AN544

Math Utility Routines

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

PLEASE NOTE: *This application note uses the old Microchip Math Routine format. It is intended for reference purposes only and is being provided for those of you still implementing Binary Coded Decimal(BCD) routines. For any new designs, please refer to application notes contained in Microchip's Embedded Control Handbook Volume II - Math Library*

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

MS-DOS is a registered trademark of Microsoft Corporation.

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_myfF	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 Y_0 using the first order Taylor polynomial expansion as:

EQUATION 1:

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

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

If Y0 is an approximate root of f(Y), then the higher order terms in the above equation are negligible.

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

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

Thus X is a better approximation for Y_0. From the previous equation, the sequence {Xn} can be generated:

EQUATION 2: $$X_n = X_{n-1} - \frac{f(X_{n-1})}{f'(X_{n-1})}, n \geq 1$$

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.

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 ADDITION AND SUBTRACTION

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

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 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:

EQUATION 4: 3-POINT FORMULA:

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	1	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 PICmicro™ series). The autocorrelation 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:

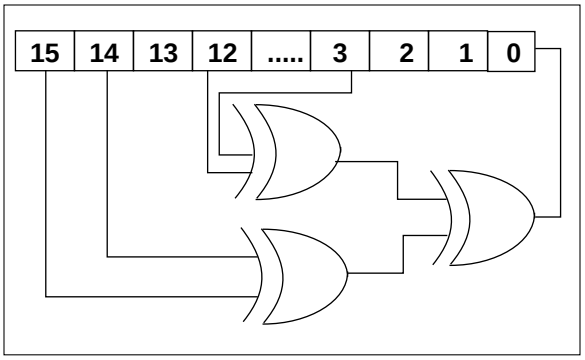


TABLE 8: DIFFERENTIATION

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: AUTOCORRELATION OF THE DATA POINTS GENERATED BY THE RANDOM NUMBER GENERATOR

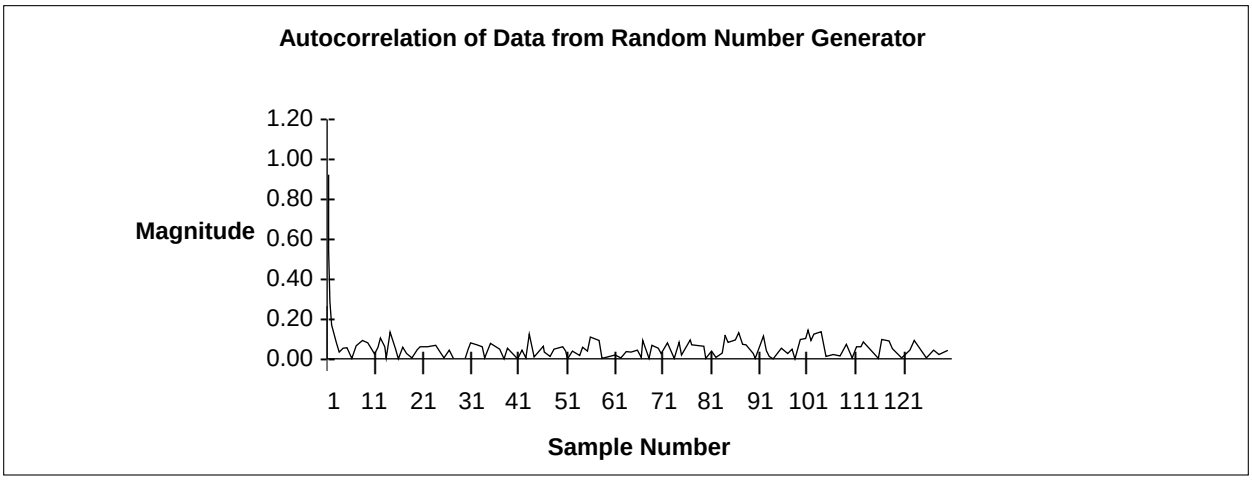
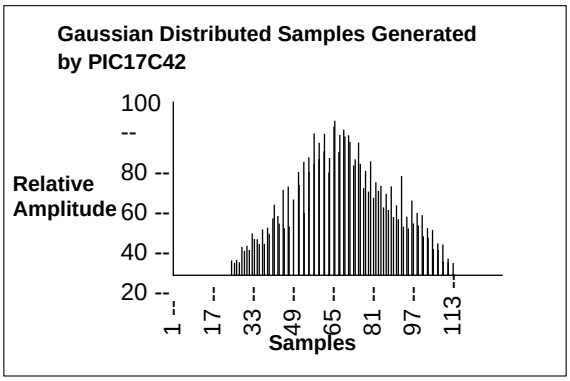


TABLE 10: RANDOM DOUBLE GENERATOR

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 PICmicro 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.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

