Introduction to the implementation:

Detailed parts:

FOUR_MUX mux1(.in0(PC_cs + 4), .in1(IF_ID_currentPC + Imm_shift), .in2(ReadData1 + Imm), .in3(IF_ID_currentPC + Imm_shift),.sel({Jump,branch_out}), .out(PC_n));
Use this mux to select the next instruction that should be executed.

TWO_MUX mux30(.in0(0), .in1(con_in), .sel(Control_select), .out(con_out));
Use this mux to select nop or control signal.

Register_File RF1 (.RegWrite (MEM_WB_RegWrite), .WriteData (WriteData), .Rs1 (IF_ID_Instruct[19:15]), .Rs2 (IF_ID_Instruct[24:20]), .Rd (MEM_WB_rd), .R_ReadData1 (ReadData1), .R_ReadData2 (ReadData2));
The register stores the data, receives the address and give out the value.

TWO_MUX
mux2(.in0(ReadData1), .in1(EX_MEM_ALUResult), .sel(Forward1), .out(set1));
TWO_MUX
mux3(.in0(ReadData2), .in1(EX_MEM_ALUResult), .sel(Forward2), .out(set2));
These two muxes are used as the forwarding path selectors.

Comp Compare (.func ({IF_ID_Instruct[2], IF_ID_Instruct[14:12]}), .in1 (set1), .in2 (set2), .zero (zero));
This component is used to compare the in1 and in2, and check whether they are the same.

Immediate_Generator ImmGen (.In (IF_ID_Instruct), .Out (Imm));
Immediate Generator is used to change the immediate part in instruction to 32bits.

TWO_MUX
mux21(.in0(MUX_ALU), .in1(ID_EX_Imm), .sel(ID_EX_ALUSrc), .out(ALUInput2)
);
This mux is used to decide which data is sent to ALU to perform calculation.

ALU ALU (.data1 (ALUInput1), .data2 (ALUInput2), .sel (ALUControl), .out (ALUResult));
The ALU takes in the data from register and ImmGen, then perform certain calculation according to the ALU_control signals.

ALU_Control ALU_Control (.ALU_op (ID_EX_ALUOp), .instruct (ID_EX_Inst), .ALU_sel (ALUControl));
The ALU_Control takes in the Instruction and ALUop and gives out the control signal of ALU.

IF_ID_Reg IF_ID (.clock (clk), .IF_Flush (IF_Flush), .IFID_write (IFID_write), .currentPC (PC_cs), .nextPC (PC_cs + 4), .Instruct (Instruct), .currentPC_out (IF_ID_currentPC), .nextPC_out (IF_ID_nPC), .Instruct_out (IF_ID_Instruct));
The IF/ID register is used to store the data of the IF period and ready to be sent to ID period according to the clock tick.

Hazard HAZ (.rs1 (IF_ID_Instruct[19:15]), .rs2 (IF_ID_Instruct[24:20]), .ID_EX_rd (ID_EX_rd), .EX_MEM_rd (EX_MEM_rd), .ID_EX_MemRead (ID_EX_MemRead), .EX_MEM_MemRead (EX_MEM_MemRead), .branch (con_in[5]), .ID_EX_RegWrite (ID_EX_RegWrite), .PC_write (PC_write), .IFID_write (IFID_write), .control_select (Control_select));
This unit is used to detect all the hazards and perform corresponding implementations.

Forwarding FU (.ID_EX_rs1 (ID_EX_rs1), .ID_EX_rs2 (ID_EX_rs2), .IF_ID_rs1 (IF_ID_Instruct[19:15]), .IF_ID_rs2 (IF_ID_Instruct[24:20]), .EX_MEM_rs2 (EX_MEM_rs2), .MEM_WB_Rd (MEM_WB_rd), .EX_MEM_Rd (EX_MEM_rd), .ID_EX_Rd (ID_EX_rd), .MEM_WB_RegWrite (MEM_WB_RegWrite), .EX_MEM_RegWrite (EX_MEM_RegWrite), .EX_MEM_MemWrite (EX_MEM_MemWrite), .MEM_WB_MemRead (MEM_WB_MemRead), .EX_MEM_MemRead (EX_MEM_MemRead), .ID_EX_MemRead (ID_EX_MemRead), .ID_EX_MemWrite (ID_EX_MemWrite), .ID_EX_RegWrite (ID_EX_RegWrite), .Branch (Branch), .ForwardA (ForwardA), .ForwardB (ForwardB), .Forward1 (Forward1), .Forward2 (Forward2), .MemSrc (Memsrc));
This is the forwarding path to implement hazards.

