Assignment -3

Computer Architecture Lab

Group no. 5

Name and ID Student 1: TATWESH MISHRA 2019B5AA1300H

Name and ID Student 2: THATHAPUDI SANJEEV PAUL JOEL - 2020AAPS0120H

Name and ID Student 3: TARUN RAJKUMAR - 2020A8PS1447H

Implement a pipeline-based MIPS processor in Verilog that can execute at least 12 instructions, including R-type, I-type, and J-type (conditional and unconditional). The instructions should be chosen so that the three hazards, namely data, structural, and control hazards, should arise, and these issues should be resolved using techniques such as stalling, flushing, and forwarding based on the optimal solution.

Part-A

Edit this document containing the following once you are done with your Verilog implementation:

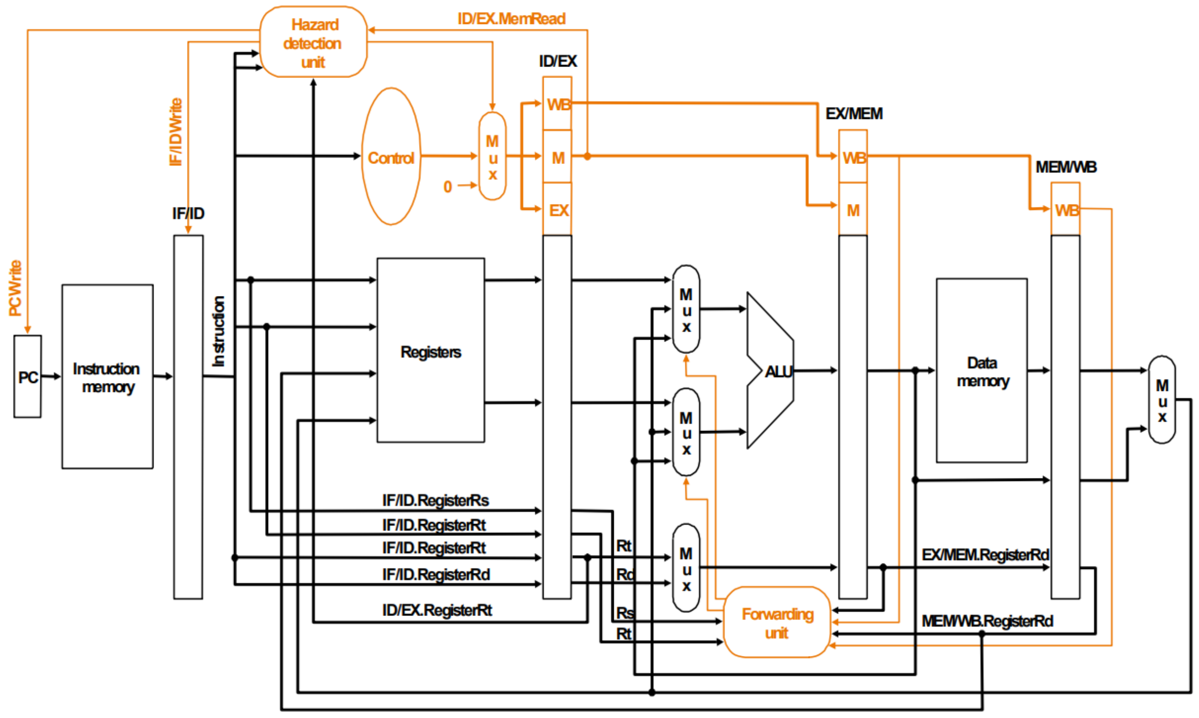
1) The program should indicate the hazard points and the result stored in each register.

2) Mention the techniques used to overcome the hazards encountered in the program chosen.

   Techniques used to tackle:

1. Structural Hazards: Introduced additional hardware like adders, sign extenders, etc.
2. Data Hazards: We used the forwarding unit for dealing with R-type and in addition used a stalling unit for dealing with hazards associated with load instruction.
3. Control Hazards: Introduced a flushing signal that asserts to true whenever a branch or jump is taken.

3) The block diagram of the processor showing the required data path, control path, and hazard detection unit.



4) The Truth Table for the control unit.

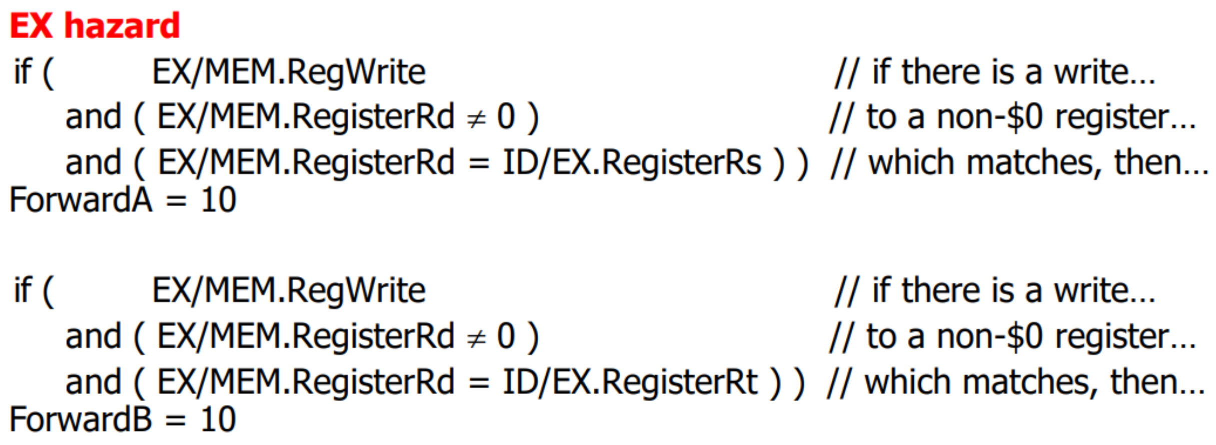
CONTROL UNIT:

Instruction	Opcode	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Jump	ALUOp	Branch
lw	100011	0	1	1	1	1	0	0	000	00
sw	101011	1	1	1	0	0	1	0	000	00
R Type	000000	1	0	0	1	0	0	0	110	00
j	000010	1	1	X	0	0	0	1	XXX	00
beq	000100	X	0	X	0	0	0	0	001	01
bne	000101	X	0	X	0	0	0	0	001	10
addi	001000	0	1	0	1	0	0	0	000	00
andi	001100	0	1	0	1	0	0	0	010	00
ori	001101	0	1	0	1	0	0	0	011	00
slti	001010	0	1	0	1	0	0	0	100	00
xori	001110	0	1	0	1	0	0	0	101	00

ALU CONTROL:

Function Field	ALU_Control	ALU_OP	ALU_Operation
XXXXXX	0010	000	Add
XXXXXX	0010	000	Add
100000	0010	110	Add
100010	0110	110	Sub
100100	0000	110	And
100101	0001	110	Or
101010	0111	110	Slt
100111	1100	110	Nor
100110	1101	110	Xor
000000	1000	110	Sll
000010	1010	110	Slr
000011	1011	110	Sra
XXXXXX	0110	001	Sub
XXXXXX	0110	001	Sub
XXXXXX	0010	000	Add
XXXXXX	0000	010	And
XXXXXX	0001	011	Or
XXXXXX	0111	100	Slt
XXXXXX	1101	101	Xor

FORWARDING UNIT:



EX hazard

```
if (        EX/MEM.RegWrite                            // if there is a write…
     and ( EX/MEM.RegisterRd ≠ 0 )                    // to a non-$0 register…
     and ( EX/MEM.RegisterRd = ID/EX.RegisterRs ) )  // which matches, then…
ForwardA = 10

if (        EX/MEM.RegWrite                            // if there is a write…
     and ( EX/MEM.RegisterRd ≠ 0 )                    // to a non-$0 register…
     and ( EX/MEM.RegisterRd = ID/EX.RegisterRt ) )  // which matches, then…
ForwardB = 10
```

MEM hazard

```
if (        MEM/WB.RegWrite                                   // if there is a write…
    and ( MEM/WB.RegisterRd ≠ 0 )                             // to a non-$0 register…
    and ( EX/MEM.RegisterRd ≠ ID/EX.RegisterRs )              // and not already a register match
                                                              // with earlier pipeline register…
    and ( MEM/WB.RegisterRd = ID/EX.RegisterRs ) )            // but match with later pipeline
                                                                 register, then…
ForwardA = 01

if (        MEM/WB.RegWrite                                   // if there is a write…
    and ( MEM/WB.RegisterRd ≠ 0 )                             // to a non-$0 register…
    and ( EX/MEM.RegisterRd ≠ ID/EX.RegisterRt )              // and not already a register match
                                                              // with earlier pipeline register…
    and ( MEM/WB.RegisterRd = ID/EX.RegisterRt ) )            // but match with later pipeline
                                                                 register, then…
ForwardB = 01
```

STALLING UNIT:

- Hazard detection unit implements the following check if to stall

```
if ( ID/EX.MemRead        // if the instruction in the EX stage is a load…
    and ( ( ID/EX.RegisterRt = IF/ID.RegisterRs )   // and the destination register
      or  ( ID/EX.RegisterRt = IF/ID.RegisterRt ) ) )  // matches either source
    register
                                          // of the instruction in the ID stage, then…
                                                 stall the pipeline
```

5) The program that you load in the instruction memory. (4 instructions minimum for each type)

Initial contents in register file and data memory:

RF[2] = 10, RF[3] = 15, RF[17] = 7, DM[8] = 31264, DM[9] = 31248

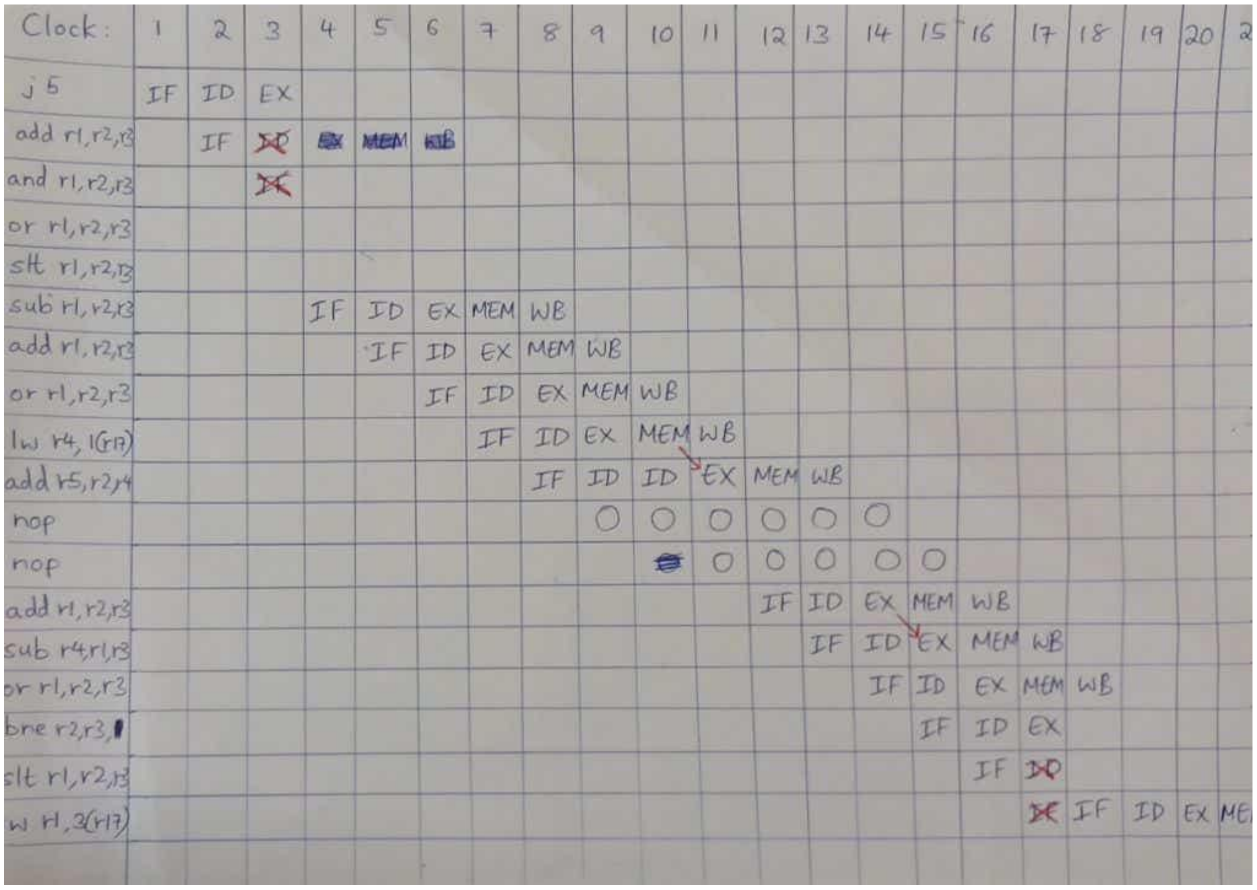
Program loaded:

