

Lab 8: Final Project

This laboratory will explore the use of floating-point numbers and operators. A floating-point co-processor will be developed as an add-on unit for a general-purpose RISC microprocessor. Additionally, an adaptive filter will be implemented in software that will run on the resulting floating-point processor.

Since this is a larger project, you may work in groups of four (or less) to complete the project.

Background

Floating-point Format

For this lab, use a 16-bit floating-point format. As shown in Figure L8.1, this format uses one sign bit, a 5-bit exponent field, and a 10-bit mantissa field such that $x = sign * 1.mantissa * 2^{exponent}$.

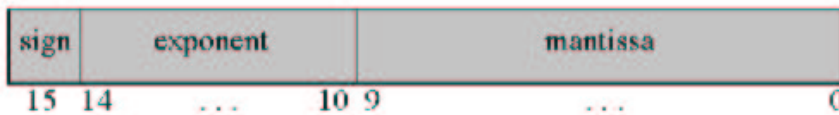


Figure L8.1: This is the format that you should use for the floating-point numbers in your FPU.

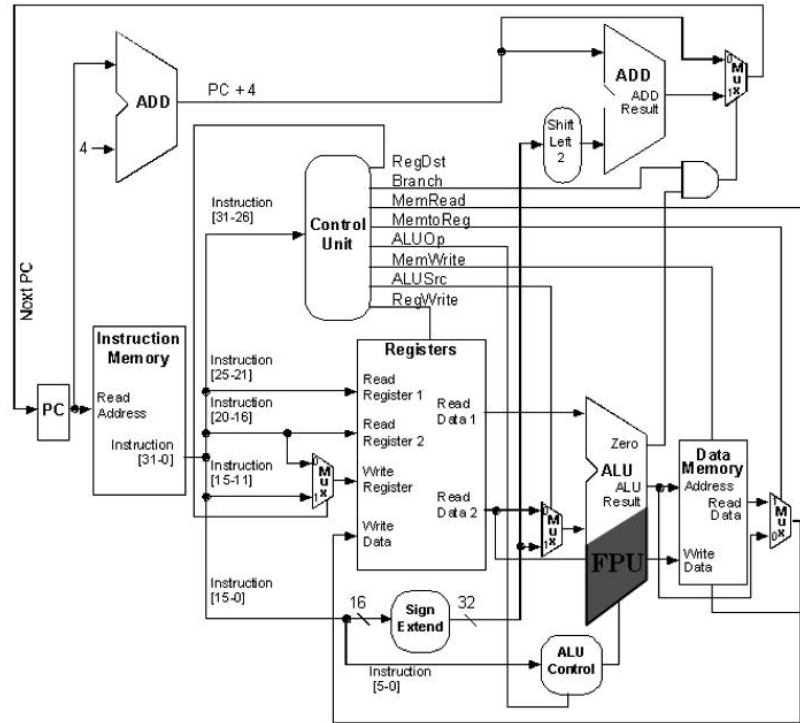


Figure L8.2: Your floating-point unit will be added to the standard single-cycle MIPS datapath as shown here.

Microprocessor

A single cycle implementation of the MIPS processor will be used for this laboratory project. A thorough description of the MIPS processor can be found in *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, by Patterson and Hennessy, Morgan Kaufman Publishers, 1998. Additionally, a description of the VHDL implementation of the MIPS processor that you will use in this lab can be found in *Rapid Prototyping of Digital Systems: A Tutorial Approach*, Second Edition, by Hamblen and Furman, Klumer Academic Publishers, 2001.

The MIPS implementation used here will employ a 32-bit instruction word and 16-bit datapath with eight general-purpose registers. Figure L8.2 shows an outline of the MIPS datapath with the addition of the floating-point unit (FPU).

```

ENTITY fpu IS
  PORT( clock      : IN  STD_LOGIC;
        data_rdy   : IN  STD_LOGIC;
        opcode     : IN  STD_LOGIC_VECTOR( 1 DOWNT0 0);
        A_in, B_in : IN  STD_LOGIC_VECTOR(15 DOWNT0 0);
        FPU_out    : OUT  STD_LOGIC_VECTOR(15 DOWNT0 0)    );
END fpu;

```

Figure L8.3: The ENTITY declaration for your floating-point unit should be the same as shown here.

