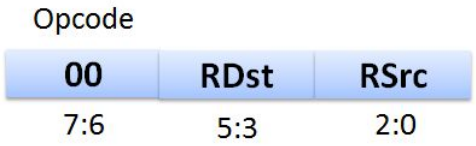
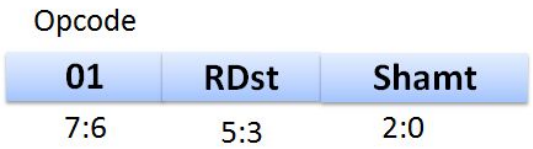
Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2018-19
CS F342: Computer Architecture Assignment (20 Marks)

1. (a) Implement 4-stage pipelined processor in Verilog. This processor supports addition (add), 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 8 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 instruction and its **8-bit instruction format** are shown below:
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 00)



Example usage: add R2, R0 (R2R2+ R0)

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 appending 2 MSB bits of PC+1 to the data specified in instruction field (5:0). Opcode for j is 11)

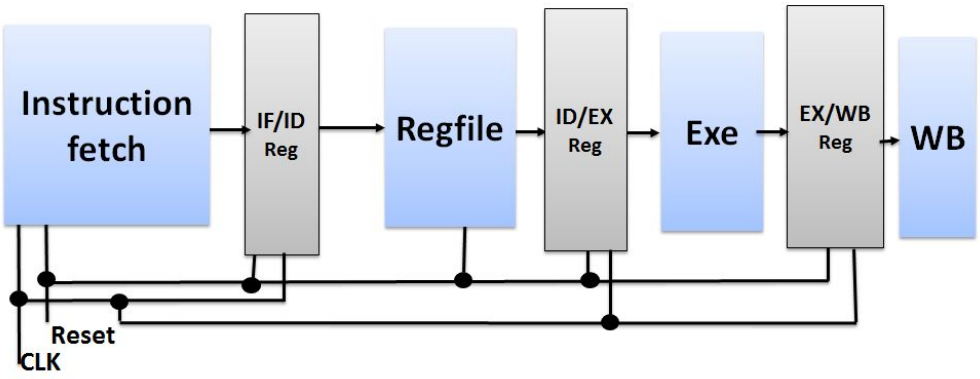
Example usage: j L1 (Jump address is calculated using Pseudo direct addressing)

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.

```
        add Rx, Ry
        sll Rx, 1
        add Ry, Rx
        j L1
        sll Ry, 3
L1:     add Rz, Ry
```

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),

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.



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).

3. Design Verilog file for the main processor.

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

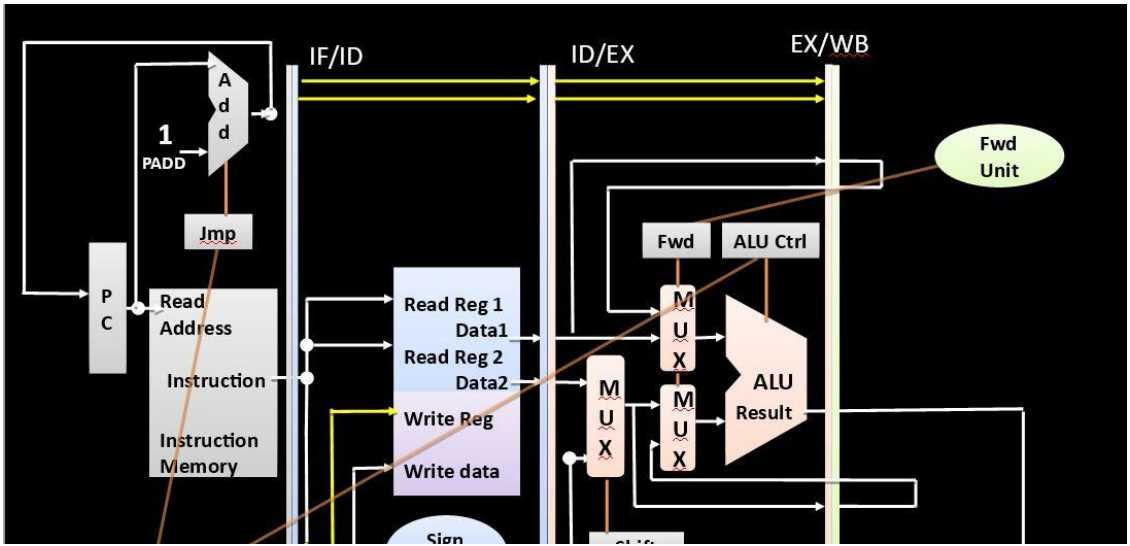
The due date for submission is 21-April-2019, 5:00 PM.

