

CS224 - Spring 2023 - Lab #4

MIPS Single-Cycle Datapath and Controller

Dates (TAs) - (Tutor(s)):

Section 1: Mon, 8 May, 8:30-12:20 in EA-Z04 (Deniz, Ece) - (Berkan)
Section 2: Wed, 3 May, 13:30-17:20 in EA-Z04 (Sepehr, Pouya) - (Berkan, Ege)
Section 3: Tue, 2 May, , 13:30-17:20 in EA-Z04 (Onur, Utku) - (Talay)
Section 4: Fri, 5 May, Fri 8:30-12:20 in EA-Z04 (Sanaz, Soheil) - (Eren)
Section 5: Wed, 3 May Feb, 8:30-12:20 in EA-Z04 (Navid, Onur) - (Yarkın)
Section 6: Fri, 5 May, 13:30-17:20 in EA-Z04 (Sanaz, Soheil) - (Yarkın)

TA Name (email address: @bilkent.edu.tr): Section (hours)

Ece Kunduracıoğlu (e.kunduracioglu@): Section 1 (Mon 08:30-10:30)
Deniz Uzel (deniz.uzel@): : Section 1 (Mon 08:30-10:30)
Navid Ghamari (navid.ghamari@): Section 2 (Wed 13:30-15:30)
Onur Yıldırım (o.yildirim@): Section 3 (Tue 13:30-17:30)
Pouya Ghahramanian (ghahramanian@): Section 2 (Wed 13:30-17:30)
Sanaz Gheibuni (sanaz.gheibuni@): Section 4 (Fri 08:30-12:30), Section 6 (Fri 13:30-15:30)
Sepehr Bakhshi (sepehr.bakhshi@): Section 2 (Wed 13:30-17:30)
Soheil Abadifard (soheil.abadifard@): Section 4 (Fri 08:30-12:30), Section 6 (Fri 13:30-15:30)
Utku Gülgeç (utku.gulgec@): Section 3 (Tue 13:30-17:30)

Tutor Name (email address: @ug.bilkent.edu.tr): Section (hours)

Berkan Şahin (berkan.sahin@): Section 1 (Mon 08:30-10:20) & Section 2 (Wed 13:30-15:20)
Hasan Ege Tunç (hasan.tunc@): Section 2 (Wed 13:30-17:20)
Atak Talay Yücel (talay.yucel@): Section 3 (Tue 13:30-17:20)
Mehmet Eren Balasar (eren.balasar@): Section 4 (Fri 08:30-12:20)
Hasan Yarkın Kurt (yarkin.kurt@): Section 5 (Wed 08:30-10:20) & Section 6 (Fri 13:30-15:20)

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. Implementing 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): SystemVerilog model for Original10, Presenting RTL instructions, datapath and controller changes for the new instructions assigned to your section,

Part 2 (50 points): Simulation of the MIPS-lite processor and Implementation and Testing of the new instructions assigned to your section.

[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

Note: Try, study, and aim to complete lab part at home before coming to the lab.

DUE DATE OF PART 1 & 2 (PRELIMINARY WORK): May 2, Tuesday, 12:30 PM (SAME FOR ALL SECTIONS)

No late submission will be accepted.

- a. Please upload your programs of preliminary work to Moodle by **12:30 PM on Tuesday, May 2nd**.
- b. Please note that the Moodle submission closes sharp at 12:30 PM and no late submissions will be accepted. You can make resubmissions before the system closes, so do not wait the last moment. Submit your work earlier and change your submitted work if necessary. Note that only the last submission will be graded.
- c. 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. For the report, use the file name **StudentID_FirstName_LastName_SecNo_PRE-LIM_LabNo_Report.pdf** Only a PDF file is accepted. Any other form of submission receives 0 (zero).

DUE DATE PART 3 (LAB WORK): (different for each section) YOUR LAB

1. 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.
2. 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.
3. 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.

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 & 2. Preliminary Work (50 points)

For the preliminary work, you need to prepare a PDF file containing your answers to Part 1.a, 1.d, 1.e, 1.f and Part 2.

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

Important Notes

1. **Assume that all Inputs are Correct.** In all lab works, if it is not explicitly stated, you may assume that program inputs are correct.

