CprE 381: Computer Organization and Assembly-Level Programming

Project Part 1 Report

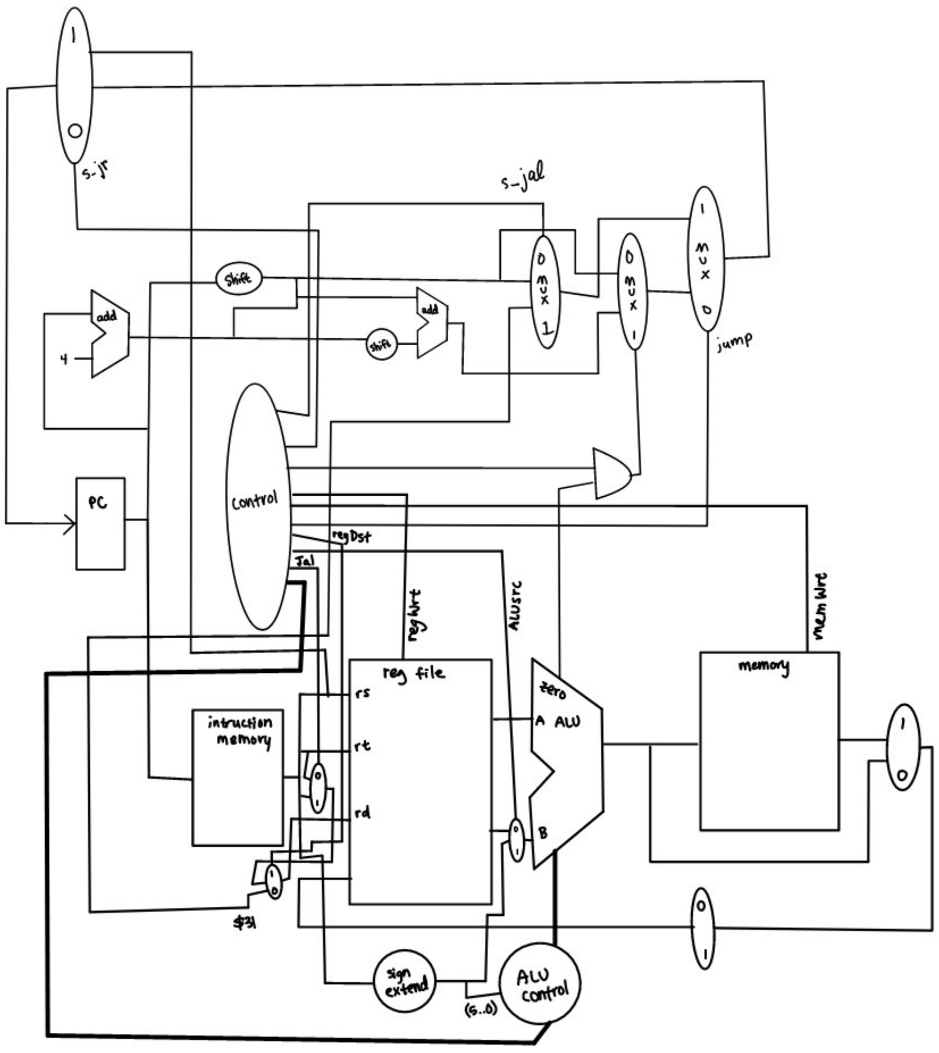
Team Members: _______Sabrina Francis & Cameron Mesman_______

_(Originally Brandon Poe as well, but he dropped class)__

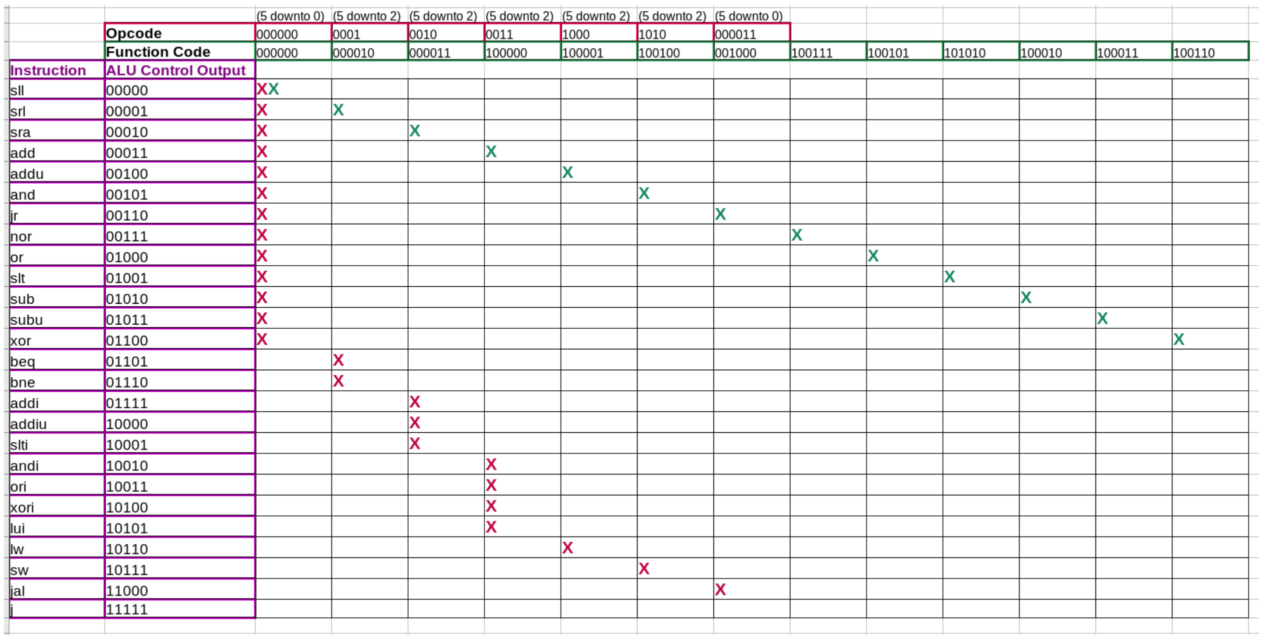
Project Teams Group #:__Proj1_7_4__

Refer to the highlighted language in the project 1 instruction for the context of the following questions.

[Part 1 (d)] Include your final MIPS processor schematic in your lab report.

