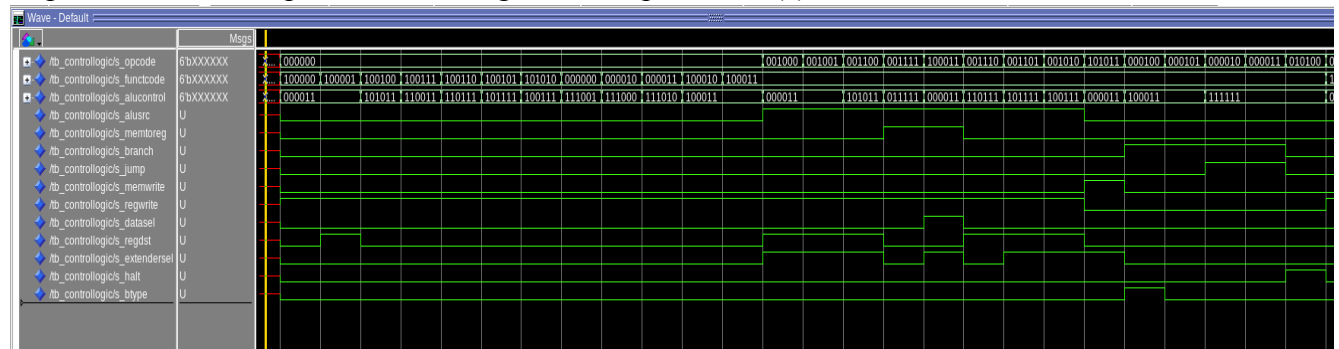


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Instruction	Opcode (Binary)	Funct (Binary)	Control Signals															
2				ALUSrc	ALUControl	MemtoReg	Memwrite	regwrite	RegDst	extendersel	jump	branch	dataset	Btype	movn	jaldst	jaldata	jr	halt
3	add	000000	100000	0	000011	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4	addi	001000	X	1	000011	0	0	1	1	1	0	0	0	0	0	0	0	0	0
5	addiu	001001	X	1	000011	0	0	1	1	1	0	0	0	0	0	0	0	0	0
6	addu	000000	100001	0	000011	0	0	1	1	0	0	0	0	0	0	0	0	0	0
7	and	000000	100100	0	101011	0	0	1	0	0	0	0	0	0	0	0	0	0	0
8	andii	001100	X	1	101011	0	0	1	1	1	0	0	0	0	0	0	0	0	0
9	lui	001111	X	1	011111	1	0	1	0	1	0	0	1	0	0	0	0	0	0
10	lw	100011	X	1	000011	1	0	1	0	1	0	0	1	0	0	0	0	0	0
11	nor	000000	100111	0	110011	0	0	1	0	0	0	0	0	0	0	0	0	0	0
12	xor	010001	100110	0	110111	0	0	1	0	0	0	0	0	0	0	0	0	0	0
13	xori	000000	001110	1	110111	0	0	1	1	1	0	0	0	0	0	0	0	0	0
14	or	000000	100101	0	101111	0	0	1	0	0	0	0	0	0	0	0	0	0	0
15	ori	001101	X	1	101111	0	0	1	1	1	0	0	0	0	0	0	0	0	0
16	slt	000000	101010	0	100111	0	0	1	0	0	0	0	0	0	0	0	0	0	0
17	slti	001010	X	1	100111	0	0	1	1	1	0	0	0	0	0	0	0	0	0
18	sll	000000	000000	0	111001	0	0	1	0	0	0	0	0	0	0	0	0	0	0
19	srl	000000	000010	0	111000	0	0	1	0	0	0	0	0	0	0	0	0	0	0
20	sra	000000	000011	0	111010	0	0	1	0	0	0	0	0	0	0	0	0	0	0
21	sw	101011	X	1	000011	0	1	0	0	1	0	0	0	0	0	0	0	0	0
22	sub	000000	100010	0	100011	0	0	1	0	0	0	0	0	0	0	0	0	0	0
23	subu	000000	100011	0	100011	0	0	1	0	0	0	0	0	0	0	0	0	0	0
24	beq	000100	X	0	100011	0	0	0	0	0	0	1	0	1	0	0	0	0	0
25	bne	000101	X	0	100011	0	0	0	0	0	0	1	0	0	0	0	0	0	0
26	j	000010	X	0	111111	0	0	0	0	0	1	1	0	0	0	0	0	0	0
27	jal	000011	X	0	111111	0	0	0	0	0	1	1	0	0	0	1	1	0	0
28	jr	000000	001000	0	111111	0	0	0	0	0	1	1	0	0	0	0	0	1	0
29	movn	000000	001011	0	111111	0	0	0	0	0	0	0	0	0	1	0	0	0	0
30	halt	010100	X	0	11111	0	0	0	0	0	0	0	0	0	0	0	0	0	1
31																			

