Using the uM-FPU with the BASIC Stamp

Micromega Corporation

Introduction

The uM-FPU is a 32-bit floating point coprocessor that can be easily interfaced with the BASIC Stamp BS2, BS2e, BS2sx, BS2p24, BS2p40 or BS2pe to provide support for 32-bit IEEE 754 floating point operations and long integer operations. The uM-FPU is easy to connect, and requires only two pins on the BASIC Stamp. The only external component required for operation is a protection resistor on the bidirectional data line.

uM-FPU Features

- 8-pin integrated circuit.
- Bi-directional serial interface requires only two wires for connection.
- 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

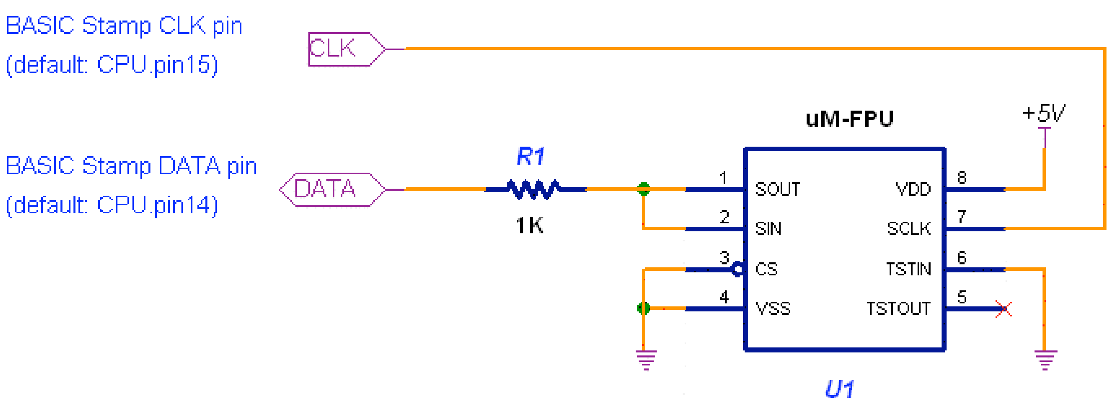
- Full set of BASIC Stamp support routines provided for easy implementation.

Connecting the uM-FPU to the BASIC Stamp

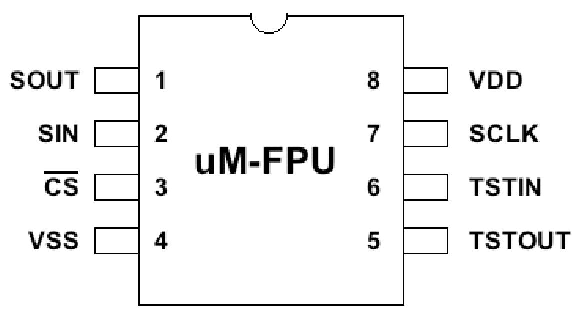
The uM-FPU requires just two pins for interfacing to the BASIC Stamp. The communication is implemented using a bidirectional serial interface that requires a clock pin and a data pin. The default setting for these pins are:

FpuClock	PIN	15
FpuData	PIN	14

The settings for these pins can be changed to suit your application. The support routines assume that the uM-FPU chip is always selected, so the FpuClock and FpuData pins should not be used for other connections as this will likely result in loss of synchronization between the BASIC Stamp and the uM-FPU coprocessor.



uM-FPU Pin Assignment



PIN DESCRIPTION

SOUT	SPI Output
SIN	SPI Input
$\overline{\text{CS}}$	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 support routines is provided to handle all of the communication between the BASIC Stamp and the uM-FPU. The template file uM-FPU.BS2 contains all of the definitions and support code. This file can be used directly as the starting point for a new program, or the definitions and support code can be copied from this file to another program. Each uM-FPU support routine is described in detail in a reference guide included as Appendix A of this document.

In order to ensure that the BASIC Stamp and the uM-FPU coprocessor are synchronized, a reset call must be done at the start of every program. The FPU_Reset routine resets the uM-FPU, confirms communications, and sets the variable fStatus to 1 if successful, or 0 if the reset fails. A sample reset call is included in the uM-FPU.BS2 file. An example of a typical reset is as follows:

```
GOSUB FPU_Reset       'reset the FPU hardware
IF fStatus = 0 THEN
  DEBUG "uM-FPU not detected."
  END
ENDIF
```

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:

Fset	A = B
Fadd	A = A + B
Fsqrt	A = sqrt(A)
Fsin	A = sin(A)

Any of the sixteen registers can be selected as the A or B registers. The variables fA and fB are used to select the A and B registers prior to calling one of the arithmetic routines. For example, the following code adds register 2 to register 1.

```
fA = 1                'select register 1 as the A register
fB = 2                'select register 2 as the B register
GOSUB Fadd            'A = A + B
```

Using constant definitions to provide meaningful names for the registers can create a more readable program.

```
Total CON 1           'total amount  (uM-FPU register 1)
Count CON 2           'current count (uM-FPU register 2)

fA = Result           'result = result + count
fB = Count
GOSUB Fadd
```

The following floating point routines are provided:

Fabs	A = |A|
Facos	A = acos (A)
Fadd	A = A + B
Fasin	A = asin(A)
Fatan	A = atan(A)
Fatan2	A = atan2(A)
Fceil	A = ceil(A)
Fcompare	Compare A and B
Fcos	A = cos(A)
Fdivide	A = A / B
Fexp	A = exp(A)
Fexp10	A = exp10(A)
Ffix	A = fix(B)
Ffloor	A = floor(A)
Fget	Get the value of A
FgetStatus	Get the floating point status of A
Finverse	A = 1 / A
Flog	A = log(A)
Flog10	A = log10(A)
Fmax	A = maximum of A and B
Fmin	A = minimum of A and B
Fmultiply	A = A * B
Fnegate	A = -A
Fpower	A = A to the power of B
Froot	A = the Bth root of A
Fround	A = round(A)
Fset	A = B
Fsin	A = sin(A)
Fsqrt	A = sqrt(A)
Fsubtract	A = A – B
Ftan	A = tan(A)
FtoDegrees	Convert radians to degrees
FtoRadians	Convert degrees to radians

Note: All of the floating point routines start with a capital F prefix.

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.

```
Xvalue CON 1            'X value (uM-FPU register 1)
Yvalue CON 2            'Y value (uM-FPU register 2)
Zvalue CON 3            'Z value (uM-FPU register 3)

  fA = Xvalue           'X = X ** 2
  fB = Xvalue
  GOSUB Fmultiply

  fA = Yvalue           'Y = Y ** 2
  fB = Yvalue
  GOSUB Fmultiply
```

```
fA = Zvalue           'Z = X + Y
fB = Xvalue
GOSUB Fset
fB = Yvalue
GOSUB Fadd

GOSUB Fsqrt           'Z = sqrt(Z)
```

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

```
fA = Result             'Result = sqrt(Value1 + Value2 + Value3)
fB = Value1
GOSUB Fset
fB = value2
GOSUB Fadd
fB = value3
GOSUB Fadd
GOSUB Fsqrt
```

Loading Floating Point Values

The PBASIC compiler does not provide support for floating point number syntax, so floating point values must be entered using alternate methods. A handy utility program called `uM-FPU Converter` is available to convert between 32-bit floating point values and hexadecimal values. There are several ways to load floating point values into the uM-FPU. Support routines 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_Float	Load floating point value directly in the code
Load_FloatData	Load floating point value from EEPROM
Load_FloatStr	Load ASCII string from EEPROM 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)

Each of these routines loads the floating point value into Register 0 and sets fB to 0. This is very convenient since it allows an arithmetic operation to follow immediately using the newly loaded value as the B value. For example:

Load an word value:

```
fA = Result
fLow = 20               'set fLow to the 16-bit integer value
GOSUB Load_FloatByte    'load the 8-bit integer value
GOSUB Fadd              'Result = Result + 20
```

Load a floating point value directly in code:

```
fA = Angle              'set fHigh is to the high 16-bits
fHigh = $41A0           ' of the floating point value 20.0
fLow = $0000            'set fLow is to the low 16-bits
GOSUB Load_Float        'load the floating point value
GOSUB Fset              'Angle = 20.0
```

Load floating point value from EEPROM:

```
Pi   DATA  $40, $49, $0F, $DB     'pi = 3.1415927

fA = Angle                'set fAddr to the EEPROM address
fAddr = Pi                ' of the floating point constant
GOSUB Load_FloatData      'load the floating point constant
GOSUB Fmultiply           'Angle = Angle * pi
```

