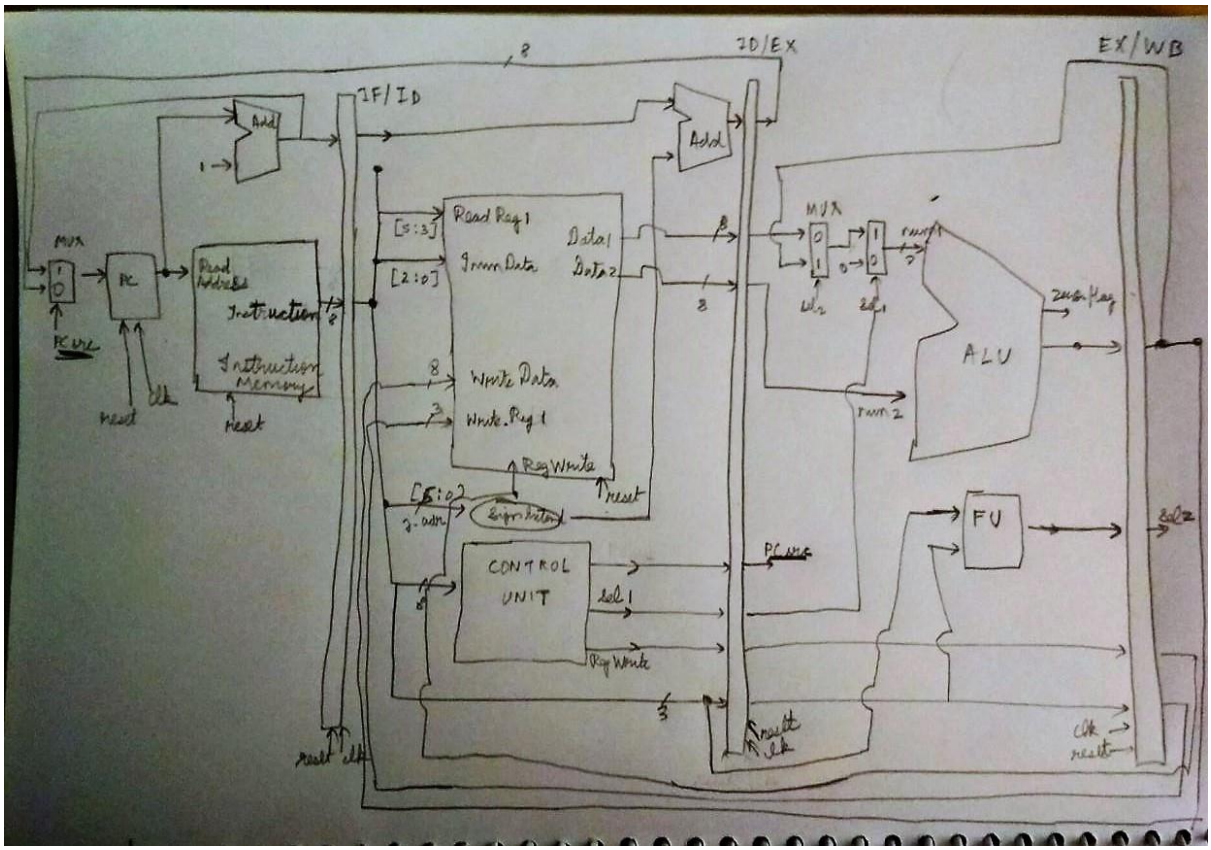
Name: Rishabh Jain

Id- 2015A3PS0232G

Ans1



Ans2

Instructions	Control Signals		
	Sel1	PCsrc	RegWrite
li	0	1	1
addi	1	1	1
j	X	0	0

Another signal is Sel2 which is generated by Forwarding Unit. This signal is 1 when forwarding happens else 0.

