FLOATING POINT HARDWARE SUPPORT ON MICROCONTROLLERS

A Design Project Report
Presented to the Engineering Division of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering (Electrical)

by
Kenny Chi Kin Lo
Project Advisor: Bruce Land
Degree Date: August 2006

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Floating Point Hardware Support on Microcontrollers

Author: Kenny Chi Kin Lo

Abstract: A piece of hardware was connected to the Atmel Mega32 microcontroller such that performance on floating point calculations was enhanced. Device level and high level user routines were developed for the hardware communication and the user applications. The success of this project allows microcontrollers to perform more complex calculations, although not faster. Computer architecture, hardware programming, floating point calculations, and microcontroller knowledge are the basic design skills for this project.

Report Approved by

Project Advisor: _____Date: _____

Executive Summary

Floating point calculation is a major limitation on complex embedded system designs because many microcontrollers do not have floating point support on chip. A convenience solution, which is commonly used, includes software package to simulate floating point calculations. However, this solution takes up significant amount of microcontroller resources, and, for the CodeVision compiler, it induces rounding discrepancy.

An alternative solution was developed using hardware approach. A piece of hardware, 8-pin uM-FPU V2 from Micromega Corporation, was used to perform floating point calculations. Appropriate configurations were required to use the hardware. Driver of hardware and high level calculation routines were developed such that future microcontroller designs could use floating point hardware to perform powerful calculations. Routines that simplify user defined function were developed such that it allows more flexibility to calculation applications.

The floating point hardware was implemented successfully on the Atmel Mega32 microcontroller, which satisfied the project goal. The driver and high level routines developed in this project can apply to Atmel Mega32 as well as any microcontrollers with three-wire SPI support. Computer architecture, hardware programming, floating point calculations, and microcontroller knowledge are the basic design skills for this project.

Table of Contents

1. Design Problem and Requirements

Microcontroller Mega32 has no floating point support on the chip. However, many applications use floating point calculations. The project goal was to provide a hardware solution on the microcontrollers such that the performance on floating point calculation could be improved. The success of this project would allow microcontrollers to perform more complicated applications.

The requirements for the floating point unit implementation were successfully achieved. The fundamental objectives of this project include:

- Install a floating point unit on the microcontroller
- Develop the driver for the floating point hardware
- Provide floating point support routines in the user level
- Simplify user defined function development

2.1 Design Constraints

The floating point unit that was used in this project does not have compatible driver written for Mega32 microcontrollers. The closest reference driver provided from the floating point unit manufacturer is written in assembly language. Understanding the assembly code is very time consuming, but it helps greatly on floating point hardware driver development. Since no compatible driver has been written, I developed the driver using C code.

2. Range of Solutions

The design problem is to provide floating point support on the microcontrollers. There are two possible solutions: Software and Hardware support.

2.1 Software Support

The software solution is already developed under C library. This solution is to include the Math package in the programs which use floating point calculations. The idea behind this solution is to simulate the floating point calculations using the integer calculations that can execute in the microcontrollers, and convert the integer results back to the floating point results. Software solution is simple, costless, and convenient.

However, the disadvantages of this solution are slow and processor intensive. For example, some simulated floating point instructions, such as sine function, take up to few thousand clock cycles. While the processor is doing calculation, it cannot perform other tasks because the microcontroller is running the simulated integer calculations.

Besides the above disadvantages, the accuracy is another problem using software solution. Observation from the software and hardware testing comparison suggests that some functions from CodeVision Math.h package are inaccurate in the last few digits of the floating point numbers. Verification shows that the software simulation is off by a small amount in some calculations, which means the number conversion during the calculations could induce a small error.

2.2 Hardware Support

The hardware solution was used on this project. The idea is to implement a separate piece of floating point unit on the microcontroller. This solution is less processor intensive and provides more accurate results. Once a floating point calculation is called, the microcontroller sends the data towards the FPU which performs floating point calculation immediately. It returns the result to the microcontroller when the calculation is completed. While the FPU is performing calculation, the microcontroller can be used in other tasks. The disadvantages of hardware solution are complicated setup and relatively expensive ($14.95).

2.2.1 Communication

The most challenging part of the hardware solution is to setup the communication. The uM-FPU that was used in the project supports two communication schemes: Serial Peripheral Interface (SPI) and Intel-Integrated Circuit bus (I^2C). SPI interface was used in the implementation because of the higher data transmitting speed and the built-in SPI interface on Mega32 microcontroller. Three wires SPI can handle data speeds up to 4 MHz, while two wires I^2C can handles data speeds up to 400 kHz.

2.2.2 Driver

The reference driver obtained from the FPU manufacturer is written in assembly language. It is very costly to modify the setting for other kinds of microcontroller. Therefore, C code version of the driver was developed. The driver includes several device level routines which are essential to make use of the floating point unit (See Section 4.2.1). This driver allows future modification and maintenance more convenience.

3. Design and Implementation

3.1 Hardware

3.1.1 Microcontroller (MCU)

The Atmel Mega32 microcontroller chip was used in this project. Development board STK500 was used to run C code and flash the code to the chip. Port pin B5, B6 and B7 were set for SPI communication (See section 4.1.3). A jumper was used to connect port pin D0 and D1 to the RS232-spare header for Serial communication that display testing messages at terminal programs. In order to receive signals correctly, terminal programs, such as Hyper terminal, should be set to 9600 baud, no parity, one stop bit, and no flow control.

Port	Pin	Direction	Purpose
A	0…7	-	Not Used
B	0…4 5 6 7	- Output Input Output	Not Used SCK MISO MOSI
C	0…7	-	Not used
D	0, 1 2…7	Output -	Serial jumper Not Used

Table 1 – Port Configuration

3.1.2 Floating Point Unit (FPU)

The 8-pin DIP uM-FPU V2 floating point coprocessor from Micromega Corporation was used as the core component. This hardware provides support for 32-bit IEEE 754 compatible floating point and 32-bit long integer calculations. It supports I^2C and SPI interface which is easily communicate with most microcontrollers. The uM-FPU

uses internal oscillator and does not require external hardware. There are two subordinate adapters; FPU debug adapter and RS-232 serial adapter, which are very useful for debugging and monitoring purposes using the uM-FPU IDE application provided. The features of uM-FPU V2 include:

- 8-Pin DIP integrated circuit
- I^2C compatible interface up to 400 kHz
- SPI compatible interface up to 4 Mhz
- 32 byte instruction buffer
- Sixteen 32-bit general purpose registers
- 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
    - Compare, Unsigned Compare, Status

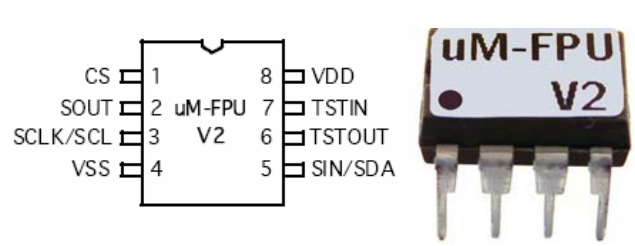


Figure 1 – Pin Diagram and uM-FPU

Pin	Name	Type	Description
1	CS	Input	Chip Select
2	SOUT	Output	SPI Output Busy/Ready
3	SCLK SCL	Input	SPI Clock I^2C Clock
4	VSS	Power	Ground
5	SIN SDA	Input In/Out	SPI Input I^2C Data
6	TSTOUT	Output	Test Output
7	TSTIN	Input	Test Input
8	VDD	Power	Supply Voltage

Table 2 – Pin Description

3.1.3 SPI Interface

The Serial Peripheral Interface (SPI) is one form of serial communication available in the AVR microcontrollers. It is a synchronous serial communication bus such that transmitter and receiver involved in SPI communication use the same clock to synchronize the detection of the bits at the receiver. The SPI bus provides relatively high-speed for short distance communication using a minimum number of microcontroller pins.

There are two kinds of SPI interface: 3-wire and 2-wire SPI. The 3-wire SPI interface uses separate data input and output pins on the microcontroller, while the 2-wire SPI interface uses a single bidirectional pin for both data input and output. It is easier to debug a 3-wire interface because the input and output data are always clearly distinguishable. Mega32 has built-in 3-wire SPI hardware support, and the I/O pins are predefined at D5, D6, and D7. Therefore, taking the advantages, 3-wire SPI was used in this project.

SPI communication involves a master device and a slave device. Both the master and a slave send and receive data simultaneously, but the master provides the clock signal for the data transfer. Therefore, the master has the speed control of data transfer, and thus, the control of overall data transfer.

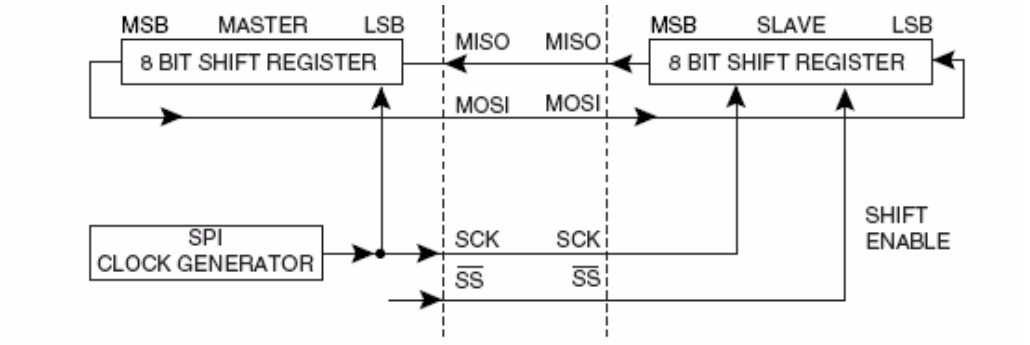


Figure 2 – SPI Communication Scheme

The connections between the master and the slave devices are shown in Figure 2. The master provides the clock signal SCK at pin B5 and eight bits of data, which are shifted out of the master-out-slave-in (MOSI) pin at B7. The same eight bits are shifted into the slave device, one bit per clock pulse, on it MOSI line. As the eight bits are

shifted out of the master and into the slave, eight bits are also shifted out of the slave on its mater-in-slave-out (MISO) pin at B6. SPI communication is a circle in which eight bits flow from the master to the slave and a different set of eight bits flows from the slave to the master. Both master and slave devices exchange data in a single communication. The Atmel Mega32 microcontroller was configured as a master devices and the uM-FPU was configured as a slave device.