Load ASCII string from EEPROM:

```
PiStr  DATA  "3.145927", 0  'zero terminated string

fA = Result               'set fAddr to the EEPROM address
fAddr = PiStr             ' of the zero terminated string
GOSUB Load_FloatStr       'load the floating point string
GOSUB Fset                'Result = 3.145927
```

Load Zero:

```
fA = Result
GOSUB Load_Zero
GOSUB Fset                'Result = 0.0
```

Load Pi:

```
fA = Result
GOSUB Load_Pi
GOSUB Fset                'Result = 3.1415927
```

In many cases it makes sense to load all the initial values for the uM-FPU registers at the start of the program or before a particular section of code. The FPU_Preload routine makes this easy to do. It takes the address of a preload vector as a parameter, and loads the uM-FPU with the specified values.
For example:

```
F0_75     CON   2       'constant 0.75   (uM-FPU register 2)
Result    CON   5       'result          (uM-FPU register 5)
E         CON   14      'constant E      (uM-FPU register 14)
F100_0    CON   15      'constant 100.0  (uM-FPU register 15)

PreloadVector   DATA  F0_75
                DATA  $3F, $40, $00, $00    '0.75
                DATA  E
                DATA  $40, $2D, $F8, $54    '2.7182818 (e)
                DATA  F100_0
                DATA  $42, $C8, $00, $00    '100.0
                DATA  0

fAddr = PreloadVector     'set fAddr to the EEPROM address
GOSUB FPU_Preload         ' of the preload vector and load values

fA = Result                'Result = ((Result * E) + .75) / 100.0
fB = E
GOSUB Fmultiply
fB = F0_75
GOSUB Fadd
fB = F100_0
```

```
GOSUB Fdivide
```

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_FloatByte routine transfers one 8-bit value, the Load_FloatWord routine transfers two 8-bit values, the Load_Float and Load_FloatData routines transfer four 8-bit values, and the Load_FloatStr routine transfers 8-bits for each character in the string (including the zero terminator). 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 zero, positive, negative, infinite or Not a Number (which occurs if an invalid operation is performed on a floating point value). To check the status of a floating point number the FgetStatus routine is used. The FgetStatus routine sets the fStatus variable with the status of the selected register. A bit definition is provided for each status bit in the fStatus variable. They are as follows:

fStatus_Zero	Zero status bit (0-not zero, 1-zero)
fStatus_Sign	Sign status bit (0-positive, 1-negative)
fStatus_NaN	Not a Number status bit (0-valid number, 1-NaN)
fStatus_Inf	Infinity status bit (0-not infinite, 1-infinite)

For example:

```
fA = Result
GOSUB FgetStatus
IF (fStatus_Sign = 1) THEN DEBUG "Result is negative"
IF (fStatus_Zero = 1) THEN DEBUG "Result is zero"
```

The Fcompare routine 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:

```
fA = Value1          'compare Value1 and Value2
fB = Value2
GOSUB Fcompare

IF (fStatus_Zero = 1) THEN
  DEBUG "Value1 = Value2"
ELSEIF (fStatus_Sign = 1) THEN
  DEBUG "Value1 < Value2"
ELSE
  DEBUG "Value1 > Value2"
ENDIF
```

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:

```
Total CON 1          'total amount  (uM-FPU register 1)
Count CON 2          'current count (uM-FPU register 2)

fA = Result          'result = result + count
fB = Count
GOSUB Ladd           '(long addition)
```

The following long integer routines are provided:

Labs	A = |A|
Ladd	A = A + B
Lcompare	Compare A and B
Ldivide	A = A / B
Lfloat	A = float(A)
Lget	Get the value of A
LgetStatus	Get the long integer status of A
Lmultiply	A = A * B
Lnegate	A = -A
Lset	A = B
Lsubtract	A = A – B
Lucompare	Compare A and B (unsigned)
Ludivide	A = A / B (unsigned)

Note: All of the long integer routines start with a capital L prefix.

Loading Long Integer Values

There are several ways to load long integer values into the uM-FPU. Support routines 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_Long	Load long integer values directly in the code
Load_LongData	Load long integer value from EEPROM
Load_LongStr	Load ASCII string from EEPROM and convert to long integer
Load_Zero	Load the long integer value 0

Each of these routines loads the long integer value into Register 0 and sets fB to 0. This is very convenient since it allows an arithmetic operation to follow immediately using the newly loaded value as the B value. For example:

Load an byte value:

```
fA = Result
fLow = 20                'set fLow to the 16-bit integer value
GOSUB Load_LongByte      'load the 8-bit integer value
GOSUB Ladd               'Result = Result + 20
```

Load a long integer value directly in code:

```
fA = Value               'set fHigh is to the high 16-bits
fHigh = $0007            ' of the long integer value 500,000
fLow = $A120             'set fLow is to the low 16-bits
GOSUB Load_Long          'load the floating point value
GOSUB Lset               'Value = 500000
```

Load long integer value from EEPROM:

```
L500K   DATA  $00, $07, $A1, $20     'constant 500,000

fA = Value               'set fAddr to the EEPROM address
fAddr = L500K            ' of the floating point constant
GOSUB Load_LongData      'load the floating point constant
GOSUB Lmultiply          'Value = Value * 500000
```

Load ASCII string from EEPROM:

```
L500KStr  DATA  "500000", 0  'zero terminated string

fA = Result              'set fAddr to the EEPROM address
fAddr = L500KStr         ' of the zero terminated string
GOSUB Load_LongStr       'load the floating point string
GOSUB Lset               'Result = 500000
```

Load Zero:

```
fA = Result
GOSUB Load_Zero
GOSUB Lset               'Result = 0
```

The FPU_Preload routine can also be used to load long integer values. See the reference guide for a full description.

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_FloatByte routine transfers one 8-bit value, the Load_FloatWord routine transfers two 8-bit values, the Load_Float and Load_FloatData routines transfer four 8-bit values, and the Load_FloatStr routine transfers 8-bits for each character in the string (including the zero terminator). 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. To check the status of a long integer number the LgetStatus routine is used. The LgetStatus routine sets the fStatus variable with the status of the selected register. A bit definition is provided for each status bit in the fStatus variable. They are as follows:

fStatus_Zero	Zero status bit (0-not zero, 1-zero)
fStatus_Sign	Sign status bit (0-positive, 1-negative)

For example:

```
fA = Result
GOSUB LgetStatus
IF (fStatus_Sign = 1) THEN DEBUG "Result is negative"
IF (fStatus_Zero = 1) THEN DEBUG "Result is zero"
```

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

```
fA = Value1          'compare Value1 and Value2
fB = Value2
GOSUB Lcompare

IF (fStatus_Zero = 1) THEN
  DEBUG "Value1 = Value2"
ELSEIF (fStatus_Sign = 1) THEN
  DEBUG "Value1 < Value2"
ELSE
  DEBUG "Value1 > Value2"
ENDIF
```

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 operators provide a convenient means of allocating temporary values.

When a left parenthesis is issued, the current value of fA 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, and fB is set to zero, and the previous value of fA is restored. The value can be used immediately in subsequent operatons. Up to five levels of parentheses can be used. In most situations, the fA variable should not be changed by the user's code inside parentheses since the value of fA is automatically set by the left and right parentheses operators.

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 parenthesis, it's easy to implement the equation while retaining the original values of X and Y. For example:

```
Xvalue CON 1            'X value (uM-FPU register 1)
Yvalue CON 2            'Y value (uM-FPU register 2)
Zvalue CON 3            'Z value (uM-FPU register 3)

  fA = Zvalue           'Z = X ** 2
  fB = Xvalue
  GOSUB Fset
  GOSUB Fmultiply

  GOSUB Left            'temp1 = Y ** 2
  fB = Yvalue
  GOSUB Fset
  GOSUB Fmultiply

  GOSUB Right           'Z = Z + temp1
  GOSUB Fadd

  GOSUB Fsqrt           'Z = sqrt(Z)
```

The following example shows Y = 10 / (X + 1):

```
Xvalue CON 1            'X value (uM-FPU register 1)
Yvalue CON 2            'Y value (uM-FPU register 2)

fA = Yvalue             'Y = 10
fLow = 10
GOSUB Load_FloatByte
GOSUB Fset

GOSUB Left              'temp1 = X + 1
fB = Xvalue
GOSUB Fset
GOSUB Load_One
GOSUB Fadd

GOSUB Right             'Y = Y / temp1
GOSUB Fdivide
```

Print routines

There are several print routines provided to display register values on the PC screen.

