CS224 - Fall 2022 - Lab #5 (v1, Nov 21, 18:34)
Implementing the MIPS Processor with Pipelined Microarchitecture

Dates:
Section 1: Wed, 30 Nov, 13:30-17:20 in EA-Z04
Section 2: Thu, 1 Dec, 13:30-17:20 in EA-Z04
Section 3: Thu, 1 Dec, 8:30-12:20 in EA-Z04
Section 4: Fri, 2 Dec, 13:30-17:20 in EA-Z04

TA name (x No of labs): email address
Kenan Çağrı Hırlak (x2): cagri.hirlak@bilkent.edu.tr
Pouya Ghahramanian (x2): ghahramanian@bilkent.edu.tr
Sepehr Bakhshi (x2): sepehr.bakhshi@bilkent.edu.tr
Soheil Abadifard (x1): soheil.abadifard@bilkent.edu.tr

TAs; Tutor:
Section 1: Pouya Ghharamanian, Sepehr Bakhshi
Section 2: Kenan Çağrı Hırlak
Section 3: Pouya Ghharamanian, Sepehr Bakhshi
Section 4: Kenan Çağrı Hırlak, Soheil Abadifard

Tutor: Yarkın Kurt (x1): yarkin.kurt@ug.bilkent.edu.tr, Lab Section 3 (Thu morning)
Tutor: Efe Yakar (x2): efe.yakar@ug.bilkent.edu.tr, Lab Section 2 (Thu afternoon),
Section 4 (Fri afternoon 2 hours, may change based on his course meetings)

Recitation: For all sections, Monday 17:30-19:30 (by Yarkın Kurt), Place: EA-Z04

Purpose: In this lab you will implement and test a pipelined MIPS processor using the digital design engineering tools (System Verilog HDL, Xilinx Vivado and FPGA, Digilent BASYS3 development board). To do this, you will need to add pipeline registers, forwarding MUXes, a hazard unit, etc to your datapath, and of course make the control pipelined as well. You will be provided a skeleton System Verilog code for the Pipelined MIPS processor and fill the necessary parts to make it work. Then, you will synthesize them and demonstrate on the BASYS3 board.

Summary
Part 1 (50 points): Preliminary Work: Design Report with Pipeline hazards evaluation and preparing test programs in MIPS.
Part 2 (50 Points): Lab Work: Implementation and simulation of the MIPS-lite pipelined processor.

[REMINDER!] In this lab, 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 (PRELIMINARY WORK): SAME FOR ALL SECTIONS
No late submission will be accepted.

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

b. Please note that the Moodle submission closes sharp at 13:30 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.

d . You will need to submit a report file (in pdf).

e. Report filename: **StudentID_FirstName_LastName_SecNo_PRELIM_LabNo_Report.pdf.** Only a PDF (pdf file) is accepted. Any other form of submission receives 0 (zero).

DUE DATE PART 2 (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 Part 2** to the Moodle Assignment, for similarity testing by MOSS. See Part 6 below for details.

c. Aim to finish all of your lab work before coming to the lab, but make sure that you upload your work after making sure that your work 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.

Provide following five 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

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

Part 1. Preliminary Work (50 points)

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.

In this part you will prepare a Design Report in PDF format named as:

StudentID_StudentFirstName_StudentLastName_SecNo_LabNo_REPORT.pdf

At the end of this lab, you will have implemented the pipelined MIPS architecture that can be seen in the file that is provided as **PipelineDatapath.PNG** and extended it with *sracc* instruction.

> **sracc:** this R-type instruction shifts the bits of RF[rs] to the right by a number of positions given in RF[rt]. Then, it increments RF[rd] by this value. Example: sracc $t0, $t1, $t2. **Hint:** To implement sracc, you might add a new read port to RF.

