Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2021-22
CS F342: Computer Architecture Assignment (30 Marks)

A. Implement 4-stage pipelined processor in Verilog. This processor supports load immediate (li), addition (add) 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:

li DestinationReg, ImmediateData (Signextends data specified in instruction field (2:0) to 8-bits and stores it in register specified by register number in RDst field. Opcode for li is 00)

Example usage: li R3, 4 (4 = 100 signextension will result in 1111100. This data moves in to R3.

add DestinationReg, SourceReg (adds data in register specified by register number in Rsrc field to data in register specified by register number in RDst field. Result is stored in register specified by register number in RDst field. Opcode for add is 01)

Example usage: add R2, R0 (R2←R2+ R0)

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)

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

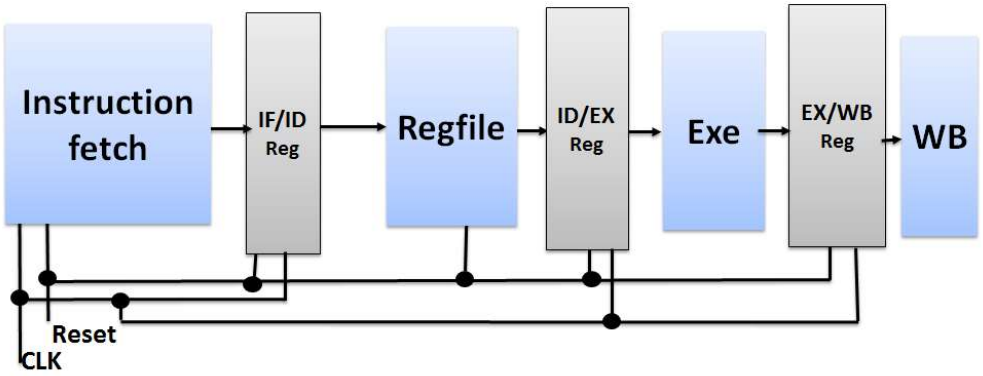
Assume the register file contains 8 registers (R0-R7) each register can hold 8-bit data. On reset register file should get initialized such that R0 = 0, R1 = 1, R2 = 2, R3 = 3 …etc. On reset assume that the instruction memory gets initialized with four instructions.

```
        li Rx, 3
        add Ry, Rx
        add Rz, Ry
        j L1
        li Rz, 4
L1:     add Rx, Rz
```

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 24-April-2022, 5:00 PM.

Name:
VISHWAS VASUKI GAUTAM **ID No: 2019A3PS0443H**

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

Answer: **Program Counter:**

```
module pc(
    input clk,
    input rst,
    input [7:0] pcIn,
    output reg [7:0] pcOut
    );

  always @(posedge clk, negedge rst)
  begin
    if(rst == 0)
        pcOut <= 0;
    else
        pcOut <= pcIn;
  end

endmodule
```

Instruction Memory:

```
module instrMem(
  input clk,
  input rst,  // Reset
  input [7:0] pc,
  output [7:0] instr
);

reg [7:0] instrMem [32:0];

// Defining the instructions in the instrMem in byte addressable manner
always @(*)
  begin
    if(rst == 0)
    begin
//    instrMem[0] = 8'b01001010;
////    instrMem[1] = 8'b11000011;
//    instrMem[2] = 8'b01100101;
//    instrMem[3] = 8'b00010011;
//    instrMem[4] = 8'b00000010;
//    instrMem[5] = 8'b01000010;

    instrMem[0] = 8'b00100011;
    instrMem[1] = 8'b01110100;
    instrMem[2] = 8'b01110110;
    instrMem[3] = 8'b11000010; //jmp
    instrMem[4] = 8'b00110100;
    instrMem[5] = 8'b01100110;

    end
  end

assign instr = instrMem[pc];

endmodule
```