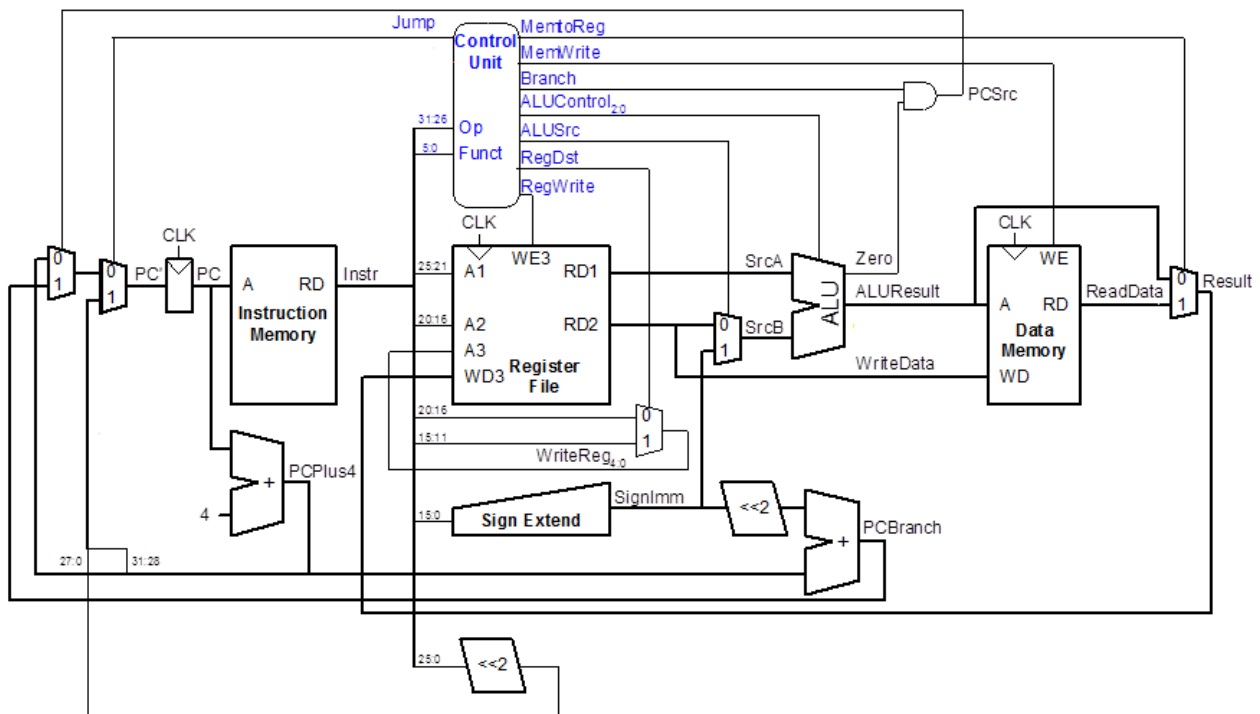


Figure 1: Datapath for Original10

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. Setup and Test – Original10 (20 points)

- a) **[3 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. 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.]
- b) Make a new Vivado Project, giving it a meaningful name for your single-cycle MIPS-lite. Import the SystemVerilog modules in “Complete MIPS model.txt” using Add Source and save everything (you don’t have to use “Complete MIPS model.txt”, you can write your code if you find it more convenient). This file contains implementation of Original10 instructions in “MIPS-lite” which are the following: **add, sub, and, or, slt, lw, sw, beq, addi, j**.
- c) Study the code and convince yourselves that the *datapath* module exactly corresponds to Figure 1 and *controller* module (including *maindec* and *aludec* modules) to Table 1 and Table 2. In the decoder tables, X means it is a “don’t care”. That is, it does not matter if it is 0 or 1.
- d) **[7 Points]** Now make a SystemVerilog testbench file and using Xilinx Vivado, simulate your MIPS-lite processor executing the test program. In your testbench, do not forget to reset your processor at the beginning. Do not change the instructions in the `imem` module, use it as is. Study the generated waveform. Find each instruction and understand its values. **Take a screenshot of the waveform and paste it to the report.** Do not include the reset time in your screenshot. Your waveform should be similar to the below image. Make sure that your output shows the exact same variables.

