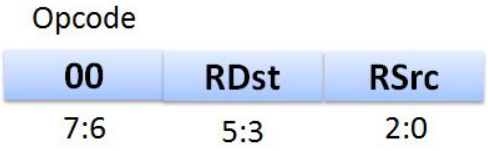
Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2018-19
CS F342: Computer Architecture Assignment (20 Marks)

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

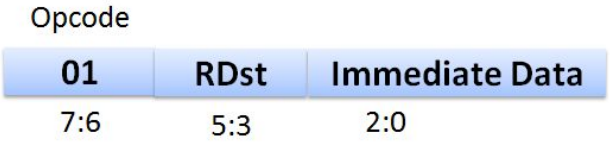
The instruction and its **8-bit instruction format** are shown below:

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 (R2R0)

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 01)



Example usage: addi R2, 3 3 gets sign extended to 8-bits 00000011 then is added to R2 (R2R2+ 00000011)

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)

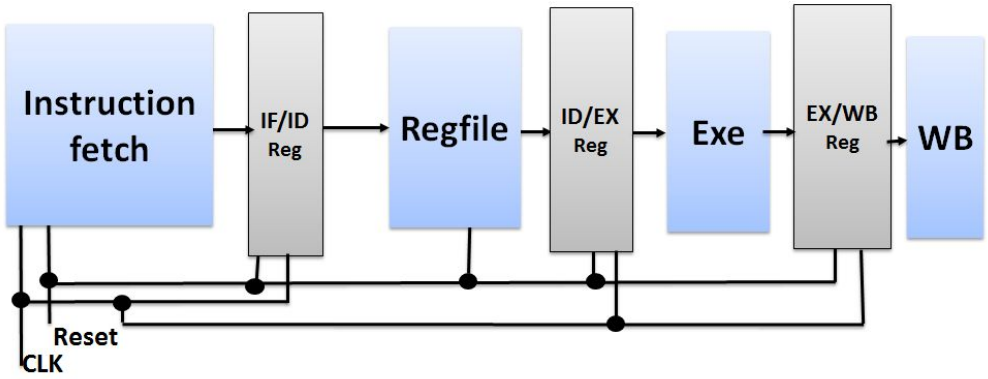
Assume the register file contains 8 registers (R0-R7) each register can hold 8-bit data. On reset register file should get initialized such that R0 = 0, R1 = 1, R2 = 2, R3 = 3 …etc. On reset assume that the instruction memory gets initialized with four instructions.

```
    mov Rx, Ry
    addi Rx, 3
    addi Ry, 2
    j L1
    mov Rz, Ry
L1: addi Rz, -3
```

