Math Utility Routines

INTRODUCTION

This application note provides some utility math routines for Microchip's second generation of high performance 8-bit microcontroller, the PIC17C42. Three assembly language modules are provided, namely ARITH.ASM, BCD.ASM and FXP_DIV.ASM. Currently in each file the following subroutines are implemented:

ARITH.ASM

- Single precision 8 x 8 unsigned multiply
- 16 x 16 double precision multiply (signed or unsigned)
- 16 / 16 double precision divide (signed or unsigned)
- 16 x 16 double precision addition
- 16 x 16 double precision subtraction
- double precision square root
- double precision numerical differentiation
- double precision numerical integration
- Pseudo Random number generation
- Gaussian distributed random number generation

BCD.ASM

- 8-bit binary to 2 digit BCD conversion
- 16-bit binary to 5 digit BCD conversion
- 5-bit BCD to 16-bit binary conversion
- 2 digit BCD addition

FXP_DIV.ASM.

The routines that are implementing in this source file are shown in Table 3.

As more routines are available, they will be added to the library. The latest routines may be obtained either through Microchip's bulletin board or by contacting your nearest Microchip sales office for a copy on a MS-DOS™ floppy.

These routines have been optimized wherever possible with a compromise between speed, RAM utilization, and code size. Some routines (multiplication and division) are provided in two forms, one optimized for speed and the other optimized for code size.

All the routines have been implemented as callable subroutines and the usage of each routine is explained below. At the end of the application note, the listing files of the above programs are given.

SINGLE PRECISION UNSIGNED MULTIPLICATION (8 X 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 1.

DOUBLE PRECISION MULTIPLICATION

This routine computes the product of 16- bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code). These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 2.

TABLE 1: SINGLE PRECISION MULTIPLICATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
mpy8x8_F	speed efficient	36	36	0	used
mpy8x8_S	code efficient	13	69	1	used

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

The data memory organization is explained in the comment section of the code. Faster execution and code space saving can be achieved by setting "MODE_FAST equ TRUE". However, setting MODE_FAST variable to TRUE restricts that operands and the 32-bit result be in data RAM locations 0x18 and 0x1F (in this mode, MOVFP and MOVPF instructions may be used to transfer data to/from any RAM location to addresses less than 0x1F). If MODE_FAST is set to FALSE, there will be no restriction on the location of the data RAM values used in this subroutine. However, the code will be slightly slower and occupies more program memory.

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

DOUBLE PRECISION DIVISION

This routine performs a 2's complement division of two 16-bit numbers and produces a 16-bit quotient with a 16-bit remainder. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code).

These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in Appendix C. The performance specs are shown in Table 3.

TABLE 2: DOUBLE PRECISION MULTIPLICATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
D_mpyF	Speed Efficient, Signed Arithmetic	204	183	1	used
D_mpyF	Speed Efficient, Unsigned Arithmetic	179	176	0	used
D_mpyS	Code Efficient, Signed Arithmetic	52	254	4	used
D_mpyS	Code Efficient, Unsigned Arithmetic	21	242	3	used

TABLE 3: FIXED POINT DIVIDE PERFORMANCE DATA

Routine	Max Cycles	Min. Cycles	Program Memory	Data Memory
16 / 8 Signed	146	135	146	5
16 / 8 Unsigned	196	156	195	4
16 / 7 Unsigned	130	130	129	4
15 / 7 Unsigned	125	125	124	4
16 / 16 Unsigned	214	187	241	7
16 / 16 Unsigned	244	180	243	6
16 / 15 Unsigned	197	182	216	6
16 / 15 Unsigned	191	177	218	6
32 / 16 Unsigned	414	363	476	9
32 / 16 Unsigned	485	459	608	9
32 / 15 Unsigned	390	359	451	8
31 / 15 Unsigned	383	353	442	8

DOUBLE PRECISION ADDITION AND SUBTRACTION

Two routines are provided. One performs a 2's complement addition and the other one performs a 2's complement subtraction of two 16-bit binary numbers. These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 4.

NEGATE A DOUBLE PRECISION NUMBER

These routines negate a double precision number (16-bit and 32-bit). Two routines and two macros are provided to negate a 16-bit number. The subroutines use indirect addressing mode and the macros use a direct addressing scheme. A macro is provided to negate a 32- bit number.

DOUBLE PRECISION SQUARE ROOT

Often in many applications, one needs to find the square root of a number. Of the many numerical methods available to compute the square root of a number, the Newton-Raphson method is one of the most attractive because of its fast convergence rate. In this method, the square root of number, N, is obtained as an approximate solution of

$$f(Y) = Y^2 - N = 0$$

The function f(Y) can be expanded about Y0 using the first order Taylor polynomial expansion as:

Equation 1:

$$f(Y) = f(Y_0) + (Y - Y_0)f'(Y_0) + \frac{(Y - Y_0)2f'' f(Y_0)}{2!} + ...$$

If X is a root of f(Y), then f(X) = 0: Therefore,

$$f(X) = f(Y_0) + (X - Y_0)f'(Y_0) + \frac{(X - Y_0)2f'' f(Y_0)}{2!} + ... = 0$$

Therefore, $f(Y_0) + (X - Y_0)f'(Y_0) = 0$

i.e., $X = Y_0 + \frac{f(Y_0)}{'f(Y_0)}$

4

TABLE 4: DOUBLE PRECISION ADDITION AND SUBTRACTION

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Dadd	4	4	0	used
Dsub	4	4	0	used

TABLE 5: NEGATE A DOUBLE PRECISION NUMBER

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Negate	7	7	0	unused
NegateAlt	7	7	0	used
NegMac	5	5	0	used
AltNegMac	5	5	0	unused
NegMac32 (32 bit)	11	11	0	used

Thus X is a better approximation for Y0. From the previous equation, the sequence {Xn} can be generated:

Equation 2: $X_n = X_{n-1} - \frac{f(X_{n-1})}{f'(X_{n-1})}, \quad n >= 1$

For our case, equation 2, reduces to:

Equation 3: $\frac{X_{n-1} + \frac{N}{X_{n-1}}}{2}$

The routine "Sqrt" in ARITH.ASM implements the above equation. Equation 3 requires that at first an initial approximation for the root is known. The better the initial approximation, the faster the convergence rate would be. In the "Sqrt" routine, the initial approximation root is set as N/2. This routine calls the double precision division routine (D_divS).

In the code size, the Division routine (Ddiv_S) size is not included.

BCD ROUTINES

Three routines are provided for general purpose BCD arithmetic:

a. BCD to binary conversion

b. Binary to BCD conversion

c. BCD addition

The BCD to binary conversion routine converts a 5 digit BCD code to a 16-bit binary number. The BCD addition routine adds two BCD digits directly without converting them at first to binary. Note the usage of the **"DAW"** instruction. The other two routines convert a binary number to a BCD code. The performance specs for the BCD routines is given in the Table 7 below.

NUMERICAL DIFFERENTIATION

This routine performs numerical differentiation of a sequence of data if the input sequence is assumed to be piecewise linear with no discontinuances (this is the case in most of the real world signals). Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. This routine uses the so called 3-Point formula to compute the differential of a sequence of numbers.

Given an equation f(t), its derivative is given by

$f'(t) = \frac{df(t)}{dt}$

The above equation can be approximated using the 3-Point formula as given below:

3-Point Formula:

$f'(t) = \frac{df(t)}{dt} = \frac{1}{2h} [f(t_0 - 2h) - 4f(T_0 - h) + 3f(t_0)]$

where t_0 is the point at which the numerical derivative is desired and "h" is the step size. The smaller the value of the step size (h), the better the approximation. In case of say, PID motor control, the step size is proportional to the time intervals at which the new sample value of the position (or speed) is obtained. Using the above equation to compute the differential, three samples are necessary (present value and the last two past values). The subroutine "Diff" is implemented so that 1/2h factor is stored already in a RAM location (location DiffK) as 1/2h and not as "h" because it is more efficient to multiply than divide.

After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, differentiation may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Diff" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

TABLE 6: DOUBLE PRECISION SQUARE ROOT

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Sqrt	22	3300 (approx.)	6	used

TABLE 7: BCD ROUTINES

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
BCDtoB	BCD to Binary	30	112	0	used
B2_BCD_Looped	Binary to BCD (16 bit) looped code	32	750	1	used
B2_BCD_Straight	Binary to BCD (16 bit) straight line code	44	572	1	used
BinBCD	Binary to BCD (8 bit)	10	62	0	unused
BCDAdd	BCD addition	5	5	0	used

NUMERICAL INTEGRATION

This routine performs numerical integration using Simpson's Three-Eighths Rule. This is a third order approximation for the function, whose integral is to be computed at a given point. Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. Given a function f(t), its integral over a range t_0 to t_3 is represented as:

t3

$\int$f(t)dt. This function is approximated as follows:

t0

Simpson's Three-Eighths Rule:

t3

$$\int f(t)dt = \frac{3h}{8}[f(t_0) + 3f(t1) + 3f(t2) + f(t3)]$$

(t_0)

The constant 3h/8 can be computed before hand and stored in a RAM location (in location IntgKLo and IntgKHi as a 16-bit number). After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, integration may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Integrate" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

PSEUDO RANDOM NUMBER GENERATOR

This routine (subroutine "Random 16" provided in ARITH.ASM) generates a pseudo random number sequence. The random points are generated using a 16 bit register and left shifting the contents with the LSB set as shown by the following schematic.

As a test, the random points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PIC16/17 series). The autocorelation of the captured data is computed using a stand alone program and is shown in Figure 2. From this figure, it can be seen that the data has a strong autocorrelation only at the origin and sharply approaches to zero within a few points. This demonstrates the randomness of the data captured.

FIGURE 1

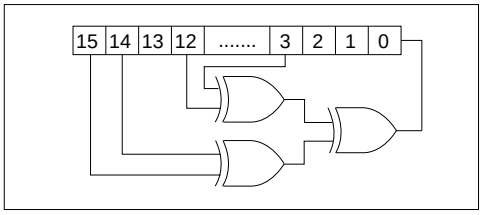


TABLE 8: DIFFERENTIATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Diff	Numerical Differentiation	34	365	10	used

TABLE 9: INTEGRATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Integrate	Numerical Integration	39	370	12	used

FIGURE 2 - AUTOCORELATION OF THE DATA POINTS GENERATED BY THE RANDOM NUMBER GENERATOR

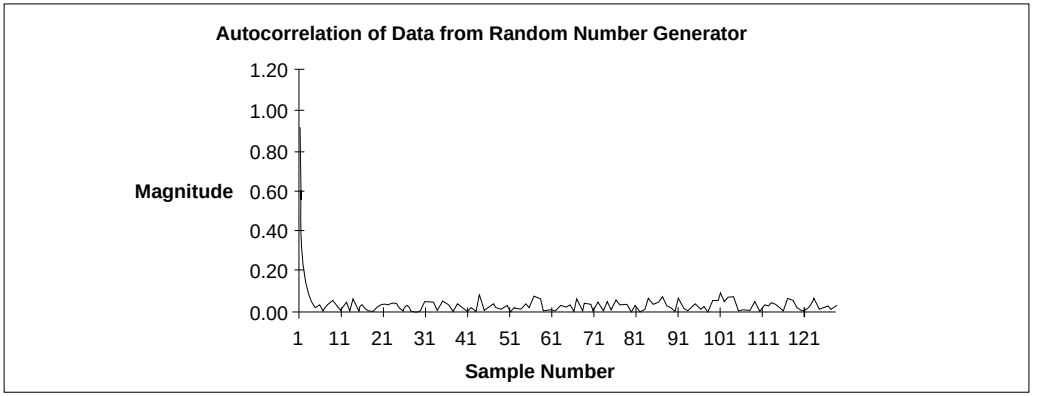
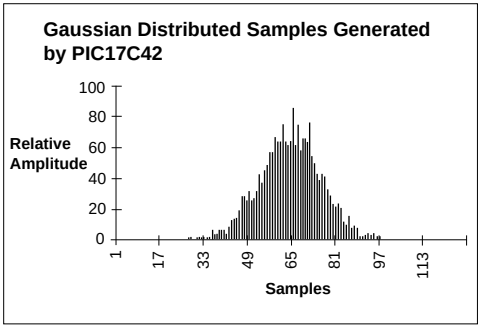


TABLE 10: RANDOM DOUBLE GENERATOR

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Random16	Pseudo Random Number Generator	12	12	0	used
Gauss	Gaussian Random Number Generator	21	452	4	used

PN (pseudo noise) sequences are widely used in digital communication systems for synchronization. These code words can also be used for data scrambling because of their good correlation properties. An interesting application of these sequences is system integrity. For example, these sequences can be regularly transmitted to a processor whose watchdog timer will time out if, say, two consecutive PN sequences do not match.

FIGURE 3 - HISTOGRAM OF THE DATA GENERATED BY THE GAUSSIAN GENERATOR



GAUSSIAN DISTRIBUTED RANDOM NUMBER GENERATOR

This routine (subroutine "Gauss" provided in ARITH.ASM) generates a sequence of random numbers with a characteristic of a normal distribution (Gaussian distributed points). This routine calls the pseudo random number generator ("random16") to obtain a near uniformly distributed random points and from these points, the Gaussian distributed points are generated. The method of generating Gaussian points is based on the "Central Limit Theorem", which states that an ensemble of average weighted sum of a sequence of uncorrelated samples tends to have a Gaussian distribution.

As a test, the Gaussian points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PIC16/17 series). A plot of the points captured is shown in Figure 3, which shows that the random points generated have the characteristics of a Gaussian distribution.

