Using the uM-FPU with the PICmicro®

Micromega Corporation

Introduction

The uM-FPU is a 32-bit floating point coprocessor that can be easily interfaced with the Microchip PICmicro® family of microcontrollers to provide support for 32-bit IEEE 754 floating point operations and long integer operations. The uM-FPU is easy to connect, using two output pins and one input pin. There are no external components required.

uM-FPU Features

- 8-pin integrated circuit.
- No additional external components
- SPI compatible interface
- Sixteen 32-bit general purpose registers for storing floating point or long integer values
- Five 32-bit temporary registers with support for nested calculations (i.e. parenthesis)
- Floating Point Operations
  - Set, Add, Subtract, Multiply, Divide
  - Sqrt, Log, Log10, Exp, Exp10, Power, Root
  - Sin, Cos, Tan
  - Asin, Acos, Atan, Atan2
  - Floor, Ceil, Round, Min, Max, Fraction
  - Negate, Abs, Inverse
  - Convert Radians to Degrees
  - Convert Degrees to Radians
  - Compare, Status
- Long Integer Operations
  - Set, Add, Subtract, Multiply, Divide, Unsigned Divide
  - Negate, Abs
  - Compare, Unsigned Compare, Status
- Conversion Functions
  - Convert 8-bit and 16-bit integers to floating point
  - Convert 8-bit and 16-bit integers to long integer
  - Convert long integer to floating point
  - Convert floating point to long integer
  - Convert floating point to ASCII
  - Convert floating point to formatted ASCII
  - Convert long integer to ASCII
  - Convert long integer to formatted ASCII
  - Convert ASCII to floating point
  - Convert ASCII to long integer

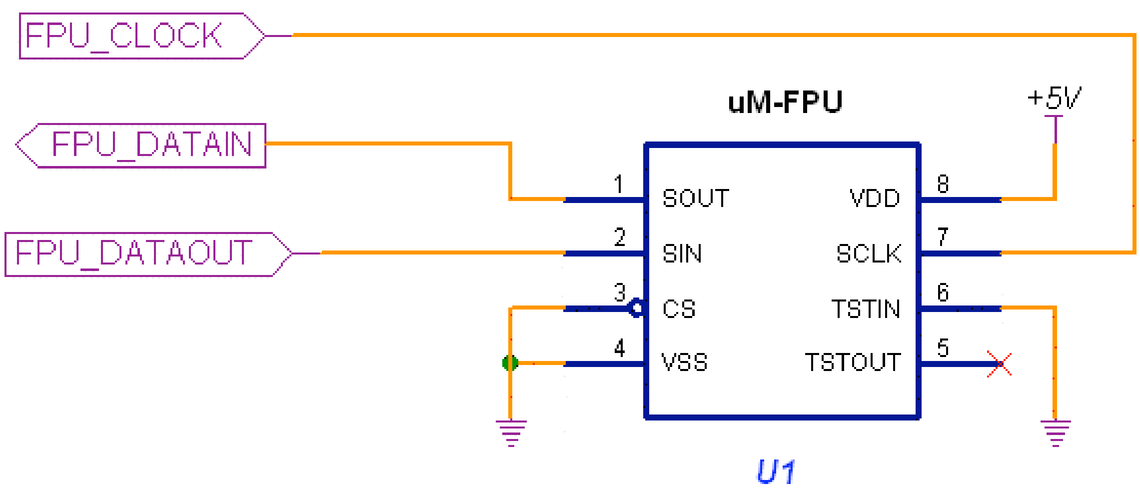
- Full set of PIC assembly support routines provided for easy implementation.

Connecting the uM-FPU to the Microchip PICmicro®

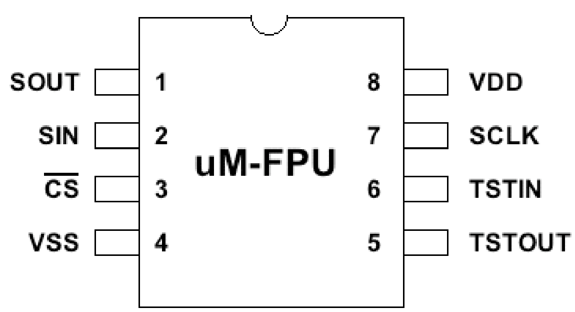
The uM-FPU requires two output pins and one input pin for interfacing to the Microchip PICmicro®. The communication is implemented using a SPI interface with the following connections:

FPU_CLOCK	clock
FPU_DATAOUT	data from PIC to uM-FPU
FPU_DATAIN	data from uM-FPU to PIC

Either a 2-wire or 3-wire SPI connection can be used, but this document, and the sample routines, use a 3-wire connection and assume the uM-FPU chip is always selected (as shown below). The pin assignments can be changed to suit your application.



uM-FPU Pin Assignment



PIN DESCRIPTION

SOUT	SPI Output
SIN	SPI Input
CS	Chip Select
VSS	Ground
TSTOUT	Test Output
TSTIN	Test Input
SCLK	SPI Clock
VDD	Power Supply Voltage (+5V)

