ECS 154B Lab 3, Spring 2017
Due by 11:59 PM on May 5, 2017
Via Canvas

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: The Hardware/Software Interface* will be very useful for this lab.

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.

Instructions to Implement

Your CPU must execute the following instructions:

- All previous instructions from Lab 2.
  - Data instructions: `ADD, ADDI, AND, ANDI, NOR, OR, ORI, SLT, SLTI, SLTU, SUB, XOR`
  - Memory access instructions: `LW, SW`
  - Control flow instructions: `BEQ, J, JAL, JR`
- `SLL`
- `SRL`

Note that the instructions for `SLTU` are incorrect. It should read:

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

Blocks

Control Unit

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.

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**
`XOR`	0	0	0	0
`SLTU`	0	0	0	1
`SLT`	0	0	1	0
`AND`	0	0	1	1
`NOR`	0	1	0	0
`SUB`	0	1	0	1
`OR`	0	1	1	0
`ADD`	0	1	1	1
`SRL`	1	0	0	0
`SLL`	1	0	0	1

Probes

The following probes have been provided to help you debug your circuit:

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.
Jump	Binary	1 if executing the jump instruction, 0 otherwise.
Jal	Binary	1 if executing the jump and link instruction, 0 otherwise.
Jr	Binary	1 if executing the jump register instruction, 0 otherwise.

Grading

To grade your assignment, the TAs will run your CPU with the instructions in the file **instructions.lst** and look at the contents of your registers after the program is finished. If you look at the comments in **instructions.mps**, it will tell you what the final states of the registers and memory locations should be.

- 50% Implementation
  - 50% for correct implementation of non-control instructions: registers 1 to 15 are all correct.
  - 50% for correct implementation of control instructions `BEQ`, `J`, `JAL`, `JR`. In addition to the above, register 16 is correct, registers 27 to 30 are zero, register 31 is correct, and the program ends in the infinite loop.
  - Partial credit at the grader's discretion.
- 50% Interactive Grading

Submission

Warning: read the submission instructions carefully. Failure to adhere to the instructions will result in a loss of points.

- Upload to Canvas the zip/tar of your .circ file along with a README file that contains:
  - The names of you and your partner.
  - Any difficulties you had.
  - Anything that doesn't work correctly and why.
  - Anything you feel that the graders should know.
- **Copy and paste the README into the text submission box when you are submitting your assignment**, as well.
- Only one partner should submit the assignment.
- You may submit your assignment as many times as you want.
- You have 3 slip days to use for this assignment.

Hints

- It is likely easier to write code that generates a ROM initialization file than coding it by hand, but it is by no means required.
- 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.
- It is helpful to construct an Excel spreadsheet of the instructions and the various control signals needed. In particular, the `&` concatenation operator and the `BIN2HEX()` function can reduce the amount of error-checking done by hand.