CprE 381 – Computer Organization and Assembly-Level Programming

Proj-A Report

Lab Partners Jacob Vaughtn
Nickolas Mitchell

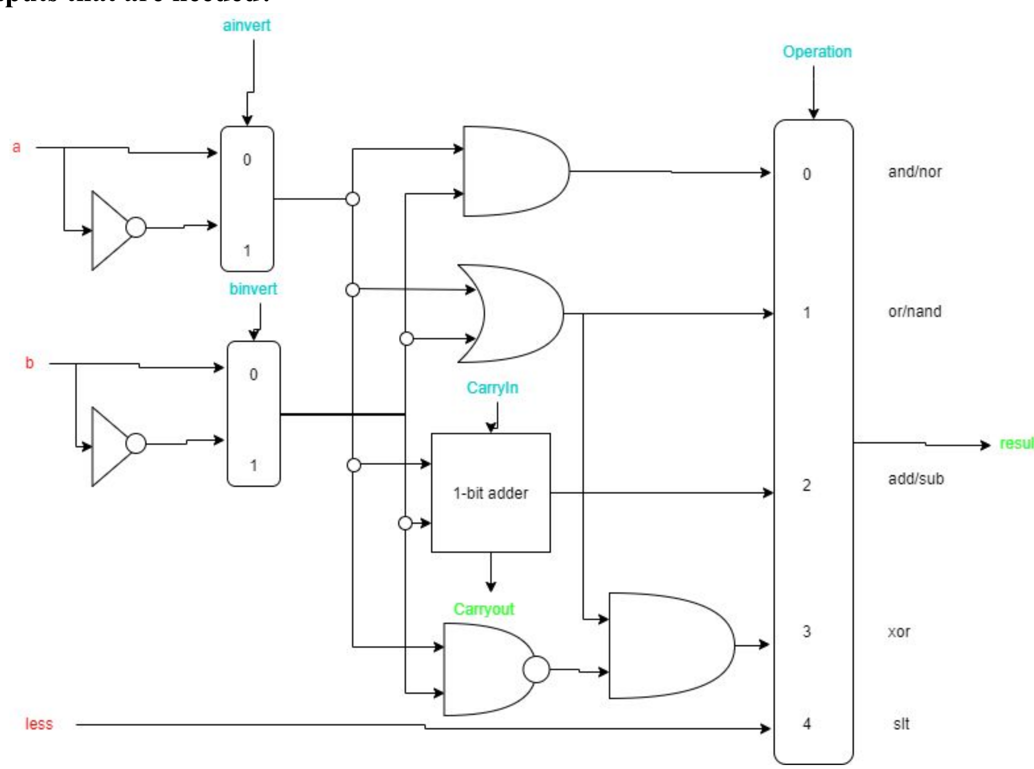
Section / Lab Time 1/4:10-6:00

Submit a typeset pdf version of this on Canvas by the due date. Refer to the highlighted language in the Proj-A instructions for the context of the following questions.

[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.

Our best practices and tips for designing, compiling, and testing VHDL modules include giving meaningful names to entities and signals, using useful comments to explain different parts of our code, perform various tests using the simulation tool, and compile often.

[Part 1 (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. What are the inputs and outputs that are needed?



Input's are *a*, *b*, *less*, and then control's for *ainvert*, *binvert*, *carryin*, and *Operation (3bit)*. The output is going to be from the operation mux: *result* and a *Carryout* from the adder.

[Part 1 (b)] In your project writeup, describe your design in terms of the VHDL coding style you chose and the control signals that are required.