Name: Rohan Tammara **ID No: 2016A8PS0332H**

Questions Related to Assignment

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

Answer:

Instructions	Control Signals					
	reg_write	jmp	alu_control	shift		
add	1	0	0	0		
sll	1	0	1	1		
j	0	1	X	X		

3. Implement the Instruction Fetch block. Copy the image of Verilog code of the Instruction fetch b

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Instruction Memory
//
// Author: Rohan Tammara
// Last Modified: 8/4/19
//////////////////////////////////////////////////////////////////////////////////


module instruction_mem(
    input [31:0] PC,
    input rst,
    output [7:0] instruction_code
    );

    reg [7:0] i_mem [7:0];

    assign instruction_code = i_mem[PC];

    always@(rst)
    begin
    if (rst == 1)
        begin
        i_mem[0] = 8'b00011101;
        i_mem[1] = 8'b01011001;
        i_mem[2] = 8'b00101011;
        i_mem[3] = 8'b11000010;
        i_mem[4] = 8'b01101011;
        i_mem[5] = 8'b00101101;
        end
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Instruction Fetch
//
// Rohan Tammara
// 23/4/19
//////////////////////////////////////////////////////////////////////////////////


module instruction_fetch(
    input clk,
    input rst,
```

Answer:

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

Answer:

```
module register_file(
    input [2:0] read_reg_num1,
    input [2:0] read_reg_num2,
    input [2:0] write_reg_num,
    input [7:0] write_data,
    output [7:0] read_data1,
    output [7:0] read_data2,
    input reg_write,
    input rst
    );

    reg [7:0] reg_mem [7:0];

    integer i = 0;
    always@(rst)
    begin
    if(rst == 1)
    begin
    reg_mem[0] = 8'd0;
    reg_mem[1] = 8'd1;
    reg_mem[2] = 8'd2;
    reg_mem[3] = 8'd3;
    reg_mem[4] = 8'd4;
    reg_mem[5] = 8'd5;
    reg_mem[6] = 8'd6;
    reg_mem[7] = 8'd7;
    end
    end

    assign read_data1 = reg_mem[read_reg_num1];
    assign read_data2 = reg_mem[read_reg_num2];

    always@(*)
    begin
    if(reg_write == 1)
    begin
```

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

Answer: Detecting data hazard I.e, condition for EX forwarding is,

EX/WB rDst = ID/EX rDst AND EX/WB RegWrite = 1.

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

Answer:

```
module forwarding_unit(
    input exwb_regwrite,
    input exwb_dst,
    input idex_src,
    output reg fwd
    );

    always@(*)
    begin
    if(exwb_regwrite == 1 && exwb_dst == idex_src)
    begin
    fwd <= 1;
    end
    end