j	5		-		08000005H
add	r1,	r2,	r3	-	00430820H
and	r1,	r2,	r3	-	00430824H
or	r1,	r2,	r3	-	00430825H
slt	r1,	r2,	r3	-	0043082AH
sub	r1,	r2,	r3	-	00430822H

add	r1,	r2,	r3	-	00430820H
or	r1,	r2,	r3	-	00430825H
lw	r4,	1(r17)		-	8E240001H
add	r5,	r2,	r4	-	00442820H
nop		-			00000000H
nop		-			00000000H
add	r1,	r2,	r3	-	00430820H
sub	r4,	r1,	r3	-	00232022H
or	r1,	r2,	r3	-	00430825H
bne	r2,	r3,	1	-	14430001H
slt	r1,	r2,	r3	-	0043082AH

sw r1, 3(r17) - AE210003H

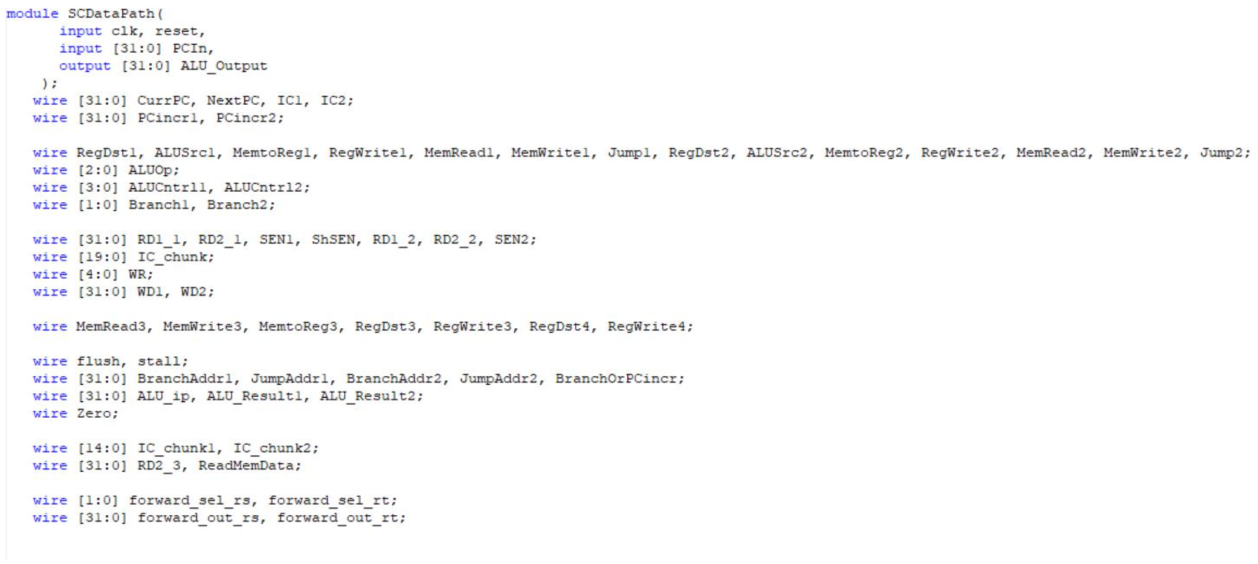
6) Show the pipeline diagram showing pipelining, the instructions, hazards, etc. An example is shown below.



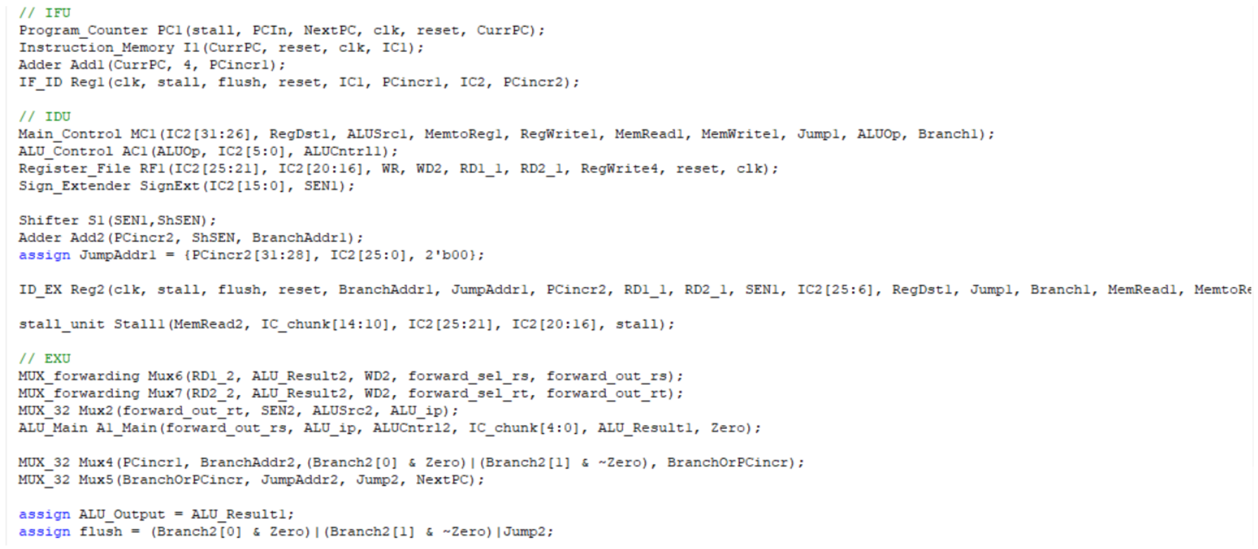
Clock:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	2
j 5	IF	ID	EX																		
add r1,r2,r3		IF	~~ID~~	EX	MEM	WB															
and r1,r2,r3			~~IF~~																		
or r1,r2,r3																					
slt r1,r2,r3																					
sub r1,r2,r3				IF	ID	EX	MEM	WB													
add r1,r2,r3					IF	ID	EX	MEM	WB												
or r1,r2,r3						IF	ID	EX	MEM	WB											
lw r4, 1(r17)							IF	ID	EX	MEM	WB										
add r5,r2,r4								IF	ID	ID	EX	MEM	WB								
nop									O	O	O	O	O	O							
nop										~~O~~	O	O	O	O	O						
add r1,r2,r3												IF	ID	EX	MEM	WB					
sub r4,r1,r3													IF	ID	EX	MEM	WB				
or r1,r2,r3														IF	ID	EX	MEM	WB			
bne r2,r3,															IF	ID	EX				
slt r1,r2,r3																IF	~~ID~~				
w r1,3(r17)																	~~IF~~	IF	ID	EX	ME

7) Paste the code from all the Verilog files/modules/mem files that are implemented.

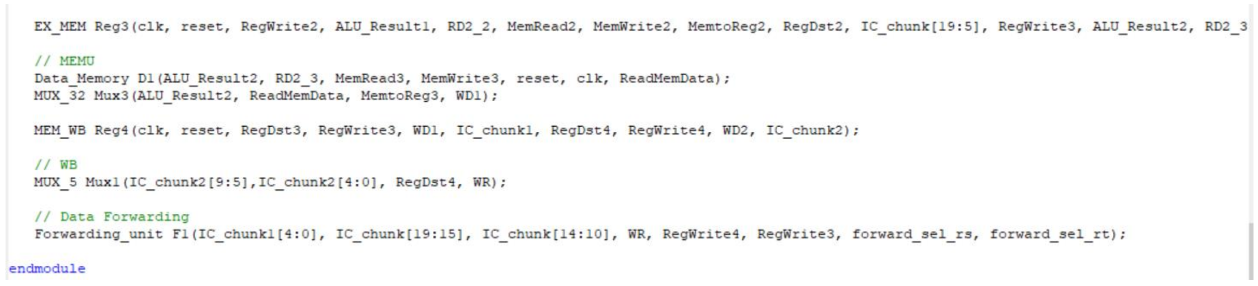
Main Module



```
module SCDataPath(
      input clk, reset,
      input [31:0] PCIn,
      output [31:0] ALU_Output
   );
   wire [31:0] CurrPC, NextPC, IC1, IC2;
   wire [31:0] PCincr1, PCincr2;

   wire RegDst1, ALUSrc1, MemtoReg1, RegWrite1, MemRead1, MemWrite1, Jump1, RegDst2, ALUSrc2, MemtoReg2, RegWrite2, MemRead2, MemWrite2, Jump2;
   wire [2:0] ALUOp;
   wire [3:0] ALUCntrl1, ALUCntrl2;
   wire [1:0] Branch1, Branch2;

   wire [31:0] RD1_1, RD2_1, SEN1, ShSEN, RD1_2, RD2_2, SEN2;
   wire [19:0] IC_chunk;
   wire [4:0] WR;
   wire [31:0] WD1, WD2;

   wire MemRead3, MemWrite3, MemtoReg3, RegDst3, RegWrite3, RegDst4, RegWrite4;

   wire flush, stall;
   wire [31:0] BranchAddr1, JumpAddr1, BranchAddr2, JumpAddr2, BranchOrPCincr;
   wire [31:0] ALU_ip, ALU_Result1, ALU_Result2;
   wire Zero;

   wire [14:0] IC_chunk1, IC_chunk2;
   wire [31:0] RD2_3, ReadMemData;

   wire [1:0] forward_sel_rs, forward_sel_rt;
   wire [31:0] forward_out_rs, forward_out_rt;
```

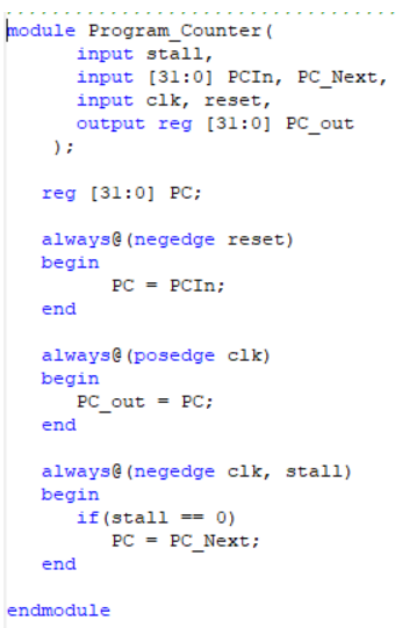


```
 // IFU
 Program_Counter PC1(stall, PCIn, NextPC, clk, reset, CurrPC);
 Instruction_Memory I1(CurrPC, reset, clk, IC1);
 Adder Add1(CurrPC, 4, PCincr1);
 IF_ID Reg1(clk, stall, flush, reset, IC1, PCincr1, IC2, PCincr2);

 // IDU
 Main_Control MC1(IC2[31:26], RegDst1, ALUSrc1, MemtoReg1, RegWrite1, MemRead1, MemWrite1, Jump1, ALUOp, Branch1);
 ALU_Control AC1(ALUOp, IC2[5:0], ALUCntrl1);
 Register_File RF1(IC2[25:21], IC2[20:16], WR, WD2, RD1_1, RD2_1, RegWrite4, reset, clk);
 Sign_Extender SignExt(IC2[15:0], SEN1);

 Shifter S1(SEN1,ShSEN);
 Adder Add2(PCincr2, ShSEN, BranchAddr1);
 assign JumpAddr1 = {PCincr2[31:28], IC2[25:0], 2'b00};

 ID_EX Reg2(clk, stall, flush, reset, BranchAddr1, JumpAddr1, PCincr2, RD1_1, RD2_1, SEN1, IC2[25:6], RegDst1, Jump1, Branch1, MemRead1, MemtoRe

 stall_unit Stall1(MemRead2, IC_chunk[14:10], IC2[25:21], IC2[20:16], stall);

 // EXU
 MUX_forwarding Mux6(RD1_2, ALU_Result2, WD2, forward_sel_rs, forward_out_rs);
 MUX_forwarding Mux7(RD2_2, ALU_Result2, WD2, forward_sel_rt, forward_out_rt);
 MUX_32 Mux2(forward_out_rt, SEN2, ALUSrc2, ALU_ip);
 ALU_Main A1_Main(forward_out_rs, ALU_ip, ALUCntrl2, IC_chunk[4:0], ALU_Result1, Zero);

 MUX_32 Mux4(PCincr1, BranchAddr2,(Branch2[0] & Zero)|(Branch2[1] & ~Zero), BranchOrPCincr);
 MUX_32 Mux5(BranchOrPCincr, JumpAddr2, Jump2, NextPC);

 assign ALU_Output = ALU_Result1;
 assign flush = (Branch2[0] & Zero)|(Branch2[1] & ~Zero)|Jump2;
```



