CSE30321 Computer Architecture I

Lab 2: Implementation and Usage of a Simple Processor

Assigned: Week of September 8th. Due: September 17th.

1 What am I going to do?

- Become familiar with the Verilog code for the 6-instruction processor – more specifically the finite state machine (FSM) needed to properly control the processor. (*Suggestion:* make sure that you really understand how the FSM works. In later labs, you'll need to add new instructions to the 6-instruction processor and will have to modify this FSM accordingly.)
- Get more practice with coding in assembly – specifically for the 6-instruction processor. (In lab, we'll look at a more complex example than what we've done in class or in the HW thus far...)
- Learn to use a simple assembler/simulator. (You should find this to be a useful debugging tool that you can use in future assignments.)
- Study how changes to a processor datapath, reflected by a FSM, can affect overall performance.

2 Useful References

- Frank Vahid's Digital Design (Ch. 8, 9).
- CSE20221 Handouts on Verilog (links on course web page).

3 Processor Design in Verilog

3.1 What to do:

- Complete the Verilog implementation of a processor similar to the six-instruction processor described in Chapter 8.4 of the textbook.
- Verify and debug the processor design by functional simulation.

3.2 Procedure

1. Open ISE and create a new project called "lab2" in an empty directory. Download and extract the zip file for this lab into a separate directory (the zip file is located on the course web page). In ISE, add the extracted files to the project by clicking the Project menu and then "Add Copy of Source..." (be sure to choose "Files of type: All files" so everything is copied).
2. Look through each of the Verilog files to understand the structure of the processor. Note the hierarchy of the modules in the project and how it relates to the organization of the processor as described in Vahid Fig. 8.12.
3. Open the file "controller.v". A number of "holes" exist in the FSM that describes the controller. Fill in the holes in the FSM based on the state diagram in the class notes.
4. Use different instructions to test all six of the opcodes. Put the desired instructions in "inst_mem.v" and any desired data in "data_mem.v". Both memory files have a few lines showing how to input data (e.g. "mem[0] = 16'h0000;" in data_mem.v, and "0: douta <= 16'h0000;" in inst_mem.v). Remember to keep all your addresses in hexadecimal to avoid confusion.

5. Test your program in ModelSim or ISE Simulator. Print out parts of the waveform that demonstrate the functionality of the instructions. If the instructions do not work as intended, use ModelSim or ISE Simulator to determine why, and then fix the problems.

6. Record the number of cycles each instruction type took.

   Note: The finite state machine implemented in this processor differs slightly from that in Vahid Ch. 8. The reason for this is that the memory modules created by the ISE Core Generator have a read latency of 1 clock cycle, when in reality we expect the data from memory to be available immediately after a read. This causes a problem in states where memory is read (i.e. in Fetch and Load) because the data from memory is expected to be available in the same state we say to read. To fix this problem, wait states were inserted after the Fetch and Load states to give the memory time to output its data. In this lab the Core Generator is not used to save the frustration of having to wait for the cores to generate each time a change is made. Instead, we use a behavioral model of the memories that is basically two very large register files. In order to preserve compatibility with the Basys boards, we kept the convention of using wait states. (However, you are not required to do a Basys board download.)

4 The Assembler and Simulator

4.1 What to do:

- Write assembly code based on given pseudocode.
- Use the assembler to test assembly code.

4.2 Procedure

1. Open the assembler/simulator in your web browser (link is on course web page, Java and Flash are required to view). Assemble and step through the example program to become familiar with the controls available.

2. Create and open a file called FreqCalc.s for your assembly code. Write a "frequency calculator" program in assembly that steps through a five element array in memory and counts the occurence of a certain value (store this count in memory). Following is the pseudocode for this program:

```
a = [1 24 24 17 1];
i = 0;
j = 1; // this is the value to search for
count = 0;
for(i = 0; i < 5; i++)
    if a[i] == j
    count++;
```

3. Test your program in the assembler by copying your code from FreqCalc.s to the first text box in the simulator. Click the "Assemble" button, which will update the next text box with the appropriate machine code. Step through your program until the final count is written to data memory, and take a screenshot of this. Include this screenshot of the final contents of data memory in your report.

4. Simulate your assembly with ModelSim or ISE Simulator by copying your code from FreqCalc.s into the instruction memory (by editing inst_mem.v) and the array values in the data memory (by editing data_mem.v). Print out waveforms that demonstrate correct functionality of your assembly code.

5. Record how many cycles it took to execute the program.

5 Question to Answer:

- Draw the state machine associated with the modified design described in Sec. 3 (i.e. where extra “wait” states are added). You do not need to show control signals.
- Assuming an instruction mix of 4 adds, 5 subtract, 3 jump-if-zeros that are taken, 6 jump-if-zeros that are not taken, 12 loads, 8 stores, and 4 load constants, what is the execution time if the processor’s clock rate is 75 MHz.
- What if the un-modified state machine is used? What is the difference in performance between the two versions?

6 What to Turn In:

For this lab, no demonstration is required. Your report should include the following:

- Answers to the questions in Sec. 5.
- Cycle data collected with proper explanation.
- The waveforms with appropriate explanation demonstrating that you properly fixed the issues in the controller, and the correct functionality of your assembly code. Be sure to annotate all waveforms to show key points of the simulation.
- The screenshot of the final memory contents of the online simulator.
- Put the modified "controller.v" and the memory contents files into one group member’s dropbox (note whose dropbox is used in the lab report, dropboxes are at: /afs/nd.edu/courses/cse/cse30321.01/dropbox/your_name).
- For the assembly task, put "FreqCalc.s" in the same dropbox.