Author: Amar Palacherla
Logic Products Division

APPENDIX A: GENERAL PURPOSE MATH ROUTINES LISTING FILE OF ARITH.ASM

```
MPASM B0.54                                                          PAGE  1
General Purpose Math Routines For PIC17C42 : Ver 1.0
0001                 #define PAGE    EJECT
                      TITLE   “General Purpose Math Routines For PIC17C42 : Ver 1.0”
                      LIST    P=17C42, C=120, T=ON, L=0, R=DEC
                     ;
                        include “17c42.h”
                     ;
                     ;*****************************************************************
                     ;       Define RAM Locations necessary For the “ARITH.ASM”
                     ;   RAM locations should be defined before calling the library math
                     ; routines
                     ;*****************************************************************
                     ;
0001            MODE_FAST       equ     TRUE
0000            SIGNED          equ     FALSE
                     ;
                     ;*****************************************************************
                     ;
                     if MODE_FAST
                      CBLOCK  0x18
0018 0004             ACCaLO, ACCaHI, ACCbLO, ACCbHI     ; Ram Locations for Arithmetic
001C 0004             ACCcLO, ACCcHI, ACCdLO, ACCdHI     ;     Routines
                        ENDC
                      else
                     CBLOCK  0x20
                          ACCaLO, ACCaHI, ACCbLO, ACCbHI
                          ACCcLO, ACCcHI, ACCdLO, ACCdHI
                       ENDC
                      endif
                     ;
                      CBLOCK
0020 0004             tempLo, tempHi, count, sign
                        ENDC
                      CBLOCK
0024 0002            NumLo, NumHi
0026 0001            iterCnt
                        ENDC
                     ;
                       CBLOCK                           ; RAM locations for “Diff” routine
0027 0003                  XnLo, XnHi, Xn_1_Lo
002A 0003                  Xn_1_Hi, Xn_2_Lo, Xn_2_Hi
002D 0002                  DiffKLo, DiffKHi             ; DiffK = h = Step Size
002F 0002                  DiffLo, DiffHi
                        ENDC
                     ;
                       CBLOCK                           ; RAM Locations for “Integrate”
0031 0004                  X0Lo, X0Hi, X1Lo, X1Hi       ; Routine
0035 0004                  X2Lo, X2Hi, X3Lo, X3Hi
0039 0002                  IntgKLo, IntgKHi             ; INTEGRATE CONST = 3*h/8
003B 0002                  IntgLo, IntgHi
                        ENDC
                    ;
                    ;*****************************************************************
                    ;
0018                mulcnd  equ     ACCaLO
0019                mulplr  equ     ACCaHI
001A               L_byte   equ     ACCbLO
001B               H_byte   equ     ACCbHI
                    ;
000A               _LUPCNT  equ     10              ; Set Desired Number of iterations
001E               SqrtLo   equ     ACCdLO          ; for Square Root Routine(NEWTON Iterations)
001F               SqrtHi   equ     ACCdHI
                    ;
                    ;  Define RAM locations for the Random Number Generators
```

```
                        ;
0018                    RandLo   equ     ACCaLO
0019                    RandHi   equ     ACCaHI          ; 16 bit Pseudo Random Number
001B                    GaussHi  equ     ACCbHI
001A                    GaussLo  equ     ACCbLO          ; 16 bit Gaussian distributed number
0020                    GaussTmp equ     tempLo
                        ;

                                ORG     0x0000
                        ;*****************************************************************
                        ;                       Math Routines Test Program
                        ;*****************************************************************
                        ;
                        ;       Load constant values to ACCa & ACCb for testing
                        ;
                        main
0000 E02D                       call    loadAB          ; result of adding ACCb+ACCa->ACCb
0001 E036                       call    D_add           ; Here Accb = 81FE
                        ;
0002 E02D                       call    loadAB          ; result of subtracting ACCb - ACCa->ACCb
0003 E03B                       call    D_sub           ; Here Accb = 7E00
                        ;
0004 E02D                       call    loadAB          ; result of multiplying ACCb*ACCa >(ACCd,ACCc)
0005 E050                       call    D_mpyS          ; Here (ACCd,ACCc) = 00FF 7E01
                        ;
0006 E02D                       call    loadAB          ; result of multiplying ACCb*ACCa->(ACCd,ACCc)
0007 E065                       call    D_mpyF          ; Here (ACCd,ACCc) = 00FF 7E01
                        ;
0008 E02D                       call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
0009 E119                       call    D_divS          ; Here (ACCd,ACCc) = 0040 003f
                        ;
000A E02D                       call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
000B E138                       call    D_divF          ; Here (ACCd,ACCc) = 0040 003f
                        ;
000C B0F3                       movlw   0xf3
000D 0125                       movwf   NumHi
000E B0F6                       movlw   0xf6            ; Set input test number = 62454
000F 0124                       movwf   NumLo           ;  = F3F6h
0010 E27D                       call    Sqrt            ; result = 00F9h = 249 (in SqrtLo)
                        ;                               ; exact sqrt(62454) = 249.9
                        ;
0011 B0FF                       movlw   0xff
0012 0119                       movwf   mulplr          ; multiplier (in mulplr) = 0FF
0013 B0FF                       movlw   0xff            ; multiplicand(W Reg )   = 0FF
0014 0118                       movwf   mulcnd
0015 E293                       call    mpy8x8_F        ; The result 0FF*0FF = FE01 is in locations
                        ;                               ; H_byte & L_byte
0016 B0FF                       movlw   0xff
0017 0119                       movwf   mulplr          ; multiplier (in mulplr) = 0FF
0018 B0FF                       movlw   0xff            ; multiplicand(W Reg )   = 0FF
0019 0118                       movwf   mulcnd
001A E2B8                       call    mpy8x8_S        ; The result 0FF*0FF = FE01 is in
                        ;                               ; H_byte & L_byte
                        ;       Test The Random Number Generators
                        ;       Capture data into trace buffer by TABLE WRITES to a
                        ;       dummy Program Memory location
                        ;
001B B0FF                       movlw   0xff
001C 010D                       movwf   tblptrl
001D B05F                       movlw   0x5f
001E 010E                       movwf   tblptrh
                        ;
001F B030                       movlw   0x30
0020 0119                       movwf   RandHi
0021 B045                       movlw   0x45
0022 0118                       movwf    RandLo
                        ;
0023 C028                       goto    GaussPoint
                        ;
```

```
                        RandPoint
0024 E311                       call    Random16
0025 A418                       tlwt    _LOW,RandLo       ; only for data capture
0026 AE19                       tablwt  _HIGH,0,RandHi    ; using PICMASTER
0027 C024                       goto    RandPoint
                        ;
                        GaussPoint
0028 E31E                       call    Gauss
0029 A41A                       tlwt    _LOW,GaussLo      ; only for data capture
002A AE1B                       tablwt  _HIGH,0,GaussHi   ; using PICMASTER
002B C028                       goto    GaussPoint
                        ;
002C C02C               self    goto    self              ; End Of Test Routines
                        ;
                        loadAB
002D B001                       movlw   0x01
002E 0119                       movwf   ACCaHI
002F B0FF                       movlw   0xff              ; loads ACCa = 01FF
0030 0118                       movwf   ACCaLO
                        ;
0031 B07F                       movlw   0x7f
0032 011B                       movwf   ACCbHI
0033 B0FF                       movlw   0xFF              ; loads ACCb = 7FFF
0034 011A                       movwf   ACCbLO
0035 0002                       return
                        ;
                        ;*****************************************************************
                        ;                 Double Precision Arithmetic Routines
                        ;
                        ;    Routines : Addition, Subtraction, Multiplication ,Division
                        ;               Square Root
                        ;
                        ;               NOTE :  MODE_FAST must first be set to either
                        ;                       TRUE or FALSE
                        ;
                        ;  MODE_FAST determines the RAM address locations of ACCa thru ACCd
                        ;
                        ;  If MODE_FAST is set TRUE, data transfers can be done efficiently
                        ;  using “MOVFP” & “MOVPF” instructions instead of indirectly moving
                        ;  at first to W Reg and then to the desired RAM locations
                        ;
                        ;               The speed increase using this way of locating ACCa to
                        ;         ACCd will result in a saving of about 20 Cycles/filter stage
                        ;         In this case ( a 2 stage filter), it is faster by 40 Cycles
                        ;
                        ;  If due to other constraints, ACCa thru ACCd cannot be set at
                        ; address 0x18 to 0x1f, then the user is required to set
                        ; MODE_FAST to FALSE
                        ;
                        ;*****************************************************************
                        ;                 Double Precision Addition
                        ;
                        ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                        ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                        ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                        ;      (c) CALL D_add
                        ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                        ;
                        ;   Performance :
                        ;               Program Memory  :  4 (excluding call & return)
                        ;               Clock Cycles    :  4 (excluding call & return)
                        ;               W Register      :  Used
                        ;               Scratch RAM     :  0
                        ;
                        ;*****************************************************************;
                        ;
                        D_add
```

```
0036 6018                   movfp    ACCaLO,wreg
0037 0F1A                   addwf    ACCbLO            ;addwf lsb
0038 6019                   movfp    ACCaHI,wreg
0039 111B                   addwfc   ACCbHI            ;addwf msb with carry
003A 0002                   return
                    ;
                    ;*****************************************************************
                    ;       ¡           Double Precision Subtraction
                    ;
                    ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                    ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                    ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                    ;      (c) CALL D_sub
                    ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                    ;
                    ;   Performance :
                    ;               Program Memory  :   4  (excluding call & return )
                    ;               Clock Cycles    :   4  (excluding call & return )
                    ;               W Register      :   Used
                    ;               scratch RAM     :   0
                    ;*****************************************************************;
                    ;
                    D_sub
003B 6018                   movfp    ACCaLO,wreg
003C 051A                   subwf    ACCbLO
003D 6019                   movfp    ACCaHI,wreg
003E 031B                   subwfb   ACCbHI
003F 0002                   return
                    ;
                    ;*****************************************************************
                    ;               Function to negate a 16 bit integer
                    ;  The two 8 bit integers are assumed to be in 2 consecutive
                    ; locations. Before calling this routine, FSR0 should be loaded with
                    ; the address of the lower byte.
                    ;       Assume that ALUSTA register is set for no autoincrement of
                    ; FSR0.
                    ;*****************************************************************
                    ;
                    negateAlt
0040 6000                   movfp    indf0,wreg
0041 8D04                   bcf      _fs1
0042 2D00                   negw     indf0
0043 8504                   bsf      _fs1
0044 6000                   movfp    indf0,wreg
0045 2900                   clrf     indf0
0046 0300                   subwfb   indf0
0047 0002                   return
                    ;
                    negate
0048 1300                   comf     indf0
0049 8D04                   bcf      _fs1
004A 1500                   incf     indf0
004B 8504                   bsf      _fs1
004C 9A04                   btfsc    _z
004D 0700                   decf     indf0
004E 1300                   comf    indf0
004F 0002                   return
                    ;
                    ;*****************************************************************
                    ;                       Double Precision Multiplication
                    ;
                    ;               ( Optimized for Code : Looped Code )
                    ;
                    ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
                    ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                    ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                    ;      (c) CALL D_mpyS
                    ;      (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
```

```
                    ;
                    ;       Performance :
                    ;       Program Memory  :   21 (UNSIGNED)
                    ;                           52 (SIGNED)
                    ;       Clock Cycles    :   242 (UNSIGNED :excluding CALL & RETURN)
                    ;                       :   254 (SIGNED :excluding CALL & RETURN)
                    ;       Scratch RAM     :   1 (used only if SIGNED arithmetic)
                    ;
                    ;       Note : The above timing is the worst case timing, when the
                    ;              register ACCb = FFFF. The speed may be improved if
                    ;              the register ACCb contains a number ( out of the two
                    ;              numbers ) with less number of 1s.
                    ;
                    ;               Double Precision Multiply ( 16x16 -> 32 )
                    ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
                    ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                    ;*****************************************************************
                    ;
                     D_mpyS         ;results in ACCd(16 msb's) and ACCc(16
                    ;

                    if      SIGNED
                    CALL    S_SIGN
                    endif
                    ;
0050 2922                   clrf    count
0051 8422                   bsf     count,4         ; set count = 16
                    ;
                    if MODE_FAST
0052 5A20                   movpf   ACCbLO,tempLo
0053 5B21                   movpf   ACCbHI,tempHi
                    else
                            movfp   ACCbLO,wreg
                            movwf   tempLo
                            movfp   ACCbHI,wreg
                            movwf   tempHi
                    endif
0054 291F                   clrf    ACCdHI
0055 291E                   clrf    ACCdLO
                    ;
                    ;       shift right and addwf 16 times
                    ;
                    mpyLoop
0056 1921                   rrcf    tempHi
0057 1920                   rrcf    tempLo
0058 9004                   btfss   _carry
0059 C05E                   goto    NoAdd           ; LSB is 0, so no need to addwf
005A 6018                   movfp   ACCaLO,wreg
005B 0F1E                   addwf   ACCdLO          ;addwf lsb
005C 6019                   movfp   ACCaHI,wreg
005D 111F                   addwfc  ACCdHI          ;addwf msb
                    NoAdd
005E 191F                   rrcf    ACCdHI
005F 191E                   rrcf    ACCdLO
0060 191D                   rrcf    ACCcHI
0061 191C                   rrcf    ACCcLO
0062 1722                   decfsz  count
0063 C056                   goto    mpyLoop
                    ;
                    if SIGNED
                            btfss   sign,MSB
                            return
                            comf    ACCcLO
                            incf    ACCcLO
                            btfsc   _z
                            decf    ACCcHI
                            comf    ACCcHI
                            btfsc   _z
                            decf                    ACCdLO
```

```
                        comf    ACCdLO
                        btfsc   _z
                        decf    ACCdHI
                        comf    ACCdHI
                        return
                endif
0064 0002               return
                endif
                ;
                ;  Assemble this section only if Signed Arithmetic Needed
                ;
                if    SIGNED
                ;
                S_SIGN
                        movfp   ACCaHI,wreg
                        xorwf   ACCbHI,w
                        movwf   sign            ; MSB of sign determines whether signed
                        btfss   ACCbHI,MSB      ; if MSB set go & negate ACCb
                        goto    chek_A
                        comf    ACCbLO
                        incf    ACCbLO
                        btfsc   _z              ; negate ACCb
                        decf    ACCbHI
                        comf    ACCbHI
                ;
                chek_A
                        btfss   ACCaHI,MSB      ; if MSB set go & negate ACCa
                        return
                        comf    ACCaLO
                        incf    ACCaLO
                        btfsc   _z              ; negate ACCa
                        decf    ACCaHI
                        comf    ACCaHI
                        return
                ;
                endif
                ;
                ;*****************************************************************
                ;                       Double Precision Multiplication
                ;
                ;               ( Optimized for Speed : straight Line Code )
                ;
                ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
                ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                ;      (c) CALL D_mpy
                ;      (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
                ;
                ;   Performance :
                ;           Program Memory  :   179 (UNSIGNED)
                ;                               204 (SIGNED)
                ;           Clock Cycles    :   176 (UNSIGNED :excluding CALL & RETURN)
                ;                           :   183 (SIGNED :excluding CALL & RETURN)
                ;
                ;       Note : The above timing is the worst case timing, when the
                ;              register ACCb = FFFF. The speed may be improved if
                ;              the register ACCb contains a number ( out of the two
                ;              numbers ) with less number of 1s.
                ;
                ;              The performance specs are for Unsigned arithmetic ( i.e,
                ;              with “SIGNED equ  FALSE “).
                ;
                ;              Upon return from subroutine, the input registers
```

```
;
;
;*********************************************************
;               Multiplication Macro
;*********************************************************
;
mulMac  MACRO
    variable i

    i = 0

    if SIGNED
      .while i < 15
    else
      .while i < 16
    endif
      .if i < 8
        btfss   ACCbLO,i        ; test low byte
      .else
        btfss   ACCbHI,i-8      ; test high byte
      .fi
        goto    NoAdd#v(i)      ; LSB is 0, so no need to addwf
        movfp   ACCaLO,wreg
        addwf   ACCdLO          ;addwf lsb
        movfp   ACCaHI,wreg
        addwfc  ACCdHI          ;addwf msb
    NoAdd#v(i)
        rrcf    ACCdHI
        rrcf    ACCdLO
        rrcf    ACCcHI
        rrcf    ACCcLO
        bcf     _carry
        i = i+1
      .endw
    if SIGNED
        rrcf    ACCdHI
        rrcf    ACCdLO
        rrcf    ACCcHI
        rrcf    ACCcLO
        bcf     _carry
    endif
    ;
    ENDM
    ;
    ;*************************************************************
    ;                       Double Precision Negate Macros
    ;*************************************************************
    AltNegMac   MACRO           fileRegLo,fileRegHi
        movfp   fileRegLo,wreg
        negw    fileRegLo
        movfp   fileRegHi,wreg
        clrf    fileRegHi
        subwfb  fileRegHi
        ENDM

    ;
    negMac  MACRO   fileRegLo, fileRegHi
        comf    fileRegLo       ; negate FileReg ( -FileReg -> FileReg )
        incf    fileRegLo
        btfsc   _z
        decf    fileRegHi
        comf    fileRegHi
    ENDM
    ;
    NegMac32    MACRO   x3,x2,x1,x0
        movfp   x3,wreg
        negw    x3
        movfp   x2,wreg
        clrf    x2
```

```
                        subwfb   x2
                        movfp    x1,wreg
                        clrf     x1
                        subwfb   x1
                        movfp    x0,wreg
                        clrf     x0
                        subwfb   x0
                        ENDM
                ;
                ;*****************************************************************;
                ;               Double Precision Multiply ( 16x16 -> 32 )
                ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
                ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                ;
                D_mpyF                          ;results in ACCd(16 msb's) and ACCc(16
                ;
                if   SIGNED
                ;
                        movfp    ACCaHI,wreg
                        xorwf    ACCbHI,w
                        movwf    sign
                        btfss    ACCbHI,MSB      ; if MSB set go & negate ACCb
                        goto     chek_A_MSB_MPY
                ;
                        negMac   ACCbLO,ACCbHI
                ;
                chek_A_MSB_MPY
                        btfss    ACCaHI,MSB      ; if MSB set go & negate ACCa
                        goto     continue_MPY
                        negMac   ACCaLO,ACCaHI
                ;
                        endif
                ;
                continue_MPY
0065 291F               clrf     ACCdHI
0066 291E               clrf     ACCdLO
0067 8804               bcf      carry
                ;
                ;       use the mulMac macro 16 times
                ;
                mulMac
0000                    variable i
0000                    i = 0
                     if SIGNED
                        .while i < 15
                     else
