Using uM-FPU V2
with the BASIC Stamp®

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, BS2pe, or BS2px to provide support for 32-bit IEEE 754 floating point operations and 32-bit long integer operations. The uM-FPU supports both I²C and 2-Wire SPI connections.

uM-FPU V2 Features
- 8-pin integrated circuit.
- I²C compatible interface up to 400 kHz
- SPI compatible interface up to 4 Mhz
- 32 byte instruction buffer
- 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. parentheses)
- 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
  - Read, Compare, Status
- Long Integer Operations
  - Set, Add, Subtract, Multiply, Divide, Unsigned Divide
  - Increment, Decrement, Negate, Abs
  - And, Or, Xor, Not, Shift
  - Read 8-bit, 16-bit, and 32-bit
  - 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 formatted ASCII
  - Convert long integer to formatted ASCII
  - Convert ASCII to floating point
  - Convert ASCII to long integer
- User Defined Functions can be stored in Flash memory
  - Conditional execution
  - Table lookup
  - Nth order polynomials
Connecting uM-FPU V2 to the BASIC Stamp using 2-wire SPI

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:

FpuClk    Pin 15
FpuOut    Pin 14
FpuIn     Pin 14

The settings for these pins can be changed to suit your application. By default, the uM-FPU chip is always selected, so the FpuClk and FpuIn/FpuOut 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.
Connecting uM-FPU V2 to the BASIC Stamp using I²C

The uM-FPU V2 can also be connected using an I²C interface. The default slaveID for the uM-FPU is $C8. The default settings for the I²C pins is:

FpuPin Pin 0 (SDA is Pin 0, SCL is Pin 1)

The settings for these pins can be changed to suit your application.
An Introduction to the uM-FPU

The following section provides an introduction to the uM-FPU using BASIC Stamp commands for all of the examples. For more detailed information about the uM-FPU, please refer to the following documents:

- **uM-FPU V2 Datasheet**: functional description and hardware specifications
- **uM-FPU V2 Instruction Set**: full description of each instruction

### uM-FPU Registers

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 temporary register and is modified by some of the uM-FPU operations. Registers 1 through 15 are available for general use. Arithmetic operations are defined in terms of an A register and a B register. Any of the 16 registers can be selected as the A or B register.

<table>
<thead>
<tr>
<th>A →</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td>32-bit Register</td>
<td></td>
</tr>
<tr>
<td>B →</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

The **FADD** instruction adds two floating point values and is defined as $A = A + B$. To add the value in register 5 to the value in register 2, you would do the following:

- Select register 2 as the A register
- Select register 5 as the B register
- Send the **FADD** instruction ($A = A + B$)

We'll look at how to send these instructions to the uM-FPU in the next section.

Register 0 is a temporary register. If you want to use a value later in your program, store it in one of the registers 1 to 15. Several instructions load register 0 with a temporary value, and then select register 0 as the B register. As you will see shortly, this is very convenient because other instructions can use the value in register 0 immediately.

### Sending Instructions to the uM-FPU

Appendix A contains a table that gives a summary of each uM-FPU instruction, with enough information to follow the examples in this document. For a detailed description of each instruction, refer to the document entitled **uM-FPU Instruction Set**.

To send instructions to the uM-FPU using an SPI interface, the **SHIFTOUT** command is used as follows:

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FADD+5]
```

To send instructions to the uM-FPU using an I²C interface, the **I2COUT** command is used as follows:
I2COUT FpuOut, FpuClk, MSBFIRST, [FADD+5]

The part inside the square brackets specifies the instructions and data to send to the uM-FPU. The part before the square brackets is always the same, and depends on whether you are using an SPI or I2C interface. It tells the BASIC Stamp how to communicate with the uM-FPU. The SHIFTOUT command will be used for the examples in this document, but everything applies equally well to the I2COUT command. Note: There is one difference when sending a Word variable. The SHIFTOUT command can use the [dataWord\16] syntax, whereas I2COUT only sends 8 bit bytes, so the syntax would be [dataWord.HIGHBYTE, dataWord.LOWBYTE].

All instructions start with an opcode that tells the uM-FPU which operation to perform. Some instructions require additional data or arguments, and some instructions return data. The most common instructions (the ones shown in the first half of the table in Appendix A), require a single byte for the opcode. For example:

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SQRT]

The instructions in the last half of the table, are extended opcodes, and require a two byte opcode. The first byte of extended opcodes is always $FE, defined as XOP. To use an extended opcode, you send the XOP byte first, followed by the extended opcode. For example:

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, ATAN]

Some of the most commonly used instructions use the lower 4 bits of the opcode to select a register. This allows them to select a register and perform an operation at the same time. Opcodes that include a register value are defined with the register value equal to 0, so using the opcode by itself selects register 0. The following command selects register 0 as the B register then calculates A = A + B.

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FADD]

To select a different register, you simply add the register value to the opcode. The following command selects register 5 as the B register then calculates A = A + B.

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FADD+5]

Let’s look at a more complete example. Earlier, we described the steps required to add the value in register 5 to the value in register 2. The command to perform that operation is as follows:

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA+2, FADD+5]

Description:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECTA+2</td>
<td>select register 2 as the A register</td>
</tr>
<tr>
<td>FADD+5</td>
<td>select register 5 as the B register</td>
</tr>
<tr>
<td></td>
<td>and calculate A = A + B</td>
</tr>
</tbody>
</table>

It’s a good idea to use constant definitions to provide meaningful names for the registers. This makes your program code easier to read and understand. The same example using constant definitions would be:

Total CON 2 'total amount (uM-FPU register)
Count CON 5 'current count (uM-FPU register)

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA+Total, FADD+Count]

Selecting the A register is such a common occurrence, it was defined as opcode $0x. The definition for SELECTA is $00, so SELECTA+Total is the same as just using Total by itself. Using this shortcut, the same example would now be:

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Total, FADD+Count]
Tutorial Examples

Now that we’ve introduced some of the basic concepts of sending instructions to the uM-FPU, let’s go through a tutorial example to get a better understanding of how it all ties together. This example will take a temperature reading from a DS1620 digital thermometer and convert it to Celsius and Fahrenheit.

Most of the data read from devices connected to the BASIC Stamp will return some type of integer value. In this example, the interface routine for the DS1620 reads a 9-bit value and stores it in a Word variable on the BASIC Stamp called rawTemp. The value returned by the DS1620 is the temperature in units of 1/2 degrees Celsius. We need to load this value to the uM-FPU and convert it to floating point. The following command is used:

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegC, LOADWORD, rawTemp\16, FSET]
```