Notice that in the given pipeline structure the branch resolution is done in the Decode stage. Note also that there is no jump instruction implemented as well. Your Design Report should contain the following items:

a) Cover page, with university name, department name, and course name and number at the top, "Design Report", Lab # (e.g. 5), Section #, and your name and ID# in the middle, and the date of your lab at the bottom.

b) **[10 points]** Give the RTL expression for sracc. The list of all hazards that can occur in this pipeline. For each hazard, give its type (data or control), its specific name ("compute-use" "load-use", "load-store", "branch" etc.), the pipeline stages that are affected.

c) **[10 points]** For each hazard, give the solution (forwarding, stalling, flushing, combination of these), and explanation of what, when, how

d) **[10 points]** Give the logic equations for each signal output by the hazard unit, as a function of the input signals that come to the hazard unit. This hazard unit should handle all the data and control hazards that can occur in your pipeline (listed in b) so that your pipelined processor computes correctly

e) **[20 points]** Think about the implementation of *sracc* instruction in pipelined MIPS processor. Write down all hazards that *sracc* might cause, and their solutions. If some of these hazards are similar to the ones you listed in Part 1-b, you can refer to them. Then, for each hazard you listed, write a small test program in MIPS assembly that will show whether the pipelined processor is working even in the presence of the hazard. You should also write a test program with no hazards. The tests should verify that *sracc* works correctly under any circumstances.

Part 2. Lab Work (50 points)

For the lab work, you only need to submit resulting code file in the txt format.

Simulation of the pipelined MIPS-lite processor (35%)

a) **[15 points]** You are given a skeleton SystemVerilog code for your pipelined MIPS processor in the file **PipelinedMIPSProcessorToFill.txt**. The places in the code that needs to be modified are shown with comment blocks above them. Fill them to implement the Pipelined Processor. You don't need to follow the skeleton code point by point. If you think your design is better, you are welcome to try it in your code, as long as your version of the code works, too. Note that in the skeleton code register file is initialized to 0 on reset. That effectively means all register values are set to 0 at the beginning.

Then, write a SystemVerilog testbench file to simulate and test your pipelined MIPS processor. You may assemble and use your own tests or use the test codes given in the **SampleTestsForHazards.txt** file. Verify that your pipelined MIPS processor works correctly, even in the presence of hazards.

If you are writing your own tests, you can use the student-written assembler tool available online to help you quickly implement your test programs (https://github.com/bilkentCraps/mips). Remember that the goal of testing is to verify that all the instructions are fully working, and that all the instructions are working even in the presence of hazards.

b) **[20 points]** Now you can extend your processor with *sracc* instruction. Implement *sracc* instruction on the pipelined MIPS processor that you have completed in Part 2-a. You may utilize your *sracc* design from Lab4. You are free to modify your previous design or come up with a completely new design. Be careful about the new hazards that *sracc* introduces. You may or may not need to modify the Hazard Unit. **Hint:** Try to design and implement *sracc* in a way that it does not introduce a lot of new hazards.

Simulate and test your extended processor using the codes you have written in Part 1-e. Verify that the *sracc* instruction, as well as all the original instructions, works correctly, even in the presence of hazards.

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

Implementation and Testing with BASYS3 (15%)

In previous sections, you have designed and implemented an extension for pipelined 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 and regwrite outputs of the top module (along with any other control signals that you want to bring out for viewing) can go to LEDs, 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 32-bit signals, you should only send the low-order bits of PC and dataadr to 7-segment display. [Consider why it isn't necessary to see all 32 bits of these busses, just the low-order bits are enough.]

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 module (for hand-pushed clock and reset signals), and 1 instantiation 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 regwrite (and possibly other outputs) going to LEDs.

b) 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, SEG and DP signals to the 7-segment display, and memwrite and regwrite going to LEDs) of the BASYS3 board and its FPGA.

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 *sracc* instruction is working correctly in hardware, and all 10 of the old instructions are also still working, call the TA and show it for grade. The purpose of the questions is to determine your knowledge 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.**

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

Part 6. Submit Your Code for MOSS Similarity Testing

1. Submit your Lab Work MIPS 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. Please note that we have several students and efficiency is important.
4. *Even if you didn't finish, or didn't get the MIPS 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 7. 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 8. 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. As indicated earlier: Attendance is mandatory and the preliminary work is graded only if you submit your lab work with the observation/permission of your TA.
3. The questions asked by the TA will have an effect on your lab score.
4. 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.
5. You must be in lab, working on the lab, from the time lab starts until your work is finished and you leave.