                        .while i < 16
                endif
                        .if i < 8
                        btfss   ACCbLO,i         ; test low byte
                        .else
                        btfss    ACCbHI,i-8      ; test high byte
                        .fi
                        goto     NoAdd#v(i)      ; LSB is 0, so no need to addwf
                        movfp    ACCaLO,wreg
                        addwf    ACCdLO          ;addwf lsb
                        movfp    ACCaHI,wreg
                        addwfc   ACCdHI          ;addwf msb
                NoAdd#v(i)
                        rrcf     ACCdHI
                        rrcf     ACCdLO
                        rrcf     ACCcHI
                        rrcf     ACCcLO
                        bcf      _carry
                        i = i+1
                .endw
                        .if i < 8
0068 901A               btfss    ACCbLO,i        ; test low byte
                        .else
```

```
                                        btfss      ACCbHI,i-8      ; test high byte

                                    .fi

0069 C06E                           goto    NoAdd0          ; LSB is 0, so no need to addwf
006A 6018                           movfp    ACCaLO,wreg
006B 0F1E                           addwf     ACCdLO         ;addwf lsb
006C 6019                           movfp    ACCaHI,wreg
006D 111F                           addwfc    ACCdHI        ;addwf msb
                    NoAdd0
006E 191F                           rrcf     ACCdHI
006F 191E                           rrcf     ACCdLO
0070 191D                           rrcf     ACCcHI
0071 191C                           rrcf     ACCcLO
0072 8804                           bcf      _carry
0001                                i = i+1

                                    .if i < 8
0073 911A                           btfss      ACCbLO,i      ; test low byte
                                    .else
                                        btfss      ACCbHI,i-8      ; test high byte

                                    .fi

0074 C079                           goto    NoAdd1          ; LSB is 0, so no need to addwf
0075 6018                           movfp    ACCaLO,wreg
0076 0F1E                           addwf     ACCdLO         ;addwf lsb
0077 6019                           movfp    ACCaHI,wreg
0078 111F                           addwfc    ACCdHI        ;addwf msb
                    NoAdd1
0079 191F                           rrcf     ACCdHI
007A 191E                           rrcf     ACCdLO
007B 191D                           rrcf     ACCcHI
007C 191C                           rrcf     ACCcLO
007D 8804                           bcf      _carry
0002                                i = i+1

                                    .if i < 8
007E 921A                           btfss      ACCbLO,i      ; test low byte
                                    .else
                                        btfss      ACCbHI,i-8      ; test high byte

                                    .fi

007F C084                           goto    NoAdd2          ; LSB is 0, so no need to addwf
0080 6018                           movfp    ACCaLO,wreg
0081 0F1E                           addwf     ACCdLO         ;addwf lsb
0082 6019                           movfp    ACCaHI,wreg
0083 111F                           addwfc    ACCdHI        ;addwf msb
                    NoAdd2
0084 191F                           rrcf     ACCdHI
0085 191E                           rrcf     ACCdLO
0086 191D                           rrcf     ACCcHI
0087 191C                           rrcf     ACCcLO
0088 8804                           bcf      _carry
0003                                i = i+1

                                    .if i < 8
0089 931A                           btfss      ACCbLO,i        ; test low byte
                                    .else
                                        btfss      ACCbHI,i-8        ; test high byte

                                    .fi

008A C08F                           goto    NoAdd3            ; LSB is 0, so no need to addwf
008B 6018                           movfp    ACCaLO,wreg
008C 0F1E                           addwf     ACCdLO           ;addwf lsb
008D 6019                           movfp    ACCaHI,wreg
008E 111F                           addwfc    ACCdHI          ;addwf msb
```

```
                        NoAdd3
008F 191F                  rrcf     ACCdHI
0090 191E                  rrcf     ACCdLO
0091 191D                  rrcf     ACCcHI
0092 191C                  rrcf     ACCcLO
0093 8804                  bcf      _carry
0004                       i = i+1
                        .if i < 8
0094 941A                  btfss    ACCbLO,i        ; test low byte
                        .else
                           btfss    ACCbHI,i-8      ; test high byte
                        .fi
0095 C09A                   goto    NoAdd4          ; LSB is 0, so no need to addwf
0096 6018                   movfp   ACCaLO,wreg
0097 0F1E                   addwf   ACCdLO          ;addwf lsb
0098 6019                   movfp   ACCaHI,wreg
0099 111F                  addwfc   ACCdHI          ;addwf msb
                        NoAdd4
009A 191F                  rrcf     ACCdHI
009B 191E                  rrcf     ACCdLO
009C 191D                  rrcf     ACCcHI
009D 191C                  rrcf     ACCcLO
009E 8804                  bcf      _carry
0005                       i = i+1
                        .if i < 8
009F 951A                  btfss    ACCbLO,i        ; test low byte
                        .else
                           btfss    ACCbHI,i-8      ; test high byte
                        .fi
00A0 C0A5                  goto     NoAdd5          ; LSB is 0, so no need to addwf
00A1 6018                  movfp    ACCaLO,wreg
00A2 0F1E                  addwf    ACCdLO          ;addwf lsb
00A3 6019                  movfp    ACCaHI,wreg
00A4 111F                  addwfc   ACCdHI          ;addwf msb
                        NoAdd5
00A5 191F                  rrcf     ACCdHI
00A6 191E                  rrcf     ACCdLO
00A7 191D                  rrcf     ACCcHI
00A8 191C                  rrcf     ACCcLO
00A9 8804                  bcf      _carry
0006                       i = i+1
                        .if i < 8
00AA 961A                  btfss    ACCbLO,i        ; test low byte
                        .else
                           btfss    ACCbHI,i-8      ; test high byte
                        .fi
00AB C0B0                  goto     NoAdd6          ; LSB is 0, so no need to addwf
00AC 6018                  movfp    ACCaLO,wreg
00AD 0F1E                  addwf    ACCdLO          ;addwf lsb
00AE 6019                  movfp    ACCaHI,wreg
00AF 111F                  addwfc   ACCdHI          ;addwf msb
                        NoAdd6
00B0 191F                  rrcf     ACCdHI
00B1 191E                  rrcf     ACCdLO
00B2 191D                  rrcf     ACCcHI
00B3 191C                  rrcf     ACCcLO
00B4 8804                  bcf      _carry
0007                       i = i+1
                        .if i < 8
00B5 971A                  btfss    ACCbLO,i       ; test low byte
                        .else
                           btfss    ACCbHI,i-8      ; test high byte
                        .fi

00B6 C0BB                  goto     NoAdd7          ; LSB is 0, so no need to addwf
00B7 6018                  movfp    ACCaLO,wreg
00B8 0F1E                  addwf    ACCdLO          ;addwf lsb
00B9 6019                   movfp   ACCaHI,wreg
00BA 111F                  addwfc   ACCdHI          ;addwf msb
```

```
                        NoAdd7
00BB 191F                   rrcf    ACCdHI
00BC 191E                   rrcf    ACCdLO
00BD 191D                   rrcf    ACCcHI
00BE 191C                   rrcf    ACCcLO
00BF 8804                   bcf     _carry
0008                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00C0 901B                   btfss   ACCbHI,i-8      ; test high byte
                        .fi
00C1 C0C6                   goto    NoAdd8          ; LSB is 0, so no need to addwf
00C2 6018                   movfp   ACCaLO,wreg
00C3 0F1E                   addwf   ACCdLO          ;addwf lsb
00C4 6019                   movfp   ACCaHI,wreg
00C5 111F                   addwfc  ACCdHI          ;addwf msb
                        NoAdd8
00C6 191F                   rrcf    ACCdHI
00C7 191E                   rrcf    ACCdLO
00C8 191D                   rrcf    ACCcHI
00C9 191C                   rrcf    ACCcLO
00CA 8804                   bcf     _carry
0009                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00CB 911B                   btfss   ACCbHI,i-8      ; test high byte
                        .fi
00CC C0D1                   goto    NoAdd9          ; LSB is 0, so no need to addwf
00CD 6018                   movfp   ACCaLO,wreg
00CE 0F1E                   addwf   ACCdLO          ; addwf lsb
00CF 6019                    movfp  ACCaHI,wreg
00D0 111F                   addwfc  ACCdHI          ; addwf msb
                        NoAdd9
00D1 191F                   rrcf    ACCdHI
00D2 191E                   rrcf    ACCdLO
00D3 191D                   rrcf    ACCcHI
00D4 191C                   rrcf    ACCcLO
00D5 8804                   bcf      _carry
000A                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00D6 921B                    btfss  ACCbHI,i-8      ; test high byte
                        .fi
00D7 C0DC                   goto    NoAdd10         ; LSB is 0, so no need to addwf
00D8 6018                   movfp   ACCaLO,wreg
00D9 0F1E                   addwf   ACCdLO          ; addwf lsb
00DA 6019                   movfp   ACCaHI,wreg
00DB 111F                   addwfc  ACCdHI          ; addwf msb
                        NoAdd10
00DC 191F                   rrcf    ACCdHI
00DD 191E                   rrcf    ACCdLO
00DE 191D                   rrcf    ACCcHI
00DF 191C                   rrcf    ACCcLO
00E0 8804                   bcf     _carry
000B                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00E1 931B                   btfss   ACCbHI,i-8      ; test high byte
                        .fi
00E2 C0E7                   goto    NoAdd11         ; LSB is 0, so no need to addwf
00E3 6018                   movfp   ACCaLO,wreg
00E4 0F1E                   addwf   ACCdLO          ;addwf lsb
00E5 6019                   movfp   ACCaHI,wreg
```

```
00E6 111F                   addwfc  ACCdHI          ;addwf msb
                          NoAdd11
00E7 191F                   rrcf    ACCdHI
00E8 191E                   rrcf    ACCdLO
00E9 191D                   rrcf    ACCcHI
00EA 191C                   rrcf    ACCcLO
00EB 8804                   bcf     _carry
000C                        i = i+1
                          .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                          .else
00EC 941B                   btfss   ACCbHI,i-8      ; test high byte
                          .fi
00ED C0F2                   goto    NoAdd12         ; LSB is 0, so no need to addwf
00EE 6018                   movfp   ACCaLO,wreg
00EF 0F1E                   addwf   ACCdLO          ;addwf lsb
00F0 6019                   movfp   ACCaHI,wreg
00F1 111F                   addwfc  ACCdHI          ;addwf msb
                          NoAdd12
00F2 191F                   rrcf    ACCdHI
00F3 191E                   rrcf    ACCdLO
00F4 191D                   rrcf    ACCcHI
00F5 191C                   rrcf    ACCcLO
00F6 8804                   bcf     _carry
000D                        i = i+1
                          .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                          .else
00F7 951B                   btfss   ACCbHI,i-8      ; test high byte
                          .fi
00F8 C0FD                   goto    NoAdd13         ; LSB is 0, so no need to addwf
00F9 6018                   movfp   ACCaLO,wreg
00FA 0F1E                   addwf   ACCdLO          ;addwf lsb
00FB 6019                   movfp   ACCaHI,wreg
00FC 111F                   addwfc  ACCdHI          ;addwf msb
                          NoAdd13
00FD 191F                   rrcf    ACCdHI
00FE 191E                   rrcf    ACCdLO
00FF 191D                   rrcf    ACCcHI
0100 191C                   rrcf    ACCcLO
0101 8804                   i = i+1
                          .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                          .else
0102 961B                   btfss   ACCbHI,i-8      ; test high byte
                          .fi
0103 C108                   goto    NoAdd14         ; LSB is 0, so no need to addwf
0104 6018                   movfp   ACCaLO,wreg
0105 0F1E                   addwf   ACCdLO          ;addwf lsb
0106 6019                   movfp   ACCaHI,wreg
0107 111F                   addwfc  ACCdHI          ;addwf msb
                          NoAdd14
0108 191F                   rrcf    ACCdHI
0109 191E                   rrcf    ACCdLO
010A 191D                   rrcf    ACCcHI
010B 191C                   rrcf    ACCcLO
010C 8804                   bcf     _carry
000F                        i = i+1
                          .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                          .else
010D 971B                   btfss   ACCbHI,i-8      ; test high byte
                          .fi
010E C113                   goto    NoAdd15         ; LSB is 0, so no need to addwf
010F 6018                   movfp   ACCaLO,wreg
0110 0F1E                   addwf   ACCdLO          ;addwf lsb
0111 6019                   movfp   ACCaHI,wreg
```

```
0112 111F                   addwfc  ACCdHI          ;addwf msb
                         NoAdd15
0113 191F                   rrcf    ACCdHI
0114 191E                   rrcf    ACCdLO
0115 191D                   rrcf    ACCcHI
0116 191C                   rrcf    ACCcLO
0117 8804                   bcf     _carry
0010                        i = i+1
                        if SIGNED
                            rrcf    ACCdHI
                            rrcf    ACCdLO
                            rrcf    ACCcHI
                            rrcf    ACCcLO
                            bcf     _carry
                        endif
                    ;
                    ;
                        if SIGNED
                            btfss   sign,MSB        ; negate (ACCc,ACCd)
                            return
                    NegMac32 ACCcHI,ACCcLO,ACCdHI, ACCdLO
                            return
                            else
0118 0002                   return
                    endif
                    ;
                    ;*****************************************************************
                    ;                       Double Precision Division
                    ;
                    ;               ( Optimized for Code : Looped Code )
                    ;
                    ;*****************************************************************;
                    ;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
                    ;                                               Remainder in ACCc (16 bits)
                    ;      (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
                    ;      (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
                    ;      (c) CALL D_div
                    ;      (d) The 16 bit result is in location ACCbHI & ACCbLO
                    ;      (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
                    ;
                    ;   Performance :
                    ;            Program Memory  :   31 (UNSIGNED)
                    ;                                39 (SIGNED)
                    ;               Clock Cycles    : 300 (UNSIGNED : excluding CALL & RETURN)
                    ;                                 312 (SIGNED : excluding CALL & RETURN)
                    ;
                    ;       NOTE :
                    ;            The performance specs are for Unsigned arithmetic ( i.e,
                    ;               with “SIGNED equ  FALSE “).
                    ;
                    ;*****************************************************************
                    ;       Double Precision Divide ( 16/16 -> 16 )
                    ;
                    ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ):16 bit output
                    ; with Quotiont in ACCb(ACCbHI,ACCbLO)and Remainder in ACCc (ACCcHI,ACCcLO).
                    ;
                    ;               B/A = (Q) + (R)/A
                    ;       or      B = A*Q + R
                    ;
                    ;                       where   B :     Numerator
                    ;                               A :     Denominator
                    ;                               Q :     Quotiont (Integer Result)
                    ;                               R :     Remainder
                    ;
                    ;      Note :  Check for ZERO Denominator or Numerator is not performed
                    ;               A ZERO Denominator will produce incorrect results
                    ;
                    ;  SIGNED Arithmetic :
                    ;  In case of signed arithmetic, if either
```

```
                        ;  numerator or denominator is negative, then both Q & R are
                        ;  represented as negative numbers
                        ;               -(B/A) = -(Q) + (-R)/A
                        ;       or      -B = (-Q)*A + (-R)
                        ;
                        ;*****************************************************************
                        ;
                        D_divS
                        ;
0119 8404                  bsf      _fs0
011A 8504                  bsf      _fs1            ; set no auto-incrment for fsr0

                        if    SIGNED
                           CALL     S_SIGN
                        endif
                        ;
011B 2922                  clrf     count
011C 8422                  bsf      count,4         ; set count = 16
011D 291D                  clrf     ACCcHI
011E 291C                  clrf     ACCcLO
011F 291E                  clrf     ACCdLO
0120 291F                  clrf     ACCdHI
                        ;
                        ;  Looped code
                        ;
                        divLoop
0121 8804                  bcf      _carry
0122 1B1A                  rlcf     ACCbLO
0123 1B1B                  rlcf     ACCbHI
0124 1B1C                  rlcf     ACCcLO
0125 1B1D                  rlcf     ACCcHI
0126 6019                  movfp    ACCaHI,wreg
0127 041D                  subwf    ACCcHI,w        ;check if a>c
0128 9204                  btfss    _z
0129 C12C                  goto     notz
012A 6018                  movfp    ACCaLO,wreg
012B 041C                  subwf    ACCcLO,w        ; if msb equal then check lsb
                        notz
012C 9004                  btfss    _carry          ; carry set if c>a
012D C133                  goto     nosub           ; if c < a
                        subca
012E 6018                  movfp    ACCaLO,wreg     ; c-a into c
012F 051C                  subwf    ACCcLO
0130 6019                  movfp    ACCaHI,wreg
0131 031D                  subwfb   ACCcHI
0132 8004                  bsf      _carry          ;shift a 1 into d (result)
                        nosub
0133 1B1E                  rlcf     ACCdLO
0134 1B1F                  rlcf     ACCdHI
0135 1722                  decfsz   count
0136 C121                  goto     divLoop

                        ;
                        if SIGNED
                           btfss    sign,MSB
                           return
                           movlw    ACCcLO
                           movwf    fsr0
                           call     negate
                           movlw    ACCdLO
                           movwf    fsr0
                           call     negate
                           return
                           else
0137 0002                  return
                        endif
                        ;
                        ;*****************************************************************
                        ;                       Double Precision Division
```

```
;
;               ( Optimized for Speed : straight Line Code )
;
;*****************************************************************;
;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
;                                       Remainder in ACCc (16 bits)
;
;      (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
;      (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
;      (c) CALL D_div
;      (d) The 16 bit result is in location ACCbHI & ACCbLO
;      (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;
;               B/A = (Q) + (R)/A
;       or      B = A*Q + R
;
;                       where   B :     Numerator
;                               A :     Denominator
;                               Q :     Quotiont (Integer Result)
;                               R :     Remainder
;
;       Note :  Check for ZERO Denominator or Numerator is not performed
;               A ZERO Denominator will produce incorrect results
;
;       SIGNED Arithmetic :
;                       In case of signed arithmetic, if either
;  numerator or denominator is negative, then both Q & R are
;  represented as negative numbers
;               -(B/A) = -(Q) + (-R)/A
;       or      -B = (-Q)*A + (-R)
;
;   Performance :
;            Program Memory     : 325 (UNSIGNED)
;                                 354 (SIGNED)
;               Clock Cycles    : 250 (UNSIGNED : excluding CALL & RETURN)
;                               : 260 (SIGNED : excluding CALL & RETURN)
;
;*****************************************************************;
;       division macro
;
divMac  MACRO
   variable i

       i = 0
      .while i < 16
;
           bcf       _carry
           rlcf      ACCbLO
           rlcf      ACCbHI
           rlcf      ACCcLO
           rlcf      ACCcHI
           movfp     ACCaHI,wreg
           subwf     ACCcHI,w            ;check if a>c
           btfss     _z
           goto      notz#v(i)
           movfp     ACCaLO,wreg
           subwf     ACCcLO,w            ;if msb equal then check lsb
notz#v(i)  btfss     _carry               ;carry set if c>a
           goto      nosub#v(i)           ; if c < a
subca#v(i) movfp     ACCaLO,wreg          ;c-a into c
           subwf     ACCcLO
           movfp     ACCaHI,wreg
           subwfb    ACCcHI
           bsf       _carry              ;shift a 1 into d (result)
nosub#v(i) rlcf      ACCdLO
           rlcf      ACCdHI
           i=i+1
           .endw
;
```

```
                          ENDM
                        ;
                        ;*****************************************************************
                        ;       Double Precision Divide ( 16/16 -> 16 )
                        ;
                        ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
                        ;          with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
                        ;          ACCcHI,ACCcLO).
                        ;
                        ;   NOTE:  Before calling this routine, the user should make sure that
                        ;          the Numerator(ACCb) is greater than Denominator(ACCa). If
                        ;          the case is not true, the user should scale either Numerator
                        ;          or Denominator or both such that Numerator is greater than
                        ;          the Denominator.
                        ;
                        ;
                        ;*****************************************************************
                        ;
                        D_divF
                        ;
                           if   SIGNED
                            movfp    ACCaHI,wreg
                            xorwf    ACCbHI,w
                            movwf    sign
                            btfss    ACCbHI,MSB          ; if MSB set go & negate ACCb
                            goto     chek_A_MSB_DIV
                        ;
                            negMac  ACCbLO,ACCbHI
                        ;
                        chek_A_MSB_DIV
