Using the uM-FPU with the PICAXE

Micromega Corporation

Introduction

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

uM-FPU Features

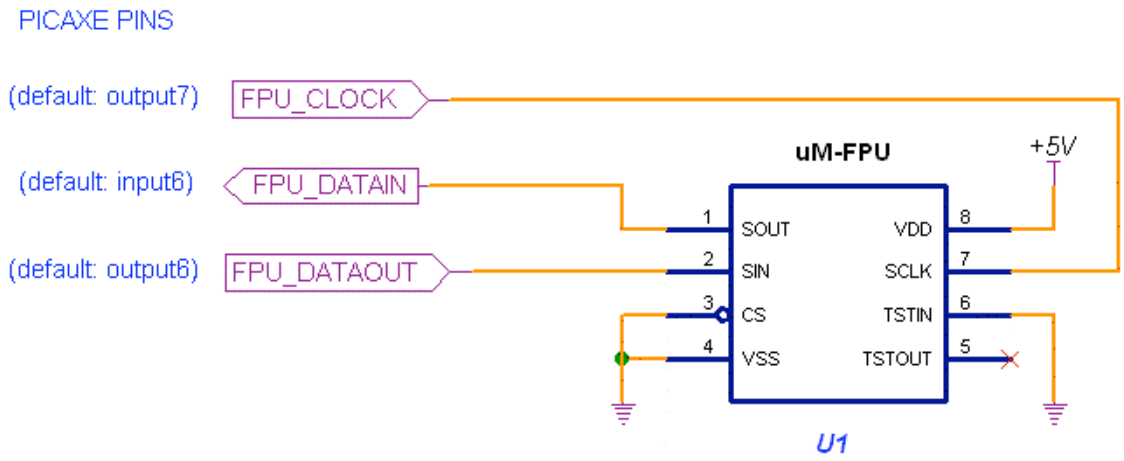
- 8-pin integrated circuit.
- No additional external components
- SPI compatible interface
- Sixteen 32-bit general purpose registers for storing floating point or long integer values
- Five 32-bit temporary registers with support for nested calculations (i.e. parenthesis)
- Floating Point Operations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

Connecting the uM-FPU to the PICAXE

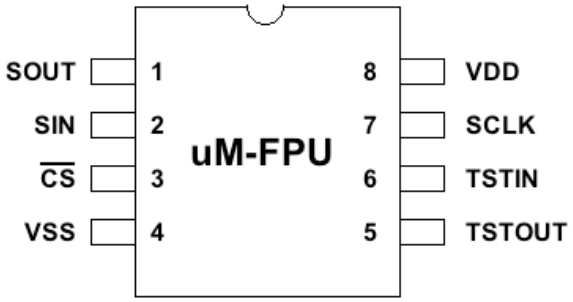
The uM-FPU requires two output pins and one input pin for interfacing to the PICAXE. The communication is implemented using a SPI interface. The default setting for these pins are:

```
FPU_CLOCK     = output7
FPU_DATAOUT   = output6
FPU_DATAIN    = input6
```

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 FPU_CLOCK, FPU_DATAOUT and FPU_DATAIN pins should not be used for other connections as this will likely result in loss of synchronization between the PICAXE and the uM-FPU coprocessor.



uM-FPU Pin Assignment



PIN DESCRIPTION

SOUT	SPI Output
SIN	SPI Input
$\overline{CS}$	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 PICAXE and the uM-FPU. The template file uM-FPU.BAS 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 PICAXE and the uM-FPU coprocessor are synchronized, reset code must be executed at the start of every program. The required reset code is included in the uM-FPU.BAS file. The code is as follows:

```
low FPU_CLOCK                   'reset the uM-FPU coprocessor
low FPU_DATAOUT
pulsout FPU_CLOCK, 30           'send 300 usec reset pulse
pause 2                         'wait for reset
```

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

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

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

Commands are sent to the uM-FPU by setting the `opcode` variable to the value of the command opcode and calling the `fpu_command` sub procedure. (See Appendix B for a summary of opcodes. Symbols for all opcodes have been defined in the uM-FPU.bas file) For example:

```
opcode = SQRT               'A = SQRT(A)
gosub fpu_command
```

Any of the sixteen registers can be selected as the A or B registers. The A register is set with the SELECTA command and the B register is set with the SELECTB command. The `reg` variable is used to specify the register for the command. For example:

```
reg = 1                     'Select Register 1 as A register
opcode = SELECTA
gosub fpu_command
```

The B register is automatically selected by many of the uM-FPU commands, and the `fpu_command` sub procedure using the `reg` variable to select the B register. A separate SELECTB command is not often required.