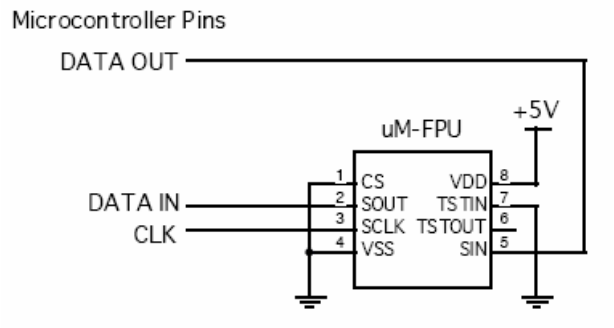


Figure 3 – 3-wire SPI Connection

Microcontoller Pin	Pin location	uM-FPU Pin	Description
FPU_CLOCK	B7	SCLK	clock
FPU_DATAIN	B6	SIN	input to MCU (MISO), output from FPU
FPU_DATAOUT	B7	SOUT	output from MCU (MOSI), input to FPU

Table 3 – SPI Pin Description

3.2 Software

3.2.1 FPU Driver Development

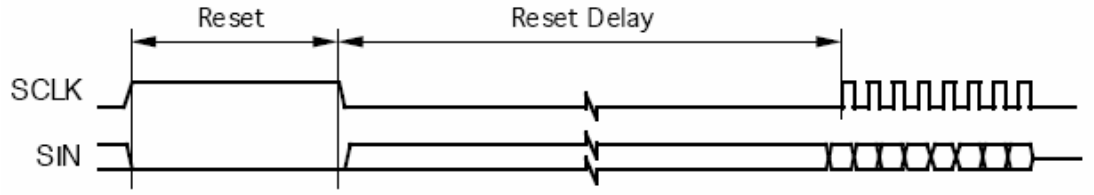
After setting up the hardware, driver must be developed to configure the appropriate settings. Device level support routines were the fundamental communication interface using SPI, and were developed under the file umfpu-v2.h (See Appendix at Section 7.3). There were seven steps for the implementation.

1. Implement initial version of fpu_reset that configures and resets MCU and FPU
2. Implement fpu_send that sends a byte through SPI
3. Implement fpu_wait that waits until the buffer is empty and the result is ready
4. Implement fpu_readDelay that is required for read setup delay
5. Implement fpu_read that reads a byte from FPU through SPI
6. Add a synchronization check to fpu_reset
7. Create FPU opcode definition

3.2.1.1 Implement initial version of fpu_reset routine

This routine configures the direction of the pins which are used for SPI interface, and enables SPI control register, SPCR, with correct settings. The uM-FPU must be reset at the start of every program to establish synchronization with microcontroller. The fpu_reset routine sends a reset pulse, waits for reset completion, checks for proper synchronization and then reads the response byte. Since fpu_send and fpu_read routines have not been developed at this stage, the synchronization check will be added at step 6. To make a reset, the SIN line must be set Low while the SCLK line is set High for a minimum of 500 microseconds. The reset operation will not occur until the SCLK line returns Low. A delay of 8 milliseconds is recommended after the Reset to ensure that the Reset is complete. All uM-FPU registers are reset to the special value NaN (Not a Number), which is equal to hexadecimal value 0x7FC00000.

C prototype: unsigned char fpu_reset (void);
Return: sync character (0x5C), if reset is successful.



Item	Min	Typical	Max	Unit
Reset - SCLK Output High	500			usec
Reset - SIN Output Low	100			usec
Reset Delay	3	8		msec

Figure 4 – Reset Timing Diagram

3.2.1.2 Implement fpu_send routine

The fpu_send routine sends a byte to the FPU through SPI Mode 0 format. The SPI configuration was done at step 1. To send data using SPI, assigns the 8-bit data byte to the SPI data register (SPDR), and waits for the transmission complete. Checks SPI status register (SPSR) to determine a successful transmission.

Oscilloscope was used to confirm the timing parameters meet the specifications. The minimum data period should be 10 microseconds. A delay is required at the end of the fpu_send routine to ensure the minimum data period is provided. The minimum data period is the time from the beginning of one byte to the beginning of the next byte. In our setting, a delay of 5 microseconds is called to guarantee minimum data period is achieved.

C prototype: void fpu_send (unsigned char dataByte);
Return: none

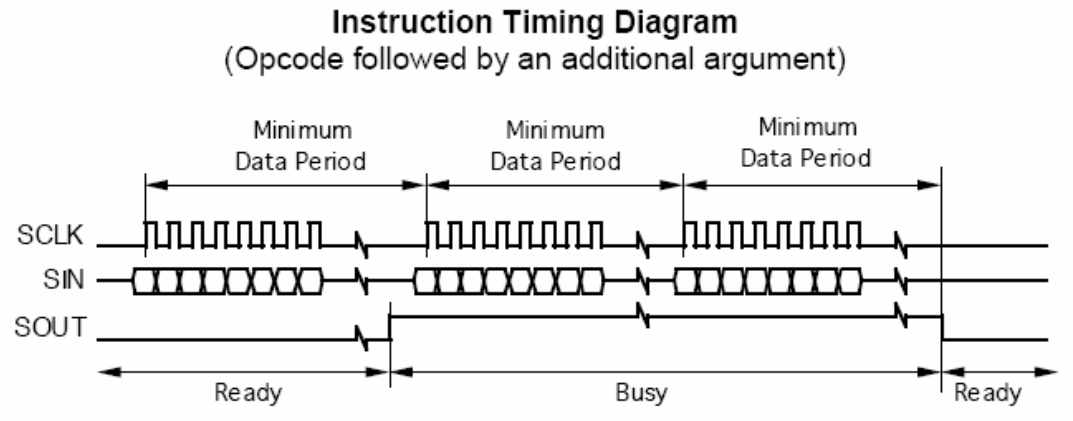


Figure 5 – Minimum Data Period

3.2.1.3 Implement fpu_wait routine

The fpu_wait routine is used to ensure that the 32-byte instruction buffer is empty, and the result is ready for access. Therefore, it must be called at least once for every 32 bytes of data sent and before data is read from uM-FPU. The Busy status is asserted as soon as a valid opcode is received. The Ready status is asserted when the instruction buffer is empty. If the uM-FPU is Ready, the SOUT pin is held Low. If the uM-FPU is Busy, the SOUT pin is held High. This routine stalls the microcontroller when the uM-FPU is busy.

C prototype: void fpu_wait (void);
Return: none

3.2.1.4 Implement fpu_readDelay routine

Instructions that read data from the uM-FPU require a minimum 90 microseconds delay after the opcode has been sent before data can be read. The readDelay routine simply calls the delay function for 90 microseconds.

C prototype: void fpu_readDelay (void);

Return: none

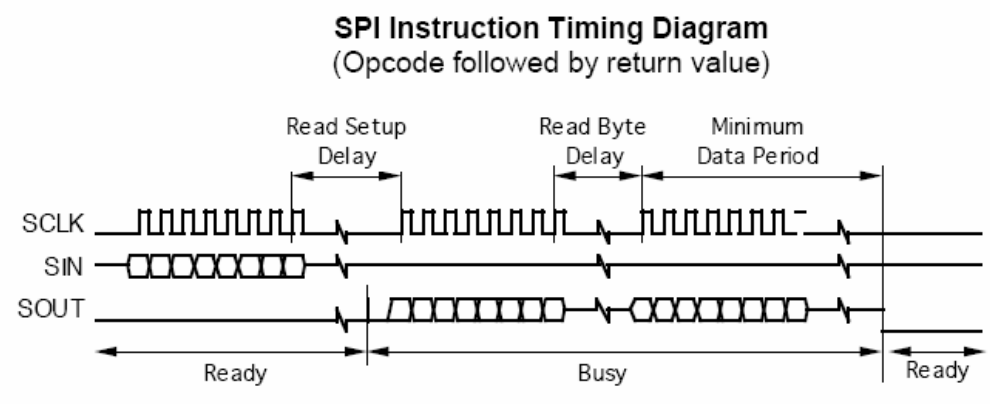


Figure 6 – Read Delay for any read Opcode

Item	Min	Max	Unit
SCLK Frequency		4	MHz
SCLK Output Low (per bit)	0.125		usec
SCLK Output High (per bit)	0.125	60	usec
Minimum Data Period – normal operation	10		usec
Minimum Data Period – debug trace enabled	15		usec
Read Setup Delay – normal operation	90		usec
Read Setup Delay – debug trace enabled	180		usec
Read Byte Delay – normal operation	10		usec
Read Byte Delay – debug trace enabled	15		usec

Table 5 – Timing Summary

3.2.1.5 Implement fpu_read routine

The fpu_read routine reads a byte from the uM-FPU through SPI interface. This implementation is similar to the fpu_send routine. Clear the SPI data register before

receiving data, and then wait for the transmission. Five microseconds delay is required to guarantee the minimum data period.

C prototype: unsigned char fpu_read (void);
Return: 8-bit byte

3.2.1.6 Add a synchronization check to reset routine

At this stage, fpu_send and fpu_read are successfully implemented. Add synchronization code to the end of the fpu_reset routine. First, send the SYNC opcode to the FPU. Next, call fpu_readDelay routine, and finally, call fpu_read routine to receive the returned value after synchronization. Successful reset and synchronization returns value 0x5C.

3.2.1.7 Create Opcode Definition

After setting up the routines mentioned above, it is convenience to create a file that contains all the uM-FPU Opcode definitions. Such definition file is obtained from the assembly reference driver. With some modification, the definition file is integrated in the driver umfpu-v2.h file.

3.2.2 Instruction Set

After the driver implementation, the uM-FPU is ready to perform calculations. Microcontroller sends instructions and data to the uM-FPU, which executes the instructions and produces results. The uM-FPU coprocessor has its own instruction set on the architecture. One can refer to the detail instruction reference from Micromega Corporation web site (see References at Section 7). A brief discussion of FPU instruction set will be shown. It provides opportunities for individuals that would like to define customized functions.