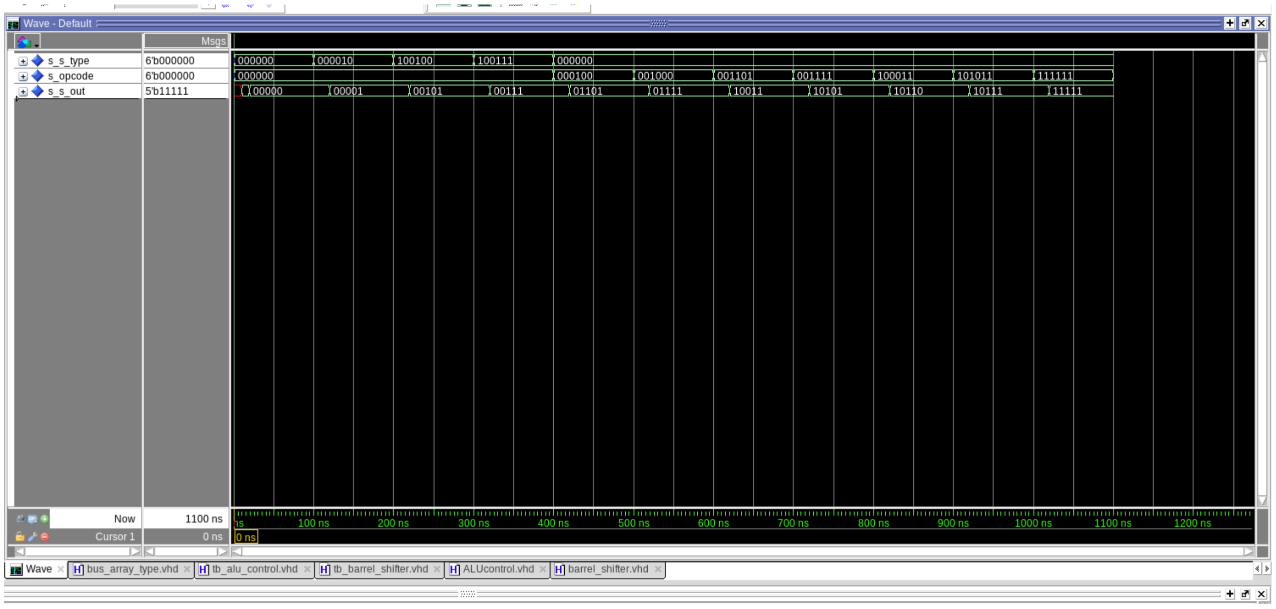


[Part 2 (a.i)] Create a spreadsheet detailing the list of *M* instructions to be supported in your project alongside their binary opcodes and funct fields, if applicable. Create a separate column for each binary bit. Inside this spreadsheet, create a new column for the *N* control signals needed by your datapath implementation. The end result should be an *N*M* table where each row corresponds to the output of the control logic module for a given instruction.



		(5 downto 0)	(5 downto 2)	(5 downto 2)	(5 downto 2)	(5 downto 2)	(5 downto 2)	(5 downto 0)						
	Opcode	000000	0001	0010	0011	1000	1010	000011						
	Function Code	000000	000010	000011	100000	100001	100100	001000	100111	100101	101010	100010	100011	100110
Instruction	**ALU Control Output**													
sll	00000	XX												
srl	00001	X	X											
sra	00010	X		X										
add	00011	X			X									
addu	00100	X				X								
and	00101	X					X							
jr	00110	X						X						
nor	00111	X							X					
or	01000	X								X				
slt	01001	X									X			
sub	01010	X										X		
subu	01011	X											X	
xor	01100	X												X
beq	01101		X											
bne	01110		X											
addi	01111			X										
addiu	10000			X										
slti	10001			X										
andi	10010				X									
ori	10011				X									
xori	10100				X									
lui	10101				X									
lw	10110					X								
sw	10111						X							
jal	11000							X						
j	11111													

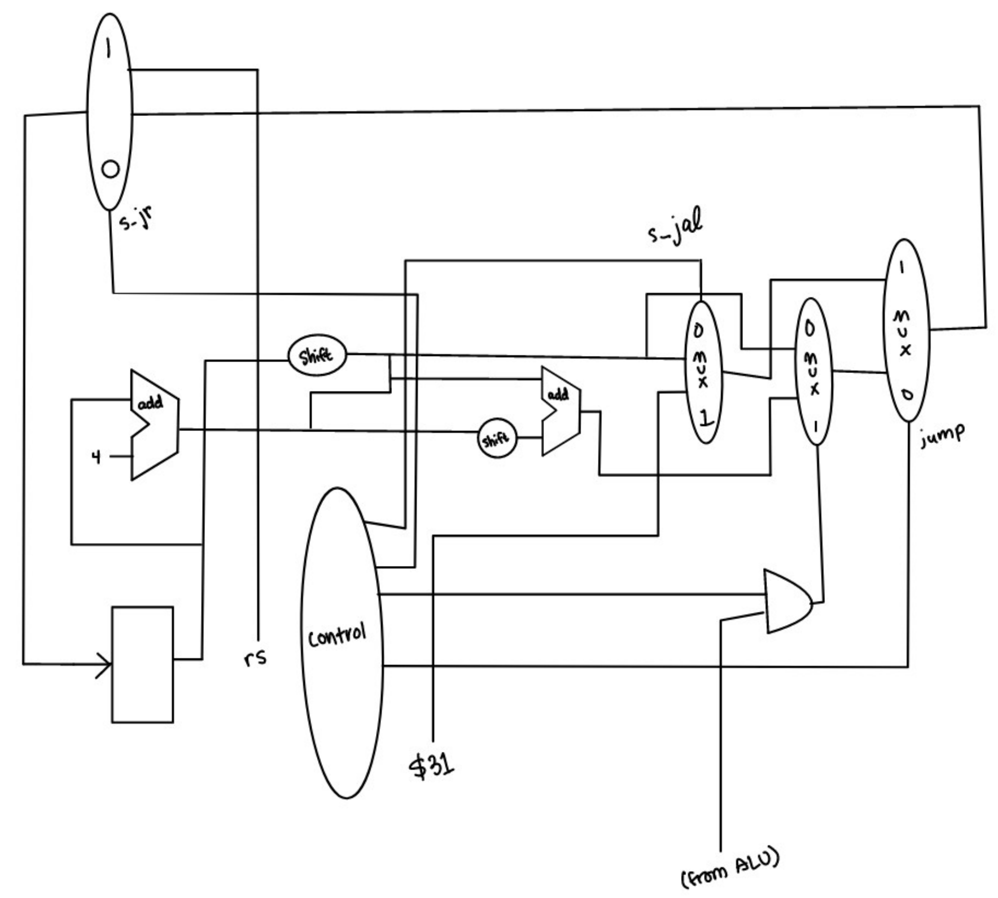
[Part 2 (a.ii)] Implement the control logic module using whatever method and coding style you prefer. Create a testbench to test this module individually, and show that your output matches the expected control signals from problem 1(a).



