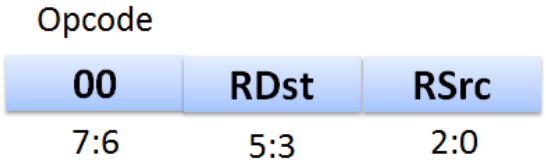
Birla Institute of Technology and Science – Pilani, Hyderabad Campus
Second Semester 2017-18
CS F342: Computer Architecture Assignment (20 Marks)

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

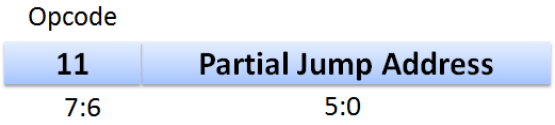


Example usage: mov R2, R0 (R2R0)

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 adding PC+1 to the Signextended data specified in instruction field (5:0). Opcode for j is 11)



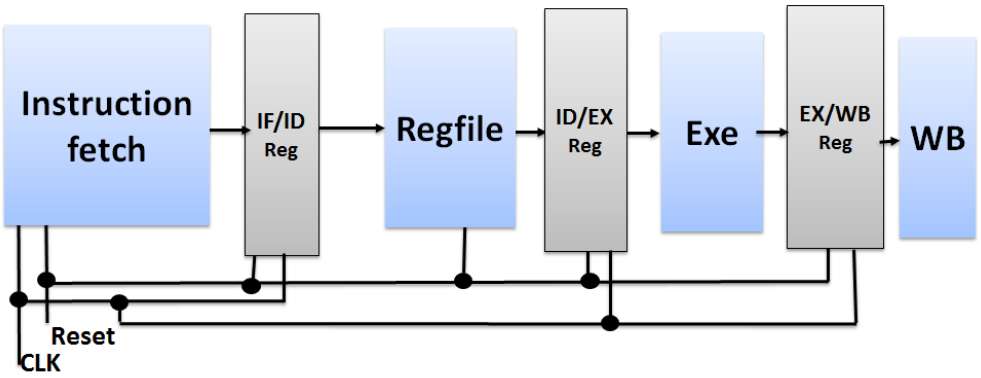
Example usage: j L1 (Jump address is calculated using PC relative addressing)

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
        sll Rx, 1
        mov Ry, Rx
        j L1
        sll Ry, 3
L1:     mov 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

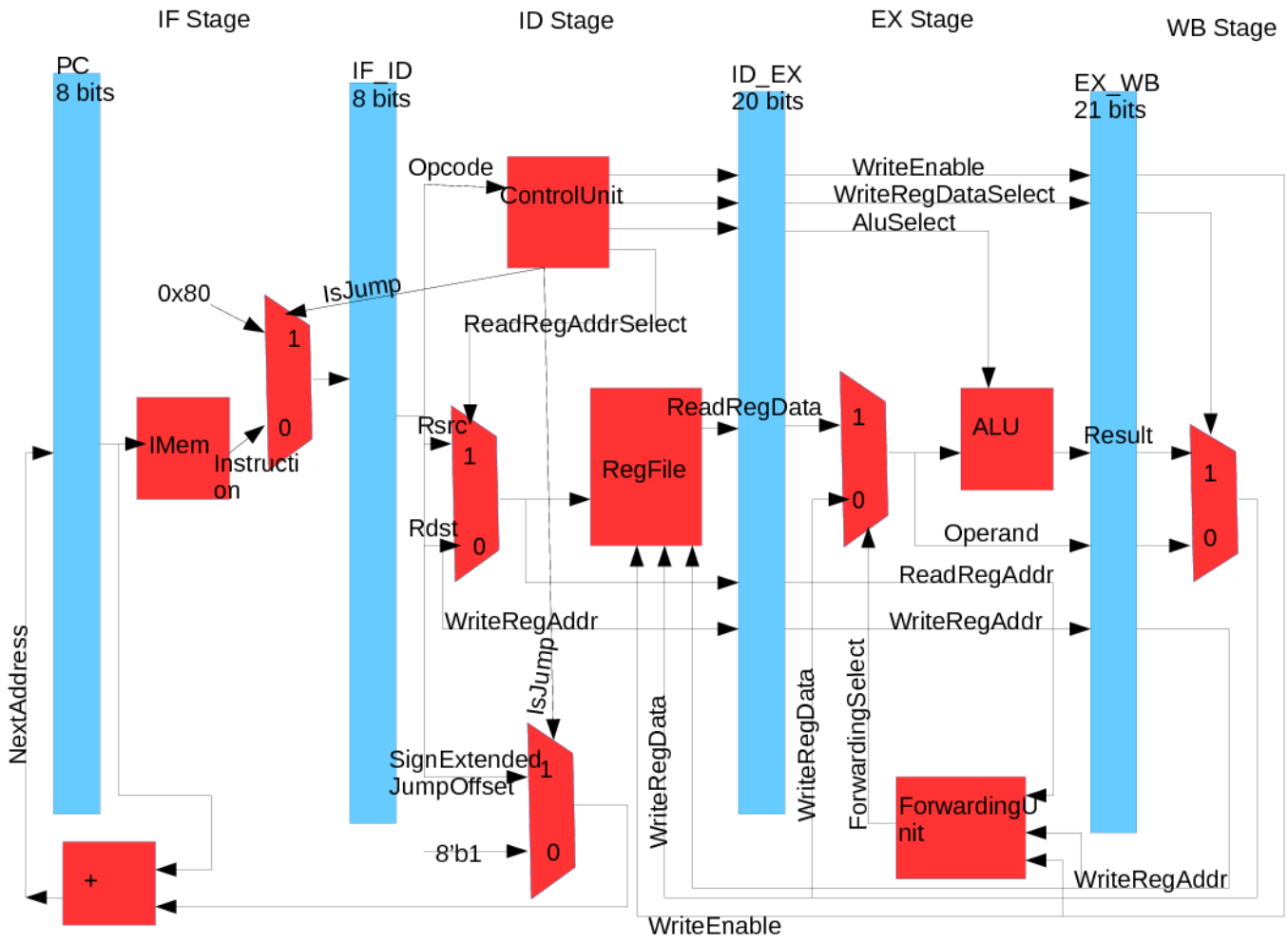
The due date for submission is 20-April-2018, 5:00 PM.