The uM-FPU contains 16 general purpose registers, register 0 to register 15, which can be used to store floating point or long integer values. All register are working registers, but register 0 is reserved for some instructions such as integer divide that

returns two values. The uM-FPU executes instructions in the same sequence of arriving to the buffer. Arithmetic operations are defined in terms of register A and register B. Register A and B can be any of the sixteen registers and should be selected before an operation. For example:

- The follow two instructions multiply the floating point value of register 1 and 2

Opcode	Instruction	Description
01	SELECTA+1	Select register 1 as A
82	FMUL+2	Select register 2 as B, calculate A = A x B

- The follow two instructions calculate integer division of register 1 and 2

Opcode	Instruction	Description
01	SELECTA+1	Select register 1 as A
D2	LDIV+2	Select register 2 as B, calculate A = A / B Register 0 = A % B

- The follow two instructions calculate Cosine of register 15

Opcode	Instruction	Description
0F	SELECTA+15	Select register 15 as A
E6	COS	Calculate A = cos (A)

- The follow four instructions calculate the register 2 (B) th root of register 3 (A)

Opcode	Instruction	Description
03	SELECTA+3	Select register 3 as A
12	SELECTB+2	Select register 2 as B
FE	XOP	Extended Opcode
E1	ROOT	A = the Bth root of A

The uM-FPU has a 32 instruction buffer. If more than 32 bytes of data is sent to specify a sequence of operations, fpu_wait routine must be called at least every 32 bytes to ensure that the instruction buffer does not overflow. Prior to reading the result, fpu_wait routine must also be called to ensure all instructions have been executed.

Some instructions are used to write values into registers. For simplicity, this project included high level routines which store 32-bit floating point values and 32-bit long integer values to the uM-FPU registers. Those routines will be discussed in the Section 4.2.4.

3.2.3 Instruction Reference

Similar to the software solution routines defined at file <Math.h>, high level user routines were developed. To use such hardware routines, the header file "fpu_support.h" which contains the high level routines definition and SPI interface driver must be included. The calls are defined as close as the software routines. The description of each routine is documented below. These routines are blocking so that sequential routine callings will not overflow the uM-FPU instruction buffer.

ABS **|A|**
Prototypes: float fpu_ABS (float A);
Description: Calculates the absolute value of the floating point value A

ACOS **acos (A)**
Prototypes: float fpu_ACOS (float A);
Description: Calculates the arc cosine of an angle in the range 0.0 through pi.

ASIN **asin (A)**
Prototypes: float fpu_ASIN (float A);
Description: Calculates the arc sine of an angle in the range of –pi/2 through pi/2.

ATAN **atan (A)**
Prototypes: float fpu_ATAN (float A);
Description: Calculates the arc tangent of an angle in the range of –pi/2 through pi/2.

ATAN2 **atan (A/B)**
Prototypes: float fpu_ATAN2 (float A, float B);
Description: Calculates the arc tangent of an angle in the range of –pi/2 through pi/2. This function is used to convert rectangular coordinates (A, B) to polar coordinates (r, theta).

CEIL **ceil (A)**
Prototypes: float fpu_CEIL (float A);
Description: Calculates the floating point value equal to the nearest integer that is greater than or equal to the floating point value A.

COS **cos (A)**
Prototypes: float fpu_COS (float A);
Description: Calculates the cosine of the angle (in radians) of A.

DEGREES **Convert radians to degrees**
Prototypes: float fpu_DEGREES (float A);
Description: The floating point value A is converted from radians to degrees.

EXP **exp (A)**
Prototypes: float fpu_EXP (float A);
Description: Calculates the value of e (2.7182818) raised to the power of the floating point value A.

EXP10 **exp10 (A)**
Prototypes: float fpu_EXP10 (float A);
Description: Calculates the value of 10 raised to the power of the floating point value A.

FADD **A + B**
Prototypes: float fpu_FADD (float A, float B);
Description: The floating point value B is added to the floating point value A.

FCOMPARE **Compare A and B**
Prototypes: char fpu_FCOMPARE (float A, float B);
Description: Compares the floating point values A and B. Return the status byte of comparison. The status byte is set as follows:
Bit 2 Not-a-Number, Set if either value is not a valid number
Bit 1 Sign, Set if A < B
Bit 0 Zero, Set if A = B
If neither Bit 0 or Bit 1 is set, A > B

FDIV **A / B**
Prototypes: float fpu_FDIV (float A, float B);
Description: The floating point value A is divided by the floating point value B.

FIX **fix (A)**
Prototypes: long fpu_FIX (float A);
Description: Converts the floating point value A to a long integer value.

FLOAT **float (A)**
Prototypes: float fpu_FLOAT (long A);
Description: Converts the long integer value in register A to a floating point value.

FLOOR **floor (A)**
Prototypes: float fpu_FLOOR (float A);
Description: Calculates the floating point value equal to the nearest integer that is less than or equal to the floating point value A.

FMUL **A * B**
Prototypes: float fpu_FMUL (float A, float B);
Description: The floating point value A is multiplied by the floating point value B.

FRACTION **fractional part of A**
Prototypes: float fpu_FRACTION (float A);
Description: Returns the fractional part the floating point value A.

FSTATUS **get the floating point status of A**
Prototypes: char fpu_FSTATUS (float A);
Description: Get the status of the floating point value A. The status byte is set as follows:

Bit 3 Infinity,	Set if the value is an infinity
Bit 2 Not-a-Number,	Set if the value is not a valid number
Bit 1 Sign,	Set if the value is negative
Bit 0 Zero,	Set if the value is zero

FSUB **A – B**
Prototypes: float fpu_FSUB (float A, float B);
Description: The floating point value B is subtracted from the floating point value A.

INVERSE **1 / A**
Prototypes: float fpu_INVERSE (float A);
Description: The inverse of the floating point value A.

LABS **|A|**
Prototypes: long fpu_LABS (long A);
Description: The absolute value of the long integer value A.

LADD **A + B**
Prototypes: long fpu_LADD (long A, long B);
Description: The long integer value B is added to the long integer value A.

LAND **A AND B**
Prototypes: long fpu_LAND (long A, long B);
Description: The bitwise AND of the long integer values A and B.

LCOMPARE Compare A and B
Prototypes: char fpu_LCOMPARE (long A, long B);
Description: Compares the signed long integer values A and B. The status byte is set as follows:

Bit 1 Sign,	Set if A < B
Bit 0 Zero,	Set if A = B

If neither Bit 0 or Bit 1 is set, A > B

LDEC **A - 1**
Prototypes: long fpu_LDEC (long A);
Description: The long integer value A is decremented by one.

LDIV **A / B**
Prototypes: long fpu_LDIV (long A, long B);
Description: The long integer value A is divided by the long integer value B.

LMOD **A % B**
Prototypes: long fpu_LMOD (long A, long B);
Description: The remainder of long integer division A by B.

LINC **A + 1**
Prototypes: long fpu_LINC (long A);
Description: The long integer value A is incremented by one.

LMUL **A * B**
Prototypes: long fpu_LMUL (long A, long B);
Description: The long integer value A is multiplied by the long integer value B.

LNEGATE **-A**
Prototypes: long fpu_LNEGATE (long A);
Description: The negative of the long integer value A.

LNOT **NOT A**
Prototypes: long fpu_LNOT (long A);
Description: The bitwise complement of the value A.

LOADE **Load floating point value of e**
Prototypes: float fpu_LOADE (void);
Description: Loads the floating point value of e (2.7182818)

LOADPI **Load floating point value of Pi**
Prototypes: float fpu_LOADPI (void);
Description: Loads the floating point value of pi (3.1415927).

LOG **log (A)**
Prototypes: float fpu_LOG (float A);
Description: Calculates the natural log of the floating point value A. The number e (2.7182818) is the base of the natural system of logarithms.

LOG10 **log10 (A)**
Prototypes: float fpu_LOG10 (float A);
Description: Calculates the base 10 logarithm of the floating point value A.

LOR **A OR B**
Prototypes: long fpu_LOR (long A, long B);
Description: The bitwise OR of the values A and B.

LSHIFT **A shifted by B bit positions**

Prototypes: long fpu_LSHIFT (long A, long B);

Description: Long integer value A is shifted by the number of bit positions specified by the long integer value B. The value A is shifted left if value B is positive and right if the value is negative.

LSTATUS **get the long integer status of A**

Prototypes: char fpu_LSTATUS (long A);

Description: Get the status of the long integer value A. The status byte is set as follows:

Bit 1 Sign, Set if the value is negative

Bit 0 Zero, Set if the value is zero

LSUB **A – B**

Prototypes: long fpu_LSUB (long A, long B);

Description: The long integer value B is subtracted from the long integer value A.

LTST **return the status of A AND B**

Prototypes: char fpu_LTST (long A, long B);

Description: Returns a status byte based on the result of a bitwise AND of the values A and B. The status byte is set as follows:

Bit 1 Sign, Set if the value is negative

Bit 0 Zero, Set if the value is zero

LUCOMPARE Compare A and B (unsigned)

Prototypes: char fpu_LUCOMPARE (unsigned long A, unsigned long B);

Description: Compares the unsigned long integer values A and B. The status byte is set as follows:

Bit 1 Sign, Set if A < B

Bit 0 Zero, Set if A = B

If neither Bit 0 or Bit 1 is set, A > B

LUDIV **A / B (unsigned)**

Prototypes: unsigned long fpu_LUDIV (unsigned long A, unsigned long B);

Description: The unsigned long integer value A is divided by the unsigned long integer value B.

LUMOD **A % B (unsigned)**

Prototypes: unsigned long fpu_LUMOD (unsigned long A, unsigned long B);

Description: The remainder of unsigned long integer division.

LXOR **A XOR B**

Prototypes: long fpu_LXOR (long A, long B);

Description: The bitwise XOR of the long integer values A and B.

MAX **maximum of A and B**
Prototypes: float fpu_MAX (float A, float B);
Description: The maximum of floating point values A and B.

MIN **minimum of A and B**
Prototypes: float fpu_MIN (float A, float B);
Description: The minimum of floating point values A and B.

NEGATE **-A**
Prototypes: float fpu_NEGATE (float A);
Description: The negative of the floating point value A.

POWER **A raised to the power of B**
Prototypes: float fpu_POWER (float A, float B);
Description: The floating point value A is raised to the power of the floating point value B.

RADIANS **Convert degrees to radians**
Prototypes: float fpu_RADIANS (float A);
Description: The floating point value A is converted from degrees to radians.

ROOT **the B th root of A**
Prototypes: float fpu_ROOT (float A, float B);
Description: Calculates the nth root of the floating point value A where the value n is equal to the floating point value B. It is equivalent to raising A to the power of (1/B).

ROUND **round (A)**
Prototypes: float fpu_ROUND (float A);
Description: The floating point value equal to the nearest integer to the floating point value A.

SIN **sin (A)**
Prototypes: float fpu_SIN (float A);
Description: Calculates the sine of the angle (in radians) A.