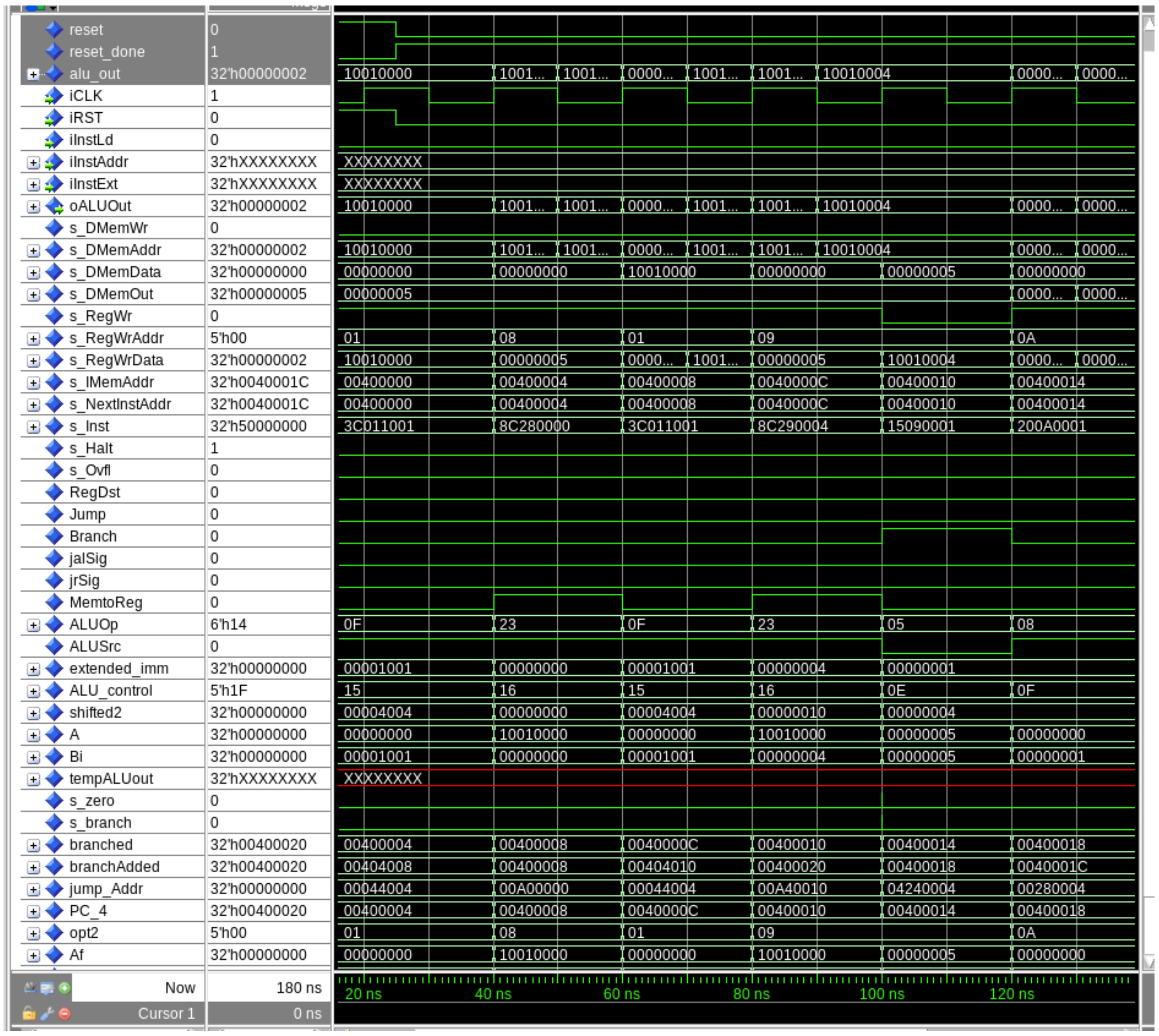
```
s_s_type    6'b000000   000000  000010  100100  100111  000000
s_opcode    6'b000000   000000                          000100  001000  001101  001111  100011  101011  111111
s_s_out     5'b11111    00000   00001   00101   00111   01101   01111   10011   10101   10110   10111   11111

Now     1100 ns
Cursor 1   0 ns

ns  100 ns  200 ns  300 ns  400 ns  500 ns  600 ns  700 ns  800 ns  900 ns  1000 ns  1100 ns  1200 ns
0 ns
```

The waveform above shows the two inputs and the output of our ALU control. s_s_type represents the function code and s_opcode represents the opcode. Each output matches our expected output values. The expected values can be seen in the testbench file.

[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.
Fetch logic must support a branch function, jump instruction, and PC+4 to increment all other instructions. There will be 3 types of control for the fetch logic, branch, jump, and regular increment. Also a halt instruction to stop the program.
[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?



[Part 2 (b.iii)] Implement your new instruction fetch logic using VHDL. Use Modelsim 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 Modelsim waveforms in your writeup.



reset	0
reset_done	1
alu_out	32'h00000002
iCLK	1
iRST	0
iInstLd	0
iInstAddr	32'hXXXXXXXX
iInstExt	32'hXXXXXXXX
oALUOut	32'h00000002
s_DMemWr	0
s_DMemAddr	32'h00000002
s_DMemData	32'h00000000
s_DMemOut	32'h00000005
s_RegWr	0
s_RegWrAddr	5'h00
s_RegWrData	32'h00000002
s_IMemAddr	32'h0040001C
s_NextInstAddr	32'h0040001C
s_Inst	32'h50000000
s_Halt	1
s_Ovfl	0
RegDst	0
Jump	0
Branch	0
jalSig	0
jrSig	0
MemtoReg	0
ALUOp	6'h14
ALUSrc	0
extended_imm	32'h00000000
ALU_control	5'h1F
shifted2	32'h00000000
A	32'h00000000
Bi	32'h00000000
tempALUout	32'hXXXXXXXX
s_zero	0
s_branch	0
branched	32'h00400020
branchAdded	32'h00400020
jump_Addr	32'h00000000
PC_4	32'h00400020
opt2	5'h00
Af	32'h00000000
Now	180 ns
Cursor 1	0 ns

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

Srl shifts the input right and shifts in zeros. Sra shifts the input right but shifts in the sign bit. There is no sla instruction because shifting in a sign bit to the least significant bit is not helpful and actually changes the value of the input, so it would have almost no uses.