Print_Float	displays floating point value on the PC screen
Print_FloatFormat	dsiplays formatted floating point value on the PC screen
Print_Long	displays signed long integer on the PC screen
Print_LongFormat	displays formatted long integer on the PC screen
Print_Hex	displays 32-bit hexadecimal value on the PC screen
Print_Version	displays the uM-FPU version string on the PC screen

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

```
fA = Angle              'displays Angle in default float format
GOSUB Print_Float
```
Value displayed: 3.1415927

```
fA = Angle              'display Angle in 6.2 float format
Fformat = 62
GOSUB Print_FloatFormat
```
Value displayed: 3.1416

```
fA = Total              'displays Total in default long format
GOSUB Print_Long
```
Value displayed: -2000

```
fA = Total              'display Total in long format
Fformat = 10            'signed, width of 10
GOSUB Print_LongFormat
```
Value displayed: -2000

```
fA = Total              'display Total in long format
Fformat = 110           'unsigned, width of 10
GOSUB Print_LongFormat
```
Value displayed: 4294965296

```
fA = Angle              'display Angle in hex format
GOSUB Print_Hex
```
Value displayed: $4049 0FDB

```
GOSUB Print_Version     ‘display uM-FPU version
```
Value displayed: uM-FPU V1.0

Sample Code

```
'The following example takes an integer value representing the diameter
'of a circle in centimeters, converts the value to inches and
'calculates the circumference and area in inches and square inches.
'Note: the uM-FPU definitions and support routines are not shown.

'-------------------- constants ----------------------------------------

DiameterIn      CON   4   'diameter in inches (uM-FPU register 4)
Circumference   CON   5   'circumference      (uM-FPU register 5)
Area            CON   6   'area               (uM-FPU register 6)
Pi              CON   7   'constant pi        (uM-FPU register 7)
F2_0            CON   8   'constant 2.0       (uM-FPU register 8)
F2_54           CON   9   'constant 2.54      (uM-FPU register 9)

'-------------------- variables ----------------------------------------

diameterCm      VAR   Byte        'diameter in centimeters

'-------------------- EEPROM data --------------------------------------

PreloadVector   DATA  Pi
                DATA  $40, $49, $0F, $DB      'pi = 3.1415927
                DATA  F2_0
                DATA  $40, $00, $00, $00      '2.0
                DATA  F2_54
                DATA  $40, $22, $8F, $5C      '2.54
                DATA  0

'=======================================================================
'-------------------- main routine -------------------------------------
'=======================================================================

Main:

  DEBUG CR, CR, "Conversion Example"
  DEBUG CR, "------------------", CR

  GOSUB FPU_Reset                 'reset the FPU hardware
  IF fStatus = 0 THEN
    DEBUG "uM-FPU not detected."
    END
  ELSE
    GOSUB FPU_Version             'display the uM-FPU version number
    DEBUG CR
  ENDIF

  fAddr = PreloadVector           'load floating point initial values
  GOSUB FPU_Preload

  'diameter in centimeters is the input value
  '-----------------------------------------
  diameterCm = 25
  DEBUG CR, "Diameter (cm):       ", DEC diameterCm
```

```
'calculate diameter in inches: DiameterIn = diameterCm / 2.54
'-------------------------------------------------------------
fA = DiameterIn                 'convert integer value
fLow = diameterCm               ' to floating point
GOSUB Load_FloatByte
GOSUB Fset
fB = F2_54                      'divide by 2.54
GOSUB fDivide

DEBUG CR, "Diameter (in.):      "
GOSUB Print_Float               'display the diameter

'Circumference = DiameterIn * Pi
'---------------------------------------
fA = Circumference
fB = DiameterIn
GOSUB Fset
fB = Pi
GOSUB Fmultiply

DEBUG CR, "Circumference (in.): "
GOSUB Print_Float                'display the circumference

'Area = (DiameterIn / 2)**2 * Pi
'--------------------------------
fA = Area                        'Area = DiameterIn / 2.0
fB = DiameterIn
GOSUB Fset
fB = F2_0
GOSUB Fdivide

fB = fA
GOSUB Fmultiply                  'Area = Area * Area * Pi
fB = Pi
GOSUB Fmultiply

DEBUG CR, "Area (sq.in.):       "
GOSUB Print_Float                'display the area

DEBUG CR, CR, "Done.", CR        'end of program
END
```

Appendix A
Reference for uM-FPU BASIC Stamp routines

Initialization and Setup Routines

FPU_Reset	Reset the uM-FPU and confirm communications
FPU_Preload	Load the uM-FPU with initial values stored in EEPROM
FPU_Version	Display uM-FPU version string on the PC screen

Left and Right Parentheses

Left	Save A register and select new temporary register as A register
Right	Return value in register 0 and restore previous A register

Floating Point Routines

Fabs	A = |A|
Facos	A = acos (A)
Fadd	A = A + B
Fasin	A = asin(A)
Fatan	A = atan(A)
Fatan2	A = atan2(B/A)
Fceil	A = ceil(A)
Fcompare	Compare A and B
Fcos	A = cos(A)
Fdivide	A = A / B
Fexp	A = exp(A)
Fexp10	A = exp10(A)
Ffix	A = fix(B)
Ffloor	A = floor(A)
Fget	Get the value of A
FgetStatus	Get the status of A
Finverse	A = 1 / A
Flog	A = log(A)
Flog10	A = log10(A)
Fmax	A = maximum of A and B
Fmin	A = minimum of A and B
Fmultiply	A = A * B
Fnegate	A = -A
Fpower	A = A to the power of B
Froot	A = the Bth root of A
Fround	A = round(A)
Fset	A = B
Fsin	A = sin(A)
Fsqrt	A = sqrt(A)
Fsubtract	A = A – B
Ftan	A = tan(A)
FtoDegrees	Convert radians to degrees
FtoRadians	Convert degrees to radians

Long Integer Routines

Labs	A = |A|
Ladd	A = A + B
Lcompare	Compare A and B
Ldivide	A = A / B
Lfloat	A = float(A)
Lget	Get the value of A
LgetStatus	Get the long integer status
Lmultiply	A = A * B
Lnegate	A = -A
Lset	A = B
Lsubtract	A = A – B
Lucompare	Compare A and B (unsigned)
Ludivide	A = A / B (unsigned)

Load Routines

Load_E	Load register 0 with floating point value of e (2.7182818)
Load_Float	Load register 0 with floating point value
Load_FloatByte	Load register 0 with 8-bit signed integer converted to floating point
Load_FloatData	Load register 0 with floating point value in EEPROM
Load_FloatStr	Load register 0 with floating point string in EEPROM
Load_FloatUByte	Load register 0 with 8-bit unsigned integer converted to floating point
Load_FloatUWord	Load register 0 with 16-bit unsigned integer converted to floating point
Load_FloatWord	Load register 0 with 16-bit signed integer converted to floating point
Load_Fraction	Load register 0 with the fractional portion of A
Load_Long	Load register 0 with long integer value
Load_LongByte	Load register 0 with 8-bit signed integer converted to long integer
Load_LongData	Load register 0 with long integer value in EEPROM
Load_LongStr	Load register 0 with long integer string in EEPROM
Load_LongUByte	Load register 0 with 8-bit unsigned integer converted to long integer
Load_LongUWord	Load register 0 with 16-bit unsigned integer converted to long integer
Load_LongWord	Load register 0 with 16-bit signed integer converted to long integer
Load_One	Load register 0 with floating point value of 1.0
Load_Pi	Load register 0 with floating point value of Pi (3.1415927)
Load_Zero	Load register 0 with zero (long integer or floating point)

Print Routines

Print_Float	Display floating point value on the PC screen
Print_FloatFormat	Display formatted floating point value on the PC screen
Print_Hex	Display 32-bit hexadecimal value on the PC screen
Print_Long	Display signed long integer value on the PC screen
Print_LongFormat	Display formatted long integer value on the PC screen

Variables used as parameters

fA	Nib	Used to select the A register
fB	Nib	Used to select the B register
fHigh	Word	The high 16-bits of a local float or long value
fLow	Word	The low 16-bits of a local float or long value
fAddr	Word	EEPROM address for load routines
fFormat	Byte	Format variable for printing
fStatus	Byte	Contains the status from last FgetStatus or Fcompare call

