uM-FPU Application Note 1

Floating Point Calculations

Micromega Corporation

This application note shows examples of how to perform floating point calculations using the uM-FPU V2 floating point coprocessor.

Brief Summary of the uM-FPU

For a full description of the uM-FPU, please refer to the following documents: *uM-FPU V2 Datasheet*, *uM-FPU V2 Instruction Set,* and the reference documentation for each of the supported microcontrollers.

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 registers. For operations such as add, subtract, multiply and divide, the value in the A register is modified by the value in the B register and result is stored in the A register. For example:

Operation	uM-FPU Instruction	Description
Floating point set	`FSET`	A = B
Floating point add	`FADD`	A = A + B
Floating point subtract	`FSUB`	A = A – B
Floating point multiply	`FMUL`	A = A * B
Floating point divide	`FDIV`	A = A / B

The steps required to perform one of these operations on the uM-FPU are:

Select one of the 16 registers as the A register
Select one of the 16 registers as the B register
Perform the operation

The most commonly used instructions use the lower 4 bits of the opcode to select a register. This allows the instruction to select a register and perform an operation at the same time. The register value is added to the opcode definition to create the uM-FPU instruction. To select register 2 as the A register, the following instruction would be used:

```
SELECTA+2
```

The `FADD`, `FSUB`, `FMUL` and `FDIV` instructions use the lower 4 bits of the opcode to select the B register before performing the operation. To add register 1 to the A register, the following instruction would be used:

```
FADD+1
```

The full sequence of uM-FPU instructions to add register 1 to register 2 is therefore:

```
SELECTA+2
FADD+1
```

To create more readable programs, names are generally assigned to the register values. This application note uses equations with the variables *x*, *y*, and *z*, so we define three uM-FPU registers `X`, `Y` and `Z` to store these values. The method of defining symbols varies depending on the microcontroller being used. See the sample programs for specific examples. Pseudo-code is used for the following definitions:

```
define X  as 1          ; x value          uM-FPU register 1
define Y  as 2          ; y value          uM-FPU register 2
define Z  as 3          ; z value          uM-FPU register 3
define C1 as 4          ; constant c1      uM-FPU register 4
define C2 as 5          ; constant c2      uM-FPU register 5
```

Using names for the registers, the sequence of uM-FPU instructions to add X to Y is:

```
SELECTA+Y
FADD+X
```

(Note: Since the opcode definition for the `SELECTA` is `$00, SELECTA+Y` is the same as just using `Y` itself. This shortcut is generally used when writing code, but to make the examples easier to read we won't use the shortcut in this application note.)

Register 0 is used by many uM-FPU instructions to load temporary values. For example, the `LOADBYTE` instruction reads the next byte value, converts it to floating point and stores the result in register 0. Once a value is in register 0, it can easily be used by another instruction. To add 10 to `X` we would use the following instructions:

```
SELECTA+X
LOADBYTE, 10
FADD
```

The `LOADBYTE` instruction stores 10.0 in register 0, and the `FADD` instruction adds register 0 to `X`. (Note: Since `FADD` is the same as `FADD+0`, the `+0` is generally not used when writing code.)

The most common uM-FPU instructions use single byte opcodes, but there are also extended opcodes that require two bytes. 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, to set register 0 to the value 0.0, the `XOP`, `LOADZERO` instruction is used.

Examples

The following examples show how to translate common mathematical equations into the uM-FPU instructions required to perform the calculation. A brief explanation of each example is provided.

---

x = 0.0

```
SELECTA+X               ; select X as the A register
XOP, LOADZERO           ; load 0.0 to register 0
FSET                    ; X = 0.0
```

The `XOP`, `LOADZERO`, `XOP`, `LOADONE`, `XOP`, `LOADPI` and `XOP`, `LOADE` instructions provide a convenient way to load the values of several commonly used constants.

---

x = x + 1.0

```
SELECTA+X               ; select X as the A register
XOP, LOADONE            ; load 1.0 to register 0
FADD                    ; X = X + 1.0
```

This shows how convenient it is to use the value loaded in register 0 with another instruction.