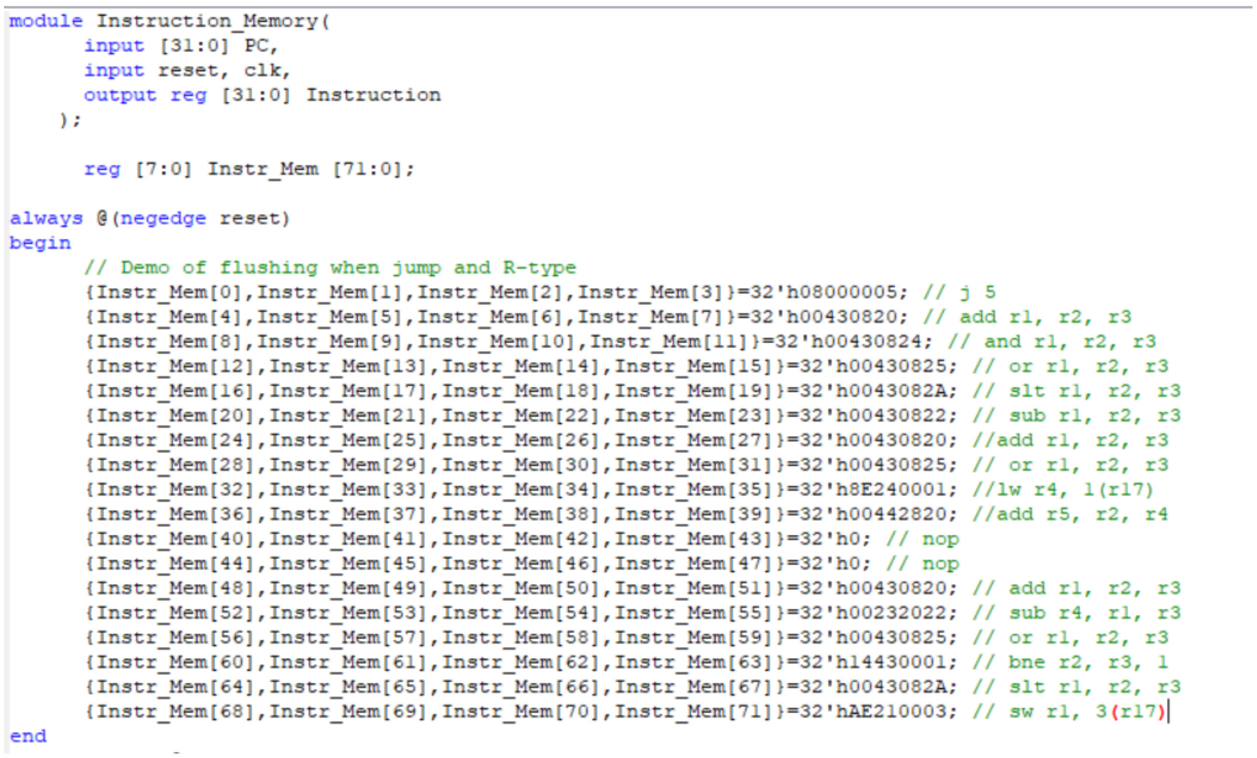
```
   EX_MEM Reg3(clk, reset, RegWrite2, ALU_Result1, RD2_2, MemRead2, MemWrite2, MemtoReg2, RegDst2, IC_chunk[19:5], RegWrite3, ALU_Result2, RD2_3

   // MEMU
   Data_Memory D1(ALU_Result2, RD2_3, MemRead3, MemWrite3, reset, clk, ReadMemData);
   MUX_32 Mux3(ALU_Result2, ReadMemData, MemtoReg3, WD1);

   MEM_WB Reg4(clk, reset, RegDst3, RegWrite3, WD1, IC_chunk1, RegDst4, RegWrite4, WD2, IC_chunk2);

   // WB
   MUX_5 Mux1(IC_chunk2[9:5],IC_chunk2[4:0], RegDst4, WR);

   // Data Forwarding
   Forwarding_unit F1(IC_chunk1[4:0], IC_chunk[19:15], IC_chunk[14:10], WR, RegWrite4, RegWrite3, forward_sel_rs, forward_sel_rt);

endmodule
```

Program Counter

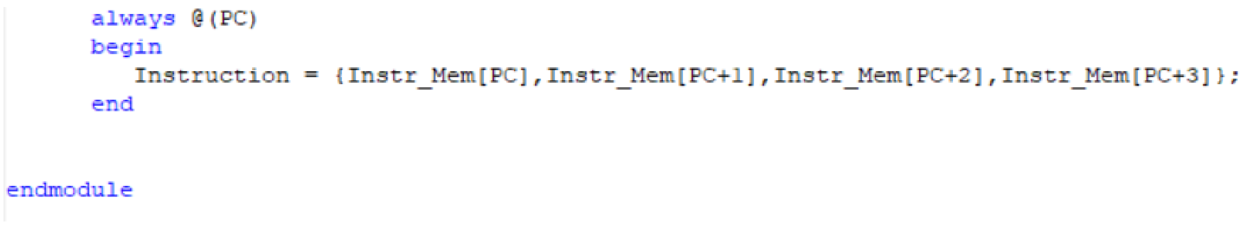


```
module Program_Counter(
      input stall,
      input [31:0] PCIn, PC_Next,
      input clk, reset,
      output reg [31:0] PC_out
    );

   reg [31:0] PC;

   always@(negedge reset)
   begin
         PC = PCIn;
   end

   always@(posedge clk)
   begin
      PC_out = PC;
   end

   always@(negedge clk, stall)
   begin
      if(stall == 0)
         PC = PC_Next;
   end

endmodule
```

Instruction Memory

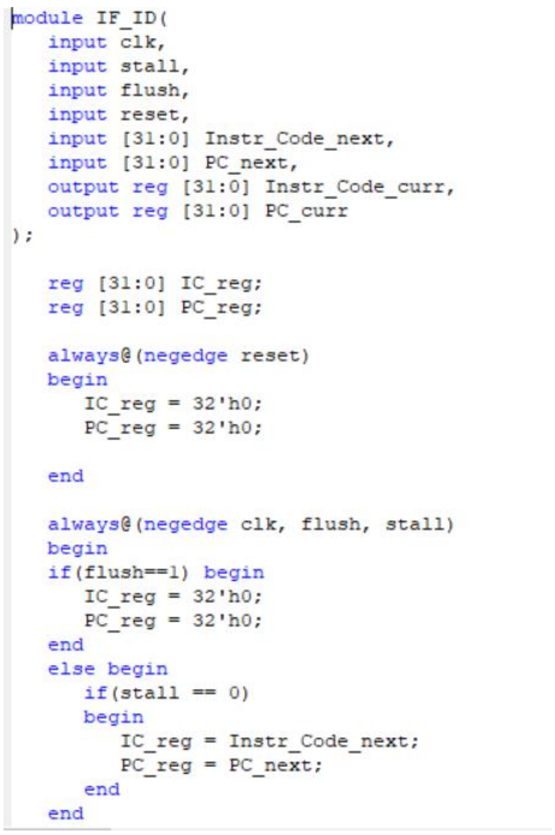


```
module Instruction_Memory(
      input [31:0] PC,
      input reset, clk,
      output reg [31:0] Instruction
    );

      reg [7:0] Instr_Mem [71:0];

always @(negedge reset)
begin
      // Demo of flushing when jump and R-type
      {Instr_Mem[0],Instr_Mem[1],Instr_Mem[2],Instr_Mem[3]}=32'h08000005; // j 5
      {Instr_Mem[4],Instr_Mem[5],Instr_Mem[6],Instr_Mem[7]}=32'h00430820; // add r1, r2, r3
      {Instr_Mem[8],Instr_Mem[9],Instr_Mem[10],Instr_Mem[11]}=32'h00430824; // and r1, r2, r3
      {Instr_Mem[12],Instr_Mem[13],Instr_Mem[14],Instr_Mem[15]}=32'h00430825; // or r1, r2, r3
      {Instr_Mem[16],Instr_Mem[17],Instr_Mem[18],Instr_Mem[19]}=32'h0043082A; // slt r1, r2, r3
      {Instr_Mem[20],Instr_Mem[21],Instr_Mem[22],Instr_Mem[23]}=32'h00430822; // sub r1, r2, r3
      {Instr_Mem[24],Instr_Mem[25],Instr_Mem[26],Instr_Mem[27]}=32'h00430820; //add r1, r2, r3
      {Instr_Mem[28],Instr_Mem[29],Instr_Mem[30],Instr_Mem[31]}=32'h00430825; // or r1, r2, r3
      {Instr_Mem[32],Instr_Mem[33],Instr_Mem[34],Instr_Mem[35]}=32'h8E240001; //lw r4, 1(r17)
      {Instr_Mem[36],Instr_Mem[37],Instr_Mem[38],Instr_Mem[39]}=32'h00442820; //add r5, r2, r4
      {Instr_Mem[40],Instr_Mem[41],Instr_Mem[42],Instr_Mem[43]}=32'h0; // nop
      {Instr_Mem[44],Instr_Mem[45],Instr_Mem[46],Instr_Mem[47]}=32'h0; // nop
      {Instr_Mem[48],Instr_Mem[49],Instr_Mem[50],Instr_Mem[51]}=32'h00430820; // add r1, r2, r3
      {Instr_Mem[52],Instr_Mem[53],Instr_Mem[54],Instr_Mem[55]}=32'h00232022; // sub r4, r1, r3
      {Instr_Mem[56],Instr_Mem[57],Instr_Mem[58],Instr_Mem[59]}=32'h00430825; // or r1, r2, r3
      {Instr_Mem[60],Instr_Mem[61],Instr_Mem[62],Instr_Mem[63]}=32'h14430001; // bne r2, r3, 1
      {Instr_Mem[64],Instr_Mem[65],Instr_Mem[66],Instr_Mem[67]}=32'h0043082A; // slt r1, r2, r3
      {Instr_Mem[68],Instr_Mem[69],Instr_Mem[70],Instr_Mem[71]}=32'hAE210003; // sw r1, 3(r17)
end
```

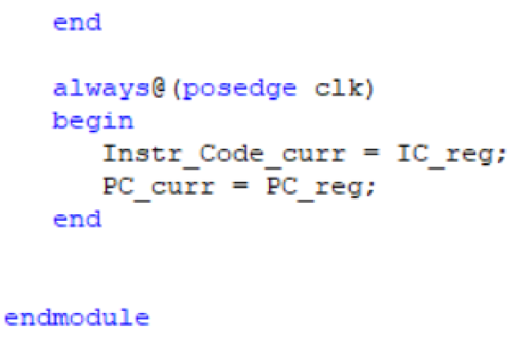


```
      always @(PC)
      begin
         Instruction = {Instr_Mem[PC],Instr_Mem[PC+1],Instr_Mem[PC+2],Instr_Mem[PC+3]};
      end


endmodule
```

