AN669

Embedding Assembly Routines into C Language Using a Floating Point Routine as an Example

Authors: Rick Evans
Richard Fischer
Microchip Technology, Inc.

INTRODUCTION

With the advent of MPLAB-C, the Microchip C-compiler, many PICmicro™ users need to embed existing assembly language routines and/or Microchip application notes into C. This application note explains how to embed an assembly language program into MPLAB-C, version 1.10, and the issues therein. For example, embedding interrupt save and restore must be done using assembly language. Also, critical timing routines may require assembly. The 32-bit floating point multiply routine from AN575 is used to illustrate this process. The remaining 32-bit floating point math routines are embedded into individual C functions and are included in the file accompanying this application note.

PROCEDURE

For this example, we'll use a PIC16C74A with 4K Program Memory, and 192 bytes of RAM.

Embedding assembly routines

In order to embed an assembly language routine in C code place the `#asm` and `#endasm` directives around the assembly routine. Furthermore, if this is a subroutine, as is the case with the floating point multiply, then embed the assembly code within a C function declaration. The `#asm` construct is illustrated in Example 1 with an excerpt from the 32-bit floating point routine.

EXAMPLE 1: #ASM, #ENDASM CONSTRUCT

```
void fpm32(void)
{
#asm

FPM32     MOVF    AEXP,W      ;test for zero
          BTFSS   _Z          ;arguements
          MOVF    BEXP,W
          BTFSC   _Z
          GOTO    RES032M

M32BNE0   MOVF    AARGB0,W
          XORWF   BARGB0,W
          MOVWF   SIGN         ;save sign
          MOVF    BEXP,W       ;in SIGN
          ADDWF   EXP, F
          MOVLW   EXPBIAS-1

   ;...etc.
#endasm
}
```

Locating the Routine in Program Memory, `GOTO`s and `CALL`s

There are two 2K word pages of program memory in the PIC16C74A. Program memory 000h to 7FFh is page 0, 800h to FFFh is page 1. By making `fpm32()` a C function, MPLAB-C initializes the appropriate page bit in the PCLATH register before the subroutine call is made. (See data sheet for more on PCLATH).

A potential problem could arise, however, if the new C function, `fpm32()`, crosses the page boundary (7FFh,800h). MPLAB-C does not insert code into the assembly code to initialize the page bits (remember MPLAB-C does take care of paging for function calls). That means it is up to the programmer to either; 1) add assembly language to initialize PCLATH appropriately, or 2) move the entire `#asm` function within a single page. Option 1 involves more work. The programmer must first compile the C code, then analyze the listing file to see if the assembly function crossed a page boundary. Finally, add the appropriate assembly language to initialize PCLATH then re-compile. This solution is not desirable since every time new C code is added to or deleted from the program, the routine, `fpm32()` can potentially move across the page boundary. Option 2 is the simplest solution - to locate the C function in a single page.

To illustrate, lets force `fpm32()` to cross the page boundary. A pragma directive is required to locate a routine (Example 2).

EXAMPLE 2: FORCING FPM32 TO CROSS THE PAGE BOUNDARY

```
#pragma memory ROM [MAXROM-0x7F0] @ 0x7F0;
#include "fpm32.inc"
```

The listing file generated is shown in Example 3. Notice the statement `GOTO MTUN32` at address 0x7FC. However, the routine `MTUN32` is located at address 0x801. Remember, with the PIC16C74A the `GOTO` instruction only has an eleven bit address range. With the `GOTO MTUN32` example, one more bit of address is needed to branch to 0x801 from 0x7FC. The extra bit of address is located in the PCLATH register. That means assembly code would have to be inserted into the floating point routines to initialize PCLATH before each `GOTO`. Since this solution is not desirable, the best approach is to locate the floating point subroutine in a single page. For example, change the `pragma` directive in Example 2 to locate the routine at 0x800.

It is important to note that when `fpm32()` is called as a C function, the page bit in PCLATH is updated by MPLAB-C. In other words MPLAB-C adds the necessary assembly language code needed to call `fpm32()` or any other C function. The C function is called correctly, but once within the C function, the raw embedded assembly language might have `GOTO`s or `CALL`s that cross over the page boundary and cause problems.

