Lab 4 - Implementation of a Simple Pipelined processor

Implement pipelined datapath (shown in the figure below) of a processor in verilog. This processor supports data transfer (mov) instructions only. The processor has Reset, CLK as inputs and no outputs. The processor has instruction fetch unit, register file and write back unit. The processor also contains two pipelined registers IF/ID and ID/WB. When reset is activated the PC is initialized to 0, the instruction memory and register file get loaded by predefined values. When the instruction unit starts fetching the first instruction the pipeline registers contain unknown values. When the second instruction is being fetched in IF unit, the IF/ID registers will hold the instruction code for first instruction. When the third instruction is being fetched by IF unit, the IF/ID register contains the instruction code of second instruction and ID/WB register contains information (destination address and data) related to first instruction.

The instruction and the 8-bit instruction format are shown below:

Mov DestinationReg, SourceReg

Rd	Rs
7:4	3:0

Assume the register file contains 16 registers (R0-R15) each register can hold 8-bit data. The register file does not require clock or any other control signals except for those shown in the figure. On reset assume that the instruction memory gets initialized with three instructions.

Mov Ry, Rx

Mov Rx, Rz

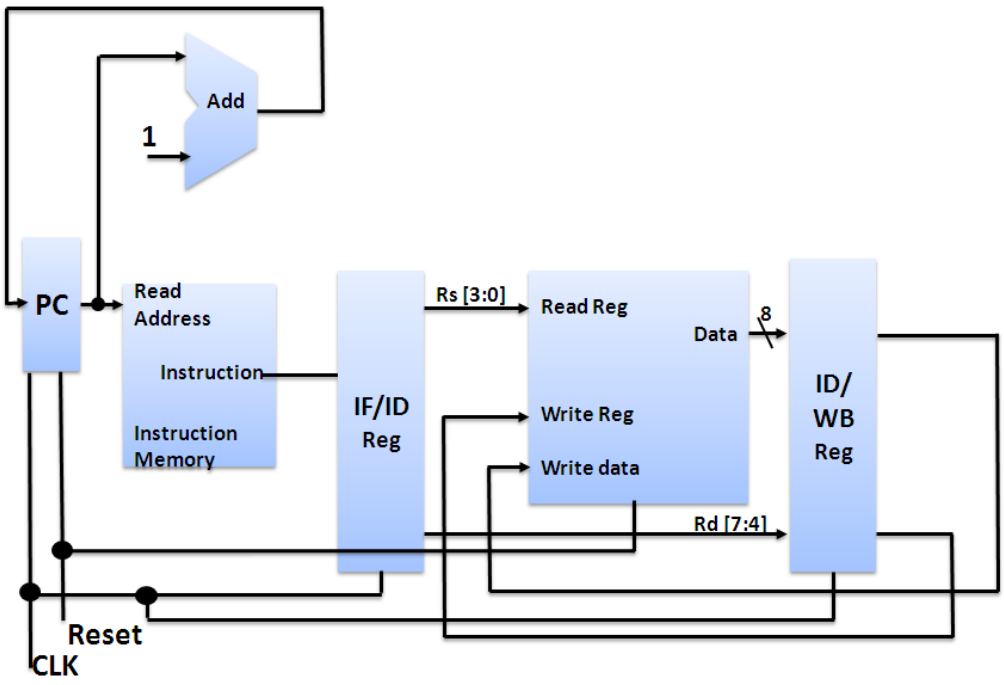
Mov Rz, Ry

Where xyz are related to last 3 digits of your ID No.

If ID number: 20XXXXXX123G, then x =1,y=2, z=3

If your ID number contains ‘0’ in the last two digits then consider ‘0’ as ‘10’. For example, If ID number: 20XXXXXX103H, then x =1,y=10, z=3

If two of the digits of your ID number are same, then take one of the digits as it is and replace the other digit with a value between 11-15. For example, If ID number: 20XXXXXX155H, then x=1, Y=5, z= 11,



P.T.O

Name: Siddhant Singh Vattsa **ID No:** 2020A3PS1832G

1. **Implement the Instruction Fetch block. Copy the <u>image</u> of Verilog code of the Instruction fetch block here**

Answer: The Instruction Fetch block functions use both the PC, and the Instruction Memory Unit. Both the Verilog codes have been pasted below.

```
1  `timescale 1ns / 1ps
2
3  module Counter_Pipe(
4      input clk,
5      input reset,
6      output [7:0] Instruction_Code
7      );
8
9      reg [1:0] PC;
10
11     Instruction_Memory_Pipe pp1 (PC, reset, Instruction_Code);
12     always @(posedge clk, negedge reset)
13     begin
14         if(reset == 1'b0) PC <= 2'b0;
15         else PC <= PC + 1;
16     end
17
18 endmodule
```

Code for PC (Program Counter)