Adaptive Filtering

*** This section will be completed by next week. You do not need it yet. ***

Task 1: Floating-point Unit

To be compatible with the provided MIPS microprocessor implementation, your floating-point unit should be defined with the ENTITY statement shown in Figure L8.3. Of course, feel free to add as many extra signals as you need for debugging while you design the FPU, but remove these signals before integrating your FPU with the MIPS processor.

Consider designing the FPU as a state machine. Use the **opcode** signal to determine which state path to follow and thus which floating-point operation to perform. Figure L8.4 shows one possibility for the overall FPU design.

Write the VHDL implementation for a floating-point adder, subtractor, and multiplier. You may want to do this in Altera's MAX+plus II software since it has an easy-to-use VHDL simulator. A tutorial for the Altera software is available if you are not familiar with it.

Task 2: Integrating your FPU with the MIPS processor

Download the MIPS processor files from WebCT. Add your **fpu.vhd** file to the project directory, and compile the top-level design file, **top_spm.vhd**.

Test your processor and FPU's functionality with the sample assembly

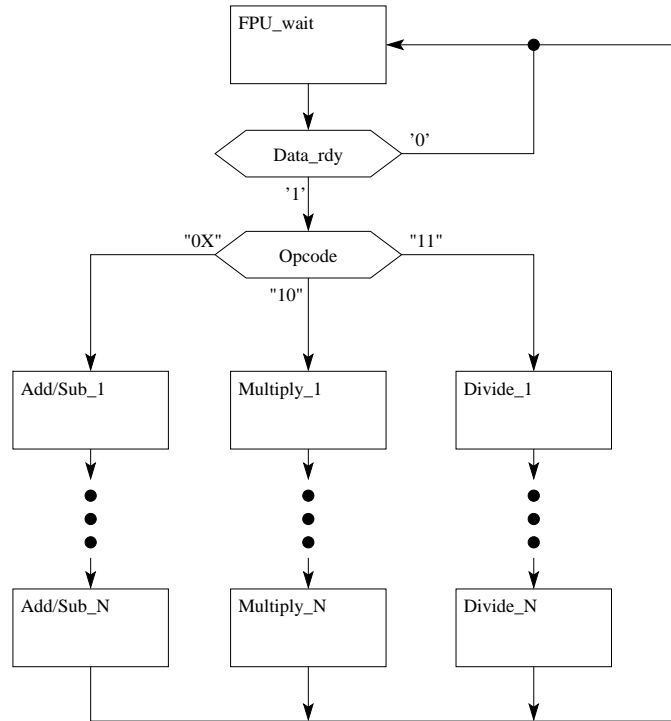


Figure L8.4: This is one possible state machine design for the FPU.

code provided.

Sample assembly code

The sample assembly file, `program.src`, shown in Figure L8.5 has been provided. This code runs a few tests of the floating-point unit. Specifically, it adds, subtracts, and multiplies positive and negative variations of 11 and 34. These values are preloaded into data memory at locations 0x02 through 0x09 as shown in Figure L8.6. Remember that the values are stored in the 16-bit floating-point format introduced earlier.

Assembly code to COE file

To load an assembly program into your processor, you must convert it to machine code and then format it in the correct format (*.coe file format as defined in the Xilinx memory datasheets).

