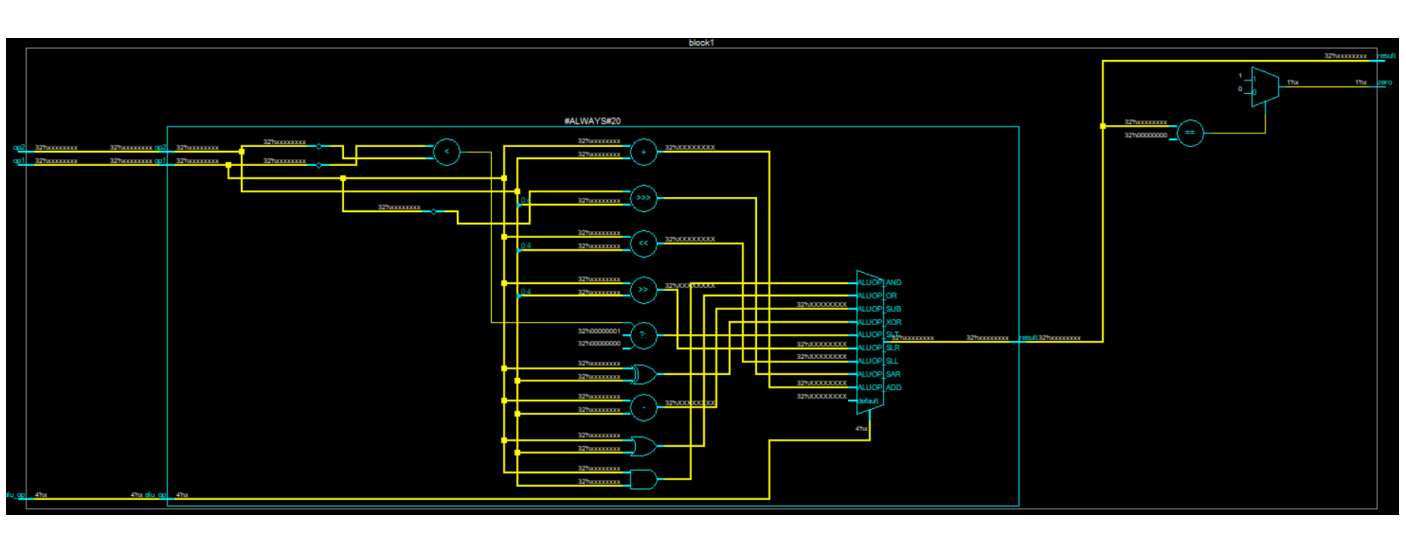
RISC - V PROCESSOR

EXERCISE 1

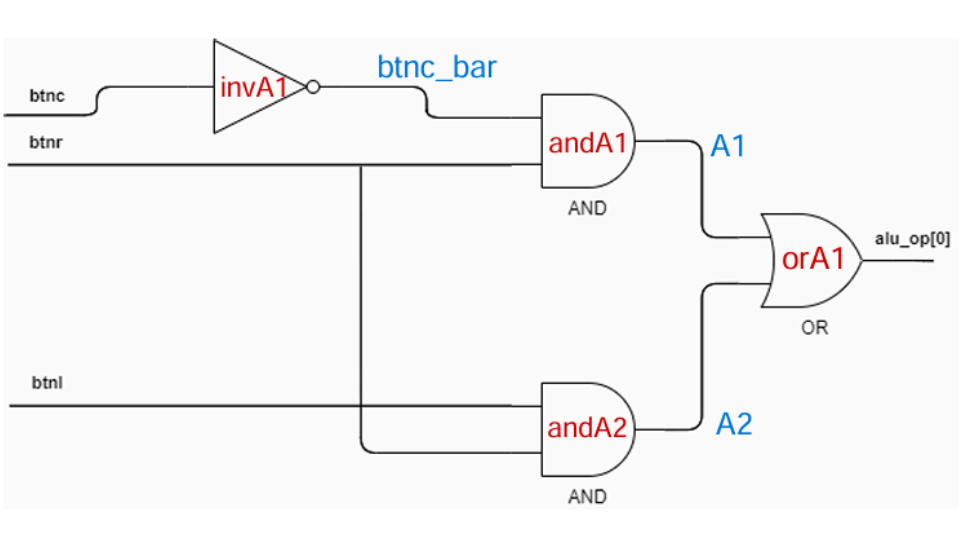
Module ALU : Module ALU: Implementation of a multiplexer that, based on alu_op, gives the desired result using a case statement. For the operations 'less than' and 'arithmetic shift', I use the 'signed()' and 'unsigned()' prefixes to convert to signed and unsigned respectively.
Note: The output zero of the ALU is active high, i.e., if result = 0 ⟹ zero = 1.
Below is the schematic diagram generated by Questa:



