Duke ECE152 – Spring 2012 – Project Part 6: Pipelined CPU

200 Points. Due electronically by 11:59pm on April 11.

This is NOT an easy assignment. Start early! That way you have time to get help.

In this part of the project, you will build a complete and pipelined processor starting from your unpipelined processor. As part of this project, you will build your processor in Quartus (VHDL highly recommended), download your design onto an FPGA prototyping board, and demonstrate that it works by running a test program provided to you. Recall that the specification for the Duke152-S12-32 Instruction Set Architecture can be found at http://people.ee.duke.edu/~sorin/ece152/project/arch-spec.pdf.

Project Part 6a: Five-Stage Single-Issue 32-bit Integer Processor

This pipelined processor has the same functional requirements as the unpipelined processor from the previous part of the project. It should use a similar version of the 5-stage pipeline presented in lecture, consisting of Fetch, Decode, Execute, Memory, and Writeback stages.

Whenever data bypassing is possible, you must use it to avoid hazards instead of stalling. If a stall is unavoidable, the hardware is responsible for managing the stall (inserting a bubble into the pipeline). Stalling until a branch, jump, or return is resolved is acceptable. However, keep in mind that bonus points will be awarded to groups with the highest performing processors at the end of the semester, and stalling does hurt performance.

You may implement your instruction decode and pipeline control (i.e. hazard detection and bypassing) combinational logic using Behavioral VHDL. This is the *only* exception to the rule about not using Behavioral VHDL. You learned how to minimize logic in ECE 52; in this class we let the CAD (Computer-Aided-Design) tools do it for us. When-else statements are recommended to generate simple combinational logic; process blocks are not.

A recommended intermediate step in this project is to pipeline your processor first without hazard stalling or bypassing, and test it with extra NOP non-operation instructions inserted into your code to eliminate data hazards. A *testGiveMeN* program with NOPs inserted to eliminate data hazards is provided at http://people.ee.duke.edu/~sorin/ece152/project/testGiveMeN_nodatahazard.asm. Note that only data hazards are eliminated; you must either flush upstream instructions on changes in control flow or add nops to the code yourself. A sample waveform to match is provided at http://people.ee.duke.edu/~sorin/ece152/project/testGiveMeN_nodatahazard.vwf. You may have to shift the keyboard inputs in time if you insert NOPs or have a faster processor.

A sample waveform for your complete pipelined processor running *testGiveMeN* is provided at http://people.ee.duke.edu/~sorin/ece152/project/testGiveMeN_pipelined.vwf. Again, you may have to shift the keyboard inputs in time if you have a faster processor.

Hint: Recall that Register $r0 must always have the value zero, so a non-zero value for it should never be forwarded from the bypass network, including if an earlier instruction had $r0 as

its destination register. Note that this also means that you should not stall on a hazard for $r0; doing so might even lock up your pipeline.

Tip: If you update your Memory Initialization Files but keep the same file names, you can go to the menu item "Processing" "Update Memory Initialization File" instead of recompiling before simulating again in Quartus with your new assembly code. You must still recompile before downloading to the FPGA board.

Project Part 6b: Hardware Demonstration

After designing and testing your processor in Quartus, you will download it to one of the Altera DE2 FPGA prototyping boards and demonstrate that it executes the "testFibonacci" program correctly and quickly. Do not wait until demo day to complete this final step; just because your design works in Quartus does not mean that there will not be hiccups getting it onto the board. A skeleton framework with all of the pin mappings and keyboard and LCD controllers is provided at http://people.ee.duke.edu/~sorin/ece152/project/skeleton.qar. Restore from the archive file (Project Restore Archived Project) and add your processor into this skeleton project. In the skeleton.vhd file, comment out line 38 "div: pll PORT MAP (inclock,clock);" and uncomment line 39 "clock <= inclock;" to run your processor at 50 MHz clock frequency.

Submitting This Assignment

To submit this assignment, create a Quartus Archive (Project Archive Project) named project6.qar of all the files needed to implement your design. Make sure that your processor file is named processor.vhd or processor.bdf. Names of lower-level files are unrestricted, but be sure to include them along with your top-level design entity in the Quartus Archive file. Email your Quartus Archive file as an attachment along with all group members' names and NetIDs to duke.ece152.spring2012@gmail.com. You will also demonstrate your working processor on an FPGA board in lab on a date TBD later.