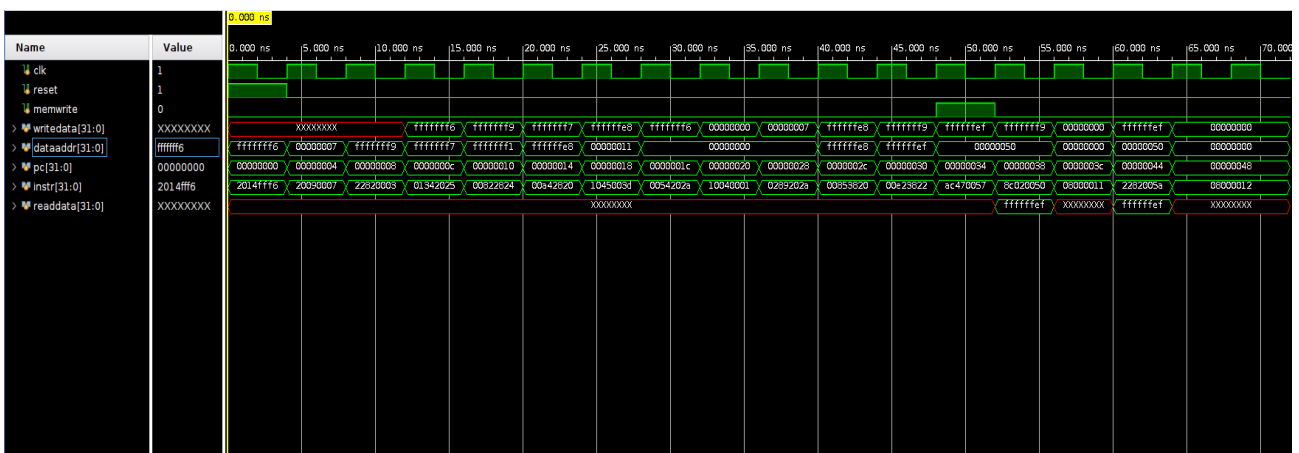


Figure 2: Example Waveform

- e) **[5 Points]** Make some observations on the waveform that you generated. These questions assume that you have used `imem` module as is without modifying any instruction there.
- In an R-type instruction what does `writedata` correspond to?
 - Why is `writedata` undefined for some of the early instructions in the program?
 - In which instructions `memwrite` becomes 1?
 - Why is `dataaddr` 0xffffffe8 when PC is 0x14?
 - In the example program, when is the output of `readdata` defined and why?
- f) **[5 Points]** Modify the ALU module in “Complete MIPS model.txt” so that when `alucont` = 011, ALU computes the result of $a \wedge b$ (a XOR b). Put the modified ALU module in your report.

Part 2. Extend Original10 (30 points)

Section	MIPS instructions
1	Base: Original10 New: “lui” + “xori”
2	Base: Original10 New: “lb” + “rol”
3	Base: Original10 New: “spc” + “rol”
4	Base: Original10 New: “lb” + “lui”
5	Base: Original10 New: “jal” + “lh”
6	Base: Original10 New: “jr” + “xori”

Table 3: Section-wise distribution of the instructions

The instructions in quotes (e.g. “rol”) are not implemented in the Original10 machine given to you. Some of these instructions are defined in the MIPS instruction set, while some 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. **Note:** For the instructions that have the same mnemonic as existing MIPS instructions (lb, lui etc.) you are not required to use the exact opcode and funct values given in the MIPS documentation. You can use any opcode and funct value as long as it does not collide with the 10 instructions implemented in the CPU given to you.

lb: This I-Type instruction is similar to lw, except it loads a single byte from the data memory. The least significant byte of RF[rd] is set to the loaded byte, and the rest of the bits of RF[rd] are set to 0. Example: lb \$t0, 41(\$s3).

lh: This I-Type instruction is similar to lw, except it loads a half-word (16 bits) from the data memory. The least significant two bytes of RF[rd] are set to the loaded half-word, and the rest of the bits are set to 0. Example: lh \$t0, 42(\$s1).

rol: This R-type instruction left rotates RF[rs] by the amount specified by the shamt field of the instruction. Left rotation is similar to shifting a binary value left, except the bits that would be lost during shifting are placed to the right. The result is stored in RF[rd]. Example: rol \$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.