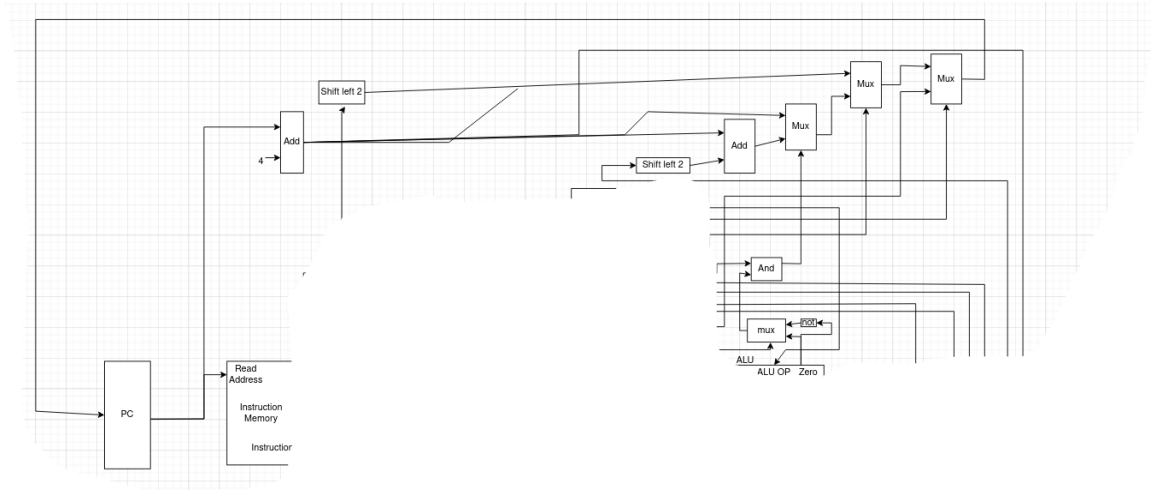
[Part 2 (a.ii)] Implement the control logic module using whatever method and coding style you prefer. Create a testbench to test this module individually and show that your output matches the expected control signals from problem 1(a).



[Part 2 (b.i)] What are the control flow possibilities that your instruction fetch logic must support? Describe these possibilities as a function of the different control flow-related instructions you are required to implement.

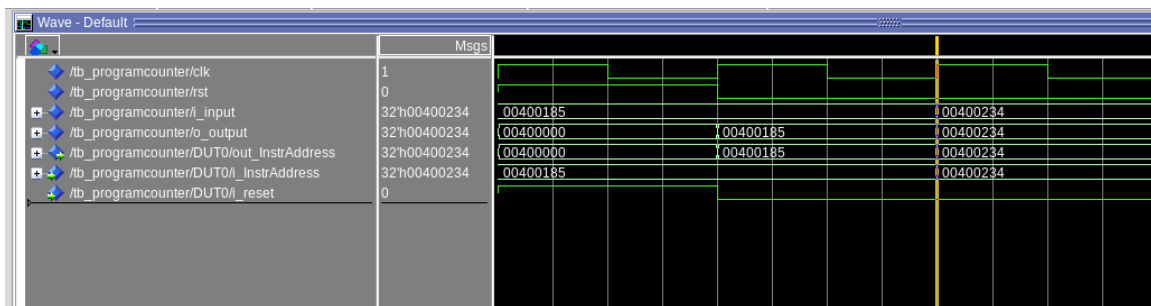
The control flow possibilities my instruction fetch logic must support is incrementing the address by 4 after a cycle, jump, jump and link, branch, branch not equal, and resetting back to the original starting address.

