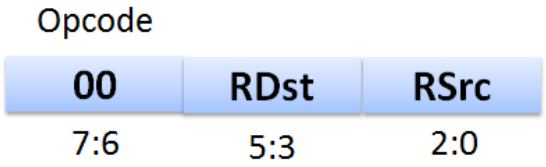
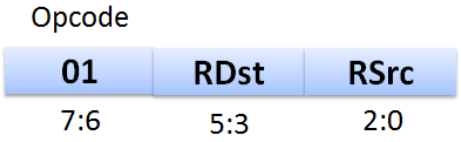
Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2018-19
CS F342: Computer Architecture Assignment (20 Marks)

1. (a) Implement 4-stage pipelined processor in Verilog. This processor supports data transfer (mov), 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 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:
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)



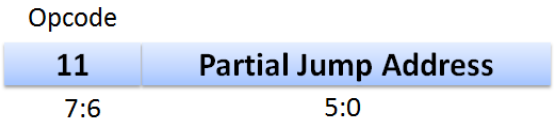
Example usage: mov R2, R0 (R2←R0)

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



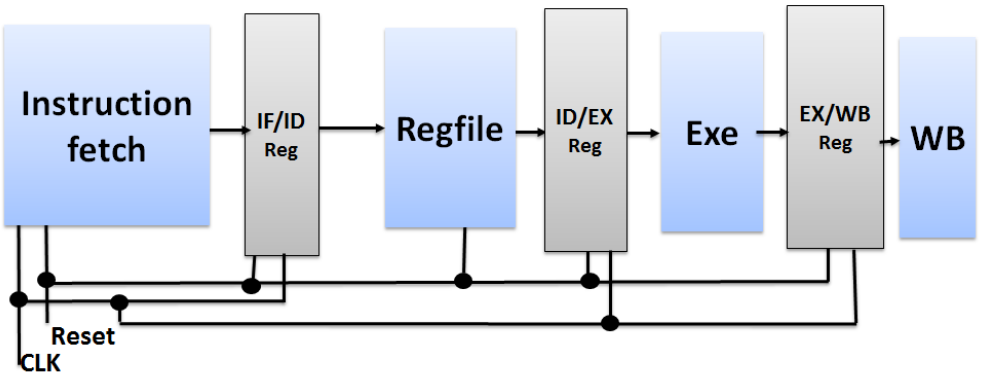
Example usage: j L1 (Jump address is calculated using Pseudo direct addressing)

