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)

Opcode

00	RDst	RSrc
7:6	5:3	2:0

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)

Opcode

01	RDst	RSrc
7:6	5:3	2:0

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)

Opcode

11	Partial Jump Address
7:6	5:0

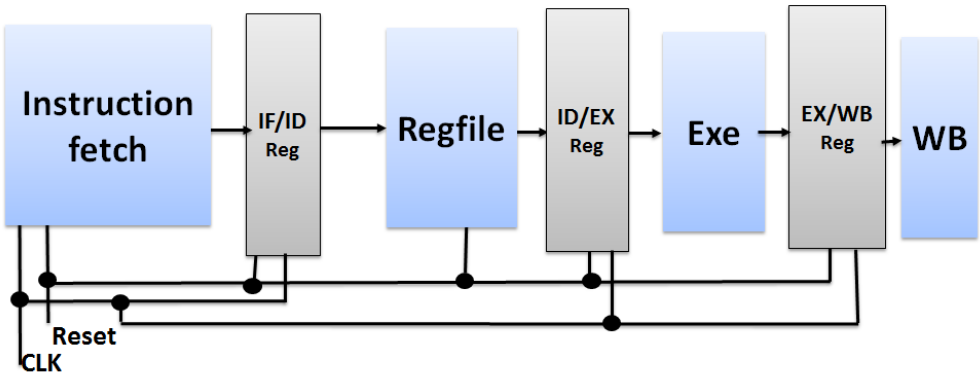
Example usage: j L1 (Jump address is calculated using PC relative addressing)

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.

```
        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

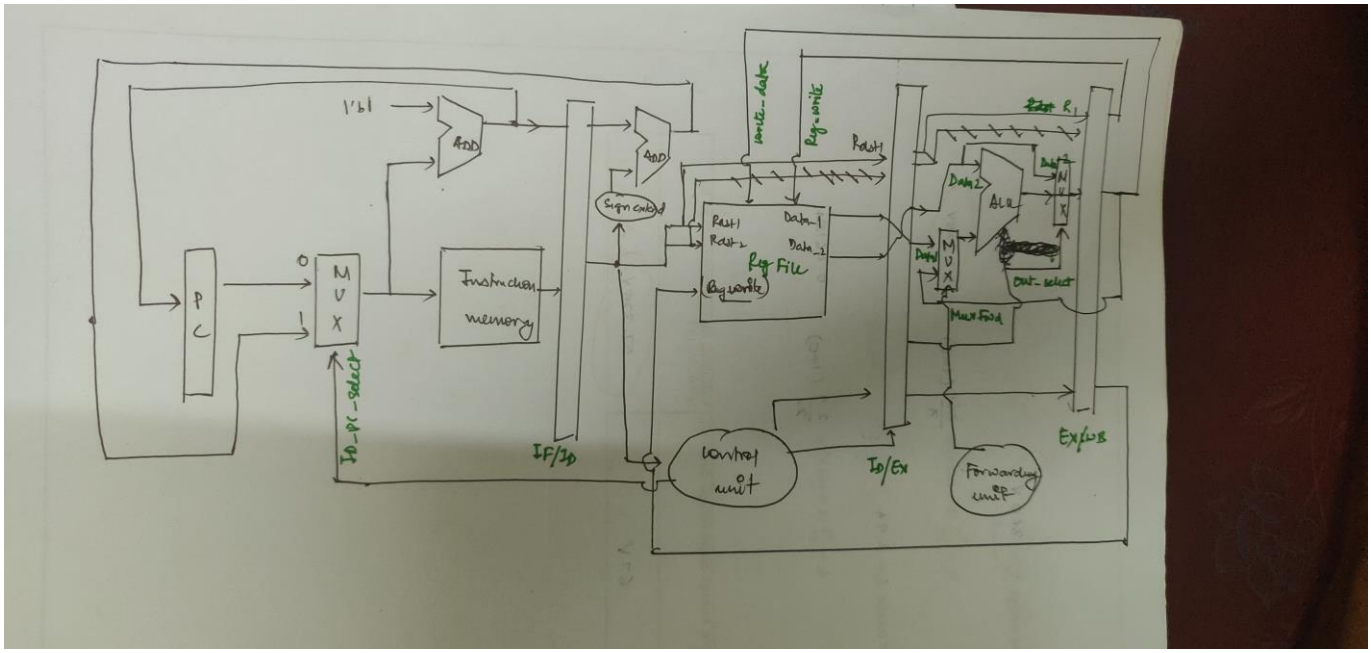
The due date for submission is 21-April-2019, 5:00 PM.

Name: Ayush Mall **ID No:** 2016A3PS0163G

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	Out_Select	PC_Select			
mov	1	1	0			
add	1	0	0			
j	0	0	1			

3. Implement the Instruction Fetch block. Copy the <u>image</u> of Verilog code of the Instruction fetch block here

Answer:

```
module IM_block(
  input [7:0] PC,
  input reset,
  output [7:0] Instruction_code);
  reg [7:0] Instruction_mem [31:0];

  assign Instruction_code = Instruction_mem[PC];

  always@(negedge reset)
  begin
    if(reset == 0)
    begin
      Instruction_mem[0] <= 8'h08;
      Instruction_mem[1] <= 8'h41;
      Instruction_mem[2] <= 8'h70;
      Instruction_mem[3] <= 8'h81;
      Instruction_mem[4] <= 8'h0E;
      Instruction_mem[5] <= 8'h4E;

    end
  end
endmodule
```

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

Answer:

```
module Reg_File(
  input [2:0] Reg1,
  input [2:0] Reg2,
  input [2:0] Write_reg,
  input [7:0] Data_Write,
  input RegWr_control,
  input reset,
  output reg [7:0] Data_Reg1,
  output reg [7:0] Data_Reg2
  );

  reg [7:0] Reg_mem [7:0];

  always@(negedge reset)
  begin
    if(reset == 0)
    begin
      Reg_mem[0] = 0;
      Reg_mem[1] = 1;
      Reg_mem[2] = 2;
      Reg_mem[3] = 3;
      Reg_mem[4] = 4;
      Reg_mem[5] = 5;
      Reg_mem[6] = 6;
      Reg_mem[7] = 7;
    end
  end

  always@(*)
  begin
    Data_Reg1 <= Reg_mem[Reg1];
    Data_Reg2 <= Reg_mem[Reg2];  //for sequential execution

    if(RegWr_control == 1)
      Reg_mem[Write_reg] <= Data_Write;
  end
endmodule
```

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

Answer: (ID_EX_Regwrite==1)&&((EX_WB_Rdst==ID_EX_Rsrc)||(EX_WB_Rdst==ID_EX_Rdst))

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

Answer:

```
module Forwarding_Unit(
  input [2:0] EX_WB_R1,ID_EX_R1,ID_EX_R2,
  input ID_EX_regwrite,
  output reg ID_Forward_control);

  always@(*)
  begin
    if((EX_WB_R1 == ID_EX_R2)&&(ID_EX_regwrite == 1))
      ID_Forward_control = 1;
    else if((EX_WB_R1 == ID_EX_R1)&&(ID_EX_regwrite == 1))
      ID_Forward_control = 1;
    else
      ID_Forward_control = 0;
  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(
  input clk, reset);


  wire [7:0] IF_PC_in,IF_PC_out,IF_PC_next;

  wire [7:0] IF_Instruction_code;

  reg [7:0] IF_ID_PC_next, IF_ID_Instruction_code;



  wire [7:0] ID_Instruction_code,ID_jmp_address;
  wire [1:0] ID_Opcode;
  wire [2:0] ID_R1,ID_R2;
  wire [5:0] ID_address;
  wire ID_regwrite,ID_Out_select,ID_PC_select,EX_Forward_control;
  wire [7:0] ID_Read_data_R1,ID_Read_data_R2,ID_PC_Jump_address,ID_PC_nex

  reg [7:0] ID_EX_Read_data_R1, ID_EX_Read_data_R2;
  reg [2:0] ID_EX_R1,ID_EX_R2;
  reg ID_EX_regwrite,ID_EX_Out_select;

  wire [7:0] EX_Read_data_R1,EX_Read_data_R2,EX_added_data,EX_final_data;
  wire [2:0] EX_R1,EX_R2;
  wire EX_regwrite,EX_Out_select;

  wire [7:0] EX_Write_data;

  reg [7:0] EX_WB_Write_data;
  reg [2:0] EX_WB_R1;
  reg EX_WB_regwrite;

  wire [2:0] WB_R1;
  wire WB_regwrite;
  wire [7:0] WB_Write_data;

//IF stage : Mux, PC_counter, IM

  //first two letters denoting which stage that signal is coming from
 PC_Reset PC(clk,reset,IF_PC_in,IF_PC_out);
 MUX_21 M1(IF_PC_out,ID_PC_Jump_address,ID_PC_select,IF_PC_next);
 Ex_block Pc_inc(1,IF_PC_next,IF_PC_in);
 IM_block IM1(IF_PC_next,reset,IF_Instruction_code);
```

```
46
47   //IF-ID Reg
48    // IF/ID Register
49     always@(posedge clk or negedge reset)
50     begin
51       if(reset == 0)
52       begin
53         IF_ID_PC_next = 0;
54         IF_ID_Instruction_code = 0;
55       end
56       else
57       begin
58         IF_ID_PC_next = IF_PC_in;
59         IF_ID_Instruction_code = IF_Instruction_code;
60
61       end
62     end
63
64
65    //ID stage : Decoder, Control Unit, Control Hazard
66     assign ID_Instruction_code = IF_ID_Instruction_code;
67     assign ID_PC_next = IF_ID_PC_next;
68
69     ID_block ID(ID_Instruction_code,ID_Opcode,ID_R1,ID_R2,ID_address);
70
71     Control_Unit CU(ID_Opcode,ID_regwrite,ID_Out_select,ID_PC_select);
72
73     Reg_File RF(ID_R1,ID_R2,WB_R1,WB_Write_data,WB_regwrite,reset,ID_Read_data_R1,ID_Read_data_R2);
74
75     Sign_Extend S1(ID_address,ID_jmp_address);
76
77     Ex_block addJump(ID_PC_next,ID_jmp_address,ID_PC_Jump_address);
78
79      // ID/EX Register
```