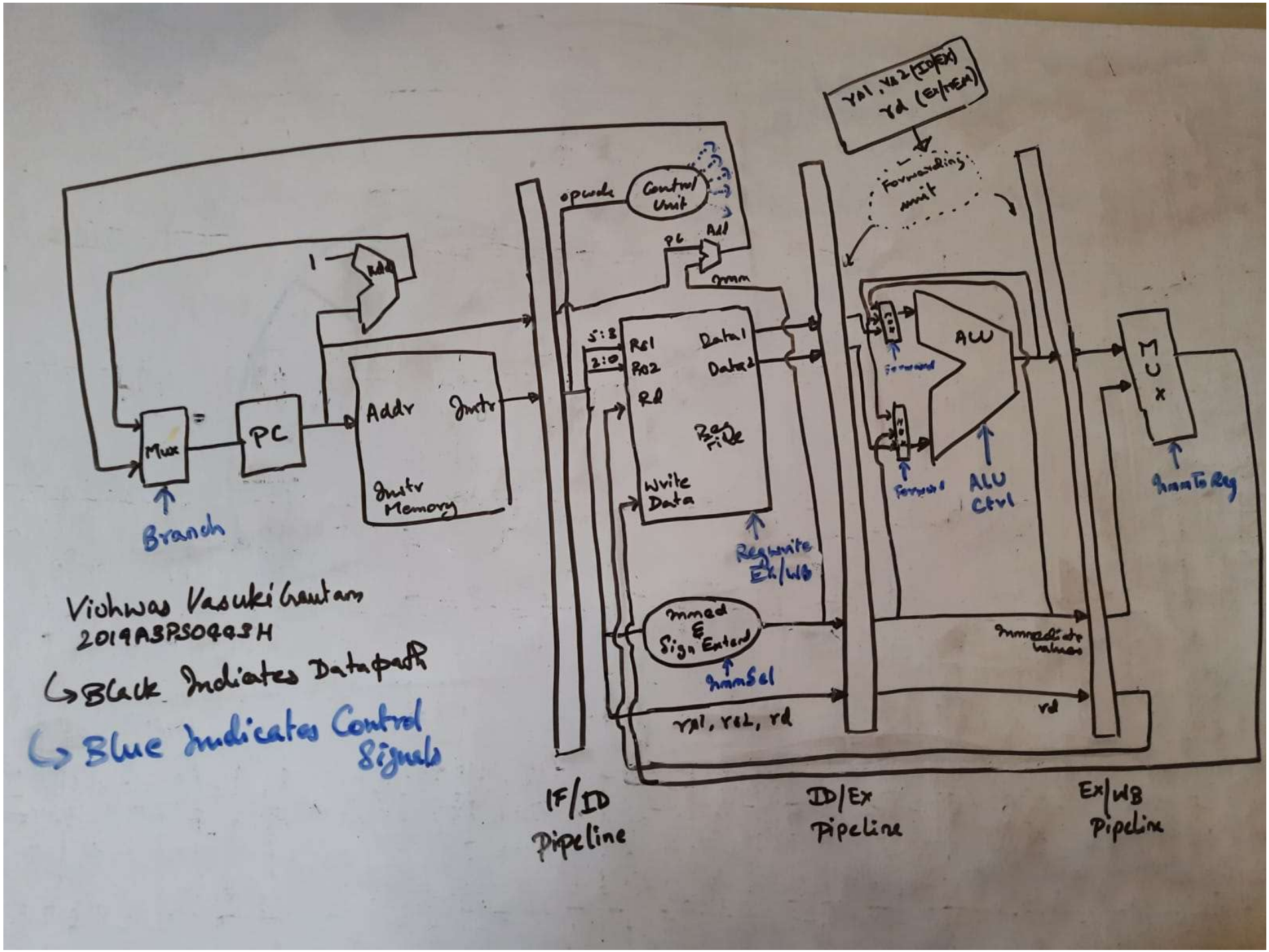
2. **List the control signals used and also the values of control signals for different instructions.**

Answer:

Instructions	Control Signals					
	ImmSel	aluCtrl	regWrite	imm2Reg	branch	
li	01	X	1	1	0	
add	00	1	1	0	0	
j	11	X	0	x	1	

3. **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:


Vishwas Vasuki Gautam
2019A3PS0403H
↳Black Indicates Datapath
↳Blue Indicates Control Signals
Branch
Mux
PC
Addr
Instr
Instr Memory
Control Unit
opcode
Add
Reg File
Read Data1
Read Data2
Write Data
RegWrite
ALU
ALU Ctrl
Forward
MUX
MemToReg
Forwarding unit
Immed & Sign Extend
ImmSel
rs1, rs2, rd
rd
IF/ID Pipeline
ID/EX Pipeline
EX/WB Pipeline

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

Answer:

A data hazard might occur when the destination register of the ADD instr is needed by another ADD instr immediately. Or when an Li instr is storing value in a register and an ADD instruction immediately needs the values. So, based on the opcode, regWrite, source & destination addresses in the id/ex and ex/wb pipeline will help us find the data hazards.