Status Bits

fStatus_Zero	Zero status bit (0-not zero, 1-zero)
fStatus_Sign	Sign status bit (0-positive, 1-negative)
fStatus_NaN	Not a Number status bit (0-valid number, 1-NaN)
fStatus_Inf	Infinity status bit (0-not infinite, 1-infinite)

Initialization and Setup Routines

FPU_Reset **Reset the uM-FPU and confirm communications.**

Parameters: none
Return: fStatus = 0 successful reset
fStatus = 1 reset failed

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 BASIC Stamp and the uM-FPU is confirmed. 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:

```
  GOSUB FPU_Reset       'reset the FPU hardware
  IF fStatus = 0 THEN
    DEBUG "uM-FPU not detected."
    END
  ENDIF
```

FPU_Preload **Load the uM-FPU with initial values stored in EEPROM.**

Parameters: fAddr address of preload vector
Return: none

Description: This routine provides a quick way to load the uM-FPU registers with constants or initial values for variables. The preload vector is stored in EEPROM and has the following format:

```
      DATA  reg, byte1, byte2, byte3, byte4
      DATA  reg, byte1, byte2, byte3, byte4
      DATA  0
```

`reg` uM-FPU register number (1 to 15)
`byte1` to `byte2` 32-bit value (byte1 is most significant byte)

Note: A zero terminator is required at the end of the preload vector.

Example:

```
Pi    CON   1          'constant pi   (uM-FPU register 1)
F2_0  CON   3          'constant 2.0  (uM-FPU register 3)
F2_54 CON   9          'constant 2.54 (uM-FPU register 9)

PreloadVector  DATA  Pi
               DATA  $40, $49, $0F, $DB  '3.1415927
               DATA  F2_0
               DATA  $40, $00, $00, $00  '2.0
               DATA  F2_54
               DATA  $40, $22, $8F, $5C  '2.54
               DATA  0

  fAddr = PreloadVector  'set fAddr to the EEPROM address
  GOSUB FPU_Preload      'load the values
```

FPU_Version Display uM-FPU version string on the PC screen.

Parameters: none
Return: none

Description: The uM-FPU version string is read from the uM-FPU and output to the PC screen using the DEBUG command.

Example:

```
GOSUB FPU_Version    'display the uM-FPU version string
```

Left and Right Parentheses

Left **Left Parenthesis**

Parameters: none
Return: fA set to temporary register number

Description: The left parenthesis command saves the current value of fA, allocates the next temporary register, and sets fA to that register number.

Special cases: • the maximum number of temporary registers is five. If the maximum number is exceeded, the value of register A is set to NaN ($7FC00000).

Example: (see below)

Right **Right Parenthesis**

Parameters: none
Return: Register 0 last temporary value
fB set to 0

Description: The right parenthesis command copies the value of the current temporary register to register 0, and sets fB to 0. If this is the last right parenthesis, fA is set to the value before the first left parenthesis, otherwise it is set to the previous temporary register number.

Special case: • if no left parenthesis is currently outstanding, then the value of register 0 is set to NaN. ($7FC00000).

Example:

```
Xvalue       CON   7   'X value (uM-FPU register 7)
YValue       CON   8   'Y value (uM-FPU register 8)
Zvalue       CON   9   'Z value (uM-FPU register 9)

  'calculate Z = sqrt(X ** 2 + Y ** 2)
  fA = Zvalue          'fA = Zvalue
  GOSUB Left           'fA = Temp1
  fB = Xvalue          'Temp1 = Xvalue * Xvalue
  GOSUB Fset
  GOSUB Fmultiply
  GOSUB Left           'fA = Temp2
  fB = Yvalue          'Temp2 = Yvalue * Yvalue
  GOSUB Fset
  GOSUB Fmultiply
  GOSUB Right          'fA = Temp1, fB = 0, reg[0] = Temp2
  GOSUB Fadd           'Temp1 = Temp1 + Temp2
  GOSUB Right          'fA = Zvalue, fB = 0, reg[0] = Temp1
  GOSUB Fset           'Zvalue = sqrt(Temp1)
  GOSUB Fsqrt
```

Floating Point Routines

Fabs **A = |A|**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the absolute value of the floating point value in register A, and stores the result in register A.

Special case: • if the value is NaN, then the result is NaN

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = |Value|
  GOSUB Fabs
```

Facos **A = acos(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Returns the arc cosine of an angle, in the range of 0.0 through pi.

Special case: • if the value is NaN or its absolute value is greater than 1, then the result is NaN

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = Facos(Value)
  GOSUB Facos
```

Fadd **A = A + B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: The floating point value in register B is added to the floating point value in register A and the result is stored in register A.

Special cases:
- if either value is NaN, then the result is NaN
- if one value is +infinity and the other is –infinity, then the result is NaN
- if one value is +infinity and the other is not –infinity, then the result is +infinity
- if one value is -infinity and the other is not +infinity, then the result is -infinity

Example:

```
Pi    CON   1           'constant pi   (uM-FPU register 1)
Angle CON   5           'current angle (uM-FPU register 5)

  fA = Angle            'Angle = Angle + Pi
  fB = Pi
  GOSUB Fadd
```

Fasin **A = asin(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Returns the arc sine of an angle, in the range of –pi/2 through pi/2.

Special cases:
- if the value is NaN or its absolute value is greater than 1, then the result is NaN
- if the value is 0.0, then the result is a 0.0
- if the value is –0.0, then the result is –0.0

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = asin(Value)
  GOSUB Fasin
```

Fatan **A = atan(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Returns the arc tangent of an angle, in the range of –pi/2 through pi/2 radians.

Special cases:
- if the value is NaN, then the result is NaN
- if the value is 0.0, then the result is a 0.0
- if the value is –0.0, then the result is –0.0

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = cos(Value)
  GOSUB Fcos
```

Fatan2 **A = atan2(B/A)**

Parameters: fA uM-FPU register number
Return: none

Description: Converts rectangular coordinates (A, B) to polar (r, theta). The value of theta is returned in register A and is determined by computing the arc tangent of the value of register B divided by the value of register A, in the range –pi to pi.

Special cases:
- if either value is NaN, then the result is NaN
- if B is 0.0 and A > 0, then the result is 0.0
- if B > 0 and finite, and A is +inf, then the result is 0.0
- if B is –0.0 and A > 0, then the result is –0.0
- if B < 0 and finite, and A is +inf, then the result is –0.0
- ifB is 0.0 and A < 0, then the result is pi
- if B > 0 and finite, and A is –inf, then the result is pi
- if B is –0.0, and A < 0, then the result is –pi
- if B < 0 and finite, and A is –inf, then the result is –pi
- if B > 0, and A is 0.0 or –0.0, then the result is pi/2

• if B is +inf, and A is finite, then the result is pi/2
• if B < 0, and A is 0.0 or –0.0, then the result is –pi/2
• if B is –inf, and A is finite, then the result is –pi/2
• if B is +inf, and A is +inf, then the result is pi/4
• if B is +inf, and A is –inf, then the result is 3*pi/4
• if B is –inf, and A is +inf, then the result is –pi/4
• if B is –inf, and A is –inf, then the result ir –3*pi/4

Example:

```
Val1  CON   5          'current value (uM-FPU register 5)
Val2  CON   5          'current value (uM-FPU register 5)
  fA = Val1            'Val1 = atan(Val2/Val1)
  fB = Val2
  GOSUB Fatan2
```

Fceil **A = ceil(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the floating point value equal to the nearest integer that is greater than or equal to the floating point value in register A. The result is stored in register A.

