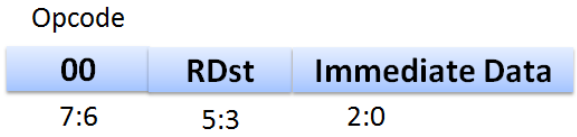
Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2018-19
CS F342: Computer Architecture Assignment (20 Marks)

1. (a) Implement 4-stage pipelined processor in Verilog. This processor supports load immediate (li), 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:
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)

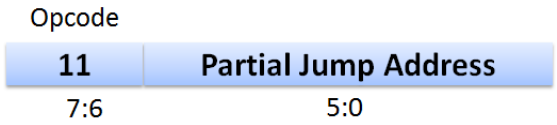


Example usage: li R3, 4 (4 = 100 sign extension will result in 1111100. This data moves in to R3.

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)



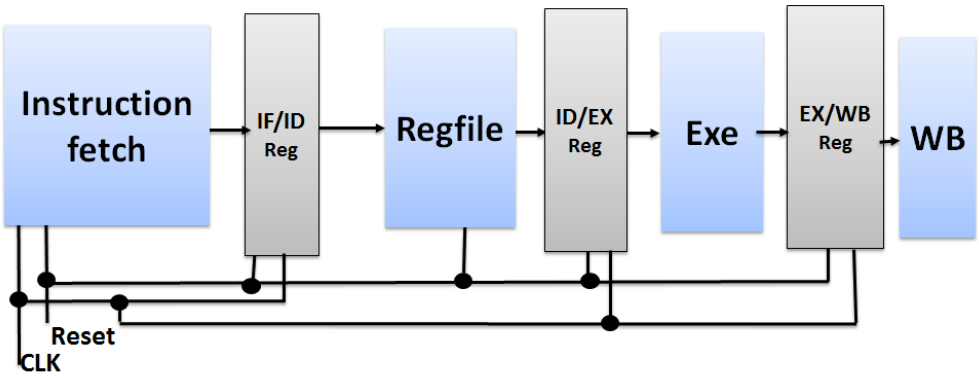
Example usage: j L1 (Jump address is calculated using Pseudo direct addressing)

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.

```
        li R1, 3
        sll R1, 1
        li R2, 2
        j L1
        sll R2, 3
L1:     li R3, 4
```

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

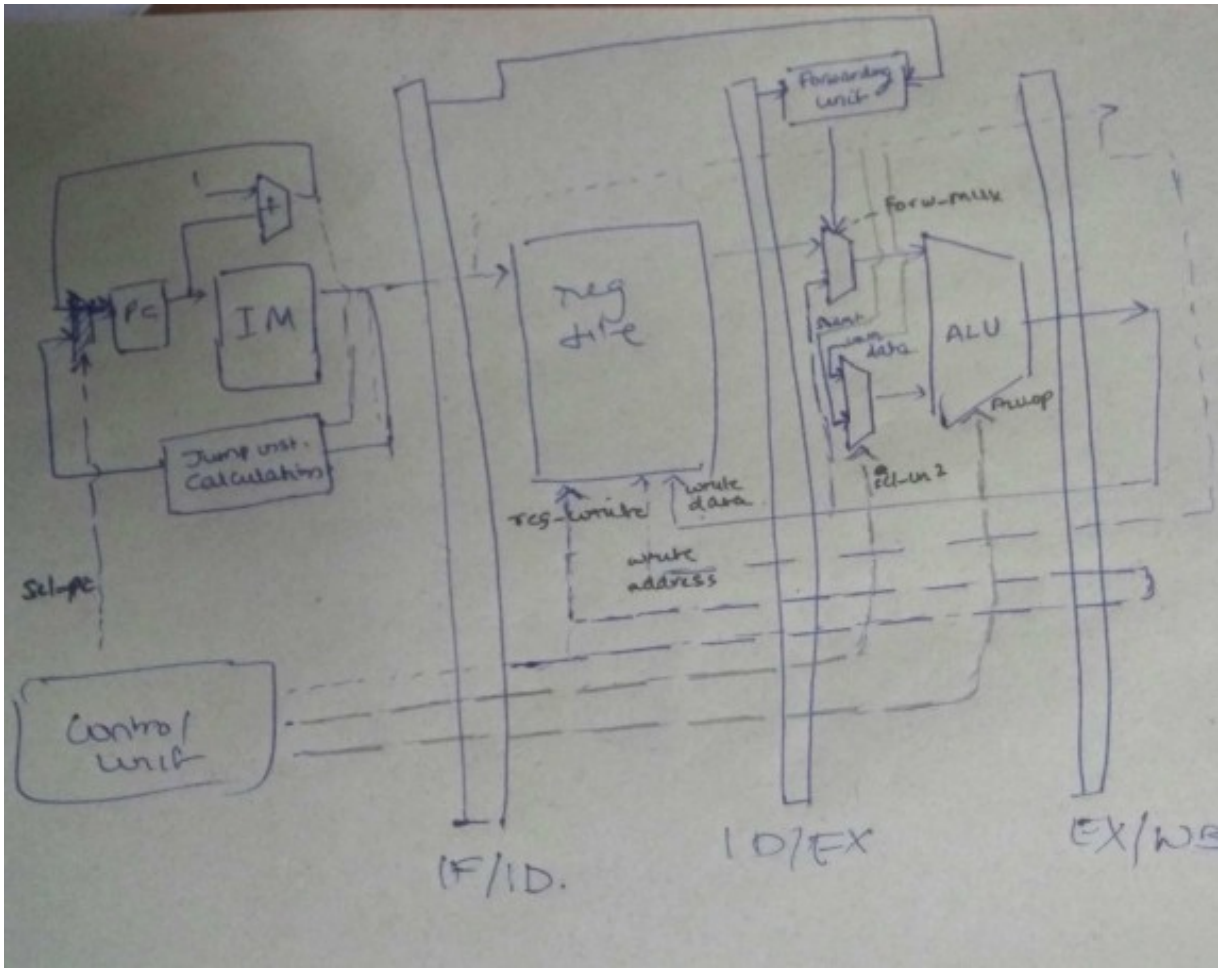
The due date for submission is 21-April-2019, 5:00 PM.

