CprE 381 – Computer Organization and Assembly Level Programming

Project Part A

*[Note from Joe: this is a TEAM assignment and should be actively collaborated on and submitted as a single project team. Although you are still required to document your design process via a lab report, your focus should be more on creating a smart design and **testing** it thoroughly than on a lengthy writeup. The TAs and I are still available to help during the lab sessions, office hours, and on Canvas, but there is now much more freedom in how to implement components and you are expected to come up with major aspects of the design as a project team. Fortunately, you have two weeks to complete this assignment.]*

0) Prelab. With your project group members, create a list of best practices / tips for designing, compiling, and testing VHDL modules based on your experiences so far with these labs.

1) Similar to how we built up the adder/subtractor module in Lab-02, the 32-bit MIPS ALU can be built using a ripple-carry connection between individual **1-bit ALU** components.

(a) Draw a schematic for a 1-bit ALU that supports the following operations: `add`/`sub` (both signed and unsigned), `slt`, `and`, `or`, `xor`, `nand`, and `nor`. To simplify the schematic, you can make use of the 1-bit full adder component you created in Lab-02. What are the inputs and outputs that are needed? *[The ALU description given in P&H Section B.5 is close to what is expected here. We are implementing several additional instructions, which will change the control and part of the datapath. Verify with your TA before proceeding.]*

(b) Implement this 1-bit ALU component in VHDL. In your project writeup, describe your design in terms of the VHDL coding style you chose and the control signals that are required. *[You may use either structural or dataflow VHDL here. It is not clear that the dataflow version of these ALU components will be any simpler to code than the structural equivalents.]*

(c) Use Modelsim to test your 1-bit ALU thoroughly to make sure it is working as expected. Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup. *[It's up to you to decide if a VHDL testbench is necessary. It is crucial that you thoroughly test all your designs from this point forward.]*

2) The **32-bit ALU** component will require three inputs (operand A, operand B, and control), and four outputs (result F, Carryout, Overflow, and Zero).

(a) Draw a simplified schematic for this 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is `slt` implemented? *[The answer to these questions can be found in P&H Section B.5. Verify with your TA before proceeding. Your design may require the use of a special 1-bit ALU for the most significant bit of the `slt` instruction.]*

(b) Implement this 32-bit ALU using structural VHDL. In your writeup, describe what challenges (if any) you faced in implementing this module.

(c) Use Modelsim to test your 32-bit ALU thoroughly to make sure it is working as expected. Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

3) The MIPS ISA also contains several shifting instructions which can be implemented independently from the main ALU. For this part we will consider the **barrel shifter** design discussed in class.

(a) Describe the difference between logical (`srl`) and arithmetic (`sra`) shifts. Why does MIPS not have a `sla` instruction?

(b) Unfortunately, barrel shifters are not covered explicitly in the P&H textbook, but the following Java applet shows how an 8-bit barrel shifter can be created using cascaded 2:1 muxes (link). Implement a 32-bit right shifter (both arithmetic and logical) using structural VHDL. In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations. *[There is a strong pattern to how the muxes are mapped in the conventional barrel shifter design. If you make sense of this pattern your code will be considerably simplified.]*

(c) In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations. Integrate this functionality into your existing design. *[Hint: it can be done by trivially modifying the input and output. An additional shifting component is not necessary.]*

(d) Use Modelsim to test your shifter designs thoroughly to make sure they are working as expected. Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.

4) You should now be able to demonstrate using a **test program** a much wider variety of behavior than your Lab-04 datapath.

(a) Integrate your new ALU into your MIPS datapath from Lab-04. *[If you have not done so already, combine the part 2) ALU and part 3) shifter into a single component with shared inputs.]*

(b) Augment your previous test program(s) with ones that test the new functionality from this lab (e.g., overflow, slt, or, and, nor, xor, sll, srl, sra, etc.). Don't simply test that they work for one value, test them for multiple values including edge cases and justify why your test plan is comprehensive. Include waveforms that demonstrate your test program is functioning.

5) [BONUS POINTS] Add a byte-sized SIMD functional unit that can support the MIPS DSP extensions.

(a) Design a separate ALU to support the following MIPS DSP instructions for 8-bit arithmetic operators: `absq_s.qb`, `addu.qb`, `addu_s.qb`, `raddu.w.qb`, `repl.qb`, `replv.qb`, and `subu.qb`. *[Note that you only need lw and sw to load and store quad bytes, so no additional memory functions are required.]*

(b) Integrate your SIMD ALU into your MIPS datapath from part 4).

(c) Augment your previous test program(s) with ones that test the new functionality from this lab. Don't simply test that they work for one value, test them for multiple values including edge cases and justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs is functioning.