Name: Dvivedi Rohan Vipulkumar **ID No:** 2014A4TS0456H

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**				
	IsJump	**ReadRegAddrSelect**	**AluSelect**	**WriteRegDataSelect**	**WriteEnable**
mov	**0**	1	x	**0**	**1**
sll	**0**	0	1	**1**	**1**
j	**1**	x	x	**x**	**0**

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

Answer:

```
module IMem( input[7:0] Address, output reg [7:0] Instruction , input Reset );
reg [7:0] imem [0:256];

always @ ( * )
begin
        Instruction = imem[Address];
end

always @ ( Reset )
begin
        if( Reset )
        begin
                imem[0] <= 8'b00100111; //      mov R4,R7
                imem[1] <= 8'b01100001; //      sll R4,1
                imem[2] <= 8'b00111100; //      mov R7,R4
                imem[3] <= 8'b11000001; //      j L1
                imem[4] <= 8'b01111011; //      sll R7,3
                imem[5] <= 8'b00001111; //      L1: mov R1,R7
        end
end

endmodule
```

4. **Implement the Register File and copy the image of Verilog code of Register file unit here.**

Answer:

```
module RegFile  (
                input [2:0] ReadRegAddr,output reg [7:0] ReadRegData,
                input [2:0] WriteRegAddr,input reg [7:0] WriteRegData,
                input WriteEnable,
                input Clk,input Reset,
                output reg [7:0] Register [0:7]
                );
// All the 8 registers are provided with a port outside of module
// so that that they can be tapped to check for all possible changes
// 2 dimensional porting may not be available in certain tool chains

always @ ( * )
begin
        ReadRegData = Register[ReadRegAddr];    // reading data from RegFile
end

always @ ( * )
begin
        Register[WriteRegAddr] <= ( ( WriteEnable & Clk ) ? WriteRegData : Register[WriteRegAddr] );
        // Writing data to RegFile when required conditions are met
end

always @ ( * )
begin
        if( Reset )
        begin
                Register[0] <= 8'b00000000;
                Register[1] <= 8'b00000001;
                Register[2] <= 8'b00000010;
                Register[3] <= 8'b00000011;
                Register[4] <= 8'b00000100;
                Register[5] <= 8'b00000101;
                Register[6] <= 8'b00000110;
                Register[7] <= 8'b00000111;
        end
end

endmodule
```

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

Answer: (WriteEnable_WB == 1'b1) && (ReadRegAddr_EX == WriteRegAddr_WB)

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

Answer: Please note, The ForwardingUnit gives ForwardingSelect That is input to multiplexer that controls operands of the ALU unit.

```
module ForwardingUnit
        (
                input WriteEnable_WB,
                input [2:0] WriteRegAddr_WB,
                input [2:0] ReadRegAddr_EX,
                output reg ForwardingSelect
        );

// ForwardingSelect == 0 then WriteRegData_WB is used as operand for ALU
// else  ReadRegData_EX is used as operand for ALU

always @ ( * )
begin
        if( WriteRegAddr_WB == ReadRegAddr_EX && WriteEnable_WB )
        begin
                ForwardingSelect = 0;
        end
        else
        begin
                ForwardingSelect = 1;
        end
end

endmodule
```

