CS224 - Fall 2020 - Lab #4 ((Version 1: November 19, 14:00)

MIPS Single-Cycle Datapath and Controller

Dates:

Section 1: Wednesday, 25 November, 13:30-16:20
Section 2: Friday, 27 November, 13:30-16:20
Section 3: Sunday, 29 November, 8:30-11:20
Section 4: Thusday, 26 November, 13:30-16:20

Submission Date: December 1, Tuesday, 23:59, for all lab days.
(No preliminary work and lab work distinction.
See the submission instructions below)

Physical Lab Administration: No pysical lab: All labs are online. Your TAs will be available during your lab hours. Other times you may reach them by email.

TAs:

Section 1 (Wed): Ege Berkay Gülcan, Zülal Bingöl
Section 2 (Fri): Pouya Ghahramanian, Yusuf Dalva
Section 3 (Sun): Yusuf Dalva, Zülal Bingöl
Section 4 (Thu): Berkay Gülcan, Eren Çalık

TA email Addresses:

Berkay Gülcan: berkay.gulcan@bilkent.edu.tr
Eren Çalık: eren.calik@bilkent.edu.tr
Pouya Ghahramanian: ghahramanian@bilkent.edu.tr
Yusuf Dalva: yusuf.dalva@bilkent.edu.tr
Zülal Bingöl: zulal.bingol@bilkent.edu.tr

Moodle Forum: There is a Moodle Forum available for sharing your problems and suggestions for their solutions. As you do this do not cross the line: Remember that we use MOSS for plagiarism check and panelize when it is detected.

Purpose: In this lab you will use an online digital design tool, EDA Playground (https://www.edaplayground.com/), to modify the single-cycle MIPS processor. Please visit Moodle for a video and FAQ about EDAPlayground. 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. Implementing the new instructions will require you to modify some SystemVerilog modules in the HDL model of the processor which is provided in the same folder as this one in Moodle. To test and prove correctness, you will simulate the microarchitecture on EDAPlayground.

Summary

Part 1 (40 points): SystemVerilog model for Original10 + New instructions assigned to your section,

Part 2 (60 points): Simulation of the MIPS-lite processor and Implementation and Testing of new instructions.

[REMINDER!] In this lab and the next one, we assume SystemVerilog knowledge, since you all took CS223 – Digital Design. If you are not familiar with these, please get used to them as soon as possible.

Part 1. (40 points)

For Part 1 you will prepare a report that contains the following 5 items (Parts b-f). You will name this report as:

StudentID_StudentFirstName_StudentLastName_SecNo_LabNo_REPORT.pdf

a) Note that in this lab you have one piece of work as there is no division like preliminary work and lab work. For all sections the submission date is the same. You have to provide a neat presentation prepared in pdf form. Provide following five lines at the top of your submission for your work (make sure that you include the course no. CS224, important for ABET documentation).

CS224
Lab
Section No.
Your Full Name
Bilkent ID

Make sure that you identify what is what at the beginning of each section by using proper titles.

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 Moodle 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 do dis-assembly by hand or use a program tool.]

c) **[5 points]** Register Transfer Level -Language- (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) **[10 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) **[10 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.

Table of instructions to implement-- by section

Section	MIPS instructions
1	Base: Original10 New: "ble", "subi"
2	Base: Original10 New: "nop", "sw+"
3	Base: Original10 New: "jm", "bge"
4	Base: Original10 New: "jalm", lui

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

Instructions in quotes (e.g. "subi") 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. The **lui instruction**: You know how it works.

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" !}

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

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)

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)

bge, 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: bge $t2, $t7, TopLoop

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.} Example: sw+ $t0, 4($t1)

Part 2: Simulation and Implementation (60 Points)

a) Similar to Part 1.a you have to provide a neat presentation prepared this time in txt form. Provide following five lines at the top of your submission for your work (make sure that you include the course no. CS224, important for ABET documentation).

CS224
Lab
Section No.
Your Full Name
Bilkent ID

Make sure that you identify what is what at the beginning of each program by using proper comments.

b) Complete the SystemVerilog model of single-cycle MIPS by designing a 32-bit ALU module (one is partly specified already in "Complete MIPS model.txt"). 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 EDAPlayground.

c) Make a new Playground, giving it a meaningful name (from the "Add a title to help you find your playground box in the share tab below), for your single-cycle MIPS-lite. Create design files for the SystemVerilog modules given in "Complete MIPS model.txt" (modified with your working ALU), and Save everything (you don't have to use Complete MIPS model.txt, you can write your code if you find it more convenient). Make the necessary changes in order to support newly added instructions.

d) 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? Extend the imem module so that it will also include newly added instructions as well.

e) Now make a SystemVerilog testbench file on EDAPlayground that uses $display() and/or $monitor() system tasks, simulate your MIPS-lite processor executing the test program. Study the results given in the Log window. Find each instruction, and understand its values. Why is writedata undefined for some of the early instructions in the program?

f) Now modify the simulation, in order to show more information. Make changes to the SystemVerilog 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.

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 SystemVerilog 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 SystemVerilog codes into a file called **StudentID_StudentFirstName_StudentLastName_SecNo_LabNo_LAB.txt.** Add the SystemVerilog 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. Also, add all the SystemVerilog modules whose code is new or

modified from “Complete MIPS model.txt”. DO NOT include any unmodified SystemVerilog modules, only those whose code you changed or created.

Submit your MIPS codes for similarity testing to the Moodle > CS224 Lab #4 Section X specific for your section. Remember that you will upload two files, named:

StudentID_StudentFirstName_StudentLastName_SecNo_LabNo_LAB.txt,
StudentID_StudentFirstName_StudentLastName_SecNo_LabNo_REPORT.pdf

containing the programs described above in the paragraph and the report you have prepared in Part 1. Be sure that the first 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 SystemVerilog codes working correctly, you must submit your code to the Moodle 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!]. We use MOSS testing only for code submissions.