[Part 2 (b.ii)] Draw a schematic for the instruction fetch logic and any other datapath modifications needed for control flow instructions. What additional control signals are needed?



branch type, jump, bne, beq, jr are the new control signals needed

[Part 2 (b.iii)] Implement your new instruction fetch logic using VHDL. Use QuestaSim to test your design thoroughly to make sure it is working as expected. Describe how the execution of the control flow possibilities corresponds to the QuestaSim waveforms in your writeup.



Within the wave form the input data for the PC is not set because the reset value is high to start. Once that comes down the input data is set in the PC and moves onto the next instruction that it takes in and stores.

[Part 2 (c.i.1)] Describe the difference between logical (`srl`) and arithmetic (`sra`) shifts. Why does MIPS not have a `sla` instruction?

SRA does the arithmetic shift which means it shifts the sign in bits, SRL does the logical shift meaning it will shift zeros.

We use SLA to shift the 32 bits in register to the left (a bit at a time). MIPS does not have SLA instruction because there are no sign bit on the right hand side of the register to extend.

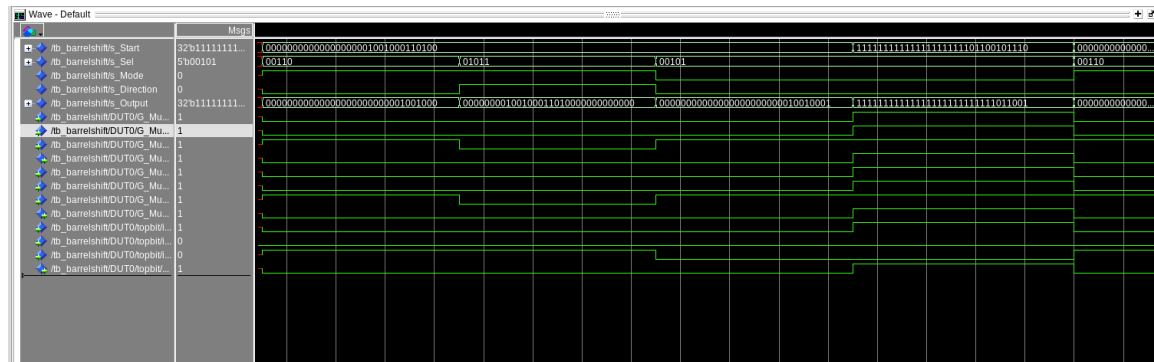
[Part 2 (c.i.2)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.

My implementation uses a preservation of the leftmost bit on the input for arithmetic shifting. For logical shifting I used rows of multiplexores to perform the shifting operations inputting zero for the buffer bits

[Part 2 (c.i.3)] In your writeup, explain how the right barrel shifter above can be enhanced to also support left shifting operations.

It can be enhanced by adding rows of multiplexores before and after the right shifter. These will reverse the input and output of the shifter

[Part 2 (c.i.4)] Describe how the execution of the different shifting operations corresponds to the QuestaSim waveforms in your writeup.



Within the waveform above following the select signal is easy to determine when we are changing the shifting function. The select signal controls which rows of muxes are activated during the shifting, as well as the direction signal shows which way our shifter is shifting.