IF/ID pipeline register

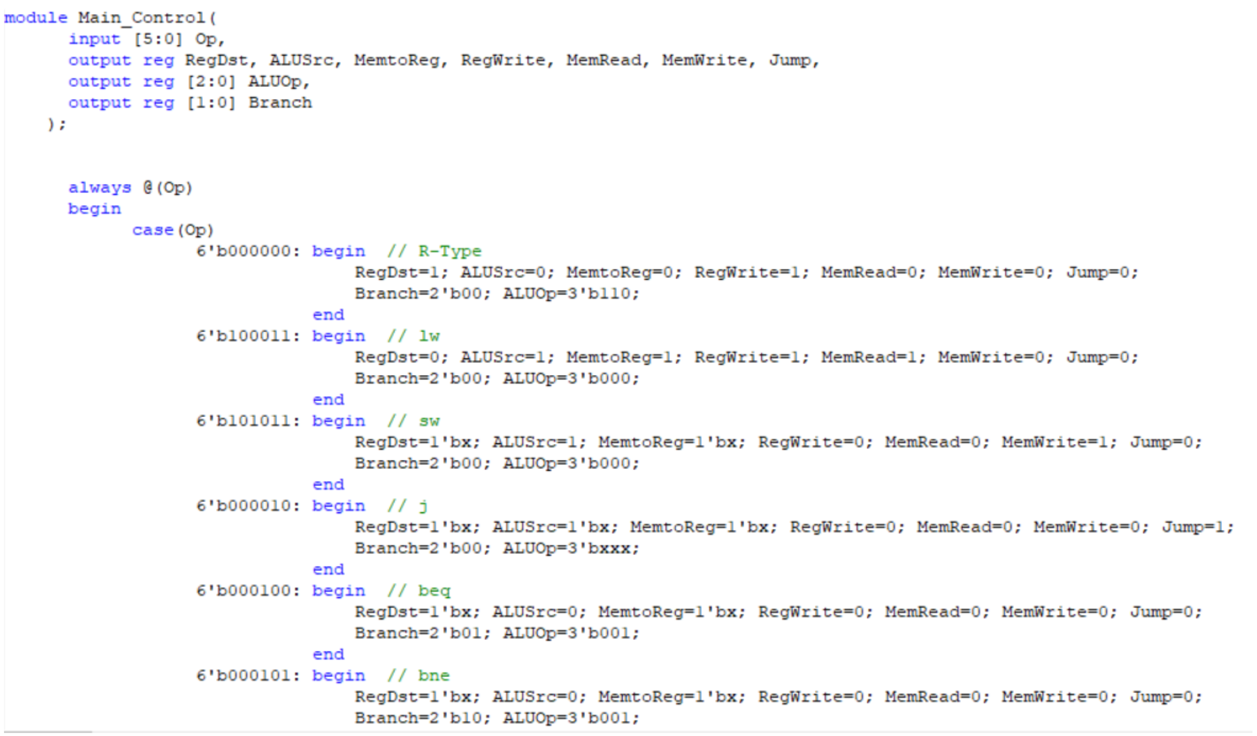


```
module IF_ID(
   input clk,
   input stall,
   input flush,
   input reset,
   input [31:0] Instr_Code_next,
   input [31:0] PC_next,
   output reg [31:0] Instr_Code_curr,
   output reg [31:0] PC_curr
);

   reg [31:0] IC_reg;
   reg [31:0] PC_reg;

   always@(negedge reset)
   begin
      IC_reg = 32'h0;
      PC_reg = 32'h0;

   end

   always@(negedge clk, flush, stall)
   begin
   if(flush==1) begin
      IC_reg = 32'h0;
      PC_reg = 32'h0;
   end
   else begin
      if(stall == 0)
      begin
         IC_reg = Instr_Code_next;
         PC_reg = PC_next;
      end
   end
```

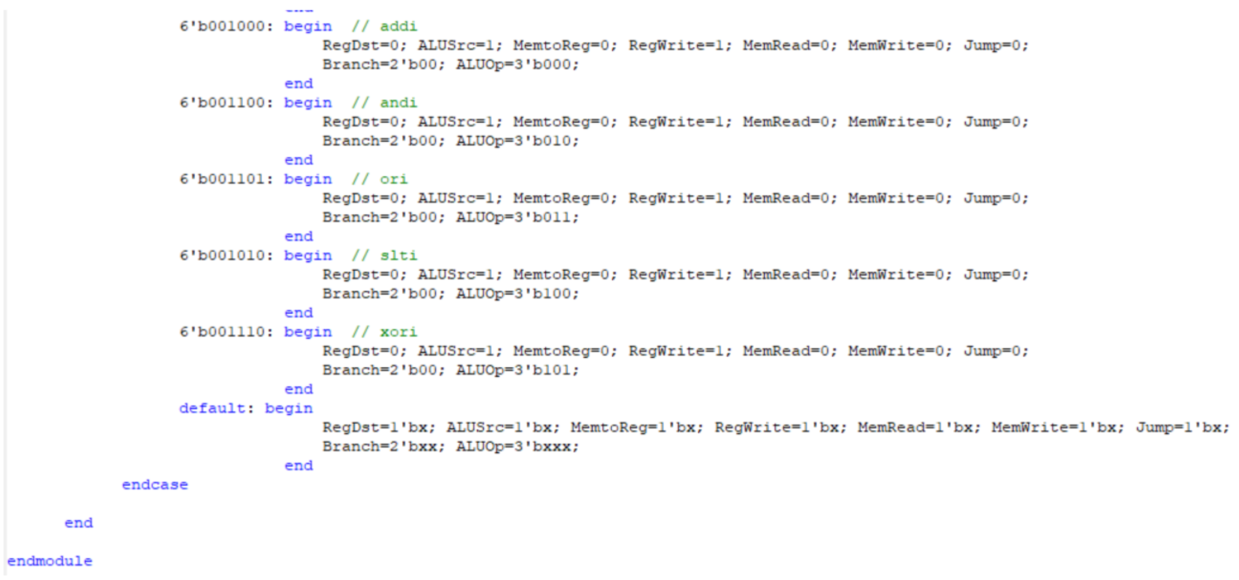


```
   end

   always@(posedge clk)
   begin
      Instr_Code_curr = IC_reg;
      PC_curr = PC_reg;
   end


endmodule
```

Main Control

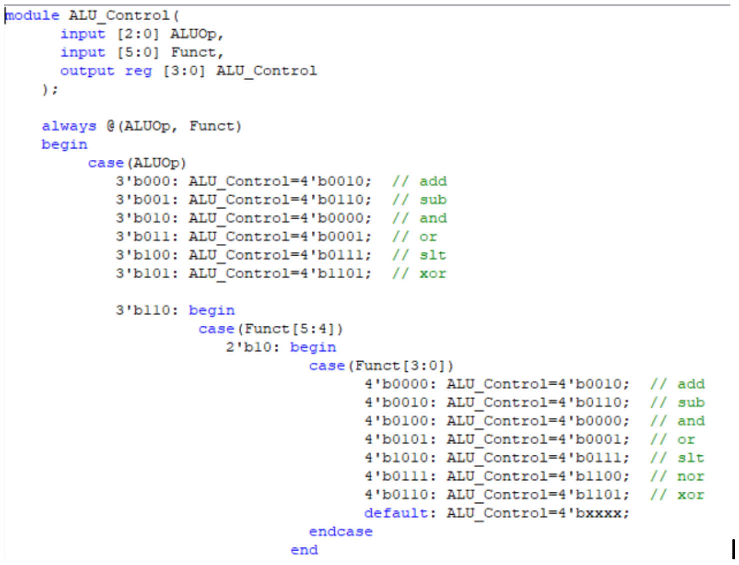


```
module Main_Control(
      input [5:0] Op,
      output reg RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Jump,
      output reg [2:0] ALUOp,
      output reg [1:0] Branch
    );


      always @(Op)
      begin
            case(Op)
                  6'b000000: begin  // R-Type
                              RegDst=1; ALUSrc=0; MemtoReg=0; RegWrite=1; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b00; ALUOp=3'b110;
                        end
                  6'b100011: begin  // lw
                              RegDst=0; ALUSrc=1; MemtoReg=1; RegWrite=1; MemRead=1; MemWrite=0; Jump=0;
                              Branch=2'b00; ALUOp=3'b000;
                        end
                  6'b101011: begin  // sw
                              RegDst=1'bx; ALUSrc=1; MemtoReg=1'bx; RegWrite=0; MemRead=0; MemWrite=1; Jump=0;
                              Branch=2'b00; ALUOp=3'b000;
                        end
                  6'b000010: begin  // j
                              RegDst=1'bx; ALUSrc=1'bx; MemtoReg=1'bx; RegWrite=0; MemRead=0; MemWrite=0; Jump=1;
                              Branch=2'b00; ALUOp=3'bxxx;
                        end
                  6'b000100: begin  // beq
                              RegDst=1'bx; ALUSrc=0; MemtoReg=1'bx; RegWrite=0; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b01; ALUOp=3'b001;
                        end
                  6'b000101: begin  // bne
                              RegDst=1'bx; ALUSrc=0; MemtoReg=1'bx; RegWrite=0; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b10; ALUOp=3'b001;
```

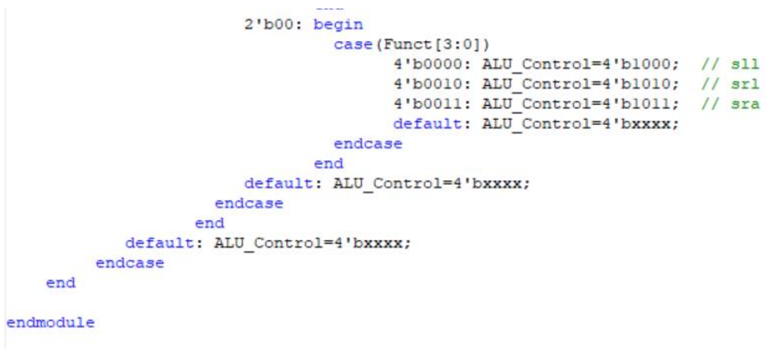


```
                  6'b001000: begin  // addi
                              RegDst=0; ALUSrc=1; MemtoReg=0; RegWrite=1; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b00; ALUOp=3'b000;
                        end
                  6'b001100: begin  // andi
                              RegDst=0; ALUSrc=1; MemtoReg=0; RegWrite=1; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b00; ALUOp=3'b010;
                        end
                  6'b001101: begin  // ori
                              RegDst=0; ALUSrc=1; MemtoReg=0; RegWrite=1; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b00; ALUOp=3'b011;
                        end
                  6'b001010: begin  // slti
                              RegDst=0; ALUSrc=1; MemtoReg=0; RegWrite=1; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b00; ALUOp=3'b100;
                        end
                  6'b001110: begin  // xori
                              RegDst=0; ALUSrc=1; MemtoReg=0; RegWrite=1; MemRead=0; MemWrite=0; Jump=0;
                              Branch=2'b00; ALUOp=3'b101;
                        end
                  default: begin
                              RegDst=1'bx; ALUSrc=1'bx; MemtoReg=1'bx; RegWrite=1'bx; MemRead=1'bx; MemWrite=1'bx; Jump=1'bx;
                              Branch=2'bxx; ALUOp=3'bxxx;
                        end
            endcase

      end

endmodule
```

ALU Control

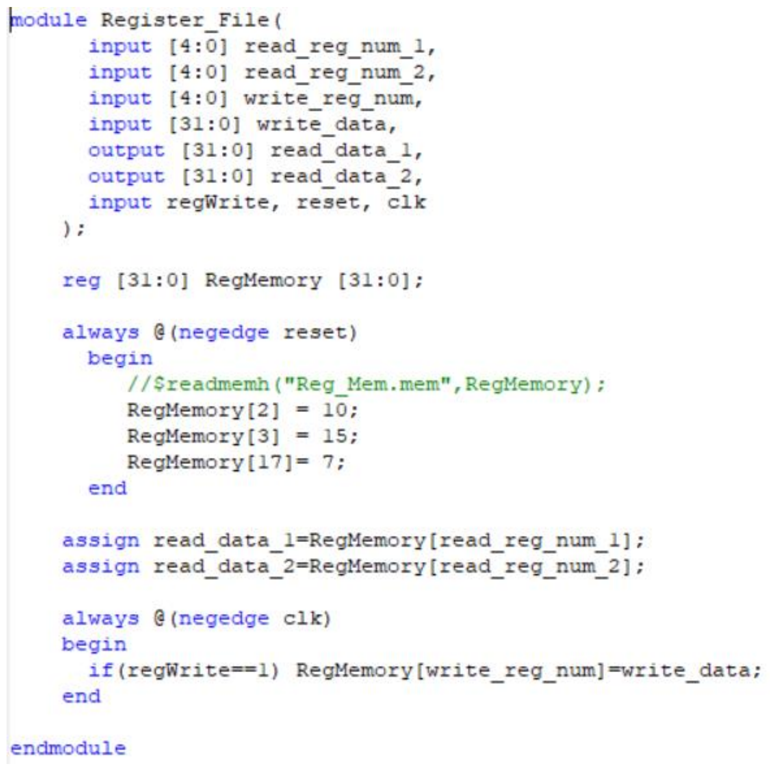


```
module ALU_Control(
      input [2:0] ALUOp,
      input [5:0] Funct,
      output reg [3:0] ALU_Control
    );

    always @(ALUOp, Funct)
    begin
         case(ALUOp)
            3'b000: ALU_Control=4'b0010;  // add
            3'b001: ALU_Control=4'b0110;  // sub
            3'b010: ALU_Control=4'b0000;  // and
            3'b011: ALU_Control=4'b0001;  // or
            3'b100: ALU_Control=4'b0111;  // slt
            3'b101: ALU_Control=4'b1101;  // xor

            3'b110: begin
                     case(Funct[5:4])
                        2'b10: begin
                                 case(Funct[3:0])
                                       4'b0000: ALU_Control=4'b0010;  // add
                                       4'b0010: ALU_Control=4'b0110;  // sub
                                       4'b0100: ALU_Control=4'b0000;  // and
                                       4'b0101: ALU_Control=4'b0001;  // or
                                       4'b1010: ALU_Control=4'b0111;  // slt
                                       4'b0111: ALU_Control=4'b1100;  // nor
                                       4'b0110: ALU_Control=4'b1101;  // xor
                                       default: ALU_Control=4'bxxxx;
                                 endcase
                               end
```



