CS224 – Fall 2017 - Lab #4

MIPS Single-Cycle Datapath and Controller

Dates:

Section 1:	Wed, Nov. 15th	13:40-17:30 in EA-Z04
Section 2:	Fri, Nov. 17th	13:40-17:30 in EA-Z04
Section 3:	Wed, Nov. 15th	08:40-12:30 in EA-Z04
Section 4:	Thu, Nov. 16th	13:40-17:30 in EA-Z04

Purpose: In this lab you will use the digital design engineering tools (System Verilog HDL, Xilinx Vivado and FPGA, Digilent BASYS3 development board) to modify the single-cycle MIPS processor. You will expand the instruction set of the "MIPS-lite" processor by adding new instructions to it. To do this, you must first determine the RTL expressions of the new instructions, then modify the datapath and control unit of the MIPS. To implement the new instructions will require you to modify some System Verilog modules in the HDL model of the processor. To test and prove correctness, you will first simulate the microarchitecture, then synthesize it and demonstrate on the BASYS3 board.

Summary

Part 1 (50 points): Preliminary Report/Preliminary Design Report: System Verilog model for Original10 + New instructions (Due date of this part is the same for all).

Part 2 (50 points): Simulation of the MIPS-lite processor and Implementation and Testing of new instructions (25%).

DUE DATE OF PART 1: SAME FOR ALL SECTIONS Dear students please bring and drop your preliminary work into the box provided in front of the lab before 3:59 pm on Tuesday October 14th. No late submission!

LAB WORK SUBMISSION TIMING: You have to show your lab work to your TA by **12:15** in the morning lab and by **17:15** in the afternoon lab. Note that you cannot wait for the last moment to do this. If you wait for the last moment and show your work after the deadline time 20 points will be taken off.

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 (50 points)

Be sure to make a copy of the report, to use during the lab. Your PDR should contain the following 7 items, one per page:

a) Cover page, with university name, department name, and course name and number at the top, "Preliminary Design Report", Lab # (e.g 4), Section #, and your name and ID# in the middle, and the date of your lab at the bottom.

b) **[5 points]** Determine the assembly language equivalent of the machine codes given in the imem module in the "Complete MIPS model.txt" file posted on Unilica for this lab. In the given System Verilog module for imem, the hex values are the MIPS machine language instructions for a small test program. Dis-assemble 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 dis-assembly by hand or use a program tool like the one in Unilica.]

c) **[5 points]** Register Transfer Level (RTL) expressions for each of the new instructions that you are adding (see list below for your section), including the fetch and the updating of the PC.

d) **[5 points]** Make any additions or changes to the datapath which are needed in order to make the RTLs for the instructions possible. The base datapath should be in black, with changes marked in red and other colors (one color per new instruction).

e) **[5 points]** 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). The base table should be in black, with changes marked in red and other colors. {Note: if you need new ALUOp bits to encode new values, you should also give a new version of Table 7.1, showing the new encodings}

f) **[10 points]** 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.

g) **[20 points]** Write a list of the System 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 System Verilog model that will be needed in order for the instructions to be added. Give the System Verilog code for each module that needs to be changed.

Table of instructions to implement-- by section

Section	MIPS instructions
Sec 1	Base: Original10 New: “ble”, “subi”
Sec 2	Base: Original10 New: “nop”, “sw+”
Sec 3	Base: Original10 New: “jalm”, lui
Sec 4	Base: Original10 New: jalr, “push”

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

Instructions in quotes (e.g.”push”) 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.

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

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

nop: this I-type instruction does nothing, changes no values, takes one clock cycle. Except for the opcode (which you need to assign a code to), the I-type instruction field values are irrelevant. Example: nop {Note: an R-type nop exists in real MIPS, it is sll $0, $0, 0. But the nop here is different, so it is “nop” !}

sw+: this I-type instruction does the normal store, as expected, plus an increment (by 4, since it is a word transfer) of the base address in RF[rs]. {Note: these kind of auto-increment instructions are useful when moving through an array of data words.}

jalm: : this I-type instruction is a jump, to the address stored in the memory location indicated in the standard way. But it also puts the return address into the register specified. Example: jalm $t5, 40($s3)

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 and Implementation

Simulation of the MIPS-lite processor (25%)

a) Complete the System Verilog model of single-cycle MIPS by designing a 32-bit ALU module (one is partly specified already in "Complete MIPS 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 Vivado project. In simulation, check its syntax, and then simulate this ALU, using a testbench that you will write. When you are sure that the 32-bit ALU is working correctly in simulation, you can now use it in the MIPS-lite datapath. When you have integrated your working 32-bit ALU into the "Complete MIPS model.txt" file, you are ready to simulate the MIPS-lite single-cycle processor in Xilinx Vivado.

b) Make a New Project, giving it a meaningful name, for your single-cycle MIPS-lite. Do Add Source for the System Verilog modules given in "Complete MIPS 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) Now make a System Verilog testbench file and using Xilinx Vivado, simulate your MIPS-lite processor executing the test program. Study the results given in the simulation window. Find each instruction, and understand its values. Why is writedata undefined for some of the early instructions in the program?