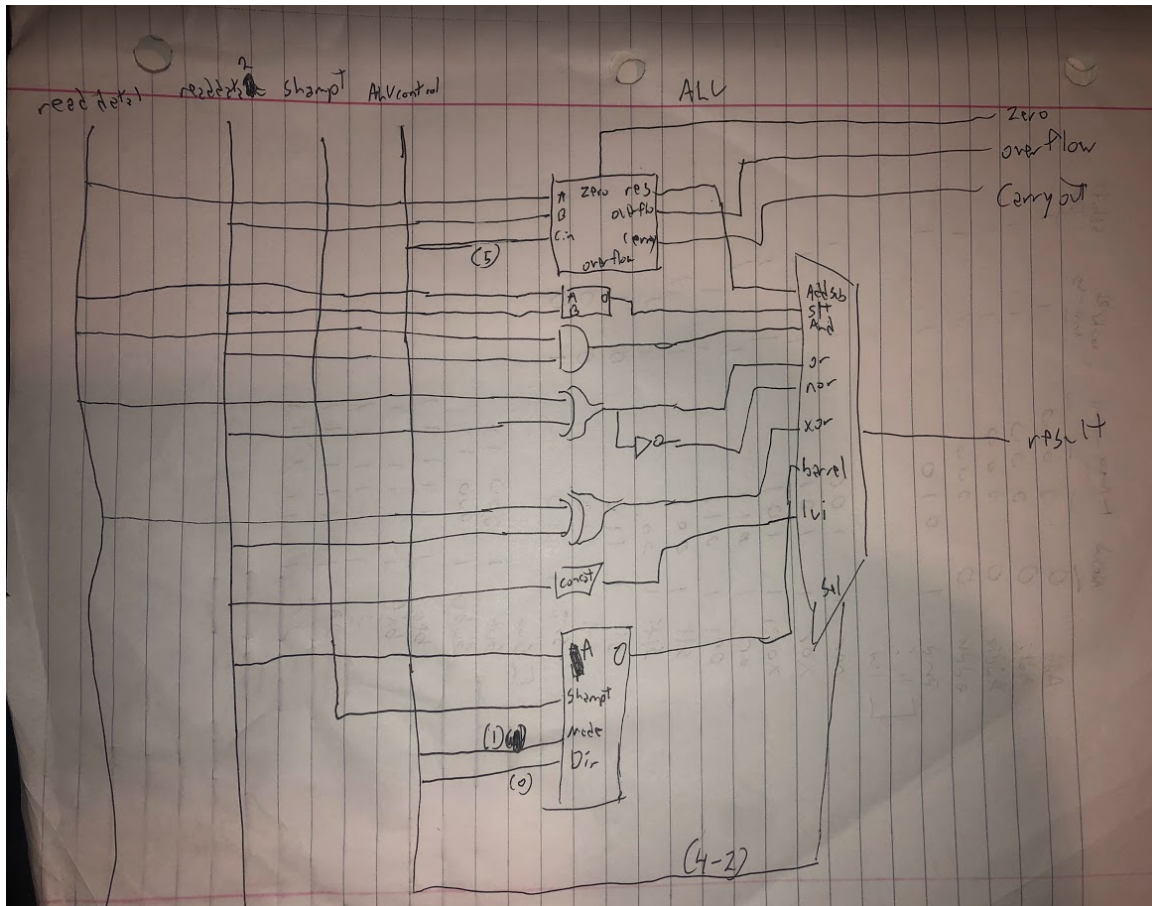
[Part 2 (c.ii.1)] In your writeup, briefly describe your design approach, including any resources you used to choose or implement the design. Include at least one design decision you had to make.

I had to make an N bit and, or, slt, xor, and a multiplexor to mux all of my different instructions the ALU implements. I chose to give the input signals to all of the modules and then decide which one gets outputted after with a multiplexor.

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

See ALU waveform. The waveform shows that by changing the ALUcontrol signal it affects what gets outputted of the ALU. If I were to simply change the control signal it would change the operation the processor is performing during that cycle.