Ans3

IFetch’s top module

```
1  //this module will be complete instruction fetch unit
2  module ifetch(clk, reset,PC_j,PCsrc, inst_code_out, PCline_out );
3  input clk, reset;
4  input [7:0]PC_j;
5  input PCsrc;
6  output [7:0]PCline_out;
7  output [7:0]inst_code_out;
8  wire [7:0] PCin, PCout, inst_code;
9  wire [7:0] PC_d, out2, PCline;
10 pc x1(PC_d, clk, reset,PCout);
11 im x2(PCout, reset, inst_code );
12 adder x3(PCout, PCin );
13 if_id reg1(PCin,inst_code, clk, reset, PCline, out2, PCsrc);
14 assign PC_d = (PCsrc == 1)? (PCin):(PC_j);
15 assign inst_code_out = out2;
16 assign PCline_out = PCline;
17 endmodule
18
```

```
1  module pc(PCin, clk, reset,PCout);
2  input [7:0] PCin;
3  input clk, reset;
4  output reg [7:0]PCout;
5  always @(posedge clk, negedge reset )
6  begin //if reset == 0, make PC = 0
7  if(reset == 0)
8  PCout <=0;
9  else //incerement PC by 4
10 PCout <= PCin;
11 end
12 endmodule
13
```

```
module im(PCout, reset, inst_code );
input [7:0] PCout;
input reset;
output [7:0] inst_code;
reg [7:0] Mem[0:7]; // this creates 8 memory location each having 1 byte size
assign inst_code  = Mem[PCout];
//this is big endian format as msb is lower address and lsb is high address
// we are reading the instrution code for which address is specified byPC
always @(negedge reset)
begin
if (reset == 0)
// if 0 then initialize the memory with 4 instructions
begin
Mem[0] = 8'h13; //li r2, 3
Mem[1] = 8'h52; //addi r2, 2
Mem[2] = 8'h6B; //addi r5, 3
Mem[3] = 8'hC5; // j L1
Mem[4] = 8'h29;// li r5, 4
//Mem[5] = 8'h6A;//addi r5, 2 01101010//7
Mem[5] = 8'h6D;  //addi r5, -3 01101101
end
end
endmodule
```

```
//this block is part of instruction fetch unit, it specifically increments PC by 1
module adder(PCout, PCin );
input [7:0]PCout;
output [7:0]PCin;
assign PCin = PCout + 1;
endmodule
```

```
module if_id( //IF/ID REGISTER
PCin,inst_code, clk, reset, PCline, out2, PCsrc );
input [7:0] PCin, inst_code;
input clk, reset, PCsrc;
output reg[7:0] PCline,out2;
always @(posedge clk, negedge reset, PCsrc)
begin
if(reset == 0 || PCsrc == 0)
begin
PCline <= 0;
out2 <=8'b10111111;
end
else
begin
PCline <= PCin;
out2 <= inst_code;
end
end
endmodule
```

Ans4 : Register File

```
module id(PCid_in, j_adr, Read_Reg_Num_1,ImmData,reset,
Write_Reg_Num, Write_Data,RegWrite, Read_Data_1, Read_Data_2,PCid_out);
// let all the control signals be generated in the id unit and then these signals will be passed to later registers.
//declare the inputs and outputs
input [2:0] Read_Reg_Num_1,Write_Reg_Num; // total  8 registers
input [7:0] Write_Data, PCid_in;
input RegWrite, reset;
input [5:0] j_adr;
input [2:0] ImmData;
output [7:0] Read_Data_1;
output [7:0] Read_Data_2, PCid_out;
//output RegWrite, Sel1, PCsrc;

//define the functionality
reg [7:0] Reg_Mem[7:0];
assign Read_Data_1 = Reg_Mem[Read_Reg_Num_1];//goes to a
assign Read_Data_2 = {{5{ImmData[2]}},ImmData}; //sign extend range is from -4 to 3
assign PCid_out = {PCid_in[7:6],j_adr};
always @( negedge reset, Write_Reg_Num, Write_Data)
begin
    if(reset == 0)
        begin
Reg_Mem[0] = 8'h00;
Reg_Mem[1] = 8'h01;
Reg_Mem[2] = 8'h02;
Reg_Mem[3] = 8'h03;
Reg_Mem[4] = 8'h04;
Reg_Mem[5] = 8'h05;
Reg_Mem[6] = 8'h06;
Reg_Mem[7] = 8'h07;
        end
if( RegWrite == 1)
    begin
    Reg_Mem[Write_Reg_Num] <= Write_Data;
    end
end

endmodule
```

Ans5

The condition to detect hazard : check the register number of register in Execute stage and the register number going to enter execute stage. If the register number are same with opcode not being for jump instruction, then hazard condition is true.

Ans6

Forwarding Unit:

```
module f_unit(reg_out_ex, reg_in_ex, Sel2);
input [2:0]reg_out_ex, reg_in_ex;
output Sel2;
assign Sel2 = (reg_out_ex == reg_in_ex)? (1):(0);
endmodule
```