                            btfss   ACCaHI,MSB           ; if MSB set go & negate ACCa
                            goto    continue_DIV
                            negMac  ACCaLO,ACCaHI
                        endif
                        ;
                        continue_DIV
0138 291D                   clrf     ACCcHI
0139 291C                   clrf     ACCcLO
013A 291E                   clrf     ACCdLO
013B 291F                   clrf     ACCdHI
                        ;
                        ; straight line code : using the macro divMac
                        ;
                            divMac
0000                        variable i

0000                        i = 0
                              .while i < 16
                        ;
                            bcf      _carry
                            rlcf     ACCbLO
                            rlcf     ACCbHI
                            rlcf     ACCcLO
                            rlcf     ACCcHI
                            movfp    ACCaHI,wreg
                            subwf    ACCcHI,w            ;check if a>c
                            btfss    _z
                            goto     notz#v(i)
                            movfp    ACCaLO,wreg
                            subwf    ACCcLO,w            ;if msb equal then check lsb
                         notz#v(i)   btfss    _carry     ;carry set if c>a
                            goto     nosub#v(i)          ; if c < a
                          subca#v(i)  movfp     ACCaLO,wreg   ;c-a into c
                            subwf    ACCcLO
                            movfp    ACCaHI,wreg
                            subwfb   ACCcHI
                            bsf      _carry              ;shift a 1 into d (result)
                          nosub#v(i)  rlcf      ACCdLO
                                      rlcf      ACCdHI
```

```
                              i=i+1
                             .endw

                          ;
013C 8804                           bcf       _carry
013D 1B1A                           rlcf      ACCbLO
013E 1B1B                           rlcf      ACCbHI
013F 1B1C                           rlcf      ACCcLO
0140 1B1D                           rlcf      ACCcHI
0141 6019                           movfp     ACCaHI,wreg
0142 041D                           subwf     ACCcHI,w    ;check if a>c
0143 9204                           btfss      _z
0144 C147                           goto      notz0
0145 6018                           movfp     ACCaLO,wreg
0146 041C                           subwf     ACCcLO,w          ;if msb equal then check lsb
0147 9004                 notz0     btfss     _carry            ;carry set if c>a
0148 C14E                           goto      nosub0            ; if c < a
0149 6018                 subca0    movfp     ACCaLO,wreg       ;c-a into c
014A 051C                           subwf     ACCcLO
014B 6019                           movfp     ACCaHI,wreg
014C 031D                           subwfb    ACCcHI
014D 8004                           bsf       _carry            ;shift a 1 into d (result)
014E 1B1E                 nosub0    rlcf      ACCdLO
014F 1B1F                           rlcf      ACCdHI
0001                      i = i+1
                          ;
0150 8804                           bcf       _carry
0151 1B1A                           rlcf      ACCbLO
0152 1B1B                           rlcf      ACCbHI
0153 1B1C                           rlcf      ACCcLO
0154 1B1D                           rlcf      ACCcHI
0155 6019                           movfp     ACCaHI,wreg
0156 041D                           subwf     ACCcHI,w          ;check if a>c
0157 9204                           btfss     _z
0158 C15B                           goto      notz1
0159 6018                           movfp     ACCaLO,wreg
015A 041C                           subwf     ACCcLO,w          ;if msb equal then check lsb
015B 9004                 notz1     btfss     _carry            ;carry set if c>a
015C C162                           goto      nosub1            ; if c < a
015D 6018                 subca1    movfp     ACCaLO,wreg       ;c-a into c
015E 051C                           subwf     ACCcLO
015F 6019                           movfp     ACCaHI,wreg
0160 031D                           subwfb    ACCcHI
0161 8004                           bsf       _carry            ;shift a 1 into d (result)
0162 1B1E                 nosub1    rlcf      ACCdLO
0163 1B1F                           rlcf      ACCdHI
0002                      i = i+1
                          ;
0164 8804                           bcf       _carry
0165 1B1A                           rlcf      ACCbLO
0166 1B1B                           rlcf      ACCbHI
0167 1B1C                           rlcf      ACCcLO
0168 1B1D                           rlcf      ACCcHI
0169 6019                           movfp     ACCaHI,wreg
016A 041D                           subwf     ACCcHI,w          ;check if a>c
016B 9204                           btfss     _z
016C C16F                           goto      notz2
016D 6018                           movfp     ACCaLO,wreg
016E 041C                           subwf     ACCcLO,w          ;if msb equal then check lsb
016F 9004                 notz2     btfss     _carry            ;carry set if c>a
0170 C176                           goto      nosub2            ; if c < a
0171 6018                 subca2    movfp     ACCaLO,wreg       ;c-a into c
0172 051C                           subwf     ACCcLO
0173 6019                           movfp     ACCaHI,wreg
0174 031D                           subwfb    ACCcHI
0175 8004                           bsf       _carry             ;shift a 1 into d (result)
0176 1B1E                 nosub2    rlcf      ACCdLO
0177 1B1F                           rlcf      ACCdHI
0003                      i = i+1
```

```
                        ;
0178 8804                           bcf     _carry
0179 1B1A                           rlcf    ACCbLO
017A 1B1B                           rlcf    ACCbHI
017B 1B1C                           rlcf    ACCcLO
017C 1B1D                           rlcf    ACCcHI
017D 6019                           movfp   ACCaHI,wreg
017E 041D                           subwf   ACCcHI,w          ;check if a>c
017F 9204                           btfss   _z
0180 C183                           goto    notz3
0181 6018                           movfp   ACCaLO,wreg
0182 041C                           subwf   ACCcLO,w          ;if msb equal then check lsb
0183 9004                   notz3   btfss    _carry           ;carry set if c>a
0184 C18A                           goto    nosub3           ; if c < a
0185 6018                   subca3  movfp   ACCaLO,wreg      ;c-a into c
0186 051C                           subwf   ACCcLO
0187 6019                           movfp   ACCaHI,wreg
0188 031D                           subwfb  ACCcHI
0189 8004                           bsf     _carry           ;shift a 1 into d (result)
018A 1B1E                   nosub3  rlcf    ACCdLO
018B 1B1F                           rlcf    ACCdHI
0004                        i = i+1
                            ;
018C 8804                           bcf     _carry
018D 1B1A                           rlcf    ACCbLO
018E 1B1B                           rlcf    ACCbHI
018F 1B1C                           rlcf    ACCcLO
0190 1B1D                           rlcf    ACCcHI
0191 6019                           movfp   ACCaHI,wreg
0192 041D                           subwf   ACCcHI,w          ;check if a>c
0193 9204                           btfss   _z
0194 C197                           goto    notz4
0195 6018                           movfp   ACCaLO,wreg
0196 041C                           subwf   ACCcLO,w          ;if msb equal then check lsb
0197 9004                   notz4   btfss   _carry            ;carry set if c>a
0198 C19E                           goto    nosub4           ; if c < a
0199 6018                   subca4  movfp   ACCaLO,wreg       ;c-a into c
019A 051C                           subwf   ACCcLO
019B 6019                           movfp   ACCaHI,wreg
019C 031D                           subwfb  ACCcHI
019D 8004                           bsf     _carry           ;shift a 1 into d (result)
019E 1B1E                   nosub4  rlcf    ACCdLO
019F 1B1F                           rlcf    ACCdHI
0005                        i = i+1
                            ;
01A0 8804                           bcf     _carry
01A1 1B1A                           rlcf    ACCbLO
01A2 1B1B                           rlcf    ACCbHI
01A3 1B1C                           rlcf    ACCcLO
01A4 1B1D                           rlcf    ACCcHI
01A5 6019                           movfp   ACCaHI,wreg
01A6 041D                           subwf   ACCcHI,w          ;check if a>c
01A7 9204                           btfss     _z
01A8 C1AB                           goto    notz5
01A9 6018                           movfp   ACCaLO,wreg
01AA 041C                           subwf   ACCcLO,w          ;if msb equal then check lsb
01AB 9004                   notz5   btfss   _carry            ;carry set if c>a
01AC C1B2                           goto    nosub5           ; if c < a
01AD 6018                   subca5  movfp   ACCaLO,wreg       ;c-a into c
01AE 051C                           subwf   ACCcLO
01AF 6019                           movfp   ACCaHI,wreg
01B0 031D                           subwfb  ACCcHI
01B1 8004                           bsf     _carry          ;shift a 1 into d (result)
01B2 1B1E                   nosub5  rlcf    ACCdLO
01B3 1B1F                           rlcf    ACCdHI
0006                        i = i+1

                            ;
01B4 8804                           bcf     _carry
```

```
01B5 1B1A                          rlcf     ACCbLO
01B6 1B1B                          rlcf     ACCbHI
01B7 1B1C                          rlcf     ACCcLO
01B8 1B1D                          rlcf     ACCcHI
01B9 6019                          movfp    ACCaHI,wreg
01BA 041D                          subwf    ACCcHI,w          ;check if a>c
01BB 9204                          btfss    _z
01BC C1BF                          goto     notz6
01BD 6018                          movfp    ACCaLO,wreg
01BE 041C                          subwf    ACCcLO,w          ;if msb equal then check lsb
01BF 9004            notz6         btfss    _carry            ;carry set if c>a
01C0 C1C6                          goto     nosub6             ; if c < a
01C1 6018            subca6        movfp    ACCaLO,wreg           ;c-a into c
01C2 051C                          subwf    ACCcLO
01C3 6019                          movfp    ACCaHI,wreg
01C4 031D                          subwfb   ACCcHI
01C5 8004                          bsf      _carry            ;shift a 1 into d (result)
01C6 1B1E            nosub6        rlcf     ACCdLO
01C7 1B1F                          rlcf     ACCdHI
0007                 i = i+1
                     ;
01C8 8804                          bcf      _carry
01C9 1B1A                          rlcf     ACCbLO
01CA 1B1B                          rlcf     ACCbHI
01CB 1B1C                          rlcf     ACCcLO
01CC 1B1D                          rlcf     ACCcHI
01CD 6019                          movfp    ACCaHI,wreg
01CE 041D                          subwf    ACCcHI,w          ;check if a>c
01CF 9204                          btfss    _z
01D0 C1D3                          goto     notz7
01D1 6018                          movfp    ACCaLO,wreg
01D2 041C                          subwf    ACCcLO,w          ;if msb equal then check lsb
01D3 9004            notz7         btfss    _carry            ;carry set if c>a
01D4 C1DA                          goto     nosub7             ; if c < a
01D5 6018            subca7        movfp    ACCaLO,wreg           ;c-a into c
01D6 051C                          subwf    ACCcLO
01D7 6019                          movfp    ACCaHI,wreg
01D8 031D                          subwfb   ACCcHI
01D9 8004                          bsf      _carry            ;shift a 1 into d (result)
01DA 1B1E            nosub7        rlcf     ACCdLO
01DB 1B1F                          rlcf     ACCdHI
0008                 i = i+1

                          ;
01DC 8804                          bcf      _carry
01DD 1B1A                          rlcf     ACCbLO
01DE 1B1B                          rlcf     ACCbHI
01DF 1B1C                          rlcf     ACCcLO
01E0 1B1D                          rlcf     ACCcHI
01E1 6019                          movfp    ACCaHI,wreg
01E2 041D                          subwf    ACCcHI,w          ;check if a>c
01E3 9204                          btfss      _z
01E4 C1E7                          goto     notz8
01E5 6018                          movfp    ACCaLO,wreg
01E6 041C                          subwf    ACCcLO,w          ;if msb equal then check lsb
01E7 9004            notz8         btfss    _carry            ;carry set if c>a
01E8 C1EE                          goto     nosub8            ; if c < a
01E9 6018            subca8        movfp    ACCaLO,wreg       ;c-a into c
01EA 051C                          subwf    ACCcLO
01EB 6019                          movfp    ACCaHI,wreg
01EC 031D                          subwfb   ACCcHI
01ED 8004                          bsf      _carry            ;shift a 1 into d (result)
01EE 1B1E            nosub8        rlcf     ACCdLO
01EF 1B1F                          rlcf     ACCdHI
0009                 i = i+1

                          ;
01F0 8804                          bcf      _carry
01F1 1B1A                          rlcf     ACCbLO
```

```
01F2 1B1B                       rlcf    ACCbHI
01F3 1B1C                       rlcf    ACCcLO
01F4 1B1D                       rlcf    ACCcHI
01F5 6019                       movfp   ACCaHI,wreg
01F6 041D                       subwf   ACCcHI,w        ;check if a>c
01F7 9204                       btfss   _z
01F8 C1FB                       goto    notz9
01F9 6018                       movfp   ACCaLO,wreg
01FA 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
01FB 9004           notz9       btfss   _carry          ;carry set if c>a
01FC C202                       goto    nosub9          ; if c < a
01FD 6018           subca9      movfp   ACCaLO,wreg     ;c-a into c
01FE 051C                       subwf   ACCcLO
01FF 6019                       movfp   ACCaHI,wreg
0200 031D                       subwfb  ACCcHI
0201 8004                       bsf     _carry          ;shift a 1 into d (result)
0202 1B1E           nosub9      rlcf    ACCdLO
0203 1B1F                       rlcf    ACCdHI
000A                i = i+1
                    ;
0204 8804                       bcf     _carry
0205 1B1A                       rlcf    ACCbLO
0206 1B1B                       rlcf    ACCbHI
0207 1B1C                       rlcf    ACCcLO
0208 1B1D                       rlcf    ACCcHI
0209 6019                       movfp   ACCaHI,wreg
020A 041D                       subwf   ACCcHI,w        ;check if a>c
020B 9204                       btfss   _z
020C C20F                       goto    notz10
020D 6018                       movfp   ACCaLO,wreg
020E 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
020F 9004           notz10      btfss   _carry          ;carry set if c>a
0210 C216                       goto    nosub10         ; if c < a
0211 6018           subca10     movfp   ACCaLO,wreg        ;c-a into c
0212 051C                       subwf   ACCcLO
0213 6019                       movfp   ACCaHI,wreg
0214 031D                       subwfb  ACCcHI
0215 8004                       bsf     _carry          ;shift a 1 into d (result)
0216 1B1E           nosub10     rlcf    ACCdLO
0217 1B1F                       rlcf    ACCdHI
000B                i = i+1
                    ;
0218 8804                       bcf     _carry
0219 1B1A                       rlcf    ACCbLO
021A 1B1B                       rlcf    ACCbHI
021B 1B1C                       rlcf    ACCcLO
021C 1B1D                       rlcf    ACCcHI
021D 6019                       movfp   ACCaHI,wreg
021E 041D                       subwf   ACCcHI,w        ;check if a>c
021F 9204                       btfss   _z
0220 C223                       goto    notz11
0221 6018                       movfp   ACCaLO,wreg
0222 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
0223 9004           notz11      btfss   _carry          ;carry set if c>a
0224 C22A                       goto    nosub11         ; if c < a
0225 6018           subca11     movfp   ACCaLO,wreg     ;c-a into c
0226 051C                       subwf   ACCcLO
0227 6019                       movfp   ACCaHI,wreg
0228 031D                       subwfb  ACCcHI
0229 8004                       bsf     _carry          ;shift a 1 into d (result)
022A 1B1E           nosub11     rlcf    ACCdLO
022B 1B1F                       rlcf    ACCdHI
000C                i = i+1
                    ;
022C 8804                       bcf     _carry
022D 1B1A                       rlcf    ACCbLO
022E 1B1B                       rlcf    ACCbHI
022F 1B1C                       rlcf    ACCcLO
0230 1B1D                       rlcf    ACCcHI
```

```
0231 6019                          movfp    ACCaHI,wreg
0232 041D                          subwf    ACCcHI,w         ;check if a>c
0233 9204                          btfss    _z
0234 C237                          goto     notz12
0235 6018                          movfp    ACCaLO,wreg
0236 041C                          subwf    ACCcLO,w         ;if msb equal then check lsb
0237 9004          notz12          btfss    _carry           ;carry set if c>a
0238 C23E                          goto     nosub12          ; if c < a
0239 6018          subca12         movfp    ACCaLO,wreg      ;c-a into c
023A 051C                          subwf    ACCcLO
023B 6019                          movfp    ACCaHI,wreg
023C 031D                          subwfb   ACCcHI
023D 8004                          bsf      _carry           ;shift a 1 into d (result)
023E 1B1E          nosub12         rlcf     ACCdLO
023F 1B1F                          rlcf     ACCdHI
000D               i = i+1

                        ;
0240 8804                          bcf      _carry
0241 1B1A                          rlcf     ACCbLO
0242 1B1B                          rlcf     ACCbHI
0243 1B1C                          rlcf     ACCcLO
0244 1B1D                          rlcf     ACCcHI
0245 6019                          movfp    ACCaHI,wreg
0246 041D                          subwf    ACCcHI,w         ;check if a>c
0247 9204                          btfss    _z
0248 C24B                          goto     notz13
0249 6018                          movfp    ACCaLO,wreg
024A 041C                          subwf    ACCcLO,w         ;if msb equal then check lsb
024B 9004          notz13          btfss    _carry           ;carry set if c>a
024C C252                          goto     nosub13          ; if c < a
024D 6018          subca13         movfp    ACCaLO,wreg      ;c-a into c
024E 051C                          subwf     ACCcLO
024F 6019                          movfp    ACCaHI,wreg
0250 031D                          subwfb    ACCcHI
0251 8004                          bsf      _carry           ;shift a 1 into d (result)
0252 1B1E          nosub13         rlcf     ACCdLO
0253 1B1F                          rlcf     ACCdHI
000E               i = i+1
                    ;
0254 8804                          bcf      _carry
0255 1B1A                          rlcf     ACCbLO
0256 1B1B                          rlcf     ACCbHI
0257 1B1C                          rlcf     ACCcLO
0258 1B1D                          rlcf     ACCcHI
0259 6019                          movfp    ACCaHI,wreg
025A 041D                          subwf    ACCcHI,w         ;check if a>c
025B 9204                          btfss    _z
025C C25F                          goto     notz14
025D 6018                          movfp    ACCaLO,wreg
025E 041C                          subwf    ACCcLO,w         ;if msb equal then check lsb
025F 9004          notz14          btfss    _carry           ;carry set if c>a
0260 C266                          goto     nosub14           ; if c < a
0261 6018          subca14         movfp    ACCaLO,wreg        ;c-a into c
0262 051C                          subwf    ACCcLO
0263 6019                          movfp    ACCaHI,wreg
0264 031D                          subwfb   ACCcHI
0265 8004                          bsf      _carry           ;shift a 1 into d (result)
0266 1B1E          nosub14         rlcf     ACCdLO
0267 1B1F                          rlcf     ACCdHI
000F               i = i+1
                        ;
0268 8804                          bcf      _carry
0269 1B1A                          rlcf     ACCbLO
026A 1B1B                          rlcf     ACCbHI
026B 1B1C                          rlcf     ACCcLO
026C 1B1D                          rlcf     ACCcHI
026D 6019                          movfp    ACCaHI,wreg
026E 041D                          subwf    ACCcHI,w         ;check if a>c
```

```
026F 9204                           btfss      _z
0270 C273                           goto     notz15
0271 6018                           movfp    ACCaLO,wreg
0272 041C                           subwf    ACCcLO,w          ;if msb equal then check lsb
0273 9004           notz15          btfss    _carry            ;carry set if c>a
0274 C27A                           goto     nosub15           ; if c < a
0275 6018           subca15         movfp    ACCaLO,wreg       ;c-a into c
0276 051C                           subwf    ACCcLO
0277 6019                           movfp    ACCaHI,wreg
0278 031D                           subwfb   ACCcHI
0279 8004                           bsf      _carry            ;shift a 1 into d (result)
027A 1B1E           nosub15         rlcf     ACCdLO
027B 1B1F                           rlcf     ACCdHI
0010                i = i+1
                     ;
                     ;
027C 027C                 if SIGNED
                                    btfss    sign,MSB          ; negate (ACCc,ACCd)
                                    return
                                    negMac   ACCcLO,ACCcHI
                                    negMac   ACCdLO,ACCdHI
                                    return
                                    else
027C 0002                           return
                                              endif
                          ;
                          ;*****************************************************************
                          ;
                          ;            Square Root By Newton Raphson Method
                          ;
                          ;    This routine computes the square root of a 16 bit number(with
                          ;  low byte in NumLo & high byte in NumHi ). After loading NumLo &
                          ;  NumHi with the desired number whose square root is to be computed,
                          ;  branch to location Sqrt ( by “GOTO  Sqrt” ). “ CALL  Sqrt” cannot
                          ;  be issued because the Sqrt function makes calls to Math routines
                          ;  and the stack is completely used up.
                          ;       The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
                          ;  The total number of iterations is set to ten. If more iterations
                          ;  are desired, change “LupCnt equ .10” to the desired value. Also,
                          ;  the initial guess value of the square root is given set as
                          ;  input/2 ( in subroutine “init” ). The user may modify this scheme
                          ;  if a better initial approximation value is known. A good initial
                          ;  guess will help the algorithm converge at a faster rate and thus
                          ;  less number of iterations required.
                          ;       Two utility math routines are used by this program : D_divS
                          ;  and D_add. These two routines are listed as seperate routines
                          ;  under double precision Division and double precision addition
                          ;  respectively.
                          ;
                          ;  Note : If square root of an 8 bit number is desired, it is probably
                          ;  better to have a table look scheme rather than using numerical
