CprE 381 – Computer Organization and Assembly-Level Programming

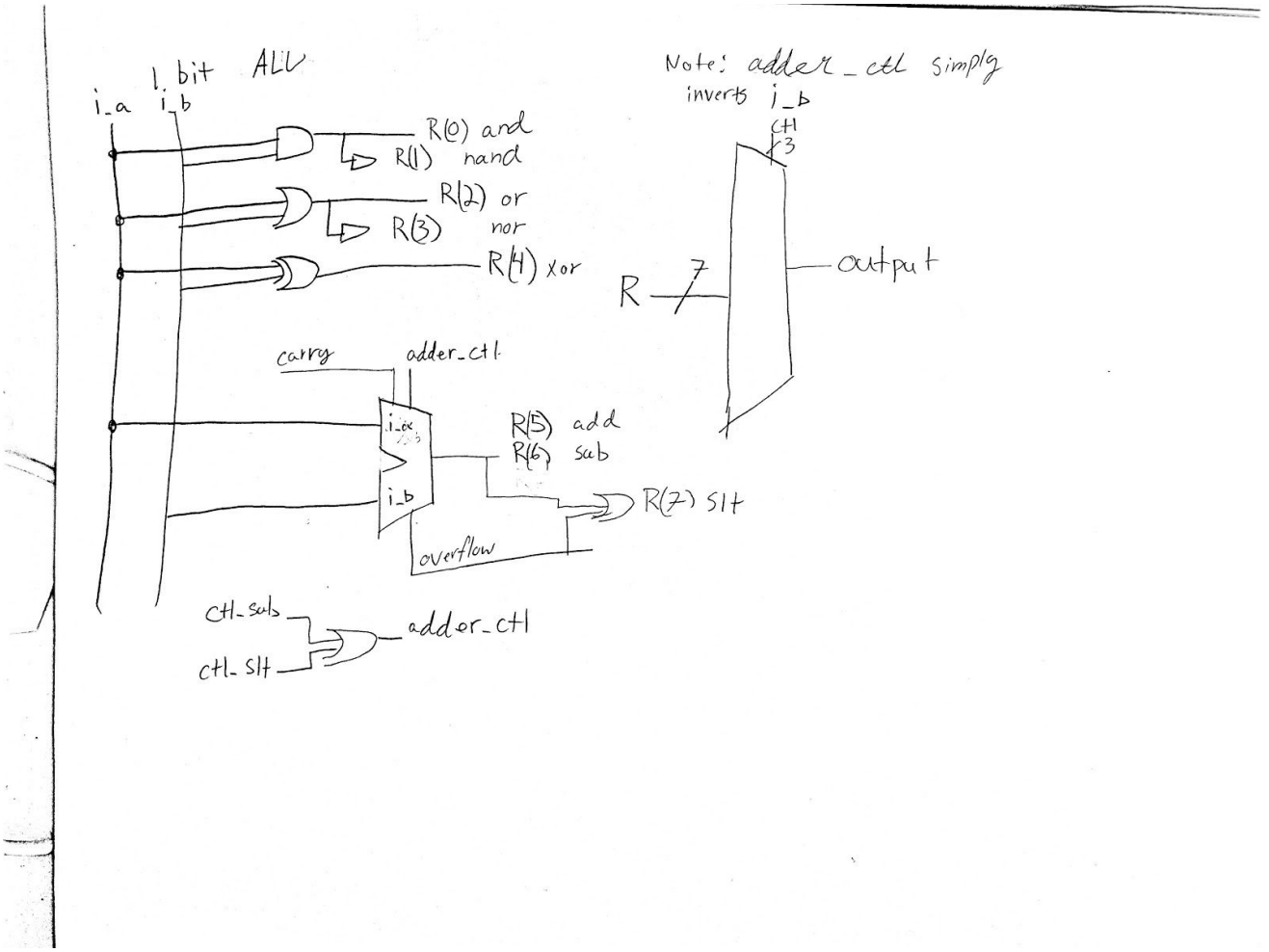
Proj-A Report

Lab Partners	Jon Schnell
	Nicholas Krabbenhoft
	Constantine Mantas
Section / Lab Time	7 Thursday 2-4pm

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.

a. [Part 0] Updated Project Team Contract (if applicable). 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.

b. [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?