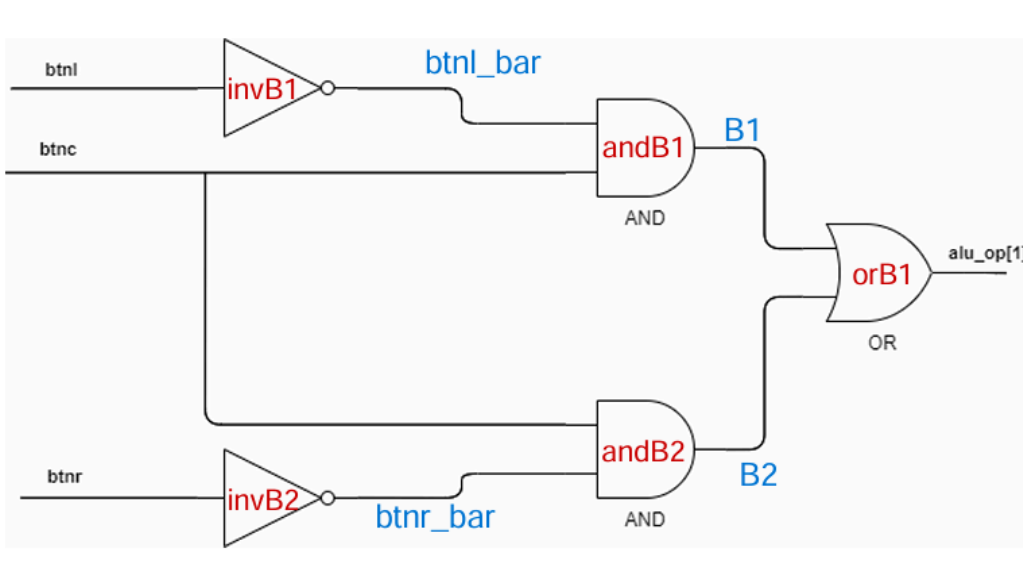
EXERCISE 2

1) module calc_enc: Implementation of the given circuits to generate alu_op using standard AND, OR, and NOT gates. In the circuits below, intermediate signals are shown in blue, used logic gates in red, and gates whose outputs are reused are in green.

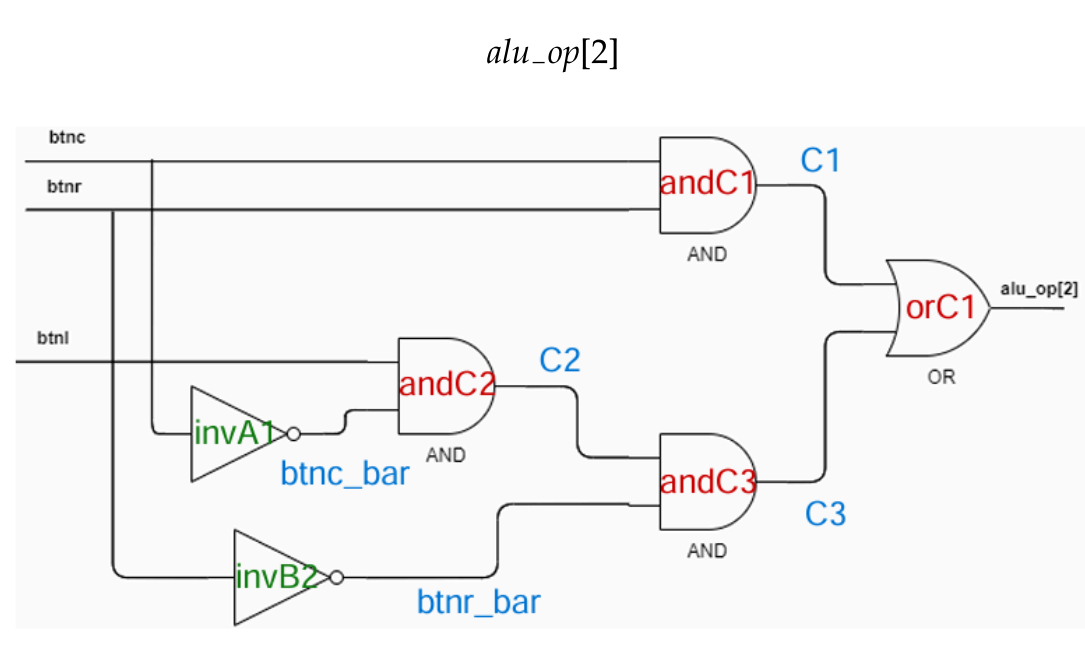
alu_op[0]