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

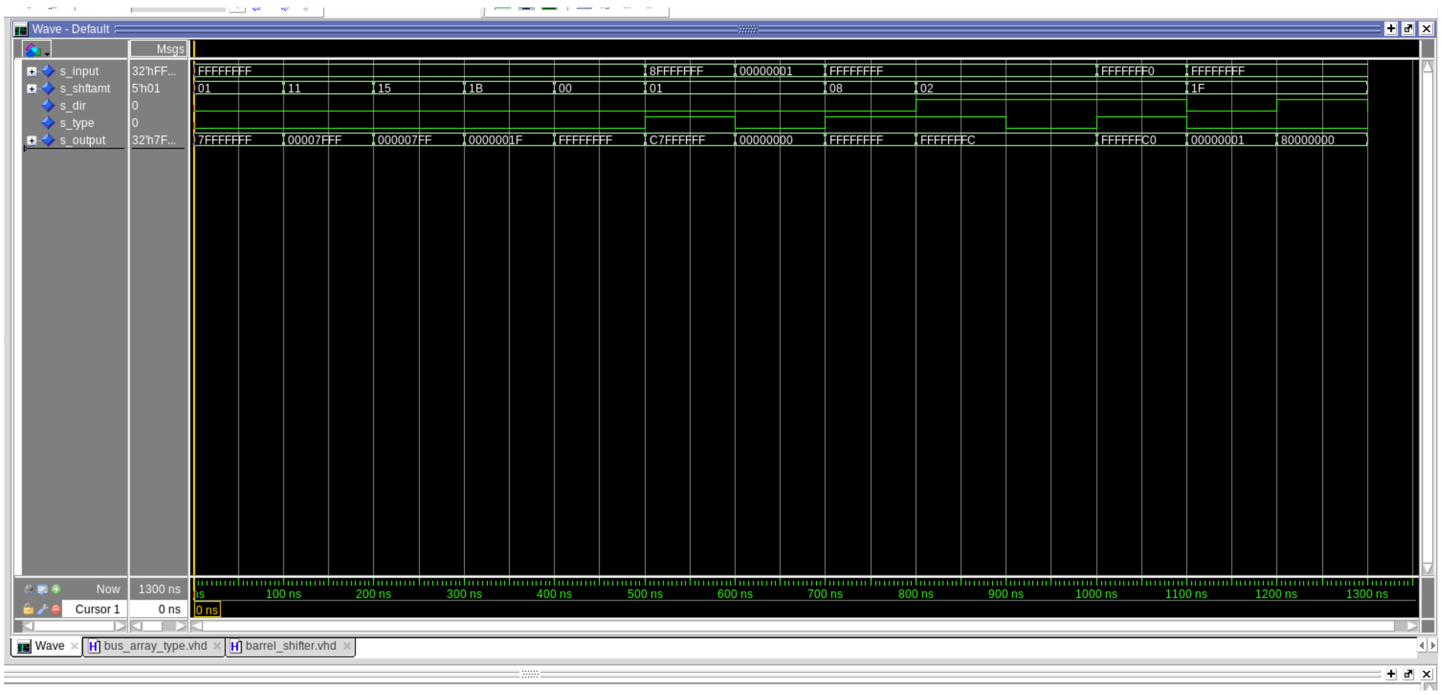
The barrel shifter component stores the value of the most significant bit (sign bit). Then, when the input is shifted and new bits are shifted in, the component uses the stored sign bit as the shift in bit.

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

To implement a left barrel shifter using the right barrel shifter, you can do the following:

1. reverse input
2. run right barrel shifter as ususal
3. reverse output

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


Wave - Default
Msgs
s_input 32'hFF... FFFFFFFF 8FFFFFFF 00000001 FFFFFFFF FFFFFFF0 FFFFFFFF
s_shftamt 5'h01 01 11 15 1B 00 01 08 02 1F
s_dir 0
s_type 0
s_output 32'h7F... 7FFFFFFF 00007FFF 000007FF 0000001F FFFFFFFF C7FFFFFF 00000000 FFFFFFFF FFFFFFFC FFFFFFC0 00000001 80000000
Now 1300 ns
Cursor 1 0 ns
100 ns 200 ns 300 ns 400 ns 500 ns 600 ns 700 ns 800 ns 900 ns 1000 ns 1100 ns 1200 ns 1300 ns
Wave
bus_array_type.vhd
barrel_shifter.vhd

This waveform shows the input and output of 13 different test cases. The first 5 cases show the srl command, shifting the input right and shifting in zeros. The s_shftamt shows how many bits the input is being shifted by, and s_output shows the value after being shifted. The 6th and 8th test cases show sra which shifts in the sign bit. Lastly, examples 9-11 show sll, which shifts the input left.

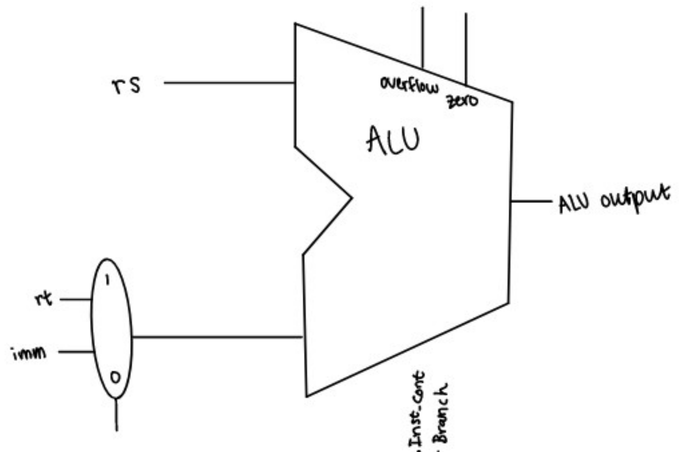
[Part 2 (c.ii.1)] In your writeup, briefly describe your design approach, including any resources you used to choose or implement the design. Include at least one design decision you had to make.

