# CS224 - Fall 2016 - Lab #6 (Version 1: Nov. 16, 3:47 pm)

## MIPS Single-Cycle Datapath and Controller

Dates: Section 3, Wednesday, 30 November 08:40-12:30
Section 1, Wednesday, 30 November 13:40-17:30
Section 4, Thursday, 1 December 13:40-17:30
Section 2, Friday, 2 December 13:40-17:30

**Purpose**: Simulation and implementation of MIPS single cycle datapath and adding some new instructions to it and changing required parts .

**ALL GROUPS**: Dear students as announced in the classroom please bring and drop your preliminary work into the box provided in front of the lab by 5:00 pm Tuesday November 29.

**At the end of the lab before submiting your code for MOSS similarity testing please READ the instructions given here**

**Students who do not follow the instructions will get 0.**

Submit your MIPS codes for similarity testing to the Unilica > Assignment specific for your section. You will upload **one file**: name_surname_section_Lab6.txt created in the relevant parts. Be sure that the file contains exactly and only the codes which are specifically detailed below. Check the specifications! *Even if you didn't finish, or didn't get the MIPS codes working, you must submit your code to the Unilica Assignment for similarity checking.* Your codes will be compared against all the other codes in the class, by the MOSS program, to determine how similar it is (as an indication of plagiarism).

You have to show your work to your TA and upload it by 11:30 in the morning lab and by 4:30 in the afternoon lab. Note that you cannot wait for the last moment to do this. Make sure that you have shown your work to the Lab TA before uploading. If you wait for the last moment and show your work after the deadline time 30 points will be taken off. If you upload before showing your work to the TA you will receive 0 (zero) for the lab work.

**If we suspect that there is cheating we will send the work with the names of the students to the university disciplinary committee.**

## Part 1. Preliminary Work / Preliminary Design Report (30 points)

You have to provide a neat presentation prepared by Word. At the top of the paper on left provide the following information and staple all papers. In this part provide the program listings with proper identification (please make sure that this info is there for proper grading of your work, otherwise some points will be taken off).

**CS224 / Your Section No.**
**Fall 2016**
**Lab No.**
**Your Name**

**Your Preliminary Design Report (PDR) should contain the following 6 items, one per page:**

a. Determine the assembly language equivalent of the machine codes given in the imem module in the "MIPS_Complete_Model.txt". In the given Verilog module for imem, the hex values are the MIPS machine language instructions for a small test program. Disassemble these codes into the equivalent assembly language instructions and give a 3-column table for the program, with one line per instruction, containing its location, machine instruction (in hex) and its assembly language equivalent. [Note: you may disassembly by hand or use a program tool.

b. Register Transfer Level (RTL) expressions for each of the new instructions that you are adding (you can see them below under the heading "Instructions to implement"), including the fetch and the updating of the PC. See RTL examples by Kevin Liston from Berkeley: https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html .

c. Make any additions or changes to the datapath which are needed in order to make the RTLs for the instructions possible.

d. Make a new row in the main control table for each new instruction being added, and if necessary add new columns for any new control signals that are needed (input or output). Be sure to completely fill in the table—all values must be specified. If any changes are needed in the ALU decoder table, give this table in its new form (with new rows, columns, etc). Please specify your changes for new instructions in the table.

e. Write a test program in MIPS assembly language, that will show whether the new instructions are working or not , and that will confirm that all existing old instructions still continue to work. Don't use any pseudo-instructions; use only real MIPS instructions that will be recognized by the new control unit.

f. Write a list of the Verilog modules that will need changes in order to make these new instructions part of the single-cycle MIPS processor's instruction set. For each module in the list, determine the new Verilog model that will be needed in order for the instructions to be added. Give the Verilog code for each module that needs to be changed.

**Instructions to implement**

The Original10 instructions in "MIPS-lite" are add, sub, and, or, slt, lw, sw, beq, addi and j.

Instructions below are not defined in the MIPS instruction set. They don't exist in any MIPS documentation, they are completely new to MIPS. You will create them, according to the definitions below, then implement them.