SQRT **sqrt (A)**
Prototypes: float fpu_SQRT (float A);
Description: Calculates the square root of the floating point value A.

TAN **tan (A)**
Prototypes: float fpu_TAN (float A);
Description: Calculates the tangent of the angle (in radians) A.

3.2.4 User Defined Function

Although high level user routines were written, one might want to generate customized functions such as combinations of few functions. Writing a user defined function allows better performance since immediate results are not returned. For large amount of calculations, customized functions would increase the performance significantly. These routines are non-blocking so that user must be aware of instruction buffer overflow (See the end of this section).

There are eight high level routines that support user defined functions, documented as follow:

Store Values in the uM-FPU register

Prototypes: void fpu_FRegisterA (float A, char reg);
Description: The floating point value A is sent to register reg as variable A.

Prototypes: void fpu_LRegisterA (long A, char reg);
Description: The long integer value A is sent to register reg as variable A.

Prototypes: void fpu_FRegisterB (float B, char reg);
Description: The floating point value B is sent to register reg as variable B.

Prototypes: void fpu_LRegisterB (long B, char reg);
Description: The long integer value B is sent to register reg as variable B.

Perform Calculation

Prototypes: void fpu_Function (char function);
Description: Perform calculations by specifying the function opcode.

Prototypes: void fpu_XOPFunction (char function);
Description: Perform calculations by specifying the 2 bytes function opcode.

Read values from the uM-FPU register

Prototypes: float fpu_FReadRegister (char reg);
Description: Read the floating point value from uM-FPU at register reg

Prototypes: long fpu_LReadRegister (char reg);
Description: Read the long integer value from uM-FPU at register reg

There are four steps to implement a user defined function successfully using the routine defined above. Here is an example of defining a customized function y = acos (x * y / z).

First, the values must be stored into the uM-FPU registers.

```
//Step one
float customized_function1 (float x, float y, float z)
begin
fpu_FRegisterA (x, 1);                  //Stores floating value x to register 1
fpu_FRegisterA (y, 2);                  //Stores floating value y to register 2
fpu_FRegisterA (z, 3);                  //Stores floating value z to register 3
end
```

Second, implement the calculations. Refer to the instruction set manual for detail function description and the special cases handling.

```
//Step two
float customized_function1 (float x, float y, float z)
begin
fpu_FRegisterA (x, 1);                  //Stores floating value x to register 1
fpu_FRegisterA (y, 2);                  //Stores floating value y to register 2
fpu_FRegisterA (z, 3);                  //Stores floating value z to register 3

fpu_send (SELECTA + 1);                 //Select register 1 as variable A
fpu_Function (FMUL + 2);                //reg 1 * reg 2 and stores in reg1
fpu_Function (FDIV + 3);                //reg1 * reg 3 and stores in reg 1
fpu_XOPFunction (ACOS);                 //acos (reg 1) and stores in reg 1
end
```

Third, return the results from uM-FPU.

```
//Step three
float customized_function1 (float x, float y, float z)
begin
fpu_FRegisterA (x, 1);                  //Stores floating value x to register 1
fpu_FRegisterA (y, 2);                  //Stores floating value y to register 2
fpu_FRegisterA (z, 3);                  //Stores floating value z to register 3

fpu_send (SELECTA + 1);                 //Select register 1 as variable A
fpu_Function (FMUL + 2);                //reg 1 * reg 2 and stores in reg1
fpu_Function (FDIV + 3);                //reg 1 * reg 3 and stores in reg 1
fpu_XOPFunction (ACOS);                 //acos (reg 1) and stores in reg 1

return fpu_FReadRegister(0x01);         //read the result from reg 1 and return
end
```

Lastly, make sure fpu_wait ( ) is called at least once for every 32 data bytes sent to avoid instruction buffer overflow. A user defined function implementation is now complete.

3.2.5 Performance Timing

Timer0 was used to trace the duration of instruction execution. The clock was prescaled to 64 and OCR0 was set to 250. The output compare match interrupt was cranked up so that timer0 produced timing with accuracy of 1ms. During this timing implementation, a bug appeared in the SPI driver. This bug was discovered and fixed. It was about the SPI interrupt enabled twice which caused a failure to the interface.

4. Testing and Results

The testing was done based on two categories: Accuracy and Performance. The accuracy test compared the results that were calculated using uM-FPU with CASIO commercial calculators and the software simulation. The accuracy test cases were written in the file “um-fpu.c”, and the results were displayed through hyperterm. The outputs of the results were attached in the Appendix 7.1. All high level user routines produced correct results.

Observation shows that the software simulation is mostly accurate, but verification shows that the software simulation is off by a small amount in some calculations, which means the number conversion during floating point simulations could induce a small error. In other words, the uM-FPU provides a more reliable and accuracy results.

The performance test compared the speeds and the throughputs. This test was executing the same instruction 10, 000 times. Timer0 was used to track the time with accuracy of 1 microsecond. The results are summarized in Table 6.

The uM-FPU has a relatively low speed of calculations. I thought the voltage was not supplying correctly but it was not the case. Although the uM-FPU executes instructions slower than the software simulation, the implementation is still useful because it is less processor intensive and it allows parallel architecture programming. For example, we can program such that after sending data byte to the uM-FPU, perform

another calculation or performing other tasks. The time used to idle for the uM-FPU calculation can be used wisely.

In addition, the performance of the calculation consists of speed and throughput. On one hand, the speed of the uM-FPU is not as good as expected and it seems have not performance gain. On the other hand, the uM-FPU decreases the work load on the microcontroller which can be used for multi-tasking. In general, the uM-FPU allows more throughputs in a given time. Therefore, the performance is increased by the amount of more work done.

The software simulation was running at Mega32 high performance microcontroller which ran at 16 MHz. The uM-FPU will demonstrate its power when it is integrated with some slower microcontrollers. Since the driver was developed fully, one can simply plug in the uM-FPU to the SPI port, include the driver, and use this floating point hardware.

Execute instruction 10, 000 times				
Instruction	Software Solution		Hardware Solution	
Types	Software Solution (ms)	Number of clock cycles	High level routines (ms)	High level routines without intermediate results (ms)
FDIV	218	348.8	3762	588
FMUL	459	734.4	5502	2330
SQRT	440	704	17494	15226
EXP	2369	3790.4	25633	23669
POWER	5311	8497.6	15217	12046
ASIN	3825	6120	34923	32651
SIN	2479	3966.4	24118	32152
FLOAT	188	300.8	3508	1328
FIX	188	300.8	3801	1750
ABS	206	329.6	2719	451
LMUL	190	304	5606	1904
LDIV	185	296	6529	2828
LINC	193	308.8	3150	576

Table 6 – Performance summary

The following is the finished picture of the project, which includes Mega32, STK500, and uM-FPU. The serial adapters were used at the early stage of the driver implementation. The project was finished at the beginning of July, 2006.

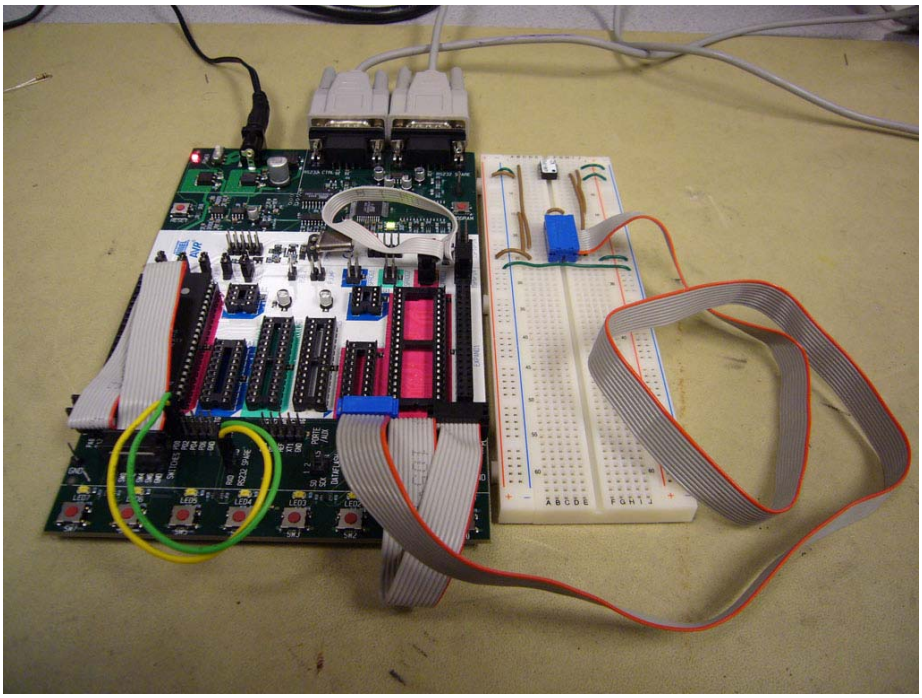


Figure 7 – Finished project

5. Conclusion

The hardware solution of the floating point support on microcontroller was successfully implemented as the original project goal. It allowed an alternative solution to the microcontroller floating point calculation. The driver development was the most challenging task in this project, I am very proud that the driver was developed successfully.

This project enhances the performance of microcontrollers which are able to run more complicate tasks. It is beneficial for the future design projects which demands strong floating point supports.

The idea behind this project is very interesting, which combines some complex concept together. The project requires knowledge of computer architecture, hardware programming, floating point calculations, and microcontroller designs. I had most of those course works but the design experiences. The experience that applies from what I have learnt to the real world design project such as reading the device specifications and debugging skills is the most valuable gain from this project.

6. References

Atmel 8-bit AVR Microcontroller datasheet. 2002.
<http://instruct1.cit.cornell.edu/courses/ee476/AtmelStuff/full32.pdf>.

Cornell University ECE 476 Designing with Microcontrollers. Spring 2006
< http://instruct1.cit.cornell.edu/courses/ee476/>.

Micromega Floating Point Unit Datasheet and Instruction Set. 2005.
< http://www.micromegacorp.com/downloads.html >.

EmbeddedC Programming and the Atmel AVR, Barnett, Delmar Learning, ISBN: 1-4018-1206-6.