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

```
MPASM 01.40 Released           ARITH.ASM   1-16-1997  15:10:04         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

             00001
             00002         TITLE   “General Purpose Math Routines For PIC17C42 : Ver 1.0”
             00003
             00004         LIST    P = 17C42, columns=120, WRAP, L=0, R = DEC
             00005 ;
             00006         include      <p17c42.inc>
             00001         LIST
             00002 ; P17C42.INC  Standard Header File, Version 1.03 Microchip Technology, Inc.
             00264         LIST
             00007
             00008         #define TRUE    1
             00009         #define FALSE   0
             00010
  00000001   00011 _INC    equ     1
  00000000   00012 _NO_INC equ     0
  00000000   00013 _LOW    equ     0
  00000001   00014 _HIGH   equ     1
             00015
             00016 ;
             00017 ;*****************************************************************
             00018 ;       Define RAM Locations necessary For the “ARITH.ASM”
             00019 ;   RAM locations should be defined before calling the library math
             00020 ; routines
             00021 ;
             00022 ;       Program:          ARITH.ASM
             00023 ;       Revision Date:
             00024 ;                         1-13-97      Compatibility with MPASMWIN 1.40
             00025 ;
             00026 ;*****************************************************************
             00027 ;
  00000001   00028 MODE_FAST       equ     TRUE
  00000000   00029 SIGNED          equ     FALSE
             00030 ;
             00031 ;*****************************************************************
             00032 ;
             00033  #if MODE_FAST
             00034
             00035   CBLOCK  0x18
  00000018   00036      ACCaLO, ACCaHI, ACCbLO, ACCbHI     ; Ram Locations for Arithmetic
  0000001C   00037      ACCcLO, ACCcHI, ACCdLO, ACCdHI     ;     Routines
             00038   ENDC
             00039
             00040  #else
             00041
             00042   CBLOCK  0x20
             00043      ACCaLO, ACCaHI, ACCbLO, ACCbHI
             00044      ACCcLO, ACCcHI, ACCdLO, ACCdHI
             00045   ENDC
             00046
             00047  #endif
             00048 ;
             00049
             00050   CBLOCK
```

```
  00000020    00051      tempLo, tempHi, count, sign
              00052   ENDC
              00053
              00054   CBLOCK
  00000024    00055         NumLo, NumHi
  00000026    00056         iterCnt
              00057   ENDC
              00058 ;
              00059         CBLOCK                          ; RAM locations for “Diff” routine
  00000027    00060                 XnLo, XnHi, Xn_1_Lo
  0000002A    00061                 Xn_1_Hi, Xn_2_Lo, Xn_2_Hi
  0000002D    00062                 DiffKLo, DiffKHi               ; DiffK = h = Step Size
  0000002F    00063                 DiffLo, DiffHi
              00064         ENDC
              00065 ;
              00066         CBLOCK                          ; RAM Locations for “Integrate”
  00000031    00067                 X0Lo, X0Hi, X1Lo, X1Hi  ; Routine
  00000035    00068                 X2Lo, X2Hi, X3Lo, X3Hi
  00000039    00069                 IntgKLo, IntgKHi           ; INTEGRATE CONST = 3*h/8
  0000003B    00070                 IntgLo, IntgHi
              00071         ENDC
              00072 ;
              00073 ;*****************************************************************
              00074 ;
  00000018    00075 mulcnd  equ     ACCaLO
  00000019    00076 mulplr  equ     ACCaHI
  0000001A    00077 L_byte  equ     ACCbLO
  0000001B    00078 H_byte  equ     ACCbHI
              00079 ;
  0000000A    00080 _LUPCNT  equ    10             ; Set Desired Number of iterations
  0000001E    00081 SqrtLo  equ     ACCdLO     ; for Square Root Routine(NEWTON Iterations)
  0000001F    00082 SqrtHi  equ     ACCdHI
              00083 ;
              00084 ;  Define RAM locations for the Random Number Generators
              00085 ;
  00000018    00086 RandLo  equ     ACCaLO
  00000019    00087 RandHi  equ     ACCaHI   ; 16 bit Pseudo Random Number
  0000001B    00088 GaussHi equ     ACCbHI
  0000001A    00089 GaussLo equ     ACCbLO ; 16 bit Gaussian distributed number
  00000020    00090 GaussTmp equ    tempLo
              00091 ;
              00092         PAGE
              00093
0000          00094         ORG     0x0000
              00095 ;*****************************************************************
              00096 ;                        Math Routines Test Program
              00097 ;*****************************************************************
              00098 ;
              00099 ;    Load constant values to ACCa & ACCb for testing
              00100 ;
0000          00101 main
0000 E02D     00102         call    loadAB          ; result of adding ACCb+ACCa->ACCb
0001 E036     00103         call    D_add           ; Here Accb = 81FE
              00104 ;
0002 E02D     00105         call    loadAB          ; result of subtracting ACCb - ACCa->ACCb
0003 E03B     00106         call    D_sub           ; Here Accb = 7E00
              00107 ;
0004 E02D     00108         call    loadAB          ; result of multiplying ACCb*ACCa->(ACCd,ACCc)
0005 E050     00109         call    D_mpyS          ; Here (ACCd,ACCc) = 00FF 7E01
              00110 ;
0006 E02D     00111         call    loadAB          ; result of multiplying ACCb*ACCa->(ACCd,ACCc)
0007 E065     00112         call    D_mpyF          ; Here (ACCd,ACCc) = 00FF 7E01
              00113 ;
0008 E02D     00114         call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
0009 E119     00115         call    D_divS          ; Here (ACCd,ACCc) = 0040 003f
              00116 ;
```

```
000A E02D     00117        call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
000B E138     00118         call    D_divF           ; Here (ACCd,ACCc) = 0040 003f
              00119 ;
000C B0F3     00120         movlw   xf3
000D 0125     00121         movwf   NumHi
000E B0F6     00122         movlw   xf6              ; Set input test number = 62454
000F 0124     00123         movwf   NumLo            ;  = F3F6h
0010 E27D     00124         call    Sqrt             ; result = 00F9h = 249 (in SqrtLo)
              00125 ;                                ; exact sqrt(62454) = 249.9
              00126 ;
0011 B0FF     00127         movlw   0xff
0012 0119     00128         movwf   mulplr           ; multiplier (in mulplr) = 0FF
0013 B0FF     00129         movlw   0xff             ; multiplicand(W Reg )   = 0FF
0014 0118     00130         movwf   mulcnd
0015 E293     00131         call    mpy8x8_F         ; The result 0FF*0FF = FE01 is in locations
              00132 ;                                ; H_byte & L_byte
0016 B0FF     00133         movlw   0xff
0017 0119     00134         movwf   mulplr           ; multiplier (in mulplr) = 0FF
0018 B0FF     00135         movlw   0xff             ; multiplicand(W Reg )   = 0FF
0019 0118     00136         movwf   mulcnd
001A E2B8     00137         call    mpy8x8_S         ; The result 0FF*0FF = FE01 is in locations
              00138 ;                                ; H_byte & L_byte
              00139 ; Test The Random Number Generators
              00140 ; Capture data into trace buffer by TABLE WRITES to a
              00141 ; dummy Program Memory location
              00142 ;
001B B0FF     00143         movlw   0xff
001C 010D     00144         movwf   TBLPTRL
001D B05F     00145         movlw   0x5f
001E 010E     00146         movwf   TBLPTRH
              00147 ;
001F B030     00148         movlw   0x30
0020 0119     00149         movwf   RandHi
0021 B045     00150         movlw   0x45
0022 0118     00151         movwf   RandLo
              00152 ;
0023 C028     00153         goto    GaussPoint
              00154 ;
0024          00155 RandPoint
0024 E311     00156         call    Random16
0025 A418     00157         tlwt    _LOW,RandLo           ; only for data capture
0026 AE19     00158         tablwt  _HIGH,0,RandHi        ; using PICMASTER
0027 C024     00159         goto    RandPoint
              00160 ;
0028          00161 GaussPoint
0028 E31E     00162         call    Gauss
0029 A41A     00163         tlwt    _LOW,GaussLo          ; only for data capture
002A AE1B     00164         tablwt  _HIGH,0,GaussHi       ; using PICMASTER
002B C028     00165         goto    GaussPoint
              00166 ;
002C C02C     00167 self    goto    self                  ; End Of Test Routines
              00168 ;
002D          00169 loadAB
002D B001     00170         movlw   0x01
002E 0119     00171         movwf   ACCaHI
002F B0FF     00172         movlw   0xff                 ; loads ACCa = 01FF
0030 0118     00173         movwf   ACCaLO
              00174 ;
0031 B07F     00175         movlw   0x7f
0032 011B     00176         movwf   ACCbHI
0033 B0FF     00177         movlw   0xFF                 ; loads ACCb = 7FFF
0034 011A     00178         movwf   ACCbLO
0035 0002     00179         return
              00180 ;
              00181         PAGE
              00182 ;*****************************************************************
```

```
                00183 ;                Double Precision Arithmetic Routines
                00184 ;
                00185 ;    Routines : Addition, Subtraction, Multiplication ,Division
                00186 ;               Square Root
                00187 ;
                00188 ;               NOTE :  MODE_FAST must first be set to either
                00189 ;                       TRUE or FALSE
                00190 ;
                00191 ;  MODE_FAST determines the RAM address locations of ACCa thru ACCd
                00192 ;
                00193 ;  If MODE_FAST is set TRUE, data transfers can be done efficiently
                00194 ;  using “MOVFP” & “MOVPF” instructions instead of indirectly moving
                00195 ;  at first to W Reg and then to the desired RAM locations
                00196 ;
                00197 ;         The speed increase using this way of locating ACCa to
                00198 ;         ACCd will result in a saving of about 20 Cycles/filter stage
                00199 ;         In this case ( a 2 stage filter), it is faster by 40 Cycles
                00200 ;
                00201 ;  If due to other constraints, ACCa thru ACCd cannot be set at
                00202 ; address 0x18 to 0x1f, then the user is required to set
                00203 ; MODE_FAST to FALSE
                00204 ;
                00205         PAGE
                00206 ;*****************************************************************
                00207 ;                Double Precision Addition
                00208 ;
                00209 ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                00210 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                00211 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                00212 ;      (c) CALL D_add
                00213 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                00214 ;
                00215 ;   Performance :
                00216 ;               Program Memory  :  4 (excluding call & return)
                00217 ;               Clock Cycles    :  4 (excluding call & return)
                00218 ;               W Register      :  Used
                00219 ;               Scratch RAM     :  0
                00220 ;
                00221 ;*****************************************************************;
                00222 ;
0036            00223 D_add
0036 6A18       00224         movfp    ACCaLO,WREG
0037 0F1A       00225         addwf    ACCbLO, F           ;addwf lsb
0038 6A19       00226         movfp    ACCaHI,WREG
0039 111B       00227         addwfc   ACCbHI, F           ;addwf msb with carry
003A 0002       00228         return
                00229 ;
                00230         PAGE
                00231 ;*****************************************************************
                00232 ;                Double Precision Subtraction
                00233 ;
                00234 ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                00235 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                00236 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                00237 ;      (c) CALL D_sub
                00238 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                00239 ;
                00240 ;   Performance :
                00241 ;               Program Memory  :   4  (excluding call & return )
                00242 ;               Clock Cycles    :   4  (excluding call & return )
                00243 ;               W Register      :   Used
                00244 ;               scratch RAM     :   0
                00245 ;*****************************************************************;
                00246 ;
003B            00247 D_sub
003B 6A18       00248         movfp    ACCaLO,WREG
```

```
003C 051A     00249         subwf     ACCbLO, F
003D 6A19     00250         movfp     ACCaHI,WREG
003E 031B     00251         subwfb    ACCbHI, F
003F 0002     00252         return
              00253 ;
              00254         PAGE
              00255 ;*****************************************************************
              00256 ;               Function to negate a 16 bit integer
              00257 ;  The two 8 bit integers are assumed to be in 2 consecutive
              00258 ; locations. Before calling this routine, FSR0 should be loaded with
              00259 ; the address of the lower byte.
              00260 ;       Assume that ALUSTA register is set for no autoincrement of
              00261 ; FSR0.
              00262 ;*****************************************************************
              00263 ;
0040          00264 negateAlt
0040 6A00     00265         movfp    INDF0,WREG
0041 8D04     00266         bcf      ALUSTA,FS1
0042 2D00     00267         negw     INDF0, F
0043 8504     00268         bsf      ALUSTA,FS1
0044 6A00     00269         movfp    INDF0,WREG
0045 2900     00270         clrf     INDF0, F
0046 0300     00271         subwfb   INDF0, F
0047 0002     00272         return
              00273 ;
0048          00274 negate
0048 1300     00275         comf    INDF0, F
0049 8D04     00276         bcf     ALUSTA,FS1
004A 1500     00277         incf    INDF0, F
004B 8504     00278         bsf     ALUSTA,FS1
004C 9A04     00279         btfsc   ALUSTA,Z
004D 0700     00280         decf    INDF0, F
004E 1300     00281         comf    INDF0, F
004F 0002     00282         return
              00283 ;
              00284         PAGE
              00285 ;*****************************************************************
              00286 ;                       Double Precision Multiplication
              00287 ;
              00288 ;               ( Optimized for Code : Looped Code )
              00289 ;
              00290 ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
              00291 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
              00292 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
              00293 ;      (c) CALL D_mpyS
              00294 ;      (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
              00295 ;
              00296 ;   Performance :
              00297 ;            Program Memory  :   21 (UNSIGNED)
              00298 ;                                52 (SIGNED)
              00299 ;            Clock Cycles    :   242 (UNSIGNED :excluding CALL & RETURN)
              00300 ;                            :   254 (SIGNED :excluding CALL & RETURN)
              00301 ;            Scratch RAM     :   1 (used only if SIGNED arithmetic)
              00302 ;
              00303 ;       Note : The above timing is the worst case timing, when the
              00304 ;              register ACCb = FFFF. The speed may be improved if
              00305 ;              the register ACCb contains a number ( out of the two
              00306 ;              numbers ) with less number of 1s.
              00307 ;
              00308 ;              Double Precision Multiply ( 16x16 -> 32 )
              00309 ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
              00310 ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
              00311 ;*****************************************************************
              00312 ;
0050          00313 D_mpyS                        ;results in ACCd(16 msb's) and ACCc(16 lsb's)
              00314 ;
```

```
                00315
                00316      #if   SIGNED
                00317         CALL    S_SIGN
                00318      #endif
                00319 ;
0050 2922       00320         clrf     count, F
0051 8422       00321         bsf      count,4                ; set count = 16
                00322 ;
                00323   #if MODE_FAST
0052 5A20       00324         movpf    ACCbLO,tempLo
0053 5B21       00325         movpf    ACCbHI,tempHi
                00326   #else
                00327         movfp    ACCbLO,WREG
                00328         movwf    tempLo
                00329         movfp    ACCbHI,WREG
                00330         movwf     empHi
                00331   #endif
0054 291F       00332         clrf     ACCdHI, F
0055 291E       00333         clrf     ACCdLO, F
                00334 ;
                00335 ; shift right and addwf 16 times
                00336 ;
0056            00337 mpyLoop
0056 1921       00338         rrcf     tempHi, F
0057 1920       00339         rrcf     tempLo, F
0058 9004       00340         btfss    ALUSTA,C
0059 C05E       00341         goto     NoAdd                  ; LSB is 0, so no need to addwf
005A 6A18       00342         movfp    ACCaLO,WREG
005B 0F1E       00343         addwf    ACCdLO, F              ;addwf lsb
005C 6A19       00344         movfp    ACCaHI,WREG
005D 111F       00345         addwfc   ACCdHI, F              ;addwf msb
005E            00346 NoAdd
005E 191F       00347         rrcf     ACCdHI, F
005F 191E       00348         rrcf     ACCdLO, F
0060 191D       00349         rrcf     ACCcHI, F
0061 191C       00350         rrcf     ACCcLO, F
0062 1722       00351         decfsz   count, F
0063 C056       00352         goto     mpyLoop
                00353 ;
                00354     #if SIGNED
                00355         btfss    sign,MSB
                00356         return
                00357         comf     ACCcLO, F
                00358         incf     ACCcLO, F
                00359         btfsc    ALUSTA,Z
                00360         decf     ACCcHI, F
                00361         comf     ACCcHI, F
                00362         btfsc    ALUSTA,Z
                00363         decf     ACCdLO, F
                00364         comf     ACCdLO, F
                00365         btfsc    ALUSTA,Z
                00366         decf     ACCdHI, F
                00367         comf     ACCdHI, F
                00368         return
                00369     #else
0064 0002       00370         return
                00371     #endif
                00372 ;
                00373 ;  Assemble this section only if Signed Arithmetic Needed
                00374 ;
                00375      #if    SIGNED
                00376 ;
                00377 S_SIGN
                00378         movfp    ACCaHI,WREG
                00379         xorwf    ACCbHI,W
                00380         movwf    sign              ; MSB of sign determines whether signed
```

```
00381         btfss    ACCbHI,MSB        ; if MSB set go & negate ACCb
00382         goto     chek_A
00383         comf     ACCbLO, F
00384         incf     ACCbLO, F
00385         btfsc    ALUSTA,Z          ; negate ACCb
00386         decf     ACCbHI, F
00387         comf     ACCbHI, F
00388 ;
00389 chek_A
00390         btfss    ACCaHI,MSB        ; if MSB set go & negate ACCa
00391         return
00392         comf     ACCaLO, F
00393         incf     ACCaLO, F
00394         btfsc    ALUSTA,Z          ; negate ACCa
00395         decf     ACCaHI, F
00396         comf     ACCaHI, F
00397         return
00398 ;
00399      #endif
00400 ;
00401         PAGE
00402 ;*****************************************************************
00403 ;                       Double Precision Multiplication
00404 ;
00405 ;               ( Optimized for Speed : straight Line Code )
00406 ;
00407 ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
00408 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
00409 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
00410 ;      (c) CALL D_mpy
00411 ;      (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
00412 ;
00413 ;   Performance :
00414 ;            Program Memory  :   179 (UNSIGNED)
00415 ;                                204 (SIGNED)
00416 ;            Clock Cycles    :   176 (UNSIGNED :excluding CALL & RETURN)
00417 ;                            :   183 (SIGNED :excluding CALL & RETURN)
00418 ;
00419 ;       Note : The above timing is the worst case timing, when the
00420 ;              register ACCb = FFFF. The speed may be improved if
00421 ;              the register ACCb contains a number ( out of the two
00422 ;              numbers ) with less number of 1s.
00423 ;
00424 ;              The performance specs are for Unsigned arithmetic ( i.e,
00425 ;              with “SIGNED equ  FALSE “).
00426 ;
00427 ;              Upon return from subroutine, the input registers
00428 ;
00429 ;
00430 ;*********************************************************
00431 ;               Multiplication Macro
00432 ;*********************************************************
00433 ;
00434 mulMac  MACRO
00435         variable i
00436
00437         variable i = 0
00438
00439         #if SIGNED
00440            variable MUL_LP_CNT = 15
00441         #else
00442            variable MUL_LP_CNT = 16
00443         #endif
00444           .while i < MUL_LP_CNT
00445 ;
00446              .if i < 8
```

```
00447                btfss      ACCbLO,i        ; test low byte
00448             .else
00449                btfss      ACCbHI,i-8      ; test high byte
00450             .fi
00451                goto       NoAdd#v(i)      ; LSB is 0, so no need to addwf
00452                movfp     ACCaLO,WREG
00453                addwf     ACCdLO, F        ;addwf lsb
00454                movfp     ACCaHI,WREG
00455                addwfc    ACCdHI, F        ;addwf msb
00456 NoAdd#v(i)
00457                rrcf      ACCdHI, F
00458                rrcf      ACCdLO, F
00459                rrcf      ACCcHI, F
00460                rrcf      ACCcLO, F
00461                bcf       ALUSTA,C
00462                variable  i = i+1
00463          .endw
00464        #if SIGNED
00465                rrcf      ACCdHI, F
00466                rrcf      ACCdLO, F
00467                rrcf      ACCcHI, F
00468                rrcf      ACCcLO, F
00469                bcf      ALUSTA,C
00470        #endif
00471 ;
00472        ENDM
00473 ;
00474        PAGE
00475 ;*****************************************************************
00476 ;                       Double Precision Negate Macros
00477 ;*****************************************************************
00478 AltNegMac       MACRO   fileRegLo,fileRegHi
00479                        movfp    fileRegLo,WREG
00480                        negw     fileRegLo, F
00481                        movfp    fileRegHi,WREG
00482                        clrf     fileRegHi, F
00483                        subwfb    fileRegHi, F
00484                ENDM
00485
00486 ;
00487 negMac  MACRO   fileRegLo, fileRegHi
00488        comf    fileRegLo, F       ; negate FileReg ( -FileReg -> FileReg )
00489        incf    fileRegLo, F
00490        btfsc   ALUSTA,Z
00491        decf    fileRegHi, F
00492        comf    fileRegHi, F
00493        ENDM
00494 ;
00495 NegMac32        MACRO   x3,x2,x1,x0
00496        movfp   x3,WREG
00497        negw    x3, F
00498        movfp   x2,WREG
00499        clrf    x2, F
00500        subwfb  x2, F
00501        movfp   x1,WREG
00502        clrf    x1, F
00503        subwfb  x1, F
00504        movfp   x0,WREG
00505        clrf    x0, F
00506        subwfb  x0, F
00507                ENDM
00508 ;
00509        PAGE
00510 ;*****************************************************************;
00511 ;               Double Precision Multiply ( 16x16 -> 32 )
00512 ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
```

```
                00513 ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                00514 ;
0065            00515 D_mpyF                          ;results in ACCd(16 msb's) and ACCc(16 lsb's)
                00516 ;
                00517      #if   SIGNED
                00518 ;
                00519         movfp    ACCaHI,WREG
                00520         xorwf    ACCbHI,W
                00521         movwf    sign
                00522         btfss    ACCbHI,MSB        ; if MSB set go & negate ACCb
                00523         goto     chek_A_MSB_MPY
                00524 ;
                00525         negMac   ACCbLO,ACCbHI
                00526 ;
                00527 chek_A_MSB_MPY
                00528         btfss    ACCaHI,MSB        ; if MSB set go & negate ACCa
                00529         goto     continue_MPY
                00530         negMac   ACCaLO,ACCaHI
                00531 ;
                00532      #endif
                00533 ;
0065            00534 continue_MPY
0065 291F       00535         clrf     ACCdHI, F
0066 291E       00536         clrf     ACCdLO, F
0067 8804       00537         bcf      ALUSTA,C
                00538
                00539 ;
                00540 ; use the mulMac macro 16 times
                00541 ;
                00542         mulMac
  0000            M         variable i
                  M
  0000            M         variable i = 0
                  M
                  M         #if SIGNED
                  M            variable MUL_LP_CNT = 15
                  M         #else
  0010            M            variable MUL_LP_CNT = 16
                  M         #endif
                  M           .while i < MUL_LP_CNT
                  M ;
                  M              .if i < 8
0068 901A         M                 btfss      ACCbLO,i        ; test low byte
                  M              .else
                  M                 btfss      ACCbHI,i-8      ; test high byte
                  M              .fi
0069 C06E         M                 goto       NoAdd0          ; LSB is 0, so no need to addwf
006A 6A18         M                 movfp      ACCaLO,WREG
006B 0F1E         M                 addwf      ACCdLO, F       ; addwf lsb
006C 6A19         M                 movfp      ACCaHI,WREG
006D 111F         M                 addwfc     ACCdHI, F       ; addwf msb
006E              M NoAdd0
006E 191F         M                 rrcf       ACCdHI, F
006F 191E         M                 rrcf       ACCdLO, F
0070 191D         M                 rrcf       ACCcHI, F
0071 191C         M                 rrcf       ACCcLO, F
0072 8804         M                 bcf        ALUSTA,C
  0001            M                 variable   i = i+1
                  M ;
                  M              .if i < 8
0073 911A         M                 btfss      ACCbLO,i        ; test low byte
                  M              .else
                  M                 btfss      ACCbHI,i-8      ; test high byte
                  M              .fi
0074 C079         M                 goto       NoAdd1          ; LSB is 0, so no need to addwf
0075 6A18         M                 movfp      ACCaLO,WREG
```

```
0076 0F1E        M                addwf     ACCdLO, F           ;addwf lsb
0077 6A19        M                movfp     ACCaHI,WREG
0078 111F        M                addwfc    ACCdHI, F           ;addwf msb
0079             M NoAdd1
0079 191F        M                rrcf      ACCdHI, F
007A 191E        M                rrcf      ACCdLO, F
007B 191D        M                rrcf      ACCcHI, F
007C 191C        M                rrcf      ACCcLO, F
007D 8804        M                bcf       ALUSTA,C
  0002           M                variable  i = i+1
                 M ;
                 M             .if i < 8
007E 921A        M                btfss     ACCbLO,i           ; test low byte
                 M             .else
                 M                btfss     ACCbHI,i-8         ; test high byte
                 M             .fi
007F C084        M                goto      NoAdd2             ; LSB is 0, so no need to addwf
0080 6A18        M                movfp     ACCaLO,WREG
0081 0F1E        M                addwf     ACCdLO, F          ; addwf lsb
0082 6A19        M                movfp     ACCaHI,WREG
0083 111F        M                addwfc    ACCdHI, F          ; addwf msb
0084             M NoAdd2
0084 191F        M                rrcf      ACCdHI, F
0085 191E        M                rrcf      ACCdLO, F
0086 191D        M                rrcf      ACCcHI, F
0087 191C        M                rrcf      ACCcLO, F
0088 8804        M                bcf       ALUSTA,C
  0003           M                variable  i = i+1
                 M ;
                 M             .if i < 8
0089 931A        M                btfss     ACCbLO,i          ; test low byte
                 M             .else
                 M                btfss     ACCbHI,i-8        ; test high byte
                 M             .fi
008A C08F        M                goto      NoAdd3              ; LSB is 0, so no need to addwf
008B 6A18        M                movfp     ACCaLO,WREG
008C 0F1E        M                addwf     ACCdLO, F         ; addwf lsb
008D 6A19        M                movfp     ACCaHI,WREG
008E 111F        M                addwfc    ACCdHI, F         ; addwf msb
008F             M NoAdd3
008F 191F        M                rrcf      ACCdHI, F
0090 191E        M                rrcf      ACCdLO, F
0091 191D        M                rrcf      ACCcHI, F
0092 191C        M                rrcf      ACCcLO, F
0093 8804        M                bcf       ALUSTA,C
  0004           M                variable  i = i+1
                 M ;
                 M             .if i < 8
0094 941A        M                btfss     ACCbLO,i         ; test low byte
                 M             .else
                 M                btfss     ACCbHI,i-8       ; test high byte
                 M             .fi
0095 C09A        M                goto      NoAdd4           ; LSB is 0, so no need to addwf
0096 6A18        M                movfp     ACCaLO,WREG
0097 0F1E        M                addwf     ACCdLO, F         ; addwf lsb
0098 6A19        M                movfp     ACCaHI,WREG
0099 111F        M                addwfc    ACCdHI, F         ; addwf msb
009A             M NoAdd4
009A 191F        M                rrcf      ACCdHI, F
009B 191E        M                rrcf      ACCdLO, F
009C 191D        M                rrcf      ACCcHI, F
009D 191C        M                rrcf      ACCcLO, F
009E 8804        M                bcf       ALUSTA,C
  0005           M                variable  i = i+1
                 M ;
                 M             .if i < 8
```

```
009F 951A         M                btfss     ACCbLO,i        ; test low byte
                  M              .else
                  M                btfss     ACCbHI,i-8      ; test high byte
                  M              .fi
00A0 C0A5         M                goto      NoAdd5          ; LSB is 0, so no need to addwf
00A1 6A18         M                movfp     ACCaLO,WREG
00A2 0F1E         M                addwf     ACCdLO, F       ; addwf lsb
00A3 6A19         M                movfp     ACCaHI,WREG
00A4 111F         M                addwfc    ACCdHI, F       ; addwf msb
00A5              M NoAdd5
00A5 191F         M                rrcf      ACCdHI, F
00A6 191E         M                rrcf      ACCdLO, F
00A7 191D         M                rrcf      ACCcHI, F
00A8 191C         M                rrcf      ACCcLO, F
00A9 8804         M                bcf       ALUSTA,C
  0006            M                variable  i = i+1
                  M ;
                  M              .if i < 8
00AA 961A         M                btfss     ACCbLO,i        ; test low byte
                  M              .else
                  M                btfss     ACCbHI,i-8      ; test high byte
                  M              .fi
00AB C0B0         M                goto      NoAdd6          ; LSB is 0, so no need to addwf
00AC 6A18         M                movfp     ACCaLO,WREG
00AD 0F1E         M                addwf     ACCdLO, F       ; addwf lsb
00AE 6A19         M                movfp     ACCaHI,WREG
00AF 111F         M                addwfc    ACCdHI, F       ; addwf msb
00B0              M NoAdd6
00B0 191F         M                rrcf      ACCdHI, F
00B1 191E         M                rrcf      ACCdLO, F
00B2 191D         M                rrcf      ACCcHI, F
00B3 191C         M                rrcf      ACCcLO, F
00B4 8804         M                bcf       ALUSTA,C
  0007            M                variable  i = i+1
                  M ;
                  M              .if i < 8
00B5 971A         M                btfss     ACCbLO,i        ; test low byte
                  M              .else
                  M                btfss     ACCbHI,i-8      ; test high byte
                  M              .fi
00B6 C0BB         M                goto      NoAdd7          ; LSB is 0, so no need to addwf
00B7 6A18         M                movfp     ACCaLO,WREG
00B8 0F1E         M                addwf     ACCdLO, F       ; addwf lsb
00B9 6A19         M                movfp     ACCaHI,WREG
00BA 111F         M                addwfc    ACCdHI, F       ; addwf msb
00BB              M NoAdd7
00BB 191F         M                rrcf      ACCdHI, F
00BC 191E         M                rrcf      ACCdLO, F
00BD 191D         M                rrcf      ACCcHI, F
00BE 191C         M                rrcf      ACCcLO, F
00BF 8804         M                bcf       ALUSTA,C
  0008            M                variable  i = i+1
                  M ;
                  M              .if i < 8
                  M                btfss      ACCbLO,i        ; test low byte
                  M              .else
00C0 901B         M                btfss      ACCbHI,i-8      ; test high byte
                  M              .fi
00C1 C0C6         M                goto       NoAdd8          ; LSB is 0, so no need to addwf
00C2 6A18         M                movfp      ACCaLO,WREG
00C3 0F1E         M                addwf      ACCdLO, F       ; addwf lsb
00C4 6A19         M                movfp      ACCaHI,WREG
00C5 111F         M                addwfc     ACCdHI, F       ; addwf msb
00C6              M NoAdd8
00C6 191F         M                rrcf       ACCdHI, F
00C7 191E         M                rrcf       ACCdLO, F
```

```
00C8 191D        M                rrcf       ACCcHI, F
00C9 191C        M                rrcf       ACCcLO, F
00CA 8804        M                bcf        ALUSTA,C
  0009           M                variable   i = i+1
                 M ;
                 M             .if i < 8
                 M                btfss      ACCbLO,i        ; test low byte
                 M             .else
00CB 911B        M                btfss      ACCbHI,i-8      ; test high byte
                 M             .fi
00CC C0D1        M                goto       NoAdd9          ; LSB is 0, so no need to addwf
00CD 6A18        M                movfp      ACCaLO,WREG
00CE 0F1E        M                addwf      ACCdLO, F       ; addwf lsb
00CF 6A19        M                movfp      ACCaHI,WREG
00D0 111F        M                addwfc     ACCdHI, F       ; addwf msb
00D1             M NoAdd9
00D1 191F        M                rrcf       ACCdHI, F
00D2 191E        M                rrcf       ACCdLO, F
00D3 191D        M                rrcf       ACCcHI, F
00D4 191C        M                rrcf       ACCcLO, F
00D5 8804        M                bcf        ALUSTA,C
  000A           M                variable   i = i+1
                 M ;
                 M             .if i < 8
                 M                btfss      ACCbLO,i        ; test low byte
                 M             .else
00D6 921B        M                btfss      ACCbHI,i-8      ; test high byte
                 M             .fi
00D7 C0DC        M                goto       NoAdd10         ; LSB is 0, so no need to addwf
00D8 6A18        M                movfp      ACCaLO,WREG
00D9 0F1E        M                addwf      ACCdLO, F       ; addwf lsb
00DA 6A19        M                movfp      ACCaHI,WREG
00DB 111F        M                addwfc     ACCdHI, F       ; addwf msb
00DC             M NoAdd10
00DC 191F        M                rrcf       ACCdHI, F
00DD 191E        M                rrcf       ACCdLO, F
00DE 191D        M                rrcf       ACCcHI, F
00DF 191C        M                rrcf       ACCcLO, F
00E0 8804        M                bcf        ALUSTA,C
  000B           M                variable   i = i+1
                 M ;
                 M             .if i < 8
                 M                btfss      ACCbLO,i        ; test low byte
                 M             .else
00E1 931B        M                btfss      ACCbHI,i-8      ; test high byte
                 M             .fi
00E2 C0E7        M                goto       NoAdd11         ; LSB is 0, so no need to addwf
00E3 6A18        M                movfp      ACCaLO,WREG
00E4 0F1E        M                addwf      ACCdLO, F       ; addwf lsb
00E5 6A19        M                movfp      ACCaHI,WREG
00E6 111F        M                addwfc     ACCdHI, F       ; addwf msb
00E7             M NoAdd11
00E7 191F        M                rrcf       ACCdHI, F
00E8 191E        M                rrcf       ACCdLO, F
00E9 191D        M                rrcf       ACCcHI, F
00EA 191C        M                rrcf       ACCcLO, F
00EB 8804        M                bcf        ALUSTA,C
  000C           M                variable   i = i+1
                 M ;
                 M             .if i < 8
                 M                btfss      ACCbLO,i        ; test low byte
                 M             .else
00EC 941B        M                btfss      ACCbHI,i-8      ; test high byte
                 M             .fi
00ED C0F2        M                goto       NoAdd12         ; LSB is 0, so no need to addwf
00EE 6A18        M                movfp      ACCaLO,WREG
```

```
00EF 0F1E        M                addwf     ACCdLO, F           ;addwf lsb
00F0 6A19        M                movfp     ACCaHI,WREG
00F1 111F        M                addwfc    ACCdHI, F           ;addwf msb
00F2             M NoAdd12
00F2 191F        M                rrcf      ACCdHI, F
00F3 191E        M                rrcf      ACCdLO, F
00F4 191D        M                rrcf      ACCcHI, F
00F5 191C        M                rrcf      ACCcLO, F
00F6 8804        M                bcf       ALUSTA,C
  000D           M                variable  i = i+1
                 M ;
                 M             .if i < 8
                 M                btfss     ACCbLO,i        ; test low byte
                 M             .else
00F7 951B        M                btfss     ACCbHI,i-8      ; test high byte
                 M             .fi
00F8 C0FD        M                goto      NoAdd13         ; LSB is 0, so no need to addwf
00F9 6A18        M                movfp     ACCaLO,WREG
00FA 0F1E        M                addwf     ACCdLO, F       ; addwf lsb
00FB 6A19        M                movfp     ACCaHI,WREG
00FC 111F        M                addwfc    ACCdHI, F       ; addwf msb
00FD             M NoAdd13
00FD 191F        M                rrcf      ACCdHI, F
00FE 191E        M                rrcf      ACCdLO, F
00FF 191D        M                rrcf      ACCcHI, F
0100 191C        M                rrcf      ACCcLO, F
0101 8804        M                bcf       ALUSTA,C
  000E           M                variable  i = i+1
                 M ;
                 M             .if i < 8
                 M                btfss     ACCbLO,i        ; test low byte
                 M             .else
0102 961B        M                btfss     ACCbHI,i-8      ; test high byte
                 M             .fi
0103 C108        M                goto      NoAdd14         ; LSB is 0, so no need to addwf
0104 6A18        M                movfp     ACCaLO,WREG
0105 0F1E        M                addwf     ACCdLO, F       ; addwf lsb
0106 6A19        M                movfp     ACCaHI,WREG
0107 111F        M                addwfc    ACCdHI, F       ; addwf msb
0108             M NoAdd14
0108 191F        M                rrcf      ACCdHI, F
0109 191E        M                rrcf      ACCdLO, F
010A 191D        M                rrcf      ACCcHI, F
010B 191C        M                rrcf      ACCcLO, F
010C 8804        M                bcf       ALUSTA,C
  000F           M                variable  i = i+1
                 M ;
                 M             .if i < 8
                 M                btfss     ACCbLO,i        ; test low byte
                 M             .else
010D 971B        M                btfss     ACCbHI,i-8      ; test high byte
                 M             .fi
010E C113        M                goto      NoAdd15         ; LSB is 0, so no need to addwf
010F 6A18        M                movfp     ACCaLO,WREG
0110 0F1E        M                addwf     ACCdLO, F       ; addwf lsb
0111 6A19        M                movfp     ACCaHI,WREG
0112 111F        M                addwfc    ACCdHI, F        ; addwf msb
0113             M NoAdd15
0113 191F        M                rrcf      ACCdHI, F
0114 191E        M                rrcf      ACCdLO, F
0115 191D        M                rrcf      ACCcHI, F
0116 191C        M                rrcf      ACCcLO, F
0117 8804        M                bcf       ALUSTA,C
  0010           M                variable  i = i+1
                 M          .endw
                 M         #if SIGNED
```

```
                M                rrcf       ACCdHI, F
                M                rrcf       ACCdLO, F
                M                rrcf       ACCcHI, F
                M                rrcf       ACCcLO, F
                M                bcf        ALUSTA,C
                M         #endif
                M ;
              00543 ;
              00544     #if SIGNED
              00545         btfss   sign,MSB        ; negate (ACCc,ACCd)
              00546         return
              00547         NegMac32 ACCcHI,ACCcLO,ACCdHI, ACCdLO
              00548         return
              00549     #else
0118 0002     00550         return
              00551     #endif
              00552 ;
              00553         PAGE
              00554 ;*****************************************************************
              00555 ;                       Double Precision Division
              00556 ;
              00557 ;               ( Optimized for Code : Looped Code )
              00558 ;
              00559 ;*****************************************************************;
              00560 ;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
              00561 ;                                              Remainder in ACCc (16 bits)
              00562 ;      (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
              00563 ;      (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
              00564 ;      (c) CALL D_div
              00565 ;      (d) The 16 bit result is in location ACCbHI & ACCbLO
              00566 ;      (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
              00567 ;
              00568 ;   Performance :
              00569 ;            Program Memory     :   31 (UNSIGNED)
              00570 ;                                   39 (SIGNED)
              00571 ;               Clock Cycles    :   300 (UNSIGNED : excluding CALL & RETURN)
              00572 ;                                   312 (SIGNED : excluding CALL & RETURN)
              00573 ;
              00574 ;       NOTE :
              00575 ;               The performance specs are for Unsigned arithmetic ( i.e,
              00576 ;               with “SIGNED equ  FALSE “).
              00577 ;
              00578 ;*****************************************************************
              00579 ;       Double Precision Divide ( 16/16 -> 16 )
              00580 ;
              00581 ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
              00582 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc (ACCcHI,ACCcLO).
              00583 ;
              00584 ;               B/A = (Q) + (R)/A
              00585 ;       or      B = A*Q + R
              00586 ;
              00587 ;                       where   B :  Numerator
              00588 ;                               A :  Denominator
              00589 ;                               Q :  Quotiont (Integer Result)
              00590 ;                               R :  Remainder
              00591 ;
              00592 ;       Note :  Check for ZERO Denominator or Numerator is not performed
              00593 ;               A ZERO Denominator will produce incorrect results
              00594 ;
              00595 ;       SIGNED Arithmetic :
              00596 ;                       In case of signed arithmetic, if either
              00597 ;  numerator or denominator is negative, then both Q & R are
              00598 ;  represented as negative numbers
              00599 ;               -(B/A) = -(Q) + (-R)/A
              00600 ;       or      -B = (-Q)*A + (-R)
              00601 ;
```

```
                00602 ;*****************************************************************
                00603 ;
0119            00604 D_divS
                00605 ;
0119 8404       00606         bsf      ALUSTA,FS0
011A 8504       00607         bsf      ALUSTA,FS1            ; set no auto-incrment for fsr0
                00608
                00609      #if    SIGNED
                00610         CALL    S_SIGN
                00611      #endif
                00612 ;
011B 2922       00613         clrf     count, F
011C 8422       00614         bsf      count,4               ; set count = 16
011D 291D       00615         clrf     ACCcHI, F
011E 291C       00616         clrf     ACCcLO, F
011F 291E       00617         clrf     ACCdLO, F
0120 291F       00618         clrf     ACCdHI, F
                00619 ;
                00620 ;  Looped code
                00621 ;
0121            00622 divLoop
0121 8804       00623         bcf      ALUSTA,C
0122 1B1A       00624         rlcf     ACCbLO, F
0123 1B1B       00625         rlcf     ACCbHI, F
0124 1B1C       00626         rlcf     ACCcLO, F
0125 1B1D       00627         rlcf     ACCcHI, F
0126 6A19       00628         movfp    ACCaHI,WREG
0127 041D       00629         subwf    ACCcHI,W               ; check if a>c
0128 9204       00630         btfss    ALUSTA,Z
0129 C12C       00631         goto     notz
012A 6A18       00632         movfp    ACCaLO,WREG
012B 041C       00633         subwf    ACCcLO,W               ; if msb equal then check lsb
012C            00634 notz
012C 9004       00635         btfss    ALUSTA,C               ; carry set if c>a
012D C133       00636         goto     nosub                  ; if c < a
012E            00637 subca
012E 6A18       00638         movfp    ACCaLO,WREG            ; c-a into c
012F 051C       00639         subwf    ACCcLO, F
0130 6A19       00640         movfp    ACCaHI,WREG
0131 031D       00641         subwfb   ACCcHI, F
0132 8004       00642         bsf      ALUSTA,C               ; shift a 1 into d (result)
0133            00643 nosub
0133 1B1E       00644         rlcf     ACCdLO, F
0134 1B1F       00645         rlcf     ACCdHI, F
0135 1722       00646         decfsz   count, F
0136 C121       00647         goto     divLoop
                00648
                00649 ;
                00650     #if SIGNED
                00651         btfss    sign,MSB
                00652         return
                00653         movlw    ACCcLO
                00654         movwf    fsr0
                00655         call     negate
                00656         movlw    ACCdLO
                00657         movwf    fsr0
                00658         call     negate
                00659         return
                00660     #else
0137 0002       00661         return
                00662     #endif
                00663 ;
                00664         PAGE
                00665 ;*****************************************************************
                00666 ;                       Double Precision Division
                00667 ;
```

```
00668 ;               ( Optimized for Speed : straight Line Code )
00669 ;
00670 ;*****************************************************************;
00671 ;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
00672 ;                                              Remainder in ACCc (16 bits)
00673 ;
00674 ;      (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
00675 ;      (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
00676 ;      (c) CALL D_div
00677 ;      (d) The 16 bit result is in location ACCbHI & ACCbLO
00678 ;      (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
00679 ;
00680 ;               B/A = (Q) + (R)/A
00681 ;       or      B = A*Q + R
00682 ;
00683 ;                       where   B :  Numerator
00684 ;                               A :  Denominator
00685 ;                               Q :  Quotiont (Integer Result)
00686 ;                               R :  Remainder
00687 ;
00688 ;       Note :  Check for ZERO Denominator or Numerator is not performed
00689 ;               A ZERO Denominator will produce incorrect results
00690 ;
00691 ;       SIGNED Arithmetic :
00692 ;                       In case of signed arithmetic, if either
00693 ;  numerator or denominator is negative, then both Q & R are
00694 ;  represented as negative numbers
00695 ;               -(B/A) = -(Q) + (-R)/A
00696 ;       or      -B = (-Q)*A + (-R)
00697 ;
00698 ;   Performance :
00699 ;            Program Memory     :   325 (UNSIGNED)
00700 ;                                   354 (SIGNED)
00701 ;               Clock Cycles    :   250 (UNSIGNED : excluding CALL & RETURN)
00702 ;                               :   260 (SIGNED : excluding CALL & RETURN)
00703 ;
00704 ;*****************************************************************;
00705 ;       division macro
00706 ;
00707 divMac  MACRO
00708         variable i
00709
00710        variable i = 0
00711       .while i < 16
00712 ;
00713                 bcf      ALUSTA,C
00714                 rlcf     ACCbLO, F
00715                 rlcf     ACCbHI, F
00716                 rlcf     ACCcLO, F
00717                 rlcf     ACCcHI, F
00718                 movfp    ACCaHI,WREG
00719                 subwf    ACCcHI,W        ; check if a>c
00720                 btfss    ALUSTA,Z
00721                 goto     notz#v(i)
00722                 movfp    ACCaLO,WREG
00723                 subwf    ACCcLO,W        ; if msb equal then check lsb
00724 notz#v(i)       btfss    ALUSTA,C        ; carry set if c>a
00725                 goto     nosub#v(i)      ; if c < a
00726 subca#v(i)      movfp    ACCaLO,WREG     ; c-a into c
00727                 subwf    ACCcLO, F
00728                 movfp    ACCaHI,WREG
00729                 subwfb   ACCcHI, F
00730                 bsf      ALUSTA,C        ; shift a 1 into d (result)
00731 nosub#v(i)      rlcf     ACCdLO, F
00732                 rlcf     ACCdHI, F
00733        variable i=i+1
```

```
                00734       .endw
                00735 ;
                00736         ENDM
                00737 ;
                00738         PAGE
                00739 ;*****************************************************************
                00740 ;       Double Precision Divide ( 16/16 -> 16 )
                00741 ;
                00742 ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
               00743 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc (ACCcHI,ACCcLO).
                00744 ;
                00745 ;   NOTE  :  Before calling this routine, the user should make sure that
                00746 ;            the Numerator(ACCb) is greater than Denominator(ACCa). If
                00747 ;            the case is not true, the user should scale either Numerator
                00748 ;            or Denominator or both such that Numerator is greater than
                00749 ;            the Denominator.
                00750 ;
                00751 ;
                00752 ;*****************************************************************
                00753 ;
0138            00754 D_divF
                00755 ;
                00756      #if   SIGNED
                00757         movfp     ACCaHI,WREG
                00758         xorwf     ACCbHI,W
                00759         movwf     sign
                00760         btfss     ACCbHI,MSB         ; if MSB set go & negate ACCb
                00761         goto      chek_A_MSB_DIV
                00762 ;
                00763         negMac    ACCbLO,ACCbHI
                00764 ;
                00765 chek_A_MSB_DIV
                00766         btfss     ACCaHI,MSB         ; if MSB set go & negate ACCa
                00767         goto      continue_DIV
                00768         negMac    ACCaLO,ACCaHI
                00769
                00770      #endif
                00771 ;
0138            00772 continue_DIV
0138 291D       00773         clrf      ACCcHI, F
0139 291C       00774         clrf      ACCcLO, F
013A 291E       00775         clrf      ACCdLO, F
013B 291F       00776         clrf      ACCdHI, F
                00777
                00778 ;
                00779 ; straight line code : using the macro divMac
                00780 ;
                00781         divMac
  0000            M         variable i
                  M
  0000            M        variable i = 0
                  M       .while i < 16
                  M ;
013C 8804         M                 bcf      ALUSTA,C
013D 1B1A         M                 rlcf     ACCbLO, F
013E 1B1B         M                 rlcf     ACCbHI, F
013F 1B1C         M                 rlcf     ACCcLO, F
0140 1B1D         M                 rlcf     ACCcHI, F
0141 6A19         M                 movfp    ACCaHI,WREG
0142 041D         M                 subwf    ACCcHI,W          ; check if a>c
0143 9204         M                 btfss    ALUSTA,Z
0144 C147         M                 goto     notz0
0145 6A18         M                 movfp    ACCaLO,WREG
0146 041C         M                 subwf    ACCcLO,W          ; if msb equal then check lsb
0147 9004         M notz0           btfss    ALUSTA,C          ; carry set if c>a
0148 C14E         M                 goto     nosub0            ; if c < a
```

```
0149 6A18         M subca0          movfp     ACCaLO,WREG         ; c-a into c
014A 051C         M                 subwf     ACCcLO, F
014B 6A19         M                 movfp     ACCaHI,WREG
014C 031D         M                 subwfb    ACCcHI, F
014D 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
014E 1B1E         M nosub0          rlcf      ACCdLO, F
014F 1B1F         M                 rlcf      ACCdHI, F
  0001            M        variable i=i+1
                  M ;
0150 8804         M                 bcf       ALUSTA,C
0151 1B1A         M                 rlcf      ACCbLO, F
0152 1B1B         M                 rlcf      ACCbHI, F
0153 1B1C         M                 rlcf      ACCcLO, F
0154 1B1D         M                 rlcf      ACCcHI, F
0155 6A19         M                 movfp     ACCaHI,WREG
0156 041D         M                 subwf     ACCcHI,W            ; check if a>c
0157 9204         M                 btfss     ALUSTA,Z
0158 C15B         M                 goto      notz1
0159 6A18         M                 movfp     ACCaLO,WREG
015A 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
015B 9004         M notz1           btfss     ALUSTA,C            ; carry set if c>a
015C C162         M                 goto      nosub1              ; if c < a
015D 6A18         M subca1          movfp     ACCaLO,WREG         ; c-a into c
015E 051C         M                 subwf     ACCcLO, F
015F 6A19         M                 movfp     ACCaHI,WREG
0160 031D         M                 subwfb    ACCcHI, F
0161 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
0162 1B1E         M nosub1          rlcf      ACCdLO, F
0163 1B1F         M                 rlcf      ACCdHI, F
  0002            M        variable i=i+1
                  M ;
0164 8804         M                 bcf       ALUSTA,C
0165 1B1A         M                 rlcf      ACCbLO, F
0166 1B1B         M                 rlcf      ACCbHI, F
0167 1B1C         M                 rlcf      ACCcLO, F
0168 1B1D         M                 rlcf      ACCcHI, F
0169 6A19         M                 movfp     ACCaHI,WREG
016A 041D         M                 subwf     ACCcHI,W            ; check if a>c
016B 9204         M                 btfss     ALUSTA,Z
016C C16F         M                 goto      notz2
016D 6A18         M                 movfp     ACCaLO,WREG
016E 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
016F 9004         M notz2           btfss     ALUSTA,C            ; carry set if c>a
0170 C176         M                 goto      nosub2              ; if c < a
0171 6A18         M subca2          movfp     ACCaLO,WREG         ; c-a into c
0172 051C         M                 subwf     ACCcLO, F
0173 6A19         M                 movfp     ACCaHI,WREG
0174 031D         M                 subwfb    ACCcHI, F
0175 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
0176 1B1E         M nosub2          rlcf      ACCdLO, F
0177 1B1F         M                 rlcf      ACCdHI, F
  0003            M        variable i=i+1
                  M ;
0178 8804         M                 bcf       ALUSTA,C
0179 1B1A         M                 rlcf      ACCbLO, F
017A 1B1B         M                 rlcf      ACCbHI, F
017B 1B1C         M                 rlcf      ACCcLO, F
017C 1B1D         M                 rlcf      ACCcHI, F
017D 6A19         M                 movfp     ACCaHI,WREG
017E 041D         M                 subwf     ACCcHI,W            ; check if a>c
017F 9204         M                 btfss     ALUSTA,Z
0180 C183         M                 goto      notz3
0181 6A18         M                 movfp     ACCaLO,WREG
0182 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
0183 9004         M notz3           btfss     ALUSTA,C            ; carry set if c>a
0184 C18A         M                 goto      nosub3              ; if c < a
```

```
0185 6A18         M subca3          movfp     ACCaLO,WREG         ; c-a into c
0186 051C         M                 subwf     ACCcLO, F
0187 6A19         M                 movfp     ACCaHI,WREG
0188 031D         M                 subwfb    ACCcHI, F
0189 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
018A 1B1E         M nosub3          rlcf      ACCdLO, F
018B 1B1F         M                 rlcf      ACCdHI, F
  0004            M        variable i=i+1
                  M ;
018C 8804         M                 bcf       ALUSTA,C
018D 1B1A         M                 rlcf      ACCbLO, F
018E 1B1B         M                 rlcf      ACCbHI, F
018F 1B1C         M                 rlcf      ACCcLO, F
0190 1B1D         M                 rlcf      ACCcHI, F
0191 6A19         M                 movfp     ACCaHI,WREG
0192 041D         M                 subwf     ACCcHI,W            ; check if a>c
0193 9204         M                 btfss     ALUSTA,Z
0194 C197         M                 goto      notz4
0195 6A18         M                 movfp     ACCaLO,WREG
0196 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
0197 9004         M notz4           btfss     ALUSTA,C            ; carry set if c>a
0198 C19E         M                 goto      nosub4              ; if c < a
0199 6A18         M subca4          movfp     ACCaLO,WREG         ; c-a into c
019A 051C         M                 subwf     ACCcLO, F
019B 6A19         M                 movfp     ACCaHI,WREG
019C 031D         M                 subwfb    ACCcHI, F
019D 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
019E 1B1E         M nosub4          rlcf      ACCdLO, F
019F 1B1F         M                 rlcf      ACCdHI, F
  0005            M        variable i=i+1
                  M ;
01A0 8804         M                 bcf       ALUSTA,C
01A1 1B1A         M                 rlcf      ACCbLO, F
01A2 1B1B         M                 rlcf      ACCbHI, F
01A3 1B1C         M                 rlcf      ACCcLO, F
01A4 1B1D         M                 rlcf      ACCcHI, F
01A5 6A19         M                 movfp     ACCaHI,WREG
01A6 041D         M                 subwf     ACCcHI,W            ; check if a>c
01A7 9204         M                 btfss     ALUSTA,Z
01A8 C1AB         M                 goto      notz5
01A9 6A18         M                 movfp     ACCaLO,WREG
01AA 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01AB 9004         M notz5           btfss     ALUSTA,C            ; carry set if c>a
01AC C1B2         M                 goto      nosub5              ; if c < a
01AD 6A18         M subca5          movfp     ACCaLO,WREG         ; c-a into c
01AE 051C         M                 subwf     ACCcLO, F
01AF 6A19         M                 movfp     ACCaHI,WREG
01B0 031D         M                 subwfb    ACCcHI, F
01B1 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
01B2 1B1E         M nosub5          rlcf      ACCdLO, F
01B3 1B1F         M                 rlcf      ACCdHI, F
  0006            M        variable i=i+1
                  M ;
01B4 8804         M                 bcf       ALUSTA,C
01B5 1B1A         M                 rlcf      ACCbLO, F
01B6 1B1B         M                 rlcf      ACCbHI, F
01B7 1B1C         M                 rlcf      ACCcLO, F
01B8 1B1D         M                 rlcf      ACCcHI, F
01B9 6A19         M                 movfp     ACCaHI,WREG
01BA 041D         M                 subwf     ACCcHI,W            ; check if a>c
01BB 9204         M                 btfss     ALUSTA,Z
01BC C1BF         M                 goto      notz6
01BD 6A18         M                 movfp     ACCaLO,WREG
01BE 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01BF 9004         M notz6           btfss     ALUSTA,C            ; carry set if c>a
01C0 C1C6         M                 goto      nosub6              ; if c < a
```

```
01C1 6A18        M subca6          movfp     ACCaLO,WREG         ; c-a into c
01C2 051C        M                 subwf     ACCcLO, F
01C3 6A19        M                 movfp     ACCaHI,WREG
01C4 031D        M                 subwfb    ACCcHI, F
01C5 8004        M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
01C6 1B1E        M nosub6          rlcf      ACCdLO, F
01C7 1B1F        M                 rlcf      ACCdHI, F
  0007           M        variable i=i+1
                 M ;
01C8 8804        M                 bcf       ALUSTA,C
01C9 1B1A        M                 rlcf      ACCbLO, F
01CA 1B1B        M                 rlcf      ACCbHI, F
01CB 1B1C        M                 rlcf      ACCcLO, F
01CC 1B1D        M                 rlcf      ACCcHI, F
01CD 6A19        M                 movfp     ACCaHI,WREG
01CE 041D        M                 subwf     ACCcHI,W            ; check if a>c
01CF 9204        M                 btfss     ALUSTA,Z
01D0 C1D3        M                 goto      notz7
01D1 6A18        M                 movfp     ACCaLO,WREG
01D2 041C        M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01D3 9004        M notz7           btfss     ALUSTA,C            ; carry set if c>a
01D4 C1DA        M                 goto      nosub7              ; if c < a
01D5 6A18        M subca7          movfp     ACCaLO,WREG         ; c-a into c
01D6 051C        M                 subwf     ACCcLO, F
01D7 6A19        M                 movfp     ACCaHI,WREG
01D8 031D        M                 subwfb    ACCcHI, F
01D9 8004        M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
01DA 1B1E        M nosub7          rlcf      ACCdLO, F
01DB 1B1F        M                 rlcf      ACCdHI, F
  0008           M        variable i=i+1
                 M ;
01DC 8804        M                 bcf       ALUSTA,C
01DD 1B1A        M                 rlcf      ACCbLO, F
01DE 1B1B        M                 rlcf      ACCbHI, F
01DF 1B1C        M                 rlcf      ACCcLO, F
01E0 1B1D        M                 rlcf      ACCcHI, F
01E1 6A19        M                 movfp     ACCaHI,WREG
01E2 041D        M                 subwf     ACCcHI,W            ; check if a>c
01E3 9204        M                 btfss     ALUSTA,Z
01E4 C1E7        M                 goto      notz8
01E5 6A18        M                 movfp     ACCaLO,WREG
01E6 041C        M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01E7 9004        M notz8           btfss     ALUSTA,C            ; carry set if c>a
01E8 C1EE        M                 goto      nosub8              ; if c < a
01E9 6A18        M subca8          movfp     ACCaLO,WREG         ; c-a into c
01EA 051C        M                 subwf     ACCcLO, F
01EB 6A19        M                 movfp     ACCaHI,WREG
01EC 031D        M                 subwfb    ACCcHI, F
01ED 8004        M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
01EE 1B1E        M nosub8          rlcf      ACCdLO, F
01EF 1B1F        M                 rlcf      ACCdHI, F
  0009           M        variable i=i+1
                 M ;
01F0 8804        M                 bcf       ALUSTA,C
01F1 1B1A        M                 rlcf      ACCbLO, F
01F2 1B1B        M                 rlcf      ACCbHI, F
01F3 1B1C        M                 rlcf      ACCcLO, F
01F4 1B1D        M                 rlcf      ACCcHI, F
01F5 6A19        M                 movfp     ACCaHI,WREG
01F6 041D        M                 subwf     ACCcHI,W            ; check if a>c
01F7 9204        M                 btfss     ALUSTA,Z
01F8 C1FB        M                 goto      notz9
01F9 6A18        M                 movfp     ACCaLO,WREG
01FA 041C        M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01FB 9004        M notz9           btfss     ALUSTA,C            ; carry set if c>a
01FC C202        M                 goto      nosub9              ; if c < a
```

```
01FD 6A18        M subca9          movfp     ACCaLO,WREG         ; c-a into c
01FE 051C        M                 subwf     ACCcLO, F
01FF 6A19        M                 movfp     ACCaHI,WREG
0200 031D        M                 subwfb    ACCcHI, F
0201 8004        M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
0202 1B1E        M nosub9          rlcf      ACCdLO, F
0203 1B1F        M                 rlcf      ACCdHI, F
  000A           M        variable i=i+1
                 M ;
0204 8804        M                 bcf       ALUSTA,C
0205 1B1A        M                 rlcf      ACCbLO, F
0206 1B1B        M                 rlcf      ACCbHI, F
0207 1B1C        M                 rlcf      ACCcLO, F
0208 1B1D        M                 rlcf      ACCcHI, F
0209 6A19        M                 movfp     ACCaHI,WREG
020A 041D        M                 subwf     ACCcHI,W            ; check if a>c
020B 9204        M                 btfss     ALUSTA,Z
020C C20F        M                 goto      notz10
020D 6A18        M                 movfp     ACCaLO,WREG
020E 041C        M                 subwf     ACCcLO,W            ; if msb equal then check lsb
020F 9004        M notz10          btfss     ALUSTA,C            ; carry set if c>a
0210 C216        M                 goto      nosub10             ; if c < a
0211 6A18        M subca10         movfp     ACCaLO,WREG         ; c-a into c
0212 051C        M                 subwf     ACCcLO, F
0213 6A19        M                 movfp     ACCaHI,WREG
0214 031D        M                 subwfb    ACCcHI, F
0215 8004        M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
0216 1B1E        M nosub10         rlcf      ACCdLO, F
0217 1B1F        M                 rlcf      ACCdHI, F
  000B           M        variable i=i+1
                 M ;
0218 8804        M                 bcf       ALUSTA,C
0219 1B1A        M                 rlcf      ACCbLO, F
021A 1B1B        M                 rlcf      ACCbHI, F
021B 1B1C        M                 rlcf      ACCcLO, F
021C 1B1D        M                 rlcf      ACCcHI, F
021D 6A19        M                 movfp     ACCaHI,WREG
021E 041D        M                 subwf     ACCcHI,W            ; check if a>c
021F 9204        M                 btfss     ALUSTA,Z
0220 C223        M                 goto      notz11
0221 6A18        M                 movfp     ACCaLO,WREG
0222 041C        M                 subwf     ACCcLO,W            ; if msb equal then check lsb
0223 9004        M notz11          btfss     ALUSTA,C            ; carry set if c>a
0224 C22A        M                 goto      nosub11             ; if c < a
0225 6A18        M subca11         movfp     ACCaLO,WREG         ; c-a into c
0226 051C        M                 subwf     ACCcLO, F
0227 6A19        M                 movfp     ACCaHI,WREG
0228 031D        M                 subwfb    ACCcHI, F
0229 8004        M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
022A 1B1E        M nosub11         rlcf      ACCdLO, F
022B 1B1F        M                 rlcf      ACCdHI, F
  000C           M        variable i=i+1
                 M ;
022C 8804        M                 bcf       ALUSTA,C
022D 1B1A        M                 rlcf      ACCbLO, F
022E 1B1B        M                 rlcf      ACCbHI, F
022F 1B1C        M                 rlcf      ACCcLO, F
0230 1B1D        M                 rlcf      ACCcHI, F
0231 6A19        M                 movfp     ACCaHI,WREG
0232 041D        M                 subwf     ACCcHI,W            ; check if a>c
0233 9204        M                 btfss     ALUSTA,Z
0234 C237        M                 goto      notz12
0235 6A18        M                 movfp     ACCaLO,WREG
0236 041C        M                 subwf     ACCcLO,W            ; if msb equal then check lsb
0237 9004        M notz12          btfss     ALUSTA,C            ; carry set if c>a
0238 C23E        M                 goto      nosub12             ; if c < a
```

```
0239 6A18         M subca12         movfp     ACCaLO,WREG         ; c-a into c
023A 051C         M                 subwf     ACCcLO, F
023B 6A19         M                 movfp     ACCaHI,WREG
023C 031D         M                 subwfb    ACCcHI, F
023D 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
023E 1B1E         M nosub12         rlcf      ACCdLO, F
023F 1B1F         M                 rlcf      ACCdHI, F
  000D            M         variable i=i+1
                  M ;
0240 8804         M                 bcf       ALUSTA,C
0241 1B1A         M                 rlcf      ACCbLO, F
0242 1B1B         M                 rlcf      ACCbHI, F
0243 1B1C         M                 rlcf      ACCcLO, F
0244 1B1D         M                 rlcf      ACCcHI, F
0245 6A19         M                 movfp     ACCaHI,WREG
0246 041D         M                 subwf     ACCcHI,W            ; check if a>c
0247 9204         M                 btfss     ALUSTA,Z
0248 C24B         M                 goto      notz13
0249 6A18         M                 movfp     ACCaLO,WREG
024A 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
024B 9004         M notz13          btfss     ALUSTA,C            ; carry set if c>a
024C C252         M                 goto      nosub13             ; if c < a
024D 6A18         M subca13         movfp     ACCaLO,WREG         ; c-a into c
024E 051C         M                 subwf     ACCcLO, F
024F 6A19         M                 movfp     ACCaHI,WREG
0250 031D         M                 subwfb    ACCcHI, F
0251 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
0252 1B1E         M nosub13         rlcf      ACCdLO, F
0253 1B1F         M                 rlcf      ACCdHI, F
  000E            M         variable i=i+1
                  M ;
0254 8804         M                 bcf       ALUSTA,C
0255 1B1A         M                 rlcf      ACCbLO, F
0256 1B1B         M                 rlcf      ACCbHI, F
0257 1B1C         M                 rlcf      ACCcLO, F
0258 1B1D         M                 rlcf      ACCcHI, F
0259 6A19         M                 movfp     ACCaHI,WREG
025A 041D         M                 subwf     ACCcHI,W            ; check if a>c
025B 9204         M                 btfss     ALUSTA,Z
025C C25F         M                 goto      notz14
025D 6A18         M                 movfp     ACCaLO,WREG
025E 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
025F 9004         M notz14          btfss     ALUSTA,C            ; carry set if c>a
0260 C266         M                 goto      nosub14             ; if c < a
0261 6A18         M subca14         movfp     ACCaLO,WREG         ; c-a into c
0262 051C         M                 subwf     ACCcLO, F
0263 6A19         M                 movfp     ACCaHI,WREG
0264 031D         M                 subwfb    ACCcHI, F
0265 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
0266 1B1E         M nosub14         rlcf      ACCdLO, F
0267 1B1F         M                 rlcf      ACCdHI, F
  000F            M         variable i=i+1
                  M ;
0268 8804         M                 bcf       ALUSTA,C
0269 1B1A         M                 rlcf      ACCbLO, F
026A 1B1B         M                 rlcf      ACCbHI, F
026B 1B1C         M                 rlcf      ACCcLO, F
026C 1B1D         M                 rlcf      ACCcHI, F
026D 6A19         M                 movfp     ACCaHI,WREG
026E 041D         M                 subwf     ACCcHI,W            ; check if a>c
026F 9204         M                 btfss     ALUSTA,Z
0270 C273         M                 goto      notz15
0271 6A18         M                 movfp     ACCaLO,WREG
0272 041C         M                 subwf     ACCcLO,W            ; if msb equal then check lsb
0273 9004         M notz15          btfss     ALUSTA,C            ; carry set if c>a
0274 C27A         M                 goto      nosub15             ; if c < a
```

```
0275 6A18         M subca15         movfp     ACCaLO,WREG         ; c-a into c
0276 051C         M                 subwf     ACCcLO, F
0277 6A19         M                 movfp     ACCaHI,WREG
0278 031D         M                 subwfb    ACCcHI, F
0279 8004         M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
027A 1B1E         M nosub15         rlcf      ACCdLO, F
027B 1B1F         M                 rlcf      ACCdHI, F
  0010            M        variable i=i+1
                  M       .endw
                  M ;
                00782 ;
                00783     #if SIGNED
                00784         btfss   sign,MSB                      ; negate (ACCc,ACCd)
                00785         return
                00786         negMac  ACCcLO,ACCcHI
                00787         negMac  ACCdLO,ACCdHI
                00788         return
                00789     #else
027C 0002       00790         return
                00791     #endif
                00792 ;
                00793         PAGE
                00794 ;*****************************************************************
                00795 ;
                00796 ;            Square Root By Newton Raphson Method
                00797 ;
                00798 ;    This routine computes the square root of a 16 bit number(with
                00799 ;  low byte in NumLo & high byte in NumHi ). After loading NumLo &
                00800 ;  NumHi with the desired number whose square root is to be computed,
                00801 ;  branch to location Sqrt ( by “GOTO  Sqrt” ). “ CALL  Sqrt” cannot
                00802 ;  be issued because the Sqrt function makes calls to Math routines
                00803 ;  and the stack is completely used up.
                00804 ;       The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
                00805 ;  The total number of iterations is set to ten. If more iterations
                00806 ;  are desired, change “LupCnt equ .10” to the desired value. Also,
                00807 ;  the initial guess value of the square root is given set as
                00808 ;  input/2 ( in subroutine “init” ). The user may modify this scheme
                00809 ;  if a better initial approximation value is known. A good initial
                00810 ;  guess will help the algorithm converge at a faster rate and thus
                00811 ;  less number of iterations required.
                00812 ;       Two utility math routines are used by this program : D_divS
                00813 ;  and D_add. These two routines are listed as seperate routines
                00814 ;  under double precision Division and double precision addition
                00815 ;  respectively.
                00816 ;
                00817 ;  Note : If square root of an 8 bit number is desired, it is probably
                00818 ;         better to have a table look scheme rather than using numerical
                00819 ;         methods.
                00820 ;               This method is computationally quite intensive and
                00821 ;         slow, but very accurate and the convergence rate is high
                00822 ;
                00823 ;   Performance :
                00824 ;               Program Memory  :   22 (excluding D_divS subroutine)
                00825 ;               Clock Cycles    :   3000 (approximately,with 10 iterations)
                00826 ;
                00827 ;       The #of cycles depends on Number of Iterations Selected.
                00828 ; In a lot of cases 5 or less iterations may be sufficient
                00829 ;
                00830 ;
                00831 ;*****************************************************************
                00832 ;               Newton-Raphson Method
                00833 ;*************************************************************
                00834
027D            00835 Sqrt
027D E28B       00836         call    SqrtInit            ; compute initial sqrt = Num/2
027E            00837 nextIter
```

```
                00838    #if MODE_FAST
027E 7A24       00839         movfp    NumLo,ACCbLO
027F 7B25       00840         movfp    NumHi,ACCbHI
                00841    #else
                00842         movfp    NumLo,WREG
                00843         movwf    ACCbLO
                00844         movfp    NumHi,WREG
                00845         movwf    ACCbHI
                00846    #endif
                00847 ;
0280 E119       00848         call     D_divS          ; double precision division
                00849                                 ; double precision addition
0281 6A1E       00850         movfp    ACCdLO,WREG     ; ACCd + ACCa -> ACCd
0282 0F18       00851         addwf    ACCaLO, F       ; addwf lsb
0283 6A1F       00852         movfp    ACCdHI,WREG
0284 1119       00853         addwfc   ACCaHI, F       ; addwf msb
                00854                                 ; now divide by 2
0285 8804       00855         bcf      ALUSTA,C
0286 1919       00856         rrcf     ACCaHI, F
0287 1918       00857         rrcf     ACCaLO, F
                00858 ;
0288 1726       00859         decfsz   iterCnt, F
0289 C27E       00860         goto     nextIter
028A 0002       00861         return                   ; End Sqrt
                00862 ;
028B            00863 SqrtInit
028B B00A       00864         movlw    _LUPCNT
028C 0126       00865         movwf    iterCnt         ; set number of iterations
                00866   #if MODE_FAST
028D 7925       00867         movfp    NumHi,ACCaHI
028E 7824       00868         movfp    NumLo,ACCaLO
                00869   #else
                00870         movfp    NumHi,WREG
                00871         movwf    ACCaHI
                00872         movfp    NumLo,WREG       ; set initial guess root = NUM/2
                00873         movwf    ACCaLO
                00874   #endif
028F 8804       00875         bcf      ALUSTA,C
0290 1919       00876         rrcf     ACCaHI, F
0291 1918       00877         rrcf     ACCaLO, F        ; set initial sqrt = Num/2
0292 0002       00878         return
                00879 ;
                00880         PAGE
                00881 ;*****************************************************************
                00882 ;                   8x8 Software Multiplier
                00883 ;               ( Fast Version : Straight Line Code )
                00884 ;
                00885 ;   The 16 bit result is stored in 2 bytes
                00886 ;
                00887 ; Before calling the subroutine " mpy ", the multiplier should
                00888 ; be loaded in location " mulplr ", and the multiplicand in
                00889 ; " mulcnd " . The 16 bit result is stored in locations
                00890 ; H_byte & L_byte.
                00891 ;
                00892 ;       Performance :
                00893 ;                    Program Memory  :  36 words
                00894 ;                    # of cycles     :  36 (excluding call & return)
                00895 ;                    Scratch RAM     :   0 locations
                00896 ;                    W Register      :  Used
                00897 ;
                00898 ;  This routine is optimized for speed efficiency ( straight line code )
                00899 ;  For code efficiency, refer to "mult8x8S.asm" ( looped code )
                00900 ;*****************************************************************
                00901 ;     Define a macro for adding & right shifting
                00902 ;
                00903 multiply    MACRO
```

```
                00904         variable  i ;
                00905           variable i = 0
                00906           .while i < 8
                00907                 btfsc     mulplr,i
                00908                 addwf    H_byte, F
                00909                 rrcf    H_byte, F
                00910                 rrcf    L_byte, F
                00911             variable i = i+1 ;
                00912           .endw
                00913         ENDM                    ; End of macro
                00914 ;
                00915 ;
0293            00916 mpy8x8_F
0293 291B       00917         clrf     H_byte, F
0294 291A       00918         clrf     L_byte, F
0295 6A18       00919         movfp    mulcnd,WREG     ; move the multiplicand to W reg.
0296 8804       00920         bcf      ALUSTA,C        ; Clear the carry bit in the status Reg.
                00921 ;
                00922         multiply
  0000              M         variable  i ;
  0000              M           variable i = 0
                    M           .while i < 8
0297 9819           M                 btfsc     mulplr,i
0298 0F1B           M                 addwf     H_byte, F
0299 191B           M                 rrcf      H_byte, F
029A 191A           M                 rrcf      L_byte, F
  0001              M             variable i = i+1 ;
029B 9919           M                 btfsc     mulplr,i
029C 0F1B           M                 addwf     H_byte, F
029D 191B           M                 rrcf      H_byte, F
029E 191A           M                 rrcf      L_byte, F
  0002              M             variable i = i+1 ;
029F 9A19           M                 btfsc     mulplr,i
02A0 0F1B           M                 addwf     H_byte, F
02A1 191B           M                 rrcf      H_byte, F
02A2 191A           M                 rrcf      L_byte, F
  0003              M             variable i = i+1 ;
02A3 9B19           M                 btfsc     mulplr,i
02A4 0F1B           M                 addwf     H_byte, F
02A5 191B           M                 rrcf      H_byte, F
02A6 191A           M                 rrcf      L_byte, F
  0004              M             variable i = i+1 ;
02A7 9C19           M                 btfsc     mulplr,i
02A8 0F1B           M                 addwf     H_byte, F
02A9 191B           M                 rrcf      H_byte, F
02AA 191A           M                 rrcf      L_byte, F
  0005              M             variable i = i+1 ;
02AB 9D19           M                 btfsc     mulplr,i
02AC 0F1B           M                 addwf     H_byte, F
02AD 191B           M                 rrcf      H_byte, F
02AE 191A           M                 rrcf      L_byte, F
  0006              M             variable i = i+1 ;
02AF 9E19           M                 btfsc     mulplr,i
02B0 0F1B           M                 addwf     H_byte, F
02B1 191B           M                 rrcf      H_byte, F
02B2 191A           M                 rrcf      L_byte, F
  0007              M             variable i = i+1 ;
02B3 9F19           M                 btfsc     mulplr,i
02B4 0F1B           M                 addwf     H_byte, F
02B5 191B           M                 rrcf      H_byte, F
02B6 191A           M                 rrcf      L_byte, F
  0008              M             variable i = i+1 ;
                    M           .endw
                00923 ;
02B7 0002       00924         return
                00925 ;
```

```
                00926         PAGE
                00927 ;*****************************************************************
                00928 ;                   8x8 Software Multiplier
                00929 ;                ( Code Efficient : Looped Code )
                00930 ;
                00931 ;   The 16 bit result is stored in 2 bytes
                00932 ;
                00933 ; Before calling the subroutine " mpy ", the multiplier should
                00934 ; be loaded in location " mulplr ", and the multiplicand in
                00935 ; " mulcnd " . The 16 bit result is stored in locations
                00936 ; H_byte & L_byte.
                00937 ;
                00938 ;       Performance :
                00939 ;                     Program Memory  :  13 words (excluding call & return)
                00940 ;                     # of cycles     :  69       (excluding call & return)
                00941 ;                     Scratch RAM     :  1 byte
                00942 ;                     W Register      :  Used
                00943 ;
                00944 ;  This routine is optimized for code efficiency ( looped code )
                00945 ;  For time efficiency code refer to "mult8x8F.asm" ( straight line code )
                00946 ;*****************************************************************
                00947 ;
02B8            00948 mpy8x8_S
02B8 291B       00949         clrf     H_byte, F
02B9 291A       00950         clrf     L_byte, F
02BA 2922       00951         clrf     count, F
02BB 8322       00952         bsf      count,3         ; set count = 8
02BC 6A18       00953         movfp    mulcnd,WREG
02BD 8804       00954         bcf      ALUSTA,C        ; Clear the carry bit in the status Reg.
02BE            00955 loop
02BE 9819       00956         btfsc     mulplr,0
02BF 0F1B       00957         addwf     H_byte, F
02C0 191B       00958         rrcf      H_byte, F
02C1 191A       00959         rrcf      L_byte, F
02C2 2119       00960         rrncf     mulplr, F
02C3 1722       00961         decfsz    count, F
02C4 C2BE       00962         goto      loop
                00963 ;
02C5 0002       00964         return
                00965 ;
                00966         PAGE
                00967 ;*****************************************************************
                00968 ;                       Numerical Differenciation
                00969 ;
                00970 ;       The so called "Three-Point Formula" is implemented to
                00971 ; differenciate a sequence of points (uniformly sampled).
                00972 ;       The eqn implemented is :
                00973 ;             f'(Xn) = [ f(Xn - 2h) - 4*f(Xn - h) + 3*f(Xn)]*0.5/h
                00974 ;    where Xn is the present sample and 'h' is the step size.
                00975 ;
                00976 ;       The above formula may be rewritten as :
                00977 ;
                00978 ;         f'(Xn) = [ 0.5*f(Xn -2) - 2*f(Xn - 1) + 0.5*3*f(Xn)]*1/DiffK
                00979 ;  where DiffK = h = Step Size
                00980 ;
                00981 ;   This differenciation routine can be used very effectively
                00982 ; in the computation of the differential component part in
                00983 ; a PID Loop calculation in Motor Control Applications
                00984 ;
                00985 ;       Double precision arithmetic is used throught
                00986 ;  The present sample value is assumed to be in locations
                00987 ; (XnHi, XnLo). The past two values are assumed to be in locations
                00988 ; (Xn_1_Hi, Xn_1_Lo) & (Xn_2_Hi, Xn_2_Lo).
                00989 ;       The output value is located in DiffHi & DiffLo. No overflow
                00990 ; checking mechanism is implemented. If the values are limited
                00991 ; to 12 bits, then the user need not worry about overflows
```

```
                00992 ;
                00993 ;  It is user's responsibility to update the past values with the
                00994 ; present values before calling this routine.
                00995 ;  After computation, the present value Xn is not moved to Xn_1
                00996 ; because the user may want these values to be intact for other
                00997 ; computations ( say numerical integration)
                00998 ;       Also it is user's responsibility to set past 2 values
                00999 ; (Xn_1 & Xn_2) values to be zero on initialization.
                01000 ;
                01001 ;*****************************************************************
                01002 ;
02C6            01003 Diff
02C6 6A2B       01004         movfp     Xn_2_Lo,WREG
02C7 0E27       01005         addwf     XnLo,W
02C8 011A       01006         movwf     ACCbLO
02C9 6A2C       01007         movfp     Xn_2_Hi,WREG
02CA 1028       01008         addwfc    XnHi,W
02CB 011B       01009         movwf     ACCbHI          ; Y = f(Xn-2) + f(Xn)
                01010 ;
02CC 6A27       01011         movfp     XnLo,WREG
02CD 0F1A       01012         addwf     ACCbLO, F
02CE 6A28       01013         movfp     XnHi,WREG
02CF 111B       01014         addwfc    ACCbHI, F
02D0 6A27       01015         movfp     XnLo,WREG
02D1 0F1A       01016         addwf     ACCbLO, F
02D2 6A28       01017         movfp     XnHi,WREG
02D3 111B       01018         addwfc    ACCbHI, F      ; Y = f(Xn-2) + 3*f(Xn)
                01019 ;
02D4 8804       01020         bcf      ALUSTA,C
02D5 191B       01021         rrcf     ACCbHI, F
02D6 191A       01022         rrcf     ACCbLO, F      ; Y = 0.5*[ f(Xn-2) + 3*f(Xn) ]
                01023
                01024 ;
02D7 6A29       01025         movfp    Xn_1_Lo,WREG
02D8 051A       01026         subwf    ACCbLO, F
02D9 6A2A       01027         movfp    Xn_1_Hi,WREG
02DA 031B       01028         subwfb   ACCbHI, F
02DB 6A29       01029         movfp    Xn_1_Lo,WREG
02DC 051A       01030         subwf    ACCbLO, F
02DD 6A2A       01031         movfp    Xn_1_Hi,WREG
02DE 031B       01032         subwfb   ACCbHI, F      ; Y = 0.5*[f(Xn-2) + 3*f(Xn)] - 2*f(Xn-1)
                01033 ;
02DF 6A2D       01034         movfp    DiffKLo,WREG
02E0 0118       01035         movwf    ACCaLO
02E1 6A2E       01036         movfp    DiffKHi,WREG
02E2 0119       01037         movwf    ACCaHI
                01038 ;
02E3 E119       01039         call    D_divS
02E4 6A1A       01040         movfp   ACCbLO,WREG
02E5 012F       01041         movwf   DiffLo
02E6 6A1B       01042         movfp   ACCbHI,WREG
02E7 0130       01043         movwf   DiffHi          ; result = Y/h
                01044 ;
02E8 0002       01045         return
                01046 ;
                01047         PAGE
                01048 ;*****************************************************************
                01049 ;
                01050 ;                       Numerical Integration
                01051 ;
                01052 ;
                01053 ;       Simpson's Three-Eighths Rule is implemented
                01054 ;
                01055 ;       Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*3*h/8
                01056 ;
                01057 ;  where 'h' is the step size and the integral is over the
```

```
                01058 ;  range X0 to X3
                01059 ;       The above equation can be rewritten as
                01060 ;
                01061 ;       Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*IntgK
                01062 ;
                01063 ;   where IntgK = 3*h/8 (in locations (IntgKHi, IntgKHi)
                01064 ;
                01065 ;   This Integration routine can be used very effectively
                01066 ; in the computation of the integral component part in
                01067 ; a PID Loop calculation in Motor Control Applications
                01068 ;
                01069 ;       Double precision arithmetic is used throught
                01070 ;  The three input values over which the integral is to be computed
                01071 ; are assumed to be in locations (X0Lo,X0Hi), (X1Lo,X1Hi) , (X2Lo,X2Hi)
                01072 ; and (X3Lo,X3Hi)
                01073 ;       The output value is located in IntgHi & IntgLo. No overflow
                01074 ; checking mechanism is implemented. If the values are limited
                01075 ; to 12 bits, then the user need not worry about overflows
                01076 ;
                01077 ;  It is user's responsibility to update the past values with the
                01078 ; present values before calling this routine.
                01079 ;  After computation, the present value Xn is not moved to Xn_1
                01080 ; because the user may want these values to be intact for other
                01081 ; computations ( say numerical integration)
                01082 ;       Also it is user's responsibility to set past 2 values
                01083 ; (Xn_1 & Xn_2) values to be zero on initialization.
                01084 ;
                01085 ;
                01086 ;*****************************************************************
                01087 ;
02E9            01088 Integrate
02E9 6A31       01089         movfp     X0Lo,WREG
02EA 0E37       01090         addwf     X3Lo,W
02EB 011A       01091         movwf     ACCbLO
02EC 6A32       01092         movfp     X0Hi,WREG
02ED 1038       01093         addwfc    X3Hi,W
02EE 011B       01094         movwf     ACCbHI          ; Intg = f(X0) + f(X3)
                01095 ;
02EF 6A33       01096         movfp     X1Lo,WREG
02F0 0F1A       01097         addwf     ACCbLO, F
02F1 6A34       01098         movfp     X1Hi,WREG
02F2 111B       01099         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +X1
02F3 6A33       01100         movfp     X1Lo,WREG
02F4 0F1A       01101         addwf     ACCbLO, F
02F5 6A34       01102         movfp     X1Hi,WREG
02F6 111B       01103         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +2*X1
02F7 6A33       01104         movfp     X1Lo,WREG
02F8 0F1A       01105         addwf     ACCbLO, F
02F9 6A34       01106         movfp     X1Hi,WREG
02FA 111B       01107         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1
                01108 ;
02FB 6A35       01109         movfp     X2Lo,WREG
02FC 0F1A       01110         addwf     ACCbLO, F
02FD 6A36       01111         movfp     X2Hi,WREG
02FE 111B       01112         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1 + X2
02FF 6A35       01113         movfp     X2Lo,WREG
0300 0F1A       01114         addwf     ACCbLO, F
0301 6A36       01115         movfp     X2Hi,WREG
0302 111B       01116         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1 + 2*X2
0303 6A35       01117         movfp     X2Lo,WREG
0304 0F1A       01118         addwf     ACCbLO, F
0305 6A36       01119         movfp     X2Hi,WREG
0306 111B       01120         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1 + 3*X2
                01121 ;
0307 6A39       01122         movfp     IntgKLo,WREG
0308 0118       01123         movwf     ACCaLO
```

```
0309 6A3A     01124         movfp     IntgKHi,WREG
030A 0119     01125         movwf     ACCaHI       ;  ACCa = IntgK (prepare for multiplication)
              01126 ;
030B E050     01127        call      D_mpyS       ; make sure to set for either SIGNED or UNSIGNED
030C 6A1E     01128         movfp     ACCdLO,WREG
030D 013B     01129         movwf     IntgLo       ; 32 bit result in ACCd & ACCc
030E 6A1F     01130         movfp     ACCdHI,WREG
030F 013C     01131         movwf     IntgHi       ; upper 16 bits = result
              01132 ;
0310 0002     01133         return
              01134 ;
              01135         PAGE
              01136 ;*****************************************************************
              01137 ;
              01138 ;                       Random Number Generator
              01139 ;
              01140 ; This routine generates a 16 Bit Pseudo Sequence Random Generator
              01141 ; It is based on Linear shift register feedback. The sequence
              01142 ; is generated by (Q15 xorwf Q14 xorwf Q12 xorwf Q3 )
              01143 ;
              01144 ;    The 16 bit random number is in location RandHi(high byte)
              01145 ;  & RandLo (low byte)
              01146 ;
              01147 ;       Before calling this routine, make sure the initial values
              01148 ; of RandHi & RandLo are NOT ZERO
              01149 ;       A good chiose of initial random number is 0x3045
              01150 ;*****************************************************************
              01151 ;
0311          01152 Random16
0311 1A19     01153         rlcf      RandHi,W
0312 0C19     01154         xorwf     RandHi,W
0313 1B0A     01155         rlcf      WREG, F      ; carry bit = xorwf(Q15,14)
              01156 ;
0314 1D19     01157         swapf     RandHi, F
0315 1C18     01158         swapf     RandLo,W
0316 230A     01159         rlncf     WREG, F
0317 0C19     01160         xorwf     RandHi,W     ; LSB = xorwf(Q12,Q3)
0318 1D19     01161         swapf     RandHi, F
0319 B501     01162         andlw     0x01
031A 1B18     01163         rlcf      RandLo, F
031B 0D18     01164         xorwf     RandLo, F
031C 1B19     01165         rlcf      RandHi, F
031D 0002     01166         return
              01167 ;
              01168         PAGE
              01169 ;*****************************************************************
              01170 ;                       Gaussian Noise Generator
              01171 ;
              01172 ; This routine generates a 16 Bit Gaussian distributed random
              01173 ; points. This routine calls the routine “Random16”, which
              01174 ; generates a psuedo random noise sequence. Gaussian noise
              01175 ; is computed using the CENTRAL LIMIT THEOREM.
              01176 ;       The Central Limit Theorem states that the average weighted
              01177 ; sum of uncorelated samples tends to have a Gaussian distribution
              01178 ; For practical purposes, the sum could be over a sample size
              01179 ; of 32 Random numbers. Better results could result if a larger
              01180 ; sample size is desired. For faster results, a sum over 16 samples
              01181 ; would also be adequate ( say, for applications like Speech synthesis,
              01182 ; channel simulations, etc).
              01183 ;
              01184 ;       The 16 bit Gaussian distributed point is in locations
              01185 ; GaussHi & GaussLo
              01186 ;
              01187 ;       Before calling this routine, the initial seed of Random
              01188 ; number should be NON ZERO ( refer to notes on “Random16” routine
              01189 ;
```

```
             01190 ;*****************************************************************
             01191 ;
031E         01192 Gauss
031E 2922    01193         clrf     count, F
031F 8522    01194         bsf      count,5         ; set Sample size = 32
0320 291A    01195         clrf     GaussLo, F
0321 291B    01196         clrf     GaussHi, F
0322 2920    01197         clrf     GaussTmp, F
             01198 ;
0323         01199 NextGauss
0323 E311    01200         call     Random16        ; get a random value
0324 6A18    01201         movfp    RandLo,WREG
0325 0F1A    01202         addwf    GaussLo, F
0326 6A19    01203         movfp    RandHi,WREG
0327 111B    01204         addwfc   GaussHi, F
0328 290A    01205         clrf     WREG, F
0329 1120    01206         addwfc   GaussTmp, F
032A 1722    01207         decfsz   count, F
032B C323    01208         goto     NextGauss       ; sum 16 random numbers
             01209 ;
032C B005    01210         movlw    5
032D         01211 GaussDiv16
032D 1920    01212         rrcf     GaussTmp, F
032E 191B    01213         rrcf     GaussHi, F
032F 191A    01214         rrcf     GaussLo, F           ; weghted average
0330 170A    01215         decfsz   WREG, F              ; divide by 32
0331 C32D    01216         goto     GaussDiv16
             01217 ;
0332 0002    01218         return
             01219 ;
             01220
             01221         END                      ; End Of arith.asm
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0300 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX-------------

All other memory blocks unused.

Program Memory Words Used:   819


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

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