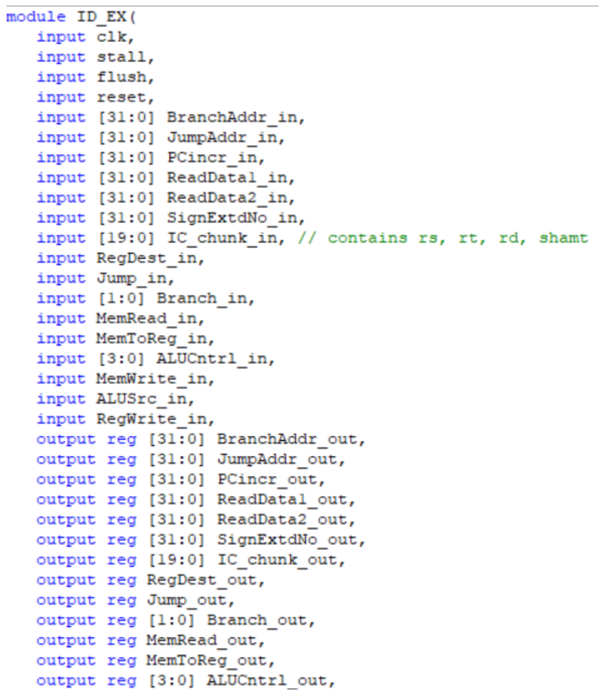
```
                        2'b00: begin
                                 case(Funct[3:0])
                                       4'b0000: ALU_Control=4'b1000;  // sll
                                       4'b0010: ALU_Control=4'b1010;  // srl
                                       4'b0011: ALU_Control=4'b1011;  // sra
                                       default: ALU_Control=4'bxxxx;
                                 endcase
                               end
                        default: ALU_Control=4'bxxxx;
                     endcase
                   end
            default: ALU_Control=4'bxxxx;
         endcase
    end

endmodule
```

Register File

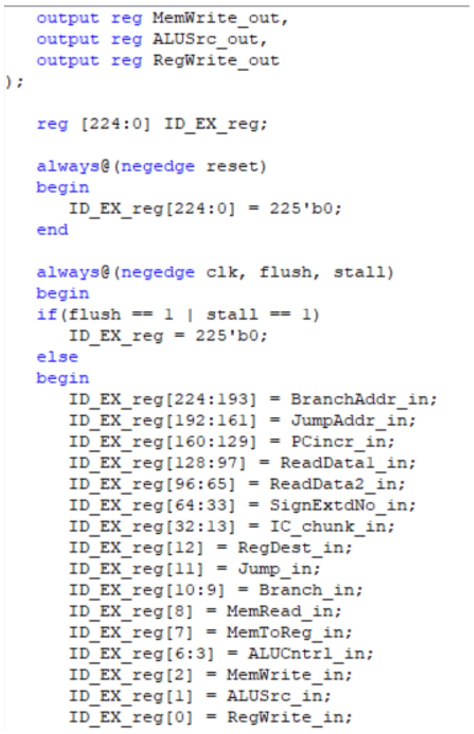


```
module Register_File(
      input [4:0] read_reg_num_1,
      input [4:0] read_reg_num_2,
      input [4:0] write_reg_num,
      input [31:0] write_data,
      output [31:0] read_data_1,
      output [31:0] read_data_2,
      input regWrite, reset, clk
    );

    reg [31:0] RegMemory [31:0];

    always @(negedge reset)
      begin
         //$readmemh("Reg_Mem.mem",RegMemory);
         RegMemory[2] = 10;
         RegMemory[3] = 15;
         RegMemory[17]= 7;
      end

    assign read_data_1=RegMemory[read_reg_num_1];
    assign read_data_2=RegMemory[read_reg_num_2];

    always @(negedge clk)
    begin
      if(regWrite==1) RegMemory[write_reg_num]=write_data;
    end

endmodule
```

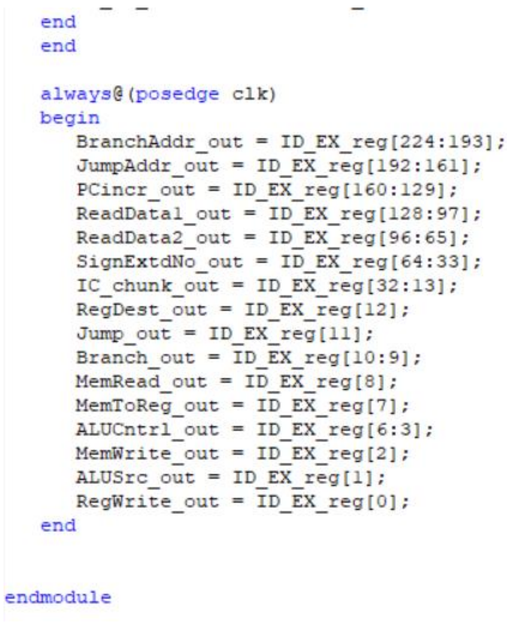
ID/EX Pipeline Register



```
module ID_EX(
   input clk,
   input stall,
   input flush,
   input reset,
   input [31:0] BranchAddr_in,
   input [31:0] JumpAddr_in,
   input [31:0] PCincr_in,
   input [31:0] ReadData1_in,
   input [31:0] ReadData2_in,
   input [31:0] SignExtdNo_in,
   input [19:0] IC_chunk_in, // contains rs, rt, rd, shamt
   input RegDest_in,
   input Jump_in,
   input [1:0] Branch_in,
   input MemRead_in,
   input MemToReg_in,
   input [3:0] ALUCntrl_in,
   input MemWrite_in,
   input ALUSrc_in,
   input RegWrite_in,
   output reg [31:0] BranchAddr_out,
   output reg [31:0] JumpAddr_out,
   output reg [31:0] PCincr_out,
   output reg [31:0] ReadData1_out,
   output reg [31:0] ReadData2_out,
   output reg [31:0] SignExtdNo_out,
   output reg [19:0] IC_chunk_out,
   output reg RegDest_out,
   output reg Jump_out,
   output reg [1:0] Branch_out,
   output reg MemRead_out,
   output reg MemToReg_out,
   output reg [3:0] ALUCntrl_out,
```

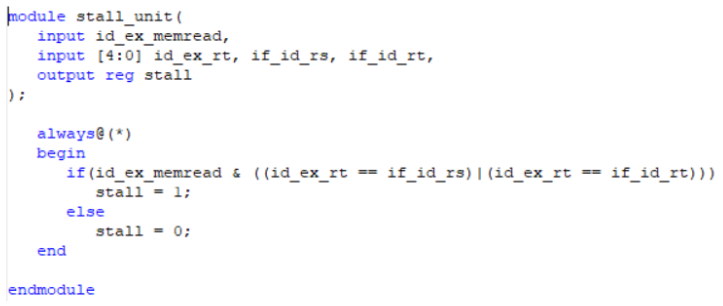


```
   output reg MemWrite_out,
   output reg ALUSrc_out,
   output reg RegWrite_out
);

   reg [224:0] ID_EX_reg;

   always@(negedge reset)
   begin
      ID_EX_reg[224:0] = 225'b0;
   end

   always@(negedge clk, flush, stall)
   begin
   if(flush == 1 | stall == 1)
      ID_EX_reg = 225'b0;
   else
   begin
      ID_EX_reg[224:193] = BranchAddr_in;
      ID_EX_reg[192:161] = JumpAddr_in;
      ID_EX_reg[160:129] = PCincr_in;
      ID_EX_reg[128:97] = ReadData1_in;
      ID_EX_reg[96:65] = ReadData2_in;
      ID_EX_reg[64:33] = SignExtdNo_in;
      ID_EX_reg[32:13] = IC_chunk_in;
      ID_EX_reg[12] = RegDest_in;
      ID_EX_reg[11] = Jump_in;
      ID_EX_reg[10:9] = Branch_in;
      ID_EX_reg[8] = MemRead_in;
      ID_EX_reg[7] = MemToReg_in;
      ID_EX_reg[6:3] = ALUCntrl_in;
      ID_EX_reg[2] = MemWrite_in;
      ID_EX_reg[1] = ALUSrc_in;
      ID_EX_reg[0] = RegWrite_in;
```



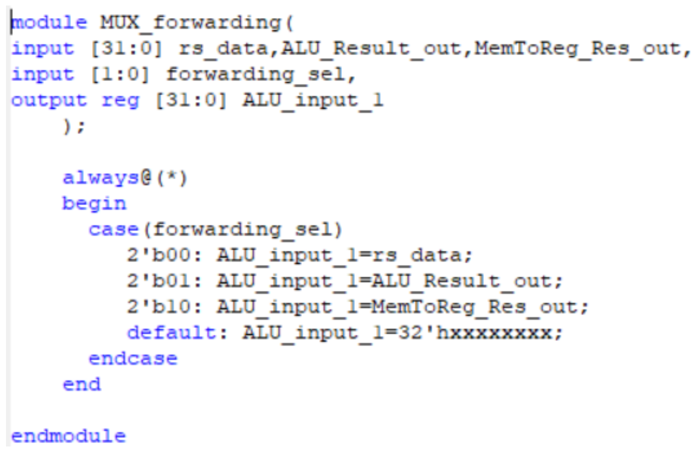
```
   end
   end

   always@(posedge clk)
   begin
      BranchAddr_out = ID_EX_reg[224:193];
      JumpAddr_out = ID_EX_reg[192:161];
      PCincr_out = ID_EX_reg[160:129];
      ReadData1_out = ID_EX_reg[128:97];
      ReadData2_out = ID_EX_reg[96:65];
      SignExtdNo_out = ID_EX_reg[64:33];
      IC_chunk_out = ID_EX_reg[32:13];
      RegDest_out = ID_EX_reg[12];
      Jump_out = ID_EX_reg[11];
      Branch_out = ID_EX_reg[10:9];
      MemRead_out = ID_EX_reg[8];
      MemToReg_out = ID_EX_reg[7];
      ALUCntrl_out = ID_EX_reg[6:3];
      MemWrite_out = ID_EX_reg[2];
      ALUSrc_out = ID_EX_reg[1];
      RegWrite_out = ID_EX_reg[0];
   end


endmodule
```