For example, the following code adds register 2 to register 1.

```
reg = 1
opcode = SELECTA            'Select Register 1 as A register
gosub fpu_command

reg = 2                     'reg specifies the B register
opcode = FADD               'A = A + B
gosub fpu_command           '(Register 1 = Register 1 + Register 2)
```

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

```
symbol Total      = 1   'total amount  (uM-FPU register 1)
symbol Value      = 2   'current value (uM-FPU register 2)

reg = Total             'Select Total as A register
opcode = SELECTA
gosub fpu_command

reg = Value             'Total = Total + Value
opcode = FADD
gosub fpu_command
```

The following floating point commands are provided:

SET	A = B
FADD	A = A + B
FSUB	A = A – B
FMUL	A = A * B
FDIV	A = A / B
ABS	A = |A|
ACOS	A = acos (A)
ASIN	A = asin(A)
ATAN	A = atan(A)
ATAN2	A = atan2(A)
CEIL	A = ceil(A)
COS	A = cos(A)
EXP	A = exp(A)
EXP10	A = exp10(A)
FCOMPARE	Compare A and B
FIX	A = fix(B)
FLOOR	A = floor(A)
FSTATUS	Get the floating point status of A
GET	Get the value of A
INV	A = 1 / A
LOG	A = log(A)
LOG10	A = log10(A)
MAX	A = maximum of A and B
MIN	A = minimum of A and B
NEGATE	A = -A
POWER	A = A to the power of B
ROOT	A = the Bth root of A
ROUND	A = round(A)
SIN	A = sin(A)
SQRT	A = sqrt(A)
TAN	A = tan(A)
DEGREES	Convert radians to degrees
RADIANS	Convert degrees to radians

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

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


reg = Xvalue             'X = X ** 2
opcode = SELECTA
gosub fpu_command

opcode = FMUL
gosub fpu_command

reg = Yvalue             'Y = Y ** 2
opcode = SELECTA
gosub fpu_command

opcode = FMUL
gosub fpu_command

reg = Zvalue             'Z = X + Y
opcode = SELECTA
gosub fpu_command

reg = Xvalue
opcode = SET
gosub fpu_command

reg = Yvalue
opcode = FADD
gosub fpu_command

opcode = SQRT            'Z = sqrt(Z)
gosub fpu_command
```

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

```
reg = Result             'Result = sqrt(Value1 + Value2 + Value3)
opcode = SELECTA
gosub fpu_command
reg = Value1
opcode = SET
gosub fpu_command
reg = value2
opcode = FADD
gosub fpu_command
reg = value3
opcode = FADD
gosub fpu_command
opcode = SQRT
gosub fpu_command
```

Note: The `opcode` value must be set for each command because the value of `opcode` will be changed by the `fpu_command` sub procedure. After each command, the value of `reg` is set to the value of the B register. As a result, in many cases the next command can be executed without having to change the value of `reg`.

Loading Floating Point Values

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

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
LOADZERO	Load the floating point value 0.0
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)

Load a signed byte value:

```
opcode = LOADBYTE        'load byte value and convert to float
gosub fpu_command
dataByte = n             '(where n is a byte variable)
gosub fpu_sendByte       '8-bit byte is sent using fpu_sendByte
```

Load an unsigned word value:

```
opcode = LOADUWORD       'load byte value and convert to float
gosub fpu_command
dataWord = 50000
gosub fpu_sendWord       '16-bit word is sent using fpu_sendWord
```

Load Zero:

```
opcode = LOADZERO        'load register 0 with 0.0
gosub fpu_command
```

Load Pi:

```
opcode = LOADPI          'load register 0 with 3.1415927
gosub fpu_command
```

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

Load a floating point value directly in code:

```
reg = Angle              'write 32-bit value to register
opcode = WRITEB          ' and select register as B register
gosub fpu_command
dataWord = $41A0        '(floating point value 20.0)
gosub fpu_sendWord
dataword = $0000
gosub fpu_sendWord
```

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

