###### *CSE 260 – Introduction to Digital Logic and Computer Design Jon Turner*

Lab 5 Report

##### *Your name: 5/1/2014*

***Part A***. (40 points) In this part, you are required to implement three new instructions (*lShift, rShift* and *or*) for the *WashU-2* processor, as described in the lab writeup. In addition, you are required to write a test program to verify that your instructions work correctly and then run a simulation of the processor executing your test program. You should write your test program in the *WashU-2* assembly language (your program should execute each of the instructions at least three times) and then use the provided assembler to convert the assembly language version to machine language. On a Mac or Linux computer, this is most easily done by typing the following lines in a shell window.

javac Assembler.java

java Assembler myProgramFile

where *myProgramFile* is the name of the file that contains your test program. This will also work on *oasis.cec.wustl.edu* using a command window. If you’re more comfortable running *java* programs using Eclipse, you can do that too. The output from the assembler is a list of initializers for the *WashU-2*’s memory. You should paste the output into the initialization section of the memory array within the *ram* component. You can use the provided *testTop1* testbench to start the processor running.

Paste a copy of the VHDL for the *cpu* component of the *WashU-2* below. **Highlight your changes to the code by making them bold**.

paste your code here

Paste a copy of your test program below.

paste your code here

Simulate the processor using the provided *testTop1* testbench. Paste a copy of your simulation output below. Your waveform window should include all the memory signals, the processor *state*, the *tick* register, plus *IAR*, *PC* and *ACC*. Your simulation output should demonstrate all of the instructions working correctly. You will probably need more than one screenshot to show this adequately. Make sure your signals are well-organized, with labeled dividers for the memory signals and the processor signals. Explain how the simulation demonstrates the correct operation of all the instructions.

*paste your screenshot(s) here*

***Part B***. (40 points) In your repository, you will find a file, *divide.asm* that contains an incomplete version of a division subprogram. Complete the program and paste a copy of the completed program below (after you have gotten it working correctly). Be sure to include all of the comments from the original file, so that there is an obvious correspondence between the assembly language and the provided pseudo-code. **Highlight your changes by making them bold.**

paste your code here

In your repo, you will also find a file called *main.asm* that contains an incomplete version of a test program for the division subprogram, plus a file *mult.asm* that contains a multiplication subprogram that is needed for the test program. Complete the test program and paste a copy of the completed program below. **Highlight your changes by making them bold.**

paste your code here

Combine the three files into a single file with the main program first, followed by the division and multiplication program. (On a Mac or Linux computer, this can be done by typing “*cat main.asm divide.asm mult.asm >testDiv.asm*”, for example.) Next, use the assembler to produce the machine language version and paste the output of the assembler into the *ram* module (replacing the code from the previous part). Now, simulate the processor running this program. Increase the length of the final *wait* in the testbench to at least 10 ms. For this simulation, your waveform window should include the signals *i, j, q, r, div\_x, div\_y, div\_q, div\_r, mult\_a, mult\_b* and *mult\_prod* from the *ram* component. These correspond to program variables in the main program, the division subprogram and the multiply subprogram. You will find these useful when debugging. Use the unsigned radix for all these signals.

Paste the simulation output showing the entire time period when the program variable *i* in the main program is equal to 23. Make sure that the values of *j, q* and *r* are clearly visible throughout this period. Verify that the results computed by the division subroutine and saved in *q* and *r* are correct.

*paste your screenshot here*

***Part C.*** (30 points). In this part, you are to implement the interrupt instructions described in the lab writeup. You are also to extend the console, so that it implements the “continuous input” mode. Paste a copy of your VHDL for the modified *cpu* below. **Highlight your changes by making them bold.**

paste your code here

Paste a copy of your VHDL for the modified *console* below. **Highlight your changes by making them bold.**

paste your code here

***Part D***. (50 points) In this part, you will write an *Etch-a-Sketch* program that runs whenever new data is written to memory, causing an interrupt to be triggered. Before your program can run, you need to setup the interrupt subsystem so that the interrupt vector points to the start of your interrupt program, and interrupts are enabled. Write a short code fragment that does this and insert it at the start of the main program you wrote earlier to test the division program. Paste a copy of this code fragment below.

paste your code here

You will find a file with an incomplete version of the *Etch-a-Sketch* program in your repository. Complete this program. Note that the program needs to use the division subroutine we wrote earlier, but since the main program is also going to be performing divisions, we will need a separate copy of the division program that can be used here. Call it *div2* and adjust all the labels in this copy so that they start with *div2* instead of *div*. Paste a copy of the *Etch-a-Sketch* program below. **Highlight your changes by making them bold.**

paste your code here

You will need to combine your *Etch-a-Sketch* program with the other components. Combine the modified version of *main*, the *division* subprogram, the *multiply* subprogram, the *Etch-Sketch* subprogram and the *div2* subprogram, into a single file, in that order. Than, use the assembler to generate the machine language version and paste the output into the *ram*. Simulate your program with the provided *testTop2* testbench. This includes input signals from the knob that should cause your *Etch-a-Sketch* program to draw a square on the screen. Paste a copy of the waveform window covering the time period from 3.4 ms to 4 ms. Your waveform window should include the signals *i, j, p* and *q* from the *ram* module, as before, and the signals *etch\_pix, etch\_x, etch\_y, etch\_p, etch\_q, etch\_r* from the *ram* module. It should also include the interrupt signal.

*paste your screenshot here*

Verify that the several of the values computed by your program for *etch\_x* and *etch\_y* are correct, given the *etch\_pix values.* Explain how the values match.

Verify that several of the values of *etch\_p* are correct, given the values of *etch\_x* and *etch\_y*. Explain how the values match.

Verify that the test program for the division subprogram continues to work correctly, even when it is being interrupted periodically by the *Etch-a-Sketch* program.

Next, find a place in the simulation where the interrupt signal goes high and zoom in on this part of the simulation. Here, your waveform window should show the memory signals, the processor registers and all the interrupt signals including the two shadow signals.

*paste your screenshot here*

Verify that all signal changes that happen here are correct. Explain.

From this point in the simulation scan ahead to the point where the interrupt subprogram returns (you should see *intEn* go high at this point). Paste a copy of the waveform below.

*paste your screenshot here*

Verify that the processor is correctly handling the interrupt return. Explain.

***Part E***. (10 points) Proceed to this part only after you have completed the simulation in Part *F* and have convinced yourself that the complete circuit will work correctly when transferred to the prototype board. Prototype your circuit using one of the prototype boards available in Bryan 316. Once you have your circuit loaded onto the board and you have convinced yourself that it works correctly, fill in your name below on the printed copy and have one of the TAs check it and sign their name below, after assigning the appropriate number of demo points..

Student name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ has successfully demonstrated the *Etch-a-Sketch* program running on the modified *WashU-2*.

TA name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TA signature:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Demo points (out of 10):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comments (if the program does not work 100% correctly, make a note of all issues below):