7. Appendix

7.1 Hyper Terminal Output Running the Testing Code

```
Starting...

uM-FPU reset successfully.

|-2.56| = 2.56000
acos (0.5) = 1.04720
asin (0.5) = 0.52360
atan (0.5) = 0.46365
atan (0.5/0.2) = 1.19029
ceil (999.999) = 1000.00000
cos (5.5) = 0.70867
degrees (15.70796327 radians) = 899.99987
exp (5.0) = 148.41310
exp10 (5.5) = 316227.68750
9.9 + 8.8 = 18.70000
50.5 / 0.49 = 103.06121
-50.5 / 0.0 = ff800000
0.0 / 1.0 = 0
(long) 99.5 = 99
(float) 50L = 50.00000
floor (999.999) = 999.00000
5.5 x (-0.6) = -3.30000
fraction 8.87654321 = 0.87654
5.8 - 1.2 = 4.60000
1/20.5 = 0.048
labs (-987654) = 987654
-987654 + 123456789 = 122469135
0xFFFFFFFF AND 0x01010101 = 1010101
10000 - 1 = 9999
999 / 10 = 99
999 mod 10 = 9
10000 + 1 = 10001
99 x 999 = 98901
lnegate (1234567) = -1234567
LNOT (0x00FF00FF) = ff00ff00
e = 2.71828
PI = 3.14159
log (9998) = 9.21014
log10 (9997) = 3.99987
0x00000000 OR 0x01010101 = 1010101
0x12345678 << 4 = 23456780
0x12345678 >> 4 = 1234567
-987654 - (-123456789) = 122469135
999 / 10 = 99
999 mod 10 = 9
0x10101010 XOR 0x01010101 = 11111111
0x01010101 XOR 0x01010101 = 0
MAX (-1.234, 2.345) = 2.34500
MIN (-2.234, 3.345) = -2.23400
negate 9.876 = -9.87600
power (2.6, 8.8) = 4485.02539
radians (15.70796327 degrees) = 0.27416
```

```
root (4000, -2.5) = 0.03624
round (-15.0796327) = -15.00000
sin (1.5) = 0.99749
sqrt (2.5) = 1.58114
tan (4.5) = 4.63732
timeus = 192 ms , longtemp = 10012
timeus = 3158 ms , longtemp = 10012
timeus = 575 ms , longtemp = 10012
reg 1 = 1.20000
reg 2 = 100
reg 3 = 3.60000
reg 4 = 200
Sqrt ( Sin(0.5) ) * (Sqrt ( 200 )) = 9.79210, time = 57
Sqrt ( Sin(0.5) ) * (Sqrt ( 200 )) = 9.79209, time = 8
```

7.2 The uM-FPU testing code (umfpu-v2.c)

```
//*********************************************************
//Author: Kenny Chi Kin Lo
//Meng Project Spring 2006
//Implement floating point support to Mega32 microcontroller
//*********************************************************

#include <Mega32.h>
#include <stdio.h>
#define begin {
#define end }
#include <MATH.h>
#include "fpu_support.h"

unsigned long timeus;   //time in micro seconds
int i;

//variables
float floattemp;
long longtemp;

//*********************************************************
//timer 0 compare ISR
interrupt [TIM0_COMP] void timer0_compare(void)
begin
  //Decrement the three times if they are not already zero
 timeus++;

end


//*********************************************************
void main(void)
begin

//serial setop for debugging using printf, etc.
UCSRB = 0x18 ;
UBRRL = 103 ;
//set up timer 0
  TIMSK=2;        //turn on timer 0 cmp match ISR
  OCR0 = 250;     //set the compare re to 250 time ticks
                  //prescalar to 64 and turn on clear-on-match
  TCCR0=0b00001011;

putsf("\r\nStarting...\r\n");

if (fpu_reset() != SYNC_CHAR)       //reset failed
begin
      putsf("uM-FPU not detected.");
      return;
end
else                                //reset successfully
putsf("uM-FPU reset successfully.\r\n");
fpu_send(XOP);
fpu_send(TRACEOFF);
#asm ("sei");
```

```
//Testing
//Accuracy Test
floattemp = fpu_ABS((float) -2.56);
printf("|-2.56| = %f\r\n", floattemp);

floattemp = fpu_ACOS((float) 0.5);
printf("acos (0.5) = %f\r\n", floattemp);

floattemp = fpu_ASIN((float) 0.5);
printf("asin (0.5) = %f\r\n", floattemp);

floattemp = fpu_ATAN((float) 0.5);
printf("atan (0.5) = %f\r\n", floattemp);

floattemp = fpu_ATAN2((float) 0.5, 0.2);
printf("atan (0.5/0.2) = %f\r\n", floattemp);

floattemp = fpu_CEIL((float) 999.999);
printf("ceil (999.999) = %f\r\n", floattemp);

floattemp = fpu_COS((float) 5.5);
printf("cos (5.5) = %f\r\n", floattemp);

floattemp = fpu_DEGREES((float) 15.70796327);
printf("degrees (15.70796327 radians) = %f\r\n", floattemp);

floattemp = fpu_EXP((float) 5.0);
printf("exp (5.0) = %f\r\n", floattemp);

floattemp = fpu_EXP10((float) 5.5);
printf("exp10 (5.5) = %f\r\n", floattemp);

floattemp = fpu_FADD((float) 9.9, (float) 8.8);
printf("9.9 + 8.8 = %f\r\n", floattemp);

fpu_FCOMPARE((float) -0.5, (float) 0.0);
fpu_FCOMPARE((float) 0.0, (float) -0.5);
fpu_FCOMPARE((float) 0.0, (float) -0.0);

floattemp = fpu_FDIV((float) 50.5, (float) 0.49);
printf("50.5 / 0.49 = %f\r\n", floattemp);

floattemp = fpu_FDIV((float) -50.5, (float) 0.0);
//return  -inf (0xFF800000)
//          +inf (0x7F800000)
printf("-50.5 / 0.0 = %lx\r\n", floattemp);

floattemp = fpu_FDIV((float) 0.0, (float) 1.0);
//return  +0.0 (0x00000000)
//          -0.0 (0x80000000)
printf("0.0 / 1.0 = %lx\r\n", floattemp);

longtemp  = fpu_FIX((float) 99.5);
printf("(long) 99.5 = %ld\r\n", longtemp);
```

```
floattemp = fpu_FLOAT((long) 50);
printf("(float) 50L = %f\r\n", floattemp);

floattemp = fpu_FLOOR((float) 999.999);
printf("floor (999.999) = %f\r\n", floattemp);

floattemp = fpu_FMUL((float) 5.5, (float) -0.6);
printf("5.5 x (-0.6) = %f\r\n", floattemp);

floattemp = fpu_FRACTION((float) 8.87654321);
printf("fraction 8.87654321 = %f\r\n", floattemp);

fpu_FSTATUS((float) -0.5);
fpu_FSTATUS(fpu_FDIV((float) 5.0, (float) 0.0));

floattemp = fpu_FSUB((float) 5.8, (float) 1.2);
printf("5.8 - 1.2 = %f\r\n", floattemp);

floattemp = fpu_INVERSE((float) 20.5);
printf("1/20.5 = %f\r\n", floattemp);

longtemp  = fpu_LABS((long) -987654);
printf("labs (-987654) = %ld\r\n", longtemp);

longtemp  = fpu_LADD((long) -987654, (long) 123456789);
printf("-987654 + 123456789 = %ld\r\n", longtemp);

longtemp  = fpu_LAND((long) 0xFFFFFFFF, (long) 0x01010101);
printf("0xFFFFFFFF AND 0x01010101 = %lx\r\n", longtemp);

fpu_LCOMPARE((long) 5000, (long) -1);
fpu_LCOMPARE((long) 5000, (long) 50000);
fpu_LCOMPARE((long) 5000, (long) 5000);

longtemp  = fpu_LDEC((long) 10000);
printf("10000 - 1 = %ld\r\n", longtemp);

longtemp  = fpu_LDIV((long) 999, (long) 10);
printf("999 / 10 = %ld\r\n", longtemp);

longtemp  = fpu_LMOD((long) 999, (long) 10);
printf("999 mod 10 = %ld\r\n", longtemp);

longtemp  = fpu_LINC((long) 10000);
printf("10000 + 1 = %ld\r\n", longtemp);

longtemp  = fpu_LMUL((long) 99, (long) 999);
printf("99 x 999 = %ld\r\n", longtemp);

longtemp  = fpu_LNEGATE((long) 1234567);
printf("lnegate (1234567) = %ld\r\n", longtemp);

longtemp  = fpu_LNOT((long) 0x00FF00FF);
printf("LNOT (0x00FF00FF) = %lx\r\n", longtemp);

floattemp = fpu_LOADE();
printf("e = %f\r\n", floattemp);
```

```
floattemp = fpu_LOADPI();
printf("PI = %f\r\n", floattemp);

floattemp = fpu_LOG((float) 9998);
printf("log (9998) = %f\r\n", floattemp);

floattemp = fpu_LOG10((float) 9997);
printf("log10 (9997) = %f\r\n", floattemp);

longtemp  = fpu_LOR((long) 0x00000000, (long) 0x01010101);
printf("0x00000000 OR 0x01010101 = %lx\r\n", longtemp);

longtemp  = fpu_LSHIFT((long) 0x12345678, (long) 4);
printf("0x12345678 << 4 = %lx\r\n", longtemp);

longtemp  = fpu_LSHIFT((long) 0x12345678, (long) -4);
printf("0x12345678 >> 4 = %lx\r\n", longtemp);

fpu_LSTATUS((long) -1000);

longtemp  = fpu_LSUB((long) -987654, (long) -123456789);
printf("-987654 - (-123456789) = %ld\r\n", longtemp);

fpu_LTST((long) -1000, (long) -1000);
fpu_LTST((long) -1000, (long) 0);
fpu_LUCOMPARE((unsigned long) 5000,  (unsigned long) -1);
fpu_LUCOMPARE((unsigned long) 50000, (unsigned long) 5000);
fpu_LUCOMPARE((unsigned long) 5000,  (unsigned long) 5000);

longtemp  = (long) fpu_LUDIV((unsigned long) 999, (unsigned long) 10);
printf("999 / 10 = %ld\r\n", longtemp);

longtemp  = (long) fpu_LUMOD((unsigned long) 999, (unsigned long) 10);
printf("999 mod 10 = %ld\r\n", longtemp);

longtemp  = fpu_LXOR((long) 0x10101010, (long) 0x01010101);
printf("0x10101010 XOR 0x01010101 = %lx\r\n", longtemp);

longtemp  = fpu_LXOR((long) 0x01010101, (long) 0x01010101);
printf("0x01010101 XOR 0x01010101 = %lx\r\n", longtemp);

floattemp = fpu_MAX((float) -1.234, (float) 2.345);
printf("MAX (-1.234, 2.345) = %f\r\n", floattemp);

floattemp = fpu_MIN((float) -2.234, (float) 3.345);
printf("MIN (-2.234, 3.345) = %f\r\n", floattemp);

floattemp = fpu_NEGATE((float) 9.876);
printf("negate 9.876 = %f\r\n", floattemp);

floattemp = fpu_POWER((float) 2.6, (float) 8.8);
printf("power (2.6, 8.8) = %f\r\n", floattemp);

floattemp = fpu_RADIANS((float) 15.70796327);
printf("radians (15.70796327 degrees) = %f\r\n", floattemp);
```

```
floattemp = fpu_ROOT((float) 4000, (float) -2.5);
printf("root (4000, -2.5) = %f\r\n", floattemp);

floattemp = fpu_ROUND((float) -15.0796327);
printf("round (-15.0796327) = %f\r\n", floattemp);

floattemp = fpu_SIN((float) 1.5);
printf("sin (1.5) = %f\r\n", floattemp);

floattemp = fpu_SQRT((float) 2.5);
printf("sqrt (2.5) = %f\r\n", floattemp);

floattemp = fpu_TAN((float) 4.5);
printf("tan (4.5) = %f\r\n", floattemp);


//Performance Test

  timeus = 0;
  longtemp = 12L;
  for(i=0; i < 10000;i++)
  begin
  longtemp =  longtemp + 1;
  if(i % 16) fpu_wait();
  end
  printf("timeus = %ld ms , longtemp = %ld\r\n", timeus, longtemp);

  timeus = 0;
  longtemp = 12L;
  for(i=0; i < 10000;i++)
  begin
  longtemp = fpu_LINC(longtemp);
  end
  printf("timeus = %ld ms , longtemp = %ld\r\n", timeus, longtemp);

  timeus = 0;
      longtemp = 12L;
      y = &longtemp;

      fpu_send(XOP);
      fpu_send(LWRITEA+3);          //Write the floating value A to
register 0
      fpu_send(* (y+3));
      fpu_send(* (y+2));
      fpu_send(* (y+1));
      fpu_send(* (y));

      longtemp = 1L;
      y = &longtemp;

      fpu_send(XOP);
      fpu_send(LWRITEB+1);          //Write the floating value A to
register 0
      fpu_send(* (y+3));
      fpu_send(* (y+2));
      fpu_send(* (y+1));
      fpu_send(* (y));
```

```
      for(i=0; i < 10000;i++)
      begin
      fpu_send(XOP);
      fpu_send(LINCA);        //Perform Calculation
      if(i % 16) fpu_wait();  //every time i = 16, wait instruction
buffer empty
     end

      fpu_wait();             //Return the result from register 0
      fpu_send(XOP);
      fpu_send(LREAD+3);
      fpu_readDelay();
      y = &longtemp;
      *(y+3) = fpu_read();
      *(y+2) = fpu_read();
      *(y+1) = fpu_read();
      *(y)   = fpu_read();
  printf("timeus = %ld ms , longtemp = %ld\r\n", timeus, longtemp);


//User defined functions
//void  fpu_FRegisterA(float A, char reg);
//void  fpu_LRegisterA(long A, char reg);
//void  fpu_FRegisterB(float A, char reg);
//void  fpu_LRegisterB(long A, char reg);
//float fpu_FReadRegister(char reg);
//long  fpu_LReadRegister(char reg);
//void  fpu_Function(char function)
//void  fpu_XOPFunction(char function)

floattemp = 1.2;
longtemp = 14;
fpu_FRegisterA(1.2, 1);
fpu_LRegisterA(100, 2);
fpu_FRegisterB(3.6, 3);
fpu_LRegisterB(200, 4);
printf("reg 1 = %f\r\n" , fpu_FReadRegister(1));
printf("reg 2 = %ld\r\n", fpu_LReadRegister(2));
printf("reg 3 = %f\r\n" , fpu_FReadRegister(3));
printf("reg 4 = %ld\r\n", fpu_LReadRegister(4));

//Example
//Defined A = Sqrt ( Sin(0.5) ) * (Sqrt ( 200 ))
timeus= 0;
for (i = 0; i < 10; i++)
begin
fpu_FRegisterA(0.5, 1);       //put 0.5 into register 1
fpu_FRegisterB(200.0, 2);     //put 200.0 into register 2
fpu_Function(SIN);            //SINE (A in reg 1)
fpu_Function(SQRT);           //SQRT (SINE (0.5))
fpu_Function(SELECTA+2);      //Select reg 2 as A
fpu_Function(SQRT);           //SQRT (A in reg 2)
fpu_Function(FMUL+1);         //FMUL A = A * B, (reg 2 = reg 2 * reg 1)
fpu_wait();                   //Make sure calling fpu_wait() at least
                              //every 32 bytes to ensure that the
                              //instruction buffer does not overflow
```

```
end
floattemp = fpu_FReadRegister(2);   //Read reg 2
printf("Sqrt ( Sin(0.5) ) * (Sqrt ( 200 )) = %f, time = %ld\r\n",
floattemp, timeus);

timeus = 0;
for (i=0; i<10;i++) floattemp = sqrt(sin(0.5)) * (sqrt (200));
printf("Sqrt ( Sin(0.5) ) * (Sqrt ( 200 )) = %f, time = %ld\r\n",
floattemp, timeus);

end
```