Name: shreyas ravishankar **ID No:** 2016AAPS0180H

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**					
	sel_pc	**aluc**	**reg_write**	**sel_in2**		
li	**0**	**0**	**1**	**0**		
sll	**0**	**1**	**1**	**1**		
j	**1**	**1**	**0**	**0**		

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

```
module instr_mem(PC,inst_out,rst);

  input rst;
  input [4:0]PC;
  output[7:0] inst_out;

  reg [7:0] I_m[31:0];

  always@(negedge rst)
    begin
    I_m[0]= 8'b0;
    I_m[1]= 8'b00001011;
    I_m[2]= 8'b01001001;
    I_m[3]= 8'b00010010;
    I_m[4]= 8'b11000001;
    I_m[5]= 8'b01010011;
    I_m[6]= 8'b00011100;
    I_m[7]= 8'h10;
    I_m[8]= 8'h20;
    I_m[9]= 8'h00;


    end

assign inst_out= {I_m[PC]};


endmodule
```

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

```
module reg_file(read2,read1,write_data,write_reg,write_addr,rst,reg1,reg2);

  // a regfile of #8 , 8 bit registers
  input[2:0] read1,read2,write_addr;  //write_addr is for the write address
  input[7:0] write_data; // the data to be written
  input write_reg;  // whether to write or not
  input rst;
  output reg [7:0] reg1,reg2;

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


  //decoding the address

  always@(*) // changed from assign statement to always. why did  it work?
  begin
    reg1= reg_mem[read1];           //reading
    reg2= reg_mem[read2];
    if(write_reg ==1'b1)           //writing
        reg_mem[write_addr]<= write_data;  //working only with non blocking statements

  end

  always@(posedge rst)
    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;

    end

endmodule
```

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

Answer: reg_ID_EX_instruction[5:3]== reg_IF_ID_instruction[5:3] . This covers all cases of forwarding ((li,li),(li,sll),(sll,li),(sll,sll)).

6. **Implement the forwarding unit and copy the image of Verilog code of forwarding unit here.**

Answer: The forwarding unit is not a seperate module , it has been implemented as part of the main processor module.

```
always@(*)
begin
        if((reg_ID_inst_out!=0) && (reg_IF_inst_out) && (reg_ID_inst_out[5:3]== reg_IF_inst_out[5:3]))
                flag=1;
        else
                flag=0;
end

always@(posedge clk)
begin
        if(flag==1)
                sel_forw=1;
        else
                sel_forw=0;
end

always@(*)              //forwarding_mux which is placed before alu input 1
begin
if(sel_forw==1)
        in1= reg_EX_alu_result;
else
        in1= reg_ID_reg1_data;
end
```