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)**

Answer:

```
module Processor (input Reset,input Clk,output [7:0] Register [0:7]);

// The main Pipeline Registers
reg [7:0] PC;
reg [7:0] IF_ID;
reg [19:0] ID_EX;
reg [20:0] EX_WB;

//on reset we make all registers to value 0
always @ (posedge Reset)
begin
        PC    <= 0;
        IF_ID <= 0;
        ID_EX <= 0;
        EX_WB <= 0;
end
```

```
// All wire connections declarations
/*
       a Signal can persists in pipeline for multiple stages
              Signal_ID is the same signal in instruction decode stage
              Signal_EX is the signal in execute stage
              Signal_WB is the same signal in write back stage
*/
wire [7:0] NextAddress;         // Next Address Calculation
wire [7:0] Instruction;
wire [1:0] Opcode;
wire [2:0] Rsrc;
wire [2:0] Rdst;
wire [2:0] Shamt_ID,Shamt_EX;
wire [5:0] JumpOffset;
wire [7:0] JumpOffsetExtended;
wire [2:0] ReadRegAddr_ID,ReadRegAddr_EX;
wire [2:0] WriteRegAddr_ID,WriteRegAddr_EX,WriteRegAddr_WB;
wire [7:0] ReadRegData_ID,ReadRegData_EX;
wire [7:0] WriteRegData_WB;
wire [7:0] Operand_EX,Operand_WB;
wire [7:0] Result_EX,Result_WB;
wire WriteEnable_ID,WriteEnable_EX,WriteEnable_WB;
wire WriteRegDataSelect_ID,WriteRegDataSelect_EX,WriteRegDataSelect_WB;
wire AluSelect_ID,AluSelect_EX;
wire ReadRegAddrSelect_ID;
wire IsJump_ID;
wire ForwardingSelect;
```

```
/*
        Format ::::
        StageName stage start
                combinational logic relating to that stage only
        StageName stage end
*/

// IF stage start
        IMem IMEM (PC,Instruction,Reset);       // Only Instruction Memory is there in IF stage
// IF stage end

// ID stage start
        // Instruction to Instruction Fields
        assign {Opcode,Rdst,Rsrc} = IF_ID;
        assign Shamt_ID = Rsrc;
        assign JumpOffset = IF_ID[5:0];

        assign ReadRegAddr_ID = ((ReadRegAddrSelect_ID)?Rsrc:Rdst);             // Select register to read

        assign NextAddress = ((IsJump_ID)?JumpOffsetExtended:8'b1) + PC;        // Calculation of jump address for next instruction

        assign WriteRegAddr_ID = Rdst;          // address of writing register for writeback stage

        SignExtend SIGNEXTEND (JumpOffset,JumpOffsetExtended);  // signextend jump offset

        // 2 Important modules of processor below , ControlUnit and RegFile
        ControlUnit CONTROLUNIT (Opcode,IsJump_ID,ReadRegAddrSelect_ID,AluSelect_ID,WriteRegDataSelect_ID,WriteEnable_ID);
        RegFile REGFILE (ReadRegAddr_ID,ReadRegData_ID,WriteRegAddr_WB,WriteRegData_WB,WriteEnable_WB,Clk,Reset,Register);
// ID stage end
```

```
// EX stage start
        // Take Appropritate Signals From ID_EX stage for computation in EX stage
        assign {Shamt_EX,WriteRegAddr_EX,ReadRegAddr_EX,ReadRegData_EX,AluSelect_EX,WriteRegDataSelect_EX,WriteEnable_EX} = ID_EX;

        // The Operand of ALU is selected by the forwarding unit
        // we can use either data read from reg file or write data from writeback stage
        assign Operand_EX = ((ForwardingSelect)?ReadRegData_EX:WriteRegData_WB);

        // The main module of EX stage is ALU
        Alu ALU (Operand_EX,Shamt_EX,Result_EX,AluSelect_EX);

        // The Forwarding unit Assists EX stage to execute instruction with RAW,EX-EX data dependency
        ForwardingUnit FORWARDINGUNIT (WriteEnable_WB,WriteRegAddr_WB,ReadRegAddr_EX,ForwardingSelect);
// EX stage end

// WB stage start
        // Take Appropriate signals from EX_WB pipeline register for computation in Writeback stage
        assign {WriteRegAddr_WB,Operand_WB,Result_WB,WriteRegDataSelect_WB,WriteEnable_WB} = EX_WB;

        // The Register file declared above is given with Signals from Write back stage to write to register file
        // The Write data can be Result of ALU for sll instruction
        // or ALU operand for mov instruction, which is selected by WriteRegDataSelect_WB which was earlier generated by controlunit
        assign WriteRegData_WB = ((WriteRegDataSelect_WB)?Result_WB:Operand_WB);
// WB stage end

// Forward all pipeline registers with corresponding signals every clock
always @ (posedge Clk)
begin
  PC    <= {NextAddress};
  IF_ID <= ((IsJump_ID)?8'b10000000:Instruction); // We require a stall every jump, 0x80 is a NOP
  ID_EX <= {Shamt_ID,WriteRegAddr_ID,ReadRegAddr_ID,ReadRegData_ID,AluSelect_ID,WriteRegDataSelect_ID,WriteEnable_ID};
  EX_WB <= {WriteRegAddr_EX,Operand_EX,Result_EX,WriteRegDataSelect_EX,WriteEnable_EX};
end

endmodule
```

