# CprE 381

# Project Part 1 Report

Lab Partners Vishal Joel and Daniel Limanowski

Section/Lab Time B/Thurs @ 10AM

a. [Part 0] 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, both working individually and as a group.

- Using comments on code that may not be straightforward - Liberal comments are mandatory to maintain reusable code.
- Proper indentation to ensure readability and reuse. (1 tab indents)
- Naming convention: All lowercase names with underscores, for readability and name delimiting.
- Naming convention: Signals should be named using ‘s_’ -> s_signal

b. [Part 1 (a)] Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

- The purpose of sra is to support signed numbers represented in two's complement. The most significant bit, which is ‘1’ if the value is negative, is duplicated when shifted right. MIPS does not have an sla instruction because a left-shift arithmetic would shift in zeroes from the LSB - this would be the exact same maneuver as a sll, making sla unnecessary.

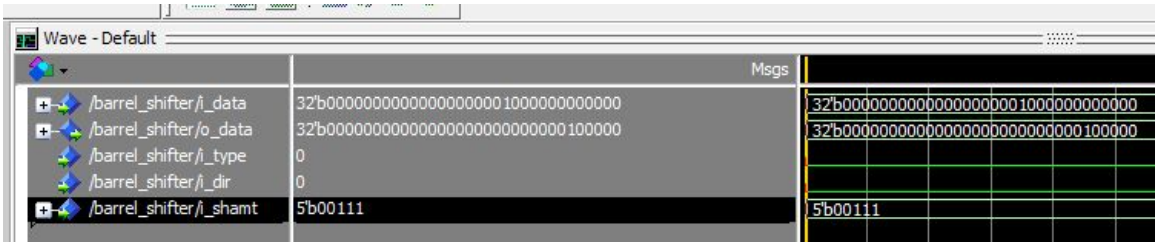
c. [Part 1 (b)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.
- The VHDL code uses a “with-select” operation to set a “shift bit” if the inputted “type” bit is set to “0” (logical) or “1” (arithmetic). If logical, “shift bit” is set to “0”. If arithmetic, “shift bit” is set to the most-significant-bit (MSB) of the inputted data.

d. [Part 1 (c)] In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations.
- The operation of the right barrel shifter need not be altered. A 2-1 multiplexer can be placed in front of the right shifter with a direction bit as the selector. The two inputs to the multiplexer would be 1) the normal input data to be shifted and 2) the inputted data in REVERSE order. This reversing would require another module to be built that is simply a for loop that swaps all the positions in the input vector. data(31) becomes reverse_data(0) and so on. If a left-shift is desired, the selector chooses the reversed data to be inputted to the right-shift module. After

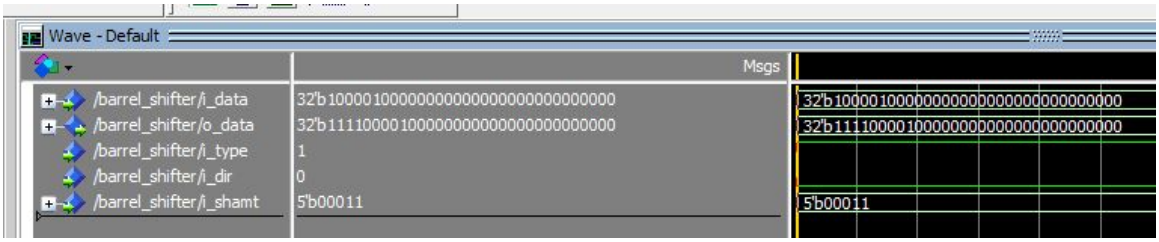
the shift is performed, the left-shift data is reversed one more time to bring it to proper order.

e. [Part 1 (d)] Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.