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
21 module Instruction_Memory_Pipe(
22      input [1:0] Read_Address,
23      input reset,
24      output [7:0] Instruction_Code
25      );
26
27      reg [7:0] Mem [2:0];
28     assign Instruction_Code = Mem[Read_Address];
29
30     always @(reset)
31     begin
32      if(reset == 1'b0) begin
33       Mem[0] = 8'h38;
34       Mem[1] = 8'h82;
35       Mem[2] = 8'h23;
36      end
37      end
38
39 endmodule
```

Code for Instruction Memory Unit

2. **Implement the Register File and copy the image of Verilog code of Register file unit here.**

Answer:

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////
21
22
23 module Register_File_Pipe(
24      input [3:0] Read_Reg,
25      input [3:0] Write_Reg,
26      input [7:0] Write_Data,
27      output [7:0] Data,
28      input reset
29      );
30
31      reg [7:0] Memory [15:0];
32
33      assign Data = Memory[Read_Reg];
34      always @(*)Memory[Write_Reg] = Write_Data;
35
36      always @(*) begin
37      if(reset == 1'b0) begin
38           Memory[8] = 8'd6;
39           Memory[2] = 8'd9;
40           Memory[3] = 8'd4;
41           end
42      end
43
44
45 endmodule
```

3. **Implement complete processor in Verilog (using all the datapath blocks). Copy the image of Verilog code of the processor here.**

Answer:

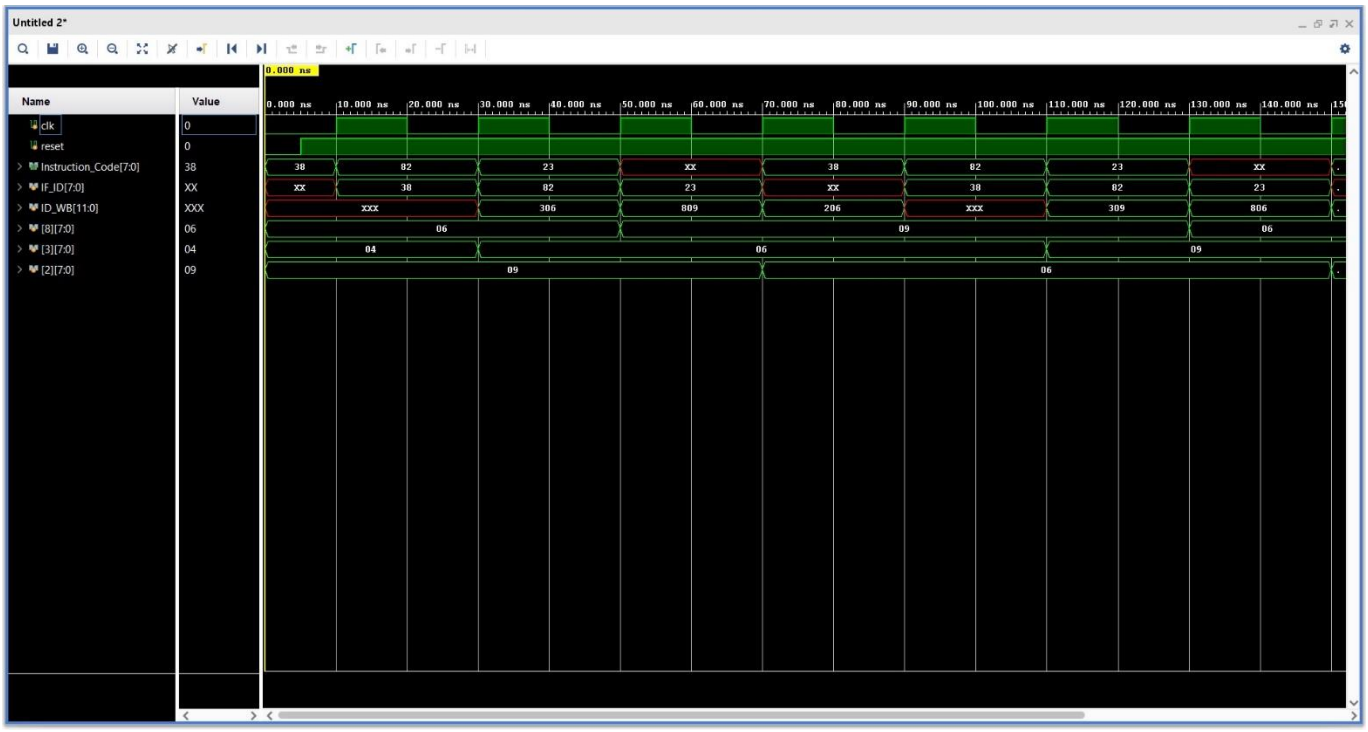
```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
21 module Processor_Pipe(
22      input clk,
23      input reset
24      );
25      reg [7:0] IF_ID;
26      wire [7:0] Write_Data;
27      wire [7:0] Instruction_Code;
28      reg [11:0] ID_WB;
29      wire [7:0] Data;
30
31      Counter_Pipe pipe0 (clk, reset, Instruction_Code);
32      Register_File_Pipe pipe1 ( IF_ID[3:0], ID_WB[11:8], Write_Data, Data, reset);
33      assign Write_Data = ID_WB[7:0];
34      always @(posedge clk) begin
35          IF_ID <= Instruction_Code;
36          ID_WB <= { IF_ID[7:4], Data };
37      end
38 endmodule
```