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.

```
        mov Rx, Ry
        add Ry, Rx
        add Rz, Ry
        j L1
        mov Rx, Rz
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),

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

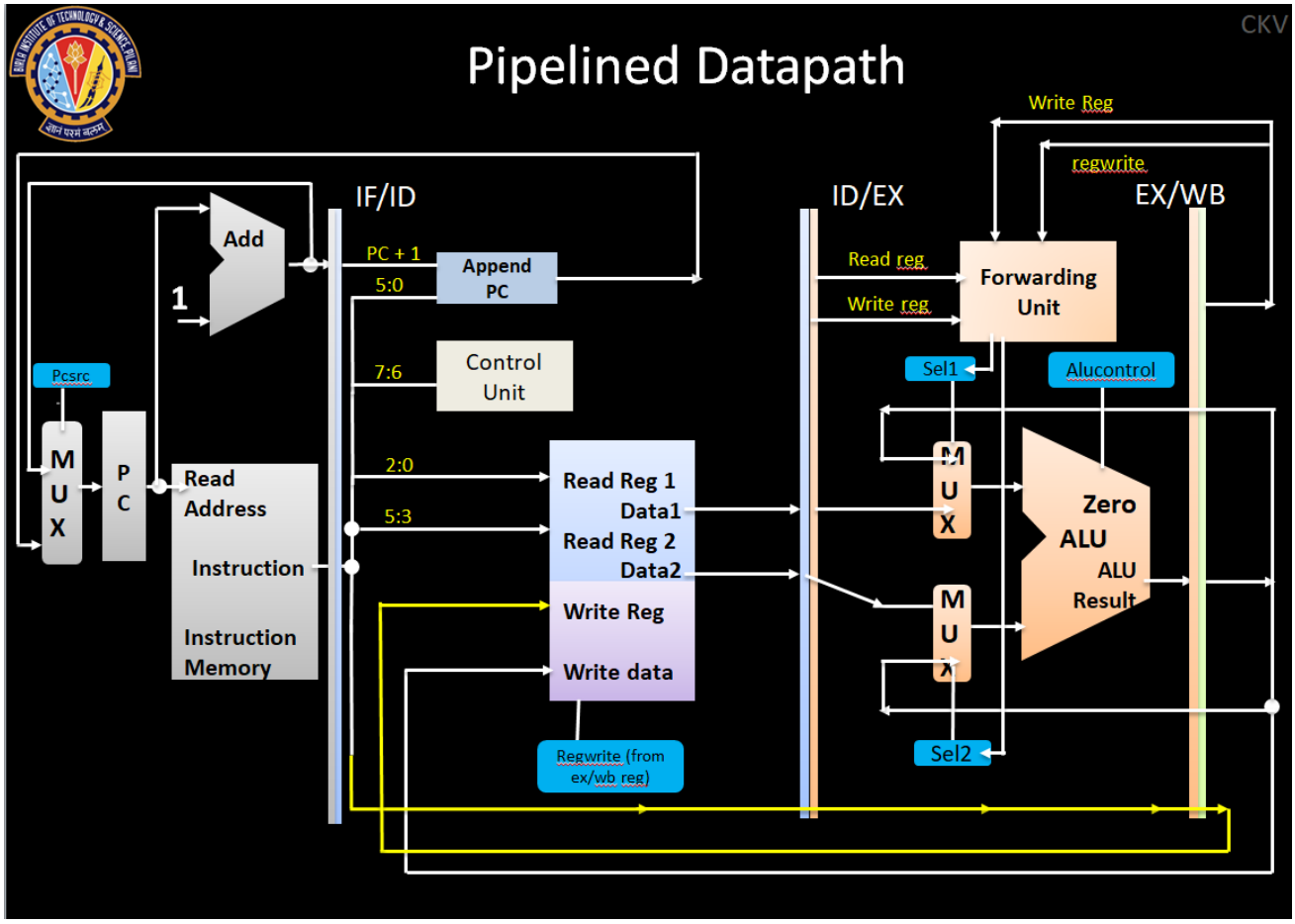
The due date for submission is 21-April-2019, 5:00 PM.

Name: Neil Thanawala **ID No:** 2015A8PS0517G

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:

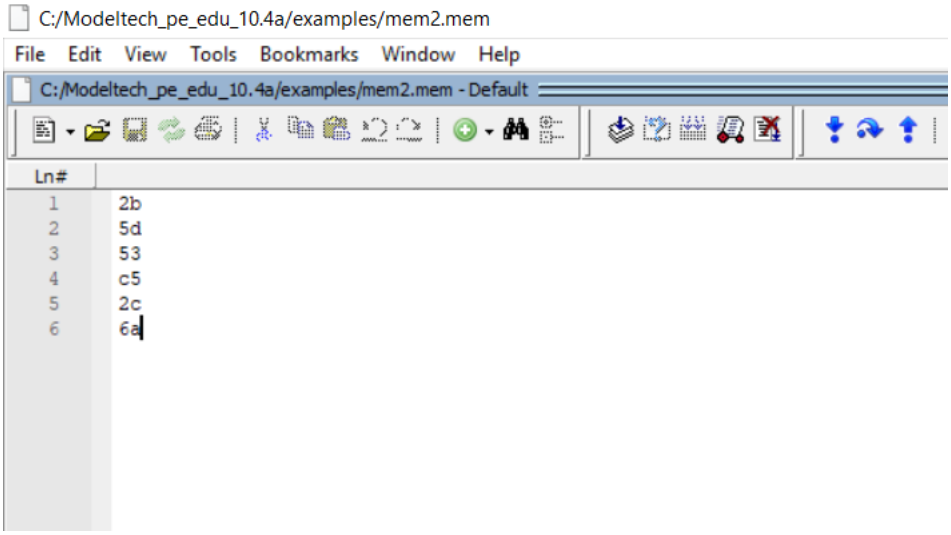
Instructions	**Control Signals**					
	regwrite	alucontrol	pcsrc			
mov	1	0	0			
add	1	1	0			
j	0	x	1			

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

Answer:

```
module instr_fetch(input clk, input reset, input [7:0] pc_i, output [7:0] instr_code, output reg [7:0] pc_o, output reg [7:0] pc_to_ifid);

instr_mem insmem(pc_o, reset, instr_code);

always @(posedge clk ,negedge reset)
begin
    if(reset==0)
    begin
        pc_o<=0;
        pc_to_ifid<=1;
    end
    else
    begin
        pc_o<=pc_i;
        pc_to_ifid<=pc_i +1;
    end
end
endmodule

module instr_mem(
input [7:0] pc,
input reset,
output [7:0] instr_code);

reg [7:0] mem [10:0];

assign instr_code = mem[pc];

initial
begin
    $readmemh("mem2.mem", mem);

end
endmodule
```

mem2.mem



Instructions Executed:

1. Mov r5, r3 (2b)
2. Add r3, r5 (5d)
3. Add r2, r3 (53)
4. j L1 (c5)
5. Mov r5, r2 (2c)
6. L1 : add r5, r2 (6a)

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

Answer:

```
module reg_file(read_reg_1, read_reg_2, write_reg, write_data, read_data_1, read_data_2,  regwrite);
input [2:0] read_reg_1, read_reg_2;
input [2:0] write_reg;
input [7:0] write_data;
output [7:0] read_data_1, read_data_2;
input regwrite;

reg [7:0] regfile [7:0];

assign read_data_1 = regfile[read_reg_1];
assign read_data_2 = regfile[read_reg_2];

initial begin
regfile [0] = 32'h00;
regfile [1] = 32'h01;
regfile [2] = 32'h02;
regfile [3] = 32'h03;
regfile [4] = 32'h04;
regfile [5] = 32'h05;
regfile [6] = 32'h06;
regfile [7] = 32'h07;
end

always @ *
begin
    if(regwrite==1)
    begin
        regfile[write_reg] <= write_data;
    end
end
endmodule
```

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

Answer: 1. Ex/wb regwrite=1 & id/ex read reg = ex/wb write reg
2. Ex/wb regwrite=1 & id/ex write reg = ex/wb write reg

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

Answer:

```
module forwarding_unit(idex_rd_reg_1, idex_rd_reg_2, exwb_wr_reg, exwb_regwrite, sel1, sel2);
input [2:0] idex_rd_reg_1,idex_rd_reg_2, exwb_wr_reg;
input exwb_regwrite;
output reg sel1, sel2;

always @ *
begin
    sel1 = 0;
    sel2 = 0;
    if(exwb_regwrite==1'b1 & (idex_rd_reg_1==exwb_wr_reg))
        sel1 = 1;
    else if(exwb_regwrite==1'b1 & (idex_rd_reg_2==exwb_wr_reg))
        sel2 = 1;
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)

Answer:

```
module processor(clk, reset);
input reset, clk;

wire [7:0] instr_code, instr_code_ifid_i, instr_code_ifid_o;
wire [7:0] pc, pc_ifid_i, pc_ifid_o, pc_idex_i, pc_idex_o, pc_final,pc_if_o, jump_add;
wire [7:0] read_data_1_idex_i,read_data_2_idex_i,read_data_1_idex_o,read_data_2_idex_o;
wire [7:0] write_data, read_data_1_x, read_data_2_x,  alu_result, alures_exwb_i, alures_exwb_o, read_data_1_fw_ex, read_data_2_fw_ex;
wire [2:0] read_reg, read_reg_idex_i, read_reg_idex_o, write_reg, write_reg_idex_i, write_reg_idex_o, write_reg_exwb_i, write_reg_exwb_o;
wire regwrite, regwrite_idex_i, regwrite_idex_o, regwrite_exwb_i, regwrite_exwb_o;
wire alucontrol, alucontrol_idex_i, alucontrol_idex_o, stall;
wire pcsrc, pcsrc_idex_i, pcsrc_idex_o, fw_sel_1, fw_sel_2;

//regfile inputs
assign read_reg = instr_code_ifid_o[2:0];
assign write_reg = instr_code_ifid_o[5:3];
assign instr_code_ifid_i = instr_code;

//idex reg inputs
assign pc_idex_i = pc_ifid_o;
assign read_data_1_idex_i = read_data_1_x;
assign read_data_2_idex_i = read_data_2_x;
assign write_reg_idex_i = write_reg;
assign regwrite_idex_i = regwrite;
assign alucontrol_idex_i = alucontrol;
assign pcsrc_idex_i = pcsrc;
assign read_reg_idex_i = read_reg;

//exwb reg inputs
assign alures_exwb_i = alu_result;
assign regwrite_exwb_i = regwrite_idex_o;
assign write_reg_exwb_i = write_reg_idex_o;

instr_fetch if1(clk, reset, pc_final, instr_code, pc_if_o, pc_ifid_i);
ifidreg ifid(clk, reset, stall, instr_code_ifid_i, pc_ifid_i, instr_code_ifid_o, pc_ifid_o);
j_append app(instr_code_ifid_o[5:0], pc_ifid_o, jump_add);
mux pcval(pcsrc, pc_ifid_i, jump_add, pc_final);

reg_file rf(read_reg, write_reg, write_reg_exwb_o, alures_exwb_o, read_data_1_x, read_data_2_x,  regwrite_exwb_o);
idexreg idex(clk, reset, stall,
read_reg_idex_i, read_data_1_idex_i,  read_data_2_idex_i, write_reg_idex_i, regwrite_idex_i, alucontrol_idex_i, read_reg_idex_o, read_data_1_idex_o, read_data_2_idex_o, write_reg_ide

alu al(read data 1 fw ex, read data 2 fw ex, alucontrol idex o, alu result);

alu al(read_data_1_fw_ex, read_data_2_fw_ex, alucontrol_idex_o, alu_result);
control_unit cu(instr_code_ifid_o[7:6], alucontrol, pcsrc, regwrite, stall);
exwbreg exwb(clk ,reset, alures_exwb_i, regwrite_exwb_i, write_reg_exwb_i, write_reg_exwb_o, alures_exwb_o, regwrite_exwb_o);

forwarding_unit fu(read_reg_idex_o, write_reg_idex_o, write_reg_exwb_o, regwrite_exwb_o, fw_sel_1, fw_sel_2);
mux fwd1(fw_sel_1, read_data_1_idex_o, alures_exwb_o, read_data_1_fw_ex);
mux fwd2(fw_sel_2, read_data_2_idex_o, alures_exwb_o, read_data_2_fw_ex);


endmodule
```