Using the uM-FPU Floating Point Routines

A full set of assembler support routines is provided to handle all of the communication between the PIC and the uM-FPU. The routines are designed for use with the MPLAB IDE using the MPASM Assembler and MPLINK Object Linker. The routines could easily be adapted to other assemblers. The interface files are as follows:

umfpu.asm	High level routines for each uM-FPU function
umfpu.inc	Include file containing definitions for each function
fpusw_4.asm	Low level interface routines using software(bit-bang) SPI, 4 MHz
fpusw_20.asm	Low level interface routines using software(bit-bang) SPI, 20 MHz
fpuhw_4.asm	Low level interface routines using hardware SPI, 4 MHz
fpuhw_20.asm	Low level interface routines using hardware SPI, 20 MHz
delay_4.asm	Delay routine, 4 Mhz
delay_20.asm	Delay routine, 20 Mhz
serial.asm	Serial port routines to print data

MPLAB project files and linker files are provided for each of the sample applications. The files can be used directly to test the sample applications, or used as the starting point for a new program. Each uM-FPU support routine is described in the reference guide included as Appendix A of this document.

In order to ensure that the PIC and the uM-FPU coprocessor are synchronized, the reset routine must be called at the start of every program. This routine sets up the input/output pins and resets the uM-FPU. It is called as follows:

```
call   fpu_reset          ;reset the uM-FPU coprocessor
```

The uM-FPU contains sixteen 32-bit registers, numbered 0 through 15, which are used to store floating point or long integer values. Register 0 is reserved for use as a working register and is modified by some of the uM-FPU operations. Registers 1 through 15 are available for general use.

Arithmetic operations on the uM-FPU are defined in terms of A and B registers. For example:

FADD	A = A + B
FDIV	A = A / B
SQRT	A = sqrt(A)
SIN	A = sin(A)

To perform an operation, the appropriate function is called. For example:

```
call   sqrt              ;A = SQRT(A)
```

Any of the sixteen registers can be selected as the A or B registers. Two variables called regA and regB are used to specify the current register values. Macros SELECTA and SELECTB are used to set these variables. For example:

```
selectA 1               ;select Register 1 as A register
```

The B register is automatically selected by many of the uM-FPU commands. Since the interface routines set the regB variable appropriately, a separate SELECTB call is often not required.

The following code adds register 2 to register 1.

```
selectA 1               ;select Register 1 as A register
selectB 2               ;select Register 2 as B register
call fadd               ;A = A + B
```

Using symbol definitions to provide meaningful names for the uM-FPU registers creates a more readable program. The following code is the same as above, but uses symbol names.

```
#define Total      1      ;total amount  (uM-FPU register 1)
#define Value      2      ;current value (uM-FPU register 2)

selectA Total             ;Select Total as A register
selectB Value             ;Select Value as B register
call fadd                 ;Total = Total + Value
```

The following floating point routines are provided:

fset	A = B
fadd	A = A + B
fsub	A = A – B
fmul	A = A * B
fdiv	A = A / B

abs	A = |A|
acos	A = acos (A)
asin	A = asin(A)
atan	A = atan(A)
atan2	A = atan2(A)
ceil	A = ceil(A)
cos	A = cos(A)
exp	A = exp(A)
exp10	A = exp10(A)
fcompare	Compare A and B
fix	A = fix(B)
floor	A = floor(A)
fraction	register 0 = fractional part of A
fread	Read the value of A
fstatus	Get the floating point status of A
inverse	A = 1 / A
log	A = log(A)
log10	A = log10(A)
max	A = maximum of A and B
min	A = minimum of A and B
negate	A = -A
pow	A = A to the power of B
root	A = the Bth root of A
round	A = round(A)
sin	A = sin(A)
sqrt	A = sqrt(A)
tan	A = tan(A)
degrees	Convert radians to degrees
radians	Convert degrees to radians

The following example implements the equation Z = SQRT(X**2 + Y**2). The equation is broken into several steps: the X value is squared (multiplied by itself), the Y value is squared, the Z value is set to the sum of the squares, and the square root function is called to get the final result.

```
#define Xvalue 1        ;X value (uM-FPU register 1)
#define Yvalue 2        ;Y value (uM-FPU register 2)
#define Zvalue 3        ;Z value (uM-FPU register 3)


selectA Xvalue          ;X = X ** 2
selectB Xvalue
call fmul

selectA Yvalue          ;Y = Y ** 2
selectB Yvalue
call fmul

selectA Zvalue          ;Z = X + Y
selectB Xvalue
call fset
selectB Yvalue
call fadd

call sqrt               ;Z = sqrt(Z)
```

The value of A register is not changed by the uM-FPU support routines. If multiple operations are performed on the same register it isn't necessary to select it each time, only when it needs to change. For example:

```
selectA Result          ;Result = sqrt(Value1 + Value2 + Value3)
selectB Value1
call fset
selectB Value2
call fadd
selectB Value3
call fadd
call sqrt
```

Alternate Floating Point Format

Several compilers for the PICmicro® use a slightly modified version of the standard IEEE 754 floating point format. The alternate format is shown below:

Exponent	S	Mantissa
31 24	23	22 0

The uM-FPU uses the standard IEEE 754 format (as described in Appendix C) by default, but it can also support the alternate PIC format. To use the alternate PIC format, the following function call should be made immediately after a reset:

```
call picmode
```

All internal data on the uM-FPU is still stored in standard IEEE 754 format, but when the uM-FPU is in PIC mode an automatic conversion is done by the writeA, writeB and read functions so the PIC program can store floating point data in the alternate format.

