Assignment -3(Computer Architecture Lab)

Group no. P3, Grp. 8

Name and ID Student 1: Varun Santurkar, 2020APS1319H

Name and ID Student 2: Rithika Susarla, 2020AAPS13127H

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:

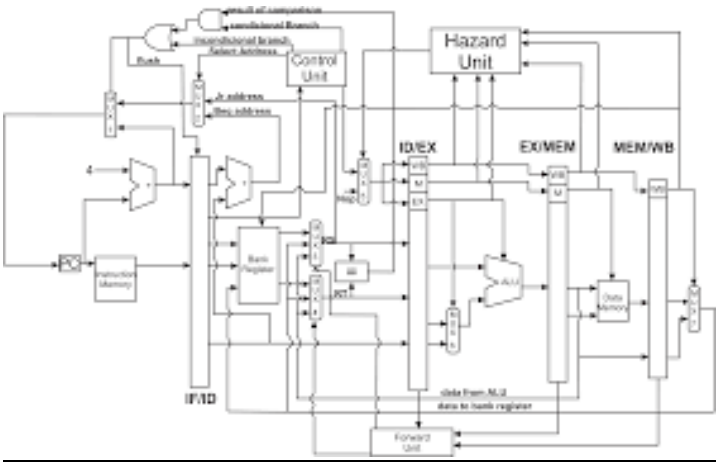
1) *The program should indicate the hazard points and the result stored in each register. (given in diagram for 6th question)*
2) *Mention the techniques used to overcome the hazards encountered in the program chosen.*

   Data hazards: Overcome by **forwarding** the value from ex for r-type instructions and from mem for load instructions. An additional **flush** is performed for load instructions to forward value from mem to ex stage.

   Control Hazards: Overcome by **stalling** the program for two operations until the branch address has been computed.

   Structural Hazards: By using **separate pipelined** stages and **separate data memory and instruction memory**, structural hazards have been eliminated.

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.

Opcode	operation	RegDst	RegWrite	AluSrc	MemWrite	aluop	MemtoReg	MemRead	jump	Branch
000000	r-type	1	1	0	0	10	0	0	0	0
100011	i-type (load)	0	1	1	0	00	1	1	0	0
101011	i-type (store)	0	0	1	1	00	0	0	0	0
000100	i-type (bne)	0	0	0	0	01	0	0	0	1
000101	i-type (beq)	0	0	0	0	11	0	0	0	1
001000	i-type (addi)	0	1	1	0	00	0	0	0	0
001001	i-type (subi)	0	1	1	0	01	0	0	0	0
000010	j-type (jump)	0	0	0	0	00	0	0	1	0
111111	NOP	0	0	0	0	00	0	0	0	0

ALU Control Signals:

aluop	funct	alucontrol
00	XXXXXX	0010
01	XXXXXX	0100
11	XXXXXX	0101

10	1000000	0010
10	100010	0110
10	100100	0000
10	100101	0001
10	101010	0111
10	010100	1000
10	010101	1001
10	010111	1010

ALU operations:

alucontrol	operation
0000	and
0001	or
0010	add
0110	sub
1000	sll
1001	set arithmetic/logical for sr
1010	decide barrel shifter direction
0100	zero for bne
0101	zero for beq

0111	slt

Instruction Formats:

1) R-type instructions:
   Format:

   Rd<= Rs op Rt

Opcode (6)	Rs (5)	Rt (5)	Rd (5)	Shamt (5)	Funct (6)

Opcode	**Funct**	**Shamt**	**Operation**
000000	100000	XXXXXX	Add
000000	100010	XXXXXX	Sub
000000	100100	XXXXXX	And
000000	100101	XXXXXX	Or
000000	101010	XXXXXX	Set Less Than
000000	010100	shamt	Left Shift
000000	010101	shamt	Right Shift
000000	010111	shamt	Barrel Shift

2) I-type instructions:

   Format:

   Rt <= mem(Rs+offset) OR Rt<=Rs op offset

Opcode (6)	Rs (5)	Rt (5)	Offset (16)

Opcode	Operation
100011	Load Word
101011	Store Word
000100	Branch Equal
000101	Branch Not Equal
001000	Add Immediate
001001	Sub Immediate

3) J-type instructions:

Format:

Opcode (6)	Jump Address (26)