Special cases:
• if the value is NaN, then the result is NaN
• if the value is +infinity or -infinity, then the result is +infinity or -infinity
• if the value is 0.0 or –0.0, then the result is 0.0 or –0.0
• if the value is less than zero but greater than –1.0, then the result is –0.0

Example:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'Value = ceil(Value)
  GOSUB Fceil
```

Fcompare **Compare A and B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: fStatus set to the status of the comparison

Description: Compare the floating point value in register A with the floating point value in register B. The status of the result is stored in the fStatus variable. The status is positive if the value in register A is greater than the value in register B. The status is negative if the value in register A is less than the value in register B. The status is zero if the value in register A is equal to the value in register B. Bit definitions are used to test the status as follows:

fStatus_Zero Zero status bit (0 if A not equal to B; 1 if A equals B)
fStatus_Sign Sign status bit (0 if A is greater than B; 1 if A is less than B)
fStatus_NaN NaN status bit (0 if valid numbers; 1 if A or B is Not-a-Number)

Special case: • if either value is NaN, then the result is NaN

Example:

```
Value1   CON   6        'First value  (uM-FPU register 6)
Value2   CON   7        'Second value (uM-FPU register 7)

  fA = Value1           'compare Value1 and Value2
  fB = Value2
  GOSUB Fcompare

  IF (Fstatus_Zero = 1) THEN
    DEBUG "Value1 = Value2"
  ELSEIF (Fstatus_Sign = 1) THEN
    DEBUG "Value1 < Value2"
  ELSE
    DEBUG "Value1 > Value2"
  ENDIF
```

Fcos **A = cos(A)**

Parameters: fA uM-FPU register number
Return: none

Description: When this routine is called, register A should contain a floating point value representing the angle in radians. The cosine of the angle is calculated and the result is stored in register A.

Special case:
- if the value is NaN or an infinity, then the result is NaN

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = cos(Value)
  GOSUB Fcos
```

Fdivide **A = A / B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: The floating point value in register A is divided by the floating point value in register B and the result is stored in register A.

Special cases:
- if either value is NaN, then the result is NaN
- if both values are zero or both values are infinity, then the result is NaN
- if the B value is zero and the A value is not zero, then the result is infinity
- if the B value is infinity, then the result is zero

Example:

```
Angle CON   5           'current angle (uM-FPU register 5)
F2_0  CON   3           'constant 2.0  (uM-FPU register 6)

  fA = Angle            'Angle = Angle / 2.0
  fB = F2_0
  GOSUB Fdivide
```

Fexp **A = exp(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the value of e (2.7182818) raised to the power of the floating point value in register A. The result is stored in A.

Special cases:
- if the value is NaN, then the result is NaN
- if the value is +infinity or greater than 88, then the result is +infinity
- if the value is –infinity or less than -88, then the result is 0.0

Example:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'Value = exp(Value)
  GOSUB Fexp
```

Fexp10 **A = exp10(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the value of 10 raised to the power of the floating point value in register A. The result is stored in A.

Special cases:
- if the value is NaN, then the result is NaN
- if the value is +infinity or greater than 38, then the result is +infinity
- if the value is –infinity or less than -38, then the result is 0.0

Example:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'Value = exp10(Value)
  GOSUB Fexp10
```

Ffix **A = fix(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Converts the floating point value in register A to a long integer value and stores the result in register A.

Special cases:
- if the value is NaN, then the result is zero
- if the value is +infinity or greater than the maximum signed long integer, then the result is the maximum signed long integer (decimal: 2147483647, hex: $7FFFFFFF)
- if the value is –infinity or less than the minimum signed long integer, then the result is the minimum signed long integer (decimal: -2147483648, hex: $80000000)

Example:

```
Value    CON   3      'current value  (uM-FPU register 3)

  fA = Value          'Value contains floating point
  GOSUB Ffloat        'Value is converted to long integer
```

Ffloor **A = floor(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the floating point value equal to the nearest integer that is less than or equal to the floating point value in register A. The result is stored in register A.

Special cases:
- if the value is NaN, then the result is NaN
- if the value is +infinity or -infinity, then the result is +infinity or -infinity
- if the value is 0.0 or –0.0, then the result is 0.0 or –0.0

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = floor(Value)
  GOSUB Ffloor
```

Fget **Get the floating point value of the A register**

Parameters: fA uM-FPU register number
Return: fHigh high 16 bits of the floating point value in register A
fLow low 16 bits of the floating point value in register A

Description: The high 16 bits of the floating point value in register A are returned in fHigh and the low 16 bits are returned in fLow.

Example:

```
Angle CON   5           'current angle (uM-FPU register 5)

  fA = Angle            'get the value of Angle
  GOSUB Fget
```

FgetStatus **Get the floating point status of A**

Parameters: fA uM-FPU register number
Return: fStatus set to the floating point status of the value in register A

Description: Get the status of the floating point value in register A. The fStatus variable is set to the status of the value. Four bit definitions are used to test the status as follows:

fStatus_Zero Zero status bit (0 – not zero; 1 – zero)
fStatus_Sign Sign status bit (0 – positive; 1 – negative)
fStatus_NaN NaN status bit (0 – valid number; 1 – Not-a-Number)
fStatus_Inf Infinity status bit (0 – not infinite; 1 – infinite)

Example:

```
Value    CON   3       'current value  (uM-FPU register 3)

  fA = Value           'get the status
  GOSUB FgetStatus

  IF (fStatus_NaN = 1) THEN DEBUG "Value is NaN"
  IF (fStatus_Inf = 1) THEN DEBUG "Value is infinite"
  IF (fStatus_Zero = 1) THEN DEBUG "Value is zero"
  IF (fStatus_Sign = 1) THEN DEBUG "Value is negative"
```

Finverse **A = 1 / A**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the inverse of the floating point value in register A, and stores the result in register A.

Special cases:
- if the value is NaN, then the result is NaN
- if the value is zero, then the result is infinity
- if the value is infinity, then the result is zero

Example:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'Value = 1 / Value
  GOSUB Finverse
```

Flog **A = log(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the natural log of the floating point value in register A. The result is stored in register A. The number e (2.7182818) is the base of the natural system of logarithms.

Special cases:
- if the value is NaN or less than zero, then the result is NaN
- if the value is +infinity, then the result is +infinity
- if the value is 0.0 or –0.0, then the result is -infinity

Example:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'Value = log(Value)
  GOSUB Flog
```

Flog10 **A = log10(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the base 10 logarithm of the floating point value in register A. The result is stored in register A.

Special cases: • if the value is NaN or less than zero, then the result is NaN
• if the value is +infinity, then the result is +infinity
• if the value is 0.0 or –0.0, then the result is -infinity

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = log10(Value)
  GOSUB Flog10
```

Fmax **A = maximum of A and B**

Parameters: fA uM-FPU register number
fB uM-FPU register number

Return: none

Description: Sets the value of register A to the maximum value of register A and register B.

Special cases: • if either value is NaN, then the result is NaN

Example:

```
Value CON   5           'current value (uM-FPU register 5)
F2_0  CON   6           'constant 2.0  (uM-FPU register 6)

  fA = Value            'Value = maximum of Value and 2.0
  fB = F2_0
  GOSUB Fmax
```

Fmin **A = minimum of A and B**

Parameters: fA uM-FPU register number
fB uM-FPU register number

Return: none

Description: Sets the value of register A to the minimum value of register A and register B.

Special cases: • if either value is NaN, then the result is NaN

Example:

```
Value CON   5           'current value (uM-FPU register 5)
F2_0  CON   6           'constant 2.0  (uM-FPU register 6)

  fA = Value            'Value = minimum of Value and 2.0
  fB = F2_0
  GOSUB Fmin
```

Fmultiply **A = A * B**

Parameters: fA uM-FPU register number
fB uM-FPU register number

Return: none

Description: The floating point value in register A is multiplied by the floating point value in register B and the result is stored in register A.

Special cases: • if either value is NaN, or one value is zero and the other is infinity, then the result is NaN
• if either values is infinity and the other is nonzero, then the result is infinity

Example:

```
Angle CON   5           'current angle (uM-FPU register 5)
F2_0  CON   6           'constant 2.0  (uM-FPU register 6)

  fA = Angle            'Angle = Angle * 2.0
  fB = F2_0
  GOSUB Fmultiply
```

Fnegate **A = -A**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the negative of the floating point value in register A, and stores the result in register A.