e) Now modify the simulation, in order to show more information. Make changes to the System 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, so that they are displayed in the simulation.

f) When you have studied the simulation results and can explain, for any instruction, the values of PC, instruction, writedata, dataaddr, and the memwrite signal, then call the TA, show your simulation demo and answer questions for grade. The purpose of the questions is to determine your knowlege level and test your ability to explain your demo, the System Verilog code and the reasons behind it, and to see if you can explain what would happen if certain changes were made to it. **To get full points from the Oral Quiz, you must know and understand everything about what you have done.**

Now save the System Verilog code of the testbench module used for your simulation in part f), plus the top-level module file that you used for part f), in their final form. Put these 2 modules together in a text file named FirstName_LastName_Lab4.txt It should contain all the System Verilog codes from these 2 modules, copy-pasted together in one .txt file, but no other modules. In Part 3 *but not now*, you will upload your file to the Unilica Assignment for your section, for similarity checking with MOSS. Each person is only allowed to upload 1 submission the Unilica Assignment, and now is not the time.

Adding New Instructions: Implementation and Testing (25%)

g) Implement the modified processor by making the necessary changes to the System Verilog modules in your Preliminary Design Report, part g). Integrate these all together into a new System Verilog model for the MIPS single-cycle processor that does the Original10 instructions plus the new instructions required for your section.

You should also consider the following: to slow down the execution to an observable rate, the clock signal should be hand-pushed, to be under user control. One clock pulse per push and release means one instruction is fetched-decoded-executed. Similarly the reset signal should be under hand-pushed user control. So these inputs need to come from push buttons, and to 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.]

h) In view of the above, create a new top-level System 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 BASYS3 board and memwrite (and possibly other outputs) going to a LED..

i) Make the constraint file that maps the inputs and outputs of your top-level System Verilog model to the inputs (100 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 BASYS3 board and its FPGA.

j) Now create a New Project, and implement it on the BASYS3 board, and test it. When both of your new instructions are working correctly in hardware, and all 10 of the old instructions are also still working, call the TA and show it for grade. *Note: the TA will ask questions to you, in a single 25-point demo and Oral Quiz,* to determine how much of the 25 points for this part of Part 2 (implementation) is deserved, based on your demo, your knowlege and ability to explain it and the System Verilog code and the reasons behind it, and what would happen if certain changes were made to it. **To get full points from the Oral Quiz, you must know and understand everything about what you have done.**

Part 3. Submit your code for MOSS similarity testing

In Part 2, you created a new top-level module for your overall system, and modified several System Verilog modules for parts of the single-cycle MIPS that needed to change in order to implement the new instructions. Now it is time to combine all the new and modified System Verilog codes into a file called FirstName_LastName_Lab4.txt . This is the file you started in Part 2 (after f, your simulation demo). Now add to it all the Verilog modules whose code is new or modified for j, the implementation demo. DO NOT include any unmodified System Verilog modules, only those whose code you changed or created. When you have done this, FirstName_LastName_Lab4.txt will contain 2 System Verilog modules from Part 2 (simulation), and several more System Verilog modules from Part 2 (implementation).

After demonstrating your working (or partially working) codes to the grader, you should immediately submit your MIPS codes for similarity testing to the Unilica > Assignment specific for your section. You

will upload one file, named **name_surname_SecNo_SysVer.txt,** containing the programs described above in the paragraph. Be sure that the file contains exactly and only the codes which are specifically detailed in Part 2. Check the specifications! *Even if you didn't finish, or didn't get the System Verilog codes working correctly, you must submit your code to the Unilica Assignment for similarity checking.* Failure to submit your codes will result in a lab score of 0. Your codes will be compared against all the other codes in all sections of the course, by the MOSS program, to determine how similar it is, as an indication of plagiarism. **So be sure that the code you submit is code that you actually wrote yourself!** [Warning: DON'T use code provided by another student, or found somewhere on the internet, etc, since MOSS will determine that yours is similar to theirs!] All students must upload their Part 2 code to Unilica > Assignment while the TA watches, at the end of your demo-for-grading time. Submissions made without the TA observing will be deleted, resulting in a lab score of 0.

Part 4. Cleanup

1) After saving any files that you might want to have in the future to your own storage device, erase all the files you created from the computer in the lab.
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.

LAB POLICIES

1. You can do the lab only in your section. Missing your section time and doing in another day is not allowed.
2. Students will earn their own individual lab grade. The questions asked by the TA will have an effect on your individual lab score.
3. Lab score will be reduced to 0 if the code is not submitted for similarity testing, or if it is plagiarized. MOSS-testing will be done, to determine similarity rates. Trivial changes to code will not hide plagiarism from MOSS—the algorithm is quite sophisticated and powerful. Please also note that obviously you should not use any program available on the web, or in a book, etc. since MOSS will find it. The use of the ideas we discussed in the classroom is not a problem.
4. You must be in lab, working on the lab, from the time lab starts until your work is finished and you leave.
5. No cell phone usage during lab.
6. Internet usage is permitted only to lab-related technical sites.
7. For labs that involve hardware for design you will always use the same board provided to you by the lab engineer.