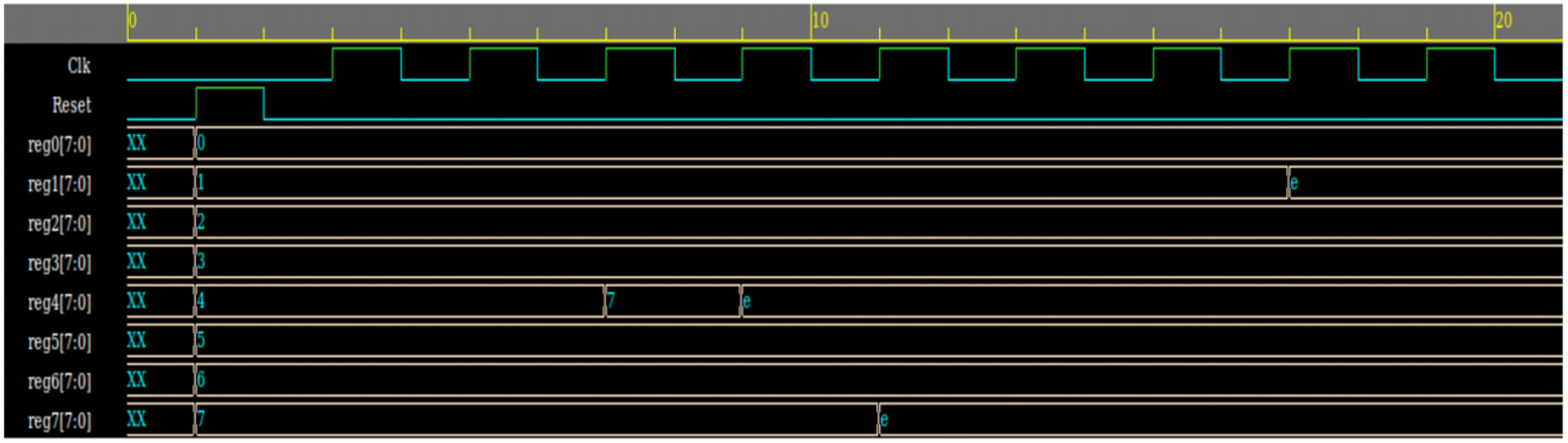
8. Test the processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.

Answer:

```
module TB();
  reg Clk,Reset;

  // Register 2 dimensional wire is used to tap into register values
  wire[7:0] Register[0:7];
  wire[7:0] reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7;
  assign reg0 = Register[0];
  assign reg1 = Register[1];
  assign reg2 = Register[2];
  assign reg3 = Register[3];
  assign reg4 = Register[4];
  assign reg5 = Register[5];
  assign reg6 = Register[6];
  assign reg7 = Register[7];

// two dimensional input output ports may not be available in certain toolchains
  Processor P (Reset,Clk,Register);

  initial begin
    // below two lines are specific to my simulator
    $dumpfile("dump.vcd");
    $dumpvars(1);

    Clk = 0;Reset = 0;
    #1;
    Clk = 0;Reset = 1;
    #1;
    Clk = 0;Reset = 0;
    #1;
    repeat(18)
      begin
        Clk = ~Clk;
        #1;
      end

  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: No Problems, Sir, I have used online compiler Simulator tool chain (and I am working on a Linux System) , available on https://www.edaplayground.com/x/5ZEZ , I have made my project available online.

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 have Implemented the complete design on my own. In case of syntax error, I have taken help from asicworld.com website.

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: Dvivedi Rohan Vipulkumar **Date: 22 April 2018**
ID No.: 2014A4TS0456H