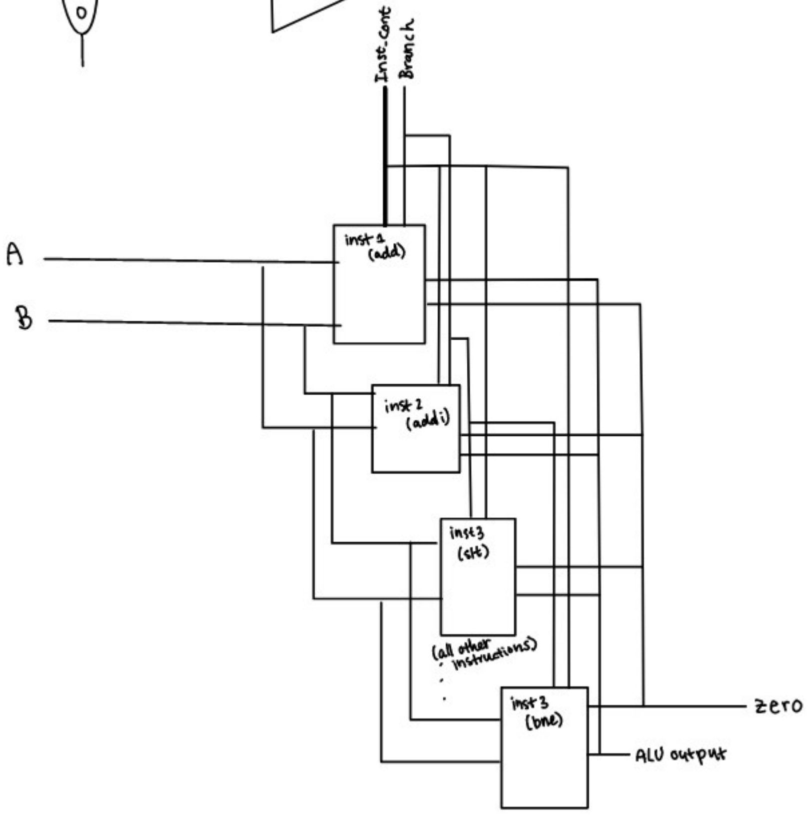
For our design approach, we changed our ALUControl to output a different signal for every instruction, and make a different calculation for every instruction in the ALU based on the inputted control signal. We decided to do this so that We could choose some of our signals within the ALU, instead of adding more to the big control unit.

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

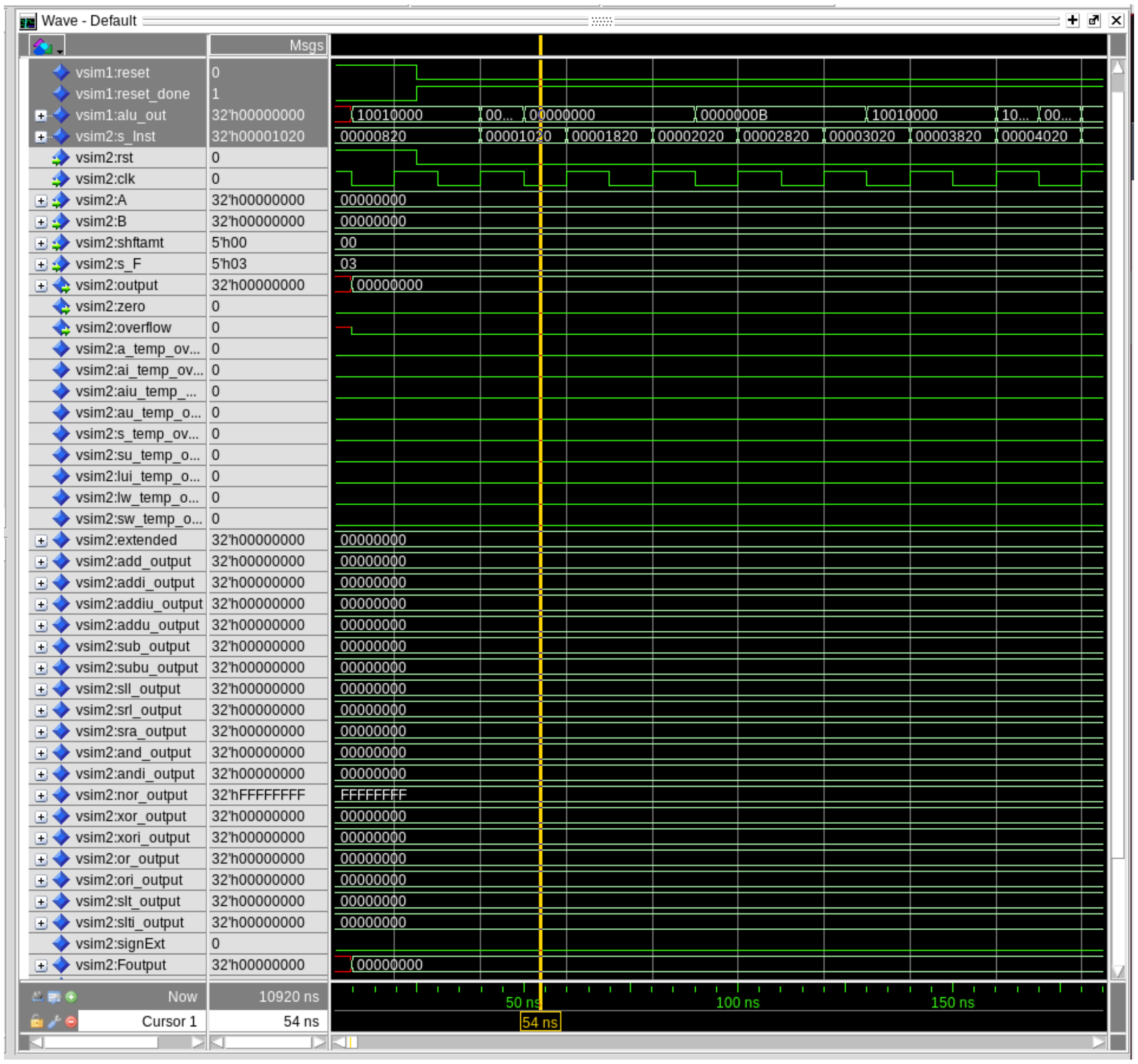
Depending on the instruction, there will be different instruction machine code, control signals, flow, and outputs in the waveforms.

[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?