```
reg = Angle              'Angle = Angle / pi
opcode = SELECTA
gosub fpu_command
```

```
opcode = LOADPI
gosub fpu_command
opcode = FDIV
gosub fpu_command

reg = Value              'Value = Value + 2
opcode = SELECTA
gosub fpu_command
opcode = LOADBYTE
gosub fpu_command
dataByte = 2
gosub fpu_sendByte
opcode = FADD
gosub fpu_command
```

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

Comparing and Testing Floating Point Values

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

IS_ZERO	Plus zero
IS_NZERO	Minus zero
IS_NEGATIVE	Negative
IS_NAN	Not-a-Number
IS_PINF	Plus infinity
IS_NINF	Minus infinity

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

```
opcode = FSTATUS
gosub fpu_command
gosub fpu_getByte
if status = IS_ZERO or status = IS_NZERO then zeroValue
if status = IS_NEGATIVE then negativeValue

  sertxd("value is positive")
…
negativeValue:
  sertxd("value is negative")
…
zeroValue:
  sertxd("value is zero")
```

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

```
opcode = FCOMPARE
gosub fpu_command
gosub fpu_getByte
if status = IS_ZERO then sameAs
if status = IS_NEGATIVE then lessThan
  sertxd("A > B")
  …
lessThan:
  sertxd("A < B")
  …
sameAs:
  sertxd("A = B")
  …
```

Using the uM-FPU Long Integer Routines

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

```
symbol Total = 1          'total amount (uM-FPU register 1)
symbol Value = 2          'current count (uM-FPU register 2)

reg = Total               'Total = Total + Value
opcode = SELECTA
gosub fpu_command
reg = Value
opcode = LADD
gosub fpu_command
```

The following long integer routines are provided:

SET	A = B
LADD	A = A + B
LSUB	A = A – B
LMUL	A = A * B
LDIV	A = A / B
LUDIV	A = A / B (unsigned)
LABS	A = |A|
LCOMPARE	Compare A and B
LFLOAT	A = float(A)
LSTATUS	Get the long integer status of A
LNEGATE	A = -A
LUCOMPARE	Compare A and B (unsigned)

Loading Long Integer Values

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

LONGBYTE	Load 8-bit signed integer and convert to long integer
LONGUBYTE	Load 8-bit unsigned integer and convert to long integer
LONGWORD	Load 16-bit signed integer and convert to long integer
LONGUWORD	Load 16-bit unsigned integer and convert to long integer
LOADZERO	Load the long integer value 0

Load a byte value:

```
opcode = LONGBYTE         'load byte value and convert to long
gosub fpu_command
dataByte = n              '(where n is a byte variable)
gosub fpu_sendByte        '8-bit byte is sent using fpu_sendByte
```

Load Zero:

```
opcode = LOADZERO         'load register 0 with 0
gosub fpu_command
```

Load a long value directly in code:

```
reg = Value               'write 32-bit value to register
opcode = WRITEB           ' and select register as B register
gosub fpu_command
dataWord = $0007         'high 16-bits of 500,000
```

```
gosub fpu_sendWord
dataword = $A120         'low 16-bits of 500,000
gosub fpu_sendWord
```

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 LONGBYTE routine transfers an additional 8-bit value, the LONGWORD routine transfers two 8-bit values, and the WRITEA and WRITEB routines transfer four 8-bit values. Minimizing the amount of data transfer will maximize the execution speed of your program.

Comparing and Testing Long Integer Values

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

IS_ZERO	Plus zero
IS_NEGATIVE	Negative

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

```
opcode = LSTATUS
gosub fpu_command
gosub fpu_getByte
if status = IS_ZERO then zeroValue
if status = IS_NEGATIVE then negativeValue

  sertxd("value is positive")
…
negativeValue:
  sertxd("value is negative")
…
zeroValue:
  sertxd("value is zero")
```

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