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

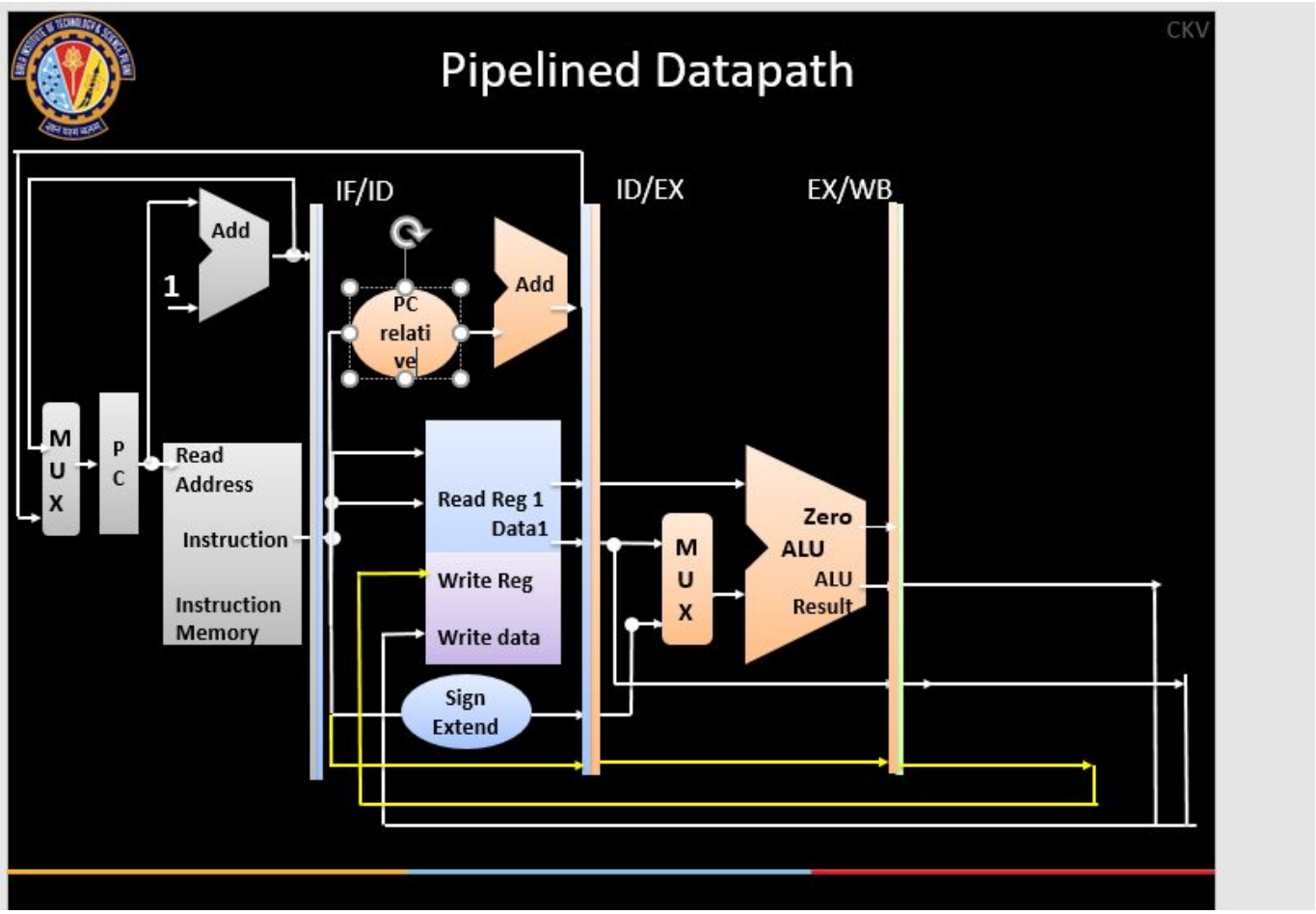
The due date for submission is 21-April-2019, 5:00 PM.