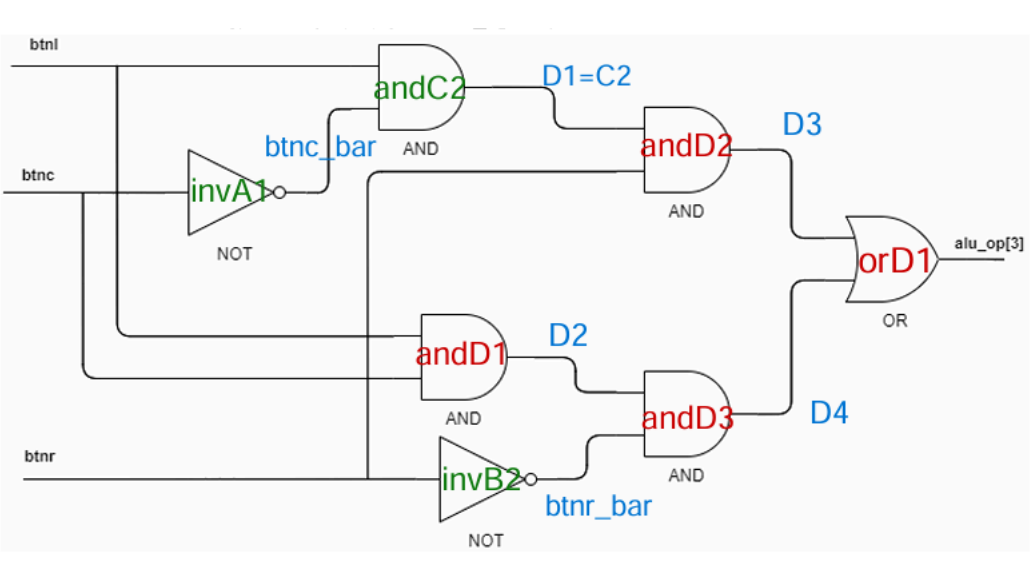
alu_op[1]



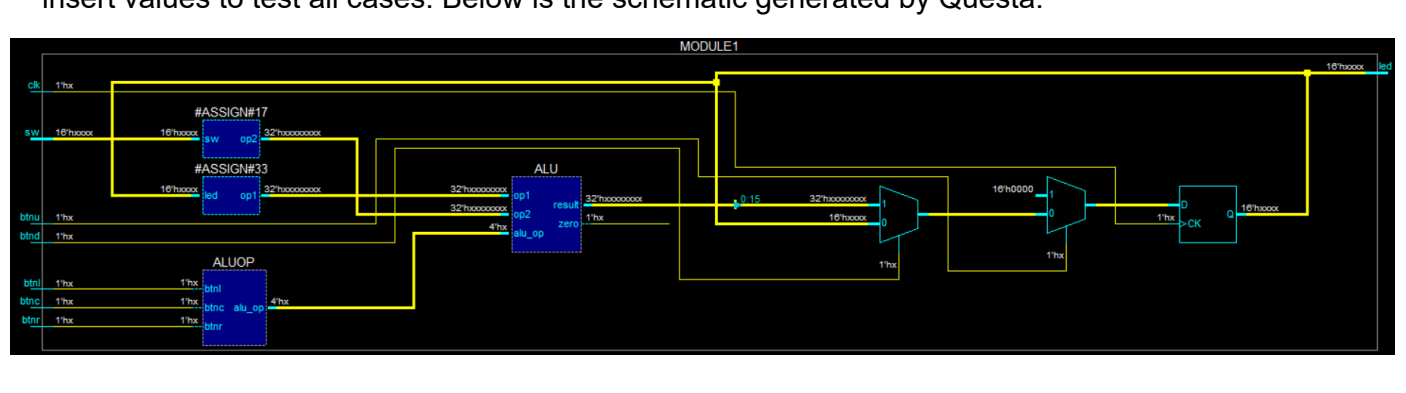
alu_op[2]