**Description:**
- `DegC` select DegC as the A register
- `LOADWORD, rawTemp\16` load rawTemp to register 0, convert to floating point, select register 0 as the B register
- `FSET` DegC = register 0 (i.e. the floating point value of rawTemp)

The uM-FPU register DegC now contains the value read from the DS1620 (converted to floating point). Since the DS1620 works in units of 1/2 degree Celsius, DegC will be divided by 2 to get the degrees in Celsius.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [LOADBYTE, 2, FDIV]
```

**Description:**
- `LOADBYTE, 2` load the value 2 to register 0, convert to floating point, select register 0 as the B register
- `FDIV` divide DegC by register 0 (i.e. divide by 2)

To get the degrees in Fahrenheit we will use the formula \( F = C \times 1.8 + 32 \). Since 1.8 and 32 are constant values, they would normally be loaded once in the initialization section of your program and used later in the main program. The value 1.8 is loaded by using the `ATOF` (ASCII to float) instruction as follows:

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [F1_8, ATOF, “1.8”, 0, FSET]
```

**Description:**
- `F1_8` select F1_8 as the A register
- `ATOF, “1.8”, 0` load the string 1.8 (note: the string must be zero terminated) convert the string to floating point, store in register 0, select register 0 as the B register
- `FSET` set F1_8 to the value in register 0 (i.e. 1.8)

The value 32 is loaded using the `LOADBYTE` instruction as follows:

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [F32, LOADBYTE, 32, FSET]
```

**Description:**
- `F32` select F32 as the A register
- `LOADBYTE, 32` load the value 32 to register 0, convert to floating point, select register 0 as the B register
- `FSET` set F32 to the value in register 0 (i.e. 32.0)

Now using these constant values we calculate the degrees in Fahrenheit as follows:

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegF, FSET+DegC, FMUL+F0_9, FADD+F32]
```

**Description:**
- `DegF` select DegF as the A register
- `FSET+DegC` set DegF = DegC
- `FMUL+F1_8` multiply DegF by 1.8
- `FADD+F32` add 32.0 to DegF
Now we print the results. There are support routines provided for printing floating point numbers. Print_Float prints an unformatted floating point value and displays up to eight digits of precision. Print_FloatFormat prints a formatted floating point number. We’ll use Print_FloatFormat to display the results. The format variable is used to select the desired format. The tens digit is the total number of characters to display, and the ones digit is the number of digits after the decimal point. The DS1620 has a maximum temperature of 125° Celsius and one decimal point of precision, so we’ll use a format of 51. Before calling the print routine the uM-FPU register is selected and the format variable is set. The following example prints the temperature in degrees Fahrenheit.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegF]
format = 51
GOSUB Print_FloatFormat
```

Sample code for this tutorial and a wiring diagram for the DS1620 are shown at the end of this document. The file demo1.bs2 is also included with the support software. There is a second file called demo2.bs2 that extends this demo to include minimum and maximum temperature calculations. If you have a DS1620 you can wire up the circuit and try out the demos.

**uM-FPU Support Software for the BASIC Stamp**

Two template files contain all of the definitions and support code required for communicating with the uM-FPU.

- umfpu-spi.bs2 provides support for a 2-wire SPI connection
- umfpu-i2c.bsp provides support for an I2C connection.

These files 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. They contain the following:

- pin definitions for the uM-FPU
- opcode definitions for all uM-FPU instructions
- various definitions for the Word variable used by the support routines
- a sample program with a place to insert your application code
- the support routines described below

**Fpu_Reset**

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 status variable to 1 if successful, or 0 if the reset failed. A sample reset call is included in the umfpu-spi.bs2 and umfpu-i2c.bsp files.

**Fpu_Wait**

The uM-FPU must have completed all calculations and be ready to return the data before sending an instruction that reads data from the uM-FPU. The Fpu_Wait routine checks the status of the uM-FPU and waits until it is ready. The print routines check the ready status, so it isn’t necessary to call Fpu_Wait before calling a print routine. If your program reads directly from the uM-FPU using the SHIFTIN or I2CIN commands, a call to Fpu_Wait must be made prior to sending the read instruction. An example of reading a byte value is as follows:

```
GOSUB Fpu_wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READBYTE]
SHIFTIN FpuOut, FpuClk, MSBPRE, [dataByte]
```

**Description:**

- wait for the uM-FPU to be ready
- send the READBYTE instruction
- read a byte value and store it in the variable dataByte

The uM-FPU V2 has a 32 byte instruction buffer. In most cases, data will be read back before 32 bytes have been sent to the uM-FPU. If a long calculation is done that requires more than 32 bytes to be sent to the uM-FPU, an Fpu_Wait call should be made at least every 32 bytes to ensure that the instruction buffer doesn’t overflow.
**Print_Version**
Prints the uM-FPU version string to the PC screen using the DEBUG command.

**Print_Float**
The value in register A is displayed on the PC screen as a floating point value 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>NaN</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>1.5e20</td>
<td>Infinity</td>
<td>-0.0</td>
</tr>
<tr>
<td>3.1415927</td>
<td>-Infinity</td>
<td>1.0</td>
</tr>
<tr>
<td>-52.333334</td>
<td>-3.5e-5</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Print_FloatFormat**
The value in register A is displayed on the PC screen as a formatted floating point value using the DEBUG command. The format variable is used to specify the desired format. The tens digit specifies the total number of characters to display and the ones digit specifies the number of digits after the decimal point. If the value is too large for the format specified, then asterisks will be displayed. If the number of digits after the decimal points is zero, no decimal point will be displayed. Examples of the display format are as follows:

<table>
<thead>
<tr>
<th>Value in A register</th>
<th>format</th>
<th>Display format</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.567</td>
<td>61 (6.1)</td>
<td>123.6</td>
</tr>
<tr>
<td>123.567</td>
<td>62 (6.2)</td>
<td>123.57</td>
</tr>
<tr>
<td>123.567</td>
<td>42 (4.2)</td>
<td><em>.</em></td>
</tr>
<tr>
<td>0.9999</td>
<td>20 (2.0)</td>
<td>1</td>
</tr>
<tr>
<td>0.9999</td>
<td>31 (3.1)</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Print_Long**
The value in register A is displayed on the PC screen as a signed long integer using the DEBUG command. The displayed value can range from 1 to 11 characters in length. Examples of the display format are as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>500000</td>
<td></td>
</tr>
<tr>
<td>-3598390</td>
<td></td>
</tr>
</tbody>
</table>

**Print_LongFormat**
The value in register A is displayed on the PC screen as a formatted long integer using the DEBUG command. The format variable is used to specify the desired format. 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:

<table>
<thead>
<tr>
<th>Value in register A</th>
<th>format</th>
<th>Display format</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>10 (signed 10)</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>110 (unsigned 10)</td>
<td>4294967295</td>
</tr>
<tr>
<td>-1</td>
<td>4 (signed 4)</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>104 (unsigned 4)</td>
<td>****</td>
</tr>
<tr>
<td>0</td>
<td>4 (signed 4)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 (unformatted)</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>6 (signed 6)</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Loading Data Values to the uM-FPU**
There are several instructions for loading integer values to the uM-FPU. These instructions take an integer value as an argument, stores the value in register 0, converts it to floating point, and selects register 0 as the B register. This allows the loaded value to be used immediately by the next instruction.

- **LOADBYTE**: Load 8-bit signed integer and convert to floating point
- **LOADUBYTE**: Load 8-bit unsigned integer and convert to floating point
LOADWORD
Load 16-bit signed integer and convert to floating point
LOADUWORD
Load 16-bit unsigned integer and convert to floating point

For example, to calculate Result = Result + 20.0

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Result, LOADBYTE, 20, FADD]

Description:
Result
load the value 20 to register 0, convert to floating point, select register 0 as the B register
FADD
add register 0 to Result

The following instructions take integer value as an argument, stores the value in register 0, converts it to a long integer, and selects register 0 as the B register.
LONGBYTE
Load 8-bit signed integer and convert to 32-bit long signed integer
LONGUBYTE
Load 8-bit unsigned integer and convert to 32-bit long unsigned integer
LONGWORD
Load 16-bit signed integer and convert to 32-bit long signed integer
LONGUWORD
Load 16-bit unsigned integer and convert to 32-bit long unsigned integer

For example, to calculate Total = Total / 100

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Total, XOP, LONGBYTE, 100, LDIV]

Description:
Total
load the value 100 to register 0, convert to long integer, select register 0 as the B register
LDIV
divide Total by register 0

There are several instructions for loading commonly used constants. These instructions load the constant value to register 0, and select register 0 as the B register.
LOADZERO
Load the floating point value 0.0 (or long integer 0)
LOADONE
Load the floating point value 1.0
LOADE
Load the floating point value of e (2.7182818)
LOADPI
Load the floating point value of pi (3.1415927)

For example, to set Result = 0.0

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Result, XOP, LOADZERO, FSET]

Description:
Result
load 0.0 the register 0 and selects register 0 as the B register
FSET
set Result to the value in register 0 (Result = 0.0)

There are two instructions for loading 32-bit floating point values to a specified register. This is one of the more efficient ways to load floating point constants, but requires knowledge of the internal representation for floating point numbers (see Appendix B). A handy utility program called uM-FPU Converter is available to convert between floating point strings and 32-bit hexadecimal values.
FWRITEA
Write 32-bit floating point value to specified register
FWRITAB
Write 32-bit floating point value to specified register

For example, to set Angle = 20.0 (the floating point representation for 20.0 is $41A00000)

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FWRITEA+Angle, $41,$A0,$00,$00]

Description:
FWRITEA+Angle
select Angle as the A register and load 32-bit value
$41,$A0,$00,$00
the value $41A00000 is loaded to Angle
There are two instructions for loading 32-bit long integer values to a specified register.

- **LWRITEA**  Write 32-bit long integer value to specified register
- **LWRITAB**  Write 32-bit long integer value to specified register

For example, to set Total = 500000

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, LWRITEA+Total, $00,$07,$A1,$20]
```

**Description:**
- XOP, LWRITEA+Total  select Total as the A register and load 32-bit value
- $00,$07,$A1,$20  the value $0007A120 is loaded to Total

There are two instructions for converting strings to floating point or long integer values.

- **ATOF**  Load ASCII string and convert to floating point
- **ATOL**  Load ASCII string and convert to long integer

For example, to set Angle = 1.5885

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Angle, ATOF, “1.5885”, 0, FSET]
```

**Description:**
- Angle  select Angle as the A register
- ATOF, “1.5885”, 0  load the string 1.5885 to the uM-FPU and convert to floating point (note the string must be zero terminated)
- FSET  set Angle to the value in register 0

For example, to set Total = 500000

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Total, ATOL, “500000”, 0, FSET]
```

**Description:**
- Total  select Total as the A register
- ATOL, “500000”, 0  load the string 500000 to the uM-FPU and convert to floating point (note the string must be zero terminated)
- FSET  set Total to the value in register 0

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 15 registers available for storage on the uM-FPU, it is often possible to preload all of the required constants. In non-critical sections of code, data and constants can be loaded as required.

### Reading Data Values from the uM-FPU

There are two instruction for reading 32-bit floating point values from the uM-FPU.

- **READFLOAT**  Reads a 32-bit floating point value from the A register.
- **FREAD**  Reads a 32-bit floating point value from the specified register.

The following commands read the floating point value from the A register

```
GOSUB Fpu_wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READFLOAT]  
SHIFTIN FpuOut, FpuClk, MSBPRE, [byte0, byte1, byte2, byte3]
```

**Description:**
- wait for the uM-FPU to be ready
- send the READFLOAT instruction
- read the 32-bit value and store it in variables byte0, byte1, byte2, byte3

There are four instruction for reading integer values from the uM-FPU.