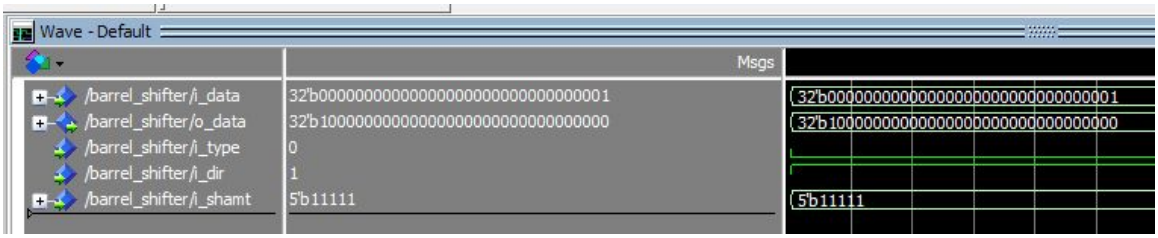
Shift-right-logical (SRL) shifts in zeroes from MSB while moving the data towards LSB.




Shift-right-arithmetic (SRA) shifts in whatever the MSB is while moving data towards LSB.



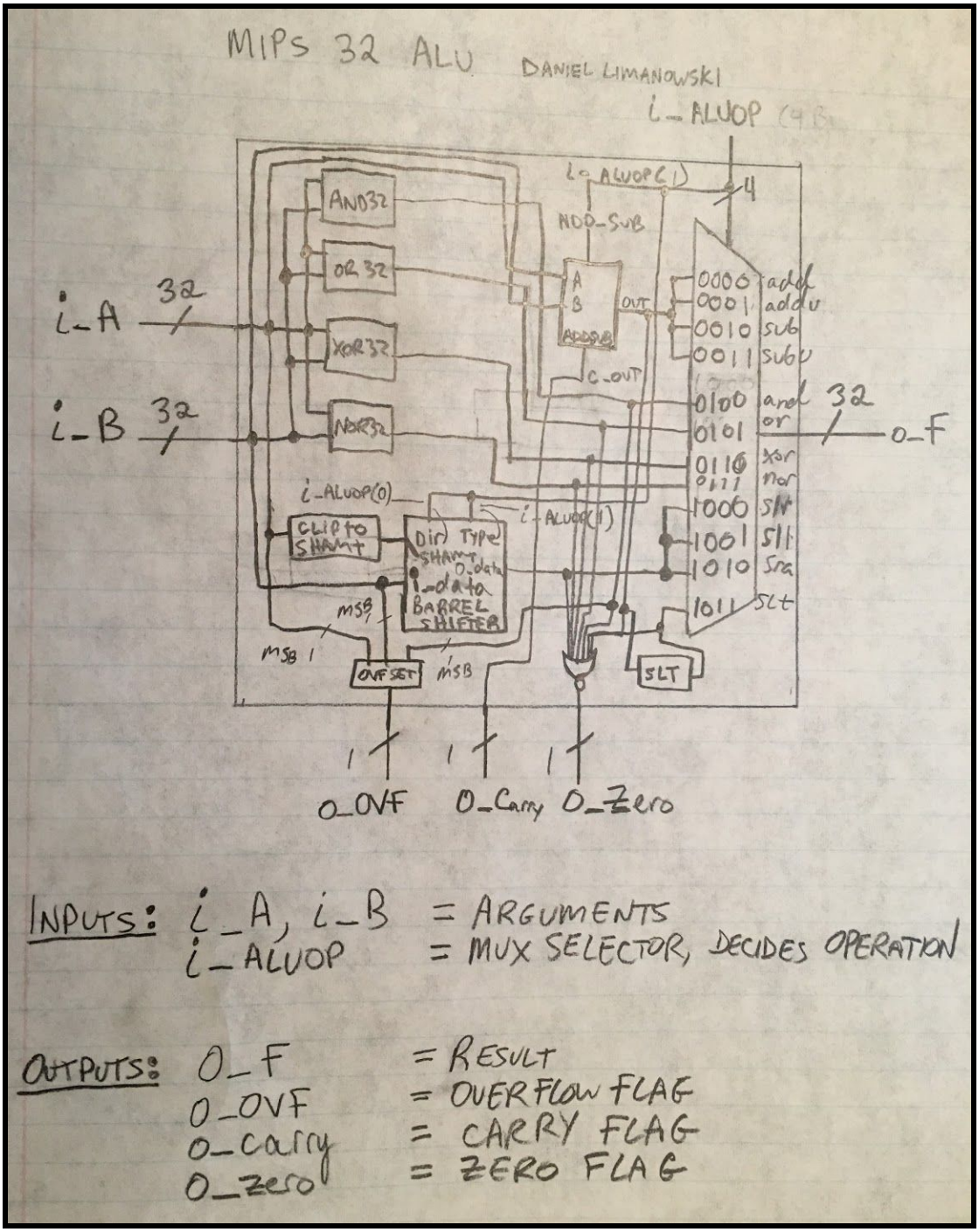

Shift-left-logical (SLL) shifts in zeroes from the LSB while moving data towards MSB.




f. [Part 2 (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?

- SLT implementation: Set-less-than instruction is implemented by checking if the result of (a - b) is negative. If negative, (a < b) is true, so we set the LSB of Result (Result(0)) to '1'. If (a-b) is positive, we set Result(0) = '0'. All other bits of Result (ie., 1..31) are always set to '0' regardless. A Less input is added to each 1-bit ALU and for ALU1-30, the Less input is forced to '0'. ALU0's Less is set to ALU31's Set output.
- Zero is calculated by checking all inputs to our ALUOP mux. If any of these inputs are greater than zero, the Zero flag is set to off. If ALL of these inputs are zero, the zero flag is set to on. Using a NOR with the mux inputs as input, the output is our zero flag value.