There are 4 inputs needed. 2 data inputs, a 3 bit wide control signal, and a carry in bit if they are planned to be implemented in a ripple fashion.

c. [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.

My style for writing the 1 bit ALU was a combination of structural and flow. I used components from previous labs wen possible but for certain functions such as the large mux at the end and the logic operations, it was easier to use flow type statements. For signals I had the first chunk of the name be their function or how they were created. For example, all of the internal_* signals move data around and the ctl_* signals are a binary result of whether that function is being called. All of the input/output signals start with in_* and out_* as well. There was 3 ctl bits needed because I am supporting 8 operations.

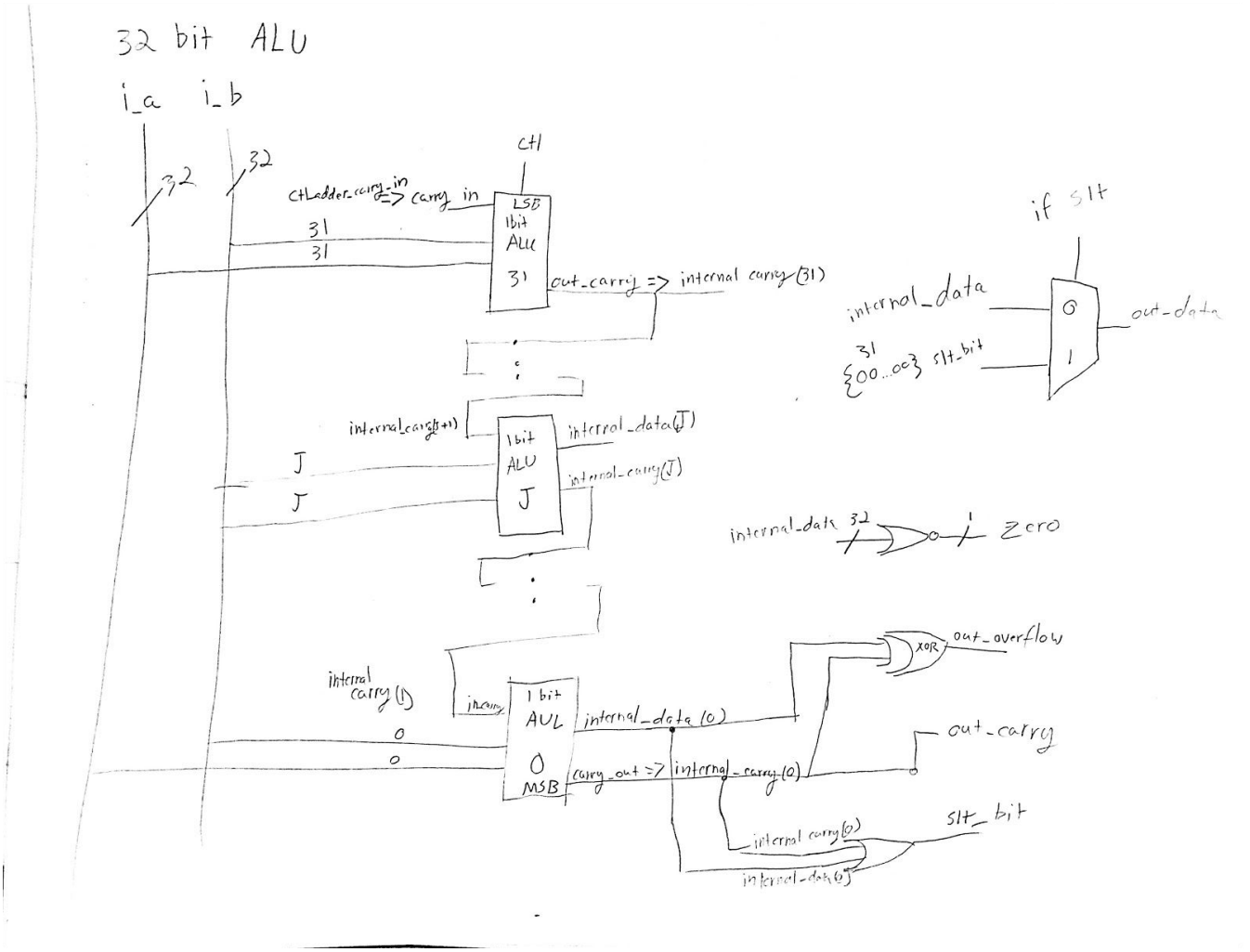
d. [Part 1 (c)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