Special case: • if the value is NaN, then the result is NaN

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = -Value
  GOSUB Fnegate
```

Fpower **A = A to the power of B**

Parameters: fA uM-FPU register number (base)
fB uM-FPU register number (exponent)
Return: none

Description: Calculates the value of the floating point value in register A raised to the power of the floating point value in register B. The result is stored in register A.

Special cases:
- if B is 0.0 or –0.0, then the result is 1.0
- if B is 1.0, then the result is the same as the A value
- if B is NaN, then the result is Nan
- if A is NaN and B is nonzero, then the result is NaN
- if |A| > 1 and B is +infinite, then the result is +infinity
- if |A| < 1 and B is -infinite, then the result is +infinity
- if |A| > 1 and B is -infinite, then the result is 0.0
- if |A| < 1 and B is +infinite, then the result is 0.0
- if |A| = 1 and B is infinite, then the result is NaN
- if A is 0.0 and B > 0, then the result is 0.0
- if A is +infinity and B < 0, then the result is 0.0
- if A is 0.0 and B < 0, then the result is +infinity
- if A is +infinity and B > 0, then the result is +infinity
- if A is -0.0 and B > 0 but not a finite odd integer, then the result is 0.0
- if the A is -infinity and B < 0 but not a finite odd integer, then the result is 0.0
- if A is -0.0 and the B is a positive finite odd integer, then the result is –0.0
- if A is -infinity and B is a negative finite odd integer, then the result is –0.0

• if A is -0.0 and B < 0 but not a finite odd integer, then the result is +infinity
• if A is -infinity and B > 0 but not a finite odd integer, then the result is +infinity
• if A is -0.0 and B is a negative finite odd integer, then the result is –infinity
• if A is -infinity and B is a positive finite odd integer, then the result is –infinity
• if A < 0 and B is a finite even integer,
then the result is equal to |A| to the power of B
• if A < 0 and B is a finite odd integer,
then the result is equal to the negative of |A| to the power of B
• if A < 0 and finite and B is finite and not an integer, then the result is NaN

Example:

```
Value    CON   3      'current value  (uM-FPU register 3)
Exponent CON   4      'exponent value (uM-FPU register 4)

  fA = Value          'Value = Value ** Exponent
  fB = Exponent
  GOSUB Fpower
```

Froot **A = the Bth root of A**

Parameters: fA uM-FPU register number (base)
fB uM-FPU register number (root)
Return: none

Description: Calculates the value of the root of the floating point value in register A. The root is specified by the floating point value in register B. It is equivalent to raising A to the power of (1/B). The result is stored in register A.

Special cases:
• see the description in Fpower for the special cases of (1/B)
• if B is infinity, then (1/B) is zero
• if B is zero, then (1/B) is infinity

Example:

```
Value    CON   3      'current value  (uM-FPU register 3)
Root     CON   4      'exponent value (uM-FPU register 4)

  fA = Value          'Value = the Root of Value
  fB = Root
  GOSUB Froot
```

Fround **A = round(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the floating point value that is equal to the nearest integer to the floating point value in register A. The result is stored in register A.

Special cases:
• if the value is NaN, then the result is NaN
• if the value is +infinity or -infinity, then the result is +infinity or -infinity
• if the value is 0.0 or –0.0, then the result is 0.0 or –0.0

Example:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'Value = round(Value)
  GOSUB Fround
```

Fset **A = B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: Sets the value of register A to the value of register B.

Example:

```
Pi    CON   1           'constant pi   (uM-FPU register 1)
Angle CON   5           'current angle (uM-FPU register 5)

  fA = Angle            'Angle = Pi
  fB = Pi
  GOSUB Fset
```

Fsin **A = sin(A)**

Parameters: fA uM-FPU register number
Return: none

Description: When this routine is called, register A should contain a floating point value representing the angle in radians. The sine of the angle is calculated and the result is stored in register A.

Special cases:
• if the value is NaN or an infinity, then the result is NaN
• if the value is 0.0, then the result is 0.0
• if the value is –0.0, then the result is –0.0

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = sin(Value)
  GOSUB Fsin
```

Fsqrt **A = sqrt(A)**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the square root of the floating point value in register A. The result is stored in register A.

Special cases:
• if the value is NaN or less than zero, then the result is NaN
• if the value is +infinity, then the result is +infinity
• if the value is 0.0 or –0.0, then the result is 0.0 or –0.0

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = sqrt(Value)
  GOSUB Fsqrt
```

Fsubtract **A = A – B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: The floating point value in register B is subtracted from the floating point value in register A and the result is stored in register A.

Special cases:
- if either value is NaN, then the result is NaN
- if both values are infinity and the same sign, then the result is NaN
- if the A value is +infinity and the B value not +infinity, then the result is +infinity
- if the A value is -infinity and the B value not -infinity, then the result is -infinity
- if the A value is not an infinity and the B value is an infinity, then the result is an infinity of the opposite sign as the B value

Example:

```
Pi    CON   1           'constant pi   (uM-FPU register 1)
Angle CON   5           'current angle (uM-FPU register 5)

  fA = Angle            'Angle = Angle - Pi
  fB = Pi
  GOSUB Fsubtract
```

Ftan **A = tan(A)**

Parameters: fA uM-FPU register number
Return: none

Description: When this routine is called, register A should contain a floating point value representing the angle in radians. The tangent of the angle is calculated and the result is stored in register A.

Special cases:
- if the value is NaN or an infinity, then the result is NaN
- if the value is 0.0, then the result is 0.0
- if the value is –0.0, then the result is –0.0

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = tan(Value)
  GOSUB Ftan
```

FtoDegrees **Convert radians to degrees**

Parameters: fA uM-FPU register number
Return: none

Description: Converts the floating point value in register A from radians to degrees. The result is stored in register A.

Special case: • if the value is NaN, then the result is NaN

Example:

```
Angle CON   5           'current angle (uM-FPU register 5)

  fA = Angle            'Angle = Angle in radians
  GOSUB FtoDegrees      'convert Angle to degrees
```

FtoRadians **Convert degrees to radians**

Parameters: fA uM-FPU register number
Return: none

Description: Converts the floating point value in register A from degrees to radians. The result is stored in register A.

Special case: • if the value is NaN, then the result is NaN

Example:

```
Angle CON   5           'current angle (uM-FPU register 5)

  fA = Angle            'Angle = Angle in degrees
  GOSUB FtoRadians      'convert Angle to radians
```

Long Integer Routines

Labs **A = |A|**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the absolute value of the long integer value in register A, and stores the result in register A.

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = |Value|
  GOSUB Labs
```

Ladd **A = A + B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: The long integer value in register B is added to the long integer value in register A and the result is stored in register A.

Example:

```
Total  CON   5          'total         (uM-FPU register 5)
Value  CON   6          'current value (uM-FPU register 6)

  fA = Total            'Total = Total + Value
  fB = Value
  GOSUB Ladd
```

Lcompare **Compare A and B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: fStatus set to the status of the comparison

Description: Compare the long integer value in register A with the long integer value in register B. The status of the result is stored in the fStatus variable. The status is positive if the value in register A is greater than the value in register B. The status is negative if the value in register A is less than the value in register B. The status is zero if the value in register A is equal to the value in register B. Bit definitions are used to test the status as follows:

fStatus_Zero Zero status bit (0 if A not equal to B; 1 if A equals B)
fStatus_Sign Sign status bit (0 if A is greater than B; 1 if A is less than B)

Special case: • if either value is NaN, then the result is NaN

Example:

```
Value1   CON   6        'First value  (uM-FPU register 6)
Value2   CON   7        'Second value (uM-FPU register 7)

  fA = Value1           'compare Value1 and Value2
  fB = Value2
  GOSUB Lcompare

  IF (Lstatus_Zero = 1) THEN
    DEBUG "Value1 = Value2"
  ELSEIF (Lstatus_Sign = 1) THEN
    DEBUG "Value1 < Value2"
  ELSE
    DEBUG "Value1 > Value2"
  ENDIF
```

Ldivide **A = A / B**

Parameters: fA uM-FPU register number
fB uM-FPU register number

Return: none

Description: The long integer value in register A is divided by the long integer value in register B and the result is stored in register A. If the value in register B is zero (divide by zero), register A will be set to the largest positive long integer ($3FFFFFFF). The remainder of the division is stored in register 0.

Example:

```
Total  CON   5          'total         (uM-FPU register 5)
Value  CON   6          'current value (uM-FPU register 6)
Rem    CON   7          'remainder     (uM-FPU register 7)

  fA = Total            'Total = Total / Value
  fB = Value
  GOSUB Ldivide

  fA = Rem              'set Rem to the remainder
  fB = 0
  GOSUB Lset
```

Lfloat **A = float(A)**

Parameters: fA uM-FPU register number

Return: none

Description: Converts the long integer value in register A to a floating point value and stores the result in register A.

Example:

```
Value    CON   3       'current value  (uM-FPU register 3)

  fA = Value           'Value contains long integer
  GOSUB Ffloat         'Value is converted to floating point
```

Lget **Get the long integer value of the A register.**

Parameters: fA uM-FPU register number
Return: fHigh high 16 bits of the long integer value in register A
fLow low 16 bits of the long integer value in register A

Description: The high 16 bits of the long integer value in register A are returned in fHigh and the low 16 bits are returned in fLow.