```
MPASM 01.40 Released              BCD.ASM   1-16-1997  15:11:16         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001
                     00002     TITLE   “BCD Arithmetic Routines : Ver 1.0”
                     00003
                     00004 ;*****************************************************************
                     00005 ;                       BCD Arithmetic Routines
                     00006 ;
                     00007 ;       Program:          BCD.ASM
                     00008 ;       Revision Date:
                     00009 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00010 ;
                     00011 ;*****************************************************************
                     00012
                     00013         LIST     P = 17C42, columns=120, WRAP, L=0
                     00014
                     00015         include      “P17C42.INC”
                     00001         LIST
                     00002 ; P17C42.INC  Standard Header File, Version 1.03  Microchip Technology, Inc.
                     00264         LIST
                     00016
                     00017
                     00018         #define TRUE    1
                     00019         #define FALSE   0
                     00020
  00000001           00021 _INC    equ     1
  00000000           00022 _NO_INC equ     0
  00000000           00023 _LOW    equ     0
  00000001           00024 _HIGH   equ     1
                     00025
                     00026
                     00027         CBLOCK   0x20
  00000020           00028                 Lbyte, Hbyte
  00000022           00029                 R2, R1, R0       ; must maintain R2, R1, R0 sequence
  00000025           00030                 count
  00000026           00031                 Num1, Num2
                     00032         ENDC
                     00033 ;
  00000026           00034 BCD     equ      Num1
  00000026           00035 Htemp   equ      Num1
  00000027           00036 Ltemp   equ      Num2
                     00037 ;
                     00038         PAGE
                     00039
0000                 00040         ORG     0x0000
                     00041 ;*****************************************************************
                     00042 ;                       BCD Arithmetic Test Program
                     00043 ;*****************************************************************
                     00044 ;
0000                 00045 main
0000 2B21            00046         setf     Hbyte, F
0001 2B20            00047         setf     Lbyte, F
                     00048 ;                                ; 16 bit binary num = 0xffff
0002 E01F            00049         call     B2_BCD_Looped  ; after conversion the Decimal Num
                     00050 ;                                ; in R0, R1, R2 = 06,55,35
0003 2B21            00051         setf     Hbyte, F
```