endmodule
```

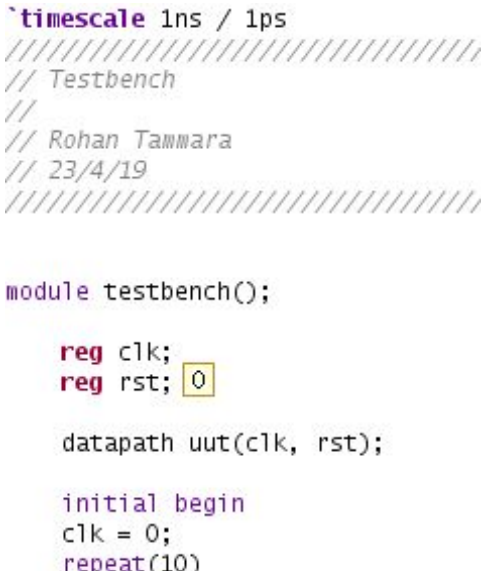
7. **Implement complete processor in Verilog (using all the Datapath blocks). Copy the <u>image</u> of Verilog code of the processor here. (Use comments to describe your Verilog implementation)**

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

    wire jmp, alu_control, shift, fwd;
    wire [7:0] instr_code;
    wire [1:0] opcode2_in, opcode3_in, opcode1_out, opcode2_out, opcode3_out;
    wire [2:0] rdst1_out, rdst2_out, rdst3_out;
    wire [2:0] rsrc1_out, rsrc2_out, rsrc3_out;
    wire reg_write2_in, reg_write2_out, reg_write3_out;
    wire [7:0] rdat1, rdat2, shamt;
    wire [7:0] rdat1_out2, rdat2_out2, rdat1_out3, rdat2_out3, shamt_out, alu_res, wb_out;

    control_unit CU(opcode1_out, jmp, reg_write2_in, alu_control, shift);

    forwarding_unit FU(reg_write3_out, rdst3_out, rdst2_out, fwd);

    instruction_fetch IF(clk, rst, jmp, instr_code);

    if_id_reg IFID(clk, rst, jmp, instr_code, rdst1_out, rsrc1_out, opcode1_out);

    register_file RF(rdst1_out, rsrc1_out, rdst1_out, wb_out, rdat1, rdat2, reg_write2_in, rst);

    id_ex_reg IDEX(clk, rst, jmp, rdat1, rdat2, shamt, rdst1_out, rsrc1_out, opcode2_in, reg_write2_in,
                   reg_write2_out, rdst2_out, rsrc2_out, opcode2_out, rdat1_out, rdat2_out, shamt_out);

    sign_extend SE(rsrc1_out, shamt);

    mux_2to1 MUX1(rdat1_out2, rdat1_out3, fwd, mux1_out);

    mux_2to1 MUX2(rdat2_out2, shamt_out, shift, rdat2_in3);

    mux_2to1 MUX3(rdat2_out2, rdat2_out3, fwd, mux3_out);

    alu ALU(mux1_out, mux3_out, alu_control, alu_res);

    ex_wb_reg EXWB(clk, rst, jmp, alu_res, rdst2_out, rsrc2_out, opcode3_in, reg_write2_out, rdat1_out2, rdat2_out2, reg_write3_out,
                   rdst3_out, rsrc3_out, opcode3_out, rdat1_out3, rdat2_out3, wb_out);

endmodule
```

Answer:

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

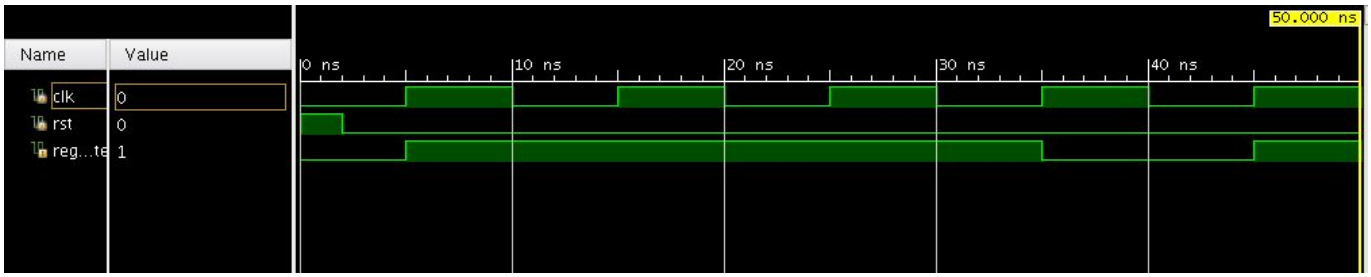
Answer:



```
`timescale 1ns / 1ps
//////////////////////////////////
// Testbench
//
// Rohan Tammara
// 23/4/19
//////////////////////////////////


module testbench();

    reg clk;
    reg rst; 0

    datapath uut(clk, rst);

    initial begin
    clk = 0;
    repeat(10)
```

9. 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: I faced problems in implementing the write back and forwarding parts of the code.

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 on my own.

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: Rohan Tammara **Date: 23/4/19**
ID No.: 2016A8PS0332H