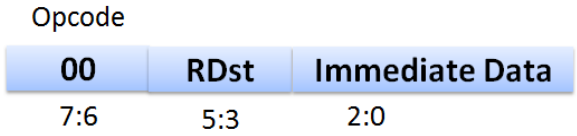
Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2021-22
CS F342: Computer Architecture Assignment (30 Marks)

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).

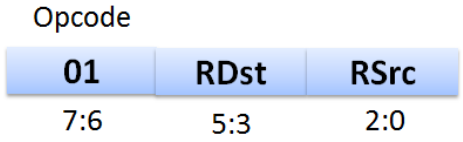
The instructions and its **8-bit instruction format** for single cycle and pipeplined processor are shown below:

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)



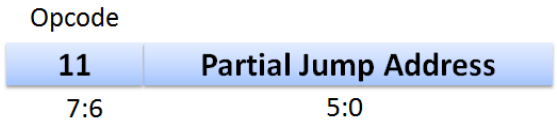
Example usage: li R3, 4 (4 = 100 signextension will result in 1111100. This data moves in to R3.

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)

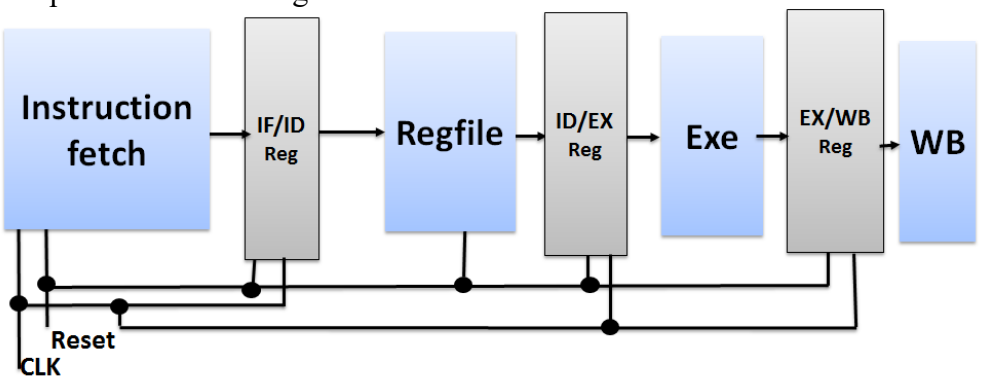
Assume the register file contains 8 registers (R0-R7) each register can hold 8-bit data. On reset register file should get initialized such that R0 = 0, R1 = 1, R2 = 2, R3 = 3 …etc. On reset assume that the instruction memory gets initialized with four instructions.

```
        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: Dhruv Makwana **ID No:** 2019A3PS0381H

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

Answer:

```
module instruction_memory (
    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) begin
        mem[0] = 8'b00_000_011;
        mem[1] = 8'b01_101_000;
        mem[2] = 8'b01_011_101;
        mem[3] = 8'b11_000001;
        mem[4] = 8'b00_011_100;
        mem[5] = 8'b01_000_011;
    end
end