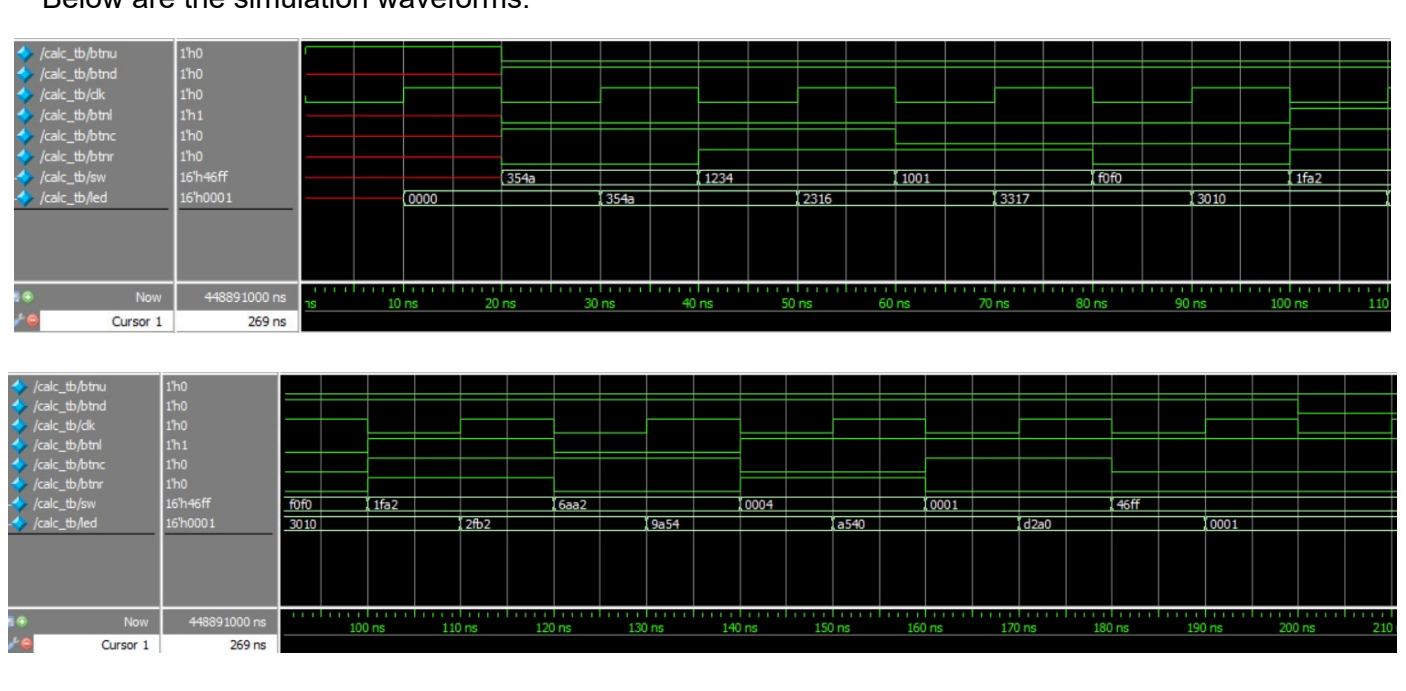
alu_op[3]



2) ***module calc*** :First I instantiate calc_enc and create the op1, op2 signals from the lw output and sw input respectively. Then I instantiate the alu and connect the alu_op port to the output of calc_enc and the op1, op2 ports to the signals created above. As for the Accumulator, I use an always block within which there's an if-else that assigns the correct value to "led" based on signals "btnd" and 'btnu'.
3) ***module calc_tb*** : To create the testbench, I define inputs as reg and outputs as wire. Then I instantiate the module calc. For the clock signal, I initialize clk to 1 (or 0) and with '#10 forever clk = ~ clk', where #10 ⟹ 10 ns delay (half clock period), forever ⟹ loop forever, and ~ ⟹ not (toggle 0→1 and 1→0), I generate a clock signal of 20 ns period. Finally, using the given input table, I insert values to test all cases. Below is the schematic generated by Questa:

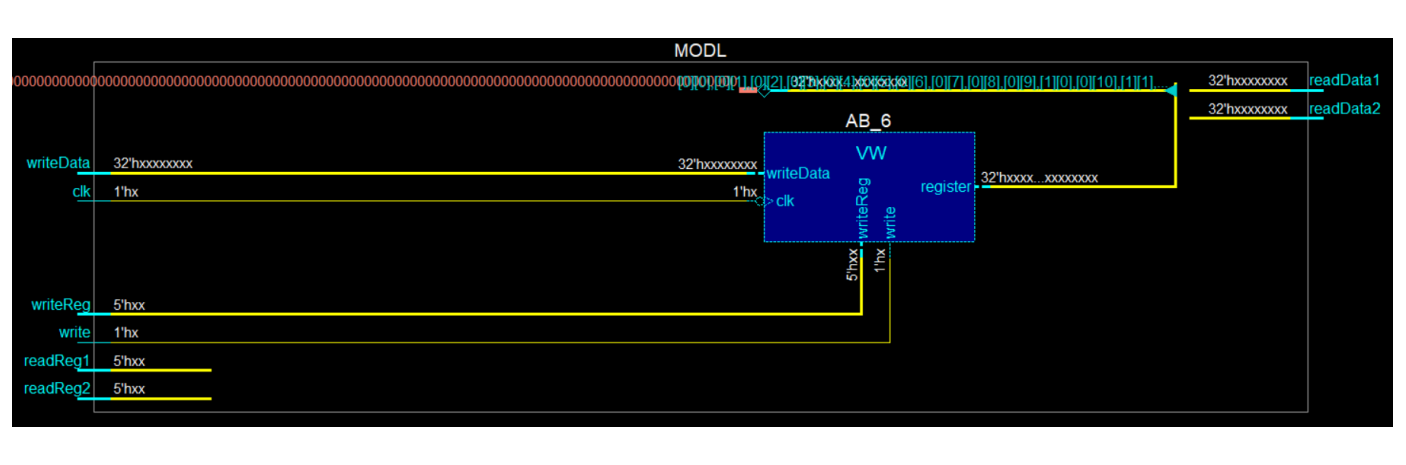


Below are the simulation waveforms:



EXERCISE 3

Module regfile : First, I introduce the parameter DATAWIDTH using #(parameter name = value). Then I create 32 registers of 32-bit each initialized to zero. To ensure writing occurs before reading, I use an if statement inside the always block to check for a write, and if true, it executes before reading. Below is the schematic generated by Questa:



ΑΣΚΗΣΗ 4

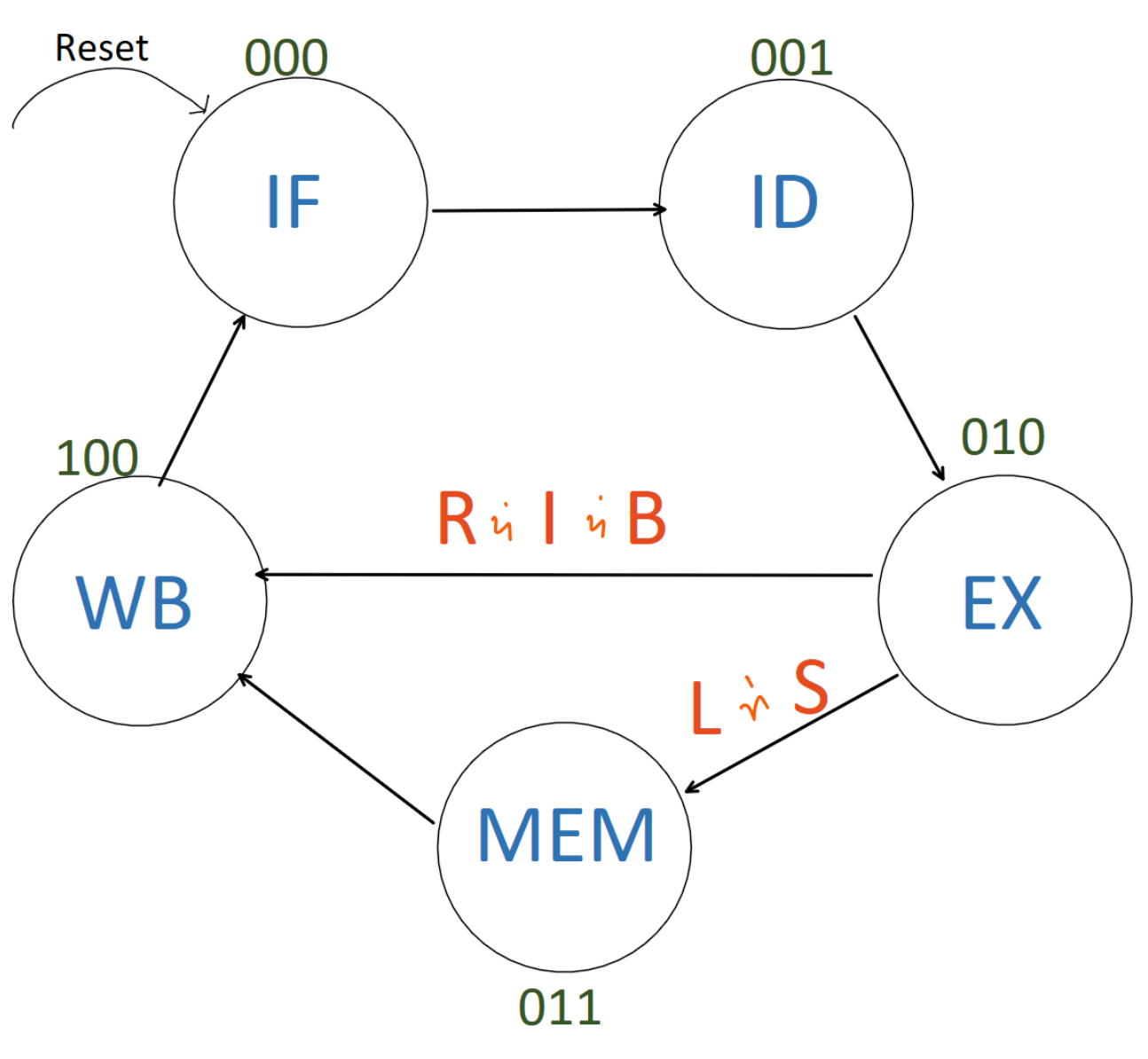
1) ***PC Block***: Initially, the reset signal rst is considered active high. An always block is used where first an if checks if rst is high and assigns INITIAL_PC to PC. Then another if checks if loadPC is high, and using an if-else block and PCSrc signal, assigns the appropriate value to PC based on branch instruction.
2) ***Instruction Decoder***: For simplicity, I define all possible outputs of the Instruction Decoder (ID) as parameters:

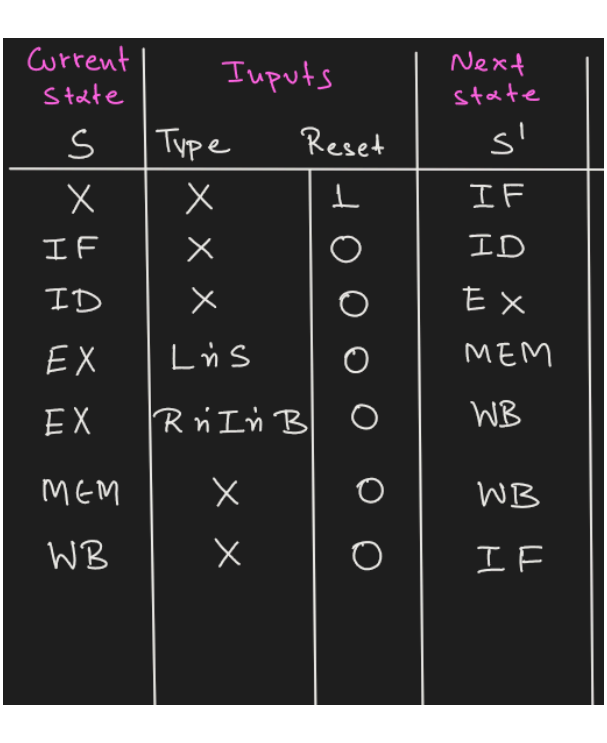
TYPE	ENCODING	INSTRUCTIONS	instr[6:0]
R_type	0→000	AND OR XOR SUB SLT SLA	7'b0110011
I_typr	1→001	ADI ORI XORI SUNI SLTI SLAI	7'b0100011
S_type	2→010	SW	7'b0000011
L_type	3→011	LW	7'b0010011
B_type	4→100	BEQ	7'b1100011

To find the instruction type, the 7 LSBs of instr are compared with the above table. These values are derived from the 7 LSBs of all instructions on page 148 of the RISC-V spec PDF.
3)) I instantiate the regfile and pass the appropriate inputs based on diagram 7 on page 11 of the assignment PDF..
4) ***Immediate Generator***: I use the instruction type output from the decoder and the table on page 148 of the RISC-V spec PDF to generate the appropriate immediate for the input instruction.
5) ***Branch Target***: As shown in diagram 7 on page 11 of the assignment PDF, by left-shifting the immediate by 1, we get the branch target.
6) ***MUX*** to select the second input of the ALU between immediate and RegRead2, selected by ALUSrc and implemented using a case statement.
7) ALU instantiation
8) ***MUX*** to select the write-back value between dReadData and ALU result based on MemtoReg signal.