```
 // ID/EX Register
always@(posedge clk or negedge reset)
begin
  if(reset ==0)
  begin
    ID_EX_R1 = 0;
    ID_EX_R2 = 0;
    ID_EX_Read_data_R1 = 0;
    ID_EX_Read_data_R2 = 0;
    ID_EX_regwrite = 0;
    ID_EX_Out_select = 0;

  end
  else
  begin
    ID_EX_R1 = ID_R1;
    ID_EX_R2 = ID_R2;
    ID_EX_Read_data_R1 = ID_Read_data_R1;
    ID_EX_Read_data_R2 = ID_Read_data_R2;
    ID_EX_regwrite = ID_regwrite;
    ID_EX_Out_select = ID_Out_select;


  end
end
```

```
    //Ex_block : ALU, Forwarding unit

    assign EX_Read_data_R1 = ID_EX_Read_data_R1;
    assign EX_Read_data_R2 = ID_EX_Read_data_R2;
    assign EX_regwrite = ID_EX_regwrite;
    assign EX_Out_select = ID_EX_Out_select;
    assign EX_R1 = ID_EX_R1;
    assign EX_R2 = ID_EX_R2;


  Forwarding_Unit FU(EX_WB_R1,ID_EX_R1,ID_EX_R2,ID_EX_regwrite,EX_Forward_control);

    MUX_21 muxFWD(EX_Read_data_R1,WB_Write_data,EX_Forward_control,EX_final_data);

    Ex_block EX(EX_final_data,EX_Read_data_R2,EX_Added_data);

    MUX_21 mux2(EX_added_data,EX_Read_data_R2,EX_Out_select,EX_Write_data);



     // EX/WB Register
    always@(posedge clk or negedge reset)
    begin
      if(reset == 0)
      begin
        EX_WB_Write_data = 0;
        EX_WB_regwrite = 0;
        EX_WB_R1 = 0;
      end
      else
      begin
        EX_WB_Write_data = EX_Write_data;
        EX_WB_regwrite = EX_regwrite;
        EX_WB_R1 = EX_R1;
      end
    end


    // WB Stage
    assign WB_Write_data = EX_WB_Write_data;
    assign WB_regwrite = EX_WB_regwrite;
    assign WB_R1 = EX_WB_R1;




    endmodule
```

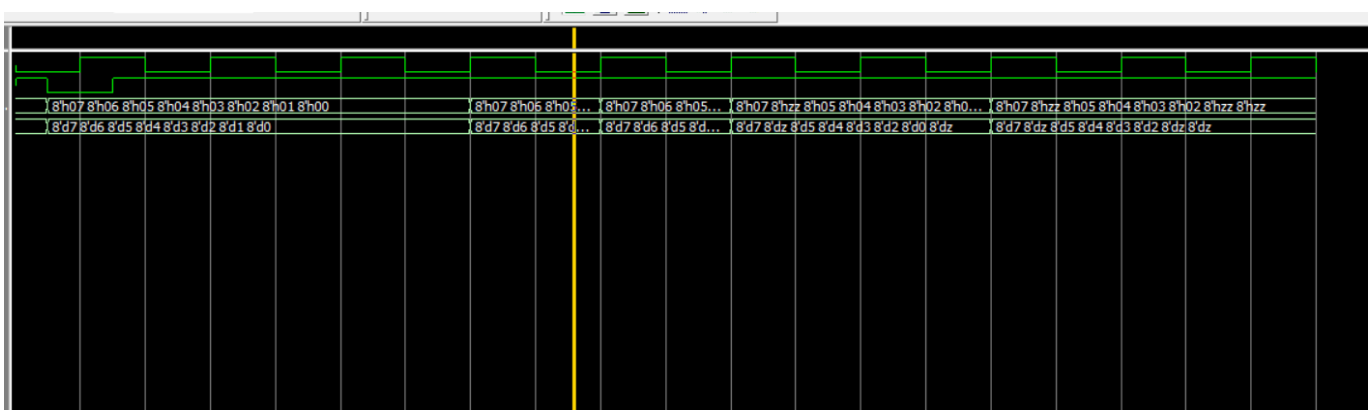
8. **Test the processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

Answer:

```
module testb_main;
  reg clk,reset;

  Processor main(clk,reset);

  initial begin
  clk = 1'b0;
  repeat(100)
  #10 clk = ~clk;
  #10 $finish;
  end

  initial begin
  reset = 1'b1;
  #5 reset = 1'b0;
  #10 reset = 1'b1;
  end

endmodule
```

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: For some reason, the R0 register would always take a high impedance value and any operation with it would give the value ZZ . The jump is happening.

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: I implemented it on my own. I had my confusions as to how to solve the control hazard. I took helo from Kushagra Shah in preventing control hazard. He also suggested that I should declare all my registers in the main processor file itself, which I did.

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: Ayush Mall **Date:** 2016A3PS0163g
ID No.: 2016a3ps0163G