Loading Floating Point Values

The MPASM assembler does not provide support for floating point number syntax, so floating point values must be entered using alternate methods. There are several ways to load floating point values into the uM-FPU. Functions are provided to:

load_floatByte	Load 8-bit signed integer and convert to floating point
load_floatUbyte	Load 8-bit unsigned integer and convert to floating point
load_floatword	Load 16-bit signed integer and convert to floating point
load_floatUword	Load 16-bit unsigned integer and convert to floating point
load_zero	Load the floating point value 0.0
load_one	Load the floating point value 1.0
load_e	Load the floating point value of e (2.7182818)
load_pi	Load the floating point value of pi (3.1415927)

The ATOF instruction can also be used to send an ASCII string to the uM-FPU which is converted to a floating point number.

Load a signed byte value:

```
call  load_floatByte    ;load 10, convert to float
movlw .10               ;send byte value
call  fpu_sendByte
```

Load an unsigned word value:

```
call  load_floatUword   ;load unsigned word, convert to float
movf  HIGH sensorValue  ;send word value MSB first
call  fpu_sendByte
movf  LOW sensorValue
call  fpu_sendByte
```

Load Zero:

```
call  load_zero         ;load register 0 with 0.0
```

Load Pi:

```
call  load_pi           ;load register 0 with 3.1415927
```

Floating point numbers are 32-bit values. (Appendix C describes the IEEE 754 32-bit floating point number format.) The easiest way to load a 32-bit floating point value is to use two 16-bit hexadecimal values. A handy utility program called `uM-FPU Converter` is available to convert between 32-bit floating point values and hexadecimal values. The `fwriteA` and `fwriteB` functions are used to load 32-bit values.

Load a floating point value directly in code:

```
selectB Angle           ;select Angle as B register
call  writeB            ;write 32-bit value to register
movlw 0x41              ;(floating point value 20.0)
call  fpu_sendByte
movlw 0xA0
call  fpu_sendByte
movlw 0x00
call  fpu_sendByte
movlw 0x00
call  fpu_sendByte
```

Since each of these commands sets the B register value, arithmetic operations can immediately follow the load command. For example:

```
selectA Angle           ;Angle = Angle / pi
call  load_pi
call  fdiv

selectA Value           ;Value = Value + 2
call  load_floatByte
movlw .2
call  fpu_sendByte
call  fadd
```

The fastest operations occur when the uM-FPU registers are already loaded with values. In time critical portions of code, floating point constants should be loaded beforehand to maximize the processing speed in the critical section. With fifteen registers available for storage on the uM-FPU, it is often possible to preload all of the required constant values. Since the load routines must send data to the uM-FPU for conversion, there is additional overhead associated with each type of load. The majority of the overhead is associated with the data transfer. For example, the load_floatByte function requires an additional 8-bit value, load_floatWord requies two 8-bit values, and writeA and writeB requires four 8-bit values. Minimizing the amount of data transfer will maximize the execution speed of your program.

Comparing and Testing Floating Point Values

A floating point value can be positive zero, negative zero, positive non-zero, negative non-zero, positive infinite, negative infinity or Not a Number (which occurs if an invalid operation is performed on a floating point value). The following symbols define the floating point status bits:

status_Zero	Zero bit	(0 – not zero, 1 – zero)
status_Sign	Sign bit	(0 – positive, 1 – negative)
status_NaN	Not-a-Number	(0 – valid number, 1 – NaN)
status_Zero	Infinity	(0 – not infinite, 1 – infinite)

The `fstatus` command is used to check the status of a floating point number. For example:

```
    call  fstatus           ;check status of A register
    btfsc status_Zero
    goto  zeroValue
    btfsc status_Sign
    goto  negativeValue

    ;value is positive
      …
negativeValue:
    ;value is negative
      …
zeroValue:
    ;value is zero
      …
```

The `fcompare` command is used to compare two floating point values. The status bits are set for the results of the operation A – B. (The selected A and B registers are not modified). For example:

```
    call  fcompare          ;compare A and B registers
    btfsc status_Zero
    goto  sameAs
    btfsc status_Sign
    goto  lessThan

    ;A > B
      …
lessThan
    ;A < B"
      …
sameAs
    ;A = B
      …
```

Using the uM-FPU Long Integer Routines

Any of the sixteen uM-FPU registers can be used to store long integer values. The support routines for long integers work in exactly the same manner as the floating point routines and are defined in terms of the A and B registers. For example:

```
#define Total 1           ;total amount (uM-FPU register 1)
#define Value 2           ;current count (uM-FPU register 2)

selectA Total             ;Total = Total + Value
selectB Value
call  ladd
```

The following long integer routines are provided:

lset	A = B
ladd	A = A + B
lsub	A = A – B
lmul	A = A * B
ldiv	A = A / B
ludiv	A = A / B (unsigned)
float	A = float(A)
labs	A = |A|
lcompare	Compare A and B
lstatus	Get the long integer status of A
lnegate	A = -A
lucompare	Compare A and B (unsigned)

Loading Long Integer Values

There are several ways to load long integer values into the uM-FPU. Commands are provided to:

load_longByte	Load 8-bit signed integer and convert to long integer
load_longUbyte	Load 8-bit unsigned integer and convert to long integer
load_longWord	Load 16-bit signed integer and convert to long integer
load_longUword	Load 16-bit unsigned integer and convert to long integer
load_zero	Load the long integer value 0

The ATOL instruction can also be used to send an ASCII string to the uM-FPU which is converted to a long integer number.

