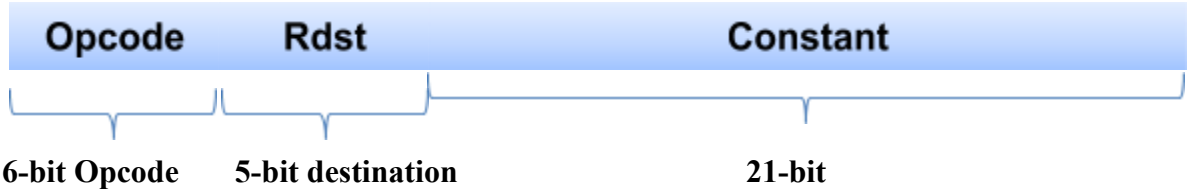
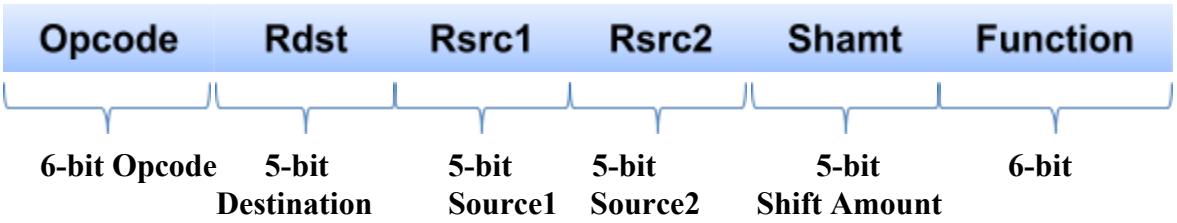
Assignment 1: Implementation of a MIPS like processor (10Marks)

Design and implement (in Verilog) datapath and control unit for a single cycle MIPS like processor (including instruction memory) which has two classes of instructions. The two classes of instructions along with the example usage and instruction decoding to be used are as below

1. **Immediate Type**
   Example: li r1, constant loads immediate signed value specified in the instruction to the register R1



2. **Register Type (R-type)**
   Example: add r1, r2, r3 adds the contents of registers r2 and r3. The result of addition is written in to the register r1



Assume there are 32 32-bit general purpose registers indicated by r0, r1, r2…r31 and corresponding register numbers (00000), (00001)……..(11111).
Assume the Opcode for Immediate type and R-type instructions as below

Instruction Class	Opcode
Immediate type	111111
Register Type	000000

Additionally R-type instructions have multiple variations defined by their function codes. The R-type instructions should include **add**, **sub**, **AND**, **OR**, **srl** (Shift right logical), **sll** (shift left logical) .The different R-type instructions that the processor should support are tabulated below.

R-type Instruction	Example usage	Opcode	Rdst	Rsrc1	Rsrc2	shamt	Function
add	**add r0, r1, r2**	**000000**	**00000**	**00001**	**00010**	**00000**	**100000**
sub	**sub r4, r5, r6**	**000000**	**00100**	**00101**	**00110**	**00000**	**100010**
AND	**and r8, r9, r10**	**000000**	**01000**	**01001**	**01010**	**00000**	**100100**
OR	**and r9, r8, r10**	**000000**	**01001**	**01000**	**01010**	**00000**	**100101**
sll	**sll r11, r6, 6**	**000000**	**01011**	**00110**	**00000***	**00110**	**000000**
srl	**srl r13, r9, 10**	**000000**	**01101**	**01001**	**00000***	**01010**	**000010**

*Second source is not used for shift operations

The processor module should have only two inputs CLK and Reset. When Reset is activated the Processor starts executing instructions from 0^{th} location of instruction memory.

As part of the assignment the following 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 (including control unit).

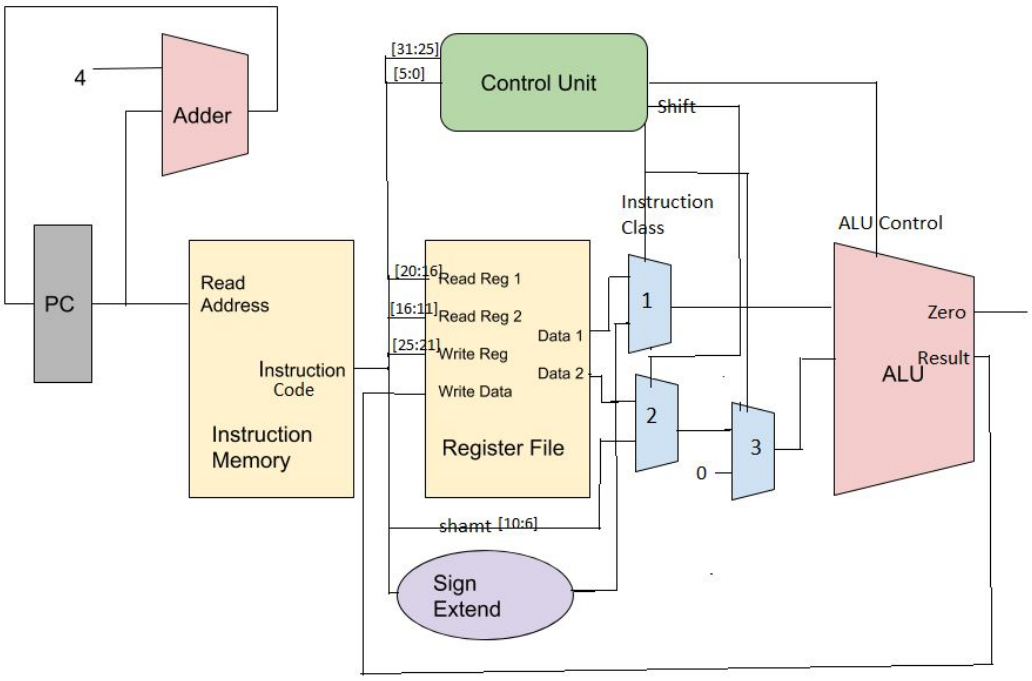
3. Design Verilog file for the main processor.