The waves are pretty self explanatory. The i_a and i_b are the inputs, the ctl values are mapped below and the carry out and data out are the carry and data respectively.

For the tests I set the inputs to 1 1 and then run all of the logical tests, then I set them to 1 0 and repeat. So on until I've done every possible logical test. For add/sub/slt I do it 1 instruction at a time and treat the 3 inputs ia ib and carry as a 3bit number and increment it for every test. I repeated until I tested every possible input. These tests are found in the testbed.do file in the 1 bit ALU folder.
See ALU1Bit_test 1 for video

command	Hex value	0bit	1bit	2bit
and	7	1	1	1
or	6	1	1	0
xor	5	1	0	1
nand	4	1	0	0
nor	3	0	1	1
add	2	0	1	0
sub	1	0	0	1
slt	0	0	0	0

e. [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?
   Overflow is simply an xor of the carry out of the Most Significant ALU and the msb of the data. Zero is simply all of the data bits nor'ed together. The slt is simply an added on mux at the end the outputs the data from the small ALU's normally but when slt is called it outputs 31 zeros and then a or of the msb and carry out from the last 1 bit ALU.



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

The largest problems in creating this was the fact that it was a ripple carry design rather than a 32 bit design. This made it significantly harder to debug. There was also some misplaced bits on how to move data between the 32 1bit ALUs. I also had some trouble deciding how to set the slt then bit, but then remembered that we had answered that problem in a homework assignment, so I just used that.