EXAMPLE 3: FPM32 FORCED TO ADDRESS 0x7F0 TO SHOW CROSSING FROM PAGE 0 TO PAGE 1

```
void fpm32 (void)
{

#asm
              .
              . some code here
              .
07F0 0838             FPM32         MOVF      AEXP,W         ;test for zero arguments
07F1 1D03                           BTFSS     _Z
07F2 0839                           MOVF      BEXP,W
07F3 1903                           BTFSC     _Z
07F4 284E                           GOTO      RES032M

07F5 0826             M32BNE0       MOVF      AARGB0,W
07F6 0633                           XORWF     BARGB0,W
07F7 00AE                           MOVWF     SIGN           ;save sign in SIGN
07F8 0839                           MOVF      BEXP,W
07F9 07B8                           ADDWF     EXP, F

07FA 307E                           MOVLW     EXPBIAS-1
07FB 1C03                           BTFSS     _C
07FC 2801                           GOTO      MTUN32         ;****** WON'T WORK !

07FD 02B8                           SUBWF     EXP,F
07FE 1803                           BTFSC     _C
07FF 2843                           GOTO      SETFOV32M      ;set multiply overflow flag
0800 2804                           GOTO      MOK32

0801 02B8             MTUN32        SUBWF     EXP,F          ;****** IN PAGE 1 !
0802 1C03                           BTFSS     _C
0803 2854                           GOTO      SETFUN32M
              .
              . some more code here
              .
#endasm
}
```


Assembly Language Variables, Include Files, etc.

For the floating point math routines of AN575, there is one include file which contains important constant and register declarations: `math16.inc`. This file of declarations is rather extensive, however, it is straightforward to convert it to C. Example 4 shows a segment of the `math16.inc` requiring some attention for the conversion.

EXAMPLE 4: MATH16.INC EXCERPT FROM AN575. ASSEMBLY LANGUAGE FILE

```
B0      equu     0
B1      equ     1
B2      equ     2
B3      equ     3
B4      equ     4
B5      equ     5
B6      equ     6
B7      equ     7
MSB     equ     7
LSB     equ     0
    .
    . etc.
    .
AARGB7  equ     0x20
AARGB6  equ     0x21
AARGB5  equ     0x22
AARGB4  equ     0x23
AARGB3  equ     0x24
AARGB2  equ     0x25
AARGB1  equ     0x26
AARGB0  equ     0x27
AARG    equ     0x27   ; most significant
                       ; byte of argument A
```

These Constant and Variable Declarations Need to be Converted to C Language Declarations

Example 5 shows the equivalent C constant and variable declarations. The equates in assembly language create constants. The equivalent C language is a `#define`. Moreover, variables are declared in assembly language by equating a variable name to a register RAM location (i.e. `AARGB7 equ 0x20`). In C the variables are declared by assigning a type to the variable. In the listing in Example 5, `AARGB7` is declared as an unsigned integer data type.

EXAMPLE 5: THE CONVERTED MATH16C.C FILE. C LANGUAGE FILE

```
#define  B0     0
#define  B1     1
#define  B2     2
#define  B3     3
#define  B4     4
#define  B5     5
#define  B6     6
#define  B7     7
#define  MSB    7
#define  LSB    0
         .
         . etc.
         .
unsigned int AARGB0 @ ACCB0;       // most significant byte of argument A
unsigned int AARGB1 @ ACCB1;
unsigned int AARGB2 @ ACCB2;
unsigned int AARGB3 @ ACCB3;
unsigned int AARGB4 @ ACCB4;
unsigned int AARGB5 @ ACCB5;
unsigned int AARGB6 @ ACCB6;
unsigned int AARGB7 @ ACCB7;       // least significant byte of argument A
unsigned int AARG   @ ACC;         // most significant byte of argument A
```

USING 32-BIT FLOATING POINT MULTIPLY

Using the 32-bit floating point multiply supplied with AN575 in a C program is straightforward. First, copy the entire routine from the file `fpm32.a16` (from AN575). Then, create a function with the same name as the assembly routine.

Lets take a well known formula:

$$A = \pi r^2$$

Let,

π = 3.141592654

r = 12.34567898 meters

Find A:

We need to convert the previous decimal numbers to Microchip 32-bit floating point. Use `fpm32` (from AN575), to solve the equation. We will use MPLAB-C and use our C function named `fpm32()`. The main routine is listed in Example 6.

AN575 comes with a handy utility called `fprep.exe`. This Microchip file is a DOS executable. When running `fprep`, you can enter in a decimal number and it displays the hexadecimal floating point number. Table 1 shows the numbers in our example and their equivalent floating point formats.