```
0004 2B20     00052         setf     Lbyte, F
0005 E03F     00053         call     B2_BCD_Straight ; same as above, but straight line code
              00054 ;
0006 B006     00055         movlw    0x06
0007 0124     00056         movwf    R0
0008 B055     00057         movlw    0x55
0009 0123     00058         movwf     1
000A B035     00059         movlw    0x35
000B 0122     00060         movwf    R2              ; setf R0R1R2 = 65535
              00061 ;
000C E082     00062         call     BCDtoB          ; after conversion Hbyte = 0xff
              00063 ;                                ; and Lbyte = 0xff
000D B099     00064         movlw    0x99
000E 0126     00065         movwf    Num1
000F B099     00066         movlw    0x99
0010 0127     00067         movwf    Num2            ; setf Num1 = Num2 = 0x99 (max BCD)
              00068 ;
0011 E093     00069         call     BCDAdd          ; after addition, Num2 = 98
              00070 ;                                ; and Num1 = 01 ( 99+99 = 198)
              00071 ;
0012 B063     00072         movlw    0x63            ; setf Wreg = 63 hex
0013 E015     00073         call     BinBCD          ; after conversion, BCD = 99
              00074 ;                                ; 63 hex = 99 decimal.
              00075 ;
0014 C014     00076 self    goto     self
              00077 ;
              00078         PAGE
              00079 ;*****************************************************************;
              00080 ;                  Binary To BCD Conversion Routine (8 bit)
              00081 ;
              00082 ;       This routine converts the 8 bit binary number in the W Reg
              00083 ; to a 2 digit BCD number in location BCD( compacted BCD Code)
              00084 ;       The least significant digit is returned in location LSD and
              00085 ; the most significant digit is returned in location MSD.
              00086 ;
              00087 ;   Performance :
              00088 ;               Program Memory  :  10
              00089 ;               Clock Cycles    :  62  (worst case when W = 63 Hex )
              00090 ;                                     ( i.e max Decimal number 99 )
              00091 ;*****************************************************************
              00092 ;
0015          00093 BinBCD
0015 2926     00094         clrf     BCD, F
0016          00095 again
0016 B1F0     00096         addlw    -10
0017 9004     00097         btfss    ALUSTA,C
0018 C01B     00098         goto     swapBCD
0019 1526     00099         incf     BCD, F
001A C016     00100         goto     again
001B          00101 swapBCD
001B B110     00102         addlw    10
001C 1D26     00103         swapf    BCD, F
001D 0926     00104         iorwf    BCD, F
001E 0002     00105         return
              00106 ;
              00107         PAGE
              00108 ;*****************************************************************
              00109 ;                Binary To BCD Conversion Routine (16 Bit)
              00110 ;                       (LOOPED Version)
              00111 ;
              00112 ;      This routine converts a 16 Bit binary Number to a 5 Digit
              00113 ; BCD Number.
              00114 ;
              00115 ;       The 16 bit binary number is input in locations Hbyte and
              00116 ; Lbyte with the high byte in Hbyte.
              00117 ;       The 5 digit BCD number is returned in R0, R1 and R2 with R0
```