Opcode	Operation
000010	Unconditional Jump

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

1) **Lw reg4, 2(reg1)** 100011 00001 00100 0000000000000010

2) **!Flush (NOP) to eliminate data hazard (mem to ex)** 11111100000000000000000000000000

3) **Lw reg5, 3(reg1)** 100011 00001 00101 0000000000000011

4) **! Flush (NOP) to eliminate data hazard (mem to ex)** 11111100000000000000000000000000

5) **Sw reg4, 2(reg0)** 101011 00000 00100 0000000000000010

6) **! Flush (NOP) to eliminate data hazard (mem to ex)** 11111100000000000000000000000000

7) **Beq reg4, reg6 to end** 000100 00100 00101 0000000000001111

8) **!1st Stall to eliminate control hazard** 11111100000000000000000000000000

9) **!2nd Stall to eliminate control hazard** 11111100000000000000000000000000

10) **Slt reg6, reg4, reg5** 000000 00100 00101 00110 00000 101010

11) **Beq reg6, reg0, to skip 3 instructions** 000100 00110 00000 0000000000000011

12) !Data hazard eliminated by forwarding

13) **!1st Stall to eliminate control hazard** 11111100000000000000000000000000

14) **!2nd Stall to eliminate control hazard** 11111100000000000000000000000000

15) **And reg5, reg5, reg4** 000000 00100 00101 00101 00000100100

16) **Or reg6, reg4, reg5** 000000 00100 00101 00110 00000 100101

17) !Data hazard eliminated by forwarding

18) **Sub reg5, reg5, reg4** 000000 00100 00101 00101 00000 100010

19) !Data hazard eliminated by forwarding

20) **Sub reg6, reg5, reg4** 000000 00101 00100 00110 00000 100010

21) **!Data hazard eliminated by forwarding**

22) **Jump to end** 000010 00000000000000000000000011

23) **!1st Stall to eliminate control hazard** 11111100000000000000000000000000

24) **!2nd Stall to eliminate control hazard** 11111100000000000000000000000000

25) **~Dummy ins to display functionality of jump~** 000000 00110 00101 00101 00000 100010

26) **~Dummy ins to display functionality of jump~** 000000 00111 00100 00110 00000 100010

27) **Jump to end** 000010 00000000000000000000000011

28) **!1st Stall to eliminate control hazard** 11111100000000000000000000000000

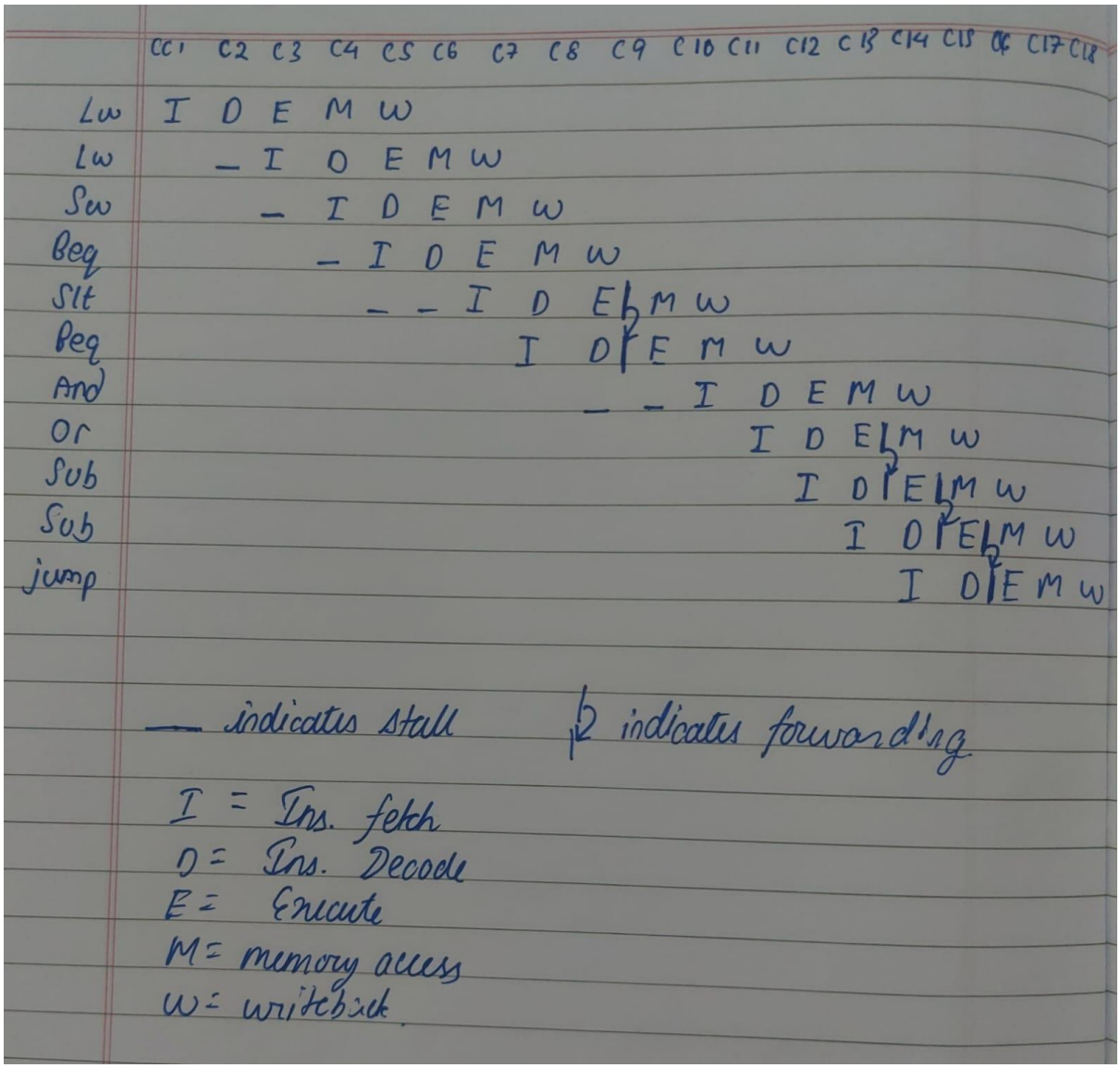
29) **!2nd Stall to eliminate control hazard** 11111100000000000000000000000000

