CS224 - Fall 2024 - Lab #4 (Version 2: 1 November 2024; 5:43 pm)

MIPS Single-Cycle Datapath and Controller

Dates: (TAs - Tutor)
Section 1: Wed, 13 Nov, 13:30-17:20 in EA-Z04 (TA: Kadri, Soheil; Tutor: Deniz: 15:30-17:20)
Section 2: Thu, 14 Nov, 13:30-17:20 in EA-Z04 (TA: Onur, Soheil; Tutor: : Deniz: 13:30-15:20)
Section 3: Thu, 14 Nov, 8:30-12:20 in EA-Z04 (TA: Onur, Pouya; Tutor: Mert)
Section 4: Fri, 15 Nov, Fri 13:30-17:20 in EA-Z04 (TA: Berkan, Kadri; Tutor: Umut)

TA Full Name (email address: @bilkent.edu.tr)
Berkan Şahin (berkan.sahin)
M. Kadri Gofralılar (kadri.gofralilar)
Onur Yıldırım (o.yildirim)
Pouya Ghahramanian (ghahramanian)
Sepehr Bakhshi (sepehr.bakhshi) (Coord.)
Soheil Abadifard (soheil.abadifard)

Tutor Full Name (email address: @ug.bilkent.edu.tr)
Deniz Çatakoğlu (deniz.catakoglu)
Mert Emre Yamalı (emre.yamali)
Umut Başar Demir (basar.demir)

Purpose: In this lab you will use the digital design engineering tools (SystemVerilog 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. Implementing the new instructions will require you to modify some SystemVerilog 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
Preliminary Work: 50 points
1. **Part 1**: Preliminary Report/Preliminary Design Report: SystemVerilog model for Original10 + New instructions (Due date of this part is the same for all).

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

[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.

Important Notes for All Labs About Attendance, Performing and Presenting the Work

You are obliged to read this document word by word and are responsible for the mistakes you make by not following the rules.

1. Not attending to the lab means 0 out of 100 for that lab. If you attend the lab but do not submit the preliminary part you will lose only the points for the preliminary part.
2. Try to complete the lab part at home before coming to the lab. Make sure that you show your work to your TAs and answer their questions to show that you know what you are doing before uploading your lab work and follow the instructions of your TAs.
3. In all labs if you are not told you may assume that inputs are correct.
4. In all labs when needed you have to provide a simple user interface for inputs and outputs.
5. Presentation of your work
   You have to provide a neat presentation prepared in TXT form for programs, and PDF form for reports. Your programs must be easy to understand and well structured.
   Provide following six lines at the top of your submission for preliminary and lab work (make sure that you include the course no. CS224, important for ABET documentation).

   CS224
   Lab No.
   Section No.
   Your Full Name
   Bilkent ID
   Date

   Please also make sure that your work is identifiable: In terms of which program corresponds to which part of the lab.
6. **If we suspect that there is cheating we will send the work with the names of the students to the university disciplinary committee. You can experiment with ChatGPT for learning; however, you cannot use/modify the code generated by it. Such an act is classified as plagiarism. Note that MOSS is capable of detecting ChatGPT code. Furthermore remember that, the code you use from ChatGPT can also be used by another student in the course. Make sure that the code you submit is really yours and has been internalized.**

DUE DATE PRELIMINARY WORK: SAME FOR ALL SECTIONS

NO late submission will be accepted. Please do not try to break this rule and any other rule we set.

a. Please upload your programs of preliminary work to Moodle by **13:30 on Wednesday, 13 November, 2024.**

b. Please note that the submission closes sharp at 13:30 and no late submissions will be accepted. You can make resubmissions so do not wait for the last moment. Submit your work earlier and change your submitted work if necessary. Note that only the last submission will be graded.

c. Please familiarize yourself with the Moodle course interface, find the submission entry early, and avoid sending an email like "I cannot see the submission interface." (As of now it is not yet opened.)

d. Do not send your work by email attachment they will not be processed. They have to be in the Moodle system to be processed.

e. Use filename **StudentID_FirstName_LastName_SecNo_PRELIM_LabNo.pdf** Only a PDF file is accepted. Any other form of submission (Word file etc.) receives 0 (zero).

DUE DATE PART LAB WORK: (different for each section) YOUR LAB DAY

a. You have to demonstrate your lab work to your TA for grading. Do this by **12:00** in the morning lab and by **17:00** in the afternoon lab. Your TAs may give further instructions on this. If you wait idly and show your work last minute, your work may not be graded.

b. At the conclusion of the demo for getting your grade, you will **upload your Lab Work** to the Moodle Assignment, for similarity testing by MOSS. See below for the details of lab work submission.

c. Try to finish all of your lab work before coming to the lab, but make sure that you upload your work after making sure that it is analyzed by your TA and/or you are given the permission by your TA to upload.

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump
R-type	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
addi	001000	1	0	1	0	0	0	00	0
j	000010	0	X	X	X	0	X	XX	1

Table 1: Main Decoder for Original10

ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)

Table 2: ALU Decoder for Original10

Part 1. Preliminary Work (50 points)

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

a) The first page must start with Course No. etc. as indicated above. Prepare a cover page for your preliminary report PDF containing the following five lines (make sure you include the course no. CS224, important for ABET documentation).
CS224,
Lab No.
Section No.
Your Full Name
Bilkent ID

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 SystemVerilog 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 disassemble by hand or use a program tool.]

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). **Make your changes on "Final Datapath.png" file.**

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). **Make your changes on Table 1: Main Decoder for Original10.** 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 2, 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 SystemVerilog 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 SystemVerilog model that will be needed in order for the instructions to be added. Give the SystemVerilog code for each module that needs to be changed.

