Lab 4 - Implementation of a Simple Pipelined Processor

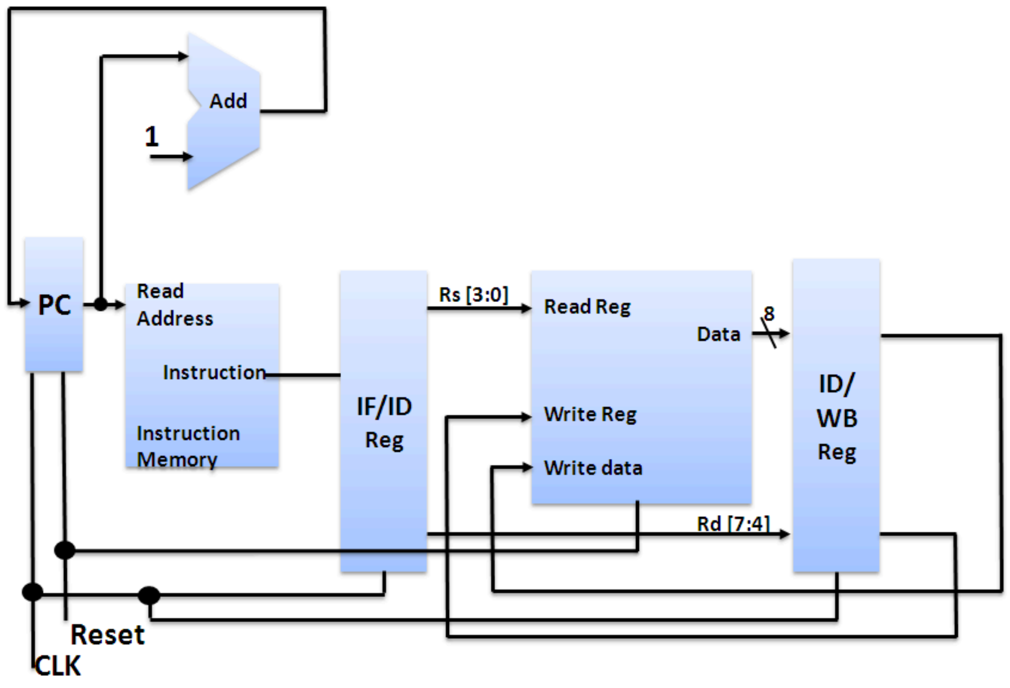
Implement pipelined datapath (shown in the figure below) of a processor in Verilog. This processor supports data transfer (mov) instructions only. The processor has Reset, CLK as inputs and no outputs. The processor has an instruction fetch unit, register file, and write back unit. The processor also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