Instruction_Memory InstructionMem (.instruct_out (Instruct), .PC(PC_cs));
Receive the PC from the PC register and search for the instruction stored in the instruction memory in advance.

Control Control (.opcode (IF_ID_Instruct[6:0]), .control_signal (con_in));
The control unit takes in the instruction and assign the control signals to the specific elements.

ID_EX_Reg ID_EX (.clock (clk), .RegWrite (RegWrite), .MemtoReg (MemtoReg), .MemRead (MemRead), .MemWrite (MemWrite), .Jump (Jump), .ALUSrc (ALUSrc), .ALUOp (ALUOp), .currentPC (IF_ID_currentPC), .nextPC (IF_ID_nPC), .Reg_rs1 (ReadData1), .Reg_rs2 (ReadData2), .Reg_rs1_addr (IF_ID_Instruct[19:15]), .Reg_rs2_addr (IF_ID_Instruct[24:20]), .Imm_Gen (Imm), .Reg_rd (IF_ID_Instruct[11:7]), .ALU_Instruct

({IF_ID_Instruct[30],IF_ID_Instruct[14:12]}), .RegWrite_out (ID_EX_RegWrite), .MemtoReg_out (ID_EX_MemtoReg), .MemRead_out (ID_EX_MemRead), .MemWrite_out (ID_EX_MemWrite), .Jump_out (ID_EX_Jump), .ALUSrc_out (ID_EX_ALUSrc), .ALUOp_out (ID_EX_ALUOp), .currentPC_out (ID_EX_currentPC), .nextPC_out (ID_EX_nPC), .Reg_rs1_out (ID_EX_ReadData1), .Reg_rs2_out (ID_EX_ReadData2), .Reg_rs1_addr_out (ID_EX_rs1), .Reg_rs2_addr_out (ID_EX_rs2), .Imm_Gen_out (ID_EX_Imm), .Reg_rd_out (ID_EX_rd), .ALU_Instruct_out (ID_EX_Inst));
The ID/EX register is used to store the data of the ID period and ready to be sent to EX period according to the clock tick.

EX_MEM_Reg EX_MEM (.clock (clk), .RegWrite (ID_EX_RegWrite), .MemtoReg (ID_EX_MemtoReg), .MemRead (ID_EX_MemRead), .MemWrite (ID_EX_MemWrite), .imm (ID_EX_Imm), .nextPC (ID_EX_nPC), .ALUResult (ALUResult), .Reg_rs2 (MUX_ALU), .Funct (ID_EX_Inst[2:0]), .Reg_rd (ID_EX_rd), .Reg_rs2_addr (ID_EX_rs2), .RegWrite_out (EX_MEM_RegWrite), .MemtoReg_out (EX_MEM_MemtoReg), .MemRead_out (EX_MEM_MemRead), .MemWrite_out (EX_MEM_MemWrite), .imm_out (EX_MEM_Imm), .nextPC_out (EX_MEM_nPC), .ALUResult_out (EX_MEM_ALUResult), .Reg_rs2_out (EX_MEM_ReadData2), .Funct_out (EX_MEM_Funct), .Reg_rd_out (EX_MEM_rd), .Reg_rs2_addr_out (EX_MEM_rs2));
The EX/MEM register is used to store the data of the EX period and ready to be sent to MEM period according to the clock tick.