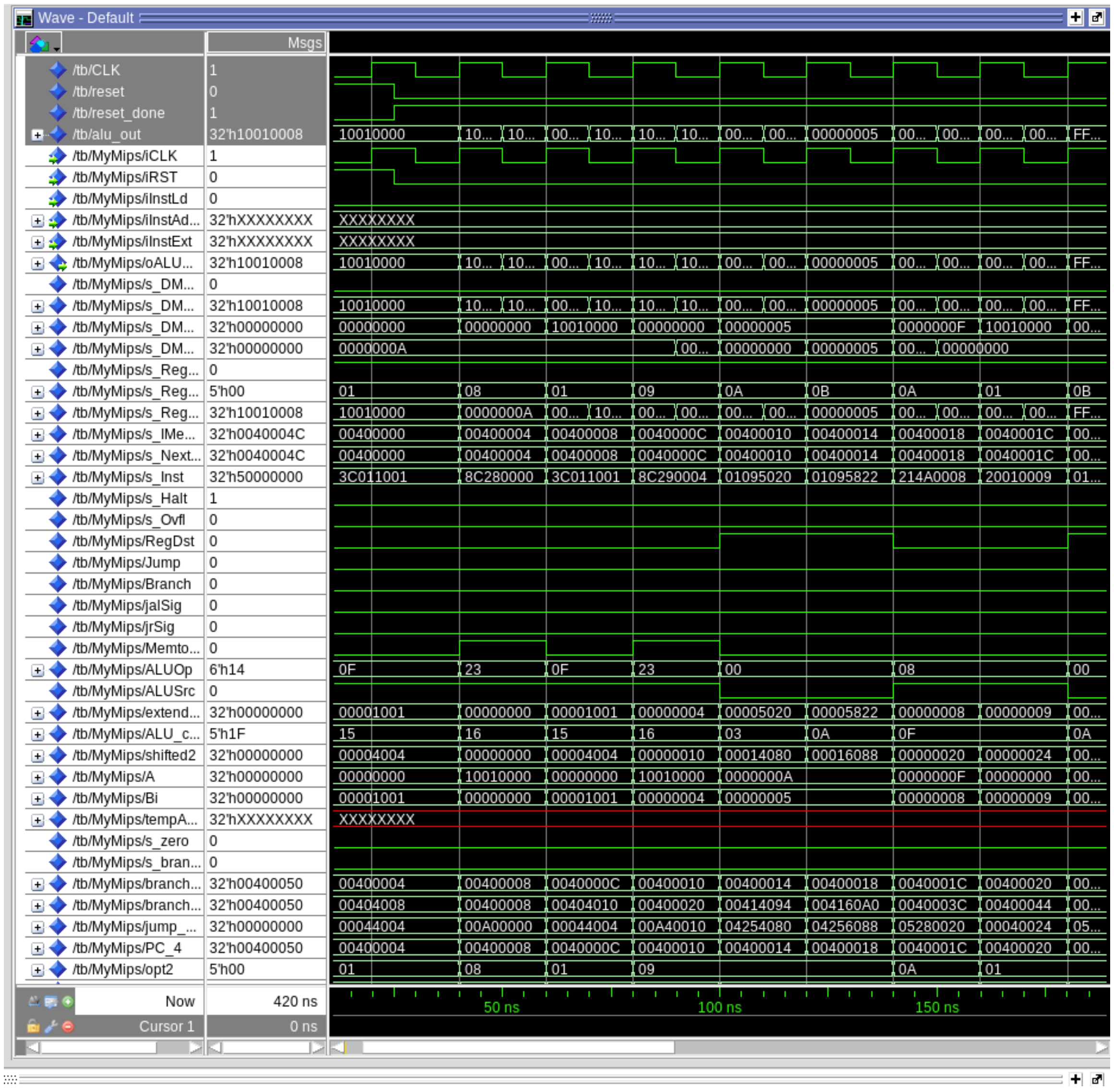
[Part 2 (c.v)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.
Based on the different instructions, the ALU will output a different branch fetch signal, ALU output, and overflow. It will also intake different
[Part 2 (c.viii)] justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.



Signal	Msgs
vsim1:reset	0
vsim1:reset_done	1
vsim1:alu_out	32'h00000000
vsim2:s_Inst	32'h00001020
vsim2:rst	0
vsim2:clk	0
vsim2:A	32'h00000000
vsim2:B	32'h00000000
vsim2:shftamt	5'h00
vsim2:s_F	5'h03
vsim2:output	32'h00000000
vsim2:zero	0
vsim2:overflow	0
vsim2:a_temp_ov...	0
vsim2:ai_temp_ov...	0
vsim2:aiu_temp_...	0
vsim2:au_temp_o...	0
vsim2:s_temp_ov...	0
vsim2:su_temp_o...	0
vsim2:lui_temp_o...	0
vsim2:lw_temp_o...	0
vsim2:sw_temp_o...	0
vsim2:extended	32'h00000000
vsim2:add_output	32'h00000000
vsim2:addi_output	32'h00000000
vsim2:addiu_output	32'h00000000
vsim2:addu_output	32'h00000000
vsim2:sub_output	32'h00000000
vsim2:subu_output	32'h00000000
vsim2:sll_output	32'h00000000
vsim2:srl_output	32'h00000000
vsim2:sra_output	32'h00000000
vsim2:and_output	32'h00000000
vsim2:andi_output	32'h00000000
vsim2:nor_output	32'hFFFFFFFF
vsim2:xor_output	32'h00000000
vsim2:xori_output	32'h00000000
vsim2:or_output	32'h00000000
vsim2:ori_output	32'h00000000
vsim2:slt_output	32'h00000000
vsim2:slti_output	32'h00000000
vsim2:signExt	0
vsim2:Foutput	32'h00000000
Now	10920 ns
Cursor 1	54 ns

