**CS224 Lab 4**

**Implementing the MIPS Processor with Pipelined Microarchitecture**

Dates: Section 1 Monday 28 March 08:40-12:30

Section 2 Friday 25 March 13:40-17:30

Section 3 Tuesday 22 March 08:40-12:30

In this lab you will convert the single-cycle MIPS processor into a pipelined processor, using the digital design engineering tools (Verilog HDL, Xilinx ISE and FPGA, Digilent BASYS2 development board) to implement and test the pipelined MIPS. 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. To implement these changes, converting the single-cycle machine into a pipelined version will require you to modify some Verilog modules in the HDL model of the processor, then synthesize them and demonstrate on the BASYS2 board.

**Preparatory work**

Make multiple printouts of the datapath and 2 tables in the “MIPS datapath and control tables.doc” file. Now you have the datapath and the control tables (for main decoder and ALU decoder) to add new instructions and pipelining to the single-cycle MIPS processor.

**Part 1: Preliminary Design Report (30%)**

For this lab, the Preliminary Design Report (PDR) is due at the same time and place for all 3 sections. *You must submit your PDR by 12:30pm March 22nd to the EA-Z04 lab room*. Your PDR will be graded and count 30% of the score for Lab 4. Be sure to make a copy of the report, for you to use during the lab time for your section. 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) MIPS processor datapath drawing for the 10-instruction “mini-MIPS” machine that does the Original10 instructions, modified to include the new (or modified) hardware and data paths needed to make it pipelined. The branch decision should be made in the Execute stage. The base datapath should be in black, with changes marked in red.

c) The list of all hazards that can occur in your pipeline. For each hazard, give its type (data or control), its specific name (“compute-use” “load-use”, “load-store” “J-type jump”, “branch” etc), the pipeline stages that are affected, the solution (forwarding, stalling, flushing, combination of these), and explanation of what, when, how.

d) 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 c) so that your pipelined processor computes correctly.

e) Make a list of the Verilog modules (not files, modules) that will need changes in order to make the MIPS processor a pipelined microarchitecture. Any new modules, such as you will need for the hazard unit, should also be listed. For each module, indicate if it is new, or changed.

f) For each module in the part e) list, give the Verilog code.

g) Write some test programs, in MIPS assembly, that will show whether the pipelined processor is working or not. Each of your test programs should be designed to catch problems, if there are any, in the execution of MIPS instructions in your pipelined machine. You can use the student-written assembler tool available online to help you quickly implement your test programs. 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. [Hint: write a no-hazard test program, to verify that the pipeline works correctly. Then write a program that has one kind of hazard, then another, then another, etc until all the hazards have been tested.]

**MIPS instructions to implement:**

|  |  |
| --- | --- |
| Section | Instructions |
| Sec 1 | Original10 [optional: Sec1 instructions from Lab 3] |
| Sec 2 | Original10 [optional: Sec2 instructions from Lab 3] |
| Sec 3 | Original10 [optional: Sec3 instructions from Lab 3] |

The Original10 instructions are add, sub, and, or, slt, lw, sw, beq, addi and j. If you want to challenge yourself (and impress the TA !), you may optionally add other instructions from Lab 3.

**Part 2: Implementation and Testing (70%)**

To implement the modified processor, you will need to assemble the modified Verilog modules that model the changes you made to implement the pipelined microarchitecture. 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 that instructions advance one stage in the pipeline. Once the pipeline is full, it means that each push will cause one instruction to finish unless a flush or stall caused nothing to complete that cycle. 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 and regwrite outputs (along with any other control signals that you want to bring out for viewing) can go to LEDs, 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.] So a new top-level Verilog module is needed, containing top.v as well as 2 instantiations of pulse\_controller, and 1 instantiation of display\_controller, and some hand-pushed signals coming from push buttons, and some anode and cathode outputs going to the 7-segment display unit on the BASYS2 board, and some signals going to LEDs.

For each module that you wrote or modified, you will want to test it in simulation, to make sure it works functionally.When you have integrated all the Verilog modules together and your whole pipelined MIPS is working in simulation with the test programs you wrote, call the TA or Tutor and show it for grade. (This part is worth 20 points.)