TABLE 1: PICmicro™ 32-BIT FLOATING POINT REPRESENTATIONS OF OUR EXAMPLE

	Microchip Floating Point Equivalent			
Decimal Number	**EXP**	**B0 (MSB)**	**B1**	**B2 (LSB)**
π = 3.141592654	0x80	0x49	0x0F	0xDB
r = 12.34567898 meters	0x82	0x45	0x87	0xE7
A = 478.8283246 m^2 -- fprep.exe calculated result	0x87	0x6F	0x6A	0x07
A = 478.8283246 m^2 -- PIC16C74A measured result using MPLAB 3.12 and PICMASTER 16J probe	0x87	0x6F	0x6A	0x07

EXAMPLE 6: MAIN ROUTINE TO TEST OUT OUR NEW 32-BIT FLOAT MULTIPLY IN C

```
#include "16c74a.h"
#include "math16c.c"
#include "fpm32.inc"
                              // Notice that fpm32 is located in page 0
                              // Thus, all GOTOs reside in the same page.
void main (void)
{
  AEXP   = 0X80;              // PI = 3.141592654
  AARGB0 = 0X49;
  AARGB1 = 0X0F;
  AARGB2 = 0XDB;
  BEXP   = 0X82;              // r = 12.34567898
  BARGB0 = 0X45;
  BARGB1 = 0X87;
  BARGB2 = 0XE7;

  fpm32();                    // AARG =  PI * r
                              // you must reload r into BARG since
                              // fpm32() destroys BARG.
  BEXP   = 0X82;              // r = 12.34567898
  BARGB0 = 0X45;
  BARGB1 = 0X87;
  BARGB2 = 0XE7;
  fpm32();                    // AARG = (PI*r)*r
  while(1);
}
```

SUMMARY

For this discussion only the 32-bit floating point multiply is used. However, the same principles of embedded assembly language routines into C code can be used with other assembly language routines. A summary list of a step- by- step process to embed assembly code into your C code is below:

- Convert assembly register `EQU` equates to C variable types such as `unsigned int.`
- Convert constants to `#define` in C.
- Place the assembly code into a subroutine using `#asm` and `#endasm`
- To avoid paging issues in parts with multiple program memory pages, force the code to an address where it will not cross a page boundary. For example:

```
#pragma memory ROM [MAXROM-0x800] @ 0x800;
```

- Macros and conditional assembly will have to be rewritten in actual in-line assembly code. The MPLAB-C compiler does not support these higher level assembly options to the same degree as the assembler, MPASM.

For your convenience, all the 32-bit floating point routines in application note AN575 are provided in a zip file along with this application note. Each routine has been separated to work as a stand-alone routine. There is a separate file for each floating point routine. The files may be included individually into your C code. Table 2 shows a list of all the files and routines included with this application note.

TABLE 2: 32-BIT FLOATING POINT C FILES/FUNCTIONS INCLUDED WITH THIS APPLICATION NOTE

AN575 Original Assembly Routine/file *	Equivalent C file/function	Purpose
-	example.c	The example `main()` routine calculating the area given the radius. (uses fpm32)
FLO2432	flo2432.inc	24-bit integer to 32-bit floating point conversion
FLO3232	flo3232.inc	32-bit integer to 32-bit floating point conversion
FPD32	fpd32.inc	32-bit floating point divide
FPM32	fpm32.inc	32-bit floating point multiply
FPA32 FPS32	fpsa32.inc fps32() 32-bit subtract fpa32() 32-bit add	32-bit floating point add 32-bit floating point subtract
INT3224	int3224.inc	32-bit floating point to 24-bit integer conversion
INT3232	int3232.inc	32-bit floating point to 32-bit integer conversion
NRM3232	nrm3232.inc	32-bit normalization of unnormalized 32-bit floating point numbers
NRM4032	nrm4032.inc	32-bit normalization of unnormalized 40-bit floating point numbers
math16.inc	math16c.c	variables and constants need for the floating point functions

* Check Microchip web site and bulletin board for latest code.

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.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rfPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

Printed on recycled paper.

DNV Certification, Inc. USA
DNV MSC The Netherlands Accredited by the RvA
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.

MICROCHIP

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: 65-334-8870 Fax: 65-334-8850

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

01/18/02