Ans7

Top Module

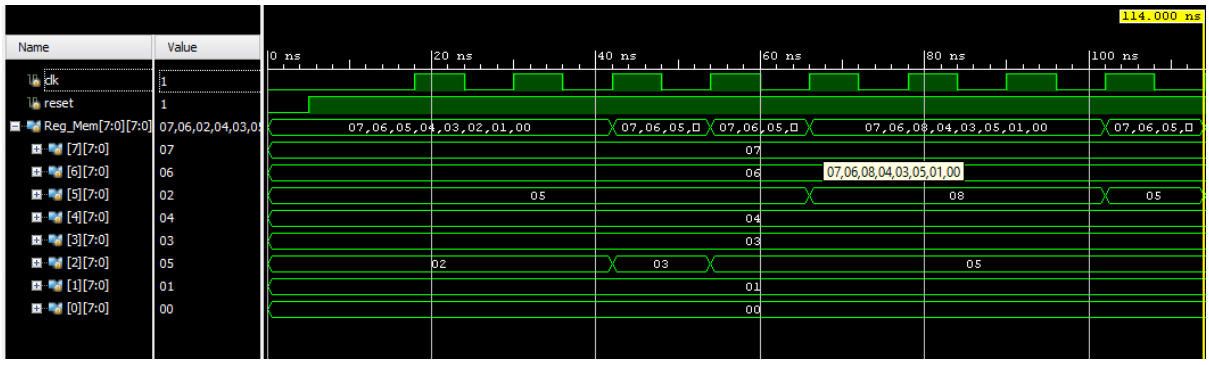
```
module top(clk, reset );
input clk, reset;
wire [7:0] inst_code_out;
wire [2:0] Read_Reg_Num_1,Write_Reg_Num;
wire [2:0] ImmData;
wire [7:0] Write_Data;
wire [7:0]  Read_Data_1, Read_Data_2;
wire [5:0] j_adr,j_adr_out;
wire [7:0] inst_code_idex_in, inst_code_idex_out,inst_code_ex_wb_out;
wire [7:0]PC_j,PCline_out;
wire [7:0]num1, ImmData_out,num11,num111;
wire PCsrc, Sel1, RegWrite, RegWrite_exwb_out;
wire zeroflag;
wire [7:0] aluRes,aluRes_out;
//wire [1:0] opcode;
wire [7:0] PCline_idex_in;
wire PCsrc_idex_out, RegWrite_idex_out, Sel1_idex_out,Sel2,Sel2_later;
wire [2:0]reg_in_ex,reg_out_ex;

ifetch mod1(clk, reset,PC_j,PCsrc_idex_out, inst_code_out, PCline_out ); // this module initializes ifunit and if/id reg

id mod2( PCline_out,j_adr,Read_Reg_Num_1,ImmData,reset,
Write_Reg_Num, Write_Data,RegWrite_exwb_out, Read_Data_1, Read_Data_2,PCline_idex_in);

id_ex reg2(
Sel1, RegWrite,PCsrc,inst_code_idex_in, Read_Data_2, Read_Data_1, j_adr,PCline_idex_in,clk, reset,
num111,PC_j, ImmData_out,inst_code_idex_out, j_adr_out, PCsrc_idex_out, RegWrite_idex_out, Sel1_idex_out);

alu mod3(
num1,ImmData_out, zeroflag,aluRes);

ex_wb reg3(Sel2,RegWrite_idex_out,clk,reset, aluRes,inst_code_idex_out,aluRes_out,inst_code_ex_wb_out,RegWrite_exwb_out,Sel2_later );
//change the jump address calculation
//control mod4(opcode,Sel1, PCsrc,RegWrite);
f_unit mod4(reg_out_ex,reg_in_ex, Sel2);
```

```

//assign opcode = inst_code_out[7:6];
assign Read_Reg_Num_1 = inst_code_out[5:3];
assign ImmData = inst_code_out[2:0];
assign Write_Data = aluRes_out;//paste the write data here
assign j_adr = inst_code_out[5:0];
assign inst_code_idex_in = inst_code_out;
assign reg_out_ex = inst_code_idex_out[5:3];
assign  reg_in_ex = inst_code_idex_in[5:3];
assign num111 = (Sel2_later)? (aluRes_out):(num111);
assign num1 = (Sel1_idex_out == 0) ? 8'h00 : num11 ;
assign Write_Reg_Num = inst_code_ex_wb_out[5:3];
assign Sel1 = (inst_code_out[7:6] == 2'b00)? (0):(1); // all these control signals will pass through the registers to their destination
assign PCsrc = (inst_code_out[7:6]==2'b11)? (0):(1);
assign RegWrite = (inst_code_out[7:6]==2'b00 || inst_code_out[7:6]==2'b01)? (1):(0);

endmodule

```

Ans8

```
1  module test;
2  reg clk, reset;
3
4  top test1(clk, reset );
5  initial begin
6  clk = 0;
7  #12
8  repeat(18)
9  begin
10 #6 clk = ~clk; //this generates four clock cycles with period of 10 units
11 end
12 end
13
14 initial begin
15 reset = 0;
16 #5
17 reset = 1;
18 #25
19 reset = 1;
20 #70
21 reset = 1;
22 #50
23 reset = 0;
24 #5
25 reset = 1;
26 #5
27 $finish;
28 end
29 initial begin
30 #114
31 $finish;
32 end
33 endmodule
```

Ans9



Unrelated Questions :

1. I learned a lot from implementation of pipelined processor, so making this design wasn't that difficult. I made some syntax errors in procedural assignment, missed some begin end

blocks. The good idea here is to have modular code, design the processor such that it is scalable.

2. I have 100% designed the processor by myself. I was thinking about optimal design, discussed with some of my friends. Implementation was done by myself.

Honor Code Declaration by student:

()My answers to the above questions are my own work.

() I have not shared the codes/answers written by me with any other students. (I might have helped clear doubts of other students).

() I have not copied other's code/answers to improve my results. (I might have got some doubts cleared from other students).