30) **~Dummy ins to display functionality of jump~** 000000 00110 00101 00101 00000 100010

31) **~Dummy ins to display functionality of jump~** 000000 00111 00100 00110 00000 100010

32) **-end-**

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



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

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 30.03.2023 18:22:42
// Design Name:
// Module Name: main
// Project Name:
// Target Devices:
// Tool Versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module instrmem(pc, instructions);
input [31:0] pc;
output wire [31:0] instructions;
reg [7:0] Mem [128:0];
initial $readmemb("ins.mem",Mem);
assign instructions={Mem[pc], Mem[pc+1], Mem[pc+2], Mem[pc+3]};
endmodule

module logic_right_shift(
   output [31:0]out,
   input [31:0]in,
   input arith,
   input [5:0] s
   );
   wire [31:0]w0,w1,w2,w3,w4;
   wire z;
   assign z = arith & in[31];
   mux_2_1_32b u0(w0,in,{32{z}},s[5]);
   mux_2_1_32b u1(w1,w0,{{16{z}}, w0[31:16]},s[4]);
   mux_2_1_32b u2(w2,w1,{{8{z}}, w1[31:8]},s[3]);
   mux_2_1_32b u3(w3,w2,{{4{z}}, w2[31:4]},s[2]);
   mux_2_1_32b u4(w4,w3,{{2{z}}, w3[31:2]}, s[1]);
   mux_2_1_32b u5(out,w4,{z, w4[31:1]}, s[0]);
endmodule

module logic_left_shift(
   output [31:0]out,
   input [31:0]in,
   input [5:0] s
   );

   wire [31:0]w0,w1,w2,w3,w4;
   wire z;

   assign z=0;

   mux_2_1_32b u0(w0,in,{32{z}},s[5]);
   mux_2_1_32b u1(w1,w0,{in[15:0], {16{z}}},s[4]);
```

