOVPF           WREG,AARG+B1

                RLCF            ACC+B5                  ; rotate next bit for rounding

                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            BROUND32                ; aligned if BEXP = 0

BLOOP32A        BCF             _C                      ; right shift by BEXP
                RRCF            AARG+B1
                RRCF            AARG+B2
                DCFSNZ          BEXP
                GOTO            BROUND32                ; aligned if BEXP = 0
                BCF             _C
                RRCF            AARG+B1
                RRCF            AARG+B2
                DCFSNZ          BEXP
                GOTO            BROUND32                ; aligned if BEXP = 0
                BCF             _C                      ; at most 3 right shifts are
                RRCF            AARG+B1                 ; possible
                RRCF            AARG+B2
                GOTO            BROUND32

BNIB32          MOVLW           3                       ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            BLOOP32
                SUBWF           BEXP                    ; BEXP = BEXP -3

                SWAPF           AARG+B2,W
                MOVPF           WREG,ACC+B5             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,AARG+B2

                SWAPF           AARG+B1,W
                ANDLW           0xF0
                ADDWF           AARG+B2

                SWAPF           AARG+B1,W
                ANDLW           0x0F
                MOVPF           WREG,AARG+B1

                SWAPF           AARG+B0,W
                ANDLW           0xF0
                ADDWF           AARG+B1

                SWAPF           AARG+B0,W
                ANDLW           0x0F
                MOVPF           WREG,AARG+B0

                RLCF            ACC+B5                  ; rotate next bit for rounding

                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            BROUND32                ; aligned if BEXP = 0

BLOOP32         BCF             _C                      ; right shift by BEXP
                RRCF            AARG+B0
                RRCF            AARG+B1
                RRCF            AARG+B2
                DCFSNZ          BEXP
                GOTO            BROUND32            ; aligned if BEXP = 0
                BCF             _C
                RRCF            AARG+B0
                RRCF            AARG+B1
                RRCF            AARG+B2
                DCFSNZ          BEXP
                GOTO            BROUND32            ; aligned if BEXP = 0
                BCF             _C                  ; at most 3 right shifts are
                RRCF            AARG+B0             ; possible
                RRCF            AARG+B1
                RRCF            AARG+B2

BROUND32        BTFSC           FPFLAGS,RND
                BTFSS           AARG+B2,LSB
                GOTO            BLIGNED32
                CLRF            WREG
                ADDWFC          AARG+B2             ; add carry for rounding
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0

                BTFSS           _C                  ; has rounding caused carryout?
                GOTO            BLIGNED32
                RRCF            AARG+B0             ; if so, right shift
                RRCF            AARG+B1
                RRCF            AARG+B2
                INFSNZ          EXP                 ; test for floating point overflow
                GOTO            SETFOV32

BLIGNED32       BTFSS           TEMP,MSB            ; negate if signs opposite
                GOTO            AOK32
                CLRF            WREG
                COMF            AARG+B2
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
                GOTO            AOK32

USEA32          MOVFP           AARG+B0,SIGN        ; save sign in SIGN
                BSF             AARG+B0,MSB         ; make MSB’s explicit
                BSF             BARG+B0,MSB

                MOVFP           BEXP,WREG           ; compute shift count in BEXP
                SUBWF           AEXP,W
                MOVPF           WREG,BEXP
                BTFSC           _Z
                GOTO            ALIGNED32

                MOVLW           7
                CPFSGT          BEXP                ; do byte shift if BEXP >= 8
                GOTO            ANIB32
                SUBWF           BEXP                ; BEXP = BEXP - 7
                RLCF            BARG+B2             ; rotate next bit for rounding
                MOVFP           BARG+B1,WREG
                MOVPF           WREG,BARG+B2
                MOVFP           BARG+B0,WREG
                MOVPF           WREG,BARG+B1
                CLRF            BARG+B0
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            AROUND32

                CPFSGT          BEXP                ; do another byte shift if BEXP >= 8
                GOTO            ANIB32A
                SUBWF           BEXP                ; BEXP = BEXP - 7
                RLCF            BARG+B2             ; rotate next bit for rounding
                MOVFP           BARG+B1,WREG
                MOVPF           WREG,BARG+B2
```

5

```
                CLRF            BARG+B1
                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            AROUND32

                CPFSGT          BEXP                    ; if BEXP still >= 8, then
                GOTO            ANIB32B                 ; BARG = 0 relative to AARG

                MOVFP           SIGN,AARG+B0            ; return AARG
                RETLW           0x00

ANIB32B         MOVLW           3                       ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP32B
                SUBWF           BEXP                    ; BEXP = BEXP -3

                SWAPF           BARG+B2,W
                MOVPF           WREG,ACC+B5             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B2
                RLCF            ACC+B5                  ; rotate next bit for rounding

                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            AROUND32                ; aligned if BEXP = 0

ALOOP32B        BCF             _C                      ; right shift by BEXP
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
                BCF             _C                      ; at most 3 right shifts are
                RRCF            BARG+B2                 ; possible
                GOTO            AROUND32