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

```
`include "alu.v"
`include "control.v"
`include "instr_mem.v"
`include "reg_file.v"
`include "registers2.v"

module parent2(clk,rst);

	input clk,rst;
	reg [4:0] PC, PC_in;
	wire[4:0] PC_plus;
	wire sel_pc, reg_ID_sel_in2, reg_EX_reg_write, aluc, reg_write, sel_in2, reg_ID_aluc;
	wire [7:0] reg_ID_inst_out, reg_IF_inst_out, reg_EX_alu_out, reg1,reg2, inst_out, reg_EX_alu_result, reg_EX_inst_out;
	wire[7:0] alu_result,reg_ID_reg1_data;
	reg[7:0] in2,in1;
	reg flag,sel_forw;

	assign PC_plus= PC+1;

	control cc(.inst_out(inst_out), .sel_pc(sel_pc), .aluc(aluc), .reg_write(reg_write),.sel_in2(sel_in2));

	instr_mem instr1(PC,inst_out,rst);

	reg_file reg_file1(.read1(reg_IF_inst_out[5:3]),.write_data(reg_EX_alu_result),.write_reg(reg_EX_reg_write),
			.write_addr(reg_EX_inst_out[5:3]), .rst(rst), .reg1(reg1));

	alu alu1(.in1(in1),.in2(in2),.aluc(reg_ID_aluc),.out(alu_result));


	registers2 ree(clk,rst,reg_write,sel_in2,aluc,inst_out,reg1,alu_result,                //registers file containing all the pipelined registers
			reg_IF_inst_out,reg_ID_sel_in2,reg_ID_aluc,reg_EX_reg_write,
			reg_EX_alu_result,reg_EX_inst_out,reg_ID_reg1_data,reg_ID_inst_out);




		always@(negedge rst)                    //pc updation
			PC<=0;
		always@(posedge clk)
			PC<= PC_in;

		always@(*)                               //PC mux , for jump instruction
		begin
			if(sel_pc== 1'b0)
				PC_in<= PC_plus;
			else
				PC_in<= PC_plus+{2'b00,inst_out[5:0]};
		end



		always@(*)                               //mux for sel_in2
		begin
			if(reg_ID_sel_in2==1'b0)
			begin
				if(reg_ID_inst_out[2]==1)
					in2= {5'b11111,reg_ID_inst_out[2:0]};
				else
					in2= {5'b0,reg_ID_inst_out[2:0]};

			end

			else
				in2=  reg_ID_inst_out[2:0];
		end
```

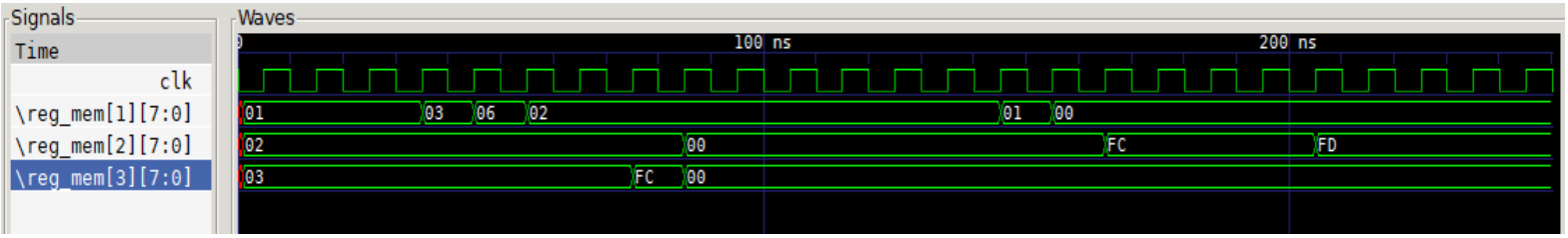
```
	//forwarding

	always@(*)		//checking conditions for forwarding
	begin
		if((reg_ID_inst_out!=0) && (reg_IF_inst_out) && (reg_ID_inst_out[5:3]== reg_IF_inst_out[5:3]))
			flag=1;
		else
			flag=0;
	end

	always@(posedge clk)
	begin
		if(flag==1)
			sel_forw=1;
		else
			sel_forw=0;
	end

	always@(*)		//forwarding_mux which is placed before alu input 1
	begin
	if(sel_forw==1)
		in1= reg_EX_alu_result;
	else
		in1= reg_ID_reg1_data;
	end

endmodule
```

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

```
`timescale 1ns/1ns
module parent_test();

	reg clk,rst;

parent2 par(clk,rst);


initial
begin

	clk=0;
	repeat(50)
	#5 clk= ~clk;

end

initial
begin
	$dumpfile("parent2.vcd");
	$dumpvars(0);
	rst=0;
	#1;
	rst=1;
	#1;
	rst=0;
	#200;
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):



Instructions that were implemented were-

```
        li R1, 3
        sll R1, 1
        li R2, 2
        j L1
        sll R2, 3
L1:     li R3, 4
```

Unrelated Questions

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

Answer: Trying to come up with a suitable datapath for the required set of instructions was a challenge. Moreover I had initially implemented the jump checking in the alu stage. This design was tough to debug and it probably required some stalls and nops. But then I changed my datapath so that jump checking happens in IF stage itself. This made the program easier to debug.

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 did the assignment completely 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: Shreyas Ravishankar **Date:** 23/4/19
ID No.: 2016AAPS0180H