- **READBYTE**: Reads the lower 8 bits of the value in the A register.
- **READWORD**: Reads the lower 16 bits of the value in the A register.
- **READLONG**: Reads a 32-bit long integer value from the A register.
- **LREAD**: Reads a 32-bit long integer value from the specified register.

The following commands read the lower 8 bits from the A register

```
GOSUB Fpu_wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READBYTE]
SHIFTIN FpuOut, FpuClk, MSBPRE, [dataByte]
```

*Description:*
- wait for the uM-FPU to be ready
- send the READBYTE instruction
- read a byte value and store it in the variable dataByte

### 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 FSTATUS instruction is sent, and the returned byte is stored in the status variable. A bit definition is provided for each status bit in the status variable. They are as follows:

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

For example:

```
GOSUB Fpu_wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FSTATUS]
SHIFTIN FpuOut, FpuClk, MSBPRE, [status]
IF (status_Sign = 1) THEN DEBUG "Result is negative"
IF (status_Zero = 1) THEN DEBUG "Result is zero"
```

The FCOMPARE instruction 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, the following commands compare the values in registers Value1 and Value2.

```
GOSUB Fpu_wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Value1, SELECTB+Value2, FCOMPARE]
SHIFTIN FpuOut, FpuClk, MSBPRE, [status]
IF (status_Zero = 1) THEN
  DEBUG "Value1 = Value2"
ELSEIF (status_Sign = 1) THEN
  DEBUG "Value1 < Value2"
ELSE
  DEBUG "Value1 > Value2"
ENDIF
```

### 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 LSTATUS instruction is sent, and the returned byte is stored in the status variable. A bit definition is provided for each status bit in the status variable. They are as follows:
**status_Zero**  
Zero status bit (0-not zero, 1-zero)

**status_Sign**  
Sign status bit (0-positive, 1-negative)

For example:

GOSUB Fpu_wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, LSTATUS]
SHIFTIN FpuOut, FpuClk, MSBPRE, [status]
IF (status_Sign = 1) THEN DEBUG "Result is negative"
IF (status_Zero = 1) THEN DEBUG "Result is zero"

The LCOMPARE and LUCOMPARE instructions are used to compare two long integer values. The status bits are set for the results of the operation A – B (The selected A and B registers are not modified). LCOMPARE does a signed compare and LUCOMPARE does an unsigned compare. For example, the following commands compare the values in registers Value1 and Value2.

GOSUB Fpu_wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Value1, SELECTB+Value2, XOP, LCOMPARE]
SHIFTIN FpuOut, FpuClk, MSBPRE, [status]
IF (status_Zero = 1) THEN
  DEBUG "Value1 = Value2"
ELSEIF (status_Sign = 1) THEN
  DEBUG "Value1 < Value2"
ELSE
  DEBUG "Value1 > Value2"

**Left and Right Parenthesis**

Mathematical equations are often expressed with parenthesis to define the order of operations. For example  
\[ Y = \frac{(X-1)}{(X+1)} \]. The LEFT and RIGHT parenthesis instructions provide a convenient means of allocating temporary values and changing the order of operations.

When a LEFT parenthesis instruction is sent, the current selection for the A register is saved and the A register is set to reference a temporary register. Operations can now be performed as normal with the temporary register selected as the A register. When a RIGHT parenthesis instruction is sent, 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 value in register 0 can be used immediately in subsequent operations. Parenthesis can be nested for up to five levels. In most situations, the user’s code does not need to select the A register inside parentheses since it is selected automatically by the LEFT and RIGHT parentheses instructions.

In the following example the equation \[ Z = \sqrt{X^2 + Y^2} \] is calculated. Note that the original values of X and Y are retained.

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)

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Zvalue, FSET+Xvalue, FMUL+Xvalue]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, LEFT, FSET+Yvalue, FMUL+Yvalue]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, RIGHT, FADD, FSQRT]

**Description:**

- **Zvalue**  
  select Zvalue as the A register
- **FSET+Xvalue**  
  Zvalue = Xvalue
- **FMUL+Xvalue**  
  Zvalue = Zvalue * Xvalue (i.e. X**2)
- **XOP, LEFT**  
  save current A register selection, select temporary register as A register (temp)
- **FSET+Yvalue**  
  temp = Yvalue
- **FMUL+Yvalue**  
  temp = temp * Yvalue (i.e. Y**2)
- **XOP, RIGHT**  
  store temp to register 0, select Zvalue as A register (previously saved selection)
FADD         add register 0 to Zvalue (i.e. X**2 + Y**2)
SQRT         take the square root of Zvalue

The following example shows $Y = \frac{10}{X + 1}$:

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [Yvalue, LOADBYTE, 10, FSET]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, LEFT, FSET+Xvalue, XOP, LOADONE, FADD]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, RIGHT, FDIV]
```

**Description:**
- **Yvalue** select Yvalue as the A register
- **LOADBYTE, 10** load the value 10 to register 0, convert to floating point, select register 0 as the B register
- **FSET** $Yvalue = 10.0$
- **XOP, LEFT** save current A register selection, select temporary register as A register (temp)
- **FSET+Xvalue** temp = Xvalue
- **XOP, LOADONE** load 1.0 to register 0 and select register 0 as the B register
- **FADD** temp = temp + 1 (i.e. X+1)
- **XOP, RIGHT** store temp to register 0, select Yvalue as A register (previously saved selection)
- **FDIV** divide Yvalue by the value in register 0

**Further Information**

The following documents are also available:
- **uM-FPU V2 Datasheet** provides hardware details and specifications
- **uM-FPU V2 Instruction Reference** provides detailed descriptions of each instruction
- **uM-FPU Application Notes** various application notes and examples

Check the Micromega website at [www.micromegacorp.com](http://www.micromegacorp.com) for up-to-date information.
DS1620 Connections for Demo 1

Sample Code for Tutorial (Demo1.bs2)

'This program demonstrates how to use the uM-FPU floating point coprocessor connected to the Basic Stamp over an 2-wire SPI interface. It takes temperature readings from a DS1620 digital thermometer, converts them to 'floating point and displays them in degrees Celsius and degrees Fahrenheit.

'==============================================================================
'------- uM-FPU definitions (V2.1) -------------------------------------------
'==============================================================================

FpuClk PIN 15 ' SPI clock (connects to SCLK/SCL)
FpuOut PIN 14 ' SPI data out (connects to SIN/SDA)
FpuIn PIN 14 ' SPI data in  (connects to SOUT)

#IF ($STAMP = BS2SX) OR ($STAMP = BS2P) OR ($STAMP = BS2PX) #THEN
ResetTime CON 625 ' 500 usec reset pulse
#ELSE
ResetTime CON 250 ' 500 usec reset pulse
#ENDIF

'------- uM-FPU opcodes ------------------------------------------------------
SELECTA CON $00 ' select A register
SELECTB CON $10 ' select B register
FSET CON $50 ' A = REG
FADD CON $60 ' A = A + REG (float)
FSUB CON $70 ' A = A - REG (float)
FMUL CON $80 ' A = A * REG (float)
FDIV CON $90 ' A = A / REG (float)
SYNC CON $F0 ' synchronization
LOADBYTE CON $F4 ' write signed byte to register 0, convert to float
LOADWORD CON $F6 ' write signed word to register 0, convert to float
READSTR CON $F8 ' read zero terminated string
ATOF CON $FA ' convert ASCII to float, store in A
FTOA CON $FB ' convert float to ASCII
XOP CON $FE ' extended opcode
VERSION CON $FF ' (XOP) get version string

SyncChar CON $5C ' sync character
'----------------------------- uM-FPU variables -----------------------------------------

dataWord         VAR  Word         ' data word
dataHigh         VAR  dataWord.HIGHBYTE ' high byte of dataWord
dataLow          VAR  dataword.LOWBYTE  ' low byte of dataLow
dataByte         VAR  dataLow       ' (alternate name)
opcode           VAR  dataHigh      ' opcode (same as dataHigh)
reg              VAR  dataHigh      ' register (same as dataHigh)
format           VAR  dataLow       ' format (same as dataLow)
status           VAR  dataLow       ' status (same as dataLow)

'==============================================================================

'----------------------------- DS1620 pin definitions -----------------------------

DS_RST           PIN  12         ' DS1620 reset/enable
DS_CLK           PIN  11         ' DS1620 clock
DS_DATA          PIN  10         ' DS1620 data

'----------------------------- uM-FPU register definitions -----------------------

DegC             CON  1          ' degrees Celsius
DegF             CON  2          ' degrees Fahrenheit
F1_8             CON  3          ' constant 1.8
F32              CON  4          ' constant 32.0

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

rawTemp          VAR  Word        ' raw temperature reading

'==============================================================================

'----------------------------- initialization ---------------------------------------

Reset:
  DEBUG CLS, "Demo 1:"

  GOSUB Fpu_Reset               ' initialize uM-FPU
  IF status <> SyncChar THEN
    DEBUG "uM-FPU not detected."
  ELSE
    GOSUB Print_Version         ' display version string
  ENDIF
  DEBUG CR, "--------------"

  GOSUB Init_DS1620             ' initialize DS1620

  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [F1_8, ATOF, "1.8", 0, FSET]
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [F32, LOADBYTE, 32, FSET]
Main:

GOSUB Read_DS1620          ' get temperature reading from DS1620
DEBUG CRSRX, 0, 3, "Raw Temp: ", IHEX4 rawTemp

    ' send rawTemp to uM-FPU
    ' convert to floating point
    ' store in register
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegC, LOADWORD, rawTemp\16, FSET]

SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegC, LOADBYTE, 2, FDIV]

    ' degF = degC * 1.8 + 32
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegF, FSET+DegC, PMUL+P1_8, FADD+F32]

DEBUG CRSRX, 0, 4, "Degrees C: "    ' display degrees Celsius
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegC]
  format = 51
GOSUB Print_FloatFormat

DEBUG CRSRX, 0, 5, "Degrees F: "    ' display degrees Fahrenheit
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [DegF]
  format = 51
GOSUB Print_FloatFormat