```
opcode = LCOMPARE
gosub fpu_command
gosub fpu_getByte
if status = IS_ZERO then sameAs
if status = IS_NEGATIVE then lessThan
  sertxd("A > B")
  …
lessThan:
  sertxd("A < B")
  …
sameAs:
  sertxd("A = B")
  …
```

Left and Right Parenthesis

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

When a left parenthesis is issued, the current A register selection is saved and a new value is assigned that references a temporary register. Operations can now be performed as normal with the temporary register selected as the A register. When a right parenthesis is issued, the current value of the A register is copied to register 0, register 0 is selected as the B register, and the previous A register selection is restored. The register 0 value can be used immediately in subsequent operations. Up to five levels of parentheses can be used. The SELECTA command should not generally be used inside parentheses since the A register is selected 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 parentheses, it's easy to implement the equation while retaining the original values of X and Y. For example:

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

'Z = sqrt(X**2 + Y**2)
'----------------------
reg = Zvalue              'Zvalue = Xvalue ** 2
opcode = SELECTA
gosub fpu_command
reg = Xvalue
opcode = SET
gosub fpu_command
opcode = FMUL
gosub fpu_command

opcode = LEFT             'temp1 = Yvalue ** 2
gosub fpu_command
reg = Yvalue
opcode = SET
gosub fpu_command
opcode = FMUL
gosub fpu_command

opcode = RIGHT            'Zvalue = Zvalue + temp1
gosub fpu_command
opcode = FADD
gosub fpu_command

opcode = SQRT             'Zvalue = sqrt(Zvalue)
gosub fpu_command
```

Another example:

```
'Y = 10 / (X + 1)
'----------------
reg = Yvalue             'Yvalue = 10
opcode = SELECTA
gosub fpu_command
opcode = LOADBYTE
gosub fpu_command
dataByte = 10
gosub fpu_sendByte
opcode = SET
gosub fpu_command

opcode = LEFT            'temp1 = Xvalue + 1
gosub fpu_command
reg = Xvalue
opcode = SET
gosub fpu_command
opcode = LOADONE
gosub fpu_command
opcode = FADD
gosub fpu_command

opcode = RIGHT           'Yvalue = Yvalue / temp1
gosub fpu_command
opcode = FDIV
gosub fpu_command
```

Print routines

There are several print routines provided to display register values on the PC screen using the `sertxd` PICAXE command.

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

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

```
reg = Angle              'select Angle as A register
opcode = SELECTA
gosub fpu_command

gosub print_Float        'displays Angle in default float format
```
Value displayed: 3.1415927

```
format = 62              'display Angle in 6.2 float format
gosub print_FloatFormat
```
Value displayed: 3.1416

```
reg = Total              'select Total as A register
opcode = SELECTA
gosub fpu_command

gosub Print_Long         'displays Total in default long format
```
Value displayed: -2000

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

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

Sample Code

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

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

symbol   Diameter       = 4   'diameter         (uM-FPU register 4)
symbol   Circumference  = 5   'circumference    (uM-FPU register 5)
symbol   Area           = 6   'area             (uM-FPU register 6)

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

symbol   inputValue     = W0  'diameter in centimeters

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

main:
      low FPU_CLOCK            'reset the uM-FPU coprocessor
      low FPU_DATAOUT
      pulsout FPU_CLOCK, 30    'send 300 usec reset pulse
      pause 2                  'wait for reset

      'get input value
      '---------------
      inputValue = 250
      sertxd(13, 10, "Diameter (mm): ", #inputValue)

      'Diameter = inputValue / 10 (convert to centimeters)
      '--------------------------------------------------
      reg = Diameter           'select Diameter as A register
      opcode = SELECTA
      gosub fpu_command

      opcode = LOADWORD        'load a word value into Reg0 and convert
      gosub fpu_command        ' to floating point.
      dataByte = inputValue    '(selects Reg0 as B register, reg = 0)
      gosub fpu_sendWord

      opcode = SET             'A = B
      gosub fpu_command        '(Diameter = Reg0)

      opcode = LOADBYTE        'load 10 into Reg0 and convert to
      gosub fpu_command        ' floating point (10.0)
      dataByte = 10            '(selects Reg0 as B register, reg = 0)
      gosub fpu_sendByte

      opcode = FDIV            'A = A / B
      gosub fpu_command        '(Diameter = Diameter / Reg0)

      sertxd(13, 10, "Diameter (cm): ")
      format = 92              'print as 9.2 floating point format
      gosub print_floatFormat
```