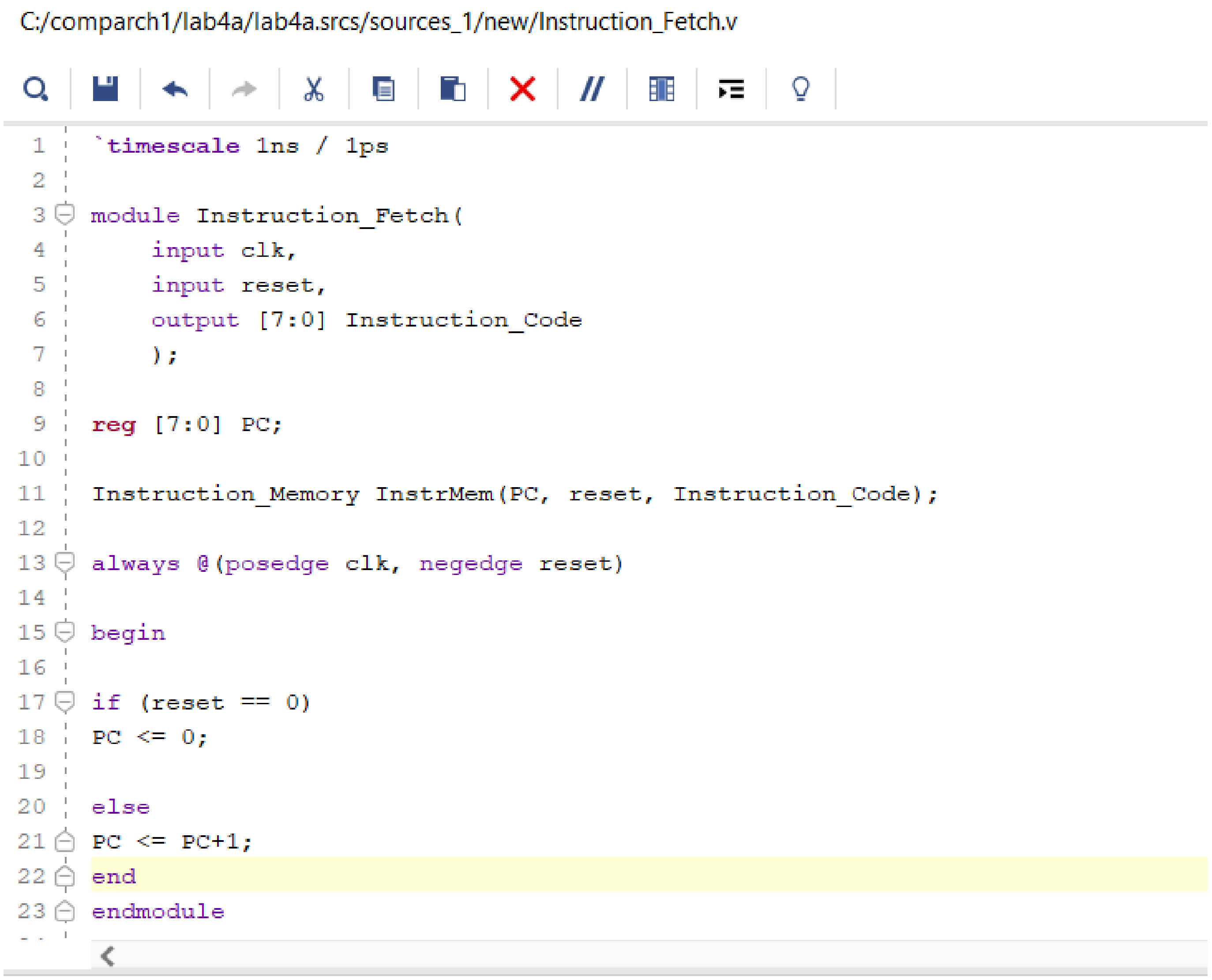
P.T.O

Name: Yashi Malik **ID No:** 2021A3PS3056G

x=10, y=5, z=6

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

Answer:

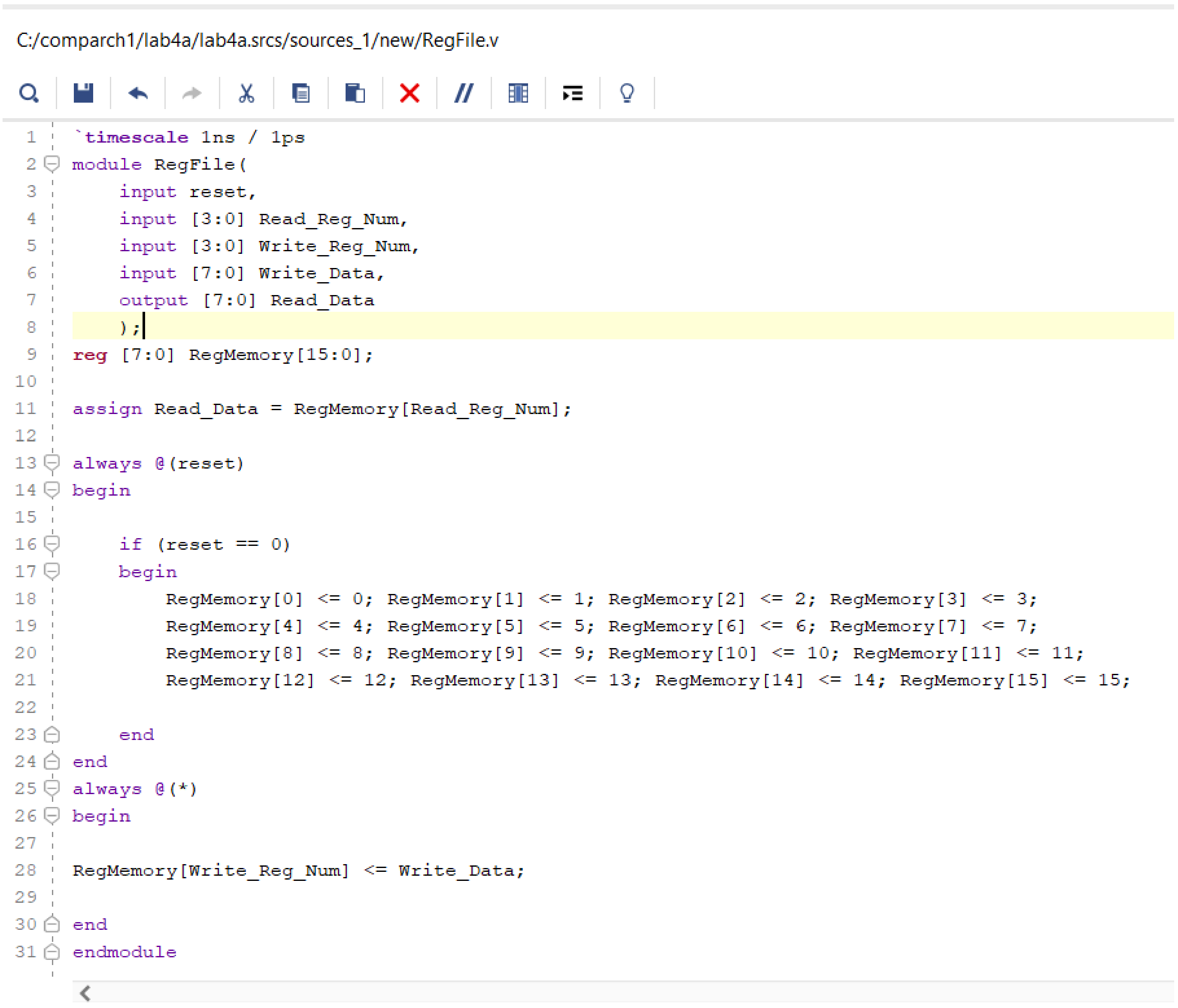


C:/comparch1/lab4a/lab4a.srcs/sources_1/new/Instruction_Fetch.v

```
`timescale 1ns / 1ps

module Instruction_Fetch(
    input clk,
    input reset,
    output [7:0] Instruction_Code
    );

reg [7:0] PC;

Instruction_Memory InstrMem(PC, reset, Instruction_Code);

always @(posedge clk, negedge reset)

begin

if (reset == 0)
PC <= 0;

else
PC <= PC+1;
end
endmodule
```