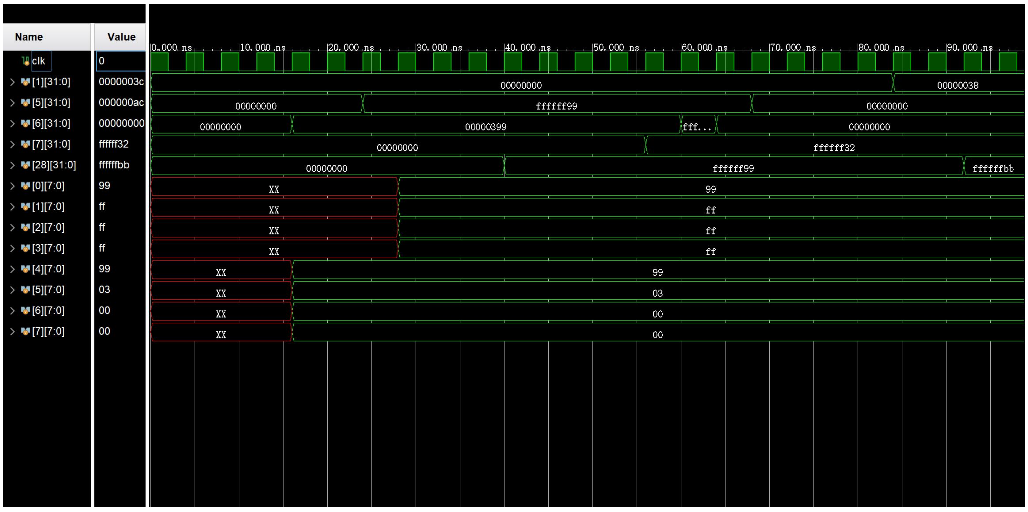
FOUR_MUX
mux6(.in0(ID_EX_ReadData1), .in1(WriteData), .in2(EX_MEM_ALUResult), .in3(EX_MEM_ALUResult), .sel(ForwardA), .out(ALUInput1));
FOUR_MUX
mux7(.in0(ID_EX_ReadData2), .in1(WriteData), .in2(EX_MEM_ALUResult), .in3(EX_MEM_ALUResult), .sel(ForwardB), .out(MUX_ALU));
TWO_MUX
mux8(.in0(EX_MEM_ReadData2), .in1(WriteData), .sel(Memsrc), .out(EX_Mem_WriteData));
These muxes are used to tackle the forwarding signals.

Data_Memory DataMem (.MemWrite (EX_MEM_MemWrite), .MemRead (EX_MEM_MemRead), .Funct (EX_MEM_Funct), .addr (EX_MEM_ALUResult), .WriteData (EX_Mem_WriteData), .R_data_out (Read_Mem));
The Data Memory serves as data store for using. It read one address and is controlled by the MemWrite&MemRead to decide whether read or write is performed.

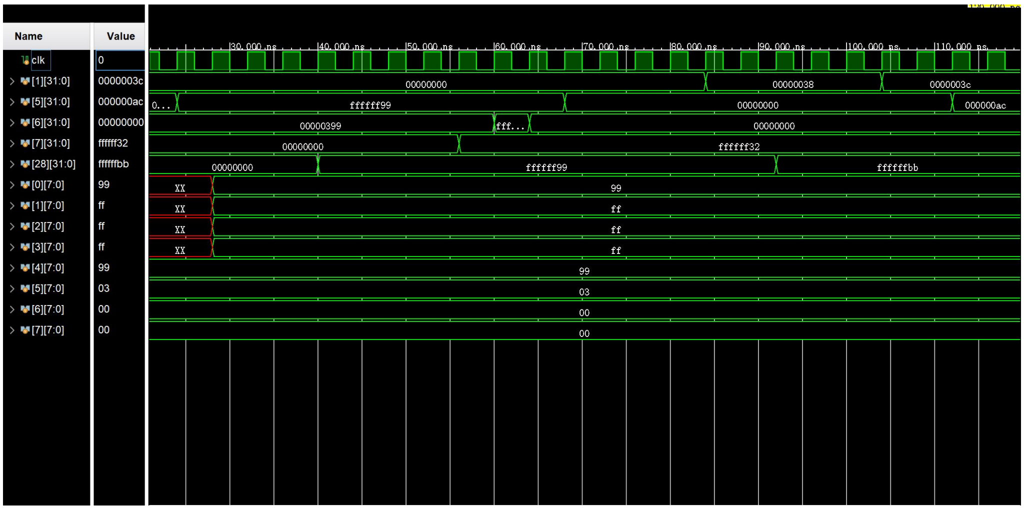
MEM_WB_Reg MEM_WB (.clock (clk), .MemRead (EX_MEM_MemRead), .RegWrite (EX_MEM_RegWrite), .MemtoReg (EX_MEM_MemtoReg), .imm (EX_MEM_Imm), .nextPC (EX_MEM_nPC), .ReadData (Read_Mem), .ALUResult (EX_MEM_ALUResult), .Reg_rd (EX_MEM_rd), .MemRead_out (MEM_WB_MemRead), .RegWrite_out (MEM_WB_RegWrite), .MemtoReg_out (MEM_WB_MemtoReg), .imm_out (MEM_WB_Imm), .nextPC_out (MEM_WB_nPC), .ReadData_out (MEM_WB_Read_Mem), .ALUResult_out (MEM_WB_ALUResult), .Reg_rd_out (MEM_WB_rd));
The MEM/WB register is used to store the data of the MEM period and ready to be sent to WB period according to the clock tick.