- Overflow is detected by checking the values of MSB of A, B, and Result of Add/Sub. If (the MSB of A = B) AND (the MSB of Result != (A or B's MSB)), then set Overflow flag. Overflow (OVF) is ONLY looked at in SIGNED operations (add, sub).
- 32-bit ALU simple schematic (NOTE: ALUOPcodes have changed. See Mux7-1 file for current opcodes):

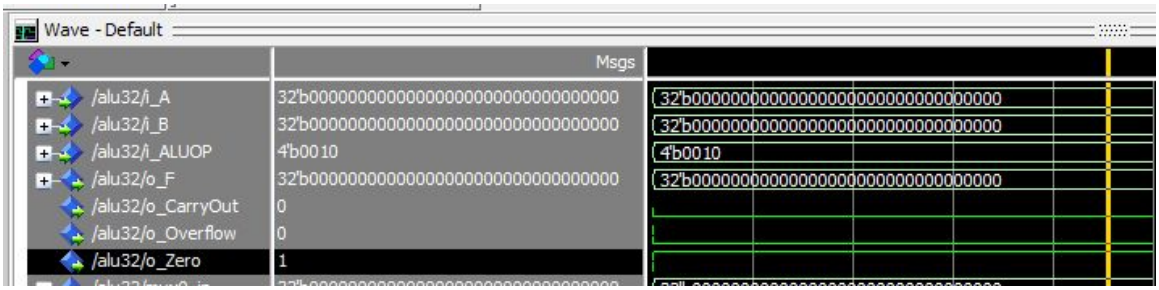


g. [Part 2 (b)] In your writeup, describe what challenges (if any) you faced in implementing this module.

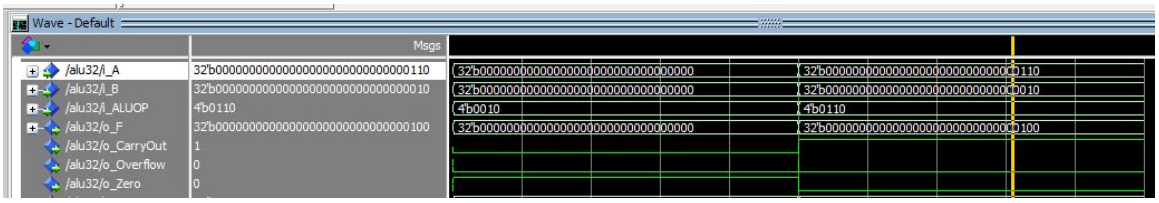
Wrapping my head around the entire ALU unit, with all of its operations, was a challenge. However, once I drew in detail the ALU and all of the components needed to implement it, actually writing the VHDL code wasn't all that bad. Office hours were a good assist in understanding what was expected from the ALU and how to get started.