The cases to forwards are:

1. (regWrite_EX_WB == 1) && (rd_EX_WB == rs1_ID_EX) && (opcode_EX_WB == 2'b01)
2. (regWrite_EX_WB == 1) && (rd_EX_WB == rs2_ID_EX) && (opcode_EX_WB == 2'b01)
3. (regWrite_EX_WB == 1) && (rd_EX_WB == rs2_ID_EX == rs1_ID_EX) && (opcode_EX_WB == 2'b01)
4. (regWrite_EX_WB == 1) && (rd_EX_WB == rs1_ID_EX) && (opcode_EX_WB == 2'b00)
5. (regWrite_EX_WB == 1) && (rd_EX_WB == rs2_ID_EX) && (opcode_EX_WB == 2'b00)

5. **Implement the forwarding unit and copy the image of Verilog code of forwarding unit here.**

Answer:

```
module forwardingUnit(
    input [2:0] rs1_ID_EX, rs2_ID_EX,
    input [2:0] rd_EX_WB,
    input [7:0] data1_ID_EX, data2_ID_EX, aluResult_EX_WB, immOut_EX_WB,
    input regWrite_EX_WB,
    input [1:0] opcode_EX_WB,

    output reg [2:0] forward,
    output reg [7:0] aluIn1,
    output reg [7:0] aluIn2
    );

    always @(*)
        begin

            if((regWrite_EX_WB == 1) && (rd_EX_WB == rs1_ID_EX) && (rd_EX_WB == rs2_ID_EX) &&
(opcode_EX_WB == 2'b01))
                begin
                forward <= 3'b101;
                aluIn1 = aluResult_EX_WB;
                aluIn2 = aluResult_EX_WB;
                end
            else if((regWrite_EX_WB == 1) && (rd_EX_WB == rs1_ID_EX) && (opcode_EX_WB == 2'b01))
                begin
                forward <= 3'b001;
                aluIn1 <= aluResult_EX_WB;
                aluIn2 <= data2_ID_EX;
                end
            else if((regWrite_EX_WB == 1) && (rd_EX_WB == rs2_ID_EX) && (opcode_EX_WB == 2'b01))
                begin
                forward <= 3'b010;
                aluIn1 <= data1_ID_EX;
                aluIn2 <= aluResult_EX_WB;
                end
            else if((regWrite_EX_WB == 1) && (rd_EX_WB == rs1_ID_EX) && (opcode_EX_WB == 2'b00))
                begin
                forward <= 3'b011;
                aluIn1 <= immOut_EX_WB;
                aluIn2 <= data2_ID_EX;
                end
            else if((regWrite_EX_WB == 1) && (rd_EX_WB == rs2_ID_EX) && (opcode_EX_WB == 2'b00))
                begin
                forward <= 3'b100;
                aluIn1 = data1_ID_EX;
                aluIn2 = immOut_EX_WB;
                end
            else
                begin
                forward <= 3'b000;
                aluIn1 = data1_ID_EX;
                aluIn2 = data2_ID_EX;
                end

        end

endmodule
```