Stall Unit



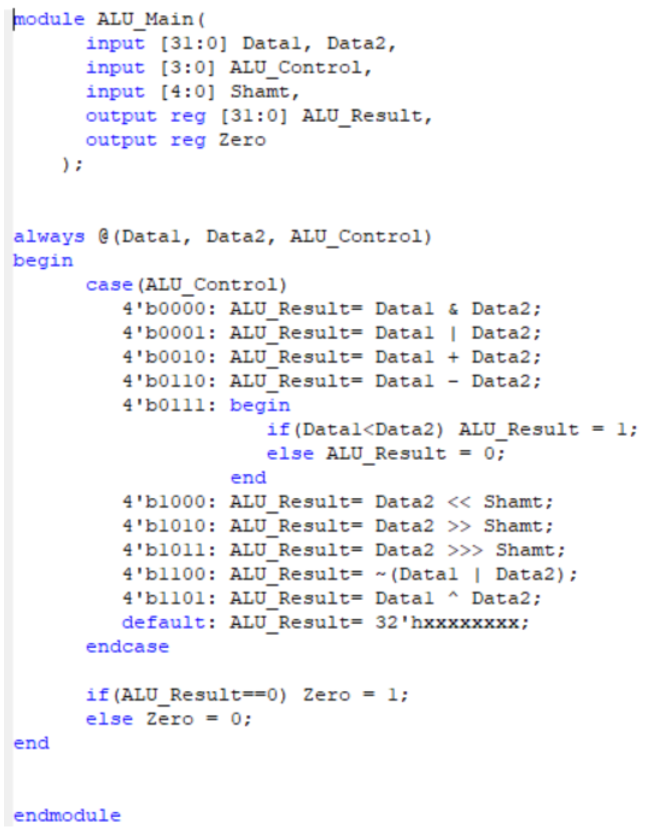
```
module stall_unit(
   input id_ex_memread,
   input [4:0] id_ex_rt, if_id_rs, if_id_rt,
   output reg stall
);

   always@(*)
   begin
      if(id_ex_memread & ((id_ex_rt == if_id_rs)|(id_ex_rt == if_id_rt)))
         stall = 1;
      else
         stall = 0;
   end

endmodule
```

MUX used in forwarding



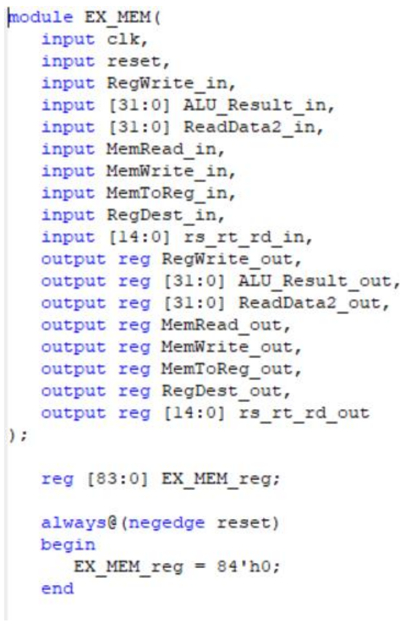
```
module MUX_forwarding(
input [31:0] rs_data,ALU_Result_out,MemToReg_Res_out,
input [1:0] forwarding_sel,
output reg [31:0] ALU_input_1
    );

    always@(*)
    begin
      case(forwarding_sel)
         2'b00: ALU_input_1=rs_data;
         2'b01: ALU_input_1=ALU_Result_out;
         2'b10: ALU_input_1=MemToReg_Res_out;
         default: ALU_input_1=32'hxxxxxxxx;
      endcase
    end

endmodule
```

ALU Main

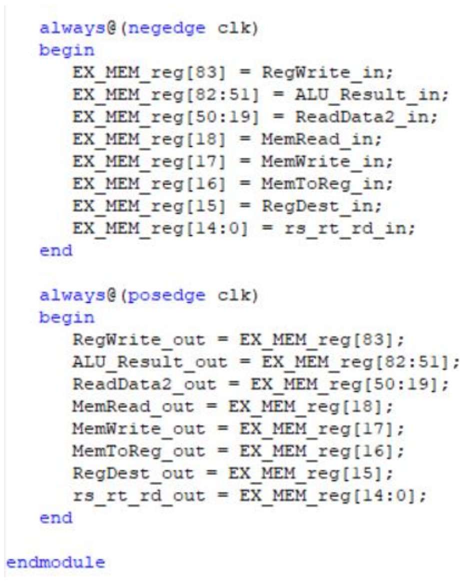


```
module ALU_Main(
      input [31:0] Data1, Data2,
      input [3:0] ALU_Control,
      input [4:0] Shamt,
      output reg [31:0] ALU_Result,
      output reg Zero
    );


always @(Data1, Data2, ALU_Control)
begin
      case(ALU_Control)
         4'b0000: ALU_Result= Data1 & Data2;
         4'b0001: ALU_Result= Data1 | Data2;
         4'b0010: ALU_Result= Data1 + Data2;
         4'b0110: ALU_Result= Data1 - Data2;
         4'b0111: begin
                     if(Data1<Data2) ALU_Result = 1;
                     else ALU_Result = 0;
                  end
         4'b1000: ALU_Result= Data2 << Shamt;
         4'b1010: ALU_Result= Data2 >> Shamt;
         4'b1011: ALU_Result= Data2 >>> Shamt;
         4'b1100: ALU_Result= ~(Data1 | Data2);
         4'b1101: ALU_Result= Data1 ^ Data2;
         default: ALU_Result= 32'hxxxxxxxx;
      endcase

      if(ALU_Result==0) Zero = 1;
      else Zero = 0;
end


endmodule
```

EX/MEM Pipeline Register

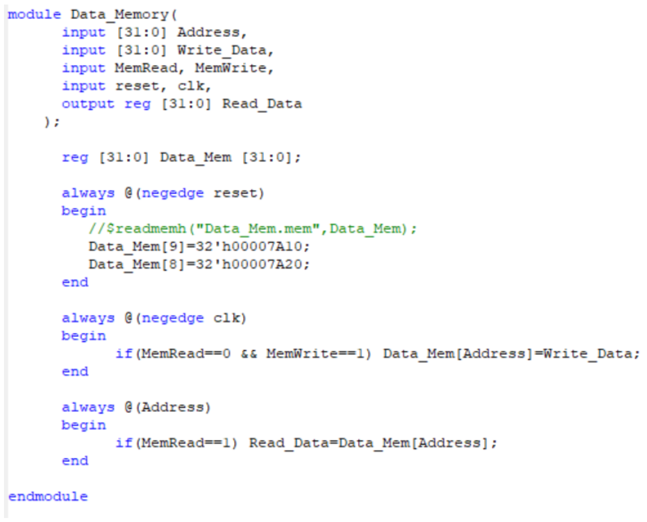


```
module EX_MEM(
   input clk,
   input reset,
   input RegWrite_in,
   input [31:0] ALU_Result_in,
   input [31:0] ReadData2_in,
   input MemRead_in,
   input MemWrite_in,
   input MemToReg_in,
   input RegDest_in,
   input [14:0] rs_rt_rd_in,
   output reg RegWrite_out,
   output reg [31:0] ALU_Result_out,
   output reg [31:0] ReadData2_out,
   output reg MemRead_out,
   output reg MemWrite_out,
   output reg MemToReg_out,
   output reg RegDest_out,
   output reg [14:0] rs_rt_rd_out
);

   reg [83:0] EX_MEM_reg;

   always@(negedge reset)
   begin
      EX_MEM_reg = 84'h0;
   end
```



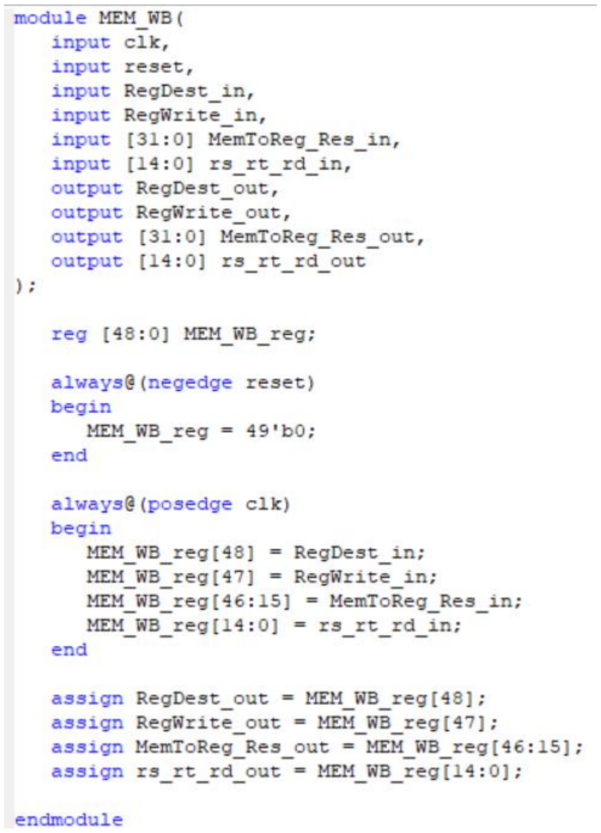
```
   always@(negedge clk)
   begin
      EX_MEM_reg[83] = RegWrite_in;
      EX_MEM_reg[82:51] = ALU_Result_in;
      EX_MEM_reg[50:19] = ReadData2_in;
      EX_MEM_reg[18] = MemRead_in;
      EX_MEM_reg[17] = MemWrite_in;
      EX_MEM_reg[16] = MemToReg_in;
      EX_MEM_reg[15] = RegDest_in;
      EX_MEM_reg[14:0] = rs_rt_rd_in;
   end

   always@(posedge clk)
   begin
      RegWrite_out = EX_MEM_reg[83];
      ALU_Result_out = EX_MEM_reg[82:51];
      ReadData2_out = EX_MEM_reg[50:19];
      MemRead_out = EX_MEM_reg[18];
      MemWrite_out = EX_MEM_reg[17];
      MemToReg_out = EX_MEM_reg[16];
      RegDest_out = EX_MEM_reg[15];
      rs_rt_rd_out = EX_MEM_reg[14:0];
   end

endmodule
```

Data Memory



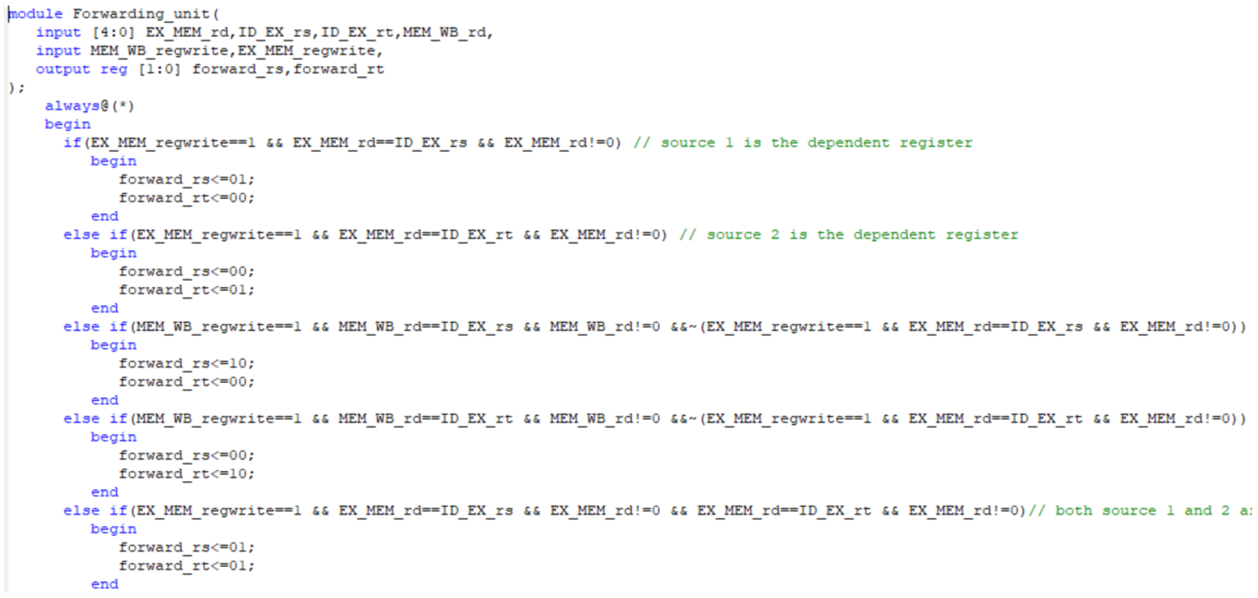
```
module Data_Memory(
      input [31:0] Address,
      input [31:0] Write_Data,
      input MemRead, MemWrite,
      input reset, clk,
      output reg [31:0] Read_Data
    );

      reg [31:0] Data_Mem [31:0];

      always @(negedge reset)
      begin
         //$readmemh("Data_Mem.mem",Data_Mem);
         Data_Mem[9]=32'h00007A10;
         Data_Mem[8]=32'h00007A20;
      end

      always @(negedge clk)
      begin
            if(MemRead==0 && MemWrite==1) Data_Mem[Address]=Write_Data;
      end

      always @(Address)
      begin
            if(MemRead==1) Read_Data=Data_Mem[Address];
      end

endmodule
```