```
                00118 ; containing the MSD in its right most nibble.
                00119 ;
                00120 ;   Performance :
                00121 ;               Program Memory  :  32
                00122 ;               Clock Cycles    :  750
                00123 ;
                00124 ;*****************************************************************;
                00125 ;
001F            00126 B2_BCD_Looped
001F 8404       00127         bsf     ALUSTA,FS0
0020 8504       00128         bsf     ALUSTA,FS1      ; set FSR0 for no auto increment
                00129 ;
0021 8804       00130         bcf     ALUSTA,C
0022 2925       00131         clrf    count, F
0023 8425       00132         bsf     count,4         ; set count = 16
0024 2924       00133         clrf    R0, F
0025 2923       00134         clrf    R1, F
0026 2922       00135         clrf    R2, F
0027            00136 loop16a
0027 1B20       00137         rlcf    Lbyte, F
0028 1B21       00138         rlcf    Hbyte, F
0029 1B22       00139         rlcf    R2, F
002A 1B23       00140         rlcf    R1, F
002B 1B24       00141         rlcf    R0, F
                00142 ;
002C 2725       00143         dcfsnz  count, F
002D 0002       00144         return
002E            00145 adjDEC
002E B022       00146         movlw   R2               ; load R2 as indirect address ptr
002F 0101       00147         movwf   FSR0
0030 E036       00148         call    adjBCD
                00149 ;
0031 1501       00150         incf    FSR0, F
0032 E036       00151         call    adjBCD
                00152 ;
0033 1501       00153         incf    FSR0, F
0034 E036       00154         call    adjBCD
                00155 ;
0035 C027       00156         goto    loop16a
                00157 ;
0036            00158 adjBCD
0036 6A00       00159         movfp   INDF0,WREG
0037 B103       00160         addlw   0x03
0038 9B0A       00161         btfsc   WREG,3          ; test if result > 7
0039 0100       00162         movwf   INDF0
003A 6A00       00163         movfp   INDF0,WREG
003B B130       00164         addlw   0x30
003C 9F0A       00165         btfsc   WREG,7          ; test if result > 7
003D 0100       00166         movwf   INDF0           ; save as MSD
003E 0002       00167         return
                00168 ;
                00169 ;*****************************************************************
                00170 ;               Binary To BCD Conversion Routine (16 Bit)
                00171 ;                      (Partial Straight Line Version)
                00172 ;
                00173 ;      This routine converts a 16 Bit binary Number to a 5 Digit
                00174 ; BCD Number.
                00175 ;
                00176 ;       The 16 bit binary number is input in locations Hbyte and
                00177 ; Lbyte with the high byte in Hbyte.
                00178 ;       The 5 digit BCD number is returned in R0, R1 and R2 with R0
                00179 ; containing the MSD in its right most nibble.
                00180 ;
                00181 ;   Performance :
                00182 ;               Program Memory  :  44
                00183 ;               Clock Cycles    :  572
```