6. **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:

```
module topUnit(
    input clk,
    input rst
    );

    wire [7:0] instrCode, pcIn, pcOut, immOut, pc_EX_WB, immOut_EX_WB;
    wire branch, branch_EX_WB, branch_ID_EX;
    wire [7:0] pc_ID_EX;
    wire [7:0] jmpAddr;
    wire flush;
    assign flush = branch;

    assign pcIn = branch? (jmpAddr):(pcOut + 1);
//    assign pcIn = pcOut + 1;

    //Instruction fetch
    pc programCounter(clk, rst, pcIn, pcOut);
    instrMem IM(clk, rst, pcOut, instrCode);

    wire [7:0] instrCode_IF_ID, pc_IF_ID;

    // IF-ID pipeline
    IF_ID_pipeline ifid(clk, rst, flush, instrCode, pcOut, instrCode_IF_ID, pc_IF_ID);

    //Instruction decode
    wire [1:0] opcode, immSel;
    wire [2:0] readReg1, readReg2, writeReg;
    wire regWrite, immToReg, aluControl, regWrite_EX_WB;
    wire [7:0] readData1, readData2, writeData_EX_WB;
    wire [2:0] rd_EX_WB;

    assign opcode = instrCode_IF_ID[7:6];
    assign readReg2 = instrCode_IF_ID[2:0];
    assign readReg1 = instrCode_IF_ID[5:3];
    assign writeReg = instrCode_IF_ID[5:3];

    registerFile regBank(clk, rst, readReg1, readReg2, rd_EX_WB, regWrite_EX_WB, readData1, readData2,
writeData_EX_WB);
    controlUnit ctrlSignal(opcode, immSel, aluControl, regWrite, immToReg, branch);
    immGen imm(instrCode_IF_ID, immSel, immOut);

    assign jmpAddr = pc_IF_ID + immOut;

    wire aluControl_ID_EX;
    wire [2:0] rd_ID_EX, rs1_ID_EX, rs2_ID_EX;
    wire regWrite_ID_EX;
    wire immToReg_ID_EX;
    wire [7:0] immOut_ID_EX, data1_ID_EX, data2_ID_EX;
    wire [1:0] opcode_ID_EX;

    //ID-EX pipeline
    ID_EX_pipeline idex(clk, rst, aluControl, readData1, readData2, jmpAddr, writeReg, readReg1,
readReg2, regWrite, branch, immToReg, immOut, opcode,
                    aluControl_ID_EX, data1_ID_EX, data2_ID_EX, pc_ID_EX, rd_ID_EX, rs1_ID_EX, rs2_ID_EX,
regWrite_ID_EX, branch_ID_EX, immToReg_ID_EX, immOut_ID_EX, opcode_ID_EX);

    // Execute
    wire [7:0] aluResult;
    wire [7:0] aluIn1, aluIn2;
       //forwarding
    wire [2:0] forward;
    wire [7:0] aluResult_EX_WB;
    wire [1:0] opcode_EX_WB;

    forwardingUnit fwd(rs1_ID_EX, rs2_ID_EX, rd_EX_WB, data1_ID_EX, data2_ID_EX, aluResult_EX_WB,
immOut_EX_WB, regWrite_EX_WB, opcode_EX_WB, forward, aluIn1, aluIn2);
    ALU ex(aluControl_ID_EX, aluIn1, aluIn2, aluResult);

    wire immToReg_EX_WB;

    // EX-WB stage
    EX_WB_pipeline exwb(clk, rst, pc_ID_EX, aluResult, immOut_ID_EX, branch_ID_EX, regWrite_ID_EX,
immToReg_ID_EX, rd_ID_EX, opcode_ID_EX,
                        pc_EX_WB, aluResult_EX_WB, immOut_EX_WB, branch_EX_WB, regWrite_EX_WB,
immToReg_EX_WB, rd_EX_WB, opcode_EX_WB);

   //Writeback
   assign writeData_EX_WB = immToReg_EX_WB? immOut_EX_WB: aluResult_EX_WB;



endmodule
```

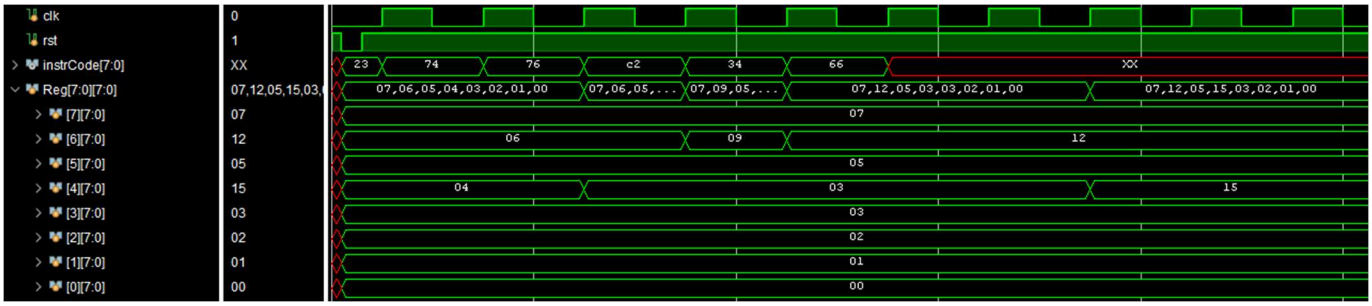
7. **Test the pipelined processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

Answer:

```
module topUnit_test(

    );
    reg clk, rst;

    topUnit uut(clk, rst);

    always #5 clk = ~clk;

    initial
        begin
            rst <= 1;
            clk <= 0;
            #1
            rst <= 0;

            #2
            rst <= 1;

            #100
            $finish;
        end
endmodule
```

8. **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: Faced problems while implementing jump instructions, and faced problems while developing the forwarding unit.

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: Implemented the processor on my own. Had healthy discussion with Naren Suraj and Shankar while implementing the processor.

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: VISHWAS VASUKI GAUTAM **Date:** 23/04/2022
ID No.: 2019A3PS0443H