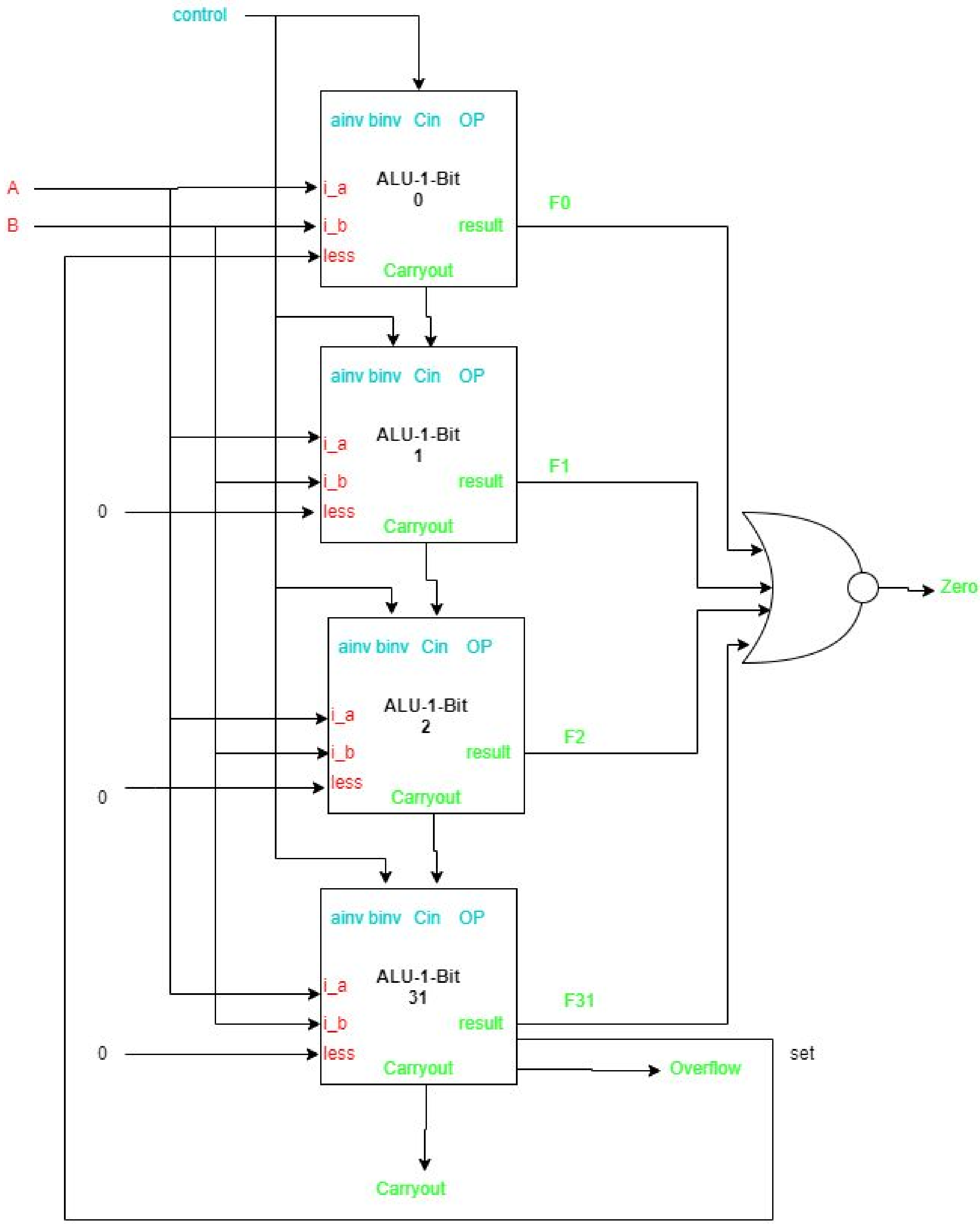
For our design we are using mostly structural coding with a little data flow mixed in. We have used this coding style mostly due to the fact this is what we are most comfortable working with. Our design makes use of 6 control bits (3 to select the operation to output, 2 to select between a and b or not a and b, and 1 carry-in bit).

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

ainvert, binvert, Carryin, Operation-3bit

000000 : AND
000001 : OR
000010 : Add
000011 : xor
011100 : slt
110000 : NOR
110001 : NAND
011010 : Sub

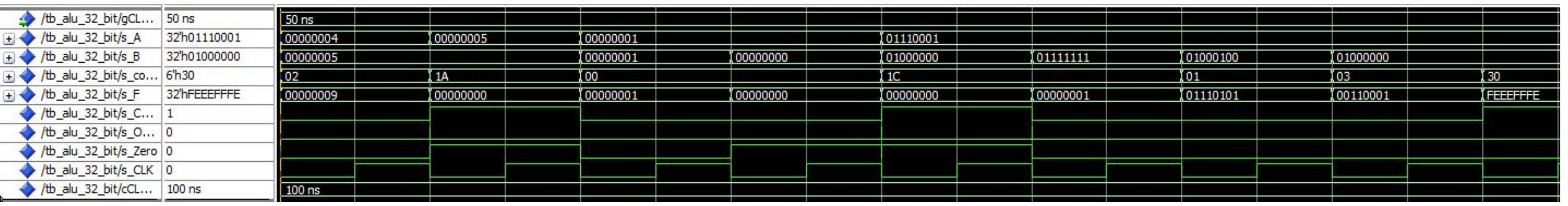
[Part 2 (a)] Draw a simplified schematic for this 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is slt implemented?



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

I thought when doing a generate that it would act like a process loop, but it is actually just setting up the circuit aka generating it and then running everything. After we understood this it was easy to make the slt functionality.

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



Here is one of my sets of test as you can see i start with a 2 operation which is Add where s_A is added to s_B and output to s_F

Next I did a 1A operation which is subtraction

then 0 operation which is and AND operation of the two inputs

1C which is a slt and this works both when a is bigger than b and vice versa

next a 01 op which is an OR

then 03 which is XOR

and finally a 30 which translates to NOR or the inputs.

these are some of my test cases to give an example o operands.