---

x = x + y

```
SELECTA+X               ; select X as A register
FADD+Y                  ; X = X + Y
```

Performing an operation using two registers is very straightforward.

---

y = 5x + 30

```
SELECTA+Y               ; select Y as A register
LOADBYTE, 5             ; load 5 to register 0
                        ;  and convert to floating point
FSET                    ; Y = 5.0
FMUL+X                  ; Y = Y * X
LOADBYTE, 30            ; load 30 to register 0
                        ;  and convert to floating point
FADD                    ; Y = Y + 30.0
```

Constant values that are integers are easy to load using the `LOADBYTE`, `LOADUBYTE`, `LOADWORD` and `LOADUWORD` instructions. These instructions load an integer value, converts it to floating point, and stores the result in register 0.

---

y = 0.15x + 0.5 *(using ATOF instruction)*

```
SELECTA+C1              ; select C1 as the A register
ATOF, ".15", 0          ; load string, convert to floating point,
                        ;  and store in register 0
FSET                    ; C1 = 0.15

SELECTA+C2              ; select C2 as the A register
ATOF, ".5", 0           ; load string, convert to floating point,
                        ;  and store in register 0
FSET                    ; C2 = 0.5

SELECTA+Y               ; select Y as A register
FSET+C1                 ; Y = C1
FMUL+X                  ; Y = Y * X
FADD+C2                 ; Y = Y + C2
```

The ATOF instruction is used to convert a zero terminated string to a floating point value. (Note: make sure there is a zero terminator on the string.)

Constants can often be loaded as part of the program initialization code. In the example above, the values of `C1` and `C2` could be set in the initialization section and then only the last four lines of code would be required in the main program.

---

y = 0.15x + 0.5 (using WRITEB instruction)

```
WRITEB+C1, $3E, $19, $99, $9A       ; load 0.15 to C1
WRITEB+C2, $3F, $00, $00, $00       ; load 0.5 to C2

SELECTA+Y               ; select Y as A register
FSET+C1                 ; Y = C1
FMUL+X                  ; Y = Y * X
FADD+C2                 ; Y = Y + C2
```

The `WRITEB` instruction is a very efficient way of loading a floating point value to a uM-FPU register, but it requires that you know the 32-bit representation for the floating point number. A conversion program, *uM-FPU Converter*, is available for download, and provides a convenient way to convert between floating point strings and the 32-bit floating point representation.

$y = x^2$

```
SELECTA+Y               ; select Y as A register
FSET+X                  ; Y = X
FMUL+X                  ; Y = Y * X
```

To calculate the square of a value you can just multiply the number by itself.

---

y = sin(x)

```
SELECTA+Y               ; select Y as A register
FSET+X                  ; Y = X
SIN                     ; Y = sin(Y)
```

Note: On the BASIC Stamp and some other microcontrollers the `SIN`, `COS`, and `TAN` instruction are defined as `FSIN`, `FCOS`, and `FTAN` to avoid conflict with reserved names.

The `SIN` instruction is an example of an instruction that operates only on the A register. It calculates the sine of the value in the A register and stores the result in the A register. If the original needs to be retained, you should first assign it to another variable as shown above.

---

y = x / y

```
SELECTA+Y               ; select Y as A register
XOP, LEFT               ; select temp register as A register
FSET+X                  ; temp = X
FDIV+Y                  ; temp = temp / Y
XOP, RIGHT              ; store temp to register 0,
                        ;  restore Y as A register
FSET                    ; Y = temp
```

In the previous examples, we have been able to use the variable on the left side of the equation to hold the partial results as each step of the equation is calculated. In the example above, this is not possible because the value on the left is used in the equation and can't be modified until after it is used. Fortunately, the uM-FPU provides temporary registers through the use of parentheses. The original equation can be rewritten as *y = (x / y),* and the calculation inside the parentheses uses a temporary register. Here's how it works. When a `XOP, LEFT` parenthesis instruction is sent, the current selection for the A register is saved, and the A register is set to a temporary register. Operations can now be performed as normal, with the temporary register selected as the A register. When a `XOP, RIGHT` parenthesis instruction is sent, the current value of the A register is copied to register 0, and the previous selection for the A register is restored. Although it sounds complicated, the sequence of instructions is quite straightforward, a left and right parenthesis simply enclose the temporary value calculations.