Load a byte value:

```
call  load_longByte       ;load byte value, convert to long
movf  n, w                ;(where n is a byte variable)
call  fpu_sendByte
```

Load Zero:

```
call  load_zero           ;load register 0 with 0.0
```

Load a long value directly in code:

```
selectB Value             ;select Value as B register
call  lwriteB             ;write 500,000 (7A120 hex) to register
movlw 0x00
call  fpu_sendByte
movlw 0x07
```

```
    call  fpu_sendByte
    movlw 0xA1
    call  fpu_sendByte
    movlw 0x20
    call  fpu_sendByte
```

The fastest operations occur when the uM-FPU registers are already loaded with values. In time critical portions of code floating point constants should be loaded beforehand to maximize the processing speed in the critical section. With fifteen registers available for storage on the uM-FPU, it is often possible to preload all of the required constant values. Since the load routines must send data to the uM-FPU for conversion, there is additional overhead associated with each type of load. The majority of the overhead is associated with the data transfer. The load_longByte routine transfers an additional 8-bit value, the load_longWord routine transfers two 8-bit values, and the lwriteA and lwriteB routines transfer four 8-bit values. Minimizing the amount of data transfer will maximize the execution speed of your program.

Comparing and Testing Long Integer Values

A long integer value can be zero, positive, or negative. The following symbols define the long status bits:

status_Zero	Zero bit	(0 – not zero, 1 – zero)
status_Sign	Sign bit	(0 – positive, 1 – negative)

The lstatus command is used to check the status of a long integer number. For example:

```
    call  lstatus            ;check status of A register
    btfsc status_Zero
    goto  zeroValue
    btfsc status_Sign
    goto  negativeValue

    ;value is positive
      …
negativeValue:
    ;value is negative
      …
zeroValue:
      …
```

The lcompare and lucompare commands are used to compare two long integer values. The status bits being set for the results of the operation A – B. (The selected A and B registers are not modified). lcompare does a signed compare and the lucompare does an unsigned compare. For example:

```
    call  lcompare           ;compare A and B registers
    btfsc status_Zero
    goto  sameAs
    btfsc status_Sign
    goto  lessThan

    ;A > B
      …
lessThan
    ;A < B"
      …
sameAs
    ;A = B
      …
```

Left and Right Parenthesis

Mathematical equations are often expressed with parenthesis to define the order of operations. For example Y = (X-1) / (X+1). The expressions inside the parentheses often need to be assigned to a temporary value before they can be used with other expressions in the equation. Temporary values are also useful to preserve the original value of a variable used in an equation. The left and right parenthesis functions provide a convenient means of allocating temporary values.

When a left parenthesis is issued, the current A register selection is saved and a new value is assigned that references a temporary register. Operations can now be performed as normal with the temporary register selected as the A register. When a right parenthesis is issued, the current value of the A register is copied to register 0, register 0 is selected as the B register, and the previous A register selection is restored. The register 0 value can be used immediately in subsequent operations. Up to five levels of parentheses can be used. The regA variable should not generally be changed by the user inside parentheses since regA is set automatically by the left and right functions.

In the example shown earlier for the equation Z = sqrt(X**2 + Y**2), the values of X and Y were modified during the calculation. Using parentheses, it's easy to implement the equation while retaining the original values of X and Y. For example:

```
#define Xvalue    1     ;X value (uM-FPU register 1)
#define Yvalue    2     ;Y value (uM-FPU register 2)
#define Zvalue    3     ;Z value (uM-FPU register 3)

;Z = sqrt(X**2 + Y**2)
;----------------------
selectA Zvalue          ;Zvalue = Xvalue ** 2
selectB Xvalue
call  fset
call  fmul

call  left              ;temp1 = Yvalue ** 2
selectB Yvalue
call  fset
call  fmul

call  right             ;Zvalue = Zvalue + temp1
call  fadd

call  sqrt              ;Zvalue = sqrt(Zvalue)
```

Another example:

```
;Y = 10 / (X + 1)
;----------------
selectA Yvalue          ;Yvalue = 10
call  load_floatByte
movlw .10
call  fpu_sendByte
call  fset

call  left              ;temp1 = Xvalue + 1
selectB Xvalue
call  fset
call  load_one
call  fadd

call  right             ;Yvalue = Yvalue / temp1
call  fdiv
```

Print routines

There are several print routines provided to display values by sending ASCII character strings to the serial port on the PIC. These routines could be used as templates to develop routines for other output devices (e.g. LCD screen).

print_float	send a floating point value to the serial port
print_floatFormat	send a formatted floating point value to the serial port
print_long	send a signed long integer to the serial port
print_longFormat	send a formatted long integer to the serial port
print_fpuString	send a string read from the uM-FPU to the serial port

The following examples assume that Angle contains the floating point value 3.1415927 and Total contains the long integer value –2000.

```
selectA Angle           ;select Angle as A register
call print_float        ;displays Angle in default float format
```
Value displayed: 3.1415927

```
movlw .64               ;display Angle in 6.4 float format
call print_floatFormat
```
Value displayed: 3.1416

```
selectA Total           ;select Total as A register
call print_long         ;displays Total in default long format
```
Value displayed: -2000

```
movlw .10               ;display Total in long format
call print_longFormat   ;signed, width of 10
```
Value displayed: -2000

```
movlw .110              ;display Total in long format
call print_longFormat   ;unsigned, width of 10
```
Value displayed: 4294965296