                          ;  methods.
                          ;         This method is computationally quite intensive and
                          ;  slow, but very accurate and the convergence rate is high
                          ;
                          ;   Performance :
                          ;             Program Memory : 22 (excluding D_divS subroutine)
                          ;              Clock Cycles : 3000 (approximately,with 10 iterations)
                          ;
                          ;       The #of cycles depends on Number of Iterations Selected.
                          ; In a lot of cases 5 or less iterations may be sufficient
                          ;
                          ;
                          ;*****************************************************************
                          ;                  Newton-Raphson Method
                          ;*************************************************************
                          Sqrt
027D E28B                    call    SqrtInit            ; compute initial sqrt = Num/2
```

```
                        nextIter
                           if MODE_FAST
027E 7A24                  movfp    NumLo,ACCbLO
027F 7B25                  movfp    NumHi,ACCbHI
                           else
                           movfp    NumLo,wreg
                           movwf    ACCbLO
                           movfp    NumHi,wreg
                           movwf    ACCbHI
                           endif
                        ;
0280 E119                  call    D_divS              ; double precision division
                                                                ; double precision addition
0281 601E                  movfp    ACCdLO,wreg        ; ACCd + ACCa -> ACCd
0282 0F18                  addwf    ACCaLO             ;addwf lsb
0283 601F                  movfp    ACCdHI,wreg
0284 1119                  addwfc   ACCaHI             ;addwf msb
                                                       ; now divide by 2
0285 8804                   bcf      _carry
0286 1919                   rrcf     ACCaHI
0287 1918                   rrcf     ACCaLO
                        ;
0288 1726                  decfsz   iterCnt
0289 C27E                  goto     nextIter
028A 0002                  return                      ; End Sqrt
                        ;
                        SqrtInit
028B B00A                   movlw    _LUPCNT
028C 0126                   movwf     iterCnt           ; set number of iterations
                         if MODE_FAST
028D 7925                  movfp    NumHi,ACCaHI
028E 7824                  movfp    NumLo,ACCaLO
                           else
                           movfp    NumHi,wreg
                           movwf    ACCaHI
                           movfp    NumLo,wreg          ; set initial guess root = NUM/2
                           movwf    ACCaLO
                           endif
028F 8804                  bcf      _carry
0290 1919                  rrcf     ACCaHI
0291 1918                  rrcf     ACCaLO             ; set initial sqrt = Num/2
0292 0002                  return
                        ;
                        ;*****************************************************************
                        ;                    8x8 Software Multiplier
                        ;                ( Fast Version : Straight Line Code )
                        ;
                        ;   The 16 bit result is stored in 2 bytes
                        ;
                        ; Before calling the subroutine " mpy ", the multiplier should
                        ; be loaded in location " mulplr ", and the multiplicand in
                        ; " mulcnd " . The 16 bit result is stored in locations
                        ; H_byte & L_byte.
                        ;
                        ;       Performance :
                        ;           Program Memory  :  36 words
                        ;           # of cycles     :  36       (excluding call & return)
                        ;           Scratch RAM     :   0 locations
                        ;           W Register      :  Used
                        ;
                        ;  This routine is optimized for speed efficiency ( straight line code
)
                        ;  For code efficiency, refer to "mult8x8S.asm" ( looped code )
                        ;*****************************************************************
                        ;     Define a macro for adding & right shifting
                        ;
                        multiply    MACRO
                           variable  i                      ;
                             i = 0
```

```
                              .while i < 8
                                    btfsc   mulplr,i
                                    addwf   H_byte
                                    rrcf    H_byte
                                    rrcf    L_byte
                                 i = i+1 ;
                                 .endw
                           ENDM                              ; End of macro
                        ;
                        ;
                        mpy8x8_F
0293 291B                           clrf    H_byte
0294 291A                           clrf    L_byte
0295 6018                           movfp   mulcnd,wreg       ; move the multiplicand to W reg.
0296 8804                           bcf     _carry            ; Clear the carry bit in the status Reg.
                        ;
                           multiply
0000                       variable  i                        ;
0000                         i = 0
                             .while i < 8
                                    btfsc   mulplr,i
                                    addwf   H_byte
                                    rrcf    H_byte
                                    rrcf    L_byte
                              i = i+1                          ;
                             .endw
0297 9819                           btfsc   mulplr,i
0298 0F1B                           addwf   H_byte
0299 191B                           rrcf    H_byte
029A 191A                           rrcf    L_byte
0001                           i = i+1              ;
029B 9919                           btfsc   mulplr,i
029C 0F1B                           addwf   H_byte
029D 191B                           rrcf    H_byte
029E 191A                           rrcf    L_byte
0002                           i = i+1              ;
029F 9A19                           btfsc   mulplr,i
02A0 0F1B                           addwf   H_byte
02A1 191B                           rrcf    H_byte
02A2 191A                           rrcf    L_byte
0003                           i = i+1              ;
02A3 9B19                           btfsc   mulplr,i
02A4 0F1B                           addwf   H_byte
02A5 191B                           rrcf    H_byte
02A6 191A                           rrcf    L_byte
0004                           i = i+1              ;
02A7 9C19                           btfsc   mulplr,i
02A8 0F1B                           addwf   H_byte
02A9 191B                           rrcf    H_byte
02AA 191A                           rrcf    L_byte
0005                           i = i+1              ;
02AB 9D19                           btfsc   mulplr,i
02AC 0F1B                           addwf   H_byte
02AD 191B                           rrcf    H_byte
02AE 191A                           rrcf    L_byte
0006                           i = i+1              ;
02AF 9E19                           btfsc   mulplr,i
02B0 0F1B                           addwf   H_byte
02B1 191B                           rrcf    H_byte
02B2 191A                           rrcf    L_byte
0007                           i = i+1              ;

02B3 9F19                   btfsc    mulplr,i
02B4 0F1B                   addwf    H_byte
02B5 191B                   rrcf     H_byte
02B6 191A                   rrcf     L_byte
0008                        i = i+1              ;
                        ;
02B7 02B7 0002          return
```

```
                        ;
                        ;*****************************************************************
                        ;                   8x8 Software Multiplier
                        ;               ( Code Efficient : Looped Code )
                        ;
                        ;   The 16 bit result is stored in 2 bytes
                        ;
                        ; Before calling the subroutine " mpy ", the multiplier should
                        ; be loaded in location " mulplr ", and the multiplicand in
                        ; " mulcnd " . The 16 bit result is stored in locations
                        ; H_byte & L_byte.
                        ;
                        ;       Performance :
                        ;           Program Memory  :  13 words (excluding call & return)
                        ;           # of cycles     :  69       (excluding call & return)
                        ;           Scratch RAM     :  1 byte
                        ;           W Register      :  Used
                        ;
                        ;  This routine is optimized for code efficiency ( looped code )
                        ;  For time efficiency code refer to "mult8x8F.asm" (straight line code)
                        ;*****************************************************************
                        ;
                        mpy8x8_S
02B8 291B                   clrf    H_byte
02B9 291A                   clrf    L_byte
02BA 2922                   clrf    count
02BB 8322                   bsf     count,3         ; set count = 8
02BC 6018                   movfp   mulcnd,wreg
02BD 8804                   bcf     _carry          ; Clear the carry bit in the status Reg.
                        loop
02BE 9819                   btfsc   mulplr,0
02BF 0F1B                   addwf   H_byte
02C0 191B                   rrcf    H_byte
02C1 191A                   rrcf    L_byte
02C2 2119                   rrncf   mulplr
02C3 1722                   decfsz  count
02C4 C2BE                   goto    loop
                        ;
02C5 0002                   return
                        ;
                        ;*****************************************************************
                        ;                       Numerical Differenciation
                        ;
                        ;       The so called "Three-Point Formula" is implemented to
                        ; differenciate a sequence of points (uniformly sampled).
                        ;       The eqn implemented is :
                        ;            f'(Xn) = [ f(Xn - 2h) - 4*f(Xn - h) + 3*f(Xn)]*0.5/h
                        ;    where Xn is the present sample and 'h' is the step size.
                        ;
                        ;       The above formula may be rewritten as :
                        ;
                        ;         f'(Xn) = [ 0.5*f(Xn -2) - 2*f(Xn - 1) + 0.5*3*f(Xn)]*1/DiffK
                        ;  where DiffK = h = Step Size
                        ;
                        ;   This differenciation routine can be used very effectively
                        ; in the computation of the differential component part in
                        ; a PID Loop calculation in Motor Control Applications
                        ;
                        ;       Double precision arithmetic is used throught
                        ;  The present sample value is assumed to be in locations
                        ; (XnHi, XnLo). The past two values are assumed to be in locations
                        ; (Xn_1_Hi, Xn_1_Lo) & (Xn_2_Hi, Xn_2_Lo).
                        ;       The output value is located in DiffHi & DiffLo. No overflow
                        ; checking mechanism is implemented. If the values are limited
                        ; to 12 bits, then the user need not worry about overflows
                        ;
                        ;  It is user's responsibility to update the past values with the
                        ; present values before calling this routine.
                        ;  After computation, the present value Xn is not moved to Xn_1
```

```
                        ; because the user may want these values to be intact for other
                        ; computations ( say numerical integration)
                        ;       Also it is user's responsibility to set past 2 values
                        ; (Xn_1 & Xn_2) values to be zero on initialization.
                        ;
                        ;*****************************************************************
                        ;
                        Diff
02C6 602B                   movfp     Xn_2_Lo,wreg
02C7 0E27                   addwf     XnLo,w
02C8 011A                   movwf     ACCbLO
02C9 602C                   movfp     Xn_2_Hi,wreg
02CA 1028                   addwfc    XnHi,w
02CB 011B                   movwf     ACCbHI          ; Y = f(Xn-2) + f(Xn)
                        ;
02CC 6027                   movfp     XnLo,wreg
02CD 0F1A                   addwf     ACCbLO
02CE 6028                   movfp     XnHi,wreg
02CF 111B                   addwfc    ACCbHI
02D0 6027                   movfp     XnLo,wreg
02D1 0F1A                   addwf     ACCbLO
02D2 6028                   movfp     XnHi,wreg
02D3 111B                   addwfc    ACCbHI          ; Y = f(Xn-2) + 3*f(Xn)
                        ;
02D4 8804                   bcf       _carry
02D5 191B                   rrcf      ACCbHI
02D6 191A                   rrcf      ACCbLO           ; Y = 0.5*[ f(Xn-2) + 3*f(Xn) ]

                        ;
02D7 6029                   movfp     Xn_1_Lo,wreg
02D8 051A                   subwf     ACCbLO
02D9 602A                   movfp     Xn_1_Hi,wreg
02DA 031B                   subwfb    ACCbHI
02DB 6029                   movfp     Xn_1_Lo,wreg
02DC 051A                   subwf     ACCbLO
02DD 602A                   movfp     Xn_1_Hi,wreg
02DE 031B                   subwfb    ACCbHI          ; Y = 0.5*[f(Xn-2) + 3*f(Xn)] - 2*f(Xn-1)
                        ;
02DF 602D                   movfp     DiffKLo,wreg
02E0 0118                   movwf     ACCaLO
02E1 602E                   movfp     DiffKHi,wreg
02E2 0119                   movwf     ACCaHI
                        ;
02E3 E119                   call      D_divS
02E4 601A                   movfp     ACCbLO,wreg
02E5 012F                   movwf     DiffLo
02E6 601B                   movfp     ACCbHI,wreg
02E7 0130                   movwf     DiffHi          ; result = Y/h
                        ;
02E8 0002                   return
                        ;
                        ;*****************************************************************
                        ;
                        ;                       Numerical Integration
                        ;
                        ;
                        ;       Simpson's Three-Eighths Rule is implemented
                        ;
                        ;       Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*3*h/8
                        ;
                        ;  where 'h' is the step size and the integral is over the
                        ;  range X0 to X3
                        ;       The above equation can be rewritten as
                        ;
                        ;       Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*IntgK
                        ;
                        ;   where IntgK = 3*h/8 (in locations (IntgKHi, IntgKHi)
                        ;
                        ;   This Integration routine can be used very effectively
```

```
                        ; in the computation of the integral component part in
                        ; a PID Loop calculation in Motor Control Applications
                        ;
                        ;       Double precision arithmetic is used throught
                        ;  The three input values over which the integral is to be computed
                        ; are assumed to be in locations (X0Lo,X0Hi), (X1Lo,X1Hi) , (X2Lo,X2Hi)
                        ; and (X3Lo,X3Hi)
                        ;       The output value is located in IntgHi & IntgLo. No overflow
                        ; checking mechanism is implemented. If the values are limited
                        ; to 12 bits, then the user need not worry about overflows
                        ;
                        ;  It is user's responsibility to update the past values with the
                        ; present values before calling this routine.
                        ;  After computation, the present value Xn is not moved to Xn_1
                        ; because the user may want these values to be intact for other
                        ; computations ( say numerical integration)
                        ;       Also it is user's responsibility to set past 2 values
                        ; (Xn_1 & Xn_2) values to be zero on initialization.
                        ;
                        ;
                        ;*****************************************************************
                        ;
                        Integrate
02E9 6031                   movfp     X0Lo,wreg
02EA 0E37                   addwf     X3Lo,w
02EB 011A                   movwf     ACCbLO
02EC 6032                   movfp     X0Hi,wreg
02ED 1038                   addwfc    X3Hi,w
02EE 011B                   movwf     ACCbHI          ; Intg = f(X0) + f(X3)
                        ;
02EF 6033                   movfp     X1Lo,wreg
02F0 0F1A                   addwf     ACCbLO
02F1 6034                   movfp     X1Hi,wreg
02F2 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +X1
02F3 6033                   movfp     X1Lo,wreg
02F4 0F1A                   addwf     ACCbLO
02F5 6034                   movfp     X1Hi,wreg
02F6 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +2*X1
02F7 6033                   movfp     X1Lo,wreg
02F8 0F1A                   addwf     ACCbLO
02F9 6034                   movfp     X1Hi,wreg
02FA 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1
                        ;
02FB 6035                   movfp     X2Lo,wreg
02FC 0F1A                   addwf     ACCbLO
02FD 6036                   movfp     X2Hi,wreg
02FE 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1 + X2
02FF 6035                   movfp     X2Lo,wreg
0300 0F1A                   addwf     ACCbLO
0301 6036                   movfp     X2Hi,wreg
0302 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1 + 2*X2
0303 6035                   movfp     X2Lo,wreg
0304 0F1A                   addwf     ACCbLO
0305 6036                   movfp     X2Hi,wreg
0306 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1 + 3*X2
                        ;
0307 6039                   movfp     IntgKLo,wreg
0308 0118                   movwf     ACCaLO
0309 603A                   movfp     IntgKHi,wreg
030A 0119                   movwf     ACCaHI          ;  ACCa=IntgK (prepare for multiplication)
                        ;
030B E050                    call     D_mpyS           ; make sure to set for either SIGNED or UNSIGNED
030C 601E                   movfp    ACCdLO,wreg
030D 013B                   movwf    IntgLo           ; 32 bit result in ACCd & ACCc
030E 601F                   movfp    ACCdHI,wreg
030F 013C                   movwf    IntgHi           ; upper 16 bits = result
                        ;
0310 0002                   return
                        ;
```

```
                        ;*****************************************************************
                        ;
                        ;                       Random Number Generator
                        ;
                        ; This routine generates a 16 Bit Pseudo Sequence Random Generator
                        ; It is based on Linear shift register feedback. The sequence
                        ; is generated by (Q15 xorwf Q14 xorwf Q12 xorwf Q3 )
                        ;
                        ;    The 16 bit random number is in location RandHi(high byte)
                        ;  & RandLo (low byte)
                        ;
                        ;       Before calling this routine, make sure the initial values
                        ; of RandHi & RandLo are NOT ZERO
                        ;       A good chiose of initial random number is 0x3045
                        ;*****************************************************************
                        ;
                        Random16
0311 1A19                  rlcf     RandHi,w
0312 0C19                  xorwf     RandHi,w
0313 1B00                  rlcf     wreg             ; carry bit = xorwf(Q15,14)
                        ;
0314 1D19                  swapf    RandHi
0315 1C18                  swapf    RandLo,w
0316 2300                  rlncf      wreg
0317 0C19                  xorwf     RandHi,w        ; LSB = xorwf(Q12,Q3)
0318 1D19                  swapf    RandHi
0319 B501                  andlw   0x01
031A 1B18                  rlcf     RandLo
031B 0D18                  xorwf     RandLo
031C 1B19                  rlcf     RandHi
031D 0002                  return
                        ;
                        ;*****************************************************************
                        ;                       Gaussian Noise Generator
                        ;
                        ; This routine generates a 16 Bit Gaussian distributed random
                        ; points. This routine calls the routine “Random16”, which
                        ; generates a psuedo random noise sequence. Gaussian noise
                        ; is computed using the CENTRAL LIMIT THEOREM.
                        ;       The Central Limit Theorem states that the average weighted
                        ; sum of uncorelated samples tends to have a Gaussian distribution
                        ; For practical purposes, the sum could be over a sample size
                        ; of 32 Random numbers. Better results could result if a larger
                        ; sample size is desired. For faster results, a sum over 16 samples
                        ; would also be adequate ( say, for applications like Speech synthesis,
                        ; channel simulations, etc).
                        ;
                        ;       The 16 bit Gaussian distributed point is in locations
                        ; GaussHi & GaussLo
                        ;
                        ;       Before calling this routine, the initial seed of Random
                        ; number should be NON ZERO ( refer to notes on “Random16” routine
                        ;
                        ;*****************************************************************
                        ;
                        Gauss
031E 2922                  clrf     count
031F 8522                  bsf      count,5         ; set Sample size = 32
0320 291A                  clrf     GaussLo
0321 291B                  clrf     GaussHi
0322 2920                  clrf     GaussTmp
                        ;
                        NextGauss
0323 E311                  call     Random16        ; get a random value
0324 6018                  movfp    RandLo,wreg
0325 0F1A                  addwf    GaussLo
0326 6019                  movfp    RandHi,wreg
0327 111B                  addwfc   GaussHi
0328 2900                  clrf     wreg
```

```
0329 1120                   addwfc  GaussTmp
032A 1722                   decfsz  count
032B C323                   goto    NextGauss        ; sum 16 random numbers
                          ;
032C B005                   movlw    5
                          GaussDiv16
032D 1920                   rrcf     GaussTmp
032E 191B                   rrcf     GaussHi
032F 191A                   rrcf     GaussLo         ; weghted average
0330 1700                   decfsz   wreg          ; divide by 32
0331 C32D                   goto     GaussDiv16
                          ;
0332 0002                   return
                          ;

                                END                      ; End Of arith.asm


Errors   :    0
Warnings :    0
```