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.

jr: This R-type instruction jumps to the location specified in the RF[rs] register. Example: jr \$ra. **Note:** You can assume that the rt and rd fields of the jr instruction are 0b00000.

lui: This I-type instruction loads the value in the imm field to the most significant 2 bytes of RF[rt]. The two least significant bytes are set to 0. Example: lui \$t0, 0x1001

jal: This J-type instruction calculates the jump address identically to the **j** instruction, but it also stores PC + 4 in the \$ra register. Example: jal 40.

xori: This I-type instruction zero-extends the given immediate then bitwise XORs it with RF[rs]. The result is written into RF[rt]. Example: xori \$t0, \$t1, 0x0040.

- a) **[3 Points]** Register Transfer Level -Language- (RTL) expressions for the new instructions that you are adding (see Table 3), including the fetch and the updating of the PC.
- b) **[15 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 for marked in **red and other colors (one color per new instruction)**. **Make your changes on “Final Datapath.png” file.**
- c) **[12 Points]** Make a new row in the main decoder table for each new instruction being added, and if necessary, add new columns for any new control signals that are needed (input or output). You can use any opcode value you want unless it conflicts with the existing opcodes in the Table 1. Be sure to completely fill in the table—all values must be specified. **Make your changes on Table 1: Main Decoder for Original10.** If any changes are needed in the ALU decoder table (Table 2), 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 2, showing the new encoding}

Part 3. Lab Work (50 points)

Simulation of the MIPS-lite processor (25%)

In this section, using your design in Part 2, you will extend the given MIPS processor with your new instructions (see Table 3).

a) Implement the modified processor by making the necessary changes to the SystemVerilog modules to support your new instructions. 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.

b) Then, using the testbench you have already written, simulate and test your extended MIPS processor. Write test codes for the new instructions you implemented, and verify that the new instructions, as well as the Original10, works correctly. **Hint:** You can extend the imem module for your tests.

When you have studied the simulation results and can explain, for any instruction, the values of PC, instruction, writedata, dataaddr, readdata 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.**

Implementation and Testing with BASYS3 (25%)

In previous sections, you have designed and implemented an extension for MIPS-lite processor and simulated your design. In this section, by programming your implementation into BASYS3 FPGA, you will witness that your code represents a real working processor.

a) First, you should consider the following: BASYS3 has a default clock with 100 MHz frequency. So, if you execute your test code with the default clock, the changes will not be observable. 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 two inputs of the top module need to come from push buttons, and to be debounced and synchronized.

The memwrite output of the top module (along with any other control signals that you want to bring out for viewing) can go to a LED, but the other 32-bit outputs should go to the 7-segment display, in order to be viewed in a human-understandable way. As 7-segment is not large enough to display all these information, you should output PC and dataaddr and convert the other 32-bit outputs to internal signals. Then, you should send the low-order bits of PC and dataaddr to 7-segment display. [Consider why it isn't necessary to see all 32 bits of these buses, just the low-order bits are enough.]

In view of the above, create a new top-level SystemVerilog module, to model the system that contains an instance of the MIPS computer (in module top), as well as 2 instances of pulse_controller module (for hand-pushed clock and reset signals), and 1 instance of display_controller module (to display values in 7-segment display). You can find these modules under the lab folder. **Hint:** You will find most of the information you need in header comments of the modules.

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.

b) 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].

c) Now program your implementation on the BASYS3 board, and test it. Make sure that the values displayed on BASYS3 are the same as the ones you obtained during simulations. 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 3 (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.**

Aim to complete lab part at home before coming to the lab.

Part 4. Submit Your Code for MOSS Similarity Testing

1. Submit your Lab Work SystemVerilog codes for similarity testing to Moodle.
2. You will upload one file. Use the file name **StudentID_FirstName_Last-Name_SecNo_LAB_LabNo.txt**
3. Only a NOTEPAD FILE (TXT file) is accepted. No TXT file upload means you get 0 from the lab. 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. 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). So be sure that the code you submit is code that you actually wrote yourself!

Part 5. 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.

Part 6. Lab Policies (Reminder)

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. 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 the lab, working on the lab, from the time lab starts until your work is finished and you leave.
5. No cell phone usage during the 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.