1. subi: this I-type instruction subtracts, using a sign-extended immediate value. subi $t2, $t7, 4

2. jm: this I-type instruction is a jump, to the address stored in the memory location indicated in the standard way. Example: jm 40($s3)

3. bge: this I-type instruction do what you would expect—branch to the target address, if the condition is met. Otherwise, the branch is not taken. Example: bge $t2, $t7, TopLoop

4. push: this I-type instruction does what you would expect—push a register value onto stack. Example: push $a3 {Note: the assembler automatically puts 29 in the rs field, and 0 into the immediate field, of the machine code instruction.}

## Part 2: Simulation of the MIPS-lite processor (30 points)

a. Complete the Verilog model of single-cycle MIPS by designing a 32-bit ALU module (one is partly specified already in "MIPS_Complete_Model.txt") and save this ALU module by itself in a new file with a meaningful name. Make this file the basis of a new Xilinx project. Now you are going to simulate your written ALU module using a testbench that you will write. To do this, first check your code syntax and then simulate your ALU module. You can use 32-bit ALU in the MIPS-lite datapath when you are sure that it is working correctly in simulation. When you have integrated your working 32-bit ALU into the "MIPS_Complete_Model.txt" file, you are ready to simulate the MIPS-lite single-cycle processor in Xilinx.

b. Make a New Project with a meaningful name for your single-cycle MIPS-lite. Add Sources of all the Verilog modules given in "MIPS_Complete_Model.txt" (modified with your working ALU), and Save everything.

c. Study the small test program loaded into instruction memory (in the imem module) that you dis-assembled in part b) of your Preliminary Design Report. What is the program attempting to do?

d. In this part, you want to simulate your MIPS-lite processor using a testbench. Make a Verilog testbench file with Xillinx to test the program in the imem module(instruction memory). Study the results given in the simulation window. Find each instruction, and understand its values.

e. Now modify the simulation, in order to point out more information. Make changes to the Verilog modules as needed so that the 32-bit values of PC and the Instruction are made to be outputs of the top-level module. Then modify the testbench file to cover thiese modification and display them in the simulation. Finally you should show all of your simulation with the values of PC, instruction, writedata, dataaddr, and the memwrite signal for any instruction. It expects you understand all part of your work and you can explain its Verilog codes and the reasons behind that.

## Part 3: Adding New Instructions: Implementation and Testing (40 points)

a. Implement the modified processor by making the necessary changes to the Verilog modules in part f your Preliminary Design Report. Integrate these all together into a new Verilog model for the MIPS single-cycle processor that does the Original10 instructions plus the new instructions. In part 3 of this lab, you want to make a new project to implement and test your new MIPS single-cycle processor on the BASYS board.

*You should also consider the following points for the implementation:*

To slow down the execution to an observable rate, the clock and reset signals should be hand-pushed, to be under user control. For example, one clock pulse per push and release means one instruction is fetched-decoded-executed. As these inputs need to come from push buttons, they should be debounced and synchronized.
The memwrite output (along with any other control signals that you want to bring out for viewing) can go to a LED, but the low-order bits of writedata (which is RF[rt]) and of dataaddr (which is the ALU result) should go to the 7-segment display, in order to be viewed in a human-understandable way. [Consider why it isn't necessary to see all 32 bits of these busses, just the low-order bits are enough.]

b. Base of last part explanations, create a new top-level Verilog module, to model the system that contains an instantiation of the MIPS computer (in module top), as well as 2 instantiations of pulse_controller, and 1 instantiation of display_controller. Your system should include some hand-pushed signals coming from push buttons, and some anode and cathode outputs going to the 7-segment display unit on the BASYS2 board and memwrite (and possibly other outputs) going to a LED.

c. Make the .ucf file that maps the inputs and outputs of your top-level Verilog model to the inputs (50 Mhz clock and pushbutton switches) and outputs (AN and CA signals to the 7-segment display, and memwrite (plus others?) going to a LED) of the BASYS2 board and its FPGA.

d. Now create a New Project, and implement it on the BASYS2 board, and test it. You should be able to test all of old and new instructions correctly in the hardware.

*Note: the TA will ask questions to you, and you should be able to describe your work and the Verilog codes and reasons behind it.* ***To get the full point of this part you must understand every steps you have done.***

## Part 4. Instruction for submitting your code

In Part 3, you created a new top-level module for your overall system, and modified several Verilog modules for parts of the single-cycle MIPS that needed to change in order to implement the new instructions. Now to combine all the new and modified Verilog codes into a file called name_surname_section_Lab6.txt . This is the file you started in Part 2. Now add to it all the Verilog modules whose code is new or modified in Part 3. DO NOT include any unmodified Verilog modules, only those whose code you changed or created. When you have done this, name_surname_section_Lab.txt.txt will contain 2 Verilog modules from Part 2, and several more Verilog modules from Part 3. Be sure that your file contains exactly and only the codes which are specifically detailed above. Check the specifications!

## Part 5. Cleanup

1) Erase all the files that you created.
2) When applicable put back all the hardware, boards, wires, tools, etc where they came from.
3) Clean up your lab desk, to leave it completely clean and ready for the next group who will come.

## Additional note

Possible lab setting to do the labs
Vivado --> Basys3 --> {System Verilog or Verilog}
xilinx --> {Basys2 or Basys3} --> Verilog