The first part (assembly to machine code conversion) is handled by a

```

TITLE EXAMPLE MIPS COMPUTER ASSEMBLY LANGUAGE TEST PROGRAM
LINES 50
LIST F,W
ORG H#00

; Read in Instruction Macro Definitions
INCLUDE MIPS.MAC

;-----
;PROGRAM START
;-----

START: LW    R2, H#0002, R0    ; Load the data from memory address 0x02 into R2
      LW    R3, H#0003, R0    ; Load the data from memory address 0x03 into R3
      FPADD R1, R2, R3        ; Add the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0010, R0    ; Store the data in R1 into memory address 0x10
      LW    R2, H#0004, R0    ; Load the data from memory address 0x04 into R2
      LW    R3, H#0005, R0    ; Load the data from memory address 0x05 into R3
      FPADD R1, R2, R3        ; Add the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0011, R0    ; Store the data in R1 into memory address 0x11
      LW    R2, H#0006, R0    ; Load the data from memory address 0x06 into R2
      LW    R3, H#0007, R0    ; Load the data from memory address 0x07 into R3
      FPADD R1, R2, R3        ; Add the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0012, R0    ; Store the data in R1 into memory address 0x12
      LW    R2, H#0008, R0    ; Load the data from memory address 0x08 into R2
      LW    R3, H#0009, R0    ; Load the data from memory address 0x09 into R3
      FPADD R1, R2, R3        ; Add the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0013, R0    ; Store the data in R1 into memory address 0x13
      LW    R2, H#0002, R0    ; Load the data from memory address 0x02 into R2
      LW    R3, H#0003, R0    ; Load the data from memory address 0x03 into R3
      FPSUB R1, R2, R3        ; Subtract the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0014, R0    ; Store the data in R1 into memory address 0x14
      LW    R2, H#0004, R0    ; Load the data from memory address 0x04 into R2
      LW    R3, H#0005, R0    ; Load the data from memory address 0x05 into R3
      FPSUB R1, R2, R3        ; Subtract the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0015, R0    ; Store the data in R1 into memory address 0x15
      LW    R2, H#0006, R0    ; Load the data from memory address 0x06 into R2
      LW    R3, H#0007, R0    ; Load the data from memory address 0x07 into R3
      FPSUB R1, R2, R3        ; Subtract the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0016, R0    ; Store the data in R1 into memory address 0x16
      LW    R2, H#0008, R0    ; Load the data from memory address 0x08 into R2
      LW    R3, H#0009, R0    ; Load the data from memory address 0x09 into R3
      FPSUB R1, R2, R3        ; Subtract the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0017, R0    ; Store the data in R1 into memory address 0x17
      LW    R2, H#0002, R0    ; Load the data from memory address 0x02 into R2
      LW    R3, H#0003, R0    ; Load the data from memory address 0x03 into R3
      FPMULT R1, R2, R3        ; Multiply the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0018, R0    ; Store the data in R1 into memory address 0x18
      LW    R2, H#0004, R0    ; Load the data from memory address 0x04 into R2
      LW    R3, H#0005, R0    ; Load the data from memory address 0x05 into R3
      FPMULT R1, R2, R3        ; Multiply the FP values in R2 and R3 and put the result in R1
      SW    R1, H#0019, R0    ; Store the data in R1 into memory address 0x19
      LW    R2, H#0006, R0    ; Load the data from memory address 0x06 into R2
      LW    R3, H#0007, R0    ; Load the data from memory address 0x07 into R3
      FPMULT R1, R2, R3        ; Multiply the FP values in R2 and R3 and put the result in R1
      SW    R1, H#001A, R0    ; Store the data in R1 into memory address 0x1A
      LW    R2, H#0008, R0    ; Load the data from memory address 0x08 into R2
      LW    R3, H#0009, R0    ; Load the data from memory address 0x09 into R3
      FPMULT R1, R2, R3        ; Multiply the FP values in R2 and R3 and put the result in R1
      SW    R1, H#001B, R0    ; Store the data in R1 into memory address 0x1B

END

```

Figure L8.5: A sample assembly file is provided that tests the floating-point unit. This code adds, subtracts, and multiplies the floating-point values stored in data memory locations 0x02 through 0x09.

```
[00..FF] : 0000;  
  
02 : 4980;  
03 : 5040;  
04 : C980;  
05 : 5040;  
06 : 4980;  
07 : D040;  
08 : C980;  
09 : D040;
```

Figure L8.6: In the provided example, data memory is initialized to hold positive and negative variations of the values 11 and 34. Remember that these values are stored in 16-bit floating-point format.

meta-assembler. For this lab, we will use the WinTim Meta Assembler developed at Georgia Tech. Download the WinTim.zip file from WebCT and unzip the files into a temporary directory on your computer. Next follow these instructions to assemble your source code:

1. Run *WinTim32.exe* (one of the files you just downloaded).
2. Open the instruction set definition file, `mips.def`, by selecting **File** \Rightarrow **Open** \Rightarrow **mips.def**.
3. Open your source code (i.e., `program.src`) by selecting **File** \Rightarrow **Open** \Rightarrow **program.src**.
4. Process the definition file by selecting the window containing `mips.def` and then select **Assembly** \Rightarrow **Meta-Assembly**. (There should be no errors.)
5. Assemble your source code by selecting the window containing `program.src` and then select **Assembly** \Rightarrow **Assemble using definition file**.
6. Open the assembled code by selecting **Output** \Rightarrow **Assembled data (as MIF)**.
7. Finally, save the assembled code by selecting **File** \Rightarrow **Save As...** then click the **Save** button.

If these steps processed without errors, then a `program.mif` file is created. Copy this file into your Altera project directory and recompile `top_spim.vhd`. Alternatively, Altera allows you to initialize the

program memory without recompiling. To do this, open the **Simulator** window and select **Initialize** ⇒ **Initialize Memory**. Select the instruction memory from the drop-down box and then click Import to specify the new **program.mif** file.

If you are using Xilinx, you will need to manually edit this file to match the format of the COE file specified by Xilinx and rename it **program.coe**. Then you must specify **program.coe** as the initial memory contents of the *instruction memory* when you create it in Xilinx's CORE Generator.

More information on the WinTim assembler can be found in Appendix D of *Rapid Prototyping of Digital Systems: A Tutorial Approach*, Second Edition, by Hamblen and Furman, Kluwer Academic Publishers, 2001.

Testing your Floating-point Unit

Use the Altera simulator to simulate the execution of the sample assembly code provided. An Altera waveform file for this project (*top_spim.scf*) is included in the provided files.

Before running the simulation, open the **Simulator** window and select **Initialize** ⇒ **Initialize Memory**. Choose the data memory from the drop-down box and take a look at the contents of memory before the simulation is run. The data memory should look like Figure L8.7. These data memory values were preloaded using the **dmemory.mif** file (same concept as using a *.coe file to initialize a Coregen memory block).

Exit the memory viewer and click on the **Start** button in the **Simulator** window. This will run the simulation. Clicking on the **Open SCF** button will open a new window and show the contents of the simulation. You can modify the inputs to create a new simulation for additional tests if needed.

Go back to the **Simulator** window and open the memory viewer again. Now, the contents of data memory should show the results of executing the sample code. Memory addresses 0x10 through 0x1B should contain the results of the floating-point calculations and should match the values shown in Figure L8.8. If your values do not match these, then there is probably an error in your implementation of the floating-point unit. Debug the errors and re-run the simulation until the values match.

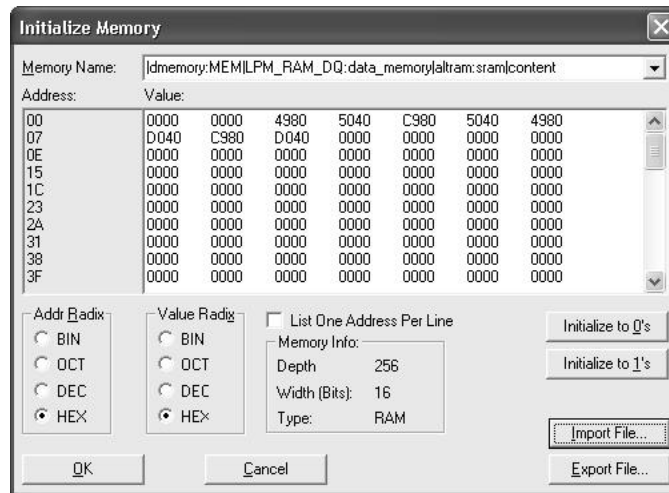


Figure L8.7: The contents of data memory *before* the simulation should have values at memory locations 0x02–0x09 and zeros everywhere else.