Now make the .ucf file that maps inputs (50 Mhz clock and pushbutton switches) and outputs (AN and CA signals to the 7-segment display and other signals to LEDs), then you can create a New Project, and implement it on the BASYS2 board, and test it. When your pipelined datapath and control are working correctly in the FPGA, in the test programs with no hazards and containing hazards, call the TA or Tutor and show it for grade. (This part is worth 30 points.)

*Now the TA will ask questions to you, in a single 20-point Oral Quiz,* to determine how much of the 70 points for Part 2 is deserved, based on your knowlege and ability to explain your demo, the 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 modified several Verilog modules for parts of the MIPS that needed to change in order to be a pipelined implemention. You also created some new modules. Now it is time to combine all the new and modified Verilog codes into a file called YourName\_YourSurname.txt . You will the upload this file to the Unilica > CS224 > Assignment for your section. While the TA or Tutor is watching, you will upload this file. Be sure that this .txt file contains exactly and only the codes which are specifically detailed above (new or modified). Check the specifications! *Even if you didn’t completely finish, or get the Verilog codes working, you must submit the YourName\_YourSurname.txt file to the Unilica Assignment for similarity checking.*  If you don’t submit your code, your grade for the lab will be 0 (see Lab Policies section, below NOTES.) 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 plaigarism). So be sure that you only submit code that you actually wrote yourself ! All students must upload their code to Unilica > Assignment while the TA or Tutor is watching, and before the deadline. NOTE: you are allowed to upload only ONE file to Unilica, so be sure it contains exactly and only the codes required.

**Part 4. Cleanup**

1. Erase all the files that you created (Verilog files, .txt files), and that Xilinx created for you. In other words, leave the lab computer exactly as you found it.
2. 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.

**CONGRATULATIONS, you are done with Lab4, and one step closer to becoming a Computer Engineer!**

------------------------------------------------------------------------------------------------------------------------------

**NOTES:**

1. Your Verilog code should be self-documenting. This means that the code contains the following 4 characteristics, enabling anyone to read and understand and maintain the code, without reference to any other document explaining that particular code.
   1. Each routine has an explanatory header at the top, describing what it does
   2. Line comments are plentiful
   3. Good use of white space and indenting make it easy to follow and read
   4. Labels and identifiers are well-chosen and meaningful, to aid understanding
2. Testing code modules individually, and proving their correctness, before integrating them together into larger blocks, is a very good idea. It allows you to isolate the errors to particular locations, to more easily find and fix them.
3. Testing Verilog modules in simulation is a good idea, before moving on to implementation. It allows you to catch bugs early, to debug easily, to make rapid changes and retest quickly
4. A helpful guide to the tasks using Xilinx and the BASYS board is found at this webpage: <http://www.cs.bilkent.edu.tr/~baray/CS223QuickStart.html>
5. The “Suggestions for Lab Success” document posted in Unilica can help you avoid errors and get a successful lab result. It has ideas for correct usage of Verilog, Xilinx ISE projects, simulation, synthesis and debugging.

**LAB POLICIES**

1. Be sure to check and adhere to the CS224 Lab Policies (Lab Policies.docx posted in Unilica)
2. Each student will earn their own individual lab grade, by how much they do, how much they learn, how much they know. The Oral Quiz will have a major effect on your lab score, so be prepared to answer questions concerning the lab topics.
3. Lab score will be reduced to 0 if the code is not submitted for similarity testing, or if it is plaigarized. MOSS-testing will be done, to determine similarity rates. Trivial changes to code will not hide plaigarism 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 it MOSS will find it. The use of the ideas we discussed in the classroom is not a problem. Summary: be sure that you only submit code that you actually wrote yourself !
4. You must be in lab, working on the lab, from the time lab starts until you finish and leave. (Bathroom and food breaks are the exception to this rule). Lateness or early departure is counted as absence from the whole lab that day.
5. No cell phone usage during lab. Tell friends not to call during the lab hours--you are busy learning how computers work !
6. Internet usage is permitted only to lab-related technical sites. No Facebook, Twitter, email, news, video games, etc--you are busy learning how computers work !