Micromega Corporation

Application Note 32

uM-FPU V3 Long Integer Calculations

This application note shows some examples of how to perform long integer calculations using the uM-FPU V3 floating point coprocessor. See *Application Note 31 - uM-FPU V3 Floating Point Calculations* for floating point examples and the *uM-FPU V3 Instruction Reference* for a full description of all instructions.

The *uM-FPU V3 IDE (Integrated Development Environment)* software provides a code generator that automatically produces uM-FPU V3 code from standard mathematical equations. It also generates code for various supported microcontrollers. Using the *uM-FPU V3 IDE* is an excellent way of becoming familiar with uM-FPU V3 instruction set, and is very useful for implementing your own equations and calculations. The *uM-FPU IDE* is available for download on the Micromega website.

Brief Overview of uM-FPU V3 chip

For a full description of the uM-FPU V3 chip, please refer to the *uM-FPU V3 Datasheet*, *uM-FPU V3 Instruction Reference,* and the reference documentation for each of the supported microcontrollers.

The uM-FPU V3 chip is a separate coprocessor with its own set of registers and instructions designed to provide microcontrollers with 32-bit floating point and long integer capabilities. The microcontroller communicates with the uM-FPU using a SPI or I^2C interface. The microcontroller sends instructions and data to the uM-FPU, and the uM-FPU performs the calculations. The microcontroller is free to do other tasks while the uM-FPU performs calculations. Results can be read back to the microcontroller or stored on the uM-FPU for later use. The uM-FPU V3 chip has 128 registers, numbered 0 through 127, that can hold 32-bit floating point or long integer values. Register 0 is often used as a temporary register and is modified by some of the uM-FPU V3 instructions. Registers 1 through 127 are available for general use.

Arithmetic instructions use the value in register A as an operand and store the result of the operation in register A. Register A can be regarded as an accumulator or working register. The `SELECTA` instruction is used to select any one of the 128 registers as register A. If an instruction requires more than one operand, the additional operand is specified by the instruction. For example,

uM-FPU Instruction	Description
`LSET, 5`	register[A] = register[5]
`LADD, 10`	register[A] = register[A] + register[10]
`LSUB, 64`	register[A] = register[A] – register[64]
`LMUL, 1`	register[A] = register[A] * register[1]
`LDIV, 16`	register[A] = register[A] / register[16]

Each of the basic long integer arithmetic instructions are provided in three different forms as shown in the table below. For example, `LADD,nn` allows any general purpose register to be added to register A. The nn byte following the opcode specifies the register to be added. The `LADD0` instruction only requires the opcode and adds register 0 to register A. The `LADDI` instruction adds a small integer value to register A. The signed byte (-128 to 127) following

the opcode is converted to long integer and added to register A. The `LADD,nn` instruction is most general, but the `LADD0` and `LADDI,bb` instructions are more efficient for many common operations.

Register nn	Register 0	Immediate value	Description
`LSET,nn`	`LSET0`	`LSETI,bb`	Set
`LADD,nn`	`LADD0`	`LADDI,bb`	Add
`LSUB,nn`	`LSUB0`	`LSUBI,bb`	Subtract
`LMUL,nn`	`LMUL0`	`LMULI,bb`	Multiply
`LDIV,nn`	`LDIV0`	`LDIVI,bb`	Divide
`LCMP,nn`	`LCMP0`	`LCMPI,bb`	Compare
`LUDIV,nn`	`LUDIV0`	`LUDIV,bb`	Unsigned Divide
`LUCMP,nn`	`LUCMP0`	`LUCMP,bb`	Unsigned Compare
`LTST,nn`	`LTST0`	`LTST,bb`	Test Bits

The uM-FPU V3 instructions to add register 1 to register 2 is as follows:

```
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 V3 instructions for $n = m$ is:

```
SELECTA, N
LADD, M
```

The uM-FPU V3 instructions required for various examples are shown on the following pages. The interface routines for sending the uM-FPU V3 code depends on the microcontroller being used. Please see the reference guides for the various supported microcontrollers for more specific information.

Examples

The following examples show how to translate common mathematical equations into the uM-FPU V3 instructions required to perform the calculation. A brief explanation of each example is provided. The *uM-FPU V3 IDE* software can be used to automatically generate code for standard mathematical equations.

n = 0

The CLRA instruction can be used to set a register to 0.

```
SELECTA, N              ; select N as register A
CLRA                    ; N = 0
```

n = -10.0

Immediate instructions can be used for small integer values.

```
SELECTA, N              ; select N as register A
LSETI, -10              ; N = -10.0
```

n = n + 1

The LINC and LDEC instruction can be used to increment or decrement a register without affecting register A.

```
LINC, N                 ; increment register N
```

n = n - 1

```
LDEC, N                 ; increment register N
```

m = m + n

Performing an operation using two registers is very straightforward.

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

t = m / n

The LDIV instruction performs a signed division and the LUDIV instruction performs an unsigned division. The result is stored in register A and the remainder is stored in register 0.