h. [Part 2 (c)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

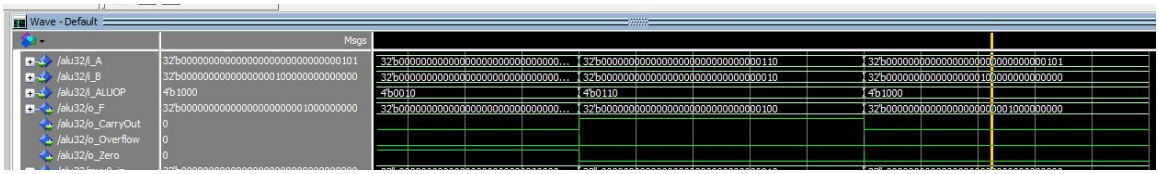
ALUOP for Add = 0b0010. We added 0+0 and the result was 0. You can see the Zero flag is properly set in the figure below.



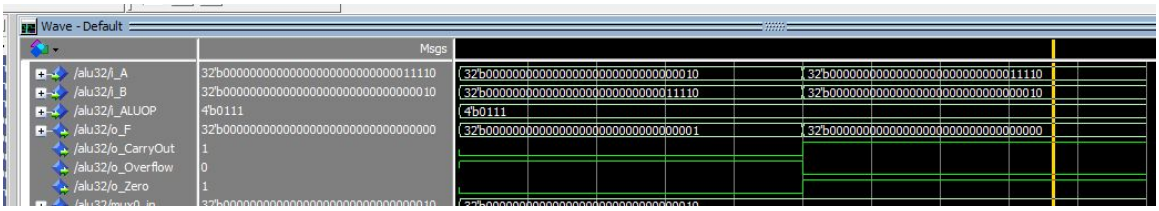

ALUOP for Sub = 0b0110. We subtracted 0b010 from 0b110 and obtained 0b100.




ALUOP for SRL is 0b1000. SHAMT (shift amount) is i_A, set to shift right 5 bits. The register data to shift is i_B, a lonely bit in a sea of 31 zeroes. You can tell the bit is properly shifted right 5 bits.

