CSE 30321: Computer Architecture I
Lab 6: Mini-Project
Assigned: Week of Nov. 3, Report Due: In lab week of Nov. 17
70 Points

A. Objectives

- Learn a new way to construct loops by expanding the ISA of Vahid's processor.
- Learn one way to do I/O with the processor.

B. References

- Frank Vahid's Digital Design (Ch. 8, 9).
- CSE20221 Handouts on Verilog (links on course web page).
- Processor hardware design concepts in Chapter 5 of P&H.
- **Note:** this lab requires more independent design compared with the previous labs. Start early!

C. Base Plus Offset Addressing

C.1 What to Accomplish

- Implement an instruction to perform base-plus-offset addressing to aid realizing loops in the Vahid's processor.

C.2 Procedure

1. Open ISE and create a new project called "lab6" in an empty directory. Download and extract the zipped Verilog files from the course web page into a different directory. In ISE, add the Verilog files by clicking "Add Copy of Source..." under the "Project" menu.

   Note: This lab is built on the original version of the processor with the separate instruction and data memories. Instructions such as `JMP` and `JMPN` are also given.

2. Extend the given processor design by adding a load instruction similar to the `lw` instruction in the MIPS ISA. Specifically, the instruction accesses memory via a base address (given in the instruction) and an offset stored in a register location. The assembly format of the instruction and its operation is given below:

   ```
   MOVR Ra, Rb, Base    // RF[Ra] = Data_Mem[RF[Rb] + Base]
   ```

   The machine code format of this instruction is

   opcode (4 bits): 8;
   Ra (4 bits): a value between 0–15;
   Rb (4 bits): a value between 0–15;
   Base (4 bits): a value in the 2's complement format.

3. Test the new instruction in ModelSim and print out waveforms demonstrating its correct execution.

4. Record the cycle count for the new instruction.

D. I/O Instructions

D.1 What to Accomplish

- Implement an instruction to accomplish simple I/O.

D.2 Procedure

1. One way to accomplish input or output is through dedicated registers similar to what you have done in the previous lab. However, to make the process more general, it is preferable to introduce new registers instead of using the registers from the register file (as you did in Lab 5). In this lab, you are asked to introduce 8 I/O registers. Assume these registers are named as Preg[7:0] (short for port registers).

2. To implement I/O operations with the newly introduced registers, you need to add an instruction that can transfer data between the register files and the port registers. The new instruction has the following assembly format and functionality:

   ```
   MOV Dx, Ra, Pb
     // If x = 0, RF[a] = Preg[b] (input); If x = 1, Preg[b] = RF[a] (output).
   ```

   The machine code for this instruction has the following format:

   opcode (4 bits): 14;
   Ra (4 bits): a value between 0–15;
   Pb (4 bits): a value between 0–7;
   Dx (4 bit): a value between 0 or 1.

   Note: `Pb` and `Dx` are not using all the bits given.

   Note: The order of fields in the machine code and that of the assembly are different. You should obey this order (and trust me they do ease the hardware design and assembler design).

3. Make necessary modifications to the datapath and control of the given processor design. It would be helpful if you first sketch your design on top of the datapth and control diagrams given in the class notes. Then proceed to modify the relevant Verilog files.

4. Test your instruction using ModelSim for both input and output, and print out waveforms demonstrating the correct operation for both cases.

5. Record the cycle count for the new instruction.

E. Implementation

E.1 What to Accomplish

- Write a simple program to test the new instructions.

E.2 Procedure

1. Write a program in assembly that searches an array stored in memory for a particular value given in Preg[0]. Upon completion of the program, Preg[5] will contain the number of times the value in Preg[0] occurs in memory. Furthermore, Preg[4] will contain either the value in Preg[0] if the value is found in the array, or the value in the array that is closest to the value in Preg[0].

2. Test you program with the Linux-based psim AND the ModelSim (or ISE) simulator. Print out register file contents and/or waveforms to show that the program functions correctly both with psim and the ModelSim (or ISE) simulator.

3. This program uses the new memory access instruction rather than self-modifying code. Discuss the differences between this method of looping and the self-modifying code in terms of performance, ease of coding, ease of understanding, and any other pros and cons you can think of.

F. What to Turn In

A formal report is required for this lab and is due in lab the week of November 17th. Your report should include the following:

- A description of the lab process, problems encountered, and how they were resolved.
- Answers to the questions in Parts C, D, and E and relevant discussions.
- Answers to the questions in the supplementary material posted on the course web site.

In addition, put all modified Verilog files and assembly for the new program into one group member's dropbox, and note whose dropbox is used in the lab report. Dropboxes are at:
/afs/nd.edu/courses/cse/cse30321.01/dropbox/your_name.