```
SELECTA, T              ; select T as register A
LSET, M                 ; T = M
LDIV, N                 ; T = T / N, register[0] = remainder
```

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

```
SELECTA, T              ; select T as register A
LSET, M                 ; T = M
```

```
LDIV, N                 ; T = T / N, register[0] = remainder
LSET0                   ; T = register[0]
```

n = 5m + 30

Constant values that are small integers can be easily handled using the immediate instructions. These instructions load an integer value, convert it to long integer, and store the result in register 0.

```
SELECTA, N              ; select N as A register
LSETI, 5                ; N = 5
LMUL, M                 ; N = N * M
LADD, 30                ; N = N + 30
```

n = 512

The LONGBYTE, LONGUBYTE, LONGWORD and LONGUWORD instructions can also be used to load constants. These instructions load an integer value, converts it to long integer, and stores the result in register 0. The LONGWORD and LONGUWORD instructions are used for 16-bit values and require two 8-bit bytes after the opcode.

```
SELECTA, N              ; select N as register A
LONGWORD, 2, 0          ; register[0] = 512 (high byte, low byte)
LSET0                   ; N = register[0]
```

m = n / 500000 (using ATOL instruction)

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.)

```
SELECTA, M              ; select M as register A
LSET, N                 ; M = N
ATOL, "500000", 0       ; load string, convert to long integer,
                        ;  and store in register 0
LDIV0                   ; M = M / register[0]
```

m = n / 500000 (using LWRITE instruction)

The LWRITE 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. The automatic code generation provided by the *uM-FPU V3 IDE* is a convenient way to generate long integer constants.

```
SELECTA, M              ; select M as register A
LWRITE0, $00, $07, $A1, $20
                        ; register[0] = 500000
LDIV0                   ; M = M / register[0]
```

m = n^2

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

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

n = m / n

In the previous examples, we've 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 `LEFT` parenthesis instruction is sent, the current selection for register A is saved, and a temporary register is set to register A. Operations can now be performed as normal, with the temporary register selected as register A. When a `RIGHT` parenthesis instruction is sent, the current value of register A is copied to register 0, and the previous selection for register A is restored. Although it sounds complicated, the sequence of instructions is quite straightforward, a `LEFT` and `RIGHT` parenthesis simply enclose the temporary value calculations.

```
SELECTA, N              ; select N as register A
LEFT                    ; select temp1 as register A
LSET, M                 ; temp1 = M
LDIV, N                 ; temp1 = temp1 / N
RIGHT                   ; register[0] = temp1, and
                        ;  restore N as register A
LSET0                   ; M = register[0]
```

t = (m + n) / 10

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

```
SELECTA, T              ; select T as register A
LSET, M                 ; T = M
LADD, N                 ; T = T + N
LDIVI, 10               ; T = T / 10
```

t = m / (n + t)

In the following example, *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 eight levels of nested parentheses.

```
SELECTA, T              ; select T as register A
LEFT                    ; select temp1 as register A
LSET, M                 ; temp = M
LEFT                    ; select temp2 as register A
LSET, N                 ; temp2 = N
LADD, T                 ; temp2 = temp2 + T
RIGHT                   ; register[0] = temp2, and
                        ;  restore temp1 as register A
LDIV0                   ; temp1 = temp1 / temp2
RIGHT                   ; register[0] = temp1, and
                        ;  restore T as register A
LSET                    ; T = temp1
```

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

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

```
SELECTA, M              ; select M as register A
LSET, N                 ; M = N
LONGBYTE, -20           ; register[0] = -20
LSHIFT, 0               ; shift M left by 20 bits
LONGBYTE, $0F           ; register[0] = $0F
LAND, 0                 ; M = lower four bits of M
```

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 (T) by using a series of divisions and remainders. 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 LUDIV unsigned divide instruction is used since the elapsed time is unsigned.

```
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 register A
LSET, T                 ; Days = T
LWRITE0, $00, $01, $51, $80  ; load 86400 to register 0
LUDIV0                  ; Days = Days / 86400

SELECTA, Hours          ; select Hours as register A
LSET0                   ; Hours = remainder of Days / 86400
LONGWORD, $0E, $10      ; load 3600 to register 0
LDIV0                   ; Hours = Hours / 3600

SELECTA, Minutes        ; select Minutes as A register
LSET0                   ; Minutes = remainder of Hours / 3600
LONGBYTE, 60            ; load 60 to register 0
LDIV0                   ; Minutes = Minutes / 60

SELECTA, Seconds        ; select Seconds as A register
LSET0                   ; Seconds = remainder of Minutes / 60
```

Read 100 unsigned 16-bit samples from a sensor, and calculate the average.

The method of implementing the loop and reading the sensor will vary depending on the microcontroller sensor used. See the sample programs for specific examples. Pseudo-code is shown below. In this example, `N` is used to accumulate the sum of 100 16-bit sensor readings. The sum is then divided by 100 to calculate the average value.

```
SELECTA, N              ; select N as register A
CLR0                    ; register[0] = 0

loop for i = 1 to 100
{
   read sensor value and store in svalue
   LONGBYTE,            ; register[0] = signed 16-bit value
   svalue               ; (high byte)
   svalue               ; (low byte)
   LADD0                ; N = N + svalue
}
LDIVI, 100              ; N = N / 100
```

Further Information

Check the Micromega website at www.micromegacorp.com for up-to-date information.