uM-FPU Application Note 2

Long Integer Calculations

Micromega Corporation

This application note shows examples of how to perform long integer (32-bit) 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
Long integer set	`LSET`	A = B
Long integer add	`LADD`	A = A + B
Long integer subtract	`LSUB`	A = A – B
Long integer multiply	`LMUL`	A = A * B
Long integer divide	`LDIV`	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 `LADD`, `LSUB`, `LMUL` and `LDIV` 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:

```
LADD+1
```

The full sequence of uM-FPU instructions to add register 3 to register 2 is therefore:

```
SELECTA+2
LADD+1
```

To create more readable programs, names are generally assigned to the register values. This application note uses equations with the variables *m*, *n*, and *t*, so we define three uM-FPU registers `M`, `N` and `T` 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 M  as 1          ; m value          uM-FPU register 1
define N  as 2          ; n value          uM-FPU register 2
define T  as 3          ; t value          uM-FPU register 3
```

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

```
SELECTA+N
LADD+M
```

(Note: Since the opcode definition for the `SELECTA` is `$00, SELECTA+N` is the same as just using `N` 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.)

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, the `XOP`, `LOADZERO` instruction is used.

Register 0 is used by many uM-FPU instructions to load temporary values. For example, the `XOP`, `LONGBYTE` instruction reads the next byte value, converts it to a long integer 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 `N` we would use the following instructions:

```
SELECTA+N
XOP, LONGBYTE, 10
LADD
```

The `XOP`, `LONGBYTE` instruction stores 10 in register 0, and the `LADD` instruction adds register 0 to `N`. (Note: Since `LADD` is the same as `LADD+0`, the `+0` is generally not used when writing code.)

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.

---

n = 0

```
SELECTA+N               ; select N as the A register
XOP, LOADZERO           ; load 0 to register 0
LSET                    ; N = 0
```

The `XOP`, `LOADZERO` instructions provide a convenient way to load the value 0.

---

n = n + 1

```
SELECTA+N               ; select N as the A register
XOP, LINCA              ; increment the A register
```

Alternatively, the XOP, LINCB instruction can be used if you don't want to change the currently selected A register.

```
SELECTB+N               ; select N as the B register
XOP, LINCB              ; increment the B register
```

---

n = n - 1

```
SELECTA+N               ; select N as the A register
XOP, LDECA              ; decrement the A register
```

Alternatively, the XOP, LDECB instruction can be used if you don't want to change the currently selected A register.

```
SELECTB+N               ; select N as the B register
XOP, LDECB              ; decrement the B register
```

---

m = m + n

```
SELECTA+M               ; select M as A register
LADD+N                  ; M = M + N
```

Performing an operation using two registers is very straightforward.

---

t = m / n

```
SELECTA+T               ; select T as A register
LSET+M                  ; T = M
LDIV+N                  ; T = T / N
```

The LDIV instruction performs a signed division. The XOP, LUDIV instruction is also available to perform an unsigned division.

---

t = m mod n (remainder of m / n)

```
SELECTA+T               ; select T as A register
LSET+M                  ; T = M
LDIV+N                  ; T = T / N
LSET                    ; T = remainder of M / N
```

Both the LDIV and XOP, LUDIV instructions store the result of the integer division in the A register and the remainder of the division in register 0.

---

n = 5m + 30

```
SELECTA+N                ; select N as A register
XOP, LONGBYTE, 5         ; load 5 to register 0
                         ;  and convert to long integer
LSET                     ; N = 5
LMUL+M                   ; N = N * M
XOP, LONGBYTE, 30        ; load 30 to register 0
                         ;  and convert to long integer
LADD                     ; N = N + 30
```

Constant values that are integers are easy to load using the XOP, LONGBYTE, XOP, LONGUBYTE, XOP, LONGWORD and XOP, LONGUWORD instructions. These instructions load an integer value, converts it to long integer, and stores the result in register 0.

---

m = n / 500000 (using ATOL instruction)

```
SELECTA+M                ; select M as the A register
LSET+N                   ; M = N
ATOL, "500000", 0        ; load string, convert to long integer,
                         ;  and store in register 0
LDIV                     ; M = M / 500000
```

The ATOL instruction is used to convert a zero terminated string to a long integer value. (Note: make sure there is a zero terminator on the string.)

---

m = n / 500000 (using LWRITEB instruction)

```
SELECTA+M                ; select M as the A register
LSET+N                   ; M = N
XOP, LWRITEB, $00, $07, $A1, $20
                         ; load 500000 to register 0
LDIV                     ; M = M / 500000
```

The XOP, LWRITEB instruction is a very efficient way of loading a long integer value to a uM-FPU register. It is often easier to express a 32-bit long integer as a hexadecimal number since most microcontroller compilers only support 8-bit and 16-bit values.

---

m = n^2

```
SELECTA+M                ; select M as A register
LSET+N                   ; M = N
LMUL+N                   ; M = M * N
```

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

---

n = m / n

```
SELECTA+N               ; select N as A register
XOP, LEFT               ; select temp register as A register
LSET+M                  ; temp = M
LDIV+N                  ; temp = temp / N
XOP, RIGHT              ; store temp to register 0,
                        ;  restore N as A register
LSET                    ; M = 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 *n = (m / n),* 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.

