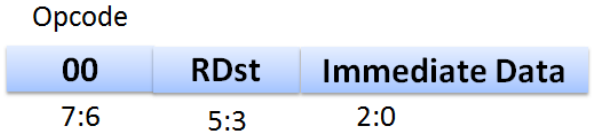
Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2021-22
CS F342: Computer Architecture Assignment (30 Marks)

A. Implement 4-stage pipelined processor in Verilog. This processor supports constant addition (addi), shift left logical (sll) 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).

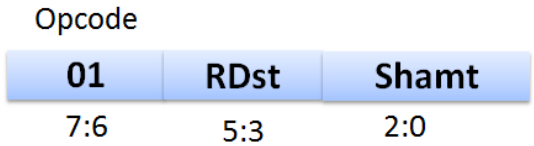
The instructions and its **8-bit instruction format** for single cycle and pipeplined processor are shown below:

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 00)



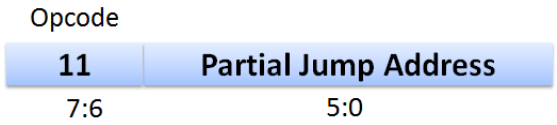
Example usage: addi R2, 3 3 gets sign extended to 8-bits 00000011 then is added to R2 (R2←R2+ 00000011)

sll DestinationReg, shiftamount (Left shifts data in register specified by register number in RDst field by shift amount and moves back result to same register. Opcode for sll is 01)



Example usage: sll R0, 4 shifts value in R0 by 4 times and store result back in 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)

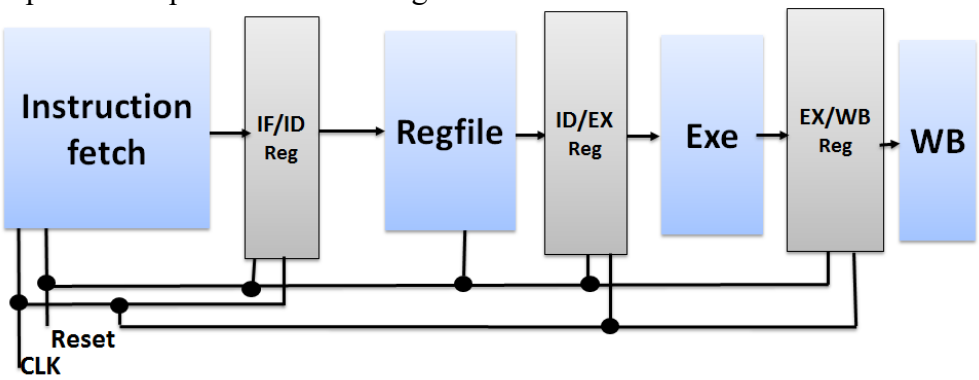
Assume the register file contains 8 registers (R0-R7) each register can hold 8-bit data. On reset register file should get initialized such that R0 = 0, R1 = 1, R2 = 2, R3 = 3 …etc. On reset assume that the instruction memory gets initialized with four instructions.

```
        addi Rx, 3
        sll Rx, 1
        addi Ry, 4
        j L1
        sll Ry, 3
L1:     addi Rz, 2
```

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: NAREN VILVA **ID No:** 2019AAPS0236H

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

Answer:

```
module InstructionFetch(input Clk, input Reset, input [7:0] PCin, output reg [7:0] PCout, output [7:0]
Instruction_Code);

InstructionMemory IM(PCout,Reset,Instruction_Code);
always@(posedge Clk,negedge Reset)
begin
    if(Reset == 0)
    PCout <= 0;
    else
    PCout <= PCin;
end
```