Additional general purpose print routines are also provided:

print_string	send a string read from ROM to the serial port
print_hex32	send a 32-bit hex string to the serial port
print_hex	send an 8-bit hex string to the serial port
print_hexDigit	send a 4-bit hex digit to the serial port
print_crlf	send a CRLF to the serial port
print_byte	send an 8-bit byte to the serial port

Sample Code

```
;-----------------------------------------------------------------------
;The following example takes an integer value representing the diameter
;of a circle in millimeters, converts the value to centimeters and
;calculates the circumference and area. For example, the inputValue
;could be a value read from a distance finding sensor. A description of
;each step of the calculations is provided
;-----------------------------------------------------------------------

      list  p=16f877
      #include <p16f877.inc>

      #include umfpu.inc       ;uM-FPU function definitions
      extern print_setup, print_string, print_floatFormat

;-------------------- uM-FPU register definitions ---------------------

#define  Diameter       4      ;diameter         (uM-FPU register 4)
#define  Circumference  5      ;circumference    (uM-FPU register 5)
#define  Area           6      ;area             (uM-FPU register 6)

;-------------------- variables ---------------------------------------
      udata
inputValue res 1               ;diameter in centimeters

;-------------------- string definitions ------------------------------
STRINGS     code
      global      stringTable
stringTable
      addwf PCL,f                    ;computed goto for strings

diameterMessage
      dt 0x0D, 0x0A, "Diameter (in.):      ", 0
circumferenceMessage
      dt 0x0D, 0x0A, "Circumference (in.): ", 0
areaMessage
      dt 0x0D, 0x0A, "Area (sq.in.):       ", 0

;-------------------- reset and interrupt vector ----------------------
STARTUP     code
      nop                            ;reset vector
      goto    main
      nop
      nop
      goto  isr                      ;interrupt vector

;-------------------- interrupt service routine -----------------------
PROG1 code
isr
      retfie                         ;(no interrupts used)
```

```
;=======================================================================
;=================== main routine ======================================
;=======================================================================

main
      call   print_setup        ;setup the serial port

      call   fpu_reset          ;reset the uM-FPU

      ;get input value
      ;---------------
      movlw .250                ;(e.g. read a sensor)
      movwf inputValue

      ;Diameter = inputValue / 10 (convert to centimeters)
      ;---------------------------------------------------

      selectA Diameter          ;select Diameter as A register
      call   load_floatUbyte    ;load unsigned byte value into Register 0
      movf   inputValue, w      ; and convert to floating point
      call   fpu_sendByte
      call   fset               ;Diameter = inputValue

      call   load_floatByte     ;load 10 into Register 0
      movlw .10                 ; and convert to floating point (10.0)
      call   fpu_sendByte
      call   fdiv               ;Diameter = Diameter / 10.0

      movlw LOW diameterMessage ;display diameter
      call   print_string
      movlw .92                 ;print as 9.2 floating point format
      call   print_floatFormat

      ;Circumference = Diameter * pi
      ;-----------------------------
      selectA Circumference     ;select Circumference as A register
      selectB Diameter          ;select Diameter as B register
      call   fset               ;Circumference = Diameter

      call   load_pi            ;load the value of pi into Register 0
      call   fmul               ;Circumference = Circumference * pi

      movlw LOW circumferenceMessage ;display circumference
      call   print_string
      movlw .92                 ;print as 9.2 floating point format
      call   print_floatFormat

      ;Area = (Diameter / 2)^2 * pi
      ;----------------------------
      selectA Area              ;select Area as register A
      selectB Diameter          ;select Diameter as B register
      call   fset               ;Area = Diameter

      call   load_floatByte     ;load 2 into Register 0
      movlw .2                  ; and convert to floating point (2.0)
      call   fpu_sendByte
```

```
call   fdiv                 ;Area = Area / 2.0
selectB Area                ;select Area as B register
call   fmul                 ;Area = Area * Area

call load_pi                ;load the value of pi into Register 0
call   fmul                 ;Area = Area * pi

movlw LOW areaMessage       ;display area
call   print_string
movlw .92                   ;print as 9.2 floating point format
call   print_floatFormat

end
```

Appendix A
Reference for uM-FPU PICmicro® routines

Initialization Routine

fpu_reset	Reset the uM-FPU

Data Transfer Routines

fpu_readByte	Get byte from the uM-FPU
fpu_sendByte	Send byte to the uM-FPU

Print Routines

print_float	Print free format floating point value
print_floatFormat	Print formatted floating point value
print_long	Print free format long value
print_longFormat	Print formatted long value

Variables used as parameters

dataByte	32-bit variable

Set by the following functions:

read	32-bit floating point value in dataByte to dataByte+3
sync	8-bit sync character in dataByte
fcompare	8-bit compare status byte in dataByte
fstatus	8-bit status byte in dataByte
lread	32-bit long integer value in dataByte to dataByte+3
lcompare	8-bit compare status byte in dataByte
lucompare	8-bit compare status byte in dataByte
lstatus	8-bit status byte in dataByte

Status Bits

status_Zero	Zero bit	(0 – not zero, 1 – zero)
status_Sign	Sign bit	(0 – positive, 1 – negative)
status_NaN	Not-a-Number	(0 – valid number, 1 – NaN)
status_Zero	Infinity	(0 – not infinite, 1 – infinite)

Initialization Routine

fpu_reset **Reset the uM-FPU**

Parameters: none

Return: none

Description: This routine must be called at the start of every application. The uM-FPU is reset to its startup condition and communication between the PIC and the uM-FPU is established. All uM-FPU registers are initialized to NaN (Not a Number) at reset, therefore any operation that uses a register before a value has been stored in the register will produce a result of NaN.