7.3 The uM-FPU SPI driver (umfpu-v2.h)

```
//*******************************************************
//Author: Kenny Chi Kin Lo
//Meng Project Spring 2006
//Implement floating point support to Mega32 microcontroller
//
//
// File: umfpu-v2.h
// This file provides opcode definitions and SPI communication setup
for the
// uM-FPU V2 floating point coprocessor.
//
// Reference from Micromega Corporation "umfpuv1.h"
//*******************************************************

#include <delay.h>
#include <SPI.h>
//#define begin {
//#define end }

//-------------------- uM-FPU opcodes --------------------------------
#define     SELECTA     0x00  // select A register
#define     SELECTB     0x10  // select B register
#define     FWRITEA     0x20  // select A and write float to register
#define     FWRITEB     0x30  // Select B and write float to register
#define     FREAD       0x40  // read float from register
#define     FSET        0x50  // A = REG
#define     LSET        0x50  // A = REG
#define     FADD        0x60  // A = A + REG (float)
#define     FSUB        0x70  // A = A - REG (float)
#define     FMUL        0x80  // A = A * REG (float)
#define     FDIV        0x90  // A = A / REG (float)
#define     LADD        0xA0  // A = A + REG (long)
#define     LSUB        0xB0  // A = A - REG (long)
#define     LMUL        0xC0  // A = A * REG (long)
#define     LDIV        0xD0  // A = A / REG (long)
#define     SQRT        0xE0  // A = sqrt(A)
#define     LOG         0xE1  // A = ln(A)
#define     LOG10       0xE2  // A = log(A)
#define     EXP         0xE3  // A = e ** A
#define     EXP10       0xE4  // A = 10 ** A
#define     SIN         0xE5  // A = sin(A) radians
#define     COS         0xE6  // A = cos(A) radians
#define     TAN         0xE7  // A = tan(A) radians
#define     FLOOR       0xE8  // A = nearest integer <= A
#define     CEIL        0xE9  // A = nearest integer >= A
#define     ROUND       0xEA  // A = nearest integer to A
#define     NEGATE      0xEB  // A = -A
#define     ABS         0xEC  // A = |A|
#define     INVERSE     0xED  // A = 1 / A
#define     DEGREES     0xEE  // A = A / (PI / 180) radians to degrees
#define     RADIANS     0xEF  // A = A * (PI / 180) degreees to radians
#define     SYNC        0xF0  // synchronization
#define     FLOAT       0xF1  // copy A to register 0 and float
#define     FIX         0xF2  // copy A to register 0 and fix
#define     FCOMPARE    0xF3  // compare A and B
```

```
#define     LOADBYTE    0xF4  //convert to float
#define     LOADUBYTE   0xF5  //convert to float
#define     LOADWORD    0xF6  //convert to float
#define     LOADUWORD   0xF7  //convert to float
#define     READSTR     0xF8  // read zero terminated string
#define     ATOF        0xF9  // convert ASCII to float, store in A
#define     FTOA        0xFA  // convert float to ASCII
#define     ATOL        0xFB  // convert ASCII to long, store in A
#define     LTOA        0xFC  // convert long to ASCII
#define     FSTATUS     0xFD  // get the status of A register
#define     XOP         0xFE  // extended opcode
#define     FNOP        0xFF  // nop
#define     FUNCTION    0x00  // (XOP) user functions 0-15
#define     READBYTE    0x90  // (XOP) read low 8 bits of A (long)
#define     READWORD    0x91  // (XOP) read low 16 bits of A (long)
#define     READLONG    0x92  // (XOP) read 32-bit long value from A
#define     READFLOAT   0x93  // (XOP) read floating point value from A
#define     LINCA       0x94  // (XOP) A = A + 1 (long)
#define     LINCB       0x95  // (XOP) A = A + 1 (long)
#define     LDECA       0x96  // (XOP) A = A - 1 (long)
#define     LDECB       0x97  // (XOP) A = A - 1 (long)
#define     LAND        0x98  // (XOP) A = A AND B (long)
#define     LOR         0x99  // (XOP) A = A OR B (long)
#define     LXOR        0x9A  // (XOP) A = A XOR B (long)
#define     LNOT        0x9B  // (XOP) A = NOT A (long)
#define     LTST        0x9C  // (XOP) status of A AND B (long)
#define     LSHIFT      0x9D  // (XOP) shift A by B bits (long)
#define     LWRITEA     0xA0  // (XOP) select A and write long to reg
#define     LWRITEB     0xB0  // (XOP) select B and write long to reg
#define     LREAD       0xC0  // (XOP) read 32-bit long from reg
#define     LUDIV       0xD0  // (XOP) A = A / REG (unsigned long)
#define     POWER       0xE0  // (XOP) A = A ** B
#define     ROOT        0xE1  // (XOP) A = the Bth root of A
#define     MIN         0xE2  // (XOP) A = minimum of A and B
#define     MAX         0xE3  // (XOP) A = maximum of A and B
#define     FRACTION    0xE4  // (XOP) load the fractional part of A
#define     ASIN        0xE5  // (XOP) A = asin(A) radians
#define     ACOS        0xE6  // (XOP) A = acos(A) radians
#define     ATAN        0xE7  // (XOP) A = atan(A) radians
#define     ATAN2       0xE8  // (XOP) A = atan(A/B)
#define     LCOMPARE    0xE9  // (XOP) long compare A and B
#define     LUCOMPARE   0xEA  // (XOP) unsigned long compare A and B
#define     LSTATUS     0xEB  // (XOP) long status
#define     LNEGATE     0xEC  // (XOP) A = -A (long)
#define     LABS        0xED  // (XOP) A = |A| (long)
#define     LEFT        0xEE  // (XOP) right parenthesis
#define     RIGHT       0xEF  // (XOP) left parenthesis
#define     LOADZERO    0xF0  // (XOP) load register 0 with zero
#define     LOADONE     0xF1  // (XOP) load register 0 with 1.0
#define     LOADE       0xF2  // (XOP) load register 0 with e
#define     LOADPI      0xF3  // (XOP) load register 0 with pi
#define     LONGBYTE    0xF4  // (XOP) signed byte convert to long
#define     LONGUBYTE   0xF5  // (XOP) unsigned byte convert to long
#define     LONGWORD    0xF6  // (XOP) signed word convert to long
#define     LONGUWORD   0xF7  // (XOP) unsigned word convert to long
#define     IEEEMODE    0xF8  // (XOP) set IEEE mode (default)
#define     PICMODE     0xF9  // (XOP) set PIC mode
```

```
#define     CHECKSUM    0xFA  // (XOP) calculate code checksum
#define     BREAK       0xFB  // (XOP) debug breakpoint
#define     TRACEOFF    0xFC  // (XOP) turn debug trace off
#define     TRACEON     0xFD  // (XOP) turn debug trace on
#define     TRACESTR    0xFE  // (XOP) send debug trace to buffer
#define     VERSION     0xFF  // (XOP) get version string
#define     SYNC_CHAR   0x5C  // synchronization character


//Use predefined SPI I/O pins
#define FPU_CLOCK       PORTB.7     //SCK - connects to SCLK
#define FPU_DATAIN      PORTB.6     //input to FPU (MISO) - to SIN
#define FPU_DATAOUT     PORTB.5     //output from FPU (MOSI) - to SOUT

//-------------------- local prototypes ------------------------------
--------
unsigned char fpu_reset(void);
void fpu_send(unsigned char dataByte);
unsigned char fpu_read(void);
void fpu_wait(void);
void fpu_readDelay(void);


//Functions for SPI communication
unsigned char fpu_reset(void)
begin
//Set PORTB Tri state buffers (MISO is input all others output)
      DDRB = 0xBF;
//Set SPI control register to MSSP, enable it, and set speed to 4MHz
      SPCR = 0x50;

        // SPI Control - 01010000
        // SPIE - Interrupt Enable (0)
        // SPI - Enable (1)
        // DORD - Data Order (0) - MSB transmitted first
        // MSTR - Master/Slave (1) - Master
        // CPOL - Clock Polarity (0) - SCK low when idle
        // CPHA - Clock Phase (0) - Sample on leading edge
        // SPI Mode 0
        // SPR1 SPR0 - Clock rate - f_osc/4

FPU_CLOCK = 0;          //set clock and data low
FPU_DATAOUT = 0;
FPU_CLOCK = 1;    //pulse clock high for 500 us
delay_us(500);
FPU_CLOCK = 0;
delay_us(8000);   //delay for 8 ms

fpu_send(0xF0);   //Start synchronization code
fpu_readDelay();
return fpu_read();
end


void fpu_send(unsigned char dataByte)
begin
```

```
SPDR = dataByte;
while(!(SPSR & (0x80)))
;

delay_us(5);                  //recommended 15 us delay using debug
mode
end


void fpu_wait(void)
begin
#asm
      fpu_wait:
      sbic  0x16, 6     ; PINB = 0x16, FPU_DATAIN = 6, wait until uM-
FPU is ready
      rjmp  fpu_wait
#endasm
return;
end


void fpu_readDelay(void)
begin
delay_us(90);                 //recommended 180 us delay using debug
mode
return;
end


unsigned char fpu_read(void)
begin

SPDR = 0x00;
/* Wait for reception complete */
while(!(SPSR & (0x80)))
;
/* Return data register */
delay_us(5);                  //recommended 15 us delay using debug
mode
return SPDR;

end
```