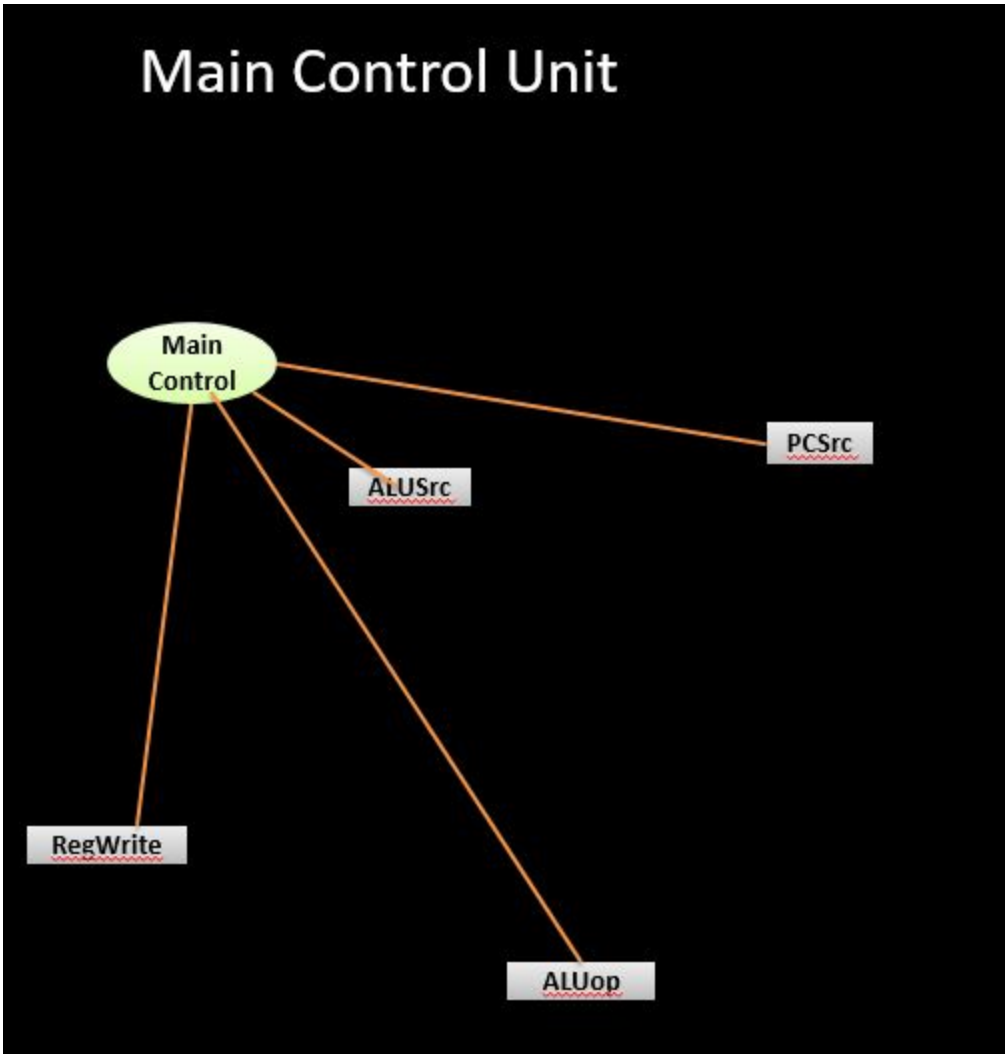
Name: VAIBHAV BALLOLI **ID No: 2016AAPS0165H**

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:


CKV
Pipelined Datapath
IF/ID
ID/EX
EX/WB
Add
1
PC
relati
ve
Add
M
U
X
P
C
Read
Address
Instruction
Instruction
Memory
Read Reg 1
Data1
Write Reg
Write data
Sign
Extend
M
U
X
Zero
ALU
ALU
Result



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

Answer:

Instructions	**Control Signals**			
	write_reg	exe_ctrl	pc_jump_sel	input_2_sel
mov	1	0	0	0
addi	1	1	0	1
j	0	X	1	X

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

Answer:

```
module instruction_fetch(
   input [7:0] instruction_address,
   input reset,
   output reg [1:0] opcode,
   output reg [2:0] rDest,
   output reg [2:0] rSrc,
   output reg [2:0] immediate_data,
   output reg [7:0] jump_address
    );

   reg [7:0] instruction_file [7:0];

   reg [7:0] instruction;

   always @ (reset)
   begin


   if (reset == 1)
      begin
      instruction_file[0] = 8'b00001000;
      instruction_file[1] = 8'b01001011;
      instruction_file[2] = 8'b01000010;
      instruction_file[3] = 8'b11000101;
      instruction_file[4] = 8'b00000000;
      instruction_file[5] = 8'b01000101;
      end
   end

   always @ (*)
   begin
      instruction = instruction_file[instruction_address];
      opcode = instruction[7:6];
      rDest = instruction[5:3];
      rSrc = instruction[2:0];
      immediate_data = rSrc;
      jump_address = {instruction_address[7:6] ,instruction[5:0]};
   end


endmodule
```

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

Answer:

```
module register_files(
      input [2:0] rSrc,
      input [2:0] rDest,
      input write_reg,
      input [7:0] writeData,
      input reset,
      output reg [7:0] srcData
    );

    reg [7:0] registers [7:0];
    integer i;

    always @ (reset)
    begin
   if (reset == 1)
   begin
      for(i=0; i<8; i=i+1)
   begin
      if (i %2 == 0)
         registers[i] = 8'b01010101;
      else
         registers[i] = 8'b00001010;
   end
   end

    end

    always @ (*)
    begin

    srcData = registers[rSrc];

    if (write_reg == 1)
      registers[rDest] = writeData;
    end


endmodule
```

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

Answer: The destination registers from the ID and EX pipeline registers should match.