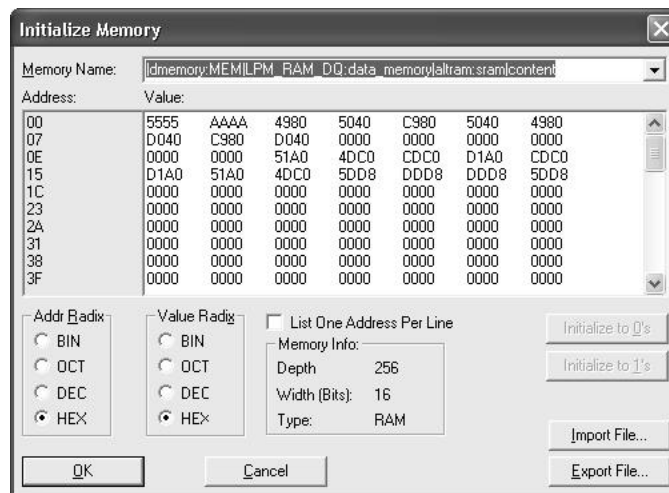


Figure L8.8: The contents of data memory *after* the simulation should have these values at memory addresses 0x10–0x1B.

Task 3: Writing an Adaptive Filter for the MIPS processor

Write assembly code to implement an adaptive filter algorithm in software. Use the expanded instruction set for your MIPS processor shown in the table below.

Operation	Format	Instruction Word												
Addition	add rd, rs, rt	<table border="1"> <tr> <td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x20</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	rs	rt	rd	0	0x20	6	5	5	5	5	6
0	rs	rt	rd	0	0x20									
6	5	5	5	5	6									
Addition Immediate	addi rt, rs, imm	<table border="1"> <tr> <td>8</td><td>rs</td><td>rt</td><td colspan="3">imm</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td colspan="3">16</td> </tr> </table>	8	rs	rt	imm			6	5	5	16		
8	rs	rt	imm											
6	5	5	16											
Addition Unsigned	addu rd, rs, rt	<table border="1"> <tr> <td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x21</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	rs	rt	rd	0	0x21	6	5	5	5	5	6
0	rs	rt	rd	0	0x21									
6	5	5	5	5	6									
Subtraction	sub rd, rs, rt	<table border="1"> <tr> <td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x22</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	rs	rt	rd	0	0x22	6	5	5	5	5	6
0	rs	rt	rd	0	0x22									
6	5	5	5	5	6									
Subtraction Unsigned	subu rd, rs, rt	<table border="1"> <tr> <td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x23</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	rs	rt	rd	0	0x23	6	5	5	5	5	6
0	rs	rt	rd	0	0x23									
6	5	5	5	5	6									
Bitwise AND	and rd, rs, rt	<table border="1"> <tr> <td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x24</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	rs	rt	rd	0	0x24	6	5	5	5	5	6
0	rs	rt	rd	0	0x24									
6	5	5	5	5	6									
Bitwise OR	or rd, rs, rt	<table border="1"> <tr> <td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x25</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	rs	rt	rd	0	0x25	6	5	5	5	5	6
0	rs	rt	rd	0	0x25									
6	5	5	5	5	6									
Shift Left Logical	sll rd, rt, shamt	<table border="1"> <tr> <td>0</td><td>X</td><td>rt</td><td>rd</td><td>shamt</td><td>0</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	X	rt	rd	shamt	0	6	5	5	5	5	6
0	X	rt	rd	shamt	0									
6	5	5	5	5	6									
Shift Right Logical	srl rd, rt, shamt	<table border="1"> <tr> <td>0</td><td>X</td><td>rt</td><td>rd</td><td>shamt</td><td>0</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	X	rt	rd	shamt	0	6	5	5	5	5	6
0	X	rt	rd	shamt	0									
6	5	5	5	5	6									
Set if Less Than	slt rd, rs, rt	<table border="1"> <tr> <td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x2A</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td> </tr> </table>	0	rs	rt	rd	0	0x2A	6	5	5	5	5	6
0	rs	rt	rd	0	0x2A									
6	5	5	5	5	6									
Load Upper Immediate	lui rt, imm	<table border="1"> <tr> <td>0x0F</td><td>0</td><td>rt</td><td colspan="3">imm</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td colspan="3">16</td> </tr> </table>	0x0F	0	rt	imm			6	5	5	16		
0x0F	0	rt	imm											
6	5	5	16											
Load Word	lw rt, offset(rs)	<table border="1"> <tr> <td>0x23</td><td>rs</td><td>rt</td><td colspan="3">Offset</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td colspan="3">16</td> </tr> </table>	0x23	rs	rt	Offset			6	5	5	16		
0x23	rs	rt	Offset											
6	5	5	16											
Store Word	sw rt, offset(rs)	<table border="1"> <tr> <td>0x2B</td><td>rs</td><td>rt</td><td colspan="3">Offset</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td colspan="3">16</td> </tr> </table>	0x2B	rs	rt	Offset			6	5	5	16		
0x2B	rs	rt	Offset											
6	5	5	16											
Branch on Equal	beq rs, rt, label	<table border="1"> <tr> <td>4</td><td>rs</td><td>rt</td><td colspan="3">Offset</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td colspan="3">16</td> </tr> </table>	4	rs	rt	Offset			6	5	5	16		
4	rs	rt	Offset											
6	5	5	16											
Branch on Not Equal	bne rs, rt, label	<table border="1"> <tr> <td>5</td><td>rs</td><td>rt</td><td colspan="3">Offset</td> </tr> <tr> <td>6</td><td>5</td><td>5</td><td colspan="3">16</td> </tr> </table>	5	rs	rt	Offset			6	5	5	16		
5	rs	rt	Offset											
6	5	5	16											
Jump	j target	<table border="1"> <tr> <td>2</td><td colspan="5">target address</td> </tr> <tr> <td>6</td><td colspan="5">26</td> </tr> </table>	2	target address					6	26				
2	target address													
6	26													
Jump and Link	jal target	<table border="1"> <tr> <td>3</td><td colspan="5">target address</td> </tr> <tr> <td>6</td><td colspan="5">26</td> </tr> </table>	3	target address					6	26				
3	target address													
6	26													
Jump Register	jr rs	<table border="1"> <tr> <td>0</td><td>rs</td><td colspan="3">0</td><td>8</td> </tr> <tr> <td>6</td><td>5</td><td colspan="3">15</td><td>6</td> </tr> </table>	0	rs	0			8	6	5	15			6
0	rs	0			8									
6	5	15			6									