7.4 High Level uM-FPU Supporting Routine (fpu_support.h)

```
//*********************************************************
//Author: Kenny Chi Kin Lo
//Meng Project Spring 2006
//Implement floating point support to Mega32 microcontroller
//
//
// File: fpu_support.h
// This file implemented floating point support on Mega32
// using Micromega FPU-V2
//
//*********************************************************

//#define begin {
//#define end }
#include "umfpu-V2.h"   //SPI Communication and Opcode definitions

//Defined Variables
char *y;
float floatreturn;
long  longreturn;


//Floating Point support Prototypes
//floating Point Support
float fpu_ABS(float A);
float fpu_ACOS(float A);
float fpu_ASIN(float A);
float fpu_ATAN(float A);
float fpu_ATAN2(float A, float B);
float fpu_CEIL(float A);
float fpu_COS(float A);
float fpu_DEGREES(float A);
float fpu_EXP(float A);
float fpu_EXP10(float A);
float fpu_FADD(float A, float B);
char  fpu_FCOMPARE(float A, float B);
float fpu_FDIV(float A, float B);
long  fpu_FIX(float A);
float fpu_FLOAT(long A);
float fpu_FLOOR(float A);
float fpu_FMUL(float A, float B);
float fpu_FRACTION(float A);
char  fpu_FSTATUS(float A);
float fpu_FSUB(float A, float B);
float fpu_INVERSE(float A);
float fpu_LOADE(void);
float fpu_LOADPI(void);
float fpu_LOG(float A);
float fpu_LOG10(float A);
float fpu_MAX(float A, float B);
float fpu_MIN(float A, float B);
float fpu_NEGATE(float A);
float fpu_POWER(float A, float B);
float fpu_RADIANS(float A);
float fpu_ROOT(float A, float B);
```

```
float fpu_ROUND(float A);
float fpu_SIN(float A);
float fpu_SQRT(float A);
float fpu_TAN(float A);

//Long Integer Support
long  fpu_LABS(long A);
long  fpu_LADD(long A, long B);
long  fpu_LAND(long A, long B);
char  fpu_LCOMPARE(long A, long B);
long  fpu_LDEC(long A);
long  fpu_LDIV(long A, long B);
long  fpu_LMOD(long A, long B);
long  fpu_LINC(long A);
long  fpu_LMUL(long A, long B);
long  fpu_LNEGATE(long A);
long  fpu_LNOT(long A);
long  fpu_LOR(long A, long B);
long  fpu_LSHIFT(long A, long B);
char  fpu_LSTATUS(long A);
long  fpu_LSUB(long A, long B);
char  fpu_LTST(long A, long B);
char  fpu_LUCOMPARE(unsigned long A, unsigned long B);
unsigned long  fpu_LUDIV(unsigned long A, unsigned long B);
unsigned long  fpu_LUMOD(unsigned long A, unsigned long B);
long  fpu_LXOR(long A, long B);

//User defined functions prototypes
void  fpu_FRegisterA(float A, char reg);
void  fpu_LRegisterA(long A, char reg);
void  fpu_FRegisterB(float B, char reg);
void  fpu_LRegisterB(long B, char reg);
float fpu_FReadRegister(char reg);
long  fpu_LReadRegister(char reg);
void  fpu_Function(char function);
void  fpu_XOPFunction(char function);

//************************************************************
//Implement Supporting Functions

//ABS A =|A|
//Calculates the absolute value
float fpu_ABS(float A)
begin
y = &A;

fpu_send(FWRITEA);            //Write the floating value A to register
0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(ABS);                //Perform calculation

fpu_wait();                   //Return result from register 0
fpu_send(FREAD);
```

```
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//ACOS A = acos (A)
//Calculates the arc cosine of an angle in the range 0.0 throught pi
//|A| must be less than equal to 1
float fpu_ACOS(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(ACOS);         //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//ASIN A = asin (A)
//Calculates the arc sine of an angle in the range -pi/2 throught pi/2
//|A| must be less than equal to 1
float fpu_ASIN(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(ASIN);         //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
```

```
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//ATAN A = atan (A)
//Calculates the arc tangent of an angle in the range -pi/2 throught
pi/2
float fpu_ATAN(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(ATAN);         //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//ATAN2 A = atan (A/B)
//Calculates the arc tangent of an angle in the range -pi/2 throught
pi/2
float fpu_ATAN2(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));
```

```
fpu_send(XOP);
fpu_send(ATAN2);        //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//CEIL A = ceil (A)
//Calculates the floating point value equal to the nearlest integer
that is
//greater than or equal to the floating point value.
float fpu_CEIL(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(CEIL);         //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//COS A = cos (A)
//Calculates the cosine of an angle in radians
float fpu_COS(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(COS);          //Perform Calucalation
```

```
fpu_wait();              //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//DEGREES
//Convert radians to degrees
float fpu_DEGREES(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(DEGREES);      //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//EXP A = exp (A)
//Calculates the value of e rasied to the power of the floating point
value
float fpu_EXP(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(EXP);          //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
```

```
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//EXP10 A = exp10 (A)
//Calculates the value of 10 rasied to the power of the floating point
value
float fpu_EXP10(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(EXP10);        //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//FADD A = A + B
//The floating point value B is added to A and the result is stored at
A
float fpu_FADD(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(FADD + 1);     //Perform Calculation
```

```
fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//FCOMPARE
//Compare floating point value A and B. Return status byte.
//Set Bit 2 if Not-a-Number
//Set Bit 1 if A < B
//Set Bit 0 if A = B, If Neither Bit 0 or Bit 1 is set, A > B
char fpu_FCOMPARE(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_wait();             //Return the status byte immediately
fpu_send(FCOMPARE);
fpu_readDelay();

return fpu_read();
end

//FDIV A = A / B
//The floating point value B is added to A and the result is stored at
A
float fpu_FDIV(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
```

```
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(FDIV + 1);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//FIX register 0 = fix (A)
//Converts the floating point value A to a long integer value
//and stores the result in register 0
long fpu_FIX(float A)
begin
y = &A;

fpu_send(FWRITEA + 1);  //Write the floating value A to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(FIX);          //Perform Calucalation

fpu_wait();
fpu_send(XOP);          //Return the result from register 0
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//FLOAT register 0 = float (A)
//Converts the long integer value in register A to a floating point
value
//and stores the result in register 0
float fpu_FLOAT(long A)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA + 1);  //Write the long integer value A to register 1
fpu_send(* (y+3));
```

```
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(FLOAT);        //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//FLOOR A = floor (A)
//Calculates the floating point value equal to the nearlest integer
that is
//less than or equal to the floating point value.
float fpu_FLOOR(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(FLOOR);        //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//FMUL A = A * B
//The floating point value B is added to A and the result is stored at
A
float fpu_FMUL(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
```

```
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(FMUL + 1);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//FRACTION
//Load register 0 with the fraction part of the floating point value
float fpu_FRACTION(float A)
begin
y = &A;

fpu_send(FWRITEA + 1);  //Write the floating value A to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(FRACTION);     //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//FSTATUS
//Return status of the floating point value
//Set Bit 3 if infinity
//Set Bit 2 if Not a number NaN
//Set Bit 1 if negative
//Set Bit 0 if zero
char fpu_FSTATUS(float A)
```

```
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_wait();             //Return the status byte immediately
fpu_send(FSTATUS);
fpu_readDelay();

return fpu_read();
end

//FSUB A = A - B
//The floating point value B is subtracted from A and the result is
stored at A
float fpu_FSUB(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(FSUB + 1);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//INVERSE A = 1/A
//The inverse of the floating point value A
float fpu_INVERSE(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
```

```
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(INVERSE);      //Perform function

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//LABS
//Absolute value of the long integer value A
long fpu_LABS(long A)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LABS);         //Perform Calucalation

fpu_wait();
fpu_send(XOP);          //Return the result from register 0
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LADD A = A + B
//The long integer value B is added to A and the result is stored at A
long fpu_LADD(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
```

```
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(LADD + 1);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LAND
//Bitwise AND
long fpu_LAND(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LAND);         //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
```

```
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LCOMPARE
//Compare long values A and B. Return status byte.
//Set Bit 1 if A < B
//Set Bit 0 if A = B, If Neither Bit 0 or Bit 1 is set, A > B
char fpu_LCOMPARE(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the long value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_wait();             //Return the status byte immediately
fpu_send(XOP);
fpu_send(LCOMPARE);
fpu_readDelay();

return fpu_read();
end

//LDEC
//decrement long integer by 1
long fpu_LDEC(long A)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LDECA);        //Perform Calucalation

fpu_wait();
fpu_send(XOP);          //Return the result from register 0
fpu_send(LREAD);
```

```
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LDIV
// long integer A is divided by the long integer B
long fpu_LDIV(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA + 1);  //Write the long value A to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 2);  //Write the floating value B to register 2