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

Answer:

```
module forwarding_unit(
      input [1:0] ex_opcode,
      input [2:0] id_ex_src_reg,
      input [2:0] id_ex_dest_reg,
      input [2:0] ex_wb_reg,
      input reset,
      output reg if_id_reset,
      output reg id_ex_reset,
      output reg ex_wb_reset,
      output reg alu_forwarded_input
    );

    always @ (*)
    begin
       if (reset == 0)
       begin
         if (id_ex_dest_reg == ex_wb_reg)
         begin
            if_id_reset = 1;
            id_ex_reset = 1;
            ex_wb_reset = 0;
            alu_forwarded_input = 1;
         end

         else
         begin
            if_id_reset = 0;
            id_ex_reset = 0;
            ex_wb_reset = 0;
            alu_forwarded_input = 0;
         end
       end
       else
       begin
         if_id_reset = 1;
         id_ex_reset = 1;
         ex_wb_reset = 1;
         alu_forwarded_input = 0;
       end
    end
```

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 mips(
    input clk,
    input reset
    );

// Program Counter
wire [7:0] previous_address, instruction_address;

// Instruction Fetch
wire [1:0] opcode, if_opcode, id_opcode, exe_opcode;
wire [2:0] rSrc, rDest, immediate_data, if_rSrc, if_rDest, id_rSrc, id_rDest,
   exe_rSrc, exe_rDest;
wire signed [2:0] if_immediate_data, id_immediate_data, exe_immediate_data;
wire [7:0] jump_address, if_jump_address, id_jump_address, exe_jump_address;
wire [7:0] if_empty;

// Register File
wire [7:0] data, id_data, exe_data, alu_data;
reg [7:0] forwarded_data;

// ALU
reg [7:0] input_2;

// Forwarding unit
wire if_id_reset, id_exe_reset, exe_wb_reset;

wire write_reg, exe_ctrl, pc_jump_sel, alu_forward_control;

program_counter pc (previous_address, clk, reset, instruction_address);
```

```
instruction_fetch i_f (previous_address, reset, opcode, rDest, rSrc, immediate_data, jump_address);

pipeline_register if_id (clk, if_id_reset, opcode, rDest, rSrc, immediate_data, jump_address, 0,
   if_opcode, if_rSrc, if_rDest, if_immediate_data, if_jump_address, if_empty);

register_files rf (if_rSrc, exe_rDest, write_reg, exe_data, reset, data);

pipeline_register id_ex (clk, id_exe_reset, if_opcode, if_rDest, if_rSrc, if_immediate_data, if_jump_address, data,
   id_opcode, id_rSrc, id_rDest, id_immediate_data, id_jump_address, id_data);
```

```
// Input 1 and 2 MUX - control signal from control and forwarding unit

always @ (*)
begin
   case(id_opcode)
   2'b01:
   begin
      input_2 = id_immediate_data;
      //$write("Immediate data chosen");
   end
   2'b00:
   begin
      //$write("id_data chosen");
      input_2 = id_data;
   end
   endcase

   if (alu_forward_control == 1)
   begin
      //$write("EXE data chosen");
      forwarded_data = exe_data;
   end
   else
   begin
      //$write("ID Data chosen");
      forwarded_data = id_data;
   end

end
```

```
exe al (forwarded_data, input_2, exe_ctrl, alu_data);

pipeline_register ex_wb (clk, exe_wb_reset, id_opcode, id_rDest, id_rSrc, id_immediate_data, id_jump_address, alu_data,
   exe_opcode, exe_rSrc, exe_rDest, exe_immediate_data, exe_jump_address, exe_data);

// /* Inputs to control module for write_reg, alu_input and control signal and branch instructions
control_unit cu (if_opcode, id_opcode, exe_opcode, write_reg, exe_ctrl, pc_jump_sel);

// /* forwarding unit for deciding the register resets and input to ALU Decide between exe data and id data
forwarding_unit fu (exe_opcode, id_rSrc, id_rDest, exe_rDest, reset, if_id_reset, id_exe_reset,
   exe_wb_reset, alu_forward_control);