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

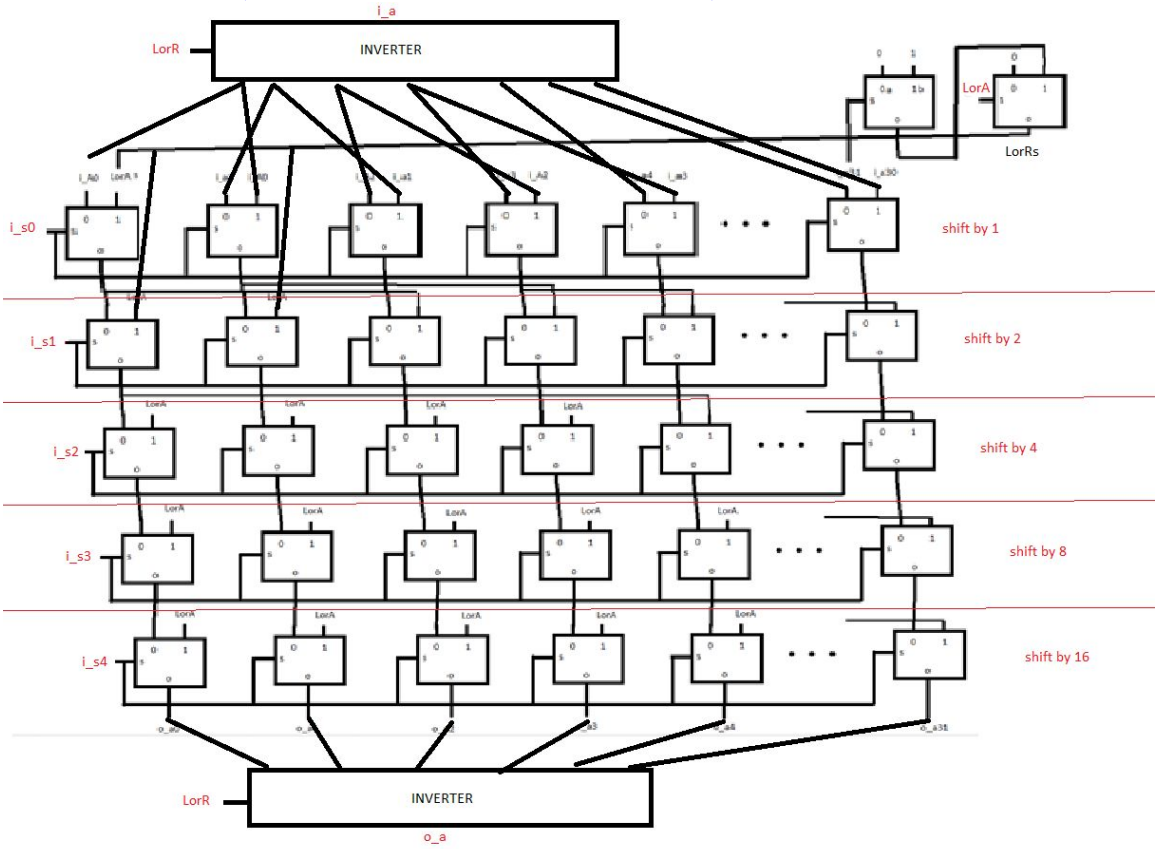
The same control pattern from the 1 bit ALU. The first tests are simply the logical ones. I used 0x55555555 and 0x33333333 because the bits are 0101 and 0011 respectively so they should test all possible outcomes for each operation. Completing that, I used the min and max numbers to test edge cases for the add/sub/slt as well as some smaller numbers to get more common cases. These tests are also part of the testbench.do file in the ALU32Bit folder. See video for the wave forms.

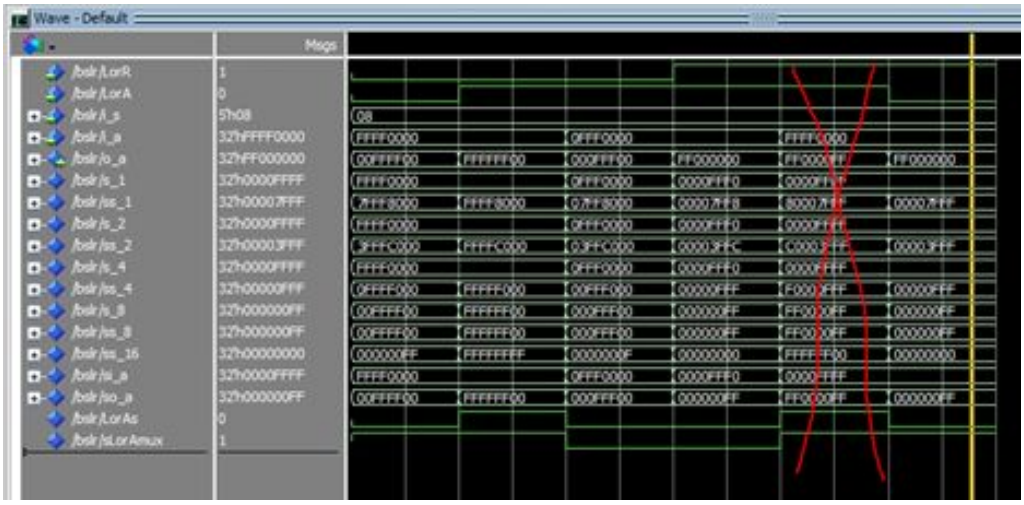
h. [Part 3 (a)] Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

Srl is a logical operation used for unsigned numbers meaning we can pad the new shifted number with 0's. Sra is an arithmetic operation used for signed numbers because when shifting signed numbers right we need to pad with 1's instead of 0's. there is no need for a shift left arithmetic because the padding will be zero for both signed and unsigned numbers.