```
   mux_2_1_32b u2(w2,w1,{w1[23:0],{8{z}}},s[3]);
   mux_2_1_32b u3(w3,w2,{w2[27:0],{4{z}}},s[2]);
   mux_2_1_32b u4(w4,w3,{w3[29:0],{2{z}}},s[1]);
   mux_2_1_32b u5(out,w4,{w4[30:0],z},s[0]);

endmodule


module barrel_shifter(
   output [31:0]out0,
   output [31:0]out1,
   output [31:0]out,
   input [31:0]in,
   input arith,
   input [5:0] s,
   input sel
   );

   wire [31:0]w1,w2;

   logic_right_shift u1(out0,in,arith,s);
   logic_left_shift u2(out1,in,s);
   mux_2_1_32b u3(out,out0,out1,sel);
endmodule

module mux_2_1_32b#(parameter N=32)(
   output [N-1 : 0] out,
   input [N-1 : 0] in0,
   input [N-1 : 0] in1,
   input sel
   );
   assign out  = sel ? in1 : in0;
endmodule

 module SCDataPath(
 input clk, input [31:0] PCin,  input reset,// main inputs
 output reg [31:0] result, output wire [31:0] instruction,
 output wire [31:0] if_output, output wire [31:0] operand_1, output wire [31:0] operand_2,
output wire [31:0] aluout,
 output wire [31:0] read_data, output wire [31:0] reg4, output wire [31:0] reg5, output wire
[31:0] reg6,
 output wire [31:0] mem1, output wire [31:0] mem2, output wire [31:0] mem3,output wire
[31:0] pcout);

reg [3:0] alucontrol;

reg lr; reg ar; //control signals
reg [1:0] aluop; //reg MemtoReg; reg MemRead;
```

```
reg [1:0] PCsrc;
wire [31:0] pcmodjump; wire [31:0] pcmodbranch;


reg [31:0] aluin2; //output of alusrc mux

reg [7:0] dMem [31:0]; //Data Memory
wire [31:0] instructions; wire [31:0] Lshiftres; wire [31:0] Rshiftres;
wire [31:0] Bshiftres; wire [31:0] Bshiftres1; wire [31:0] Bshiftres2; //wires for calculating shifter results


initial $readmemb("data.mem",dMem); //read data memory

//reg [5:0] opcode;
//reg [5:0] funct;
//reg [5:0] shamt;
//reg [4:0] Read_Reg_Num_1;
//reg [4:0] Read_Reg_Num_2;
//reg [4:0] Write_Reg_Num;
//initial Write_Reg_Num<=5'b0;
reg [31:0] extended; //register for sign extended offset
initial extended<=32'b0;// instruction sections


//Pipeline registers
reg [120:0] ifid;
reg [240:0] idex;
reg [200:0] exmem;
reg [200:0] memwb;

initial begin
ifid=120'b0;
idex=240'b0;
exmem=200'b0;
memwb=200'b0;
end

reg [31:0] pc;
reg [31:0] regmem [31:0];

initial pc<=PCin;

// reset function
always @(reset)
begin
if (reset)
begin
```

```
 pc<=32'b0;
// jumpAdd<=32'b0;
// BranchAdd<=32'b0;
// jump<=0;
// zero<=0;
// Branch<=0;
// shamt<=6'b0;
 lr<=0;
 ar<=0;
 regmem[0] <= 32'h0; //always zero, do not use for any other computation
 regmem[1] <= 32'h0;//64767000;
 regmem[2] <= 32'h0;//ha4d3a4a3;
 regmem[3] <= 32'h0;//90203898;
 regmem[4] <= 32'h0;//90203898;
 regmem[5] <= 32'h0;//99554407;
 regmem[6] <= 32'h0;
 regmem[7] <= 32'h0;
 regmem[8] <= 32'h0;
 regmem[9] <= 32'h0;
 regmem[10] <= 32'h0;
 regmem[11] <= 32'h0;
 regmem[12] <= 32'h0;
 end
 end



// start of IF stage //
instrmem insr(pc, instructions);

always @(posedge clk)
pc<=pc+3'b100;

always@(posedge clk, PCsrc)
begin
if (PCsrc==2'b01) // for branch
pc[31:0]<=pcmodbranch[31:0];
else if (PCsrc==2'b11) //for jump
pc[31:0]<=pcmodjump[31:0];
end

//now write to If/ID pipeline register
always@(posedge clk) begin
ifid[31:0]<=instructions[31:0];
ifid[63:32]<=pc[31:0];
end
// end of IF stage //
```