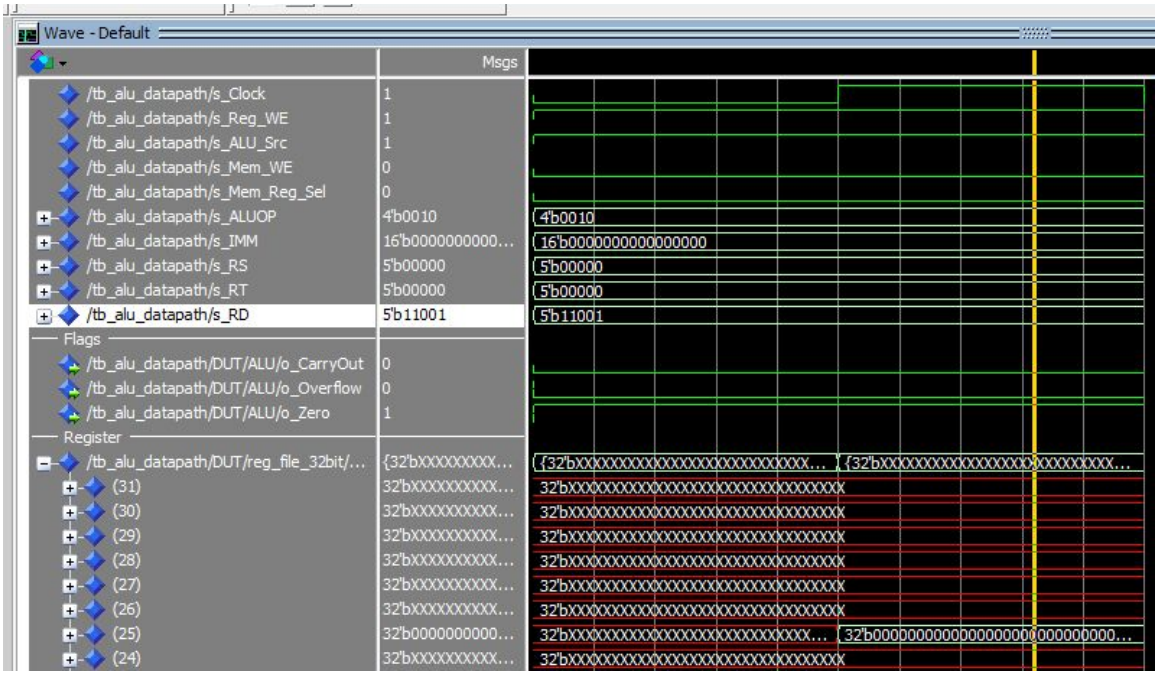



ALUOP for SLT is 0b0111. First we set A<B and see the answer is 1 (correct). Then we set A>B and see the result is also correct (= 0).

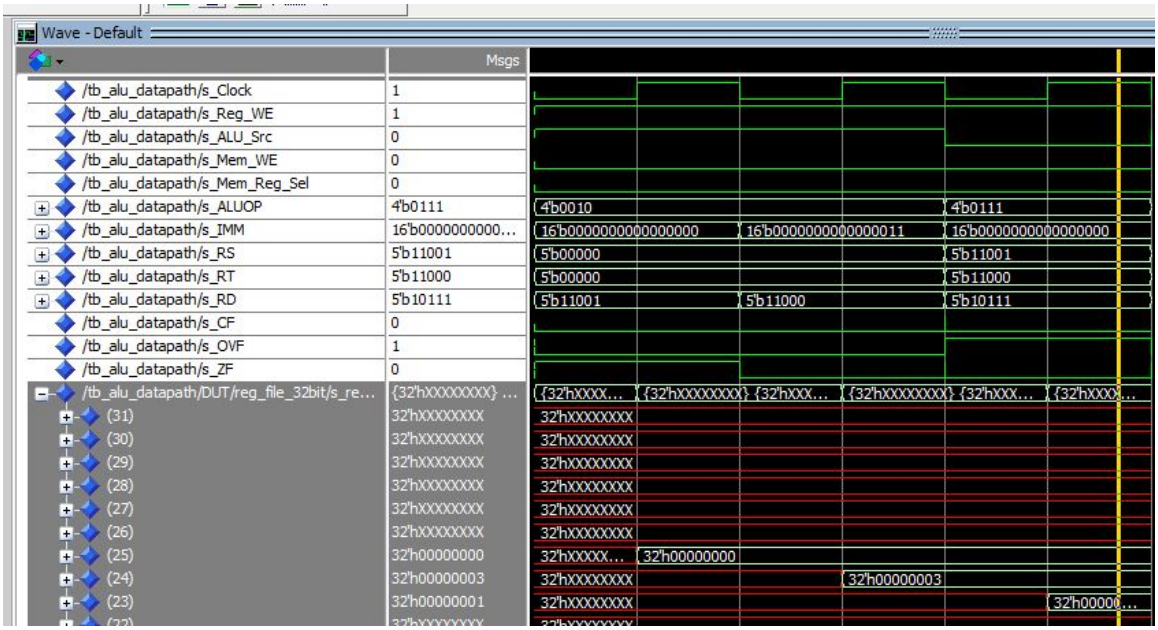



i. [Part 3(b)] justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.

Our test plan is comprehensive because I developed a testbench to ensure accuracy for all the possible commands. I have included in the report a few screenshots from my tests.




The figure above shows addition of the zero register with an immediate “0”. The ALUOP for addition is “0010” as seen in the waveform.




The above image shows the SLT operation in working condition. We compare register 25 (holding a value of 0) to register 24 (holding a value of 3). 3 is clearly greater than 0, so SLT is set to 1 (whose result is placed into $23, as seen in the waveform).

j. [Part 4(d)] Does your design meet the timing constraints that were set? Report the critical path from register to register using TimeQuest. What is the fastest clock period that your MIPS datapath can be safely run at?

Because efficiency was the forefront behind this design, our design indeed met the timing constraints.