```
                00184 ;
                00185 ;*******************************************************************;
                00186 ;
003F            00187 B2_BCD_Straight
003F 8404       00188         bsf      ALUSTA,FS0
0040 8504       00189         bsf      ALUSTA,FS1      ; set FSR0 for no auto increment
                00190 ;
0041 8804       00191         bcf      ALUSTA,C
0042 2925       00192         clrf     count, F
0043 8425       00193         bsf      count,4         ; set count = 16
0044 2924       00194         clrf     R0, F
0045 2923       00195         clrf     R1, F
0046 2922       00196         clrf     R2, F
0047            00197 loop16b
0047 1B20       00198         rlcf     Lbyte, F
0048 1B21       00199         rlcf     Hbyte, F
0049 1B22       00200         rlcf     R2, F
004A 1B23       00201         rlcf     R1, F
004B 1B24       00202         rlcf     R0, F
                00203 ;
004C 2725       00204         dcfsnz   count, F
004D 0002       00205         return                   ; DONE
004E B022       00206         movlw    R2              ; load R2 as indirect address ptr
004F 0101       00207         movwf    FSR0
                00208 ; adjustBCD
0050 6A00       00209         movfp    INDF0,WREG
0051 B103       00210         addlw    0x03
0052 9B0A       00211         btfsc    WREG,3          ; test if result > 7
0053 0100       00212         movwf    INDF0
0054 6A00       00213         movfp    INDF0,WREG
0055 B130       00214         addlw    0x30
0056 9F0A       00215         btfsc    WREG,7          ; test if result > 7
0057 0100       00216         movwf    INDF0           ; save as MSD
                00217 ;
0058 1501       00218         incf     FSR0, F
                00219 ; adjustBCD
0059 6A00       00220         movfp    INDF0,WREG
005A B103       00221         addlw    0x03
005B 9B0A       00222         btfsc    WREG,3          ; test if result > 7
005C 0100       00223         movwf    INDF0
005D 6A00       00224         movfp    INDF0,WREG
005E B130       00225         addlw    0x30
005F 9F0A       00226         btfsc    WREG,7          ; test if result > 7
0060 0100       00227         movwf    INDF0           ; save as MSD
                00228 ;
0061 1501       00229         incf     FSR0, F
                00230 ; adjustBCD
0062 6A00       00231         movfp    INDF0,WREG
0063 B103       00232         addlw    0x03
0064 9B0A       00233         btfsc    WREG,3          ; test if result > 7
0065 0100       00234         movwf    INDF0
0066 6A00       00235         movfp    INDF0,WREG
0067 B130       00236         addlw    0x30
0068 9F0A       00237         btfsc    WREG,7          ; test if result > 7
0069 0100       00238         movwf    INDF0           ; save as MSD
                00239 ;
006A C047       00240         goto     loop16b
                00241 ;
                00242         PAGE
                00243 ;*************************************************************************
                00244 ;               BCD To Binary Conversion
                00245 ;
                00246 ;       This routine converts a 5 digit BCD number to a 16 bit binary
                00247 ; number.
                00248 ;       The input 5 digit BCD numbers are asumed to be in locations
                00249 ; R0, R1 & R2 with R0 containing the MSD in its right most nibble.
```