ΑΣΚΗΣΗ 5

FSM DIAGRAM





Current state S	Inputs Type	Reset	Next state S'
X	X	1	IF
IF	X	0	ID
ID	X	0	EX
EX	L ή S	0	MEM
EX	R ή I ή B	0	WB
MEM	X	0	WB
WB	X	0	IF

top_proc :

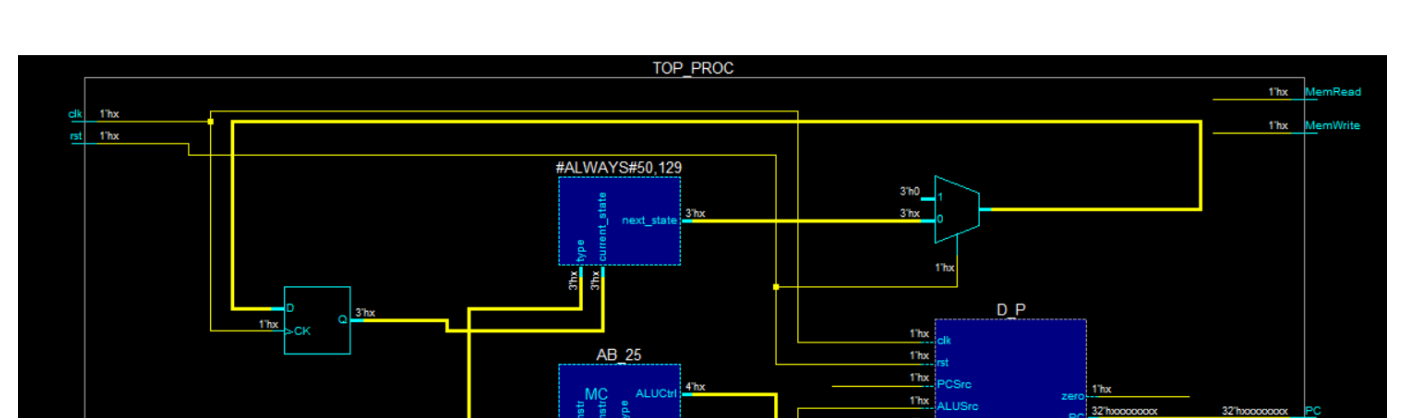
1) Instantiate the datapath.
2) Since reset is active high, when it becomes high, the system resets to state IF (000). If reset is low, state transitions occur based on the diagram/table. Each transition occurs on the next clock cycle. For ease, I define the 5 states as parameters and use a case statement inside an always block to perform the steps from the third column based on current_state. Current State Encoding Outputs:

Current State	Encoding	Outputs
IF	3'b000	-
ID	3'b001	-
EX	3'b010	-
MEM	3'b011	L⟹MemRead=1,MemWrite=0 S⟹MemRead=0,MemWrite=1
WB	3'b100	loadPC =1 B_type && zero=1⟹PCSrc=1 L⟹RegWrite=1,MemToReg=1 R⟹RegWrite=1,MemToReg=0 I⟹RegWrite=1,MemToReg=0

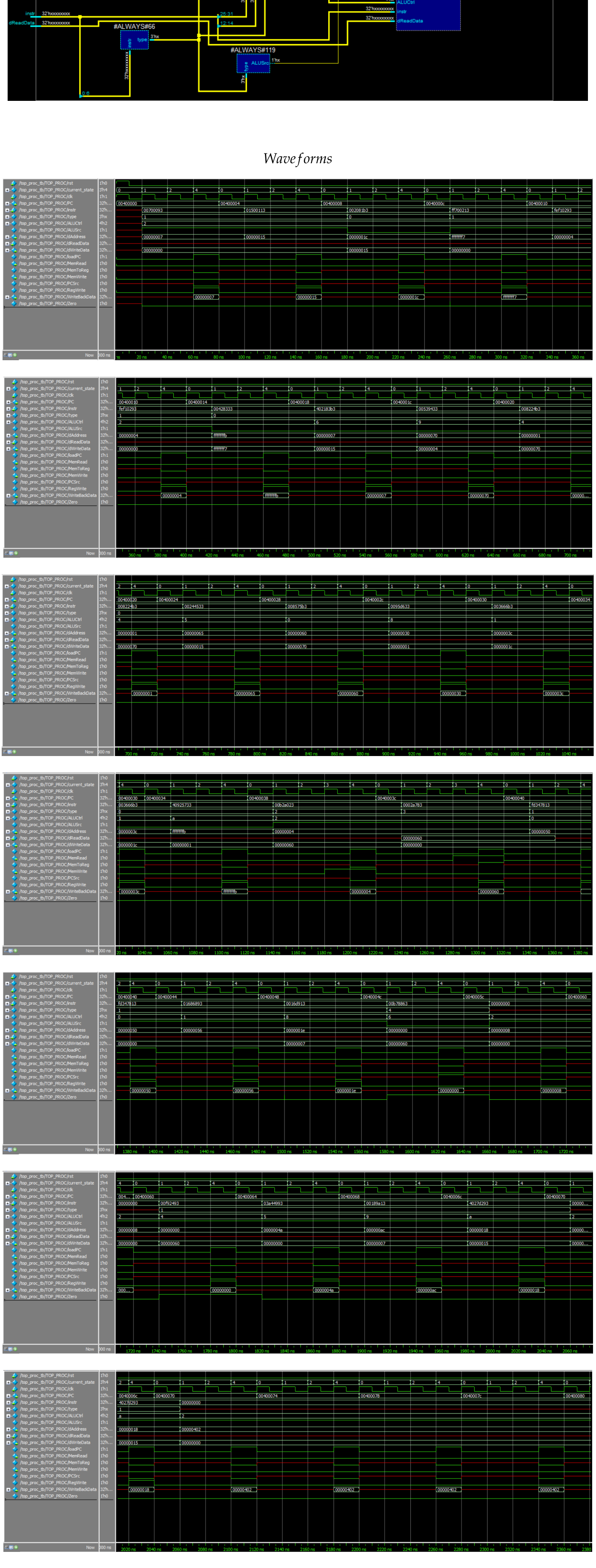
3)***ALUSrc*** : An if checks if the instruction is Register or Branch type, then ALUSrc is 0; otherwise, it's 1.
4)***ALUCtrl*** : Based on instruction type, ALUCtrl signal is determined. For R_type, the value is found using the table on page 148 of the RISC-V spec PDF. For convenience, all ALUOPs are defined as parameters.