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

I broke this problem up into 5 generate loops, one for each bit of select line because each bit represents a shift by 1, 2, 4, 8, or 16 bits. As the data flows through each select bit set to 1 will apply a shift amount corresponding to the active bit. To implement the logical or arithmetic operations offered by the shifter I created a couple multiplexers to look at the most significant bit of the input and the logical of arithmetic input if both are 1 we know to append one's on a right shift. There is no protection from trying a left arithmetic shift, it is down to the programmer to not allow this situation. (shown in waveform below with a red X).

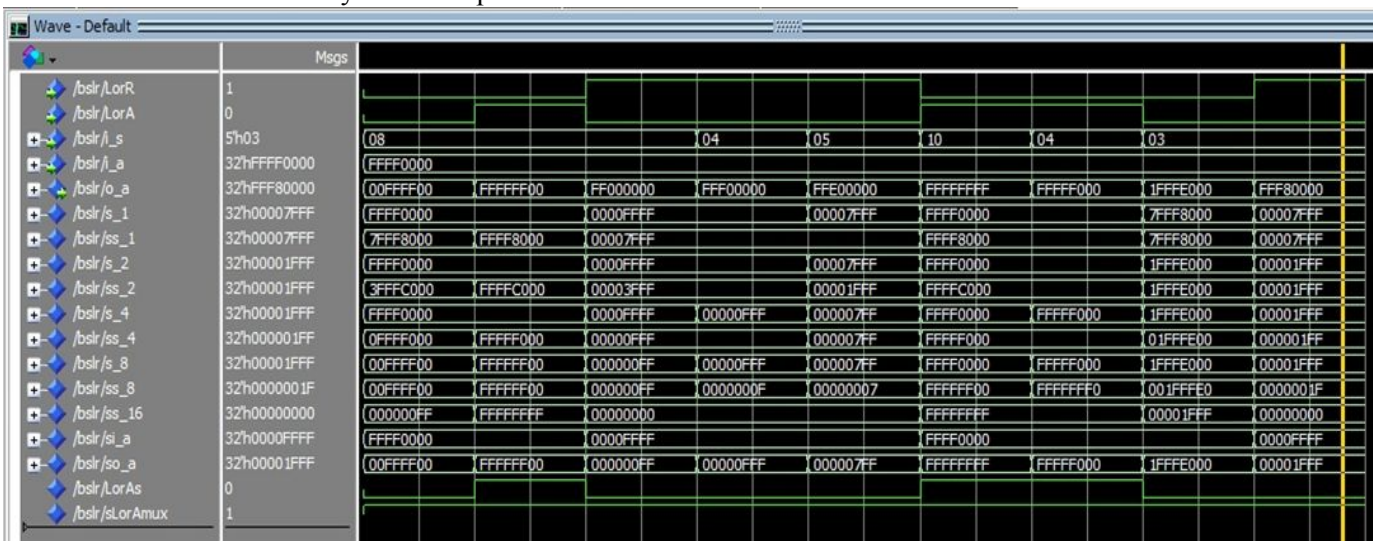




j. [Part 3 (c)] In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations.

