# ECS 154B – Winter 2014 – Lab 3
# Due by 11:55 PM on Sunday February 16, 2014

## 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 how to use microcode control in a real CPU. 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 (in the companion CD of the book) will be very useful for this lab. You can find it along with some other useful pdfs about multicycle CPUs in the smartsite resources forum for Lab 2. You must implement the CPU control using microcode as discussed in class.

## Details

In this Lab, you will have a singular RAM acting both as instruction and data memory. Just like lab1, you can have a 256 word deep memory. You can logically reserve the first 128 words for instructions and the next 128 for data. This just means that your PC should not go beyond 128th word's address. And 129-256th word can be used as read/write memory. Same as lab1, PC is incremented by 4 so in your instruction memory first instruction will be at 0x00000000, 2nd at 0x00000004, 3rd at 0x00000008 and so on.

You will be given an empty project to start your lab that includes an implementation of the MIPS ALU and a Register File. The project is available in the Resources section under Lab 3 of the course's SmartSite page. Simply download the MIPSMULTICYCLE.circ file. The blocks are described below:

- `Register File`: Same as Lab 2.
- `ALU`: Similar to lab 2, with the addition of support for sll and srl. There is now a Shamt[4..0] input.

When performing shift operations, the `B` input is shifted by the amount specified by `ShAmt`. The `ALUCtl` signals correspond to the operations listed in Table 1. These are the same values from Lab 2 with the addition of the shift operations.

Your CPU must execute the following instructions:

- All required instructions from Lab 2 (add, sub, lw, sw, and, or, nor,beq, slt, j,addi,andi,nori)
- `sll` , `srl`

| ALUCtl | Operation |
|---|---|
| 0010 | add |
| 0110 | sub |
| 0111 | slt |
| 0000 | and |
| 0001 | or |
| 1100 | nor |
| 1010 | sll |
| 1110 | srl |

Table 1: ALUCtl Operations

For this lab, you must use microcode to implement the main control unit. However, you may optimize the memory contents to only work for the required instructions. You must generate the initialization file for your control ROMs by hand. All control signals must come from this control ROM, with the following possible exception - The ALU control unit may use combinational logic similar to lab2. All other control signals must come from a ROM in the main control unit.

## Grading, Testing and Submission

In order to facilitate the grading of your lab, "LABEL" the following signals in your circuit so that the TA can identify any of the following signals in your assignment.

- ALUA, ALUB: The ALU data inputs just before they enter the ALU.
- ALUCtl: The ALU Control signals from Page 316 of the textbook.
- ALUSrc: Control signal of mux at lower ALU input.
- Branch: Control signal indicating a branch.
- Zero: The zero output from the ALU.
- Jump: Control signal indicating a jump.
- MemRead, MemWrite: Data memory control signals.
- MemtoReg: Control signal for mux that selects data written to the register file.
- RegWriteData: The write data input to the register file.

Basically, "LABEL" all the control signals coming out of your Main Control unit, similar to lab 2. You may add pins in order to facilitate debugging, but don't change the existing ones. Include a README file with the following:

- name1, SID
- name2, SID
- any known issues or comments you'd like me to see
- the boolean equations for your various control signals, along with intelligible descriptions of what your variable names are and which equations correspond to what signals in your CPU.

To test your assignment I will run your CPU with the instructions in mipsInstructions.rb file and look at the contents of your registers after the program is finished. Find the files mipsInstructions.rb/minpsInstructions.txt under the resources section of Lab3. If you look at the comments in mipsInstructions.rb it will tell you what the final states of the registers should be. For each register with a correct value at the end of the run you

will receive 1 point. There are a total of 13 points. Note the values in mipsInstructions.rb are written in decimal but the values in the registers are displayed in hex. Implementation: 90Interactive grading: 10

Submit a `tar` file containing your MIPS.circ and a README to the Lab 3 Assignment on SmartSite. Only one partner needs to actually submit the assignment. Just make sure the `tar` file contains the README file with your names so that I know who is working with whom. You can re-submit as many times as you would like. You may use up to TWO slip days per lab. I suggest very strongly that before submission you confirm that what you have in your `tar` file is actually a working MIPS multi cycle CPU.

# Hints

- Test and debug in steps. Start with a subset of the lab requirements, implement it, test it, and then add other requirements. Performing the design and test in steps will ease your debugging. For example, you could implement the Lab 2 instructions, then add the shift 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. Additionally, you will implement instructions not covered in the text, so your lab will have hardware not shown in the text. Try to read Appendix D thoroughly before starting to write the microcode.
- Refer to the lab 3 resources on smartsite to find material on microprogramming, Appendix D and FSM.
- 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.