MEM/WB Pipeline Register

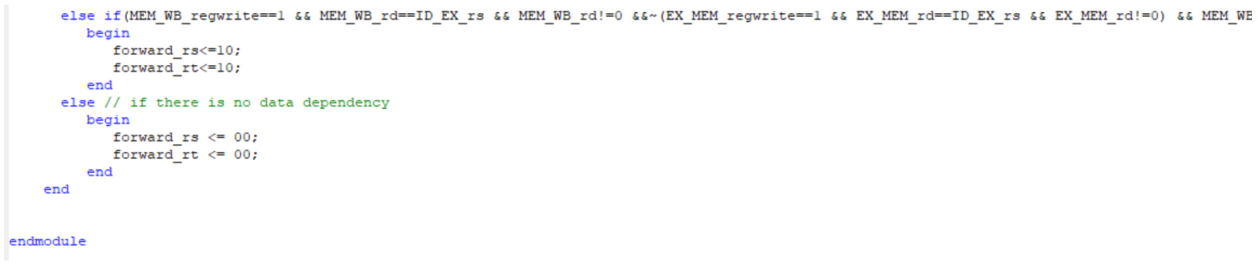


```
module MEM_WB(
   input clk,
   input reset,
   input RegDest_in,
   input RegWrite_in,
   input [31:0] MemToReg_Res_in,
   input [14:0] rs_rt_rd_in,
   output RegDest_out,
   output RegWrite_out,
   output [31:0] MemToReg_Res_out,
   output [14:0] rs_rt_rd_out
);

   reg [48:0] MEM_WB_reg;

   always@(negedge reset)
   begin
      MEM_WB_reg = 49'b0;
   end

   always@(posedge clk)
   begin
      MEM_WB_reg[48] = RegDest_in;
      MEM_WB_reg[47] = RegWrite_in;
      MEM_WB_reg[46:15] = MemToReg_Res_in;
      MEM_WB_reg[14:0] = rs_rt_rd_in;
   end

   assign RegDest_out = MEM_WB_reg[48];
   assign RegWrite_out = MEM_WB_reg[47];
   assign MemToReg_Res_out = MEM_WB_reg[46:15];
   assign rs_rt_rd_out = MEM_WB_reg[14:0];

endmodule
```

Forwarding Unit

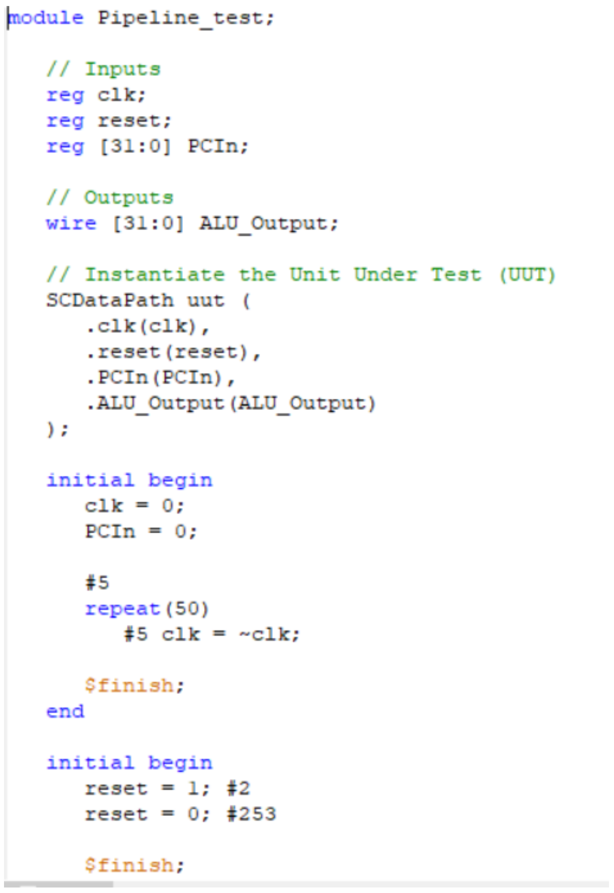


```
module Forwarding_unit(
   input [4:0] EX_MEM_rd,ID_EX_rs,ID_EX_rt,MEM_WB_rd,
   input MEM_WB_regwrite,EX_MEM_regwrite,
   output reg [1:0] forward_rs,forward_rt
);
    always@(*)
    begin
      if(EX_MEM_regwrite==1 && EX_MEM_rd==ID_EX_rs && EX_MEM_rd!=0) // source 1 is the dependent register
         begin
            forward_rs<=01;
            forward_rt<=00;
         end
      else if(EX_MEM_regwrite==1 && EX_MEM_rd==ID_EX_rt && EX_MEM_rd!=0) // source 2 is the dependent register
         begin
            forward_rs<=00;
            forward_rt<=01;
         end
      else if(MEM_WB_regwrite==1 && MEM_WB_rd==ID_EX_rs && MEM_WB_rd!=0 &&~(EX_MEM_regwrite==1 && EX_MEM_rd==ID_EX_rs && EX_MEM_rd!=0))
         begin
            forward_rs<=10;
            forward_rt<=00;
         end
      else if(MEM_WB_regwrite==1 && MEM_WB_rd==ID_EX_rt && MEM_WB_rd!=0 &&~(EX_MEM_regwrite==1 && EX_MEM_rd==ID_EX_rt && EX_MEM_rd!=0))
         begin
            forward_rs<=00;
            forward_rt<=10;
         end
      else if(EX_MEM_regwrite==1 && EX_MEM_rd==ID_EX_rs && EX_MEM_rd!=0 && EX_MEM_rd==ID_EX_rt && EX_MEM_rd!=0)// both source 1 and 2 a
         begin
            forward_rs<=01;
            forward_rt<=01;
         end
```



```
      else if(MEM_WB_regwrite==1 && MEM_WB_rd==ID_EX_rs && MEM_WB_rd!=0 &&~(EX_MEM_regwrite==1 && EX_MEM_rd==ID_EX_rs && EX_MEM_rd!=0) && MEM_WE
         begin
            forward_rs<=10;
            forward_rt<=10;
         end
      else // if there is no data dependency
         begin
            forward_rs <= 00;
            forward_rt <= 00;
         end
    end

endmodule
```

Test Bench

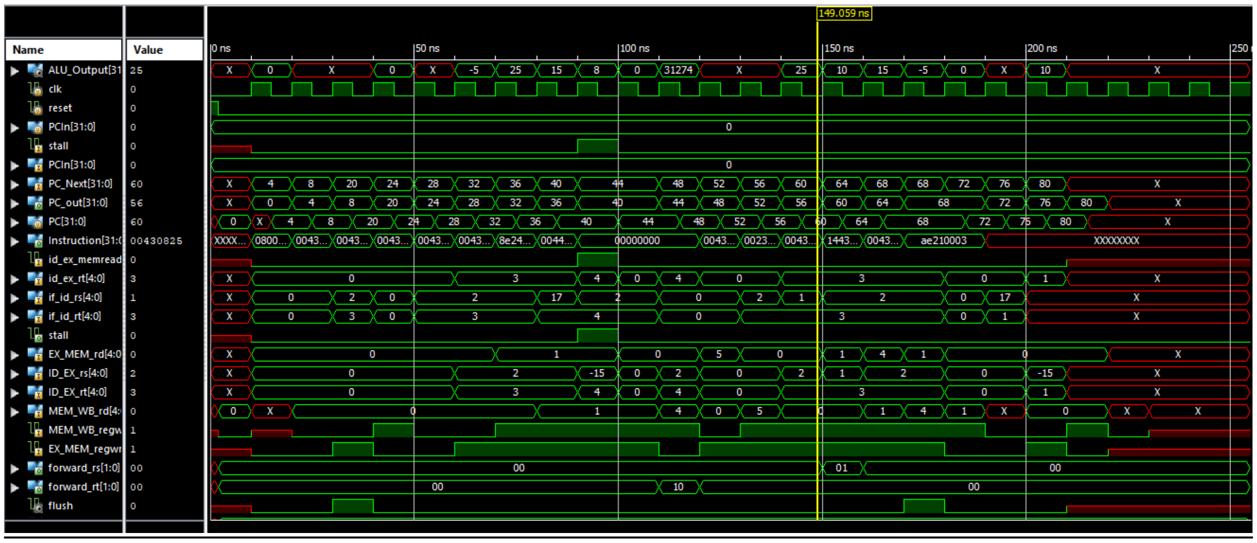


```
module Pipeline_test;

   // Inputs
   reg clk;
   reg reset;
   reg [31:0] PCIn;

   // Outputs
   wire [31:0] ALU_Output;

   // Instantiate the Unit Under Test (UUT)
   SCDataPath uut (
      .clk(clk),
      .reset(reset),
      .PCIn(PCIn),
      .ALU_Output(ALU_Output)
   );

   initial begin
      clk = 0;
      PCIn = 0;

      #5
      repeat(50)
         #5 clk = ~clk;

      $finish;
   end

   initial begin
      reset = 1; #2
      reset = 0; #253

      $finish;
```

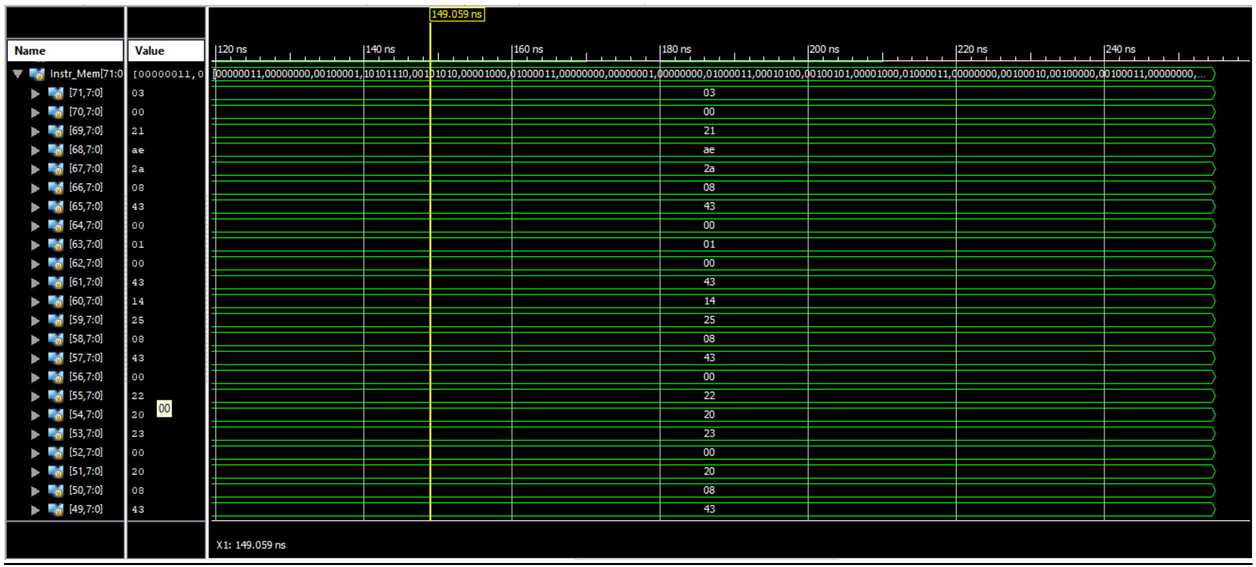
```
   end

endmodule
```

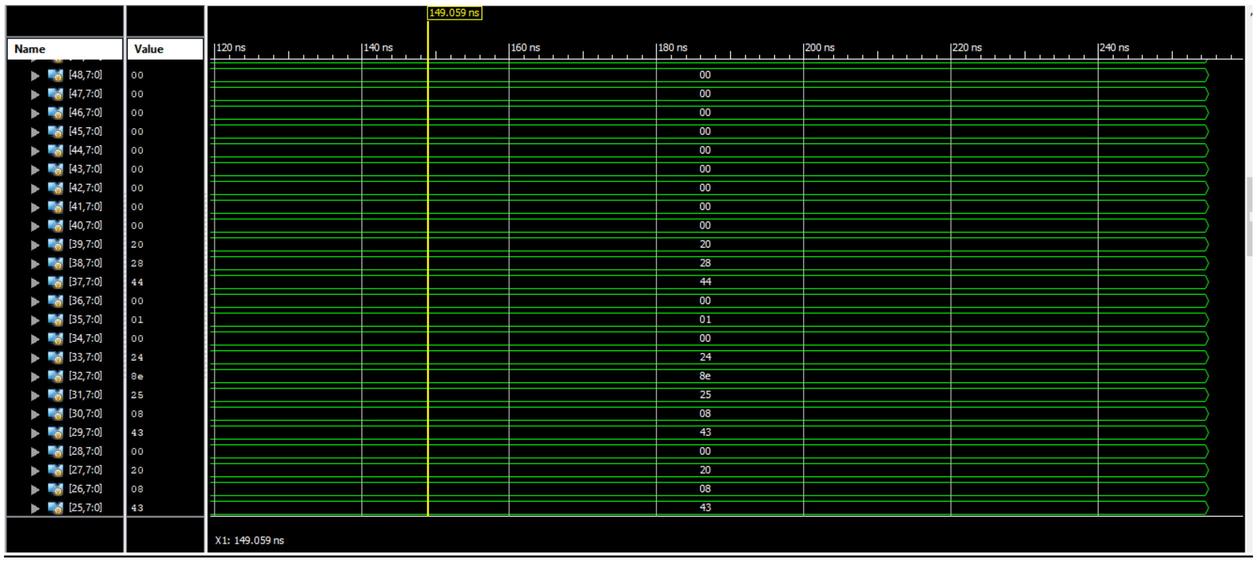
8) Paste screenshots of output waveforms.