```
'Circumference = Diameter * pi
'----------------------------
reg = Circumference     'select Circumference as A register
opcode = SELECTA
gosub fpu_command

reg = Diameter          'select Diameter as B register
opcode = SET            'A = B
gosub fpu_command       '(Circumference = Diameter)

opcode = LOADPI         'load the value of pi into Reg0
gosub fpu_command       '(selects Reg0 as B register, reg = 0)

opcode = FMUL           'A = A * B
gosub fpu_command       '(Circumference = Circumference * Reg0)

sertxd(13, 10, "Circumference (cm): ")
format = 92             print as 9.2 floating point format
gosub print_floatFormat

'Area = (Diameter / 2)^2 * pi
'---------------------------
reg = Area              'select Area as register A
opcode = SELECTA
gosub fpu_command

reg = Diameter          'select Diameter as B register
opcode = SET            'A = B
gosub fpu_command       '(Area = Diameter)

opcode = LOADBYTE       'load 2 into Reg0 and convert to
gosub fpu_command       ' floating point (2.0)
dataByte = 2            '(selects Reg0 as B register, reg = 0)
gosub fpu_sendByte

opcode = FDIV           'A = A / B
gosub fpu_command       '(Area = Area / Reg0)

reg = Area              'select Area as B register
opcode = FMUL           'A = A * B
gosub fpu_command       '(Area = Area * Area)

opcode = LOADPI         'load the value of pi into Reg0
gosub fpu_command       '(selects Reg0 as B register, reg = 0)

opcode = FMUL           'A = A * B
gosub fpu_command       '(Area = Area * Reg0)

sertxd(13, 10, "Area (sq.cm.): ")
format = 92             'print as 9.2 floating point format
gosub print_floatFormat

sertxd(13, 10, "Done.")  'end of program
end
```

Appendix A
Reference for uM-FPU PICAXE routines

The uM_FPU PICAXE interface is implemented using sub procedures. Since this is a limited resource on the PICAXE it is recommended that the PICAXE 18-X, PICAXE 28-X or PICAXE-40X be used with the option for 256 gosubs enabled. The uM-FPU interface can be run with the PICAXE clock frequency at either 4Mhz or 8MHz

Initialization Routine

reset	Reset the uM-FPU

Data Transfer Routines

fpu_command	Send command to the uM-FPU
fpu_getByte	Get byte from the uM-FPU
fpu_getWord	Get word from the uM-FPU
fpu_getStr	Get string from the uM-FPU
fpu_sendByte	Send byte to the uM-FPU
fpu_sendWord	Send word to the uM-FPU

Print Routines

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

Variables used as parameters

opcode	Word	Used to select the opcode for uM-FPU command
dataWord	Word	Used for data input/output
highByte	Byte	High byte of dataWord
dataByte	Byte	Low byte of dataWord
format	Byte	Used to specify format for printing (same byte as dataByte)
reg	Byte	Used to select register for uM-FPU command
bitcnt	Byte	Used as counter by input/output routines

Status Bits

IS_ZERO	Plus zero
IS_NZERO	Minus zero
IS_NEGATIVE	Negative
IS_NAN	Not-a-Number
IS_PINF	Plus infinity
IS_NINF	Minus infinity

Initialization Routine

reset **Reset the uM-FPU**

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 PICAXE and the uM-FPU is established. All uM-FPU registers are initialized to NaN (Not a Number) at reset, therefore any operation that uses a register before a value has been stored in the register will produce a result of NaN.

Example:

```
low FPU_CLOCK                   'reset the uM-FPU coprocessor
low FPU_DATAOUT
pulsout FPU_CLOCK, 30           'send 300 usec reset pulse
pause 2                         'wait for reset
```

Data Transfer Routines

fpu_command Send command to the uM-FPU

Parameters: `opcode` command opcode value
`reg` register value

Return: none

Description: This sub procedure sends a command to the uM-FPU. Before sending the command it checks the `opcode` to see if a register value is required. If required, the register value specified by the `reg` variable is added to `opcode`. If the B register is set to zero by the command, then `reg` is set to zero. The sub procedure waits until the uM-FPU is ready for the next command, then sends the command opcode. If the command requires additional data or returns data to the PICAXE it must be followed by the appropriate byte, word or string routine listed below.

Example:

```
symbol Angle = 2          'current Angle (uM-FPU register 2)

reg = Angle
opcode = SELECTA          'select Angle as the A register
gosub fpu_command

opcode = LOADPI           'load value of pi to Register 0
gosub fpu_command

opcode = FADD             'Angle = Angle + pi
gosub fpu_command
```

fpu_getByte Get byte from the uM-FPU

Parameters: none

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

Description: Reads an 8-bit value from the uM-FPU. This sub procedure is used after a fpu_command for commands that result in data being sent to the PICAXE.

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

Example:

```
opcode = FSTATUS          'get the floating point status
gosub fpu_command         ' of the A register
gosub fpu_getByte         '(status is same as dataByte)
```

fpu_getWord Get word from the uM-FPU

Parameters: none

Return: `dataWord` 16-bit value read from uM-FPU

Description: Reads a 16-bit value from the uM-FPU. This sub procedure is used after a fpu_command for commands that result in data being sent to the PICAXE.

Example:

```
symbol Result = 1        'long Result (uM-FPU register 1)

reg = Result             'get value of Result register
opcode = GET             '(in this example we only use
gosub fpu_command        ' the lower 16-bits)
gosub fpu_getWord        'read high 16-bits (and ignore)
gosub fpu_getWord        'read low 16-bits
```

---

fpu_getStr Get string from the uM-FPU and output using sertxd command

Parameters: none

Return: none

Description: A zero terminated string is read from the uM-FPU and output using the sertxd command. This sub procedure is used by the print routines and is not normally called directly by the user.

---

fpu_sendByte Send byte to the uM-FPU

Parameters: `dataByte` 8-bit value to send to uM-FPU

Return: none

Description: Sends an 8-bit value to the uM-FPU. This sub procedure is used after a fpu_command for commands that require additional data.

Example:

```
symbol inputValue = B0   '8-bit variable

opcode = LOADBYTE        'load inputValue to Register 0
gosub fpu_command        ' and convert to float
dataByte = inputValue
gosub fpu_sendByte
```

---

fpu_sendWord Send word to the uM-FPU

Parameters: `dataWord` 16-bit value to send to uM-FPU

Return: none

Description: Sends an 16-bit value to the uM-FPU. This sub procedure is used after a fpu_command for commands that require additional data.

Example:

```
opcode = LOADWORD        'load 10000 to Register 0
gosub fpu_command        ' and convert to float
dataWord = 10000
gosub fpu_sendWord
```

Print Routines

---

print_Float **Display floating point value on the PC screen**

Parameters: none

Return: none

Description: The floating point representation of the A register value is output using the `sertxd` PICAXE 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:

```
gosub print_float        'print float value
```

---

print_FloatFormat **Display formatted floating point value on the PC screen**

Parameters: `format` format specification

Return: none

Description: The formatted floating point representation of the A register value is output using the `sertxd` PICAXE command. The format is specified as a decimal value passed in the `format` 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	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:

```
format = 62              'print float value with 6.2 format
gosub print_floatFormat
```

---

print_Long Display signed long integer value on the PC screen

Parameters: none

Return: none

Description: The signed long integer representation of the A register value is output using the `sertxd` PICAXE 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:

```
gosub print_long          'print long value
```

---

print_LongFormat Display formatted long integer value on the PC screen

Parameters: `format` format specification

Return: none

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

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

The maximum width of the field is 15.

Example:

```
format = 10               'print long value with width of 10
gosub print_longFormat
```

Appendix B
uM-FPU Opcode Summary

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

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

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

Notes:

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

Appendix C
Floating Point Numbers

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

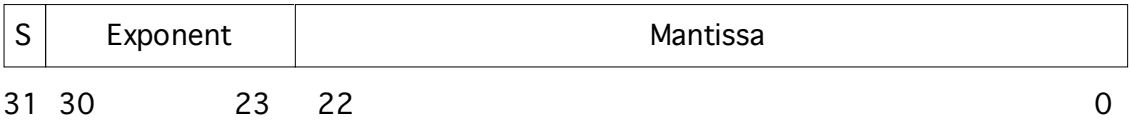
The range of numbers that can be handled by the uM-FPU is approximately $\pm 10^{38.53}$.

.

IEEE 754 32-bit Floating Point Representation

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

The 32-bit IEEE 754 representation is as follows:



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

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

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

Special Values

Zero

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

Denormalized

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

Infinity

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

Not A Number (NaN)

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

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

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