---

t = (m + n) / 10

```
SELECTA+T               ; select T as A register
LSET+M                  ; T = M
LADD+N                  ; T = T + N
XOP, LONGBYTE, 10       ; load 10 to register 0
                        ;  and convert to long integer
LDIV                    ; T = T / 10
```

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.

---

t = m / (n + t)

```
SELECTA+T               ; select T as A register
XOP, LEFT               ; select temp register as A register
LSET+M                  ; temp = M
XOP, LEFT               ; select temp2 register as A register
LSET+N                  ; temp2 = N
LADD+T                  ; temp2 = temp2 + T
XOP, RIGHT              ; store temp2 to register 0,
                        ;  restore temp as A register
LDIV                    ; temp = temp / temp2
XOP, RIGHT              ; store temp to register 0,
                        ;  restore T as A register
LSET                    ; T = temp
```

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

Extract the value of a range of bits from a long integer
m = bits 20-23 of n (right justified)

```
SELECTA+M               ; select M as A register
LSET+N                  ; M = N
XOP, LONGBYTE, -20      ; load -20 to register 0
                        ;  and select register 0 as B register
XOP, LSHIFT             ; shift M left by 20 bits
XOP, LONGBYTE, $0F      ; set lower 4 bits of register 0 to ones
XOP, LAND               ; mask off the other bits
```

The XOP, LSHIFT instruction shifts the value in the A register by the number of bits specified in the B register. If the value in the B register is positive, a left shift occurs, if the value is negative, a right shift occurs. The XOP, LAND instruction sets the A register to the result of a logical AND of the A register and the B register.

Extract time values from a long integer time count

A counter is often used to keep track of time by counting the number of ticks of a clock. In this example we assume that the clock ticks once per second. A long integer is used to keep track of the number of seconds, which can store $2^{32} - 1$, or 4,294,967,295 seconds. This provides for an elapsed time of over 136 years. The following example shows how the number of days, hours, minutes and seconds can be extracted from the time count by using a series of divisions and remainders.

```
define Days    as 4     ; days            uM-FPU register 4
define Hours   as 5     ; hours           uM-FPU register 5
define Minutes as 6     ; minutes         uM-FPU register 6
define Seconds as 7     ; seconds         uM-FPU register 7

SELECTA+Days            ; select Days as A register
LSET+T                  ; Days = T
XOP, LWRITEB, $00, $01, $51, $80  ; load 86400 to register 0
XOP, LUDIV              ; Days = Days / 86400

SELECTA+Hours           ; select Hours as A register
LSET                    ; Hours = remainder of Days / 86400
XOP, LONGWORD, $0E, $10 ; load 3600 to register 0
LDIV                    ; Hours = Hours / 3600

SELECTA+Minutes         ; select Minutes as A register
LSET                    ; Minutes = remainder of Hours / 3600
XOP, LONGBYTE, 60       ; load 60 to register 0
LDIV                    ; Minutes = Minutes / 60

SELECTA+Seconds         ; select Seconds as A register
LSET                    ; Seconds = remainder of Minutes / 60
```

The first step is to find the number of days by dividing the time value by 86,400 (the number of seconds in a day). The remainder is then divided by 3600 to get the number of hours. This remainder is divided by 60 to get the number of minutes, and the last remainder is the number of seconds. The XOP, LUDIV unsigned divide instruction is used for the first divide because the most significant bit could be a one, but the shorter LDIV instruction is used for the other divides because the most significant bit will always be zero.

Read 100 unsigned 16-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+N                ; select N as the A register
XOP, LOADZERO            ; load 0 to register 0
LSET                     ; N = 0

loop for i = 1 to 100
{
   read sensor value and store in svalue
   XOP, LONGBYTE,        ; load signed 16-bit value to register 0
   svalue(high byte)     ;  and convert to long integer
   svalue(low byte)
   LADD                  ; N = N + svalue
}
XOP, LONGBYTE, 100       ; load 100 to register 0
LDIV                     ; N = N / 100
```

In this example, `N` is used to accumulate the sum of 100 sensor readings. The sum is then divided by 100 to calculate the average value.