```
// beginning of ID stage //

//idex components

// idex[31:0]=data1
// idex[63:32]=data2
// idex[95:64]=extended
// idex[100:96]=writereg1
// idex[105:101]=writereg2
// idex[137:106]=pc
// idex[138]=RegDst
// idex[139]=AluSrc
// idex[140]=MemWrite
// idex[142:141]=aluop
// idex[143]=MemtoReg
// idex[144]=MemRead
// idex[145]=jump
// idex[146]=Branch
// idex[147]=RegWrite
// idex[179:148]=jumpadd
//idex[211:180]=branchadd
//idex[217:213]=shamt
//idex[222:218]=regnum for data1 or ifid[25:21]
//idex[227:223]=regnum for data2 or ifid[20:16]

always@(posedge clk)
 begin
 idex[31:0]<=regmem[ifid[25:21]]; //data1
 idex[63:32]<=regmem[ifid[20:16]]; //data2
 idex[222:218]<=ifid[25:21]; //data1 reg number
 idex[227:223]<=ifid[20:16]; //data2 regnumber
 idex[100:96]<=ifid[20:16];
 idex[105:101]<=ifid[15:11];
 idex[137:106]<=ifid[63:32];
end

always @(posedge clk)
begin
idex[79:64]<= ifid [15:0];
idex[217:213]<= ifid[10:6];
idex[175:148]<=ifid[25:0]<<2;
idex[195:182]<=ifid[13:0];
end
```

```
always@(posedge clk)
begin
idex[179:176]<=ifid[63:60];
end

//generate control signals
always @(posedge clk)
begin
//opcode<=ifid[31:26];
 case (ifid[31:26])
 6'b000000://r-type
 begin
 idex[138]<=1;
 idex[147]<=1;
 idex[139]<=0;
 idex[140]<=0;
 idex[142:141]<=2'b10;
 idex[143]<=0;
 idex[144]<=0;
 idex[145]<=0;
 idex[146]<=0;
 end

 6'b100011://load
 begin
 idex[138]<=0;
 idex[147]<=1;
 idex[139]<=1;
 idex[140]<=0;
 idex[142:141]<=2'b00;
 idex[143]<=1;
 idex[144]<=1;
 idex[145]<=0;
 idex[146]<=0;
 end

 6'b101011://store
 begin
 idex[138]<=0;
 idex[147]<=0;
 idex[139]<=1;
 idex[140]<=1;
 idex[142:141]<=2'b00;
 idex[143]<=0;
 idex[144]<=0;
 idex[145]<=0;
 idex[146]<=0;
 end
```

```
6'b000010://jump
begin
idex[138]<=0;
idex[147]<=0;
idex[139]<=0;
idex[140]<=0;
idex[142:141]<=2'b00;
idex[143]<=0;
idex[144]<=0;
idex[145]<=1;
idex[146]<=0;
end

6'b000100://bne
begin
idex[138]<=0;
idex[147]<=0;
idex[139]<=0;
idex[140]<=0;
idex[142:141]<=2'b01;
idex[143]<=0;
idex[144]<=0;
idex[145]<=0;
idex[146]<=1;
end

6'b000101://beq
begin
idex[138]<=0;
idex[147]<=0;
idex[139]<=0;
idex[140]<=0;
idex[142:141]<=2'b11;
idex[143]<=0;
idex[144]<=0;
idex[145]<=0;
idex[146]<=1;
end

 6'b001000: //addi
begin
idex[138]<=0;
idex[147]<=1;
idex[139]<=1;
idex[140]<=0;
idex[142:141]<=2'b00;
idex[143]<=0;
```

```
idex[144]<=0;
idex[145]<=0;
idex[146]<=0;
end

6'b001001: //subi
begin
idex[138]<=0;
idex[147]<=1;
idex[139]<=1;
idex[140]<=0;
idex[142:141]<=2'b01;
idex[143]<=0;
idex[144]<=0;
idex[145]<=0;
idex[146]<=0;
end

6'b111111: //nop
begin
idex[138]<=0;
idex[147]<=0;
idex[139]<=0;
idex[140]<=0;
idex[142:141]<=2'b00;
idex[143]<=0;
idex[144]<=0;
idex[145]<=0;
idex[146]<=0;
end

endcase
end
// end of ID stage //



//start of EX stage //


//exmem components

//exmem[31:0] pcmod for branch
//exmem[63:32] data2
//exmem[95:64] result
//exmem[96] zero
//exmem[128:97] branchadd
```