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(LDIV + 2);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD + 1);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LMOD
// long integer A is divided by the long integer B
long fpu_LMOD(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA + 1);  //Write the long value A to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
```

```
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 2);  //Write the floating value B to register 2
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(LDIV + 2);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LINC
//increment long integer by 1
long fpu_LINC(long A)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LINCA);        //Perform Calucalation

fpu_wait();
fpu_send(XOP);          //Return the result from register 0
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LMUL
// long integer A is multiplied by the long integer B
long fpu_LMUL(long A, long B)
```

```
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(LMUL + 1);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LNEGATE
//Negate a long integer
long fpu_LNEGATE(long A)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LNEGATE);      //Perform Calucalation

fpu_wait();
fpu_send(XOP);          //Return the result from register 0
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
```

```
*(y)   = fpu_read();

return longreturn;
end

//LNOT
//Bitwise complement of a long value
long fpu_LNOT(long A)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LNOT);         //Perform Calucalation

fpu_wait();
fpu_send(XOP);          //Return the result from register 0
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LOADE
//LOAD register 0 with value of e (2.7182818)
float fpu_LOADE(void)
begin

fpu_send(XOP);
fpu_send(LOADE);        //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//LOADPI
//LOAD register 0 with value of PI (3.1415927)
float fpu_LOADPI(void)
```

```
begin

fpu_send(XOP);
fpu_send(LOADPI);       //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//LOG A = log (A)
//Natural log of the floating point value A in the base of e
float fpu_LOG(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(LOG);          //Perform function

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//LOG10 A = log10 (A)
//Natural log of the floating point value A in the base of 10
float fpu_LOG10(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(LOG10);        //Perform function
```

```
fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//LOR
//Bitwise OR
long fpu_LOR(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LOR);          //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LSHIFT
//Shift the value at register A left or right by the value at register
B
//if B = 0, no shift
//shift left if B is positive, right if B is negative
long fpu_LSHIFT(long A, long B)
begin
y = &A;
```

```
fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LSHIFT);       //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LSTATUS
//Return status of the long integer value
//Set Bit 1 if negative
//Set Bit 0 if zero
char fpu_LSTATUS(long A)
begin
y = &A;
fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long integer A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_wait();
fpu_send(XOP);          //Return the status byte immediately
fpu_send(LSTATUS);
fpu_readDelay();

return fpu_read();
end

//LSUB A = A - B
//The long integer value B is subtracted from A and the result is
stored at A
```

```
long fpu_LSUB(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(LSUB + 1);     //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LTST
//Return status of the bitwise AND value of two long integer A and B
//Set Bit 1 if negative
//Set Bit 0 if zero
char fpu_LTST(long A, long B)
begin
y = &A;
fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long integer A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_wait();
```

```
fpu_send(XOP);          //Return the status byte immediately
fpu_send(LTST);
fpu_readDelay();

return fpu_read();
end

//LUCOMPARE
//Compare unsigned long values A and B. Return status byte.
//Set Bit 1 if A < B
//Set Bit 0 if A = B, If Neither Bit 0 or Bit 1 is set, A > B
char fpu_LUCOMPARE(unsigned long A, unsigned long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the long value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_wait();             //Return the status byte immediately
fpu_send(XOP);
fpu_send(LUCOMPARE);
fpu_readDelay();

return fpu_read();
end

//LUDIV
// long unsigned integer A is divided by the unsigned long integer B
unsigned long fpu_LUDIV(unsigned long A, unsigned long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA + 1);  //Write the long value A to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 2);  //Write the long value B to register 2
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
```

```
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LUDIV + 2);    //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD + 1);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LUMOD
// long unsigned integer A is divided by the unsigned long integer B
unsigned long fpu_LUMOD(unsigned long A, unsigned long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA + 1);  //Write the long value A to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 2);  //Write the long value B to register 2
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LUDIV + 2);    //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//LXOR
//Bitwise XOR
```

```
long fpu_LXOR(long A, long B)
begin
y = &A;

fpu_send(XOP);
fpu_send(LWRITEA);      //Write the long value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(LXOR);         //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(XOP);
fpu_send(LREAD);
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return longreturn;
end

//MAX
//The maximum floating point value of A and B
float fpu_MAX(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(MAX);          //Perform Calculation
```

```
fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//MIN
//The minimum floating point value of A and B
float fpu_MIN(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(MIN);          //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//NEGATE
// Negate the floating point value A
float fpu_NEGATE(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));
```

```
fpu_send(NEGATE);        //Perform Calucalation

fpu_wait();              //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//POWER
//Raised floating value A to the power of floating value B
float fpu_POWER(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(POWER);        //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//RADIANS Convert degrees to radians
float fpu_RADIANS(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
```

```
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(RADIANS);       //Perform Calucalation

fpu_wait();              //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//ROOT
//Calculate the nth root (value B) of floating value A
float fpu_ROOT(float A, float B)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

y = &B;
fpu_send(FWRITEB + 1);  //Write the floating value B to register 1
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(XOP);
fpu_send(ROOT);         //Perform Calculation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//ROUND
//Round off the floating point A to the nearest integer in floating
point value
float fpu_ROUND(float A)
begin
y = &A;
```

```
fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(ROUND);        //Perform Calucalation

fpu_wait();             //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//SIN A = sin (A)
//Calculates the sine of an angle in radians
float fpu_SIN(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(SIN);           //Perform Calucalation

fpu_wait();              //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//SQRT A = sqrt (A)
//Calculates the square root of the floating point value A
float fpu_SQRT(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
```

```
fpu_send(* (y));

fpu_send(SQRT);          //Perform Calucalation

fpu_wait();              //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//TAN A = tan (A)
//Calculates the tangent of an angle in radians
float fpu_TAN(float A)
begin
y = &A;

fpu_send(FWRITEA);      //Write the floating value A to register 0
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));

fpu_send(TAN);           //Perform Calucalation

fpu_wait();              //Return the result from register 0
fpu_send(FREAD);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();

return floatreturn;
end

//*****************************************************************
*
//User defined functions
//Write floating Value A into register reg (0-15)
void fpu_FRegisterA(float A, char reg)
begin
y = &A;

fpu_send(FWRITEA + reg);            //Write the floating value A to
register reg
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));
```

```
end

//Write long Value A into register reg (0-15)
void fpu_LRegisterA(long A, char reg)
begin
y = &A;
fpu_send(XOP);
fpu_send(LWRITEA + reg);      //Write the long value A to register reg
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));
end

//Write floating Value B into register reg (0-15)
void fpu_FRegisterB(float B, char reg)
begin
y = &B;
fpu_send(FWRITEB + reg);      //Write floating value B to register reg
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));
end

//Write long Value B into register reg (0-15)
void fpu_LRegisterB(long B, char reg)
begin
y = &B;
fpu_send(XOP);
fpu_send(LWRITEB + reg);      //Write the long value B to register reg
fpu_send(* (y+3));
fpu_send(* (y+2));
fpu_send(* (y+1));
fpu_send(* (y));
end

//Return the floating result from register reg
float fpu_FReadRegister(char reg)
begin
fpu_wait();              //Return the floating result from register reg
fpu_send(FREAD+reg);
fpu_readDelay();
y = &floatreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();
return floatreturn;
end

//Return the long result from register reg
long fpu_LReadRegister(char reg)
begin
fpu_wait();              //Return the long result from register reg
fpu_send(XOP);
fpu_send(LREAD+reg);
```

```
fpu_readDelay();
y = &longreturn;
*(y+3) = fpu_read();
*(y+2) = fpu_read();
*(y+1) = fpu_read();
*(y)   = fpu_read();
return longreturn;
end

void  fpu_Function(char function)
begin
fpu_send(function);         //Perform Calucalation
end

void  fpu_XOPFunction(char function)
begin
fpu_send(XOP);
fpu_send(function);
end
```