Example:

```
Total   CON   5          'total           (uM-FPU register 5)

  fA = Total             'get the value of Total
  GOSUB Lget
```

LgetStatus **Get the long integer status of A**

Parameters: fA uM-FPU register number
Return: fStatus set to the status of the value in register A

Description: Get the status of the long integer value in register A. The fStatus variable is set to the status of the value. Four bit definitions are used to test the status as follows:

fStatus_Zero Zero status bit (0 – not zero; 1 – zero)
fStatus_Sign Sign status bit (0 – positive; 1 – negative)

Example:

```
Value    CON   3       'current value  (uM-FPU register 3)

  fA = Value           'get the status
  GOSUB LgetStatus

  IF (fStatus_Zero = 1) THEN DEBUG "Value is zero"
  IF (fStatus_Sign = 1) THEN DEBUG "Value is negative"
```

Lmultiply **A = A * B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: The long integer value in register A is multiplied by the long integer value in register B and the result is stored in register A.

Example:

```
Total   CON   5          'total         (uM-FPU register 5)
Value   CON   6          'current value (uM-FPU register 6)

  fA = Total             'Total = Total * Value
  fB = Value
  GOSUB Lmultiply
```

Lnegate **A = -A**

Parameters: fA uM-FPU register number
Return: none

Description: Calculates the negative of the long integer value in register A, and stores the result in register A.

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'Value = -Value
  GOSUB Fnegate
```

Lset **A = B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: Sets the value of register A to the value of register B.

Example:

```
L100   CON   1          'constant 500000 (uM-FPU register 1)
Total  CON   5          'total           (uM-FPU register 5)

  fA = Total            'total = 500000
  fB = L100
  GOSUB Lset
```

Lsubtract **A = A – B**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: none

Description: The long integer value in register B is subtracted from the long integer value in register A and the result is stored in register A.

Example:

```
Total  CON   5          'total         (uM-FPU register 5)
Value  CON   6          'current value (uM-FPU register 6)

  fA = Total            'Total = Total - Value
  fB = Value
  GOSUB Lsubtract
```

Lucompare **Compare A and B (unsigned)**

Parameters: fA uM-FPU register number
fB uM-FPU register number
Return: fStatus set to the status of the comparison

Description: Compare the unsigned long integer value in register A with the unsigned long integer value in register B. The status of the result is stored in the fStatus variable. The status is positive if the value in register A is greater than the value in register B. The status is negative if the value in register A is less than the value in register B. The status is zero if the value in register A is equal to the value in register B. Bit definitions are used to test the status as follows:

fStatus_Zero	Zero status bit (0 if A not equal to B; 1 if A equals B)
fStatus_Sign	Sign status bit (0 if A is greater than B; 1 if A is less than B)

Special case: • if either value is NaN, then the result is NaN

Example:

```
Value1   CON   6       'First value  (uM-FPU register 6)
Value2   CON   7       'Second value (uM-FPU register 7)

  fA = Value1          'compare Value1 and Value2
  fB = Value2
  GOSUB Lcompare

  IF (fStatus_Zero = 1) THEN
    DEBUG "Value1 = Value2"
  ELSEIF (fStatus_Sign = 1) THEN
    DEBUG "Value1 < Value2"
  ELSE
    DEBUG "Value1 > Value2"
  ENDIF
```

Ludivide **A = A / B (unsigned)**

Parameters: fA uM-FPU register number
fB uM-FPU register number

Return: none

Description: The unsigned long integer value in register A is divided by the unsigned long integer value in register B and the result is stored in register A. If the value in register B is zero (divide by zero), register A will be set to the largest unsigned long integer ($FFFFFFFF). The remainder of the division is stored in register 0.

Example:

```
Total  CON   5         'total         (uM-FPU register 5)
Value  CON   6         'current value (uM-FPU register 6)
Rem    CON   7         'remainder     (uM-FPU register 7)

  fA = Total           'Total = Total / Value
  fB = Value
  GOSUB Ldivide

  fA = Rem             'set Rem to the remainder
  fB = 0
  GOSUB Lset
```

Load Routines

Load_E **Load register 0 with floating point value of e (2.7182818)**

Parameters: none
Return: fB set to 0

Description: Loads register 0 with the floating point value of e(2.7182818). The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5         'current value (uM-FPU register 5)

  fA = Value             'Value = Value * e
  GOSUB Load_E
  GOSUB Fmultiply
```

Load_Float **Load register 0 with floating point value**

Parameters: fHigh high 16-bits of floating point number
fLow low 16-bits of floating point number
Return: fB set to 0

Description: Loads the floating point value passed in fHigh and fLow to register 0. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5       'current value (uM-FPU register 5)

  fA = Value           'Value = Value / 10.0
  fHigh = $4120        'high 16 bits of the value 10.0
  fLow = $0000         'low 16 bits of the value 10.0
  GOSUB Load_Float
  GOSUB Fdivide
```

Load_FloatByte **Load register 0 with 8-bit signed integer converted to floating point**

Parameters: flow.LOWBYTE 8-bit signed integer value
Return: fB set to 0

Description: Loads the 8-bit unsigned integer value passed in flow.LOWBYTE into register 0, and converts it to a floating point value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5         'current value (uM-FPU register 5)

  fA = Value             'Value = Value * -10
  flow.LOWBYTE = -10
  GOSUB Load_FloatByte
  GOSUB Fmultiply
```

Load_FloatData Load register 0 with floating point value in EEPROM

Parameters: fAddr EEPROM address of the constant
Return: fB set to 0

Description: Loads a floating point constant from EEPROM to register 0. The EEPROM address is passed in fAddr. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Pi   DATA $40, $49, $0F, $DB  'pi = 3.1415927

  fA = Value            'Value = Value * pi
  fAddr = Pi
  GOSUB Load_FloatData
  GOSUB Fmultiply
```

Load_FloatStr Load register 0 with floating point string in EEPROM

Parameters: fAddr EEPROM address of the constant
Return: fB set to 0

Description: Loads a string from EEPROM at the address specified by fAddr, and sends it to the uM-FPU where it is converted to a floating point value and stored in register 0. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5         'current value (uM-FPU register 5)
PiStr DATA "3.1415927", 0 'zero terminated string

fA = Value                'Value = Value * Pi
fAddr = PiStr
GOSUB Load_FloatStr
GOSUB Fmultiply
```

Load_FloatUByte Load register 0 with 8-bit unsigned integer converted to floating point

Parameters: flow.LOWBYTE 8-bit unsigned integer value
Return: fB set to 0

Description: Loads the 8-bit unsigned integer value passed in flow.LOWBYTE into register 0, and converts it to a floating point value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5         'current value (uM-FPU register 5)

  fA = Value             'Value = Value * 10
  flow.LOWBYTE = 10
  GOSUB Load_FloatUByte
  GOSUB Fmultiply
```

Load_Fraction Load register 0 with the fractional part of A

Parameters: fA uM-FPU register number
Return: none

Description: Loads register 0 with the fractional part the floating point value in register A..

Special cases: • if the value is NaN or infinity, then the result is NaN

Example:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'get the fractional part of Value
  GOSUB Load_Fraction
```

Load_FloatUWord Load register 0 with 16-bit unsigned integer converted to floating point

Parameters: fLow 16-bit unsigned integer value
Return: fB set to 0

Description: Loads the 16-bit signed integer value passed in fLow into register 0, and converts it to a floating point value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)

  fA = Value              'Value = Value * 1000
  fLow = 1000
  GOSUB Load_FloatUWord
  GOSUB Fmultiply
```

Load_FloatWord Load register 0 with 16-bit signed integer converted to floating point

Parameters: fLow 16-bit unsigned integer value
Return: fB set to 0

Description: Loads the 16-bit unsigned integer value passed in fLow into register 0, and converts it to a floating point value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)

  fA = Value              'Value = Value * -1000
  fLow = -1000
  GOSUB Load_FloatWord
  GOSUB Fmultiply
```

Load_Fraction Load register 0 with the fractional part of A

Parameters: fA uM-FPU register number
Return: none

Description: Loads register 0 with the fractional part the floating point value in register A..