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

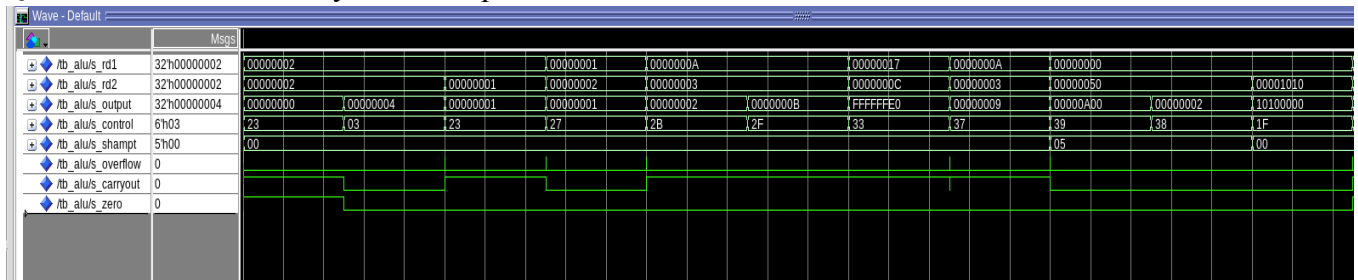


Overflow is calculated within the full adder. You xor the top and second to top bit of the output. Then you just pass it back up through the modules to the overflow output of the ALU.

Zero is calculated by subtracting the first and second input and checking whether the result is zero or not. If it is zero then the zero output is one and if it is not zero then the zero output is zero.

SLT is implemented by a N-bit SLT custom module. Within this module the inputs get fed through an adder/subtractor module that subtracts the inputs and outputs the top bit of the output.

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



When you see the control signal line switch that is your best indicator of a switch of functionality within the ALU. This line controls what module within the ALU is getting

access to the output line. As you can see by the changing out the output signal in the waveform.

[Part 2 (c.viii)] justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.

The waveform above shows that the testing plan is comprehensive because it tests all of the modules within the ALU for edge cases. This includes the zero, overflow, and carryout functionalities.

[Part 3] In your writeup, show the QuestaSim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

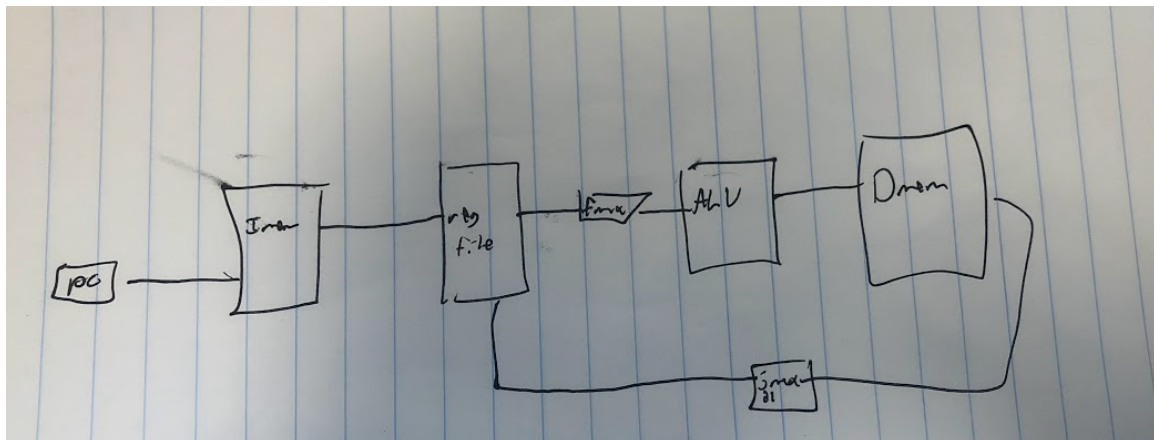
[Part 3 (a)] Create a test application that makes use of every required arithmetic/logical instruction at least once. The application need not perform any particularly useful task, but it should demonstrate the full functionality of the processor (e.g., sequences of many instructions executed sequentially, 1 per cycle while data written into registers can be effectively retrieved and used by later instructions). Name this file Proj1_base_test.s.

[Part 3 (b)] Create and test an application which uses each of the required control-flow instructions and has a call depth of at least 5 (i.e., the number of activation records on the stack is at least 4). Name this file Proj1_cf_test.s.

[Part 3 (c)] Create and test an application that sorts an array with N elements using the BubbleSort algorithm ([link](#)). Name this file Proj1_bubblesort.s.

[Part 4] Report the maximum frequency your processor can run at and determine what your critical path is. Draw this critical path on top of your top-level schematic from part 1. What components would you focus on to improve the frequency?

27.96mHz



I would want to focus on the register file and ALU of my processor to improve my frequency