ECS 154B Lab 3, Spring 2016
Due by 11:55 PM on May 6, 2016
Via SmartSite

Objectives

- Build and test a multi-cycle MIPS CPU that implements a subset of the MIPS instruction set.
- Design a microcode control unit.

Description

In this lab, you will use Logisim to build a multi-cycle CPU, to further increase your knowledge of MIPS and as an introduction to microcode control. To test your CPU, you will run assembly language programs and simulate them in Logisim. You will be provided with a base project as a starting point. Appendix D of *Computer Organization and Design*, available online at `http://booksite.elsevier.com/9780124077263/downloads/advance_contents_and_appendices/appendix_D.pdf`, will be very useful for this lab. You can find it along with a sample program that you will be tested against in the Lab 3 assignment on SmartSite. You must implement the CPU control using microcode. The microcode implementation that you pick is up to you.

Details

In this lab, you will have a single RAM acting as both instruction and data memory. Just like Lab 2, you will have a 256 word-deep memory. You can logically reserve the first 128 words for instructions and the next 128 for data. This means that, in the programs you write and you will be tested on, **the PC should not go beyond the 128th word's address. Likewise, all memory locations accessed by `LW` and `SW` should be between the 129th and 256th words of memory.** As in Lab 2, the PC is incremented by 4, so, in your instruction memory, the first instruction will be at `0x00000000`, the second at `0x00000004`, the third at `0x00000008`, and so on.

You will be given an empty project to start your lab that includes an implementation of an ALU and a Register File. The file **Lab 3 Given.circ** contains these sub-circuits.

Instructions to Implement

Your CPU must execute the following instructions:

- All previous instructions from Lab 2.
  - `AND, ANDI, ADD, ADDI, OR, ORI, SUB, SLT, SLTU*, XOR, LW, SW, BEQ, J, JAL, JR`
- `SLL`
- `SRL`

As with the previous lab, the instructions for SLTU are incorrect in the given MIPS PDF. The instructions should read:

> If rs < rt with an unsigned comparison, put 1 into rd. Otherwise put 0 into rd.

For this lab, you must use microcode to implement the main control unit. The exact microcode implementation is up to you. All control signals must come from a ROM.

Subcircuits

Register File

- The register file is the same as in Lab 2.

ALU

- The ALU will now support `SLL` and `SRL`, and there is now a Shamt[4..0] (shift amount) input.
- When performing shift operations, the B input is shifted by the amount specified by Shamt.
- The control signal **ALUCtl** is described below:

Operation	ALUCtl3	ALUCtl2	ALUCtl1	ALUCtl0
`OR`	0	0	0	0
`SLTU`	0	0	0	1
`SLT`	0	0	1	0
`ADD`	0	0	1	1
`SUB`	0	1	0	0
`XOR`	0	1	0	1
`AND`	0	1	1	0
...				
`SLR`	1	1	1	0
`SLL`	1	1	1	1

Submission

In order to facilitate the grading of your lab, include probes with following labels and radixes so that the TAs can identify any of the following signals in your assignment. Some of them are already present in the given circuit. You may add pins in order to facilitate debugging, but do not change the existing ones.

Label Name	Radix	Description
PC	Unsigned Decimal	The current value of the PC.
WrReg	Binary	1 if writing the register file, 0 otherwise.
WrAddr	Unsigned Decimal	The address of the register that is going to be written to.
WrData	Signed Decimal	The value that is going to be written to the register.
MemWr	Binary	1 if writing to memory, 0 otherwise.
MemData	Signed Decimal	The data to be written to memory.
Branch	Binary	1 if taking a branch, 0 otherwise.

Include a README file with the following:

- Name of Partner 1, student ID number
- Name of Partner 2, student ID number
- Any known issues or comments you would like the TAs to see.

Submit your circuit and README to the Lab 3 Assignment on SmartSite. Only one partner should submit the assignment. Make sure to turn in the README with your names so that the TAs know who is working with whom. You may re-submit as many times as you would like. You have 3 slip days to use with this lab.

Grading

To grade your assignment, the TAs will run your CPU with the instructions in the file **mipsInstructions.lst** and look at the contents of your registers after the program is finished. If you look at the comments in **mipsInstructions.mps**, it will tell you what the final states of the registers and memory location should be. For each register or memory location with a correct value at the end of the run, you will receive 1 point. There are 13 points total.

- 90% Implementation
- 10% Interactive Grading

Hints

- It is likely easier to write code that generates a ROM initialization file than coding it by hand.
- Test and debug in steps. Start with a subset of the lab requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts and reduce the amount of time spent debugging as a whole. For example, you could implement the R-type instructions, then add the branch instruction, and finally add the memory access instructions.
- Think about the hardware you are creating before trying it out. The text is necessarily vague and leaves out details, so do not simply copy the figures and expect your CPU to work.
- As in the last lab, remember that though the PC and data addresses are 32 bits, the instruction memory and data memory addresses are only 8 bits. Be careful which bits you use to address the memory.