Example:

```
call  fpu_reset                  ;reset the uM-FPU coprocessor
```

Data Transfer Routines

---

fpu_readByte Read byte from the uM-FPU

Parameters: none

Return: `W register, dataByte` 8-bit value read from uM-FPU

Description: Reads an 8-bit value from the uM-FPU. This routine is used after a uM-FPU instruction that results in data being sent to the PIC.

Example:

```
call   readstr              ;setup to read string
call   fpu_readByte         ;read the next character
btfsc STATUS, Z             ;check for zero terminator
return                      ;yes, then exit
…
```

---

fpu_sendByte Send byte to the uM-FPU

Parameters: `W register` 8-bit value to send to uM-FPU

Return: none

Description: Sends an 8-bit value to the uM-FPU. This routine is used after a uM-FPU instruction that requires additional data.

Example:

```
inputValue res 1            ;8-bit variable

call   load_floatByte       ;load inputValue to Register 0
movf   inputValue, w        ; and convert to float
call   fpu_sendByte
```

Print Routines

---

print_float **Send a floating point value to the serial port**

Parameters: none

Return: none

Description: The floating point representation of the A register value is output to the serial port. Up to eight significant digits will be displayed if required. Very large or very small numbers are displayed in exponential notation. The length of the displayed value is variable and can be from 3 to 12 characters in length. The special cases of NaN (Not a Number), +infinity, -infinity, and -0.0 are handled. Examples of the display format are as follows:

1.0	NaN	0.0
10e20	Infinity	-0.0
3.1415927	-Infinity	1.0
-52.333334	-3.5e-5	0.01

Example:

```
call  print_float        ;print float value
```

---

print_floatFormat **Send a formatted floating point value to the serial port**

Parameters: `W register` format specification

Return: none

Description: The formatted floating point representation of the A register value is output to the serial port. The format is specified as a decimal value passed in the `W` register. The tens digit specifies the width of the display field and the ones digit specifies the number of decimal points. If the floating point value is too large for the format specified, then asterisks will be displayed. If the number of decimal points is zero, no decimal point will be displayed. Examples of the display format are as follows:

Value in register A	format	Display format
123.567	61 (6.1)	123.6
123.567	62 (6.2)	123.57
123.567	42 (4.2)	*.**
0.9999	20 (2.0)	1
0.9999	31 (3.1)	1.0

The maximum width of the field is 9 and the maximum number of decimal points is 6.

Example:

```
movlw .62                ;print float value with 6.2 format
call  print_floatFormat
```

---

print_long **Send a signed long integer value to the serial port**

Parameters: none

Return: none

Description: The signed long integer representation of the A register value is output to the serial port.. The length of the displayed value is variable and can range from 1 to 11 characters in length. Examples of the display format are as follows:

1
500000
-3598390

Example:

```
call  print_long         ;print long value
```

---

print_longFormat **Send a formatted long integer value to the PC screen**

Parameters: `W register` format specification

Return: none

Description: The formatted long integer representation of the A register value is output to the serial port. The format is specified as a decimal value passed in the `format` variable. A value between 0 and 15 specifies the width of the display field for a signed long integer. The number is displayed right justified. If 100 is added to the format value the value is displayed as an unsigned long integer. If the value is larger than the specified width, asterisks will be displayed. If the width is specified as zero, the length will be variable. Examples of the display format are as follows:

Value in register A	format		Display format
-1	10	(signed 10)	`        -1`
-1	110	(unsigned 10)	4294967295
-1	4	(signed 4)	`  -1`
-1	104	(unsigned 4)	****
0	4	(signed 4)	`   0`
0	0	(unformatted)	0
1000	6	(signed 6)	`  1000`

The maximum width of the field is 15.

Example:

```
movlw .10                ;print long value with width of 10
call  print_longFormat
```

---

print_fpuString **Send a string read from the uM-FPU to the serial port**

Parameters: `none`

Return: none

Description: A zero terminated string is read from the uM-FPU and sent to the serial port. (This function is used by the print_float, print_floatFormat, print_long, and print_longFormat routines.)

Example:

```
call  version                  ;get the version string
call  print_fpuString          ;print the version string
```

---

print_string **Send a string read from ROM to the serial port**

Parameters: `W register` low byte of string address

Return: none

Description: A section of ROM is reserved for stored up to 256 bytes string data. A zero terminated string is read from ROM and sent to the serial port.

Example:

```
movlw LOW message1             ;print message1
call  print_string
```

---

print_hex32 **Send a 32-bit hex string to the serial port**

Parameters: `dataByte to dataByte+3`

Return: none

Description: The 32-bit value in dataByte is sent to the serial port as a hexadecimal string.

Example:

```
call  read                     ;get floating point value
call  print_hex32              ;display as hex
```

---

print_hex **Send an 8-bit hex string to the serial port**

Parameters: `W register`

Return: none

Description: The 8-bit value in the W register is sent to the serial port as a hexadecimal string.