```
                00250 ;
                00251 ;       The 16 bit binary number is output in registers Hbyte & Lbyte
                00252 ; ( high byte & low byte repectively ).
                00253 ;
                00254 ;       The method used for conversion is :
                00255 ;               input number X = abcde ( the 5 digit BCD number )
                00256 ;       X = (R0,R1,R2) = abcde = 10[10[10[10a+b]+c]+d]+e
                00257 ;
                00258 ;   Performance :
                00259 ;               Program Memory  :  30
                00260 ;               Clock Cycles    :  112
                00261 ;
                00262 ;*****************************************************************;
                00263 ;
006B            00264 mpy10b
006B B50F       00265         andlw     0x0f
006C 0F20       00266         addwf     Lbyte, F
006D 9804       00267         btfsc     ALUSTA,C
006E 1521       00268         incf      Hbyte, F
006F            00269 mpy10a
006F 8804       00270         bcf       ALUSTA,C          ; multiply by 2
0070 1A20       00271         rlcf      Lbyte,W
0071 0127       00272         movwf     Ltemp
0072 1A21       00273         rlcf      Hbyte,W           ; (Htemp,Ltemp) = 2*N
0073 0126       00274         movwf     Htemp
                00275 ;
0074 8804       00276         bcf       ALUSTA,C          ; multiply by 2
0075 1B20       00277         rlcf      Lbyte, F
0076 1B21       00278         rlcf      Hbyte, F
0077 8804       00279         bcf       ALUSTA,C          ; multiply by 2
0078 1B20       00280         rlcf      Lbyte, F
0079 1B21       00281         rlcf      Hbyte, F
007A 8804       00282         bcf       ALUSTA,C          ; multiply by 2
007B 1B20       00283         rlcf      Lbyte, F
007C 1B21       00284         rlcf      Hbyte, F          ; (Hbyte,Lbyte) = 8*N
                00285 ;
007D 6A27       00286         movfp     Ltemp,WREG
007E 0F20       00287         addwf     Lbyte, F
007F 6A26       00288         movfp     Htemp,WREG
0080 1121       00289         addwfc    Hbyte, F
0081 0002       00290         return                      ; (Hbyte,Lbyte) = 10*N
                00291 ;
                00292 ;
0082            00293 BCDtoB
0082 2921       00294         clrf      Hbyte, F
0083 6A24       00295         movfp     R0,WREG
0084 B50F       00296         andlw     0x0f
0085 0120       00297         movwf     Lbyte
0086 E06F       00298         call      mpy10a          ; result = 10a+b
                00299 ;
0087 1C23       00300         swapf     R1,W
0088 E06B       00301         call      mpy10b          ; result = 10[10a+b]
                00302 ;
0089 6A23       00303         movfp     R1,WREG
008A E06B       00304         call      mpy10b          ; result = 10[10[10a+b]+c]
                00305 ;
008B 1C22       00306         swapf     R2,W
008C E06B       00307         call      mpy10b          ; result = 10[10[10[10a+b]+c]+d]
                00308 ;
008D 6A22       00309         movfp     R2,WREG
008E B50F       00310         andlw     0x0f
008F 0F20       00311         addwf     Lbyte, F
0090 9804       00312         btfsc     ALUSTA,C
0091 1521       00313         incf      Hbyte, F        ; result = 10[10[10[10a+b]+c]+d]+e
0092 0002       00314         return                    ; BCD to binary conversion done
                00315 ;
```

```
             00316         PAGE
             00317 ;*****************************************************************;
             00318 ;
             00319 ;                 Unsigned BCD Addition
             00320 ;
             00321 ;       This routine performs a 2 Digit Unsigned BCD Addition
             00322 ; It is assumed that the two BCD numbers to be added are in
             00323 ; locations Num1 & Num2. The result is the sum of Num1+Num2
             00324 ; and is stored in location Num2 and the overflow carry is returned
             00325 ; in location Num1
             00326 ;
             00327 ;   Performance :
             00328 ;               Program Memory  :       5
             00329 ;               Clock Cycles    :       5
             00330 ;
             00331 ;*****************************************************************;
             00332 ;
0093         00333 BCDAdd
0093 6A26    00334         movfp    Num1,WREG
0094 0E27    00335         addwf     Num2,W          ; perform binary addition
0095 2F27    00336         daw     Num2, F           ; adjust for BCD addition
0096 2926    00337         clrf     Num1, F
0097 1B26    00338         rlcf     Num1, F          ; set Num1 = carry bit
0098 0002    00339         return
             00340 ;
             00341 ;*****************************************************************
             00342 ;
             00343         END
BCD Arithmetic Routines : Ver 1.0
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXX------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   153