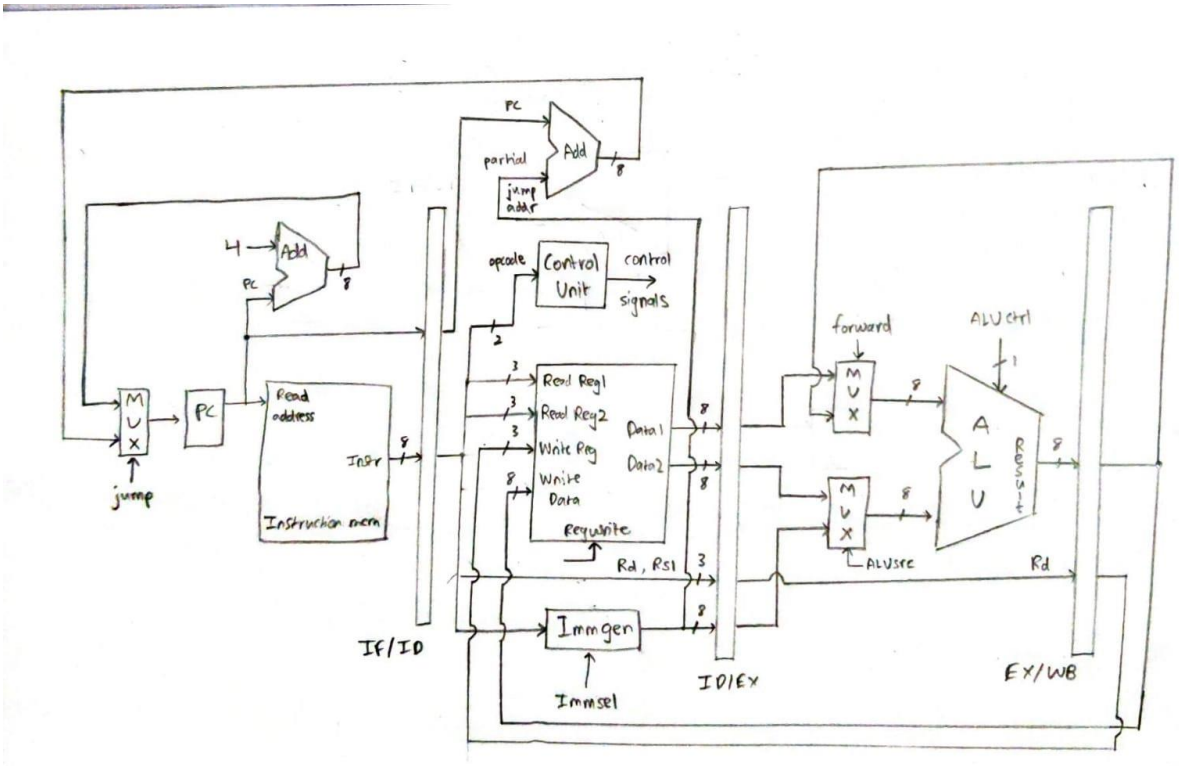
2. **List the control signals used and also the values of control signals for different instructions.**

Answer:

Instructions	**Control Signals**					
	ALUctrl	ALUsrc	RegWrite	jump	ImmSel	
addi	0	1	1	0	0	
sll	1	1	1	0	0	
j	x	x	0	1	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:



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

Answer: EX/WB.RegWrite == 1 and ID/EX.Rs1 == EX/WB.Rd

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

Answer:

```
module ForwardingUnit(Rs1_ID_EX,Rd_EX_WB,RegWrite_EX_WB,Forward);

input [2:0] Rs1_ID_EX,Rd_EX_WB;
input RegWrite_EX_WB;
output reg Forward;

always @ (*)
begin
    if(RegWrite_EX_WB == 0)
        Forward = 0;
    else if ((Rs1_ID_EX == Rd_EX_WB) && (RegWrite_EX_WB == 1))
        Forward = 1;
    else Forward = 0;
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 Processor(input Clk, input Reset);

wire [7:0] Instruction_Code,ALU_result,Read_Data_1,Read_Data_2;
wire [7:0] PC,PCout,PCplus1,PCpluslabel;
wire [7:0] extended,Data_1,Data_2;
wire ALUctrl,jump,ImmSel,ALUsrc,RegWrite,Zero;

wire [7:0] instrCode_IF_ID,PC_IF_ID;

wire [2:0] Rd_ID_EX,Rs1_ID_EX;
wire RegWrite_ID_EX,ALUctrl_ID_EX,ALUsrc_ID_EX,jump_ID_EX;
wire [7:0] PC_ID_EX,Data1_ID_EX,Data2_ID_EX,extended_ID_EX;

wire [2:0] Rd_EX_WB;
wire RegWrite_EX_WB;
wire [7:0] PCpluslabel_EX_WB,ALUresult_EX_WB;

InstructionFetch IF(Clk,Reset,PCout,PC,Instruction_Code);

// Calculate PC+1 and PC+label and select PC based on jump signal
Add8bit ADD1(PC,8'b0000_0001,PCplus1);
Add8bit ADD2(PC_IF_ID,extended,PCpluslabel);
Mux M1(PCplus1,PCpluslabel,jump,PCout);

// Flush IF/ID register when jump instruction
wire flush;
assign flush = jump;
// IF/ID Register
IF_ID_reg pipreg1(Clk,Reset,flush,Instruction_Code,PC,instrCode_IF_ID,PC_IF_ID);

wire [2:0]Rd,Rs1,Rs2;
wire [1:0]opcode;
assign Rs1 = instrCode_IF_ID[5:3];
assign Rs2 = instrCode_IF_ID[5:3];
assign Rd = instrCode_IF_ID[5:3];
assign opcode = instrCode_IF_ID[7:6];
```

```
// Register File
RegisterFile RF(Rs1,Rs2,Rd_EX_WB,ALUresult_EX_WB,
                Read_Data_1,Read_Data_2,RegWrite_EX_WB,Clk,Reset);

// Control Unit
ControlUnit CU(opcode,ALUctrl,ALUsrc,RegWrite,jump,ImmSel);

// ID/EX Register
ID_EX_reg pipreg2(Clk,Reset,Read_Data_1,Read_Data_2,extended,Rd,Rs1,
            RegWrite,ALUctrl,ALUsrc,jump,
            Data1_ID_EX,Data2_ID_EX,extended_ID_EX,Rd_ID_EX,Rs1_ID_EX,
            RegWrite_ID_EX,ALUctrl_ID_EX,ALUsrc_ID_EX,jump_ID_EX);

// Forwarding unit
wire Forward;
ForwardingUnit FU(Rs1_ID_EX,Rd_EX_WB,RegWrite_EX_WB,Forward);

Mux M2(Data2_ID_EX,extended_ID_EX,ALUsrc_ID_EX,Data_2); // Mux with inputs ReadData2,ImmGen, output to
ALU
Mux M3(Data1_ID_EX,ALUresult_EX_WB,Forward,Data_1); // Forwarding Mux with inputs ReadData1,ALUresult,
output to ALU

// ALU
ALU ALU1(Data_1,Data_2,ALUctrl_ID_EX,ALU_result,Zero);

// Immediate data generator
ImmGen IG(instrCode_IF_ID,ImmSel,extended);

// EX/WB Register
EX_WB_reg pipreg3(Clk,Reset,RegWrite_ID_EX,ALU_result,Rd_ID_EX,
                  RegWrite_EX_WB,ALUresult_EX_WB,Rd_EX_WB);
endmodule
```

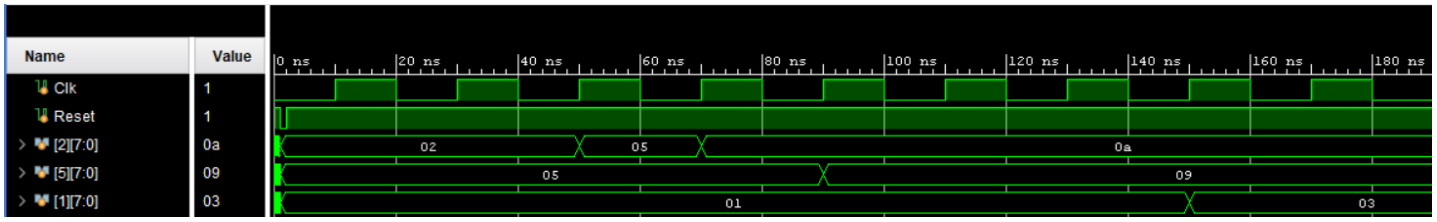
7. **Test the pipelined processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

Answer:

```
module processor_tb();

reg Clk,Reset;
Processor P1(Clk,Reset);

initial begin
Clk = 1'b0;
repeat(20)
#10 Clk = ~Clk;
end

initial begin
Reset = 1'b1;
#1 Reset = 1'b0;
#1 Reset = 1'b1;
end

endmodule
```

8. **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, and RESET):
Instructions: (for 2019AAPS0236H, x = 2 , y = 5, z = 1)
addi R2,3
sll R1,1
addi R5,4
j L1
sll R5,3
L1: addi R1,2



Unrelated Questions

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

Answer: The main issue was implementing the jump instruction to work in the single cycle processor, and then in the pipelined processor, as a flush functionality had to also be implemented. The location of the adder module for the jump address calculation was also tricky.

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: While I implemented most of the processor on my own, I took advice from Vishwas Gautam and Suraj S. for implementing the forwarding unit.

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: Naren Vilva **Date:** 24/04/22
ID No.: 2019AAPS0236H