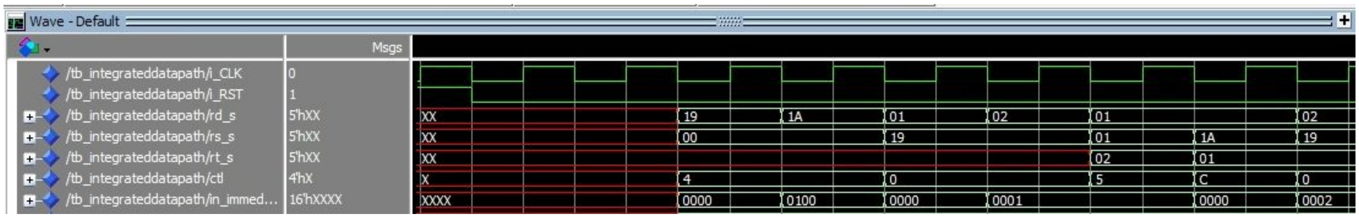
To do this we just need to invert the input before the shift and again invert back after the shift, this works because a backwards number shifted to the right is the same as a left shift. To do this I made a component called reverse that reverses a 32 bit number if the left shift bit is set to 1. I added this component to the structure before and after the output.

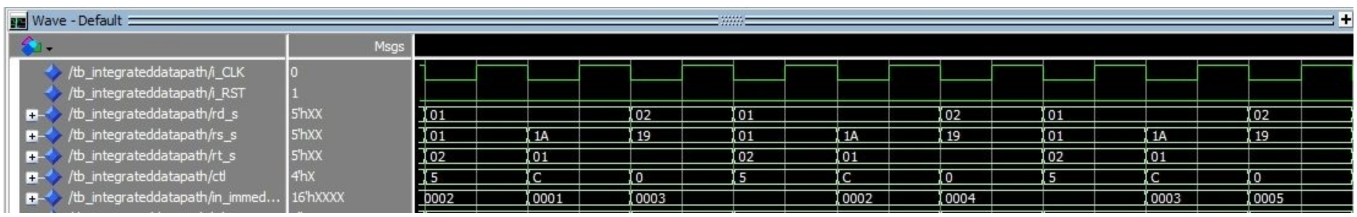
k. [Part 3 (d)] Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.

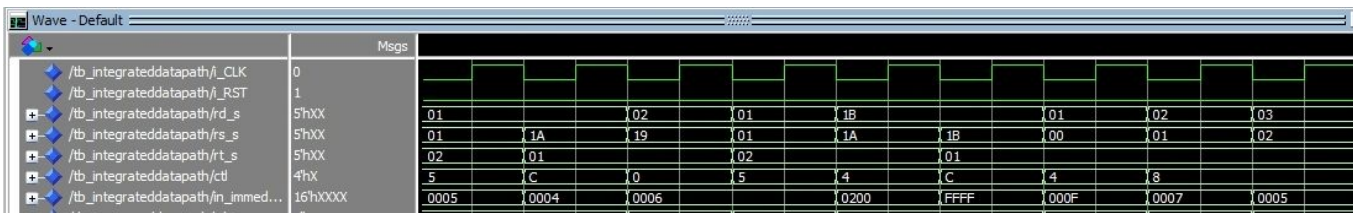


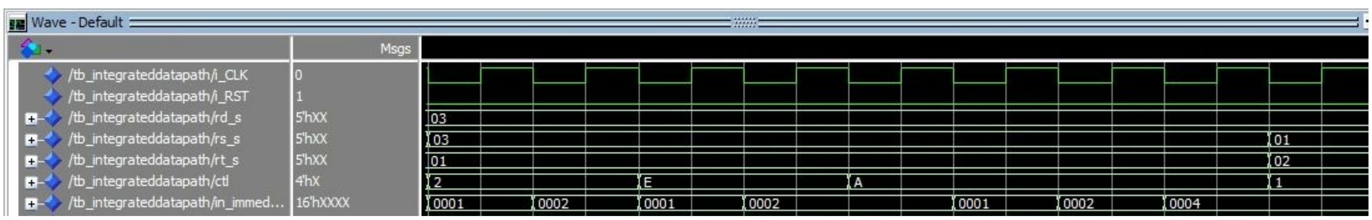
1. SRL by 8 Testing a basic right shift
2. SRA by 8 Testing the arithmetic functionality
3. SLL by 8 Testing basic left shift
4. SLL by 4 Testing basic left shift
5. SLL by 5 Testing stacking of multiple stages of shift(1and4)
6. SRA by 10 Testing combination of arithmetic and multistage shift
7. SRA by 4 Testing of arithmetic shift
8. SRL by 3 Basic right shift test
9. SLL by 3 Basic left shift test