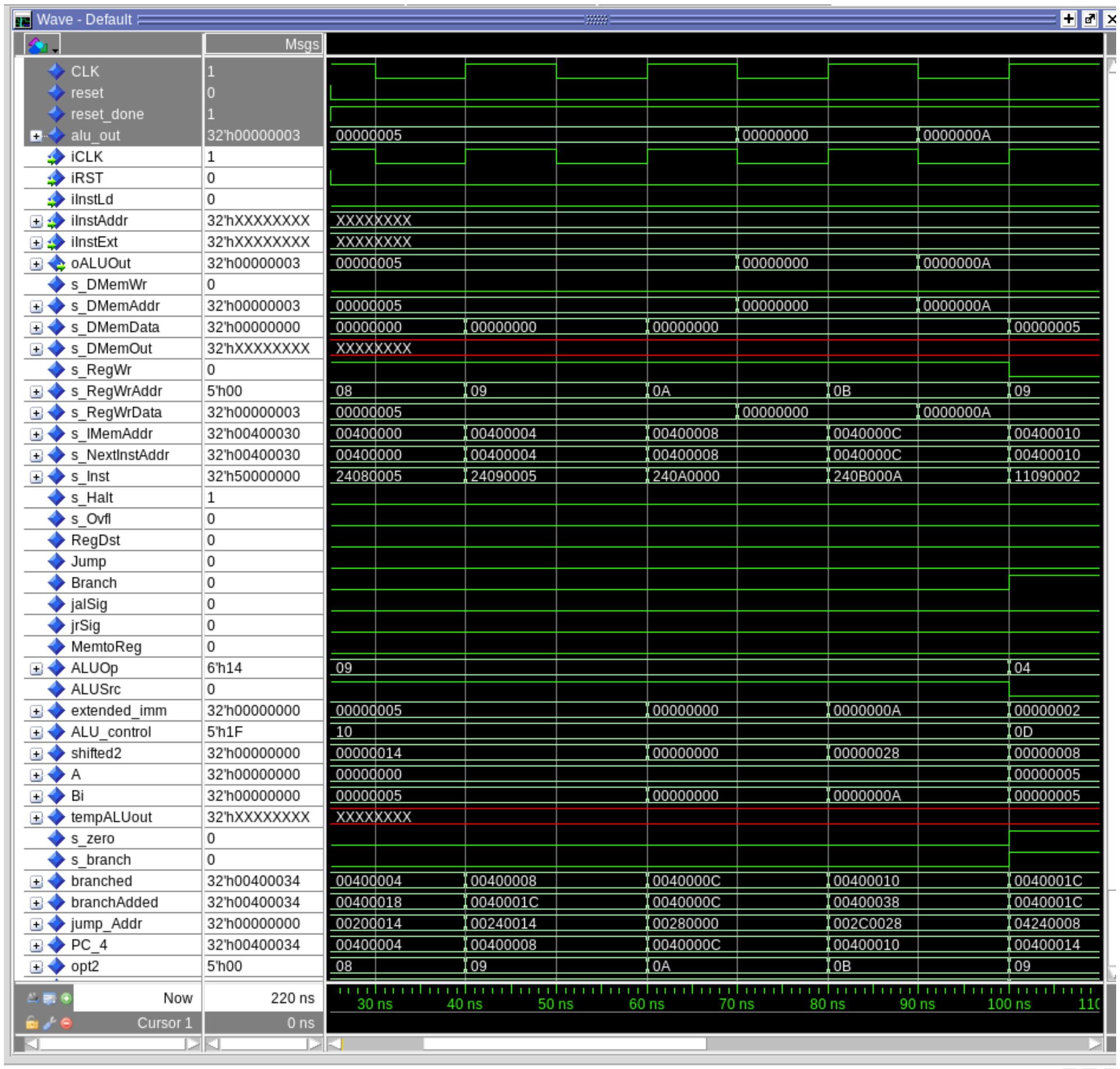
[Part 3] In your writeup, show the Modelsim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

[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.



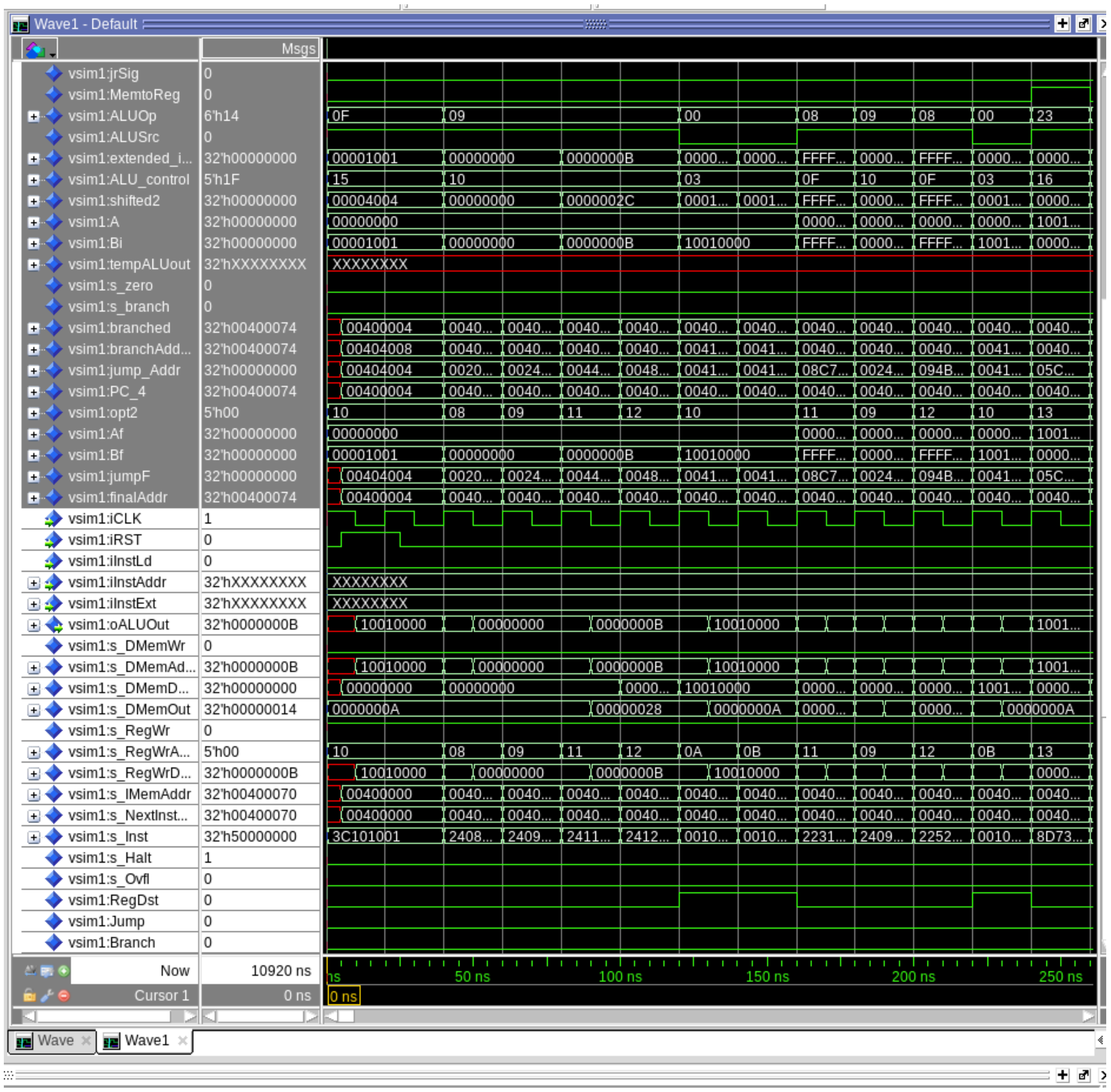
```
Wave - Default
                          Msgs
/tb/CLK                   1
/tb/reset                 0
/tb/reset_done            1
/tb/alu_out               32'h10010008   10010000 |10...|10...|00...|10...|10...|10...|00...|00...|00000005|00...|00...|00...|00...|FF...
/tb/MyMips/iCLK           1
/tb/MyMips/iRST           0
/tb/MyMips/iInstLd        0
/tb/MyMips/iInstAd...     32'hXXXXXXXX   XXXXXXXX
/tb/MyMips/iInstExt       32'hXXXXXXXX   XXXXXXXX
/tb/MyMips/oALU...        32'h10010008   10010000 |10...|10...|00...|10...|10...|10...|00...|00...|00000005|00...|00...|00...|00...|FF...
/tb/MyMips/s_DM...        0
/tb/MyMips/s_DM...        32'h10010008   10010000 |10...|10...|00...|10...|10...|10...|00...|00...|00000005|00...|00...|00...|00...|FF...
/tb/MyMips/s_DM...        32'h00000000   00000000 |00000000|10010000|00000000|00000005|0000000F|10010000|00...
/tb/MyMips/s_DM...        32'h00000000   0000000A |00...|00000000|00000005|00...|00000000
/tb/MyMips/s_Reg...       0
/tb/MyMips/s_Reg...       5'h00          01 |08 |01 |09 |0A |0B |0A |01 |0B
/tb/MyMips/s_Reg...       32'h10010008   10010000 |0000000A|00...|10...|00...|00...|00...|00...|00000005|00...|00...|00...|00...|FF...
/tb/MyMips/s_IMe...       32'h0040004C   00400000 |00400004|00400008|0040000C|00400010|00400014|00400018|0040001C|00...
/tb/MyMips/s_Next...      32'h0040004C   00400000 |00400004|00400008|0040000C|00400010|00400014|00400018|0040001C|00...
/tb/MyMips/s_Inst         32'h50000000   3C011001 |8C280000|3C011001|8C290004|01095020|01095822|214A0008|20010009|01...
/tb/MyMips/s_Halt         1
/tb/MyMips/s_Ovfl         0
/tb/MyMips/RegDst         0
/tb/MyMips/Jump           0
/tb/MyMips/Branch         0
/tb/MyMips/jalSig         0
/tb/MyMips/jrSig          0
/tb/MyMips/Memto...       0
/tb/MyMips/ALUOp          6'h14          0F |23 |0F |23 |00 |08 |00
/tb/MyMips/ALUSrc         0
/tb/MyMips/extend...      32'h00000000   00001001 |00000000|00001001|00000004|00005020|00005822|00000008|00000009|00...
/tb/MyMips/ALU_c...       5'h1F          15 |16 |15 |16 |03 |0A |0F |0A
/tb/MyMips/shifted2       32'h00000000   00004004 |00000000|00004004|00000010|00014080|00016088|00000020|00000024|00...
/tb/MyMips/A              32'h00000000   00000000 |10010000|00000000|10010000|0000000A|0000000F|00000000|00...
/tb/MyMips/Bi             32'h00000000   00001001 |00000000|00001001|00000004|00000005|00000008|00000009|00...
/tb/MyMips/tempA...       32'hXXXXXXXX   XXXXXXXX
/tb/MyMips/s_zero         0
/tb/MyMips/s_bran...      0
/tb/MyMips/branch...      32'h00400050   00400004 |00400008|0040000C|00400010|00400014|00400018|0040001C|00400020|00...
/tb/MyMips/branch...      32'h00400050   00404008 |00400008|00404010|00400020|00414094|004160A0|0040003C|00400044|00...
/tb/MyMips/jump_...       32'h00000000   00044004 |00A00000|00044004|00A40010|04254080|04256088|05280020|00040024|05...
/tb/MyMips/PC_4           32'h00400050   00400004 |00400008|0040000C|00400010|00400014|00400018|0040001C|00400020|00...
/tb/MyMips/opt2           5'h00          01 |08 |01 |09 |0A |01

Now       420 ns                  50 ns          100 ns          150 ns
Cursor 1  0 ns
```