```
//exmem[129] regwrite
//exmem[130] branch
//exmem[131] memread
//exmem[132] memwrite
//exmem[137:133] writeregnum
//exmem[169:138] pcmod for jump
//exmem[170] jump [170]
//exmem[171] memtoreg idex[143] [171]




always@(posedge clk)begin
exmem[31:0]<=pc+idex[211:180];//idex[137:106]+(idex[211:180]); //final pc for branch
exmem[169:138]<=pc+idex[179:148];//idex[137:106]+(idex[179:148]); //final pc for jump
end
//generate alucontrol
always @(posedge clk, instructions)
begin
case(idex[142:141])
2'b00: alucontrol<=4'b0010;
2'b01: alucontrol<=4'b0100;
2'b11: alucontrol<=4'b0101;
2'b10:
if (idex[69:64]==6'b100000)
   alucontrol<=4'b0010;
else if (idex[69:64]==6'b100010)
   alucontrol<=4'b0110;
else if (idex[69:64]==6'b100100)
   alucontrol<=4'b0000;
else if (idex[69:64]==6'b100101)
   alucontrol<=4'b0001;
else if (idex[69:64]==6'b101010)
   alucontrol<=4'b0111;
else if (idex[69:64]==6'b010100)
   alucontrol<=4'b1000;
else if (idex[69:64]==6'b010101)
   alucontrol<=4'b1001;
else if (idex[69:64]==6'b010111)
   alucontrol<=4'b1010;
endcase
end

assign pcmodjump[31:0]= exmem[170]? exmem[169:138]+3'b100: 32'b0;
assign pcmodbranch[31:0]= (exmem[130]&&exmem[96]) ? exmem[31:0]+3'b100: 32'b0;
```

```
//regdst mux

always@(posedge clk)
begin
 if (idex[138])
   exmem[137:133]<=idex[105:101];
 if (idex[138]!=1)
   exmem[137:133]<=idex[100:96];
 end

//beginning of forwarding stage //
reg [31:0] aluout1;
reg [31:0] aluout2;
initial begin
aluout2=0;
aluout1=0;
end
always @(*) begin
if(idex[227:223]!=0)
aluout2[5:0]<=idex[227:223];
if(idex[222:218]!=0)
aluout1[5:0]<=idex[222:218];
end

reg [31:0] idex_input1;
reg [31:0] idex_input2;
initial begin
 idex_input1=0;
 idex_input2=0;
 end

always @(*)begin
   if (memwb[71]) begin //using memwrite from exmem as a condition for forwarding memwb
or exmem
      if (aluout2[5:0]==memwb[36:32]!=0) begin
         idex_input2[31:0]<=memwb[68:37];
         idex_input1[31:0]<=idex[31:0];
         end
      else if(aluout1[5:0]==memwb[36:32]!=0) begin
         idex_input1[31:0]<=memwb[68:37];
         idex_input2[31:0]<=idex[63:32];
         end
      else begin
         idex_input1[31:0]<=idex[31:0];
         idex_input2[31:0]<=idex[63:32];
         end
   end
   else
```

```
      if (aluout2[5:0]==exmem[137:133]!=0) begin
         idex_input2[31:0]<=exmem[95:64];
         idex_input1[31:0]<=idex[31:0];
         end
      else if(aluout1[5:0]==exmem[137:133]!=0) begin
         idex_input1[31:0]<=exmem[95:64];
         idex_input2[31:0]<=idex[63:32];
         end
      else begin
         idex_input1[31:0]<=idex[31:0];
         idex_input2[31:0]<=idex[63:32];
         end
end

 always@(posedge clk, alucontrol, idex[139], idex_input2[31:0])//, resultemp[31:0], aluin2)
begin
if (idex[139])
aluin2<=idex[95:64];
else
aluin2<=idex_input2[31:0];

end

wire [31:0] Out_2;
assign Out_2=idex_input2[31:0];
logic_left_shift ls(Lshiftres,Out_2, idex[217:213]);
logic_right_shift rs(Rshiftres,Out_2,ar,idex[217:213]);
barrel_shifter bs(Bshiftres1,Bshiftres2,Bshiftres,idex_input2[31:0], ar, idex[217:213], lr); //pre
calculate shift values



//alu operations
reg [31:0] resultemp; //temporary register in order to maintain synchronization
always@(posedge clk, alucontrol, idex_input1[31:0], aluin2)
begin
case(alucontrol)
4'b0000:resultemp[31:0]<= idex_input1[31:0]&aluin2;
4'b0001:resultemp[31:0]<= idex_input1[31:0]|aluin2;
4'b0010:resultemp[31:0]<= idex_input1[31:0]+aluin2;
4'b0110:resultemp[31:0]<= idex_input1[31:0]-aluin2;
4'b1000: resultemp[31:0]<=Lshiftres;
4'b1001:
begin
if(idex_input1[31:0]%2==0)
ar<=1;
else
ar<=0;
```