Table of instructions to implement-- by section

Section	MIPS instructions
Sec 1	Base: Original10 New: “bcon”, “xnori”
Sec 2	Base: Original10 New: “jaladd”, “sw+”
Sec 3	Base: Original10 New: “bncon”, “spc”
Sec 4	Base: Original10 New: “jalsub”, “ror”

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

The instructions in quotes (e.g. “bcon”) are not implemented in the Original10 machine given to you. These instructions are not defined anywhere else, and are completely new to MIPS. In any case, you will use the definitions given below to create and implement these instructions. You may check the “Samples of new instructions.txt” file to see example usages of new instructions.

bcon (Branch if consecutive): This I-type instruction branches to the target address only if RF[rt] has the value of a consecutive address value held in RF[rs] (When the value in RF[rt] is RF[rs] + 4, and word-aligned, i.e., divisible by 4). Otherwise, the branch is not taken. **Usage:** bcon rs, rt, label. **Example:** bcon $a1, $a0, loopStart (when $a1 is 8 and $a0 is 12, the branch is taken).

xnori: This I-type instruction zero-extends the given immediate then bitwise XNORs it with RF[rs] (XNOR is the complement of the XOR operation, returns 1 if either both bits are zero or both bits are one) The result is written into RF[rt]. **Example:** xori $t0, $t1, 0x0040.

jaladd, jalsub (Jump and link with addition/subtraction): These R-type instructions jump to the locations calculated by addition or subtraction of the registers rs and rt. While jumping the 31st register ($ra) is changed to PC + 4 like a normal jal instruction. **Usage:** jaladd/jalsub rs, rt (You can assume rd and shamt are zero in this instruction). **Examples:** jalsub $a2, $a1 (when $a2 is 0x20 and $a1 is 0x04, $ra will be PC + 4 and PC will be **0x18**), jaladd $a3, $a0 (when $a3 is 0x20 and $a1 is 0x04, $ra will be PC + 4 and PC will be **0x24**)

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)

bncon (Branch if non-consecutive): This I-type instruction branches to the target address only if RF[rt] does not have the value of a consecutive address value held in RF[rs] (When the value in RF[rt] isn’t

RF[rs] + 4 OR NOT word-aligned, i.e., divisible by 4). Otherwise, the branch is not taken. Usage: bncon rs, rt, label. Example: bcon $a2, $a1, loopEnd (when $a2 is 8 and $a1 is **not** 12, branch is taken).

spc: This I-type instruction stores the value in the program counter (PC) to the memory location indicated in the standard way. Example: spc 40($s0). **Note:** You can assume that the rt field of the instruction is 0b00000 while assembling test programs for the spc instruction.

ror: This R-type instruction right rotates RF[rs] by the amount specified by the shamt field of the instruction. Right rotation is similar to shifting a binary value right, except the bits that would be lost during shifting are placed to the left. The result is stored in RF[rd]. **Example:** ror $s0, $s1, 5. **Note:** You can assume that the rt field of the instruction is 0b00000 while assembling the test programs in Part 3 of the lab.

You may check the file "Samples of new instructions.txt" to see example usages of new instructions. You may also use these samples to test your implementations. However, you must write your own test for Part 1.f.

Part 2. Simulation and Implementation

Simulation of the MIPS-lite processor (25%)

a) 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") 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 SystemVerilog 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 disassembled in part b) of your Preliminary Design Report. What is the program attempting to do?

d) Now make a SystemVerilog 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 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.

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 knowledge level and test your ability to explain your demo, the SystemVerilog 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 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. Put these 2 modules together in a text file named **StudentID_FirstName_LastName_SecNo_LAB_LabNo.txt**. It should contain all the SystemVerilog codes from these 2 modules, copy-pasted together in one .txt file, but no other modules. In Part 3 , you will upload your file to the Moodle Assignment for your section, for similarity checking with MOSS.

Adding New Instructions: Implementation and Testing (25%)

g) Implement the modified processor by making the necessary changes to the SystemVerilog modules in your Preliminary Design Report, part g). Integrate these all together into a new SystemVerilog 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 SystemVerilog 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 SystemVerilog model to the inputs (100 MHz clock and pushbutton switches) and outputs (AN, SEG and DP signals to the 7-segment display, and memwrite going to a LED) of the BASYS3 board and its FPGA. **Hint:** You can find a base constraint file for the BASYS3 board here: https://raw.githubusercontent.com/Digilent/digilent-xdc/master/Basys-3-Master.xdc You can modify this file to create your constraint file instead of writing it from scratch [You can download the file by right clicking and selecting "Save Page As…" or the equivalent option in your browser].

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 knowledge and ability to explain it and the SystemVerilog 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 Lab Work for MOSS Similarity Testing

1. Submit your Lab Work SystemVerilog codes for similarity testing to Moodle.
2. You will upload one file. Use filename **StudentID_FirstName_LastName_SecNo_LAB_LabNo.txt**
3. Only a NOTEPAD FILE (TXT file) is accepted. No TXT file upload means you get 0 from the lab. The Mac program TextEdit saves as RTF by default, which is not accepted. Mac users should select plain text from the save dialog if they are using this program. Please note that we have several students and efficiency is important.
4. *Even if you didn't finish, or didn't get the SystemVerilog codes working, you must submit your code to the Moodle Assignment for similarity checking.*
5. Comparison of your programs with other students' programs: The MOSS plagiarism detection tool determines how similar they are to other programs (as an indication of plagiarism). So be sure that the code you submit is code that you actually wrote yourself! You are not allowed to use web resources that solve the assigned programs.
6. The effectiveness of MOSS for chatbot code is quite good, with a detection rate of much higher than 50%. (When the answer is provided by a chatbot.)

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. The questions asked by the TA will have an effect on your 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 works for ChatGPT code too. 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.