[Part 3 (a)] Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

In MIPS, the logical right shift (srl) always shifts bits to the right cutting off any bits which shift past the 0th bit and adds 0's in the most significant bits up until the 31st bit. The arithmetic shift right (sra) performs exactly like a logical shift right if the most significant bit is a 0. When the most significant bit is a 1 instead of a 0, the arithmetic shift right shifts 1's. MIPS does not a shift left arithmetic (sla) instruction because signed values will never shift 1's to the left. Signed values only shift 1's to the right.

[Part 3 (b)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.

Firstly, to implement the shift right logical operation our code simply consists of 5 32bit 2:1 multiplexers which shift to the right by 1bit, 2bits, 4bits, 8bits, 16bits respectively using concatenation and 5 bit control signal which determines how many bits to shift. Next, our shift right arithmetic operation starts out by taking saving the most significant bit to a 1 bit signal then that bit is used to shift right with the previously mentioned multiplexers.

[Part 3 (c)] In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations.

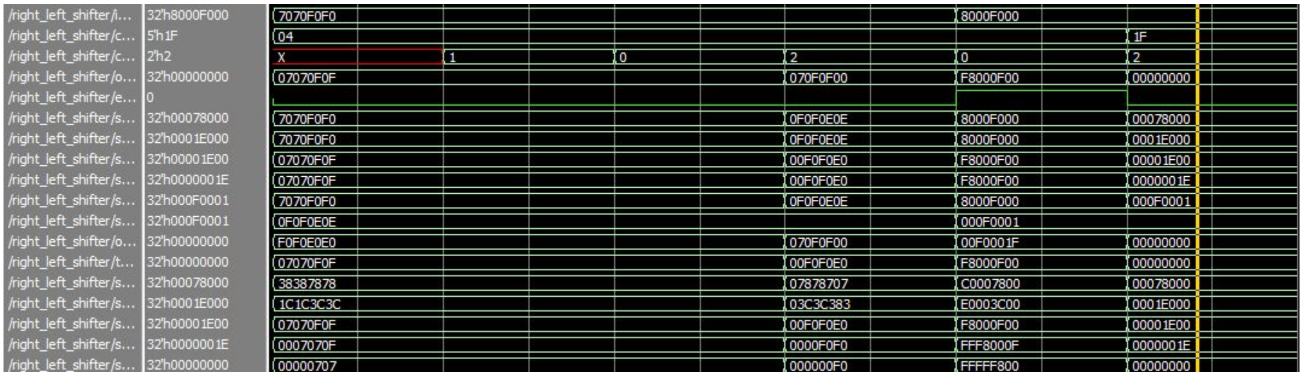
To shift right, our code flips the input bits and performs a right logical shift then the result is flipped back again to represent a shift left operation. Our code utilizes a 2 bit control signal and with else statements to select which of the three shifts our code will perform.

'00' <= arithmetic shift right

'01' <= logical shift right

'10' <= Shift left

[Part 3 (d)] Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.



For the first 6 simulations you can see I was doing variation of shift right logical and shift right arithmetic with the most significant bit being ‘0’ therefore all of these results are the same. For the next simulation I tested the shift left functionality of our circuit. The second to last simulation I tested the shift right arithmetic with the most significant bit being the ‘1’. The last simulation I tested a shift left 32bits.

[Part 4(b)] Justify why your test plan is comprehensive. Include waveforms that demonstrate your test program is functioning.

We have decided to include basic functionality tests to test if our component works in general. Furthermore we have included edge cases into our test bench as we believe these cases to be the most likely to have bugs. Unfortunately we ran into many issues with our lab 4 mips component and therefore we were unable to generate a waveform. The lab 4 mips component we tried to use seem to work fine in the lab 4 folder but when we copied it over to our project_a folder, for some reason it decider to longer write the correct values to the register file. We are currently in the process of figuring out how to fix our lab 4 mips component to move forward with future projects. Lastly, we hope our test bench will give a decent idea of how our project_a component is intended to function.

[Part 5(c) BONUS] Justify why your test plan is comprehensive. Include waveforms that demonstrate your test program is functioning.

[Feedback] You must complete this section for your lab to be graded. Please complete each column separately for each team member; I expect it to take roughly 10 minutes (do not take more than 20 minutes).

i. How many hours did you spend on this lab?

Task	**During Lab time**		**Outside of lab time**	
Team Initials	JV	NM	JV	NM
Reading lab	0	0.25	0.5	0.5
Pencil/paper design	0.5	0.5	0	0.5

VHDL design	1.5	1	3	2
Assembly coding	0	0	1	0
Simulation	1	0.5	1	1
Debugging	1	1	4	4
Report Writing	0	0.5	0.5	1
Other	0	0.25	0	0
Total	4	4	9	9

ii. If you could change one thing about the lab experience, what would it be? Why?

If we could change one thing about this lab, we would have spent more time before this lab began testing the components used in previous labs. Having components from pervious labs which don't work correctly have left us scrambling to finish project A.

iii. What was the most interesting part of the lab?

The most interesting part of this lab was figuring out how to modify our ALU to support several other MIPS instructions. We thought this was a cool process in general and hope we will be able to condense instructions into single components in future labs.