```
resultemp[31:0]<=Rshiftres;
end


4'b1010:
begin
if(idex_input1[31:0]%2==0)
begin
lr<=1;
ar<=1;
end
else
begin
lr<=0;
ar<=0;
resultemp[31:0]<=Bshiftres;
end
end


4'b0100:
begin
resultemp[31:0]<= idex_input1[31:0]-aluin2;
if (resultemp[31:0])
exmem[96]<=0;
else
exmem[96]<=1;
end


4'b0101:
begin
resultemp[31:0]<= idex_input1[31:0]-aluin2;
if (resultemp[31:0])
exmem[96]<=1;
else
exmem[96]<=0;
end

4'b0111:

if (idex_input1[31:0]<aluin2)
resultemp[31:0]<=1;
else
resultemp[31:0]<=0;
endcase
end
```

```
always@(posedge clk) begin
exmem[95:64]<=resultemp[31:0];
//forwarded signals from idex register
exmem[132]<=idex[140]; //Memwrite
exmem[131]<=idex[144]; //Memread
exmem[129]<=idex[147]; //regwrite
exmem[130]<=idex[146]; //Branch
exmem[170]<=idex[145]; //jump
exmem[63:32]<=idex_input2[31:0]; //data2
exmem[171]<=idex[143]; // memtoreg
//exmem[137:133]<=Write_Reg_Num;
end //used in order to ensure the stages are synchronized
// end of EX stage //



// start of MEM stage //

//memwb addersses
//memwb[31:0] alu result
//memwb[36:32] output of regdst mux, exmem[137:133]
//memwb[68:37] read data
//memwb[69] regwrite, exmem[129]
//memwb[70] memtoreg exmem[171]
//memwb[71] memread used as decider for forwarding block

always@(exmem[170], exmem[130], exmem[96])
begin
if (exmem[170])
PCsrc<=2'b11;
else if (exmem[130] && exmem[96] ==1)
PCsrc<=2'b01;
else
PCsrc<=2'b00;
end

always @(posedge clk)
begin
memwb[71]<=exmem[131];
memwb[69]<=exmem[129];
memwb[70]<=exmem[171];
memwb[36:32]<=exmem[137:133];
memwb[31:0]<=exmem[95:64];
end

always@(posedge clk)//, exmem[132], exmem[131])
begin
if (exmem[132])
```

```
{dMem[(4*memwb[31:0])], dMem[(4*memwb[31:0])+1], dMem[(4*memwb[31:0])+2], dMem[(4*memwb[31:0])+3]}<=exmem[63:32]; //write
if (exmem[131])
memwb[68:37]<={dMem[(4*exmem[95:64])], dMem[(4*exmem[95:64])+1], dMem[(4*exmem[95:64])+2], dMem[(4*exmem[95:64])+3]};
end
//end of MEM stage //


//start of WB stage//

//write to registers
always @(posedge clk)//, memwb[70], memwb[31:0], memwb[69])
begin

if (memwb[69])
begin
     if(memwb[70])
      regmem[memwb[36:32]]<=memwb[68:37];
      else
      regmem[memwb[36:32]]<=memwb[31:0];
end
end