// /* Decide branch module
branch b (instruction_address, if_jump_address, pc_jump_sel, previous_address);

endmodule
```

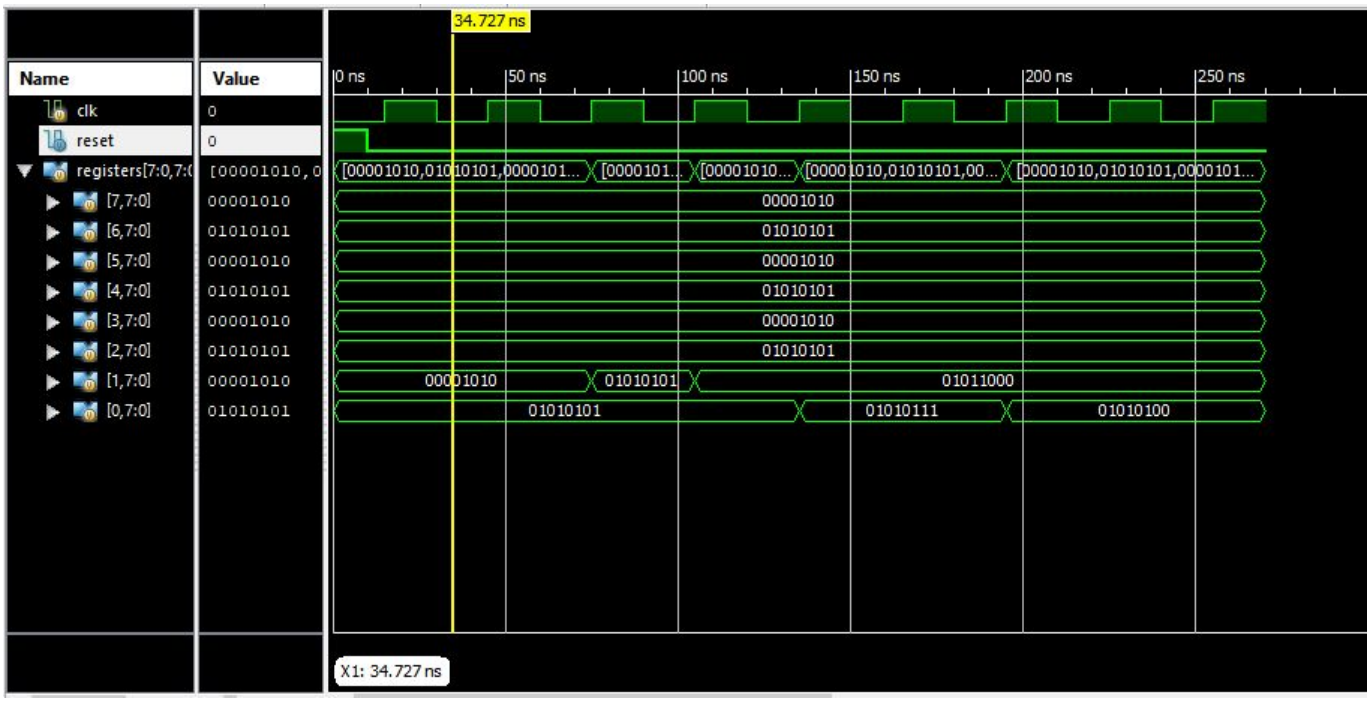
8. **Test the processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

Answer:

```
module tb_mips;

    // Inputs
    reg clk;
    reg reset;

    // Instantiate the Unit Under Test (UUT)
    mips uut (
        .clk(clk),
        .reset(reset)
    );

    initial begin
        // Initialize Inputs
        reset = 1;

        // Wait 100 ns for global reset to finish
        #10;
        reset = 0;

        // Add stimulus here

    end

    initial begin
    clk = 0;
    repeat(18)
    #15
    clk = ~clk;
    $finish;
    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: I faced issues with implementing the branch condition and the handling the forwarding control signal exactly as intended.

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 the entire processor 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: VAIBHAV BALLOLI **Date:** 23.04.19
ID No.: 2016AAPS0165H