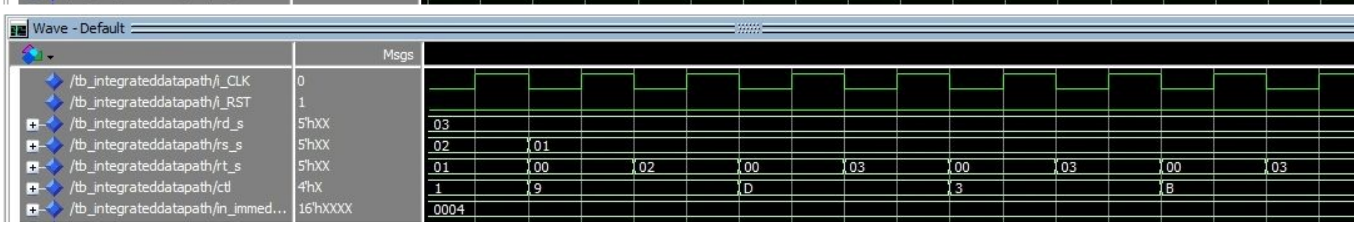
l. [Part 4(b)] Justify why your test plan is comprehensive. Include waveforms that demonstrate your test program is functioning.

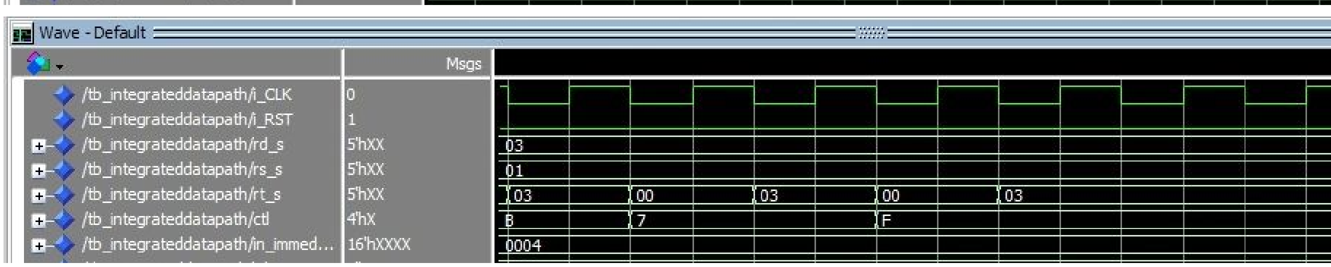












The code for the testbench of the integrated datapath first demonstrates the entire lab 4 demo, to prove that lw, sw, add, addi all work as they should. Then new cases are created and labeled for each mips operation that was added into the ALU. By adding new general cases for each op to the ALU alongside the edgecase tests for the individual parts we are confident that the current integrated datapath works as expected.

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

n. [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	js	nk	cm		nk	cm
Reading lab	.5	.75	.5			0
Pencil/paper design	1.5	1.25	1		1	0
VHDL design	3		.5		3	1
Assembly coding	.5		0			2
Simulation	.5		0		1	0
Debugging	1		0		4-5	3
Report writing	.5		0		.5	0
Other:	0		0			0
Total	7.5	2	2		10	6

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

It would be nice to actually meet partners face to face.

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

Seeing the computer come together is a very rewarding experience.