Operation	Format	Instruction Word												
Floating-point Addition	FPadd rd, rs, rt	<table border="1"> <tr> <td>0</td> <td>rs</td> <td>rt</td> <td>rd</td> <td>0</td> <td>0x10</td> </tr> <tr> <td>6</td> <td>5</td> <td>5</td> <td>5</td> <td>5</td> <td>6</td> </tr> </table>	0	rs	rt	rd	0	0x10	6	5	5	5	5	6
0	rs	rt	rd	0	0x10									
6	5	5	5	5	6									
Floating-point Subtraction	FPsub rd, rs, rt	<table border="1"> <tr> <td>0</td> <td>rs</td> <td>rt</td> <td>rd</td> <td>0</td> <td>0x12</td> </tr> <tr> <td>6</td> <td>5</td> <td>5</td> <td>5</td> <td>5</td> <td>6</td> </tr> </table>	0	rs	rt	rd	0	0x12	6	5	5	5	5	6
0	rs	rt	rd	0	0x12									
6	5	5	5	5	6									
Floating-point Multiply	FPmult rd, rs, rt	<table border="1"> <tr> <td>0</td> <td>rs</td> <td>rt</td> <td>rd</td> <td>0</td> <td>0x14</td> </tr> <tr> <td>6</td> <td>5</td> <td>5</td> <td>5</td> <td>5</td> <td>6</td> </tr> </table>	0	rs	rt	rd	0	0x14	6	5	5	5	5	6
0	rs	rt	rd	0	0x14									
6	5	5	5	5	6									

Report for Final Project

The report for the final project is due at the beginning of class on Thursday, December 5. In this report, discuss the design of the floating-point unit and the adaptive filtering code. Also, discuss the results of your filter and be sure to include and reference *all* relevant plots, diagrams, VHDL code, etc. generated during this project.

Hint: When including VHDL code in your report, only include small snippets of code that you want to highlight in figures within the text. Then be sure to reference these figures in your discussion. The full VHDL listing should be included in an Appendix.