Example:

```
movlw 0xFF                     ;get 8-bit value
call  print_hex                ;display as hex
```

---

print_hexDigit Send a 4-bit hex digit to the serial port

Parameters: `W register`

Return: none

Description: The lower 4-bit value of the W register is sent to the serial port as a hexadecimal digit.

Example:

```
movlw  0x0A                     ;get 4-bit value
call   print_hexDigit           ;display as hex
```

---

print_crlf Send a CR, LF to the serial port

Parameters: none

Return: none

Description: A carriage return and linefeed character is sent to the serial port.

Example:

```
call   print_crlf               ;send CRLF
```

---

print_byte Send an 8-bit byte to the serial port

Parameters: `W register` 8-bit value

Return: none

Description: The 8-bit value contained in the W register is sent to the serial port.

Example:

```
movlw  'P'                      ;send P to serial port
call   print_byte
```

Appendix B
uM-FPU Opcode Summary

Opcode Name	Data Type	Opcode	Arguments	Returns	B Reg	Description
SELECTA		0x				Select A register
SELECTB		1x			x	Select B register
WRITEA	Either	2x	yyyy zzzz			Write register and select A
WRITEB	Either	3x	yyyy zzzz		x	Write register and select B
READ	Either	4x		yyyy zzzz		Read register
SET	Either	5x				A = B
FADD	Float	6x			x	A = A + B
FSUB	Float	7x			x	A = A - B
FMUL	Float	8x			x	A = A * B
FDIV	Float	9x			x	A = A / B
LADD	Long	Ax			x	A = A + B
LSUB	Long	Bx			x	A = A -B
LMUL	Long	Cx			x	A = A * B
LDIV	Long	Dx			x	A = A / B
SQRT	Float	E0				A = sqrt(A)
LOG	Float	E1				A = ln(A)
LOG10	Float	E2				A = log(A)
EXP	Float	E3				A = e ** A
EXP10	Float	E4				A = 10 ** A
SIN	Float	E5				A = sin(A) radians
COS	Float	E6				A = cos(A) radians
TAN	Float	E7				A = tan(A) radians
FLOOR	Float	E8				A = nearest integer <= A
CEIL	Float	E9				A = nearest integer >= A
ROUND	Float	EA				A = nearest integer to A
NEGATE	Float	EB				A = -A
ABS	Float	EC				A = |A|
INVERSE	Float	ED				A = 1 / A
DEGREES	Float	EE				Convert radians to degrees A = A / (PI / 180)
RADIANS	Float	EF				Convert degrees to radians A = A * (PI / 180)
SYNC		F0		5C		Synchronization
FLOAT	Long	F1			0	Copy A to Register 0 Convert long to float
FIX	Float	F2			0	Copy A to Register 0 Convert float to long
FCOMPARE	Float	F3		ss		Compare A and B (floating point)

Opcode Name	Data Type	Opcode	Arguments	Returns	B Reg	Description
LOADBYTE	Float	F4	bb		0	Write signed byte to Register 0 Convert to float
LOADUBYTE	Float	F5	bb		0	Write unsigned byte to Register 0 Convert to float
LOADWORD	Float	F6	wwww		0	Write signed word to Register 0 Convert to float
LOADUWORD	Float	F7	wwww		0	Write unsigned word to Register 0 Convert to float
READSTR		F8		aa … 00		Read zero terminated string from string buffer
ATOF	Float	F9	aa … 00		0	Convert ASCII to float Store in A
FTOA	Float	FA	ff			Convert float to ASCII Store in string buffer
ATOL	Long	FB	aa … 00		0	Convert ASCII to long Store in A
LTOA	Long	FC	ff			Convert long to ASCII Store in string buffer
FSTATUS	Float	FD		ss		Get floating point status of A
FUNCTION		FE0n				User functions 0-15
FUNCTION		FE1n				User functions 16-31
FUNCTION		FE2n				User functions 32-47
FUNCTION		FE3n				User functions 48-63
LWRITEA	Long	FEAx	yyyy zzzz			Write register and select A
LWRITEB	Long	FEBx	yyyy zzzz		0	Write register and select B
LREAD	Long	FECx		yyyy zzzz		Read register
LUDIV	Long	FEDx			0	A = A / B (unsigned long)
POWER	Float	FEE0				A = A ** B
ROOT	Float	FEE1				A = the Bth root of A
MIN	Float	FEE2				A = minimum of A and B
MAX	Float	FEE3				A = maximum of A and B
FRACTION	Float	FEE4			0	Load Register 0 with the fractional part of A
ASIN	Float	FEE5				A = asin(A) radians
ACOS	Float	FEE6				A = acos(A) radians
ATAN	Float	FEE7				A = atan(A) radians
ATAN2	Float	FEE8				A = atan(A/B)
LCOMPARE	Long	FEE9		ss		Compare A and B (signed long integer)
LUCOMPARE	Long	FEEA		ss		Compare A and B (unsigned long integer)
LSTATUS	Long	FEEB		ss		Get long status of A
LNEGATE	Long	FEEC				A = -A
LABS	Long	FEED				A = |A|
LEFT		FEEE				Right parenthesis
RIGHT		FEEF			0	Left parenthesis