PAUSE 2000                   ' delay, then get the next reading
GOTO Main
END

'==============================================================================
'------------------- Init_DS1620 ---------------------------------------------
'==============================================================================

Init_DS1620:

LOW DS_RST                   ' initialize pin states
HIGH DS_CLK
PAUSE 100

HIGH DS_RST                  ' configure for CPU control
SHIFTOUT DS_DATA, DS_CLK, LSBFIRST, [$0C, $02]
LOW DS_RST
PAUSE 100

HIGH DS_RST                  ' start temperature conversions
SHIFTOUT DS_DATA, DS_CLK, LSBFIRST, [$EE]
LOW DS_RST
PAUSE 1000                   ' wait for first conversion
RETURN

'==============================================================================
'------------------- Read_DS1620 ---------------------------------------------
'==============================================================================

Read_DS1620:

HIGH DS_RST                  ' read temperature value
SHIFTOUT DS_DATA, DS_CLK, LSBFIRST, [$AA]
SHIFTIN DS_DATA, DS_CLK, LSBPRE, [rawTemp\9]
LOW DS_RST
    ' extend the sign bit
IF rawTemp.BIT8 = 1 THEN rawTemp.HIGHBYTE = $FF
RETURN

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

'==============================================================================
'---------------------- uM-FPU SPI support routines (V2.1) ----------------------
'==============================================================================

Fpu_Reset:
    LOW FpuClk                              ' set clock and data lines Low
    LOW FpuOut
    PULSOUT FpuClk, ResetTime              ' send reset pulse to uM-FPU
    PAUSE 8                                ' check for synchronization
    SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SYNC]
    SHIFTIN FpuIn, FpuClk, MSBPRE, [status]
    RETURN

Fpu_Wait:
    INPUT FpuIn                             ' (required for 2-wire interface)
Fpu_Wait2:
    IF (FpuIn = 1) THEN GOTO Fpu_Wait2      ' wait until uM-FPU is ready
    RETURN

Print_Version:
    ' get the uM-FPU version string
    SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, VERSION]
    GOTO Print_String2                     ' print it

Print_Float:
    format = 0                               ' set format to zero (free format)
    ' (fall through to Print_FloatFormat)
Print_FloatFormat:
    opcode = FTOA                            ' convert floating point to formatted ASCII
    ' (fall through to Print_String)

Print_String:
    ' send conversion command
    SHIFTOUT FpuOut, FpuClk, MSBFIRST, [opcode, format]

Print_String2:
    GOSUB Fpu_Wait                           ' wait until uM-FPU is ready
    SHIFTOUT FpuOut, FpuClk, MSBFIRST, [READSTR]
    DO
        ' display zero terminated string
        SHIFTIN FpuIn, FpuClk, MSBPRE, [dataByte]
        IF (dataByte = 0 OR dataByte > 127) THEN EXIT
        DEBUG dataByte
        LOOP
    RETURN

'======================== end of uM-FPU SPI support routines =====================
## Appendix A
uM-FPU V2 Instruction Summary (BASIC Stamp definitions)

<table>
<thead>
<tr>
<th>Opcode Name</th>
<th>Data Type</th>
<th>Opcode</th>
<th>Arguments</th>
<th>Returns</th>
<th>B Reg</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECTA</td>
<td>0x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Select A register</td>
</tr>
<tr>
<td>SELECTB</td>
<td>1x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>Select B register</td>
</tr>
<tr>
<td>FWRITEA</td>
<td>Float</td>
<td>2x</td>
<td>yyyy zzzz</td>
<td></td>
<td></td>
<td>Select A register, Write floating point value to A register</td>
</tr>
<tr>
<td>FWRITEB</td>
<td>Float</td>
<td>3x</td>
<td>yyyy zzzz</td>
<td>x</td>
<td></td>
<td>Select B register, Write floating point value to B register</td>
</tr>
<tr>
<td>FREAD</td>
<td>Float</td>
<td>4x</td>
<td>yyyy zzzz</td>
<td></td>
<td></td>
<td>Read register</td>
</tr>
<tr>
<td>FSET/LSET</td>
<td>Either</td>
<td>5x</td>
<td></td>
<td></td>
<td></td>
<td>Select B register, A = B</td>
</tr>
<tr>
<td>FADD</td>
<td>Float</td>
<td>6x</td>
<td></td>
<td>x</td>
<td></td>
<td>Select B register, A = A + B</td>
</tr>
<tr>
<td>FSUB</td>
<td>Float</td>
<td>7x</td>
<td></td>
<td></td>
<td>x</td>
<td>Select B register, A = A - B</td>
</tr>
<tr>
<td>FMUL</td>
<td>Float</td>
<td>8x</td>
<td></td>
<td>x</td>
<td></td>
<td>Select B register, A = A * B</td>
</tr>
<tr>
<td>FDIV</td>
<td>Float</td>
<td>9x</td>
<td></td>
<td></td>
<td>x</td>
<td>Select B register, A = A / B</td>
</tr>
<tr>
<td>LADD</td>
<td>Long</td>
<td>Ax</td>
<td></td>
<td></td>
<td>x</td>
<td>Select B register, A = A + B</td>
</tr>
<tr>
<td>LSUB</td>
<td>Long</td>
<td>Bx</td>
<td></td>
<td>x</td>
<td></td>
<td>Select B register, A = A - B</td>
</tr>
<tr>
<td>LMUL</td>
<td>Long</td>
<td>Cx</td>
<td></td>
<td></td>
<td>x</td>
<td>Select B register, A = A * B</td>
</tr>
<tr>
<td>LDIV</td>
<td>Long</td>
<td>Dx</td>
<td></td>
<td></td>
<td></td>
<td>Select B register, A = A / B Remainder stored in register 0</td>
</tr>
<tr>
<td>SQRT</td>
<td>Float</td>
<td>E0</td>
<td></td>
<td></td>
<td></td>
<td>A = sqrt(A)</td>
</tr>
<tr>
<td>LOG</td>
<td>Float</td>
<td>E1</td>
<td></td>
<td></td>
<td></td>
<td>A = ln(A)</td>
</tr>
<tr>
<td>LOG10</td>
<td>Float</td>
<td>E2</td>
<td></td>
<td></td>
<td></td>
<td>A = log(A)</td>
</tr>
<tr>
<td>EXP</td>
<td>Float</td>
<td>E3</td>
<td></td>
<td></td>
<td></td>
<td>A = e ** A</td>
</tr>
<tr>
<td>EXP10</td>
<td>Float</td>
<td>E4</td>
<td></td>
<td></td>
<td></td>
<td>A = 10 ** A</td>
</tr>
<tr>
<td>SIN (FSIN)</td>
<td>Float</td>
<td>E5</td>
<td></td>
<td></td>
<td></td>
<td>A = sin(A) radians</td>
</tr>
<tr>