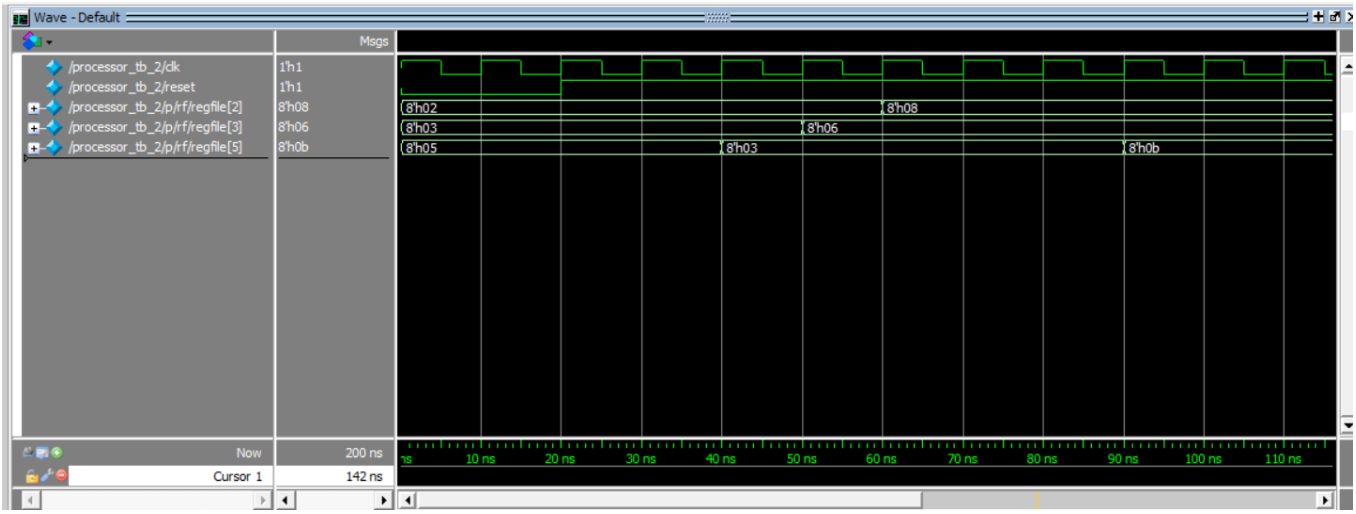
8. Test the processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.

Answer:

```
module processor_tb_2;

reg clk, reset;

processor p(clk, reset);

always #5 clk=~clk;
initial
begin
clk=1;
reset = 0;
#20;
reset=1;
#1000 $finish;
end
endmodule
```

9. 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: Updation of PC and stalling.

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: 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: Neil Thanawala **Date:** 17/4/2019
ID No.: 2015A8PS0517G