Opcode Name	Data Type	Opcode	Arguments	Returns	B Reg	Description
LOADZERO	Either	FEF0			0	Load Register 0 with zero
LOADONE	Float	FEF1			0	Load Register 0 with 1.0
LOADE	Float	FEF2			0	Load Register 0 with e
LOADPI	Float	FEF3			0	Load Register 0with pi
LONGBYTE	Long	FEF4	bb		0	Write signed byte to Register 0 Convert to long
LONGUBYTE	Long	FEF5	bb		0	Write unsigned byte to Register 0 Convert to long
LONGWORD	Long	FEF6	wwww		0	Write signed word to Register 0 Convert to long
LONGUWORD	Long	FEF7	wwww		0	Write unsigned word to Register 0 Convert to long
IEEEMODE		FEF8				Set IEEE mode (default)
PICMODE		FEF9				Set PIC mode
BREAK		FEFB				Debug breakpoint
TRACEOFF		FEFC				Turn debug trace off
TRACEON		FEFD				Turn debug trace on
TRACESTR		FEFE				Send debug string to trace buffer
CHECKSUM		FEFF			0	Calculate code checksum
VERSION		FF				Copy version string to string buffer

Notes:

Data Type	data type required by opcode
Opcode	hexadecimal opcode value
Aruments	additional data required by opcode
Returns	data returned by opcode
B Reg	value of B register after opcode executes
x	register number (0-15)
n	function number (0-63)
yyyy	most significant 16 bits of 32-bit value
zzzz	least significant 16 bits of 32-bit value
ss	status byte
bb	8-bit value
wwww	16-bit value
aa … 00	zero terminated ASCII string

Appendix C
Floating Point Numbers

Floating point numbers can store both very large and very small values by “floating” the window of precision to fit the scale of the number. Fixed point numbers can’t handle very large or very small numbers and are prone to loss of precision when numbers are divided. The representation of floating point numbers used by the uM-FPU is defined by the IEEE 754 standard.

The range of numbers that can be handled by the uM-FPU is approximately $\pm 10^{38.53}$.

.

IEEE 754 32-bit Floating Point Representation

IEEE floating point numbers have three components: the sign, the exponent, and the mantissa. The sign indicates whether the number is positive or negative. The exponent has an implied base of two. The mantissa is composed of the fraction.

The 32-bit IEEE 754 representation is as follows:

S	Exponent	Mantissa
31	30 23	22 0

Sign Bit (S)
The sign bit is 0 for a positive number and 1 for a negative number.

Exponent
The exponent field is an 8-bit field that stores the value of the exponent with a bias of 127 that allows it to represent both positive and negative exponents. For example, if the exponent field is 128, it represents an exponent of one (128 – 127 = 1). An exponent field of all zeroes is used for denormalized numbers and an exponent field of all ones is used for the special numbers +infinity, -infinity and Not-a-Number (described below).

Mantissa
The mantissa is a 23-bit field that stores the precision bits of the number. For normalized numbers there is an implied leading bit equal to one.

Special Values

Zero

A zero value is represented by an exponent of zero and a mantissa of zero. Note that +0 and –0 are distinct values although they compare as equal.

Denormalized

If an exponent is all zeros, but the mantissa is non-zero the value is a denormalized number. Denormalized numbers are used to represent very small numbers and provide for an extended range and a graceful transition towards zero on underflows. Note: The uM-FPU does not support operations using denormalized numbers.

Infinity

The values +infinity and –infinity are denoted with an exponent of all ones and a fraction of all zeroes. The sign bit distinguishes between +infinity and –infinity. This allows operations to continue past an overflow. A nonzero number divided by zero will result in an infinity value.

Not A Number (NaN)

The value NaN is used to represent a value that does not represent a real number. An operation such as zero divided by zero will result in a value of NaN. The NaN value will flow through any mathematical operation. Note: The uM-FPU initializes all of its registers to NaN at reset, therefore any operation that uses a register that has not been previously set with a value will produce a result of NaN.

Some examples of IEEE 754 32-bit floating point values displayed as four byte values are as follows:

```
0x00, 0x00, 0x00, 0x00     ;0.0
0x3D, 0xCC, 0xCC, 0xCD     ;0.1
0x3F, 0x00, 0x00, 0x00     ;0.5
0x3F, 0x40, 0x00, 0x00     ;0.75
0x3F, 0x7F, 0xF9, 0x72     ;0.9999
0x3F, 0x80, 0x00, 0x00     ;1.0
0x40, 0x00, 0x00, 0x00     ;2.0
0x40, 0x2D, 0xF8, 0x54     ;2.7182818 (e)
0x40, 0x49, 0x0F, 0xDB     ;3.1415927 (pi)
0x41, 0x20, 0x00, 0x00     ;10.0
0x42, 0xC8, 0x00, 0x00     ;100.0
0x44, 0x7A, 0x00, 0x00     ;1000.0
0x44, 0x9A, 0x52, 0x2B     ;1234.5678
0x49, 0x74, 0x24, 0x00     ;1000000.0
0x80, 0x00, 0x00, 0x00     ;-0.0
0xBF, 0x80, 0x00, 0x00     ;-1.0
0xC1, 0x20, 0x00, 0x00     ;-10.0
0xC2, 0xC8, 0x00, 0x00     ;-100.0
0x7F, 0xC0, 0x00, 0x00     ;NaN (Not-a-Number)
0x7F, 0x80, 0x00, 0x00     ;+inf
0xFF, 0x80, 0x00, 0x00     ;-inf
```