[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.



	Msgs
CLK	1
reset	0
reset_done	1
alu_out	32'h00000003
iCLK	1
iRST	0
iInstLd	0
iInstAddr	32'hXXXXXXXX
iInstExt	32'hXXXXXXXX
oALUOut	32'h00000003
s_DMemWr	0
s_DMemAddr	32'h00000003
s_DMemData	32'h00000000
s_DMemOut	32'hXXXXXXXX
s_RegWr	0
s_RegWrAddr	5'h00
s_RegWrData	32'h00000003
s_IMemAddr	32'h00400030
s_NextInstAddr	32'h00400030
s_Inst	32'h50000000
s_Halt	1
s_Ovfl	0
RegDst	0
Jump	0
Branch	0
jalSig	0
jrSig	0
MemtoReg	0
ALUOp	6'h14
ALUSrc	0
extended_imm	32'h00000000
ALU_control	5'h1F
shifted2	32'h00000000
A	32'h00000000
Bi	32'h00000000
tempALUout	32'hXXXXXXXX
s_zero	0
s_branch	0
branched	32'h00400034
branchAdded	32'h00400034
jump_Addr	32'h00000000
PC_4	32'h00400034
opt2	5'h00
Now	220 ns
Cursor 1	0 ns

[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.


Wave1 - Default

	Msgs
vsim1:jrSig	0
vsim1:MemtoReg	0
vsim1:ALUOp	6'h14
vsim1:ALUSrc	0
vsim1:extended_i...	32'h00000000
vsim1:ALU_control	5'h1F
vsim1:shifted2	32'h00000000
vsim1:A	32'h00000000
vsim1:Bi	32'h00000000
vsim1:tempALUout	32'hXXXXXXXX
vsim1:s_zero	0
vsim1:s_branch	0
vsim1:branched	32'h00400074
vsim1:branchAdd...	32'h00400074
vsim1:jump_Addr	32'h00000000
vsim1:PC_4	32'h00400074
vsim1:opt2	5'h00
vsim1:Af	32'h00000000
vsim1:Bf	32'h00000000
vsim1:jumpF	32'h00000000
vsim1:finalAddr	32'h00400074
vsim1:iCLK	1
vsim1:iRST	0
vsim1:iInstLd	0
vsim1:iInstAddr	32'hXXXXXXXX
vsim1:iInstExt	32'hXXXXXXXX
vsim1:oALUOut	32'h0000000B
vsim1:s_DMemWr	0
vsim1:s_DMemAd...	32'h0000000B
vsim1:s_DMemD...	32'h00000000
vsim1:s_DMemOut	32'h00000014
vsim1:s_RegWr	0
vsim1:s_RegWrA...	5'h00
vsim1:s_RegWrD...	32'h0000000B
vsim1:s_IMemAddr	32'h00400070
vsim1:s_NextInst...	32'h00400070
vsim1:s_Inst	32'h50000000
vsim1:s_Halt	1
vsim1:s_Ovfl	0
vsim1:RegDst	0
vsim1:Jump	0
vsim1:Branch	0
Now	10920 ns
Cursor 1	0 ns

[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 schematics. What components would you focus on to improve the frequency?
FMax: 43.07mhz

PC → Imem → ADD_BRANCH → PC

I would focus on fetch logic, as it seems our branch instructions take the longest to run.