Special cases: • if the value is NaN or infinity, then the result is NaN

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'get the fractional part of Value
  GOSUB Load_Fraction
```

Load_Long Load register 0 with long integer value

Parameters: fHigh high 16-bits of long integer
fLow low 16-bits of long integer
Return: fB set to 0

Description: Loads a long integer value to register 0. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5        'current value (uM-FPU register 5)

  fA = Value            'Value = Value / 10
  fHigh = 0             'high 16 bits of the value 10
  fLow = 10             'low 16 bits of the value 10
  GOSUB Load_Long
  GOSUB Ldivide
```

Load_LongByte Load register 0 with 8-bit signed integer converted to long integer

Parameters: fLow 8-bit signed integer value
Return: fB set to 0

Description: Loads the 8-bit integer value passed in the fLow variable into register 0, and converts it to a long integer value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)
intValue VAR   byte       'integer value

  fA = Value              'Value = intValue
  flow.LOWBYTE = intValue
  GOSUB Load_LongByte
  GOSUB Fset

  fA = Value              'Value = Value * -10
  fLow = -10
  GOSUB Load_LongByte
  GOSUB Fmultiply
```

Load_LongData **Load register 0 with long integer value in EEPROM**

Parameters: fAddr EEPROM address of the constant
Return: fB set to 0

Description: Loads a long integer constant from EEPROM to register 0. The EEPROM address is specified by fAddr. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)
L500K    DATA $00, $07, $A1, $20 'constant 500000

  fA = Value               'Value = Value * 500000
  fAddr = L500K
  GOSUB Load_LongData
  GOSUB Lmultiply
```

Load_LongStr **Load register 0 with long integer string in EEPROM**

Parameters: fAddr EEPROM address of the constant
Return: fB set to 0

Description: Loads a string from EEPROM at the address specified by fAddr, and sends it to the uM-FPU where it is converted to a long integer value and stored in register 0. The fB variable is set to zero. Another command that uses the value stored in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)
L500KStr DATA "500000", 0  'zero terminated string

fA = Value                'Value = Value * 500000
fAddr = L500KStr
GOSUB Load_LongStr
GOSUB Lmultiply
```

Load_LongUbyte   Load register 0 with 8-bit unsigned integer converted to long integer.

Parameters: fLow 16-bit signed integer value
Return: fB set to 0

Description: Loads the 16-bit integer value passed in the fLow variable into register 0, and converts it to a long integer value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)
intValue VAR   byte       'integer value

  fA = Value              'Value = intValue
  flow.LOWBYTE = intValue
  GOSUB Load_LongUByte
  GOSUB Fset

  fA = Value              'Value = Value * 1000
  fLow = 1000
  GOSUB Load_LongInt
  GOSUB Fmultiply
```

Load_LongUWord   Load register 0 with 16-bit unsigned integer converted to long integer.

Parameters: fLow 16-bit signed integer value
Return: fB set to 0

Description: Loads the 16-bit integer value passed in the fLow variable into register 0, and converts it to a long integer value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)
intValue VAR   word       'integer value

  fA = Value              'Value = intValue
  fLow = intValue
  GOSUB Load_LongInt
  GOSUB Fset

  fA = Value              'Value = Value * 1000
  fLow = 1000
  GOSUB Load_LongInt
  GOSUB Fmultiply
```

Load_LongWord **Load register 0 with 16-bit signed integer converted to long integer.**

Parameters: fLow 16-bit signed integer value
Return: fB set to 0

Description: Loads the 16-bit integer value passed in the fLow variable into register 0, and converts it to a long integer value. The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)
intValue VAR   word       'integer value

  fA = Value              'Value = intValue
  fLow = intValue
  GOSUB Load_LongInt
  GOSUB Fset

  fA = Value              'Value = Value * -1000
  fLow = -1000
  GOSUB Load_LongInt
  GOSUB Fmultiply
```

Load_LongStr **Load register 0 with long integer string in EEPROM.**

Parameters: fAddr EEPROM address of the constant
Return: fB set to 0

Description: Loads a string from EEPROM at the address specified by fAddr, and sends it to the uM-FPU where it is converted to a long integer value and stored in register 0. The fB variable is set to zero. Another command that uses the value stored in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)
L500KStr DATA "500000", 0  'zero terminated string

fA = Value                'Value = Value * 500000
fAddr = L500KStr
GOSUB Load_LongStr
GOSUB Lmultiply
```

Load_One **Load register 0 with One.**

Parameters: none
Return: fB set to 0

Description: Loads register 0 with a floating point value of 1.0. The fB variable is set to zero. Another command that uses the value stored in register 0 can follow immediately. This routine can be used to load a floating point zero or a long integer zero.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)

  fA = Value              'Value = 1.0
  GOSUB Load_One
  GOSUB Fset
```

Load_Pi **Load register 0 with value of Pi.**

Parameters: none
Return: fB set to 0

Description: Loads register 0 with the floating point value of pi (3.1415927). The fB variable is set to zero. Another command that uses the value in register 0 can follow immediately.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)

  fA = Value              'Value = Value * pi
  GOSUB Load_Pi
  GOSUB Fmultiply
```

Load_Zero **Load register 0 with Zero.**

Parameters: none
Return: fB set to 0

Description: Loads register 0 with a value of zero. The fB variable is set to zero. Another command that uses the value stored in register 0 can follow immediately. This routine can be used to load a floating point zero or a long integer zero.

Example:

```
Value    CON   5          'current value (uM-FPU register 5)

  fA = Value              'Value = 0.0
  GOSUB Load_Zero
  GOSUB Fset
```

Print Routines

Print_Float **Display floating point value on the PC screen.**

Parameters: fA uM-FPU register number
Return: none

Description: The floating point representation of the value in register A is displayed on the PC screen using the DEBUG command. 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:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'print floating point value
  GOSUB Print_Float
```

Print_FloatFormat **Display formatted floating point value on the PC screen.**

Parameters: fA uM-FPU register number
fLow format specification
Return: none

Description: The formatted floating point representation of the value in register A is displayed on the PC screen using the DEBUG command. The format is specified as a decimal value passed in the fLow variable. 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	fLow (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:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'print floating point value
  Fformat = 62          'format 6.2
  GOSUB Print_FloatFormat
```

Print_Hex **Display 32-bit hexadecimal value on the PC screen.**

Parameters: fA uM-FPU register number
Return: none

Description: The hexadecimal representation of the 32-bit value in register A is displayed on the PC screen using the DEBUG command. An example of the display format is as follows:

$4049 0FDB

Example:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'print 32-bit hex value
  GOSUB Print_Hex
```

Print_Long **Display signed long integer value on the PC screen.**

Parameters: fA uM-FPU register number
Return: none

Description: The signed long integer representation of the value in register A is displayed on the PC screen using the DEBUG command. 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:

```
Value CON   5           'current value (uM-FPU register 5)

  fA = Value            'print long integer value
  GOSUB Print_Long
```

Print_LongFormat **Display formatted long integer value on the PC screen.**

Parameters: fA uM-FPU register number
fLow format specification
Return: none

Description: The formatted long integer representation of the value in register A is displayed on the PC screen using the DEBUG command. The format is specified as a decimal value passed in the fLow 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	fLow	(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:

```
Value CON   5          'current value (uM-FPU register 5)

  fA = Value           'print long integer value
  Fformat = 10         'width is 10
  GOSUB Print_Long
```

Appendix B
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 BASIC Stamp data constants are as follows:

```
DATA  $00, $00, $00, $00     '0.0
DATA  $3D, $CC, $CC, $CD     '0.1
DATA  $3F, $00, $00, $00     '0.5
DATA  $3F, $40, $00, $00     '0.75
DATA  $3F, $7F, $F9, $72     '0.9999
DATA  $3F, $80, $00, $00     '1.0
DATA  $40, $00, $00, $00     '2.0
DATA  $40, $2D, $F8, $54     '2.7182818 (e)
DATA  $40, $49, $0F, $DB     '3.1415927 (pi)
DATA  $41, $20, $00, $00     '10.0
DATA  $42, $C8, $00, $00     '100.0
DATA  $44, $7A, $00, $00     '1000.0
DATA  $44, $9A, $52, $2B     '1234.5678
DATA  $49, $74, $24, $00     '1000000.0
DATA  $80, $00, $00, $00     '-0.0
DATA  $BF, $80, $00, $00     '-1.0
DATA  $C1, $20, $00, $00     '-10.0
DATA  $C2, $C8, $00, $00     '-100.0
DATA  $7F, $C0, $00, $00     'NaN (Not-a-Number)
DATA  $7F, $80, $00, $00     '+inf
DATA  $FF, $80, $00, $00     '-inf
```