endmodule
```

```
1   module PC (
2       input clk,
3       input rst,
4       input [7:0] jump_addr_from_ID,
5       input JumpPC,
6       output reg [7:0]PC
7   );
8
9   always @(posedge clk or negedge rst) begin
10      if(!rst) begin
11          PC <= 0;
12      end
13
14      else if(JumpPC) begin
15          PC <= jump_addr_from_ID;
16      end // else if(JumpPC)
17
18      else if (!JumpPC) begin
19          PC <= PC + 1;
20      end
21  end
22
23  endmodule
```

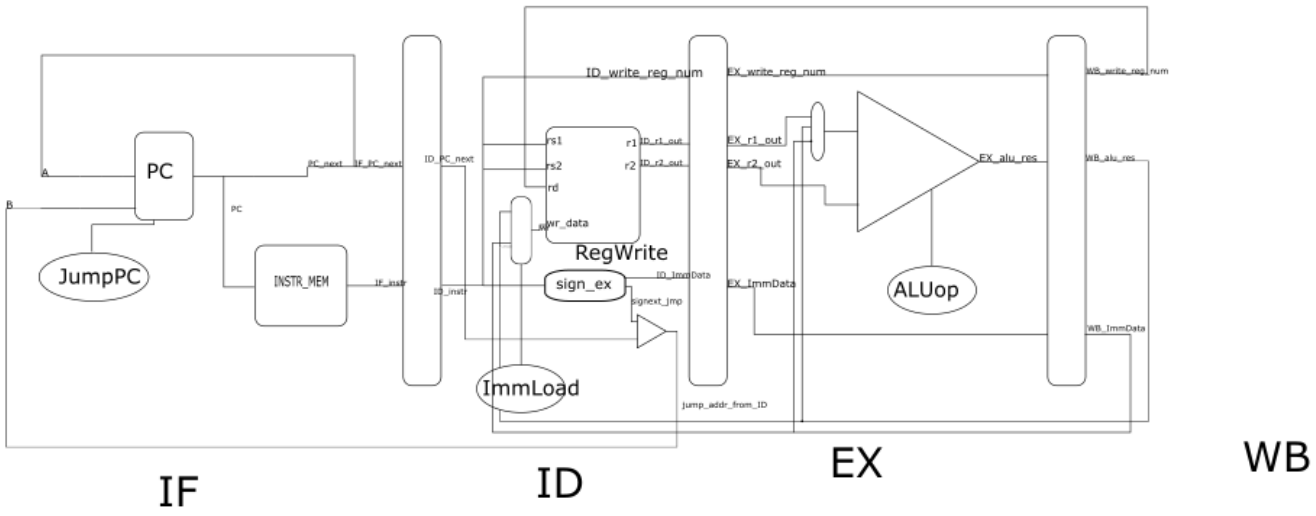
2. List the control signals used and also the values of control signals for different instructions.

Answer:

Instructions	**Control Signals**					
	JumpPC	ImmLoad	RegWrite	ALUop		
li	0	1	1	x		
add	0	0	1	0010		
j	1	0	0	x		

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: destination of WB_Instr == source of EX_instr

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

Answer:

```
23  module forwading_unit(
24      input [7:0] EX_instr,
25      input [7:0] WB_instr,
26      output reg [1:0] sel
27      );
28      always @(*) begin
29          //case 1, li and add
30          if((WB_instr[7:6] == 2'b00) && (EX_instr[7:6] == 2'b01)) begin
31              if(WB_instr[5:3] == EX_instr[2:0]) sel = 2'b10;
32          end
33          //case 2, add and add
34          else if((WB_instr[7:6] == 2'b01) && (EX_instr[7:6] == 2'b01)) begin
35              if(WB_instr[5:3] == EX_instr[2:0]) sel = 2'b01;
36          end
37
38          else
39              sel = 2'b00;
40      end
41  endmodule
42
```

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_top (
    input clk,
    input rst
);
//IF BLOCK
wire [7:0] PC, PC_mux_out;
wire [7:0] jump_addr_from_ID;
wire [7:0] IF_instr, IF_PC;
wire JumpPC;
assign IF_PC = PC;

PC pc0(clk, rst, jump_addr_from_ID, JumpPC, PC);
instruction_memory IM(PC, rst, IF_instr);
IF_ID p0(JumpPC, clk, rst, IF_instr, IF_PC, ID_instr, ID_PC_next);
//ID Block
wire [7:0] ID_instr, ID_PC_next, ID_ImmData, signext_jmp, regfile_mux_out, ID_r1_out, ID_r2_out;
wire [3:0] ID_ALUop;
wire ID_ImmLoad, ID_ALUop, ID_RegWrite;

mux2_1 regfile_mux(WB_alu_res, WB_ImmData, WB_ImmLoad, regfile_mux_out);
sign_extend signextend(ID_instr[2:0], ID_instr[5:0], ID_ImmData, signext_jmp);
reg_file regfile(clk, rst, WB_RegWrite, ID_instr[2:0], ID_instr[5:3], WB_write_reg_num, regfile_mux_out, ID_r1_out, ID_r2_out);
adder8bit jump_adder(signext_jmp, ID_PC_next, jump_addr_from_ID);
main_control contr0(rst, ID_instr[7:6], ID_ALUop, ID_RegWrite, ID_ImmLoad, JumpPC);
ID_EX p1(ID_instr, clk, rst, ID_ALUop, ID_RegWrite, ID_ImmLoad, ID_r1_out, ID_r2_out, ID_instr[5:3], ID_ImmData, EX_ALUop, EX_RegWrite, EX_ImmLoad,
    EX_r1_out, EX_r2_out, EX_write_reg_num, EX_ImmData, EX_instr);
//EX Block
wire [7:0] EX_r1_out, EX_r2_out, EX_ImmData, EX_ALU_res, EX_instr, fwd_mux_out;
wire [2:0] EX_write_reg_num;
wire EX_ImmLoad, EX_RegWrite;
wire [3:0] EX_ALUop;
wire [1:0] fwd_cntrl;
mux3_1 M3(EX_r1_out, WB_alu_res, WB_ImmData, fwd_cntrl, fwd_mux_out);
ALU alu0(fwd_mux_out, EX_r2_out, EX_ALUop, EX_ALU_res);
forwading_unit fwd(EX_instr, WB_instr, fwd_cntrl);

EX_WB p2(EX_instr, clk, rst, EX_RegWrite, EX_ImmLoad, EX_ALU_res, EX_ImmData, EX_write_reg_num, WB_RegWrite, WB_ImmLoad, WB_alu_res, WB_ImmData,
    WB_write_reg_num, WB_instr);

//WB Block
wire [7:0] WB_alu_res, WB_ImmData, WB_instr;
wire WB_RegWrite, WB_ImmLoad;
wire [2:0] WB_write_reg_num;
endmodule
```