4. **Test the processor design by initializing the instruction memory with a set of instructions (mentioned earlier) and register file with a set of values. List below the data you have filled your registers with.**

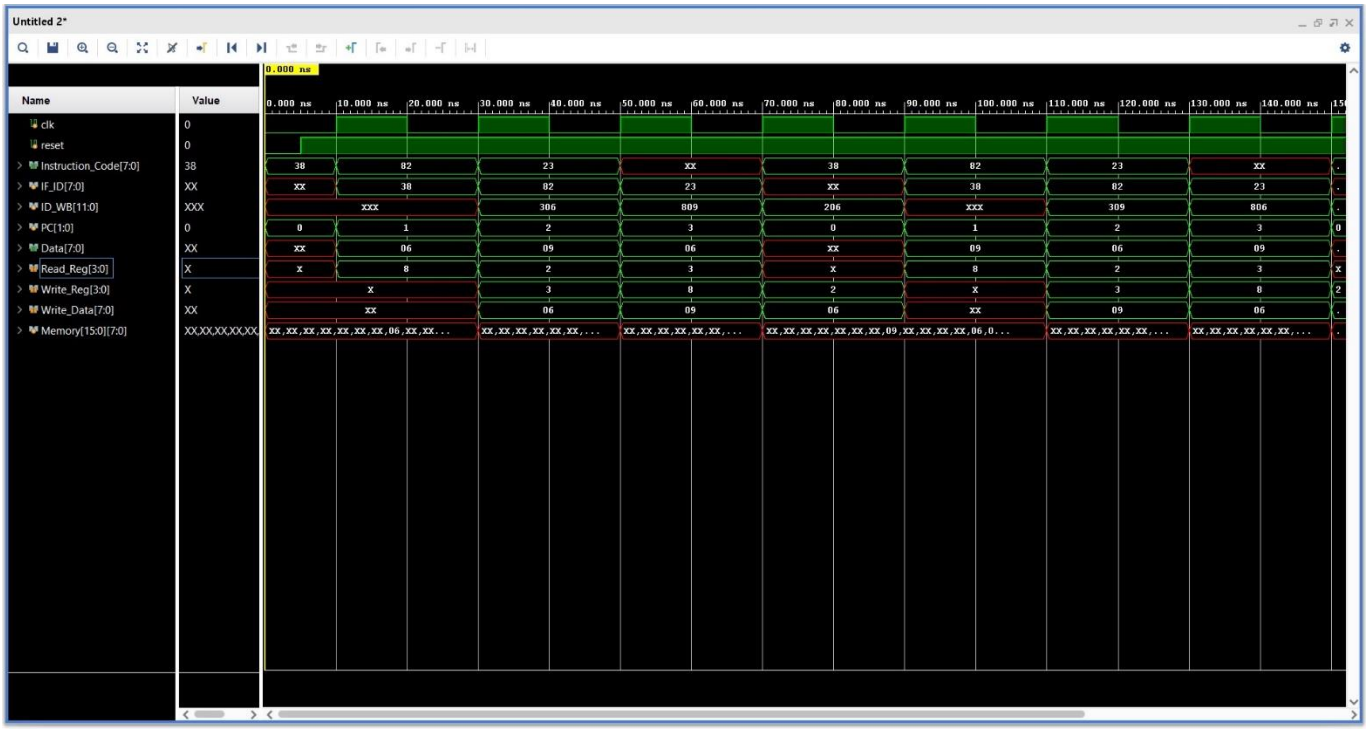
Answer: R0: X R1: X R2: 9 R3: 4 R4: X R5: X R6: X R7: X R8: 6 R9: X R10: X R11: X R12: X R13: X R14: X R15: X

5. **Verify if the register file is getting updated according to the set of instructions (mentioned earlier).**

Copy verified **Register file** waveform here (show only the Registers that get updated, CLK, RESET):



Waveforms of only Updating Registers in Register Memory



Waveforms of all important signals used for debugging

6. **Suppose the set of instructions to be executed are**
   Mov Rx, Ry
   Mov Rz, Rx
   Will there be any hazards in this processor?

Answer Yes or No: NO

Give detailed reasoning for your answer here: When the second instruction, Mov Rz, Rx is stored in the IF/ID pipelined register on a positive clock edge, data from the source register Rx is read immediately without any need for clock edge, according to the question (Register File Unit does not require any clock signal). During this time, data is being written into the Rx register immediately without any need for clock edge. Since the process is asynchronous, if there is a change in the data being written into the Rx register of the first instruction, it will be immediately reflected into the Rx register of the second instruction, and the value will be updated automatically, leading to no hazard.