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: FXP_DIV.ASM

```
MPASM 01.40 Released            FLOAT.ASM   1-16-1997  15:11:46         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001
                     00002         TITLE   “Binary Floating Arithmetic Routines For PIC17C42 : Ver 1.0”
                     00003
                     00004         LIST    P = 17C42, columns=120, WRAP, L=0
                     00005 ;
                     00006         include       “p17c42.inc”
                     00001         LIST
                     00002 ; P17C42.INC Standard Header File, Version 1.03  Microchip Technology, Inc.
                     00264         LIST
                     00007
                     00008         #define TRUE    1
                     00009         #define FALSE   0
                     00010
                     00011         #define LSB     0
                     00012         #define MSB     7
                     00013 ;
                     00014 ;*****************************************************************
                     00015 ;               Binary Floating Point Addition, Subtraction And
                     00016 ;                       Multiplication Routines
                     00017 ;
                     00018 ;  Mantissa = 16 bits
                     00019 ;  Exponent = 8 bits    ( exponent is binary and not decimal)
                     00020 ;                        i.e a number ABCD EXP(X) = 0xABCD * (2**X)
                     00021 ;
                     00022 ;       Before calling any of the following floating point routines,
                     00023 ;  it is required to set Indirect Register 0 ( FSR0 ) for
                     00024 ;  No-Autoincrement ( i.e. Set bits FS0 & FS1 in ALUSTA to 1s)
                     00025 ;
                     00026 ;
                     00027 ;       Program:          FLOAT.ASM
                     00028 ;       Revision Date:
                     00029 ;                         1-13-97   Compatibility with MPASMWIN 1.40
                     00030 ;
                     00031 ;*****************************************************************;
                     00032 ;
                     00033         CBLOCK 0x20
  00000020           00034                 ACCaLO, ACCaHI, EXPa
  00000023           00035                 ACCbLO, ACCbHI, EXPb
  00000026           00036                 ACCcLO, ACCcHI
  00000028           00037                 ACCdLO, ACCdHI
  0000002A           00038                 temp,   sign
                     00039         ENDC
                     00040 ;
  00000001           00041 Mode16  equ     TRUE     ; Change this to FALSE for 32 bit product
                     00042
                     00043         PAGE
                     00044 ;
0000                 00045         ORG     0x0000
                     00046 ;
                     00047 ;*****************************************************************
                     00048 ;                 Floating Point Routines Test Program
                     00049 ;*****************************************************************
                     00050 ;
0000                 00051 main
```

```
                00052 ;
0000 8404       00053           bsf     ALUSTA,FS0      ; set FSR0 for no autoincrement
0001 8504       00054           bsf     ALUSTA,FS1
                00055 ;
0002 E009       00056           call    loadAB          ; result of adding
ACCb(EXPb)+ACCa(EXPa)->ACCb(EXPb)
0003 E019       00057           call    F_add           ; Here Accb = 403F, EXPb = 07
                00058 ;
0004 E009       00059           call    loadAB          ; result of subtracting
ACCb(EXPb)-ACCa(EXPa)->ACCb(EXPb)
0005 E016       00060           call    F_sub           ; Here Accb = 7F7F, EXPb = 06
                00061 ;
0006 E009       00062           call    loadAB          ; result of multiplying ACCb(EXPb) *
ACCa(EXPa)->ACCb(EXPb)
0007 E03B       00063           call    F_mpy           ; Here ACCb = FF7E, EXPb = 12
                00064 ;
0008 C008       00065 self      goto    self
                00066 ;
                00067 ;    Load constant values to (ACCa, EXPa) & (ACCb, EXPb) for testing
                00068 ;
0009            00069 loadAB
0009 B001       00070           movlw   0x01
000A 0121       00071           movwf   ACCaHI
000B B0FF       00072           movlw   0xff             ; loads ACCa = 01FF EXP(4)
000C 0120       00073           movwf   ACCaLO
000D B004       00074           movlw   0x04
000E 0122       00075           movwf   EXPa
                00076 ;
000F B07F       00077           movlw   0x7f
0010 0124       00078           movwf   ACCbHI
0011 B0FF       00079           movlw   0xff             ; loads ACCb = 7fff EXP(6)
0012 0123       00080           movwf   ACCbLO
0013 B006       00081           movlw   0x06
0014 0125       00082           movwf   EXPb
0015 0002       00083           return
                00084 ;
                00085           PAGE
                00086 ;*****************************************************************
                00087 ;       Floating Point Subtraction  ( ACCb - ACCa -> ACCb )
                00088 ;
                00089 ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                00090 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
                00091 ;          the 8 bit exponent in EXPa .
                00092 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
                00093 ;          the 8 bit exponent in EXPb .
                00094 ;      (c) CALL F_sub
                00095 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits ) with
                00096 ;          the 8 bit exponent in EXPb.
                00097 ;
                00098 ;*****************************************************************;
                00099 ;
0016            00100 F_sub
0016 B020       00101           movlw   ACCaLO
0017 0101       00102           movwf     FSR0
0018 E075       00103           call    negate          ; At first negate ACCa; Then addwf
                00104 ;
                00105 ;*****************************************************************
                00106 ;       Floating Point Addition  ( ACCb + ACCa -> ACCb )
                00107 ;
                00108 ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                00109 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
                00110 ;          the 8 bit exponent in EXPa.
                00111 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
                00112 ;          the 8 bit exponent in EXPb.
                00113 ;      (c) CALL F_add
                00114 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits ) with
```

```
                00115 ;          the 8 bit exponent in EXPb
                00116 ;
                00117 ;*****************************************************************
                00118 ;
0019            00119 F_add
0019 6A22       00120         movfp      EXPa,WREG
001A 3125       00121         cpfseq     EXPb
001B C01D       00122         goto       Lbl1
001C C026       00123         goto       noAddNorm          ; if exponents are equal
001D            00124 Lbl1
001D 3025       00125         cpfslt     EXPb
001E E09E       00126         call       F_swap             ; if B > A then swap ( A<->B )
001F 6A22       00127         movfp      EXPa,WREG
0020 0525       00128         subwf      EXPb, F
0021            00129 scloop
0021 E030       00130         call       addNorm
0022 1F25       00131         incfsz     EXPb, F
0023 C021       00132         goto       scloop
0024 6A22       00133         movfp      EXPa,WREG
0025 0125       00134         movwf      EXPb
0026            00135 noAddNorm
0026 6A21       00136         movfp      ACCaHI,WREG
0027 0824       00137         iorwf      ACCbHI,W
0028 012B       00138         movwf      sign               ; save the sign ( MSB states)
0029 E036       00139         call       D_add              ; compute double precision integer addwf
002A 972B       00140         btfss      sign,MSB
002B 9724       00141         btfss      ACCbHI,MSB
002C 0002       00142         return
002D 1525       00143         incf       EXPb, F
002E 8804       00144         bcf        ALUSTA,C
002F C033       00145         goto       shftR
                00146 ;
0030            00147 addNorm
0030 8804       00148         bcf        ALUSTA,C
0031 9F24       00149         btfsc      ACCbHI,MSB
0032 8004       00150         bsf        ALUSTA,C           ; set carry if < 0
0033            00151 shftR
0033 1924       00152         rrcf       ACCbHI, F
0034 1923       00153         rrcf       ACCbLO, F
0035 0002       00154         return
                00155 ;
                00156         PAGE
                00157 ;*****************************************************************
                00158 ;       Double Precision Addition
                00159 ;
0036            00160 D_add
0036 6A20       00161         movfp      ACCaLO,WREG
0037 0F23       00162         addwf      ACCbLO, F          ; addwf lsb
0038 6A21       00163         movfp      ACCaHI,WREG
0039 1124       00164         addwfc     ACCbHI, F          ; addwf msb with carry
003A 0002       00165         return
                00166 ;
                00167         PAGE
                00168 ;*****************************************************************
                00169 ;               Binary Floating Point Multiplication
                00170 ;
                00171 ;   Multiplication :
                00172         ;   ACCb(16 bits)EXP(b) * ACCa(16 bits)EXPa -> ACCb(16 bits)EXPb
                00173 ;   where, EXP(x) represents an 8 bit exponent.
                00174 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
                00175 ;                                an 8 bit exponent in location EXPa
                00176 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
                00177 ;                                an 8 bit exponent in location EXPb
                00178 ;      (c) CALL F_mpy
                00179 ;      (d) The 16 bit result overwrites ACCb(ACCbLO & ACCbHI). The exponent
                00180 ;               is stored in EXPb and the results are normalized.
```

```
                00181 ;
                00182 ; NOTE : If one needs to get a 32 bit product( & an 8 bit exponent ),
                00183 ;        re assemble the program after changing the line " Mode16 equ TRUE"
                00184 ;        to " Mode16  equ  FALSE ".
                00185 ;        If this option is chosen, then the 32 bit result is returned in
                00186 ;        ( ACCbHI, ACCbLO, ACCcHI, ACCcLO ) and the 8 bit exponent in EXPb.
                00187 ;        This method ( with " Mode16 equ FALSE " ) is NOT Recommended.
                00188 ;
                00189 ;  If a 32 bit mantissa is desired, set MODE16 equ FALSE
                00190 ;*****************************************************************
                00191 ;
003B            00192 F_mpy
003B E07D       00193         call    S_SIGN
003C E064       00194         call    setup
003D            00195 mloop
003D 8804       00196         bcf      ALUSTA,C
003E 1929       00197         rrcf     ACCdHI, F           ; rotate d right
003F 1928       00198         rrcf     ACCdLO, F
0040 9804       00199         btfsc    ALUSTA,C
0041 E036       00200         call     D_add
0042 1924       00201         rrcf     ACCbHI, f
0043 1923       00202         rrcf     ACCbLO, F
0044 1927       00203         rrcf     ACCcHI, F
0045 1926       00204         rrcf     ACCcLO, F
0046 172A       00205         decfsz   temp, F            ; loop until all bits checked
0047 C03D       00206         goto     mloop
                00207 ;
0048 6A22       00208         movfp    EXPa,WREG
0049 0F25       00209         addwf    EXPb, F
                00210 ;
                00211     #if Mode16
004A 3324       00212         tstfsz   ACCbHI
004B C05B       00213         goto     finup              ; if ACCbHI != 0
004C 3323       00214         tstfsz   ACCbLO
004D C055       00215         goto     Shft08             ; if ACCbLO != 0 && ACCbHI == 0
                00216 ;
004E 6A27       00217         movfp    ACCcHI,WREG
004F 0124       00218         movwf    ACCbHI             ; if ACCb == 0, then move ACCc to ACCb
0050 6A26       00219         movfp    ACCcLO,WREG
0051 0123       00220         movwf    ACCbLO
0052 B016       00221         movlw    16
0053 0F25       00222         addwf    EXPb, F
0054 C05B       00223         goto     finup
                00224 ;
0055            00225 Shft08
0055 6A23       00226         movfp    ACCbLO,WREG
0056 0124       00227         movwf    ACCbHI
0057 6A27       00228         movfp    ACCcHI,WREG
0058 0123       00229         movwf    ACCbLO
0059 B008       00230         movlw    8
005A 0F25       00231         addwf    EXPb, F
                00232 ;
                00233     #endif                          ; matching endif for IF Mode16
                00234 ;
005B            00235 finup
005B 972B       00236         btfss    sign,MSB
005C C08B       00237         goto     F_norm
                00238 ;
005D B026       00239         movlw    ACCcLO
005E 0101       00240         movwf    FSR0
005F E075       00241         call     negate
0060 B023       00242         movlw    ACCbLO
0061 0101       00243         movwf    FSR0
0062 E075       00244         call     negate
0063 C08B       00245         goto     F_norm             ; normalize floating point
                00246 ;
```

```
                00247 ;*****************************************************************
                00248 ;
0064            00249 setup
0064 292A       00250         clrf     temp, F
0065 842A       00251         bsf      temp,4              ; set temp = 16
0066 6A24       00252         movfp    ACCbHI,WREG         ; move ACCb to ACCd
0067 0129       00253         movwf    ACCdHI
0068 6A23       00254         movfp    ACCbLO,WREG
0069 0128       00255         movwf    ACCdLO
006A 2924       00256         clrf     ACCbHI, F
006B 2923       00257         clrf     ACCbLO, F           ; clear ACCb ( ACCbLO & ACCbHI )
006C 0002       00258         return
                00259 ;
                00260         PAGE
                00261 ;*****************************************************************
                00262 ;       Double Precision Negate Routines
                00263 ;
006D            00264 negateAlt
006D 6A00       00265         movfp    INDF0,WREG
006E 8D04       00266         bcf      ALUSTA,FS1
006F 2D00       00267         negw     INDF0, F
0070 8504       00268         bsf      ALUSTA,FS1
0071 6A00       00269         movfp    INDF0,WREG
0072 2900       00270         clrf     INDF0, F
0073 0300       00271         subwfb   INDF0, F
0074 0002       00272         return
                00273 ;
0075            00274 negate
0075 1300       00275         comf     INDF0, F
0076 8D04       00276         bcf      ALUSTA,FS1
0077 1500       00277         incf     INDF0, F
0078 8504       00278         bsf      ALUSTA,FS1
0079 9A04       00279         btfsc    ALUSTA,Z
007A 0700       00280         decf     INDF0, F
007B 1300       00281         comf     INDF0, F
007C 0002       00282         return
                00283 ;
                00284         PAGE
                00285 ;*****************************************************************
                00286 ;   Check Sign of the Number, if so negate and set the SIGN flag
                00287 ;
007D            00288 S_SIGN
007D 6A21       00289         movfp    ACCaHI,WREG
007E 0C24       00290         xorwf    ACCbHI,W
007F 012B       00291         movwf    sign
0080 9724       00292         btfss    ACCbHI,MSB          ; if MSB set go & negate ACCb
0081 C085       00293         goto     chek_A
                00294 ;
0082 B023       00295         movlw    ACCbLO
0083 0101       00296         movwf    FSR0
0084 E075       00297         call     negate
                00298 ;
0085            00299 chek_A
0085 9721       00300         btfss    ACCaHI,MSB          ; if MSB set go & negate ACCa
0086 0002       00301         return
0087 B020       00302         movlw    ACCaLO
0088 0101       00303         movwf    FSR0
0089 E075       00304         call     negate
008A 0002       00305         return
                00306 ;
                00307         PAGE
                00308 ;*****************************************************************
                00309 ;      Normalize Routine
                00310 ; Normalizes ACCb for use in floating point calculations.
                00311 ; Call this routine as often as possible to minimize the loss
                00312 ; of precission. This routine normalizes ACCb so that the
```

```
                00313 ; mantissa is maximized and the exponent minimized.
                00314 ;
                00315 ;*****************************************************************
                00316 ;
008B            00317 F_norm                            ; normalize ACCb
008B 3324       00318         tstfsz   ACCbHI
008C C090       00319         goto     C_norm
008D 3323       00320         tstfsz   ACCbLO
008E C090       00321         goto     C_norm
008F 0002       00322         return
                00323 ;
0090            00324 C_norm
0090 9E24       00325         btfsc    ACCbHI,MSB-1
0091 0002       00326         return
0092 E095       00327         call     shftSL
0093 0725       00328         decf     EXPb, F
0094 C090       00329         goto     C_norm
                00330 ;
0095            00331 shftSL
0095 8804       00332         bcf      ALUSTA,C
                00333 ;
                00334     #if   Mode16
0096 1B26       00335         rlcf     ACCcLO, F
0097 1B27       00336         rlcf     ACCcHI, F
                00337     #endif
                00338 ;
0098 1B23       00339         rlcf     ACCbLO, F
0099 1B24       00340         rlcf     ACCbHI, F
009A 8F24       00341         bcf      ACCbHI,MSB
009B 9804       00342         btfsc    ALUSTA,C
009C 8724       00343         bsf      ACCbHI,MSB
009D 0002       00344         return
                00345 ;
                00346 ;*****************************************************************
                00347 ;  Swap ACCa & ACCb   [ (ACCa,EXPa) <--> (ACCb,EXPb) ]
                00348 ;
009E            00349 F_swap
009E 6A21       00350         movfp    ACCaHI,WREG
009F 012A       00351         movwf    temp
00A0 6A24       00352         movfp    ACCbHI,WREG       ; ACCaHI <--> ACCbHI
00A1 0121       00353         movwf    ACCaHI
00A2 6A2A       00354         movfp    temp,WREG
00A3 0124       00355         movwf    ACCbHI
                00356 ;
00A4 6A20       00357         movfp    ACCaLO,WREG
00A5 012A       00358         movwf    temp
00A6 6A23       00359         movfp    ACCbLO,WREG       ; ACCaLO <--> ACCbLO
00A7 0120       00360         movwf    ACCaLO
00A8 6A2A       00361         movfp    temp,WREG
00A9 0123       00362         movwf    ACCbLO
                00363 ;
00AA 6A25       00364         movfp    EXPb,WREG
00AB 012A       00365         movwf    temp
00AC 6A22       00366         movfp    EXPa,WREG         ; EXPa <--> EXPb
00AD 0125       00367         movwf    EXPb
00AE 6A2A       00368         movfp    temp,WREG
00AF 0122       00369         movwf    EXPa
00B0 0002       00370         return
                00371 ;
                00372 ;*****************************************************************
                00373 ;               Normalizes A Floating Point Number
                00374 ;       The number is assumed to be  (LowByte, HighByte, Exp) in
                00375 ; consecutive locations. Before calling this routine, the address
                00376 ; of the LowByte should be loaded into FSR0 (indirect register ptr)
                00377 ;*****************************************************************
                00378 ;
```

```
00B1          00379 Normalize
00B1 1501     00380         incf     FSR0, F
00B2 3300     00381         tstfsz   INDF0
00B3 C0B9     00382         goto     NextNorm1
00B4 0701     00383         decf     FSR0, F
00B5 3300     00384         tstfsz   INDF0
00B6 C0B8     00385         goto     NextNorm2
00B7 0002     00386         return
00B8          00387 NextNorm2
00B8 1501     00388         incf     FSR0, F
00B9          00389 NextNorm1
00B9 9E00     00390         btfsc    INDF0,MSB-1
00BA 0002     00391         return
00BB E0C0     00392         call     shiftNorm
00BC 1501     00393         incf     FSR0, F
00BD 0700     00394         decf     INDF0, F
00BE 0701     00395         decf     FSR0, F
00BF C0B9     00396         goto     NextNorm1
              00397 ;
00C0          00398 shiftNorm
00C0 8804     00399         bcf      ALUSTA,C
00C1 1501     00400         incf     FSR0, F
00C2 1B00     00401         rlcf     INDF0, F
00C3 1501     00402         incf     FSR0, F
00C4 1B00     00403         rlcf     INDF0, F
00C5 0701     00404         decf     FSR0, F
00C6 0701     00405         decf     FSR0, F
00C7 0701     00406         decf     FSR0, F
00C8 1B00     00407         rlcf     INDF0, F
00C9 1501     00408         incf     FSR0, F
00CA 1B00     00409         rlcf     INDF0, F
00CB 8F00     00410         bcf      INDF0,MSB
00CC 9804     00411         btfsc    ALUSTA,C
00CD 8700     00412         bsf      INDF0,MSB
00CE 0002     00413         return
              00414 ;
              00415
              00416         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXX- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   207


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV MSC The Netherlands Accredited by the RvA — DNV — ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

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

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

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

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

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

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

03/01/02