top_proc_tb : To create the testbench, I define inputs as reg and outputs as wire. Then I instantiate the top_proc, INSTRUCTION_MEMORY, and DATA_MEMORY modules. In DATA_MEMORY, the addr input corresponds to the 9 LSBs of dAddress, write enable to MemWrite, din/dout to dWriteData/ReadData. In INSTRUCTION_MEMORY, the adr input corresponds to the 9 LSBs of PC and the dout output to the instr signal. For the clock signal, I initialize clk and use '#10 forever clk = ~ clk' to create a 20 ns clock period. Then I make rst high to initialize the system and after 10 ns set it low to begin operation.

Below is the schematic generated by Questa:



Waveforms



Processor instructions executed from ROM :

No	PC(decimal)	Instruction	RISC-V	Registers (decimal)
1	0 (line:1-4)	h00700093	addi x1, x0, 7	x1=7
2	4 (line:5-8)	h01500113	addi x2, x0, 21	x2=21
3	8 (line:9-12)	h002081b3	add x3, x1, x2	x3=28
4	12 (line:13-16)	hff700213	addi x4, x0, -9	x4=-9
5	16 (line:17-20)	hfef10293	addi x5, x2, -17	x5=4
6	20 (line:21-24)	h00428333	add x6, x5, x4	x6=-5
7	24 (line:25-28)	h402183b3	sub x7, x3, x2	x7=7
8	28 (line:29-32)	h00539433	sll x8, x7, x5	x8=112 (7<<4)
9	32 (line:33-36)	h008224b3	slt x9, x4, x8	x9=1
10	36 (line:37-40)	h00244533	xor x10, x8, x2	x10=101
11	40 (line:41-44)	h008575b3	and x11, x10, x8	x11=96
12	44 (line:45-48)	h0095d633	srl x12, x11, x9	x12=48 (96>>1)
13	48 (line:49-52)	h003666b3	or x13, x12, x3	x13=60
14	52 (line:53-56)	h40925733	sra x14, x4, x9	x14=-5
15	56 (line:57-60)	h00b2a023	sw x11, 0(x5)	RAM[4]=x11
16	60 (line:61-64)	h0002a783	lw x15, 0(x5)	x15=96
17	64 (line:65-68)	hfd347813	andi x16, x8, -45	x16=80
18	68 (line:69-72)	h01686893	ori x17, x16, 22	x17=86
19	72 (line:73-76)	h0016d913	srli x18, x13, 1	x18=30
20	76 (line:77-80)	h00b78863	beq x15, x11, 16	PC ←PC+16
	92 (line:93-96)	h00000000		
21	96 (line:97-100)	h00f92493	slti x9, x18, 15	x9=0
22	100 (line:101-104)	h03a44993	xori x19, x8, 58	x19=74
23	104 (line:105-108)	h00189a13	slli x20, x17, 1	x20=172
24	108 (line:109-112)	h4027d293	srai x5, x15, 2	x5=24
	112 (line:113-116)	h00000000		