2. Implement the Register File and copy the <u>image</u> of the Verilog code of the Register file unit here.

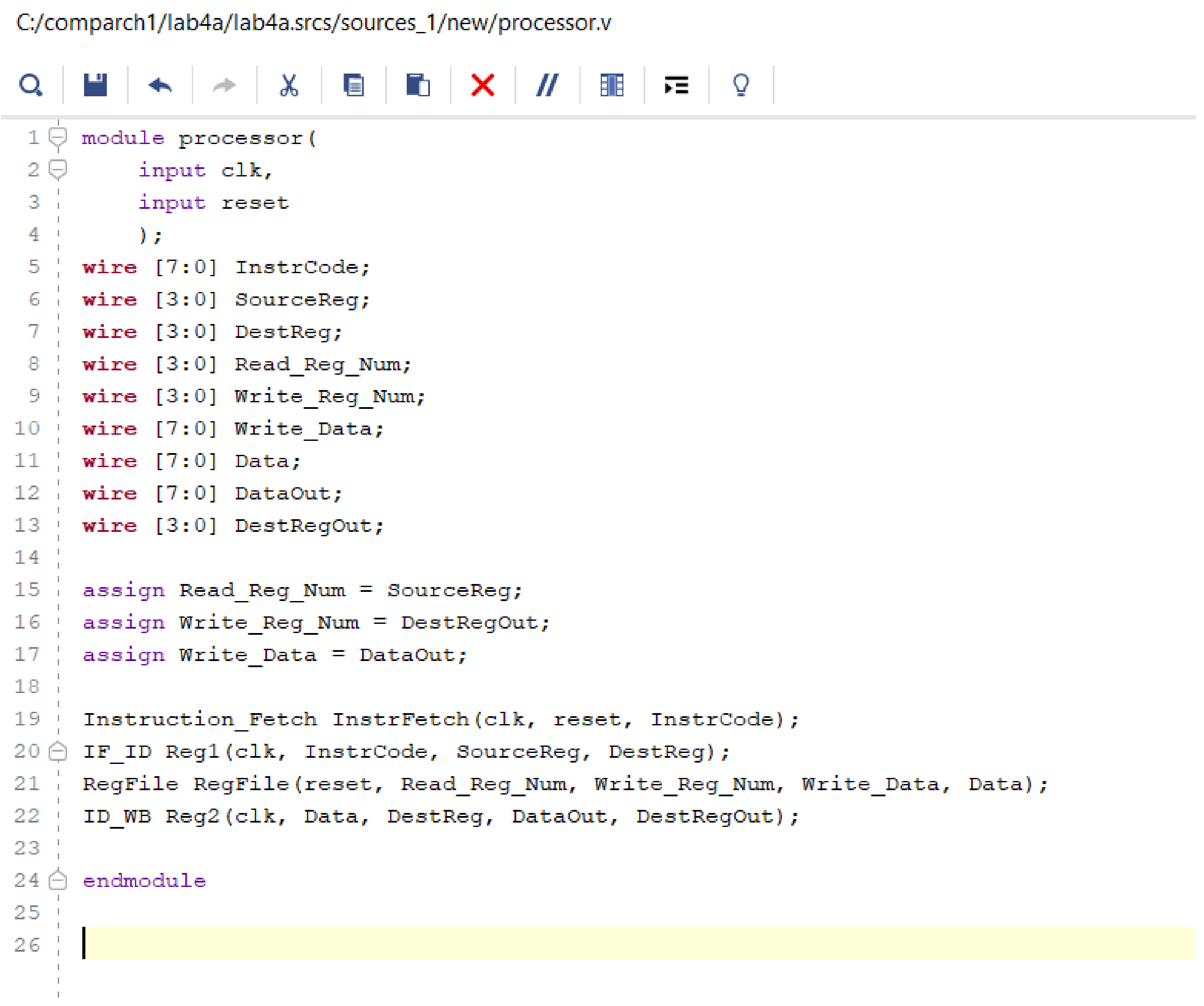
Answer:


C:/comparch1/lab4a/lab4a.srcs/sources_1/new/RegFile.v

```
`timescale 1ns / 1ps
module RegFile(
    input reset,
    input [3:0] Read_Reg_Num,
    input [3:0] Write_Reg_Num,
    input [7:0] Write_Data,
    output [7:0] Read_Data
    );
reg [7:0] RegMemory[15:0];

assign Read_Data = RegMemory[Read_Reg_Num];

always @(reset)
begin

    if (reset == 0)
    begin
        RegMemory[0] <= 0; RegMemory[1] <= 1; RegMemory[2] <= 2; RegMemory[3] <= 3;
        RegMemory[4] <= 4; RegMemory[5] <= 5; RegMemory[6] <= 6; RegMemory[7] <= 7;
        RegMemory[8] <= 8; RegMemory[9] <= 9; RegMemory[10] <= 10; RegMemory[11] <= 11;
        RegMemory[12] <= 12; RegMemory[13] <= 13; RegMemory[14] <= 14; RegMemory[15] <= 15;

    end
end
always @(*)
begin

RegMemory[Write_Reg_Num] <= Write_Data;

end
endmodule
```

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

Answer:


C:/comparch1/lab4a/lab4a.srcs/sources_1/new/processor.v

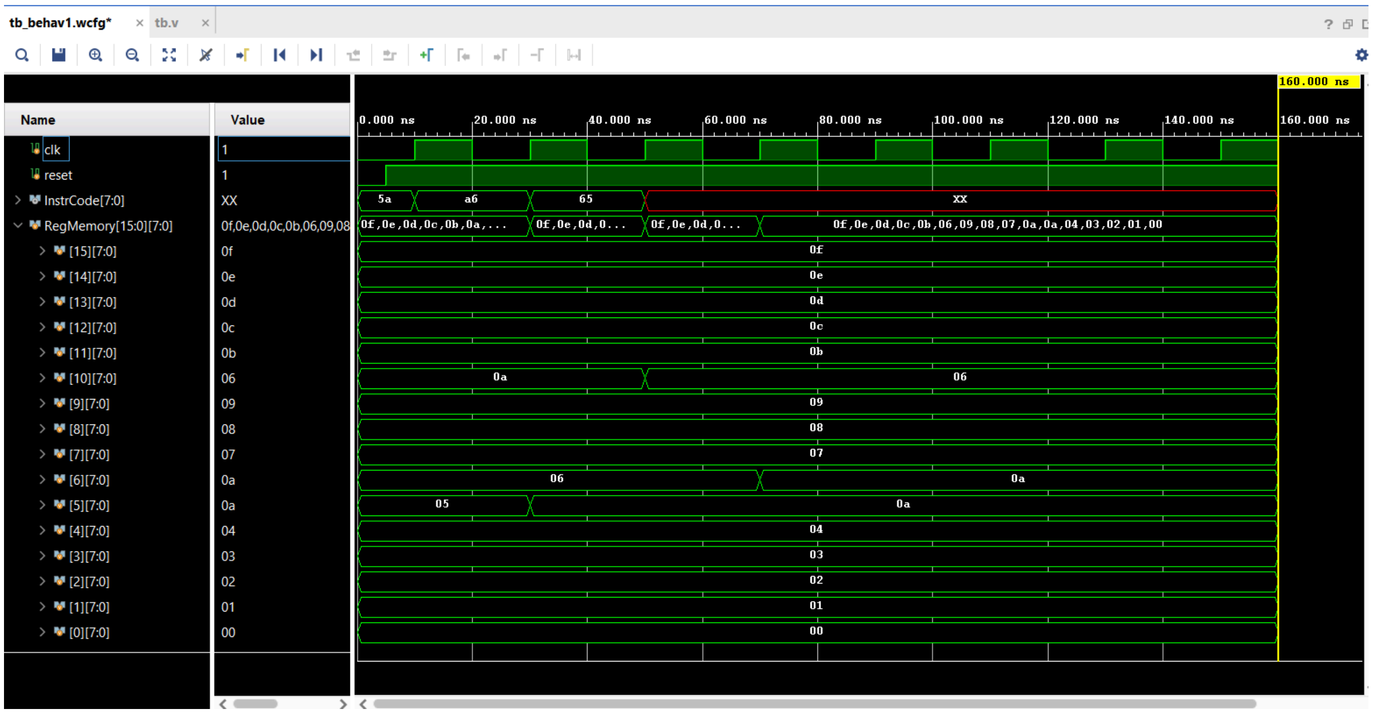
```
module processor(
    input clk,
    input reset
    );
wire [7:0] InstrCode;
wire [3:0] SourceReg;
wire [3:0] DestReg;
wire [3:0] Read_Reg_Num;
wire [3:0] Write_Reg_Num;
wire [7:0] Write_Data;
wire [7:0] Data;
wire [7:0] DataOut;
wire [3:0] DestRegOut;

assign Read_Reg_Num = SourceReg;
assign Write_Reg_Num = DestRegOut;
assign Write_Data = DataOut;

Instruction_Fetch InstrFetch(clk, reset, InstrCode);
IF_ID Reg1(clk, InstrCode, SourceReg, DestReg);
RegFile RegFile(reset, Read_Reg_Num, Write_Reg_Num, Write_Data, Data);
ID_WB Reg2(clk, Data, DestReg, DataOut, DestRegOut);

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: 0 R1:1 R2: 2 R3: 3 R4: 4 R5: 5 R6: 6 R7: 7 R8: 8 R9: 9 R10: 10 R11: 11 R12: 12 R13: 13 R14: 14 R15: 15

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

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

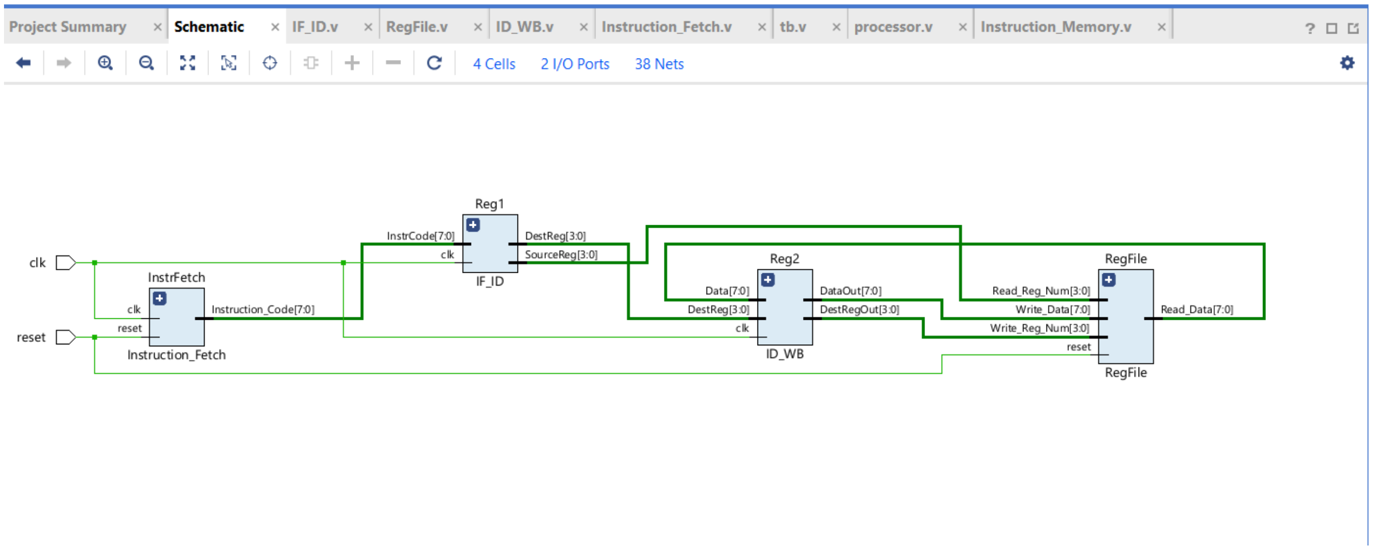


6. **Suppose the set of instructions to be executed are**
   Mov Rx, Ry
   Mov Rz, Rx
   Will there be any hazards in this processor?

Answer Yes or No: Yes it will lead to hazard

Give detailed reasoning for your answer here: When the value of Rx is updated in the first instruction in that same clock edge the value of Rx should be made available at the write-back stage of the second instruction but this new value won't appear, so it leads to Read-after-Write hazard (RAW).

7. **Synthesise the processor designed and copy the RTL generated below**


Project Summary
Schematic
IF_ID.v
RegFile.v
ID_WB.v
Instruction_Fetch.v
tb.v
processor.v
Instruction_Memory.v
4 Cells
2 I/O Ports
38 Nets
clk
reset
InstrFetch
clk
reset
Instruction_Code[7:0]
Instruction_Fetch
Reg1
InstrCode[7:0]
clk
DestReg[3:0]
SourceReg[3:0]
IF_ID
Reg2
Data[7:0]
DestReg[3:0]
clk
DataOut[7:0]
DestRegOut[3:0]
ID_WB
RegFile
Read_Reg_Num[3:0]
Write_Data[7:0]
Write_Reg_Num[3:0]
reset
Read_Data[7:0]
RegFile