APPENDIX B : BCD ARITHMETIC ROUTINES LISTING FILE OF BCD.ASM

```
MPASM B0.54                                                          PAGE  1

0001                #define PAGE    EJECT
                    ;TITLE   “BCD Arithmetic Routines : Ver 1.0”

                    ;*****************************************************************
                    ;                       BCD Arithmetic Routines
                    ;*****************************************************************

                    LIST     P=17C42, C=80, L=0, R=DEC
                    include “17c42.h”
                           CBLOCK   0x20
0020 0002                  Lbyte, Hbyte
0022 0003                  R2, R1, R0                ;must maintain R2, R1, R0 sequence
0025 0001                  count
0026 0002                  Num1, Num2
                           ENDC
                    ;
0026                BCD     equ       Num1
0026                Htemp   equ       Num1
0027                Ltemp   equ       Num2
                    ;
                         ORG          0x0000
                    ;*******************************************************

                    ;                       BCD Arithmetic Test Program
                    ;*******************************************************
                    ;
                    main
0000 2B21                  setf      Hbyte
0001 2B20                  setf      Lbyte
                    ;                                    ; 16 bit binary num = 0xfff
0002 E01F                  call      B2_BCD_Looped       ; after conversion the Deci

                    ;                                    ; in R0, R1, R2 = 06,55,35
0003 2B21                  setf      Hbyte
0004 2B20                  setf      Lbyte
0005 E03F                  call      B2_BCD_Straight  ; same as above, but straig
                    ;
0006 B006                  movlw     0x06
0007 0124                  movwf     R0
0008 B055                  movlw     0x55
0009 0123                  movwf     R1
000A B035                  movlw     0x35
000B 0122                  movwf     R2                ; setf R0R1R2 = 65535
                    ;
000C E082                  call      BCDtoB            ; after conversion Hbyte =
                    ;                                    ; and Lbyte = 0xff
000D B099                  movlw     0x99
000E 0126                  movwf     Num1
000F B099                  movlw     0x99
0010 0127                  movwf     Num2              ; setf Num1 = Num2 = 0x99
                    ;
0011 E093                  call      BCDAdd            ; after addition, Num2 = 98
                    ;                                    ; and Num1 = 01 ( 99+99 = 1
                    ;
0012 B063                  movlw    0x63              ; setf Wreg = 63 hex
0013 E015                  call      BinBCD             ; after conversion, BCD = 9
                    ;                                     ; 63 hex = 99 decimal.
                    ;
0014 C014           self    goto    self
                    ;
                    ;*******************************************************
                    ;                 Binary To BCD Conversion Routine (8 bit)
```

```
                    ;
                    ;       This routine converts the 8 bit binary number in th
                    ; to a 2 digit BCD number in location BCD( compacted BCD Co
                    ;       The least significant digit is returned in location
                    ; the most significant digit is returned in location MSD.
                    ;
                    ;   Performance :
                    ;               Program Memory  :  10
                    ;               Clock Cycles    :  62  (worst case when W =
                    ;                                      ( i.e max Decimal nu
                    ;*****************************************************
                    ;
                         BinBCD
0015 2926                   clrf     BCD
                         again
0016 B1F6                   addlw   -10
0017 9004                   btfss   _carry
0018 C01B                   goto    swapBCD
0019 1526                   incf    BCD
001A C016                   goto    again
                         swapBCD
001B B10A                   addlw   10
001C 1D26                   swapf   BCD
001D 0926                   iorwf   BCD
001E 0002                   return
                    ;
                    ;*****************************************************
                    ;                Binary To BCD Conversion Routine (16 Bit)
                    ;                       (LOOPED Version)
                    ;
                    ;      This routine converts a 16 Bit binary Number to a 5
                    ; BCD Number.
                    ;
                    ;       The 16 bit binary number is input in locations Hbyt
                    ; Lbyte with the high byte in Hbyte.
                    ;       The 5 digit BCD number is returned in R0, R1 and R2
                    ; containing the MSD in its right most nibble.
                    ;
                    ;   Performance :
                    ;               Program Memory  :  32
                    ;               Clock Cycles    :  750
                    ;
                    ;*****************************************************
                    ;
                         B2_BCD_Looped
001F 8404                   bsf      _fs0
0020 8504                   bsf      _fs1            ; set fsr0 for no auto inc
                    ;
0021 8804                   bcf      _carry
0022 2925                   clrf     count
0023 8425                   bsf      count,4         ; set count = 16
0024 2924                   clrf     R0
0025 2923                   clrf     R1
0026 2922                   clrf     R2
                         loop16a
0027 1B20                   rlcf     Lbyte
0028 1B21                   rlcf     Hbyte
0029 1B22                   rlcf     R2
002A 1B23                   rlcf     R1
002B 1B24                   rlcf     R0
                     ;
002C 2725                   dcfsnz   count
002D 0002                   return
                         adjDEC
002E B022                   movlw    R2             ; load R2 as indirect addr
002F 0101                   movwf     fsr0
0030 E036                   call    adjBCD
                    ;
0031 1501                   incf     fsr0
0032 E036                   call    adjBCD
```

```
                        ;
0033 1501                  incf    fsr0
0034 E036                  call    adjBCD
                        ;
0035 C027                  goto    loop16a
                        ;
                        adjBCD
0036 6000                  movfp    indf0,wreg
0037 B103                  addlw    0x03
0038 9B00                  btfsc    wreg,3         ; test if result > 7
0039 0100                  movwf    indf0
003A 6000                  movfp    indf0,wreg
003B B130                  addlw    0x30
003C 9F00                  btfsc    wreg,7         ; test if result > 7
003D 0100                  movwf    indf0          ; save as MSD
003E 0002                  return
                        ;
                        ;*********************************************************
                        ;               Binary To BCD Conversion Routine (16 Bit)
                        ;                      (Partial Straight Line Version)
                        ;
                        ;     This routine converts a 16 Bit binary Number to a 5
                        ; BCD Number.
                        ;
                        ;      The 16 bit binary number is input in locations Hbyt
                        ; Lbyte with the high byte in Hbyte.
                        ;      The 5 digit BCD number is returned in R0, R1 and R2
                        ; containing the MSD in its right most nibble.
                        ;
                        ;   Performance :
                        ;               Program Memory  :  44
                        ;               Clock Cycles    :  572
                        ;
                        ;*********************************************************
                        ;
                        B2_BCD_Straight
003F 8404                  bsf      _fs0
0040 8504                  bsf      _fs1            ; set fsr0 for no auto inc
                        ;
0041 8804                  bcf      _carry
0042 2925                  clrf     count
0043 8425                  bsf      count,4         ; set count = 16
0044 2924                  clrf     R0
0045 2923                  clrf     R1
0046 2922                  clrf     R2
                        loop16b
0047 1B20                  rlcf     Lbyte
0048 1B21                  rlcf     Hbyte
0049 1B22                  rlcf     R2
004A 1B23                  rlcf     R1
004B 1B24                  rlcf     R0
                        ;
004C 2725                  dcfsnz   count
004D 0002                  return                   ; DONE
004E B022                  movlw    R2              ; load R2 as indirect addr
004F 0101                  movwf    fsr0
                        ; adjustBCD
0050 6000                  movfp    indf0,wreg
0051 B103                  addlw    0x03
0052 9B00                  btfsc    wreg,3          ; test if result > 7
0053 0100                  movwf    indf0
0054 6000                  movfp    indf0,wreg
0055 B130                  addlw    0x30
0056 9F00                  btfsc    wreg,7          ; test if result > 7
0057 0100                  movwf    indf0           ; save as MSD
                        ;
0058 1501                  incf     fsr0
                        ; adjustBCD
0059 6000                  movfp    indf0,wreg
```

```
005A B103                   addlw    0x03
005B 9B00                   btfsc    wreg,3          ; test if result > 7
005C 0100                   movwf    indf0
005D 6000                   movfp    indf0,wreg
005E B130                   addlw    0x30
005F 9F00                   btfsc    wreg,7          ; test if result > 7
0060 0100                   movwf    indf0           ; save as MSD
                          ;
0061 1501                   incf     fsr0
                          ; adjustBCD
0062 6000                   movfp    indf0,wreg
0063 B103                   addlw    0x03
0064 9B00                   btfsc    wreg,3          ; test if result > 7
0065 0100                   movwf    indf0
0066 6000                   movfp    indf0,wreg
0067 B130                   addlw    0x30
0068 9F00                   btfsc    wreg,7          ; test if result > 7
0069 0100                   movwf    indf0           ; save as MSD
                          ;
006A C047                   goto     loop16b
                          ;
                          ;*******************************************************
                          ;               BCD To Binary Conversion
                          ;
                          ;       This routine converts a 5 digit BCD number to a 16
                          ; number.
                          ;       The input 5 digit BCD numbers are asumed to be in l
                          ; R0, R1 & R2 with R0 containing the MSD in its right most
                          ;
                          ;       The 16 bit binary number is output in registers Hby
                          ; ( high byte & low byte repectively ).
                          ;
                          ;       The method used for conversion is :
                          ;               input number X = abcde ( the 5 digit BCD nu
                          ;       X = (R0,R1,R2) = abcde = 10[10[10[10a+b]+c]+d]+e
                          ;
                          ;   Performance :
                          ;               Program Memory  :  30
                          ;               Clock Cycles    :  112
                          ;
                          ;*******************************************************

                          ;
                          mpy10b
006B B50F                   andlw    0x0f
006C 0F20                   addwf    Lbyte
006D 9804                   btfsc    _carry
006E 1521                   incf     Hbyte
                          mpy10a
006F 8804                   bcf      _carry          ; multiply by 2
0070 1A20                   rlcf     Lbyte,w
0071 0127                   movwf    Ltemp
0072 1A21                   rlcf     Hbyte,w         ; (Htemp,Ltemp) = 2*N
0073 0126                   movwf    Htemp
                          ;
0074 8804                   bcf      _carry          ; multiply by 2
0075 1B20                   rlcf     Lbyte
0076 1B21                   rlcf     Hbyte
0077 8804                   bcf      _carry          ; multiply by 2
0078 1B20                   rlcf     Lbyte
0079 1B21                   rlcf     Hbyte
007A 8804                   bcf      _carry          ; multiply by 2
007B 1B20                   rlcf     Lbyte
007C 1B21                   rlcf     Hbyte           ; (Hbyte,Lbyte) = 8*N
                          ;
007D 6027                   movfp    Ltemp,wreg
007E 0F20                   addwf    Lbyte
007F 6026                   movfp    Htemp,wreg
0080 1121                   addwfc   Hbyte
```

```
0081 0002                   return                  ; (Hbyte,Lbyte) = 10*N
                          ;
                          ;
                          BCDtoB
0082 2921                   clrf      Hbyte
0083 6024                   movfp     R0,wreg
0084 B50F                   andlw     0x0f
0085 0120                   movwf     Lbyte
0086 E06F                   call      mpy10a          ; result = 10a+b
                          ;
0087 1C23                   swapf     R1,w
0088 E06B                   call      mpy10b          ; result = 10[10a+b]
                          ;
0089 6023                   movfp     R1,wreg
008A E06B                   call      mpy10b          ; result = 10[10[10a+b]+c]

                          ;
008B 1C22                   swapf     R2,w
008C E06B                   call      mpy10b          ; result = 10[10[10[10a+b]+
                          ;
008D 6022                   movfp     R2,wreg
008E B50F                   andlw     0x0f
008F 0F20                   addwf     Lbyte
0090 9804                   btfsc     _carry
0091 1521                   incf      Hbyte           ; result = 10[10[10[10a+b]

0092 0002                   return                  ; BCD to binary conversion
                          ;
                          ;*******************************************************
                          ;
                          ;                 Unsigned BCD Addition
                          ;
                          ;       This routine performs a 2 Digit Unsigned BCD Additi
                          ; It is assumed that the two BCD numbers to be added are in
                          ; locations Num1 & Num2. The result is the sum of Num1+Num2
                          ; and is stored in location Num2 and the overflow carry is
                          ; in location Num1
                          ;
                          ;   Performance :
                          ;               Program Memory  :       5
                          ;               Clock Cycles    :       5
                          ;
                          ;*******************************************************
                          ;
                          BCDAdd
0093 6026                   movfp    Num1,wreg
0094 0E27                   addwf    Num2,w        ; perform binary addition

0095 2F27                   daw      Num2           ; adjust for BCD addition
0096 2926                   clrf     Num1
0097 1B26                   rlcf     Num1           ; set Num1 = carry bit
0098 0002                   return
                          ;
                          ;*******************************************************
                          ;
                                END



Errors   :    0
Warnings :    0
```