<td>COS (FCOS)</td>
<td>Float</td>
<td>E6</td>
<td></td>
<td></td>
<td></td>
<td>A = cos(A) radians</td>
</tr>
<tr>
<td>TAN (FTAN)</td>
<td>Float</td>
<td>E7</td>
<td></td>
<td></td>
<td></td>
<td>A = tan(A) radians</td>
</tr>
<tr>
<td>FLOOR</td>
<td>Float</td>
<td>E8</td>
<td></td>
<td></td>
<td></td>
<td>A = nearest integer &lt;= A</td>
</tr>
<tr>
<td>CEIL</td>
<td>Float</td>
<td>E9</td>
<td></td>
<td></td>
<td></td>
<td>A = nearest integer &gt;= A</td>
</tr>
<tr>
<td>ROUND</td>
<td>Float</td>
<td>EA</td>
<td></td>
<td></td>
<td></td>
<td>A = nearest integer to A</td>
</tr>
<tr>
<td>NEGATE</td>
<td>Float</td>
<td>EB</td>
<td></td>
<td></td>
<td></td>
<td>A = -A</td>
</tr>
<tr>
<td>ABS (FABS)</td>
<td>Float</td>
<td>EC</td>
<td></td>
<td></td>
<td></td>
<td>A = lA</td>
</tr>
<tr>
<td>INVERSE</td>
<td>Float</td>
<td>ED</td>
<td></td>
<td></td>
<td></td>
<td>A = 1 / A</td>
</tr>
<tr>
<td>DEGREES</td>
<td>Float</td>
<td>EE</td>
<td></td>
<td></td>
<td></td>
<td>Convert radians to degrees A = A / (PI / 180)</td>
</tr>
<tr>
<td>RADIANS</td>
<td>Float</td>
<td>EF</td>
<td></td>
<td></td>
<td></td>
<td>Convert degrees to radians A = A * (PI / 180)</td>
</tr>
<tr>
<td>SYNC</td>
<td>Float</td>
<td>F0</td>
<td>5C</td>
<td></td>
<td></td>
<td>Synchronization</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Long</td>
<td>F1</td>
<td>0</td>
<td></td>
<td></td>
<td>Copy A to register 0</td>
</tr>
<tr>
<td>FIX</td>
<td>Float</td>
<td>F2</td>
<td>0</td>
<td></td>
<td></td>
<td>Copy A to register 0</td>
</tr>
<tr>
<td>FCOMPARE</td>
<td>Float</td>
<td>F3</td>
<td>ss</td>
<td></td>
<td></td>
<td>Compare A and B (floating point)</td>
</tr>
<tr>
<td>LOADBYTE</td>
<td>Float</td>
<td>F4</td>
<td>bb</td>
<td>0</td>
<td></td>
<td>Write signed byte to register 0</td>
</tr>
<tr>
<td>LOADUBYTE</td>
<td>Float</td>
<td>F5</td>
<td>bb</td>
<td>0</td>
<td></td>
<td>Write unsigned byte to register 0</td>
</tr>
<tr>
<td>LOADWORD</td>
<td>Float</td>
<td>F6</td>
<td>www</td>
<td>0</td>
<td></td>
<td>Write signed word to register 0</td>
</tr>
<tr>
<td>LOADUWORD</td>
<td>Float</td>
<td>F7</td>
<td>www</td>
<td>0</td>
<td></td>
<td>Write unsigned word to register 0</td>
</tr>
</tbody>
</table>
### Appendix A - Instruction Summary

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Type</th>
<th>Opcodes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READSTR</td>
<td>Float</td>
<td>F8</td>
<td>Read zero terminated string from string buffer</td>
</tr>
<tr>
<td>ATOF</td>
<td>Float</td>
<td>F9</td>
<td>Convert ASCII to float Store in register 0</td>
</tr>
<tr>
<td>FTOA</td>
<td>Float</td>
<td>FA</td>
<td>Convert float to ASCII Store in string buffer</td>
</tr>
<tr>
<td>ATOL</td>
<td>Long</td>
<td>FB</td>
<td>Convert ASCII to long Store in register 0</td>
</tr>
<tr>
<td>LTOA</td>
<td>Long</td>
<td>FC</td>
<td>Convert long to ASCII Store in string buffer</td>
</tr>
<tr>
<td>FSTATUS</td>
<td>Float</td>
<td>FD</td>
<td>Get floating point status of A</td>
</tr>
<tr>
<td>XOP</td>
<td>Float</td>
<td>FE</td>
<td>Extended opcode prefix (extended opcodes are listed below)</td>
</tr>
<tr>
<td>NOP</td>
<td>Float</td>
<td>FF</td>
<td>No Operation</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>Float</td>
<td>FE0n, FE1n, FE2n, FE3n</td>
<td>User defined functions 0-15, User defined functions 16-31, User defined functions 32-47, User defined functions 48-63</td>
</tr>
<tr>
<td>IF_FSTATUSA</td>
<td>Float</td>
<td>FE80</td>
<td>Execute user function code if FSTATUSA conditions match</td>
</tr>
<tr>
<td>IF_FSTATUSB</td>
<td>Float</td>
<td>FE81</td>
<td>Execute user function code if FSTATUSB conditions match</td>
</tr>
<tr>
<td>IF_FCOMPARE</td>
<td>Float</td>
<td>FE82</td>
<td>Execute user function code if FCOMPARE conditions match</td>
</tr>
<tr>
<td>IF_LSTATUSA</td>
<td>Long</td>
<td>FE83</td>
<td>Execute user function code if LSTATUSA conditions match</td>
</tr>
<tr>
<td>IF_LSTATUSB</td>
<td>Long</td>
<td>FE84</td>
<td>Execute user function code if LSTATUSB conditions match</td>
</tr>
<tr>
<td>IF_LCOMPARE</td>
<td>Long</td>
<td>FE85</td>
<td>Execute user function code if LCOMPARE conditions match</td>
</tr>
<tr>
<td>IF_LTST</td>
<td>Long</td>
<td>FE87</td>
<td>Execute user function code if LTST conditions match</td>
</tr>
<tr>
<td>TABLE</td>
<td>Either</td>
<td>FE88</td>
<td>Table Lookup (user function)</td>
</tr>
<tr>
<td>POLY</td>
<td>Float</td>
<td>FE89</td>
<td>Calculate n° degree polynomial (user function)</td>
</tr>
<tr>
<td>READBYTE</td>
<td>Long</td>
<td>FE90</td>
<td>Get lower 8 bits of register A</td>
</tr>
<tr>
<td>READWORD</td>
<td>Long</td>
<td>FE91</td>
<td>Get lower 16 bits of register A</td>
</tr>
<tr>
<td>READLONG</td>
<td>Long</td>
<td>FE92</td>
<td>Get long integer value of register A</td>
</tr>
<tr>
<td>READFLOAT</td>
<td>Float</td>
<td>FE93</td>
<td>Get floating point value of register A</td>
</tr>
<tr>
<td>LINCA</td>
<td>Long</td>
<td>FE94</td>
<td>A = A + 1</td>
</tr>
<tr>
<td>LINCBE</td>
<td>Long</td>
<td>FE95</td>
<td>B = B + 1</td>
</tr>
<tr>
<td>LDECA</td>
<td>Long</td>
<td>FE96</td>
<td>A = A - 1</td>
</tr>
<tr>
<td>LDECB</td>
<td>Long</td>
<td>FE97</td>
<td>B = B - 1</td>
</tr>
<tr>
<td>LAND</td>
<td>Long</td>
<td>FE98</td>
<td>A = A AND B</td>
</tr>
<tr>
<td>LOR</td>
<td>Long</td>
<td>FE99</td>
<td>A = A OR B</td>
</tr>
<tr>
<td>LXOR</td>
<td>Long</td>
<td>FE9A</td>
<td>A = A XOR B</td>
</tr>
<tr>
<td>LNOT</td>
<td>Long</td>
<td>FE9B</td>
<td>A = NOT A</td>
</tr>
<tr>
<td>LTST</td>
<td>Long</td>
<td>FE9C</td>
<td>Get the status of A AND B</td>
</tr>
<tr>
<td>LSHIFT</td>
<td>Long</td>
<td>FE9D</td>
<td>A = A shifted by B bit positions</td>
</tr>
<tr>
<td>LWRITEA</td>