ANIB32A         MOVLW           3                       ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP32A
                SUBWF           BEXP                    ; BEXP = BEXP -3

                SWAPF           BARG+B2,W
                MOVPF           WREG,ACC+B5             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0xF0
                ADDWF           BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0x0F
                MOVPF           WREG,BARG+B1

                RLCF            ACC+B5                  ; rotate next bit for rounding

                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            AROUND32                ; aligned if BEXP = 0

ALOOP32A        BCF             _C                      ; right shift by BEXP
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
                BCF             _C                    ; at most 3 right shifts are
                RRCF            BARG+B1               ; possible
                RRCF            BARG+B2
                GOTO            AROUND32

ANIB32          MOVLW           3                     ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP32
                SUBWF           BEXP                  ; BEXP = BEXP -3

                SWAPF           BARG+B2,W
                MOVPF           WREG,ACC+B5           ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0xF0
                ADDWF           BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0x0F
                MOVPF           WREG,BARG+B1

                SWAPF           BARG+B0,W
                ANDLW           0xF0
                ADDWF           BARG+B1

                SWAPF           BARG+B0,W
                ANDLW           0x0F
                MOVPF           WREG,BARG+B0

                RLCF            ACC+B5                ; rotate next bit for rounding

                DCFSNZ          BEXP                  ; BEXP = BEXP - 1
                GOTO            AROUND32              ; aligned if BEXP = 0

ALOOP32         BCF             _C                    ; right shift by BEXP
                RRCF            BARG+B0
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32              ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B0
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32              ; aligned if BEXP = 0
                BCF             _C                    ; at most 3 right shifts are
                RRCF            BARG+B0               ; possible
                RRCF            BARG+B1
                RRCF            BARG+B2

AROUND32        BTFSC           FPFLAGS,RND
                BTFSS           BARG+B2,LSB
                GOTO            ALIGNED32
                CLRF            WREG
                ADDWFC          BARG+B2               ; add carry for rounding
                ADDWFC          BARG+B1
                ADDWFC          BARG+B0

                BTFSS           _C                    ; has rounding caused carryout?
                GOTO            ALIGNED32
                RRCF            BARG+B0               ; if so, right shift
                RRCF            BARG+B1
                RRCF            BARG+B2
                INFSNZ          EXP                   ; test for floating point overflow
                GOTO            SETFOV32
```

```
ALIGNED32       BTFSS           TEMP,MSB            ; negate if signs opposite
                GOTO            AOK32
                CLRF            WREG
                COMF            BARG+B2
                COMF            BARG+B1
                COMF            BARG+B0
                INCF            BARG+B2
                ADDWFC          BARG+B1
                ADDWFC          BARG+B0

AOK32           MOVFP           BARG+B2,WREG        ; add
                ADDWF           AARG+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          AARG+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          AARG+B0

                BTFSC           TEMP,MSB
                GOTO            ACOMP32
                BTFSS           _C
                GOTO            FIXSIGN32

                RRCF            AARG+B0             ; shift right and increment EXP
                RRCF            AARG+B1
                RRCF            AARG+B2
                INCFSZ          AEXP
                GOTO            FIXSIGN32
                GOTO            SETFOV32            ; set floating point overflow flag

ACOMP32         BTFSC           _C
                GOTO            NRM32               ; normalize and fix sign

                CLRF            WREG                ; negate, toggle sign bit and
                COMF            AARG+B2             ; then normalize
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
                BTG             SIGN,MSB
                GOTO            NRM32

                END
```

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200 Fax: 602 786-7277
Technical Support: 602 786-7627
Web: http://www.mchip.com/microhip

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770 640-0034 Fax: 770 640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508 480-9990 Fax: 508 480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 708 285-0071 Fax: 708 285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177 Fax: 214 991-8588

Dayton
Microchip Technology Inc.
35 Rockridge Road
Englewood, OH 45322
Tel: 513 832-2543 Fax: 513 832-2841

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888 Fax: 714 263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516 273-5305 Fax: 516 273-5335

AMERICAS (continued)

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408 436-7950 Fax: 408 436-7955

ASIA/PACIFIC

Hong Kong
Microchip Technology
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 2 401 1200 Fax: 852 2 401 3431

Korea
Microchip Technology
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku,
Seoul, Korea
Tel: 82 2 554 7200 Fax: 82 2 558 5934

Singapore
Microchip Technology
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65 334 8870 Fax: 65 334 8850

Taiwan
Microchip Technology
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2 717 7175 Fax: 886 2 545 0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

France
Arizona Microchip Technology SARL
2 Rue du Buisson aux Fraises
91300 Massy - France
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Muenchen, Germany
Tel: 49 89 627 144 0 Fax: 49 89 627 144 44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041
Agrate Brianza (MI) Italy
Tel: 39 039 689 9939 Fax: 39 039 689 9883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/22/95