APPENDIX C: FXP_DIV.ASM

```
;       PIC17 FIXED POINT DIVIDE ROUTINES       VERSION 1.5
;       Input:  fixed point arguments in AARG and BARG
;       Output: quotient AARG/BARG followed by remainder in REM
;       All timings are worst case cycle counts
;       It is useful to note that the additional routines FXD3115U, FXD1515U, and
;       FXD1507U can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.
;         Routine            Clocks     Function
;       FXD3216S    414 32 bit/16 bit -> 32.16 signed fixed point divide
;       FXD3216U    485 32 bit/16 bit -> 32.16 unsigned fixed point divide
;       FXD3215U    390 32 bit/15 bit -> 32.15 unsigned fixed point divide
;       FXD3115U    383 31 bit/15 bit -> 31.15 unsigned fixed point divide
;       FXD1616S    214 16 bit/16 bit -> 16.16 signed fixed point divide
;       FXD1616U    244 16 bit/16 bit -> 16.16 unsigned fixed point divide
;       FXD1615U    197 16 bit/15 bit -> 16.15 unsigned fixed point divide
;       FXD1515U    191 15 bit/15 bit -> 15.15 unsigned fixed point divide
;       FXD1608S    146 16 bit/08 bit -> 16.08 signed fixed point divide
;       FXD1608U    196 16 bit/08 bit -> 16.08 unsigned fixed point divide
;       FXD1607U    130 16 bit/07 bit -> 16.07 unsigned fixed point divide
;       FXD1507U    125 15 bit/07 bit -> 15.07 unsigned fixed point divide
                list    r=dec,x=on,t=off,p=17C42
                include <PIC17.INC>
;*****************************************************************************************
;*****************************************************************************************
;       Define divide register variables
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
AARG            equ     0x19    ; most significant byte of argument A
BARG            equ     0x22    ; most significant byte of argument B
REM             equ     0x1D    ; most significant byte of remainder
;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )
;*****************************************************************************************
;*****************************************************************************************
;       32 Bit Division Macros
SDIV3216        macro
;       Max Timing:     5+8+30*12+6 = 379 clks
;       Min Timing:     5+8+30*11+6 = 349 clks
;       PM: 5+8+30*14+6 = 439           DM: 8
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                  i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
```

```
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            SADD268
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK268
SADD268         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0

SOK268          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            SADD2616
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK2616
SADD2616        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK2616 RLCF                    ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
```

```
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            SADD2624
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK2624
SADD2624        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK2624         RLCF            ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            SOK26
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26
                endm
UDIV3216  macro
;       restore = 15/20 clks,  nonrestore = 11/14 clks
;       Max Timing: 16*15+1+16*20 = 561 clks
;       Min Timing: 16*11+1+16*14 = 401 clks
;       PM:  16*15+1+16*20 = 561                DM: 9
                variable         i
                i = 0
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK26#v(i)      RLCF            ACC+B0
                i=i+1
                endw
```

```
                i = 8
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK26#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                CLRF            TEMP
                i = 16
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
                BCF             _C
UOK26#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                i = 24
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
                BCF             _C
UOK26#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                endm
```

```
NDIV3216        macro
;       Max Timing:     10+31*15+6 = 481 clks
;       Min Timing: 10+31*14+6 = 450 clks
;       PM: 10+31*19+6 = 605            DM: 9
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            TEMP,W
                SUBWFB          TEMP
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
        BTFSS                   ACC+B0,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD268
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK268
NADD268         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK268          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
```

```
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD2616
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK2616
NADD2616        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK2616         RLCF            ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            NADD2624
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK2624
NADD2624        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
```

```
                CLRF            WREG
                ADDWFC          TEMP
NOK2624         RLCF            ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            NOK26
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
NOK26
                endm
UDIV3215        macro
;       Max Timing:     8+31*12+6 = 386 clks
;       Min Timing:     8+31*11+6 = 355 clks
;       PM: 8+31*14+6 = 448             DM: 8
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
```

```
                BTFSS           ACC+B0,LSB
                GOTO            UADD258
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK258
UADD258         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK258          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD2516
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK2516
UADD2516        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK2516         RLCF            ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD2524
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
```

```
                GOTO            UOK2524
UADD2524        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK2524         RLCF            ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            UOK25
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25
                endm
UDIV3115        macro
;       Max Timing:     5+8+30*12+6 = 379 clks
;       Min Timing:     5+8+30*11+6 = 349 clks
;       PM: 5+8+30*14+6 = 439           DM: 8
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
```

```
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD158
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK158
UADD158         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK158          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD1516
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK1516
UADD1516        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK1516         RLCF            ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD1524
                SUBWF           REM+B1
```

```
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK1524
UADD1524        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK1524 RLCF                    ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            UOK15
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15
                endm
;*****************************************************************************************
;*****************************************************************************************
;       16/16 Bit Division Macros
SDIV1616        macro
;       Max Timing:     5+8+14*12+6 = 187 clks
;       Min Timing:     5+8+14*11+6 = 173 clks
;       PM: 5+8+14*14+6 = 215           DM: 6
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
        BTFSS                   ACC+B0,LSB
                GOTO            SADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK66#v(i)
SADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
```

```
SOK66#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            SADD668
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK668
SADD668         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK668          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            SADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK66#v(i)
SADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK66#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            SOK66
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK66
                endm
UDIV1616  macro
;       restore = 15 clks,  nonrestore = 11 clks
;       Max Timing: 8*15+8*15 = 240 clks
;       Min Timing: 8*11+8*11 = 176 clks
;       PM: 8*15+8*15 = 240             DM: 6
                variable         i
                i = 0
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK66#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK66#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                i = 8
```

```
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK66#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK66#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                endm
NDIV1616        macro
;       Max Timing:     9+15*15+6 = 240 clks
;       Min Timing:     9+15*14+6 = 225 clks
;       PM: 9+15*19+6 = 300             DM: 7
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            TEMP,W
                SUBWFB          TEMP
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK66#v(i)

NADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK66#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD668
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
```

```
                GOTO            NOK668
NADD668         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK668          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK66#v(i)
NADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK66#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            NOK66
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
NOK66
                endm
UDIV1615        macro
;       Max Timing:     7+15*12+6 = 193 clks
;       Min Timing:     7+15*11+6 = 178 clks
;       PM: 7+15*14+6 = 213             DM: 6
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                i = 1
               while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD65#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK65#v(i)
UADD65#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK65#v(i)      RLCF            ACC+B0
                i=i+1
               endw
                RLCF            ACC+B1,W
```

```
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD658
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK658
UADD658         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK658          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD65#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK65#v(i)
UADD65#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK65#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            UOK65
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK65
                endm
UDIV1515        macro
;       Max Timing:     5+8+14*12+6 = 187 clks
;       Min Timing:     5+8+14*11+6 = 173 clks
;       PM: 5+8+14*14+6 = 215           DM: 6
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD55#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK55#v(i)
```

```
UADD55#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK55#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD558

                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK558
UADD558         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK558          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG

                BTFSS           ACC+B1,LSB
                GOTO            UADD55#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK55#v(i)
UADD55#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK55#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            UOK55
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK55
                endm
;_____________________________________________________________________________
;       Extra 16 Bit Divide Macros
DIV1616         macro
;       Timing: restore = 16 clks,  nonrestore = 13 clks        16*16 = 256 clks
                variable i
                i = 0
                while i < 16
                RLCF            AARG+B1
                RLCF            AARG+B0
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSS           _C
                GOTO            RS1616_#v( i )
                BSF             AARG+B1,LSB
                GOTO            OK1616_#v( i )
RS1616_#v( i )  MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
```

```
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             AARG+B1,LSB
OK1616_#v(i)
                i=i+1
                endw
                endm
DIVMAC          macro
;       Timing: restore = 19 clks,  nonrestore = 14 clks        16*19 = 304 clks
                variable i
                i = 0
                while i < 16
                RLCF            AARG+B1
                RLCF            AARG+B0
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0,W
                BTFSS           _Z
                GOTO            notz#v( i )
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1,W
notz#v( i )     BTFSS           _C
                GOTO            nosub#v( i )
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BSF             AARG+B1,LSB
                GOTO            ok#v(i)
nosub#v(i)      BCF             AARG+B1,LSB
ok#v(i)
                i=i+1
                endw
                endm
;*****************************************************************************************
;*****************************************************************************************
;
;       16/08 BIT Division Macros
SDIV1608        macro
;       Max Timing:     3+5+14*8+2 = 122 clks
;       Min Timing:     3+5+14*8+2 = 122 clks
;       PM: 3+5+14*8+2 = 122            DM: 4
                variable i
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
```

```
                ADDWF           REM+B0
                RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                endm
UDIV1608  macro
;       restore = 9/15 clks,  nonrestore = 8/11 clks
;       Max Timing: 8*9+1+8*15 = 193 clks       max
;       Min Timing: 8*8+1+8*11 = 153 clks       min
;       PM: 8*9+1+8*15 = 193            DM: 4
                variable         i
                i = 0
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                BTFSC           _C
                GOTO            UOK68#v(i)
                ADDWF           REM+B0
                BCF             _C
UOK68#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                CLRF            TEMP
                i = 8
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                BTFSC           _C
                GOTO            UOK68#v(i)
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP
                BCF             _C
UOK68#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                endm
NDIV1608        macro
;       Max Timing:     7+15*12+3 = 190 clks
;       Min Timing: 7+15*11+3 = 175 clks
;       PM: 7+15*14+3 = 220             DM: 5
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                CLRF            TEMP,W
                SUBWFB          TEMP
                RLCF            ACC+B0
```

```
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD68#v(i)
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK68#v(i)
NADD68#v(i)     ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK68#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD688
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK688
NADD688         ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK688          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD68#v(i)
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK68#v(i)
NADD68#v(i)     ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP

NOK68#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                endm
UDIV1607        macro
;       Max Timing:     5+15*8+2 = 127 clks
;       Min Timing:     5+15*8+2 = 127 clks
;       PM: 5+15*8+2 = 127              DM: 4
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
```

```
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B0

                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                 i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                endm
UDIV1507        macro
;       Max Timing:     3+5+14*8+2 = 122 clks
;       Min Timing:     3+5+14*8+2 = 122 clks
;       PM: 3+5+14*8+2 = 122            DM: 4
                variable i
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
```

```
                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                endm
;*****************************************************************************************
;*****************************************************************************************
;       32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16
;       Input:  32 bit signed fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3216S
;       Output: 32 bit signed fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     11+379+3 = 393 clks             A > 0, B > 0
;                                       14+379+17 = 410 clks            A > 0, B < 0
;                                       18+379+17 = 414 clks            A < 0, B > 0
;                                       21+379+3 = 403 clks             A < 0, B < 0
;       Min Timing:     11+349+3 = 363 clks             A > 0, B > 0
;                                       14+349+17 = 380 clks            A > 0, B < 0
;                                       18+349+17 = 384 clks            A < 0, B > 0
;                                       21+349+3 = 373 clks             A < 0, B < 0
;       PM: 21+439+16 = 476             DM: 9
FXD3216S        MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVWF           SIGN
                CLRF            REM+B0
                CLRF            REM+B1,W
                BTFSS           BARG+B0,MSB        ; if MSB set go & negate BARG
                GOTO            CA3216S
                COMF            BARG+B1
                COMF            BARG+B0
                INCF            BARG+B1
                ADDWFC          BARG+B0
CA3216S         BTFSS           AARG+B0,MSB        ; if MSB set go & negate ACCa
                GOTO            C3216S
                COMF            AARG+B3
                COMF            AARG+B2
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B3
                ADDWFC          AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
C3216S  SDIV3216
                BTFSS           SIGN,MSB        ; negate (ACCc,ACCd)
                RETLW           0x00
                COMF            AARG+B3
                COMF            AARG+B2
                COMF            AARG+B1
                COMF            AARG+B0
                CLRF            WREG
                INCF            AARG+B3
                ADDWFC          AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
                COMF            REM+B1
                COMF            REM+B0
                INCF            REM+B1
                ADDWFC          REM+B0
                RETLW           0x00

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