Simulation Results:

Data Hazard:
sw t1 4(x0)
lb t0 4(x0)

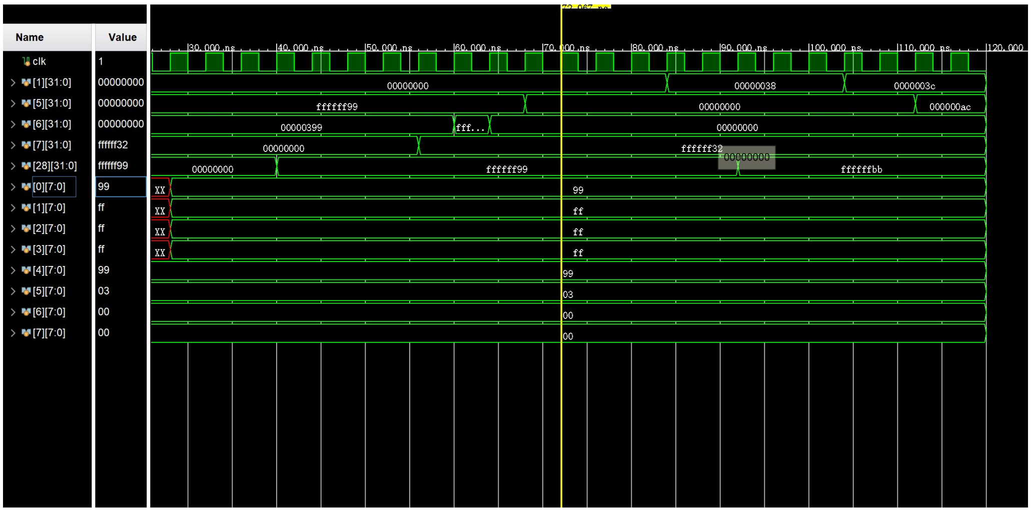


and t1 t2 t3
sub t0 t1 x0



Branch Harzard:

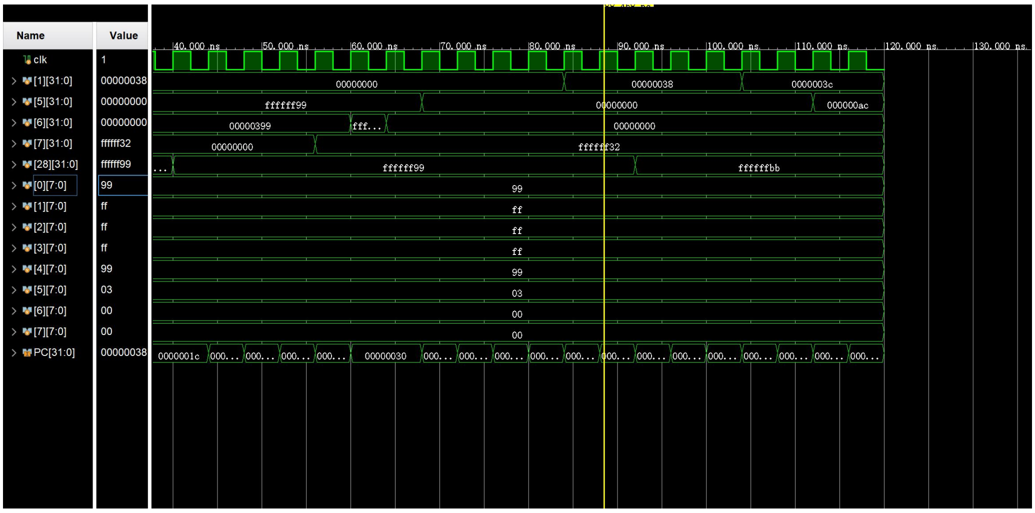
bge t0 t1 right_branch



Name	Value
clk	1
[1][31:0]	00000000
[5][31:0]	00000000
[6][31:0]	00000000
[7][31:0]	ffffff32
[28][31:0]	ffffff99
[0][7:0]	99
[1][7:0]	ff
[2][7:0]	ff
[3][7:0]	ff
[4][7:0]	99
[5][7:0]	03
[6][7:0]	00
[7][7:0]	00

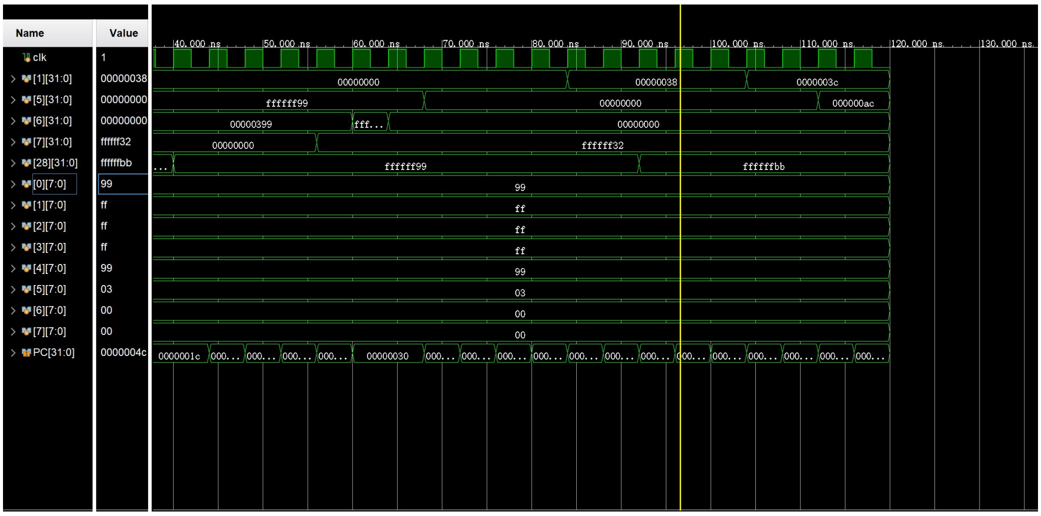
jal x1 jump_test

jalr x0 x1 0



Name	Value
clk	1
[1][31:0]	00000038
[5][31:0]	00000000
[6][31:0]	00000000
[7][31:0]	ffffff32
[28][31:0]	ffffff99
[0][7:0]	99
[1][7:0]	ff
[2][7:0]	ff
[3][7:0]	ff
[4][7:0]	99
[5][7:0]	03
[6][7:0]	00
[7][7:0]	00
PC[31:0]	00000038

jal x1 Exit



Name	Value
clk	1
[1][31:0]	00000038
[5][31:0]	00000000
[6][31:0]	00000000
[7][31:0]	ffffff32
[28][31:0]	ffffffbb
[0][7:0]	99
[1][7:0]	ff
[2][7:0]	ff
[3][7:0]	ff
[4][7:0]	99
[5][7:0]	03
[6][7:0]	00
[7][7:0]	00
PC[31:0]	0000004c

Schematic: