Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2020-21
CS F342: Computer Architecture Assignment (30 Marks)

This assignment consists of two parts, Part A and Part B:

1. Implement a Single Cycle processor in Verilog. This processor supports data transfer (mov), constant addition (addi) and unconditional jump (J) instructions only. The processor has Reset and CLK as inputs and no outputs. The processor has instruction fetch unit, registerfile (with eight 8-bit registers), Execution and Writeback unit. Read operation on the registerfile is independent of CLK whereas the write happens on the positive edge of the CLK. When reset is activated the PC register is initialized to 0, the instruction memory and register file get loaded by **predefined values.** (Assume 8-bit PC. Also Assume Address and Data size as 8-bits).

2. (a) Implement 4-stage pipelined processor in Verilog. This processor supports data transfer (mov), constant addition (addi) and Unconditional Jump (J) instructions only. The processor should implement forwarding to resolve data hazards. The processor has Reset, CLK as inputs and no outputs. The processor has instruction fetch unit, register file (with eight 8-bit registers), Execution and Writeback unit. Read and write operations on Register file can happen simultaneously and should be independent of CLK. The processor also contains three pipelined registers IF/ID, ID/EX and EX/WB. When reset is activated the PC, IF/ID, ID/EX, EX/WB registers are initialized to 0, the instruction memory and registerfile 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, ID/EX register contains information related to first instruction and so on. (Assume 8-bit PC. Also Assume Address and Data size as 8-bits).

The instructions and its **8-bit instruction format** for single cycle and pipeplined processor are shown below:

mov DestinationReg, SourceReg (Moves data in register specified by register number in Rsrc field to a register specified by register number in RDst field. Opcode for mov is 00)

Opcode		
00	RDst	RSrc
7:6	5:3	2:0

Example usage: mov R2, R0 (R2←R0)

addi DestinationReg, immediateData (adds data in register specified by register number in RDst field to sign extended data in immediate data field (2:0). Result is stored in register specified by register number in RDst field. Opcode for addi is 01)

Opcode		
01	RDst	Immediate Data
7:6	5:3	2:0

Example usage: addi R2, 3 3 gets sign extended to 8-bits 00000011 then is added to R2 (R2←R2+ 00000011)

j L1 (Jumps to an address generated by adding PC+1 to the Signextended data specified in instruction field (5:0). Opcode for j is 11)

Opcode	
11	Partial Jump Address
7:6	5:0

Example usage: j L1 (Jump address is calculated using PC relative addressing)

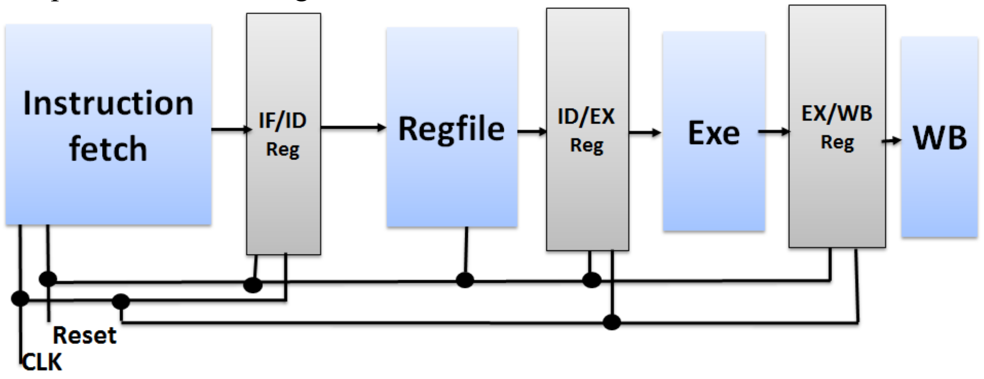
Assume the register file contains 8 registers (R0-R7) each register can hold 8-bit data. On reset register file should get initialized such that R0 = 0, R1 = 1, R2 = 2, R3 = 3 …etc. On reset assume that the instruction memory gets initialized with four instructions.

```
        mov Rx, Ry
        addi Rx, 3
        addi Ry, 2
        j L1
        mov Rz, Ry
L1:     addi Rz, -3
```

Where x, y, z are related to last 3 digits of your ID No.

If ID number: 20XXXXXXABCH, then x =A mod 8 (A%8),
y= (B+2) mod 8 ((B+2)%8),
z= (C+3) mod 8 ((C+3)%8),

Note for Pipelined Processor: A partial block level representation of 4-stage pipelined processor is shown below. **Please note that for registerfile implementation, both read and write are independent of CLK**. Write operation depends on control signal.



Submission Procedure

As part of the assignment three files should be submitted in zipped folder.

1. PDF version of this Document with all the Questions below answered with file name as **IDNO_NAME.pdf**.

2. Design Verilog Files for all the Sub-modules (instruction fetch, Register file, forwarding unit, etc).

3. Design Verilog file for both single cycle and pipelined processor.

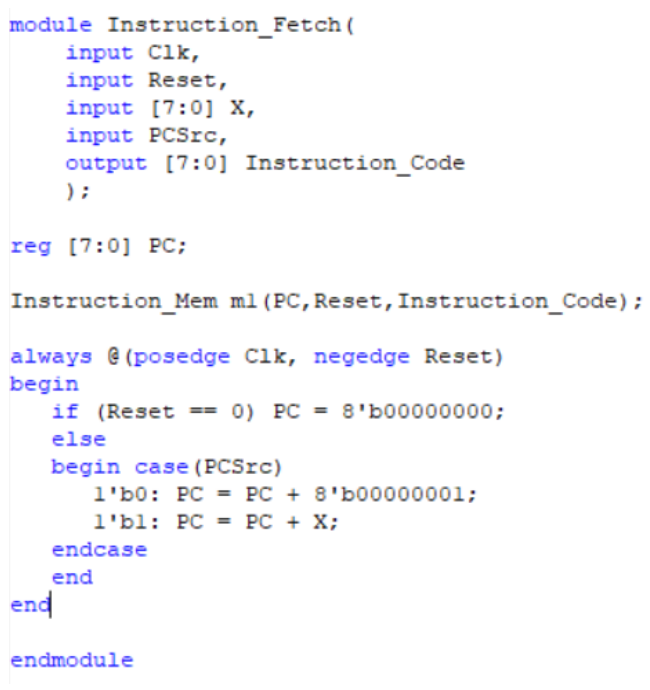
The name of the zipped folder should be in the format IDNO_NAME.zip

The due date for submission is 15-April-2021, 5:00 PM.

Name: Tejus Vidyadhar Kusur **ID No:** 2018A3PS0531H

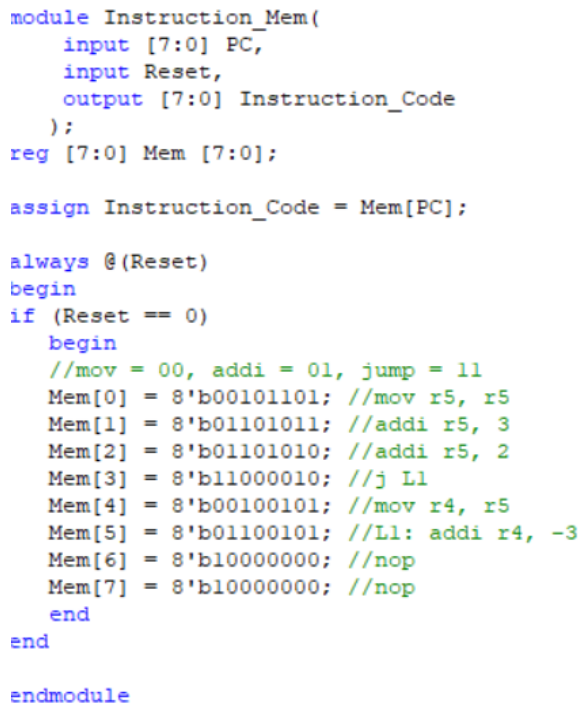
Common Questions applicable to both single cycle and pipelined processor

1. **Implement the Instruction Fetch block. Copy the image of Verilog code of the Instruction fetch block here**



```
module Instruction_Fetch(
    input Clk,
    input Reset,
    input [7:0] X,
    input PCSrc,
    output [7:0] Instruction_Code
    );

reg [7:0] PC;

Instruction_Mem m1(PC,Reset,Instruction_Code);

always @(posedge Clk, negedge Reset)
begin
   if (Reset == 0) PC = 8'b00000000;
   else
   begin case(PCSrc)
      1'b0: PC = PC + 8'b00000001;
      1'b1: PC = PC + X;
   endcase
   end
end

endmodule
```

Answer:



```
module Instruction_Mem(
    input [7:0] PC,
    input Reset,
    output [7:0] Instruction_Code
   );
reg [7:0] Mem [7:0];

assign Instruction_Code = Mem[PC];

always @(Reset)
begin
if (Reset == 0)
   begin
   //mov = 00, addi = 01, jump = 11
   Mem[0] = 8'b00101101; //mov r5, r5
   Mem[1] = 8'b01101011; //addi r5, 3
   Mem[2] = 8'b01101010; //addi r5, 2
   Mem[3] = 8'b11000010; //j L1
   Mem[4] = 8'b00100101; //mov r4, r5
   Mem[5] = 8'b01100101; //L1: addi r4, -3
   Mem[6] = 8'b10000000; //nop
   Mem[7] = 8'b10000000; //nop
   end
end

endmodule
```

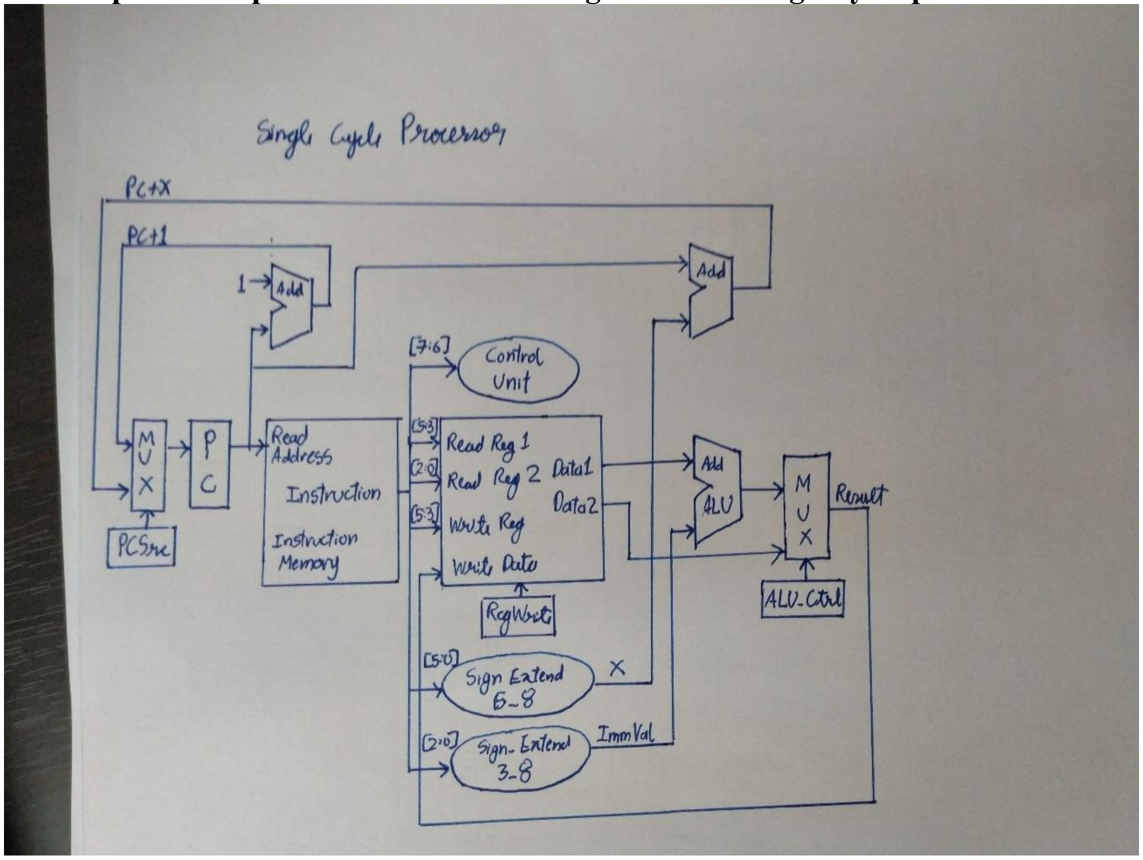
2. **List the control signals used and also the values of control signals for different instructions.**

Answer:

Instructions	**Control Signals**					
	RegWrite	ALU_Ctrl	PCSrc			
mov	1	1	0			
addi	1	0	0			
j	0	0	1			

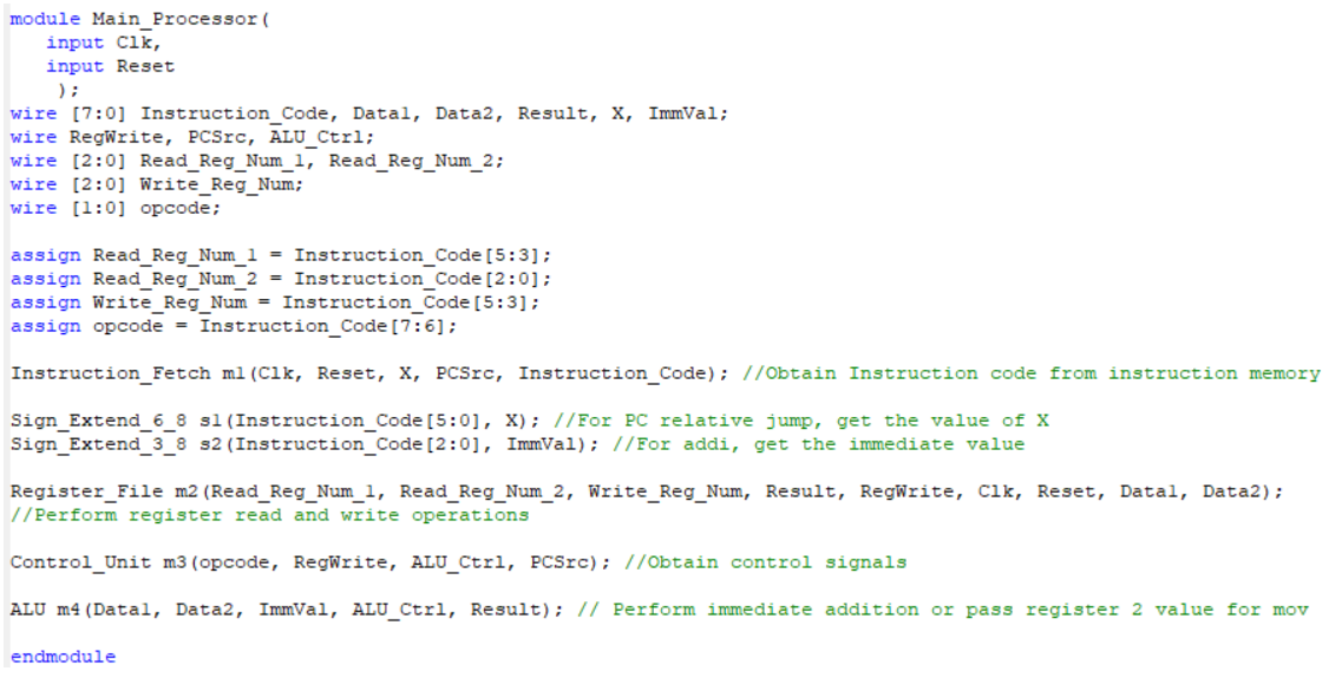
Questions applicable to single cycle processor

1. **Draw the complete Datapath and show control signals of the single cycle processor.**

Answer: 

2. **Implement complete single cycle processor in Verilog (using all the Datapath blocks). Copy the <u>image</u> of Verilog code of the processor here. (Use comments to describe your Verilog implementation)**

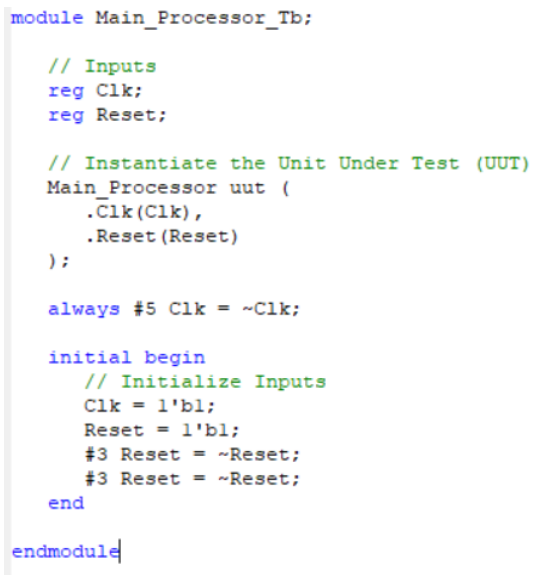
Answer:



```
module Main_Processor(
   input Clk,
   input Reset
    );
wire [7:0] Instruction_Code, Data1, Data2, Result, X, ImmVal;
wire RegWrite, PCSrc, ALU_Ctrl;
wire [2:0] Read_Reg_Num_1, Read_Reg_Num_2;
wire [2:0] Write_Reg_Num;
wire [1:0] opcode;

assign Read_Reg_Num_1 = Instruction_Code[5:3];
assign Read_Reg_Num_2 = Instruction_Code[2:0];
assign Write_Reg_Num = Instruction_Code[5:3];
assign opcode = Instruction_Code[7:6];

Instruction_Fetch m1(Clk, Reset, X, PCSrc, Instruction_Code); //Obtain Instruction code from instruction memory

Sign_Extend_6_8 s1(Instruction_Code[5:0], X); //For PC relative jump, get the value of X
Sign_Extend_3_8 s2(Instruction_Code[2:0], ImmVal); //For addi, get the immediate value

Register_File m2(Read_Reg_Num_1, Read_Reg_Num_2, Write_Reg_Num, Result, RegWrite, Clk, Reset, Data1, Data2);
//Perform register read and write operations

Control_Unit m3(opcode, RegWrite, ALU_Ctrl, PCSrc); //Obtain control signals

ALU m4(Data1, Data2, ImmVal, ALU_Ctrl, Result); // Perform immediate addition or pass register 2 value for mov

endmodule
```

3. **Test the single cycle processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

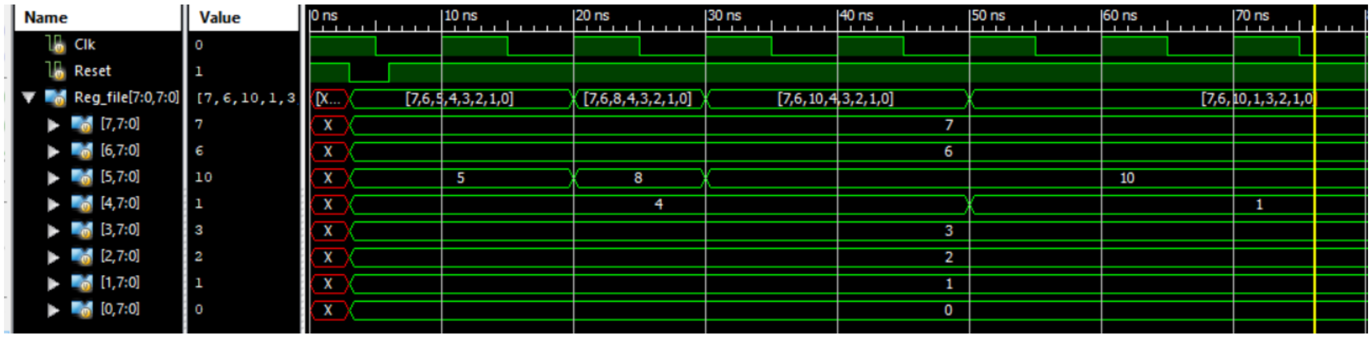
Answer:



```
module Main_Processor_Tb;

   // Inputs
   reg Clk;
   reg Reset;

   // Instantiate the Unit Under Test (UUT)
   Main_Processor uut (
      .Clk(Clk),
      .Reset(Reset)
   );

   always #5 Clk = ~Clk;

   initial begin
      // Initialize Inputs
      Clk = 1'b1;
      Reset = 1'b1;
      #3 Reset = ~Reset;
      #3 Reset = ~Reset;
   end

endmodule
```

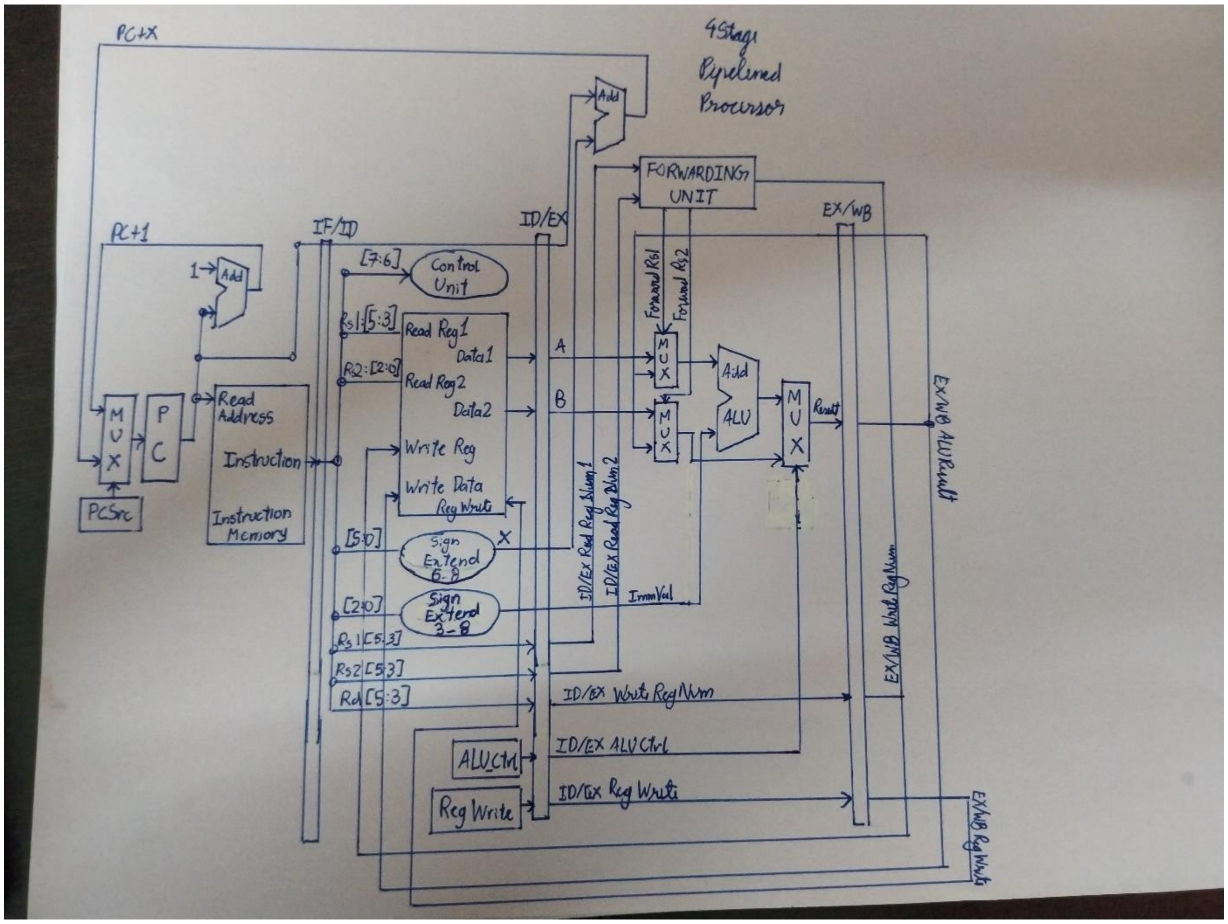
4. **Verify if the register file is getting updated according to the set of instructions (mentioned earlier).**

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



<u>Questions applicable to pipelined processor</u>

1. **Draw the complete Datapath and show control signals of the 4-stage pipelined processor. A sample Datapath for 5-stage pipelined MIPS processor has been discussed in class. A ppt named Assignmenthelp.ppt contains this 5-stage processor and is uploaded in CMS. You can modify this according to your specification.**

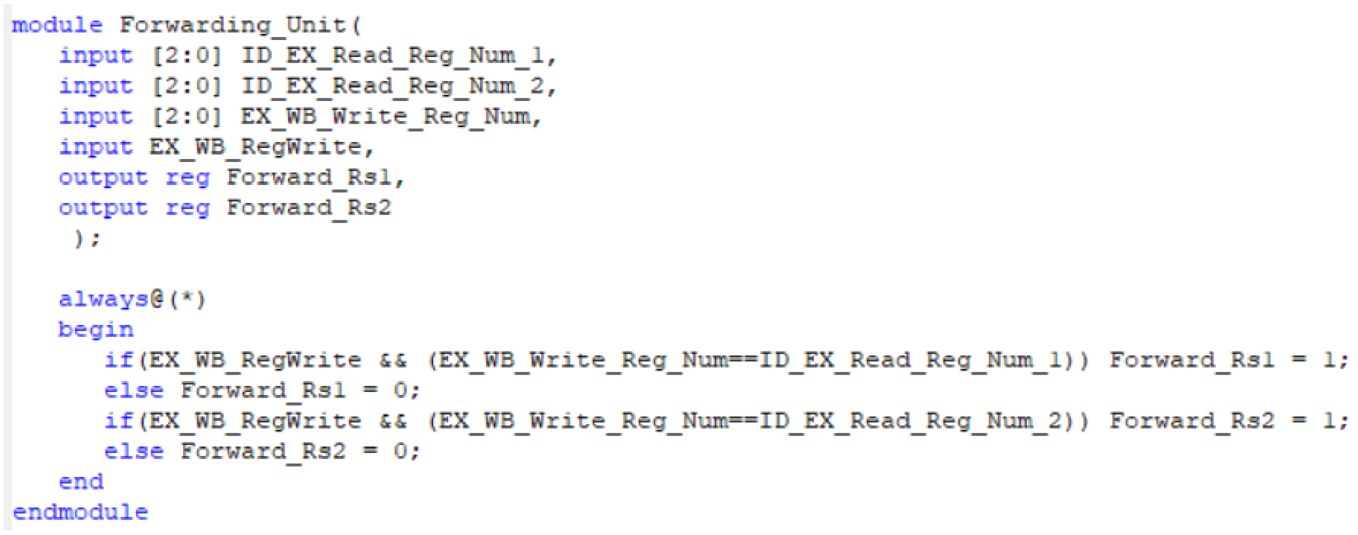
Answer: 

2. Determine the condition that can be used to detect data hazard?

Answer: If (EX_WB_RegWrite && EX_WB_Write_Reg_Num==ID_EX_Read_Reg_Num_1) is true, i.e. the value of resistor being read for the next instruction is being updated in the current instruction, data hazard is detected and this is mitigated through forwarding.

3. Implement the forwarding unit and copy the <u>image</u> of Verilog code of forwarding unit here.

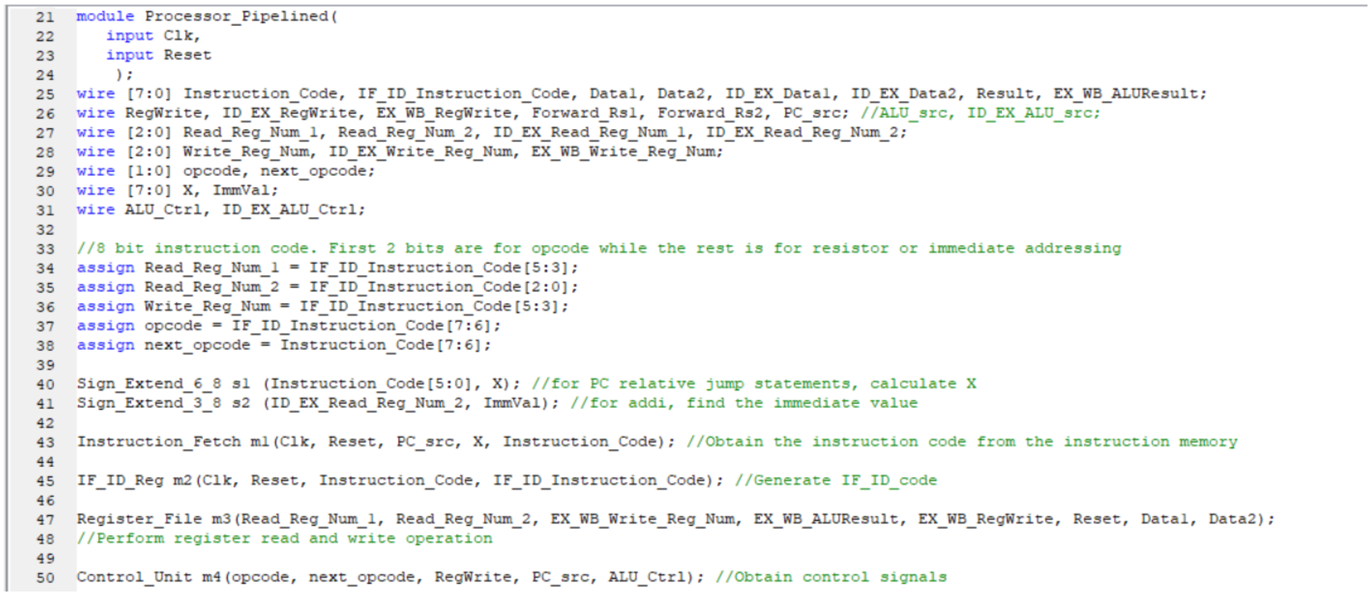
Answer:



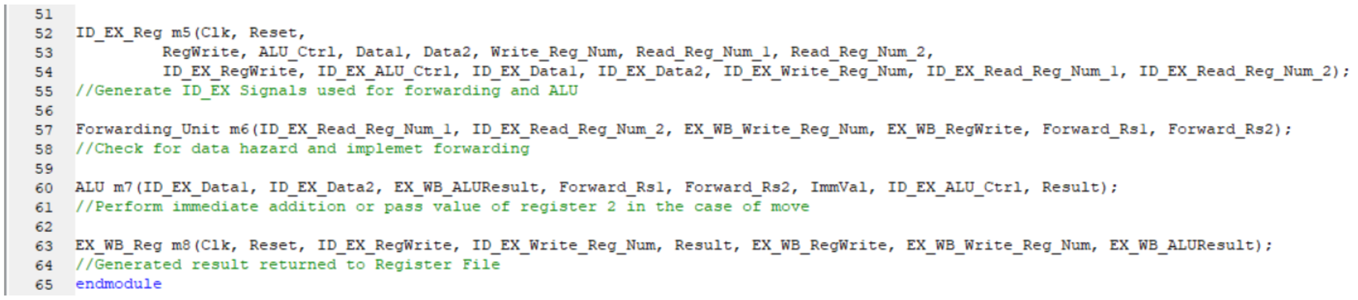
```verilog
module Forwarding_Unit(
   input [2:0] ID_EX_Read_Reg_Num_1,
   input [2:0] ID_EX_Read_Reg_Num_2,
   input [2:0] EX_WB_Write_Reg_Num,
   input EX_WB_RegWrite,
   output reg Forward_Rs1,
   output reg Forward_Rs2
    );

   always@(*)
   begin
      if(EX_WB_RegWrite && (EX_WB_Write_Reg_Num==ID_EX_Read_Reg_Num_1)) Forward_Rs1 = 1;
      else Forward_Rs1 = 0;
      if(EX_WB_RegWrite && (EX_WB_Write_Reg_Num==ID_EX_Read_Reg_Num_2)) Forward_Rs2 = 1;
      else Forward_Rs2 = 0;
   end
endmodule
```

4. Implement complete pipelined processor in Verilog (using all the Datapath blocks). Copy the <u>image</u> of Verilog code of the processor here. (Use comments to describe your Verilog implementation)

Answer:

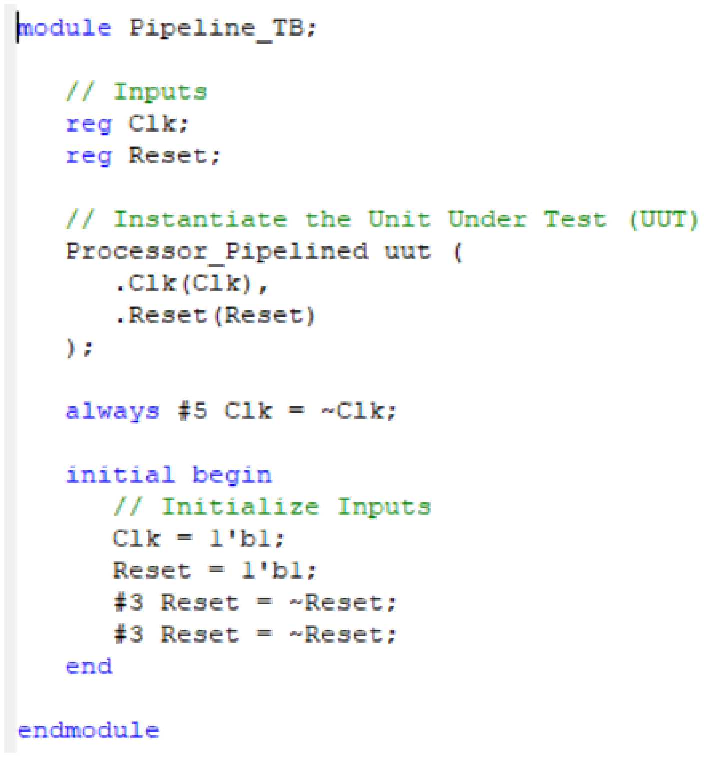


```verilog
module Processor_Pipelined(
   input Clk,
   input Reset
    );
wire [7:0] Instruction_Code, IF_ID_Instruction_Code, Data1, Data2, ID_EX_Data1, ID_EX_Data2, Result, EX_WB_ALUResult;
wire RegWrite, ID_EX_RegWrite, EX_WB_RegWrite, Forward_Rs1, Forward_Rs2, PC_src; //ALU_src, ID_EX_ALU_src;
wire [2:0] Read_Reg_Num_1, Read_Reg_Num_2, ID_EX_Read_Reg_Num_1, ID_EX_Read_Reg_Num_2;
wire [2:0] Write_Reg_Num, ID_EX_Write_Reg_Num, EX_WB_Write_Reg_Num;
wire [1:0] opcode, next_opcode;
wire [7:0] X, ImmVal;
wire ALU_Ctrl, ID_EX_ALU_Ctrl;

//8 bit instruction code. First 2 bits are for opcode while the rest is for resistor or immediate addressing
assign Read_Reg_Num_1 = IF_ID_Instruction_Code[5:3];
assign Read_Reg_Num_2 = IF_ID_Instruction_Code[2:0];
assign Write_Reg_Num = IF_ID_Instruction_Code[5:3];
assign opcode = IF_ID_Instruction_Code[7:6];
assign next_opcode = Instruction_Code[7:6];

Sign_Extend_6_8 s1 (Instruction_Code[5:0], X); //for PC relative jump statements, calculate X
Sign_Extend_3_8 s2 (ID_EX_Read_Reg_Num_2, ImmVal); //for addi, find the immediate value

Instruction_Fetch m1(Clk, Reset, PC_src, X, Instruction_Code); //Obtain the instruction code from the instruction memory

IF_ID_Reg m2(Clk, Reset, Instruction_Code, IF_ID_Instruction_Code); //Generate IF_ID_code

Register_File m3(Read_Reg_Num_1, Read_Reg_Num_2, EX_WB_Write_Reg_Num, EX_WB_ALUResult, EX_WB_RegWrite, Reset, Data1, Data2);
//Perform register read and write operation

Control_Unit m4(opcode, next_opcode, RegWrite, PC_src, ALU_Ctrl); //Obtain control signals
```



```verilog

ID_EX_Reg m5(Clk, Reset,
         RegWrite, ALU_Ctrl, Data1, Data2, Write_Reg_Num, Read_Reg_Num_1, Read_Reg_Num_2,
         ID_EX_RegWrite, ID_EX_ALU_Ctrl, ID_EX_Data1, ID_EX_Data2, ID_EX_Write_Reg_Num, ID_EX_Read_Reg_Num_1, ID_EX_Read_Reg_Num_2);
//Generate ID_EX Signals used for forwarding and ALU

Forwarding_Unit m6(ID_EX_Read_Reg_Num_1, ID_EX_Read_Reg_Num_2, EX_WB_Write_Reg_Num, EX_WB_RegWrite, Forward_Rs1, Forward_Rs2);
//Check for data hazard and implemet forwarding

ALU m7(ID_EX_Data1, ID_EX_Data2, EX_WB_ALUResult, Forward_Rs1, Forward_Rs2, ImmVal, ID_EX_ALU_Ctrl, Result);
//Perform immediate addition or pass value of register 2 in the case of move

EX_WB_Reg m8(Clk, Reset, ID_EX_RegWrite, ID_EX_Write_Reg_Num, Result, EX_WB_RegWrite, EX_WB_Write_Reg_Num, EX_WB_ALUResult);
//Generated result returned to Register File
endmodule
```

5. **Test the pipelined processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

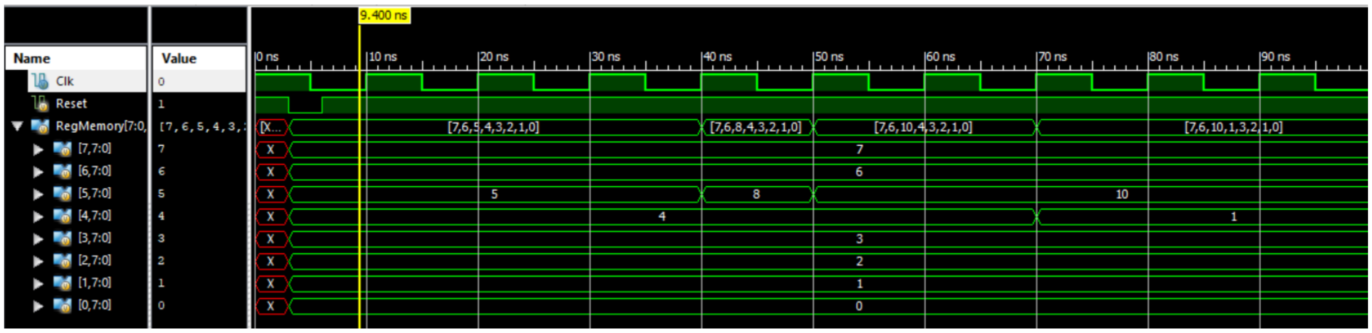


```verilog
module Pipeline_TB;

    // Inputs
    reg Clk;
    reg Reset;

    // Instantiate the Unit Under Test (UUT)
    Processor_Pipelined uut (
        .Clk(Clk),
        .Reset(Reset)
    );

    always #5 Clk = ~Clk;

    initial begin
        // Initialize Inputs
        Clk = 1'b1;
        Reset = 1'b1;
        #3 Reset = ~Reset;
        #3 Reset = ~Reset;
    end

endmodule
```

Answer:

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

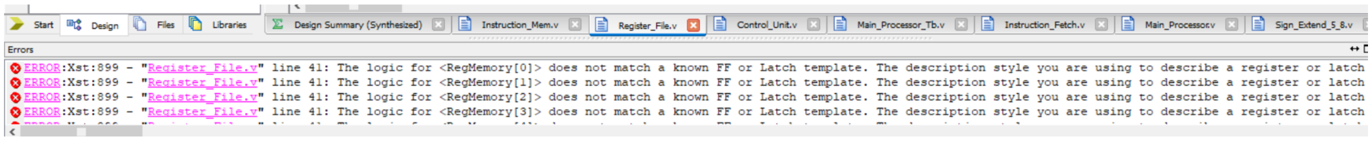
Copy verified **<u>Register file</u>** waveform here (show only the Registers that get updated, CLK, and RESET):



Unrelated Questions

What were the problems you faced during the implementation of the processor?

Answer: Register memory had refused to be implemented with the following error:



```
ERROR:Xst:899 - "Register_File.v" line 41: The logic for <RegMemory[0]> does not match a known FF or Latch template. The description style you are using to describe a register or latch
ERROR:Xst:899 - "Register_File.v" line 41: The logic for <RegMemory[1]> does not match a known FF or Latch template. The description style you are using to describe a register or latch
ERROR:Xst:899 - "Register_File.v" line 41: The logic for <RegMemory[2]> does not match a known FF or Latch template. The description style you are using to describe a register or latch
ERROR:Xst:899 - "Register_File.v" line 41: The logic for <RegMemory[3]> does not match a known FF or Latch template. The description style you are using to describe a register or latch
```

Upon asking for help from my friends for assistance to implement register file they said that the version of Xilinx could potentially lead to this error and upon removing for loop, the model could be simulated on testbench.

Did you implement the processor on your own? If you took help from someone whose help did you take? Which part of the design did you take help for?

Answer: Yes, I implemented the processor on my own with the small exception of register file errors for which I had to ask for help to identify the mistakes and rectify them.

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).

Name: Tejus Vidyadhar Kusur **Date:** 15-03-2021
ID No.: 2018A3PS0531H