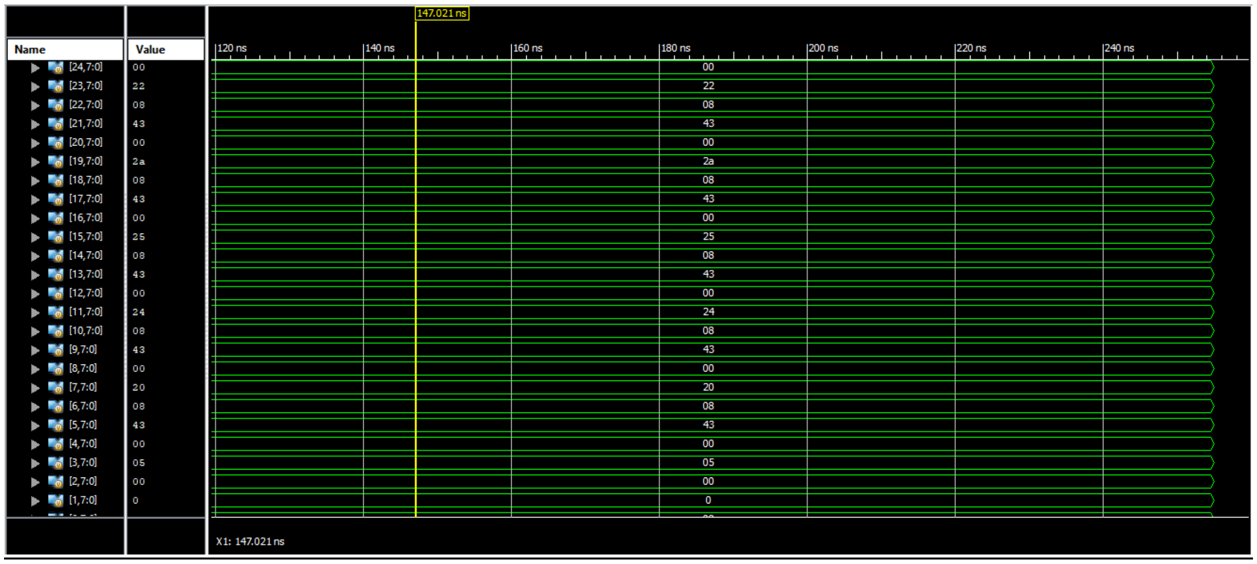
<u>**ALU Output, Flush, PC ,Forwarding unit and Stalling unit signals**</u>



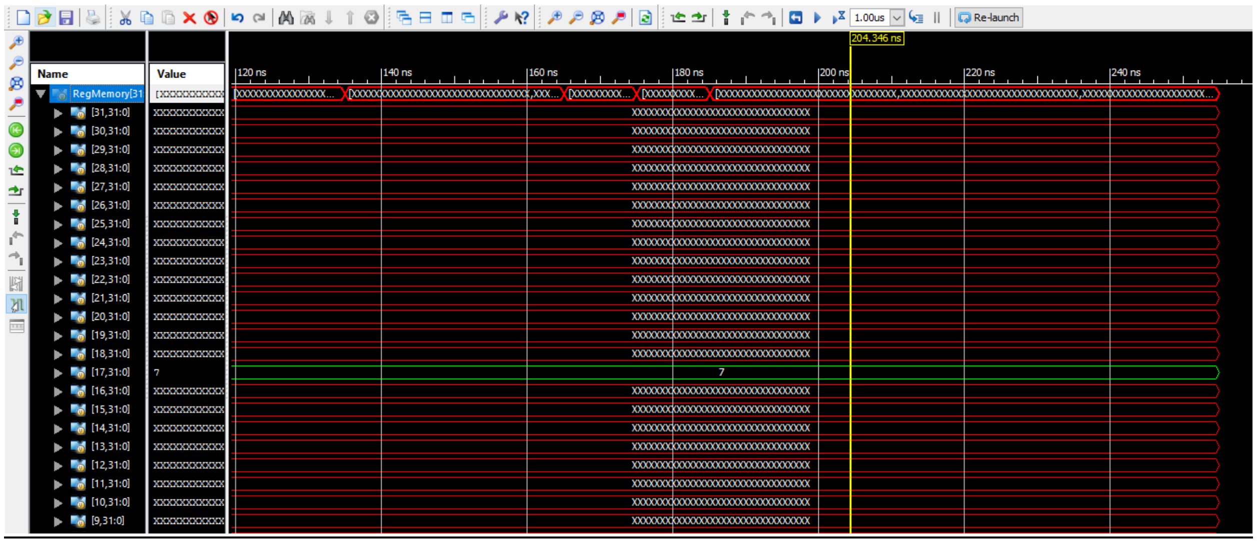
Instruction memory:

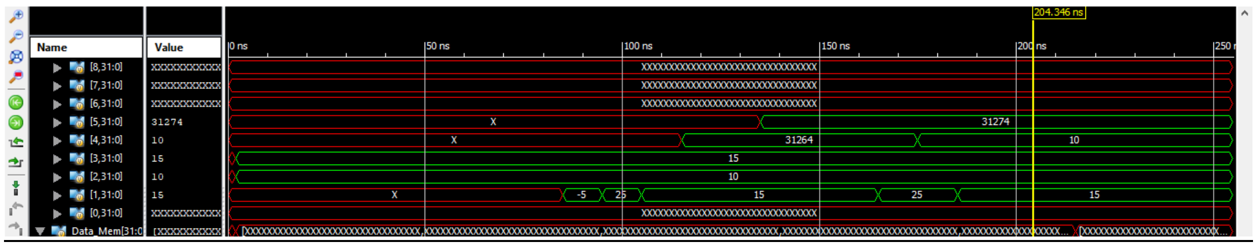



```
Name      Value   149.059 ns
          120 ns  140 ns  160 ns  180 ns  200 ns  220 ns  240 ns
[48,7:0]  00      00
[47,7:0]  00      00
[46,7:0]  00      00
[45,7:0]  00      00
[44,7:0]  00      00
[43,7:0]  00      00
[42,7:0]  00      00
[41,7:0]  00      00
[40,7:0]  00      00
[39,7:0]  20      20
[38,7:0]  28      28
[37,7:0]  44      44
[36,7:0]  00      00
[35,7:0]  01      01
[34,7:0]  00      00
[33,7:0]  24      24
[32,7:0]  8e      8e
[31,7:0]  25      25
[30,7:0]  08      08
[29,7:0]  43      43
[28,7:0]  00      00
[27,7:0]  20      20
[26,7:0]  08      08
[25,7:0]  43      43
X1: 149.059 ns
```

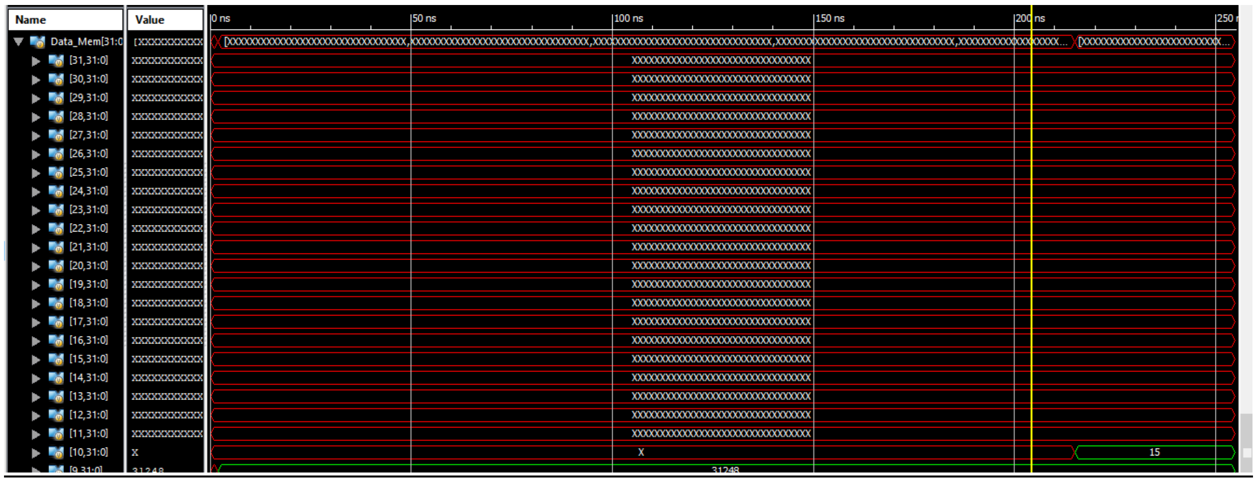

```
Name      Value   147.021 ns
          120 ns  140 ns  160 ns  180 ns  200 ns  220 ns  240 ns
[24,7:0]  00      00
[23,7:0]  22      22
[22,7:0]  08      08
[21,7:0]  43      43
[20,7:0]  00      00
[19,7:0]  2a      2a
[18,7:0]  08      08
[17,7:0]  43      43
[16,7:0]  00      00
[15,7:0]  25      25
[14,7:0]  08      08
[13,7:0]  43      43
[12,7:0]  00      00
[11,7:0]  24      24
[10,7:0]  08      08
[9,7:0]   43      43
[8,7:0]   00      00
[7,7:0]   20      20
[6,7:0]   08      08
[5,7:0]   43      43
[4,7:0]   00      00
[3,7:0]   05      05
[2,7:0]   00      00
[1,7:0]   0       0
X1: 147.021 ns
```

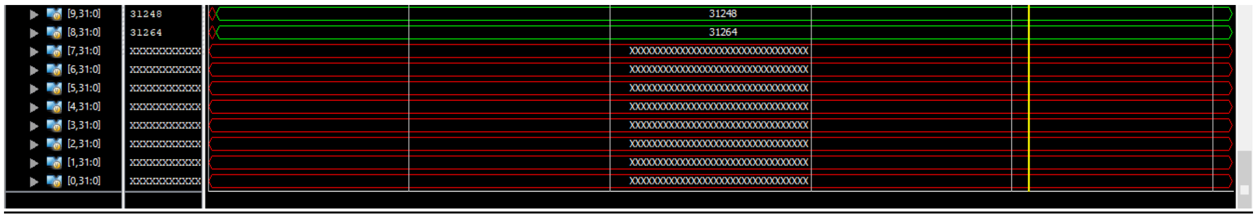
Register Memory





Data Memory:





<u>Part-B</u>

1) Upload all the. v and .mem files in a single zipped folder.

2) Upload the project file .ise or. prj. (Xilinx ISE/Vivado).

3) Upload this document after editing. (One document per group)

Note

1) Any help taken should be mentioned in the document, without which it will be considered a clear malpractice case, and **no marks** will be awarded.

2) Please stick to your respective lab groups.

Deadline: 23.04.2023 (11 p.m)