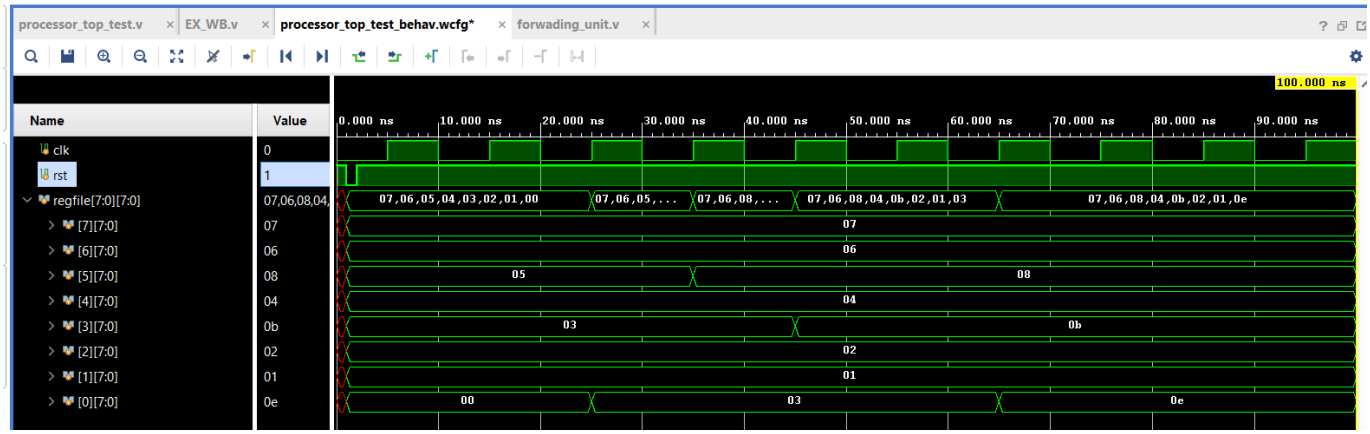
7. **Test the pipelined processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

Answer:

```
module processor_top_test ();

reg clk, rst;

    processor_top proc(clk, rst);

    initial begin
        $dumpfile("out.vcd");
        $dumpvars;
        clk = 0;
        forever #5 clk = !clk;
    end

    initial begin
        rst = 1;
        #1;
        rst = 0;
        #1;
        rst = 1;
        #200 $finish;
    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):



Unrelated Questions

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

Answer: Implementation of jumping with pipelines was hard.

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: The datapath and code was discussed with Parth Chauhan, but the code is independent

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: Dhruv Makwana **Date:** 24/4/22
ID No.: 2019A3PS0381H