---

z = (x + y) / 10

```
SELECTA+Z               ; select Z as A register
FSET+X                  ; Z = X
FADD+Y                  ; Z = Z + Y
LOADBYTE, 10            ; load 10 to register 0
                        ;  and convert to floating point
FDIV                    ; Z = Z / 10.0
```

The uM-FPU executes instructions in the order in which they occur. In the example above, the parentheses are not necessary. The addition is done first, followed by the divide.

---

y = (x + 1) / (x – 1)

```
SELECTA+Y               ; select Y as A register
FSET+X                  ; Y = X
XOP, LOADONE            ; set register 0 to 1.0
FADD                    ; Y = Y + 1.0
XOP, LEFT               ; select temp register as A register
FSET+X                  ; temp = X
XOP, LOADONE            ; set register 0 to 1.0
FSUB                    ; temp = temp - 1.0
XOP, RIGHT              ; store temp to register 0,
                        ;  restore Y as A register
FDIV                    ; Y = Y / temp
```

In the example above, the first set of parentheses are not necessary, since *y* can be used to hold the result of the first part of the equation *(x + 1)*. The second part of the equation requires parentheses to calculate the temporary value *(x – 1)*.

---

z = x / (y + z)

```
SELECTA+Z               ; select Z as A register
XOP, LEFT               ; select temp register as A register
FSET+X                  ; temp = X
XOP, LEFT               ; select temp2 register as A register
FSET+Y                  ; temp2 = Y
FADD+Z                  ; temp2 = temp2 + Z
XOP, RIGHT              ; store temp2 to register 0,
                        ;  restore temp as A register
```

```
FDIV                    ; temp = temp / temp2
XOP, RIGHT              ; store temp to register 0,
                        ;  restore Z as A register
FSET                    ; Z = temp
```

In the example above, *z* is used on the right of the equation so a temporary value is required. The equation can be rewritten as *z = (x / (y + z))*. The uM-FPU supports up to five levels of nested parentheses.

---

z = sqrt($x^2 + y^2$)

```
SELECTA+Z               ; select Z as A register
XOP, LEFT               ; select temp register as A register
FSET+X                  ; temp = X
FMUL+X                  ; temp = temp * X
XOP, RIGHT              ; store temp to register 0,
                        ;  restore Z as A register
FSET                    ; Z = temp
XOP, LEFT               ; select temp register as A register
FSET+Y                  ; temp = Y
FMUL+Y                  ; temp = temp * Y
XOP, RIGHT              ; store temp to register 0,
                        ;  restore Z as A register
FADD                    ; Z = Z + temp
SQRT                    ; Z = sqrt(Z)
```

Calculating x^2 and y^2 requires temporary values, so the equation can be rewritten as $z = sqrt((x^2) + (y^2))$.

Read 100 unsigned 8-bit samples from a sensor, and calculate the average.

The method of implementing the loop and reading the sensor varies with the microcontroller and sensor used. See the sample programs for specific examples. Pseudo-code is used below.

```
SELECTA+X               ; select X as the A register
XOP, LOADZERO           ; load 0.0 to register 0
FSET                    ; X = 0.0

loop for i = 1 to 100
{
   read sensor value and store in svalue
   wait for uM-FPU to be ready
   LOADUBYTE, svalue    ; load unsigned 8-bit value to register 0
                        ;  and convert to floating point
   FADD                 ; X = X + svalue (as floating point)
}
LOADUBYTE, 100          ; load 100.0 to register 0
FDIV                    ; X = X / 100.0
```

In this example, `X` is used to accumulate the sum of 100 sensor readings. The sum is then divided by 100 to calculate the average value. Note: Since each reading sends 3 bytes to the uM-FPU and there are 100 passes through the loop, it is important to check the status of the uM-FPU inside the loop to ensure that the 32 byte uM-FPU instruction buffer is not exceeded.