<td>Long</td>
<td>FEAx</td>
<td>Write register and select A</td>
</tr>
<tr>
<td>LWRITEB</td>
<td>Long</td>
<td>FEBx</td>
<td>Write register and select B</td>
</tr>
<tr>
<td>LREAD</td>
<td>Long</td>
<td>FECx</td>
<td>Read register</td>
</tr>
<tr>
<td>LUDIV</td>
<td>Long</td>
<td>FEDx</td>
<td>Select B register, A = A / B (unsigned) Remainder stored in register 0</td>
</tr>
<tr>
<td>POWER</td>
<td>Float</td>
<td>FEE0</td>
<td>A = A raised to the power of B</td>
</tr>
<tr>
<td>ROOT</td>
<td>Float</td>
<td>FEE1</td>
<td>A = the Bth root of A</td>
</tr>
</tbody>
</table>
## Appendix A - Instruction Summary

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Data Type</th>
<th>Opcode</th>
<th>Arguments</th>
<th>Returns</th>
<th>B Reg</th>
<th>x</th>
<th>n</th>
<th>yyyy</th>
<th>zzzz</th>
<th>ss</th>
<th>bb</th>
<th>www</th>
<th>aa … 00</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN (FMIN)</td>
<td>Float</td>
<td>FEE2</td>
<td></td>
<td>A = minimum of A and B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX (FMAX)</td>
<td>Float</td>
<td>FEE3</td>
<td></td>
<td>A = maximum of A and B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRACTION</td>
<td>Float</td>
<td>FEE4</td>
<td>0</td>
<td>Load Register 0 with the fractional part of A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASIN</td>
<td>Float</td>
<td>FEE5</td>
<td></td>
<td>A = asin(A) radians</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACOS</td>
<td>Float</td>
<td>FEE6</td>
<td></td>
<td>A = acos(A) radians</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATAN</td>
<td>Float</td>
<td>FEE7</td>
<td></td>
<td>A = atan(A) radians</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATAN2</td>
<td>Float</td>
<td>FEE8</td>
<td></td>
<td>A = atan(A/B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOMPARE</td>
<td>Long</td>
<td>FEE9</td>
<td>ss</td>
<td>Compare A and B (signed long integer)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUCOMpare</td>
<td>Long</td>
<td>FEEA</td>
<td>ss</td>
<td>Compare A and B (unsigned long integer)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTATUS</td>
<td>Long</td>
<td>FEEB</td>
<td>ss</td>
<td>Get long status of A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LNEGATE</td>
<td>Long</td>
<td>FEEC</td>
<td></td>
<td>A = -A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LABS</td>
<td>Long</td>
<td>FEED</td>
<td></td>
<td>A =</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEFT</td>
<td></td>
<td>FEEE</td>
<td></td>
<td>Left parenthesis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RIGHT</td>
<td></td>
<td>FEFF</td>
<td>0</td>
<td>Right parenthesis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOADZERO</td>
<td>Float</td>
<td>FEF0</td>
<td>0</td>
<td>Load Register 0 with Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOADONE</td>
<td>Float</td>
<td>FEF1</td>
<td>0</td>
<td>Load Register 0 with 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOADE</td>
<td>Float</td>
<td>FEF2</td>
<td>0</td>
<td>Load Register 0 with e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOADPI</td>
<td>Float</td>
<td>FEF3</td>
<td>0</td>
<td>Load Register 0 with pi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONGBYTE</td>
<td>Long</td>
<td>FEF4</td>
<td>bb</td>
<td>Write signed byte to register 0 Convert to long</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONGUBYTE</td>
<td>Long</td>
<td>FEF5</td>
<td>bb</td>
<td>Write unsigned byte to register 0 Convert to long</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONGWORD</td>
<td>Long</td>
<td>FEF6</td>
<td>www</td>
<td>Write signed word to register 0 Convert to long</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONGUWORD</td>
<td>Long</td>
<td>FEF7</td>
<td>www</td>
<td>Write unsigned word to register 0 Convert to long</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEEEMODE</td>
<td>Long</td>
<td>FEF8</td>
<td></td>
<td>Set IEEE mode (default)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICMODE</td>
<td>Long</td>
<td>FEF9</td>
<td></td>
<td>Set PIC mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHECKSUM</td>
<td>Long</td>
<td>FEF0</td>
<td>0</td>
<td>Calculate checksum for uM-FPU code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BREAK</td>
<td></td>
<td>FEF8</td>
<td>0</td>
<td>Debug breakpoint</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRACEOFF</td>
<td></td>
<td>FEF9</td>
<td></td>
<td>Turn debug trace off</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TraceOn</td>
<td></td>
<td>FED0</td>
<td></td>
<td>Turn debug trace on</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRACESTR</td>
<td></td>
<td>FEEF</td>
<td>aa … 00</td>
<td>Send debug string to trace buffer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERSION</td>
<td></td>
<td>FEEF</td>
<td></td>
<td>Copy version string to string buffer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **Data Type**: data type required by opcode
- **Opcode**: hexadecimal opcode value
- **Arguments**: 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
- **www**: 16-bit value
- **aa … 00**: zero terminated ASCII string
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:

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**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 $+\infty$, $-\infty$ 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.
Appendix B – Floating Point Numbers

**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:

```plaintext
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     '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     '-1.0
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
```