```
;       32/16 Bit Unsigned Fixed Point Divide 32/16 -> 32.16
;       Input:  32 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3216U
;       Output: 32 bit unsigned fixed point quotient in AARG+B0, AARG+B1AARG+B2,AARG+B3
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     2+481+2 = 485 clks
;       Min Timing:     2+450+2 = 459 clks
;       PM: 2+605+1 = 608               DM: 9
FXD3216U        CLRF            REM+B0
                CLRF            REM+B1
                NDIV3216
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       32/15 Bit Unsigned Fixed Point Divide 32/15 -> 32.15

;       Input:  32 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3215U
;       Output: 32 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     2+386+2 = 390 clks
;       Min Timing:     2+355+2 = 359 clks
;       PM: 2+448+1 = 451               DM: 8
FXD3215U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV3215
                 RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       31/15 Bit Unsigned Fixed Point Divide 31/15 -> 31.15
;       Input:  31 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3115U
;       Output: 31 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     2+379+2 = 383 clks
;       Min Timing:     2+349+2 = 353 clks
;       PM: 2+439+1 = 442               DM: 8
FXD3115U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV3115
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16
;       Input:  16 bit fixed point dividend in AARG+B0, AARG+B1
;               16 bit fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD1616S
;       Output: 16 bit fixed point quotient in AARG+B0, AARG+B1
;               16 bit fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     11+187+3 = 201 clks             A > 0, B > 0
;                                       14+187+13 = 214 clks            A > 0, B < 0
;                                       14+187+13 = 214 clks            A < 0, B > 0
;                                       17+187+3 = 207 clks             A < 0, B < 0
;       Min Timing:     11+173+3 = 187 clks             A > 0, B > 0
;                                       14+173+13 = 200 clks            A > 0, B < 0
;                                       14+173+13 = 200 clks            A < 0, B > 0
;                                       17+173+3 = 193 clks             A < 0, B < 0
;       PM: 14+215+12 = 241             DM: 7
FXD1616S        MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVWF           SIGN
                CLRF            REM+B0
                CLRF            REM+B1,W
```

```
                BTFSS           BARG+B0,MSB        ; if MSB set go & negate BARG
                GOTO            CA1616S

                COMF            BARG+B1
                COMF            BARG+B0
                INCF            BARG+B1
                ADDWFC          BARG+B0
CA1616S         BTFSS           AARG+B0,MSB        ; if MSB set go & negate ACCa
                GOTO            C1616S
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B1
                ADDWFC          AARG+B0
C1616S  SDIV1616
                BTFSS           SIGN,MSB            ; negate (ACCc,ACCd)
                RETLW           0x00
                COMF            AARG+B1
                COMF            AARG+B0
                CLRF            WREG
                INCF            AARG+B1
                ADDWFC          AARG+B0
                COMF            REM+B1
                COMF            REM+B0
                INCF            REM+B1
                ADDWFC          REM+B0
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD1616U
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     2+240+2 = 244 clks
;       Min Timing:     2+176+2 = 180 clks
;       PM: 2+240+1 = 243               DM: 6
FXD1616U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV1616
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/15 Bit Unsigned Fixed Point Divide 16/15 -> 16.15
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD1615U
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     2+193+2 = 197 clks
;       Min Timing:     2+178+2 = 182 clks
;      PM: 2+213+1 = 216               DM: 6
FXD1615U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV1615
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15
;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;      Use:    CALL    FXD1515U
;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     2+187+2 = 191 clks
;       Min Timing:     2+173+2 = 177 clks
```

```
;       PM: 2+215+1 = 218               DM: 6
FXD1515U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV1515
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;
;       16/8 Bit Signed Fixed Point Divide 16/08 -> 16.08
;       Input:  16 bit fixed point dividend in AARG+B0, AARG+B1
;               8 bit fixed point divisor in BARG+B0
;       Use:    CALL    FXD1608S
;       Output: 16 bit fixed point quotient in AARG+B0, AARG+B1
;               8 bit fixed point remainder in REM+B0
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     10+122+3 = 135 clks             A > 0, B > 0
;                                       11+122+11 = 144 clks            A > 0, B < 0
;                                       13+122+11 = 146 clks            A < 0, B > 0
;                                       14+122+3 = 139 clks             A < 0, B < 0
;       Min Timing:     10+122+3 = 135 clks             A > 0, B > 0
;                                       11+122+11 = 144 clks            A > 0, B < 0
;                                       13+122+11 = 146 clks            A < 0, B > 0
;                                       14+122+3 = 139 clks             A < 0, B < 0
;       PM: 14+122+10 = 146             DM: 5
FXD1608S        MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVWF           SIGN
                CLRF            REM+B0,W
                BTFSS           BARG+B0,MSB             ; if MSB set go & negate BARG
                GOTO            CA1608S
                COMF            BARG+B0
                INCF            BARG+B0
CA1608S         BTFSS           AARG+B0,MSB             ; if MSB set go & negate ACCa
                GOTO            C1608S
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B1
                ADDWFC          AARG+B0
C1608S  SDIV1608
                BTFSS           SIGN,MSB                ; negate (ACCc,ACCd)
                RETLW           0x00
                COMF            AARG+B1
                COMF            AARG+B0
                CLRF            WREG
                INCF            AARG+B1
                ADDWFC          AARG+B0
                COMF            REM+B0
                INCF            REM+B0
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;
;       16/8 Bit Unsigned Fixed Point Divide 16/08 -> 16.08
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;                8 bit unsigned fixed point divisor in BARG+B0
;       Use:    CALL    FXD1608U
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               8 bit unsigned fixed point remainder in REM+B0
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     1+193+2 = 196 clks
;       Min Timing:     1+153+2 = 156 clks
;       PM: 1+193+1 = 195               DM: 4
FXD1608U        CLRF            REM+B0
                UDIV1608
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;
;       16/7 Bit Unsigned Fixed Point Divide 16/07 -> 16.07
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;                7 bit unsigned fixed point divisor in BARG+B0
;       Use:    CALL    FXD1607U
```

```
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     1+127+2 = 130 clks
;       Min Timing:     1+127+2 = 130 clks
;       PM: 1+127+1 = 129               DM: 4
FXD1607U        CLRF            REM+B0
                UDIV1607
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;
;       15/7 Bit Unsigned Fixed Point Divide 15/07 -> 15.07
;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;                7 bit unsigned fixed point divisor in BARG+B0
;       Use:    CALL    FXD1507U
;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0
;       Result: AARG, REM  <—  AARG / BARG
;       Max Timing:     1+122+2 = 125 clks
;       Min Timing:     1+122+2 = 125 clks
;       PM: 1+122+1 = 124             DM: 4
FXD1507U        CLRF            REM+B0
                UDIV1507
                RETLW           0x00
                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