 //testing code to view memory and registers in waveforms
assign instruction=instructions;
assign if_output=ifid[31:0];
assign operand_1=idex_input1[31:0];
assign operand_2=aluin2;
assign aluout=resultemp[31:0];//exmem[95:64];//resultemp[31:0];//
assign read_data=memwb[68:37];
assign reg4=regmem[4];
assign reg5=regmem[5];
assign reg6=regmem[6];
assign mem1={dMem[4], dMem[5], dMem[6], dMem[7]};
assign mem2={dMem[8], dMem[9], dMem[10], dMem[11]};
assign mem3={dMem[12], dMem[13], dMem[14], dMem[15]};
assign pcout=alucontrol;
 endmodule
```

TESTBENCH:

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
```

```
//
// Create Date:   12:10:39 03/30/2023
// Design Name:
// Module Name:
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: insfetch
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module SCcpu_tb;

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

	// Outputs
wire [31:0] instructions;
wire [31:0] ALUout; wire [31:0] if_output; wire [31:0] operand_1; wire [31:0] operand_2; wire
[31:0] aluout;
wire [31:0] read_data; wire [31:0] reg4; wire [31:0] reg5; wire [31:0] reg6;
wire [31:0] mem1; wire [31:0] mem2; wire [31:0] mem3; wire [31:0] pcout;
	// Instantiate the Unit Under Test (UUT)
	SCDataPath uut (
		.clk(clk),
		.PCin(PCin),
		.reset(reset),
		.instruction(instructions),
		.result(ALUout),
		.if_output(if_output),
		.operand_1(operand_1),
		.operand_2(operand_2),
		.aluout(aluout),
		.read_data(read_data),
		.reg4(reg4),
		.reg5(reg5),
		.reg6(reg6),
		.mem1(mem1),
		.mem2(mem2),
		.mem3(mem3),
```

```
                .pcout(pcout)

        );


                //
                initial begin
                reset = 1'b1;
                #10 reset = 1'b0;
                #8 reset = 1'b0;
                #10 reset = 1'b0;
                end
                //start PC from first instruction
                initial begin
                PCin = 32'b0;
                end

                initial begin
                clk = 1'b0;
                repeat(60) #5 clk = ~clk;
                $finish;
                end

endmodule
```

DATA.MEM:

```
01100100
01110110
01110000
00000001
10100100
11010011
10100100
10100011
10010000
00100000
00111000
10011000
10011001
01000100
01010101
00000111
01100100
01110110
01110000
00000001
11100100
```

11110110
11110000
10000001
01010000
00101100
00111100
10011110
11111001
01011100
01111111
11111111

INS.MEM

10001100
00100100
00000000
00000010

11111100
00000000
00000000
00000000

10001100
00100101
00000000
00000011

11111100
00000000
00000000
00000000

10101100
00000100
00000000
00000010

11111100
00000000
00000000
00000000

00010000
10000101
00000000
00001111

11111100
00000000
00000000
00000000

11111100
00000000
00000000
00000000

00000000
10000101
00110000
00101010

00010100
11000000
00000000
00000011

11111100
00000000
00000000
00000000

11111100
00000000
00000000
00000000

00000000
10000101
00101000
00100100

00000000
10000101
00110000
00100101

00000000
10000101
00101000
00100010

00000000
10100100

00110000
00100010

00010000
00000000
00000000
00000111

11111100
00000000
00000000
00000000

11111100
00000000
00000000
00000000

00000000
11000101
00101000
00100010

00000000
11100100
00110000
00100010

00010000
00000000
00000000
00000011

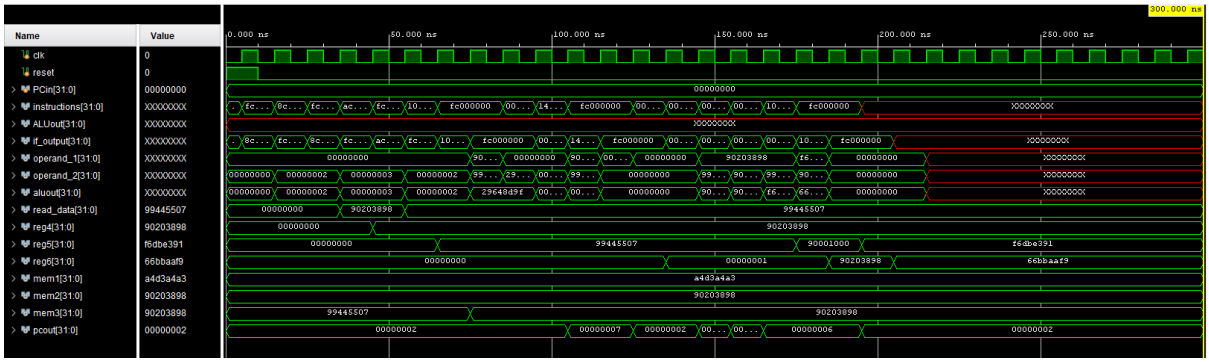
11111100
00000000
00000000
00000000

11111100
00000000
00000000
00000000

00000000
11000101
00101000
00100010

00000000
11100100
00110000
00100010

8) Paste screenshots of output waveforms.



Part-B

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)