The name of the zipped folder should be in the format IDNO_NAME.zip

The due date for submission is 3-April-2019, 5:00 PM.

Q6.1. Draw the block level design of the processor (datapath + control unit) for above specifications. (you can modify the design given in the class ppts and copy the image of final design here)

Answer:



Q6.2. List the different blocks that will be required for implementation of datapath of the above processor.

Answer: Instruction Memory, Instruction Fetch, Register File, ALU, Sign Extended, Multiplexers

Q6.3. Most of the datapath blocks that are listed above have already been implemented as part of previous labs. Implement the blocks which have not been implemented in the previous labs and copy the images of those Verilog codes here.

```
module sign_extend(
    input [20:0] constant,
    output reg [31:0] extended_const
    );

    always@(*)
    begin
    extended_const <= { {11{constant[20]}}, constant[20:0]};
    end

endmodule
```

Answer:

```
module mux_2to1(
    input [31:0] A,
    input [31:0] B,
    input select,
    output [31:0] out
    );

    assign out = select ? A:B;

endmodule
```

Q6.4. Assume Main control unit generates all the control signals. List different control signals that will be required for the above processor. Also specify the value of the control signals for different instructions.

Answer:

Control Signal Name	instruction_class	reg_write	shift	alu_control
li r1, 8	1	1	X	0010
add r0, r1, r2	0	1	0	0010

sub r4, r5, r6	0	1	0	0100
and r8, r9, r10	0	1	0	0000
and r9, r8, r10	0	1	0	0000
sll r11, r6, 6	0	1	1	1000
srl r13, r9, 10	0	1	1	1001

Q6.5. Implement the main control unit and copy the <u>image</u> of Verilog code of Main control unit here.

Answer:

```
module control_unit(
    input [5:0] opcode,
    input [5:0] funct,
    output instruction_class,
    output reg_write,
    output shift,
    output reg [3:0] alu_control
    );

    assign instruction_class = (opcode == 6'b000000) ? 0:1;
    assign reg_write = 1;
    assign shift = (funct[6:3] == 4'b0000) ? 1:0;

    always@(*)
    begin
        if(opcode == 6'b000000)
        begin
            case(funct)
                6'b100000:
                begin
                alu_control <= 4'b0010;
                end
                6'b100010:
                begin
                alu_control <= 4'b0100;
                end
                6'b100100:
                begin
                alu_control <= 4'b0000;
                end
                6'b100101:
                begin
                alu_control <= 4'b0001;
                end
                6'b000000:
                begin
                alu_control <= 4'b1000;
                end
                6'b000010:
                begin
                alu_control <= 4'b1001;
                end
            endcase
        end
        else
        begin
            alu_control <= 4'b0010;
        end
    end

endmodule
```

Q6.6. Implement complete processor in Verilog (Instantiate all the datapath blocks and main control unit as modules). Copy the <u>image</u> of Verilog code of the processor here.

```
module datapath(
    input clk,
    input rst
    );

    wire [31:0] instruction_code;
    wire [5:0] opcode;
    wire [4:0] read_reg_num1;
    wire [4:0] read_reg_num2;
    wire [4:0] write_reg_num;
    wire [4:0] shamt;
    wire [5:0] funct;
    wire [20:0] constant;
    wire instruction_class;
    wire shift;
    wire zero;
    wire [3:0] alu_control;
    wire [31:0] alu_result;
    wire [31:0] read_data1;
    wire [31:0] read_data2;
    wire [31:0] const_extended, shamt_extended;
    wire [31:0] mux1_out, mux2_out, mux3_out;

    parameter z = 0;

    assign opcode = instruction_code[31:26];
    assign write_reg_num = instruction_code[25:21];
    assign read_reg_num1 = instruction_code[20:16];
    assign read_reg_num2 = instruction_code[15:11];
    assign shamt = instruction_code[10:6];
    assign funct = instruction_code[5:0];
    assign constant = instruction_code[20:0];
    assign shamt_extended = {27'd0, shamt};

    instruction_fetch IF(
        clk,
        rst,
        instruction_code
        );

    control_unit CTRL(
        opcode,
        funct,
        instruction_class,
        reg_write,
        shift,
        alu_control
        );

    register_file RF(
        read_reg_num1,
        read_reg_num2,
        write_reg_num,
        alu_result,
        read_data1,
        read_data2,
        reg_write,
        clk,
        rst
        );
```

Answer:

```
    sign_extend SE(
        constant,
        const_extended
        );

    mux_2to1 MUX1(
        const_extended,
        read_data1,
        instruction_class,
        mux1_out
        );

    mux_2to1 MUX2(
        shamt_extended,
        read_data2,
        shift,
        mux2_out
        );

    mux_2to1 MUX3(
        z,
        mux2_out,
        instruction_class,
        mux3_out
        );

    alu ALU(
        mux1_out,
        mux3_out,
        alu_control,
        zero,
        alu_result
        );

endmodule
```

Q6.7. Test the processor design by initializing the instruction memory with a set of instructions (at least 5 instructions). List below the instructions you have used to initialize the instruction memory. Verify if the register file is changing according to the instructions. (Register file contains unknowns, you can initialize the register file or you can load values into the register file using li instruction specified earlier).

Sequence of Instructions Implemented:

li r1, 8

add r0, r1, r2

sub r4, r5, r6

and r8, r9, r10

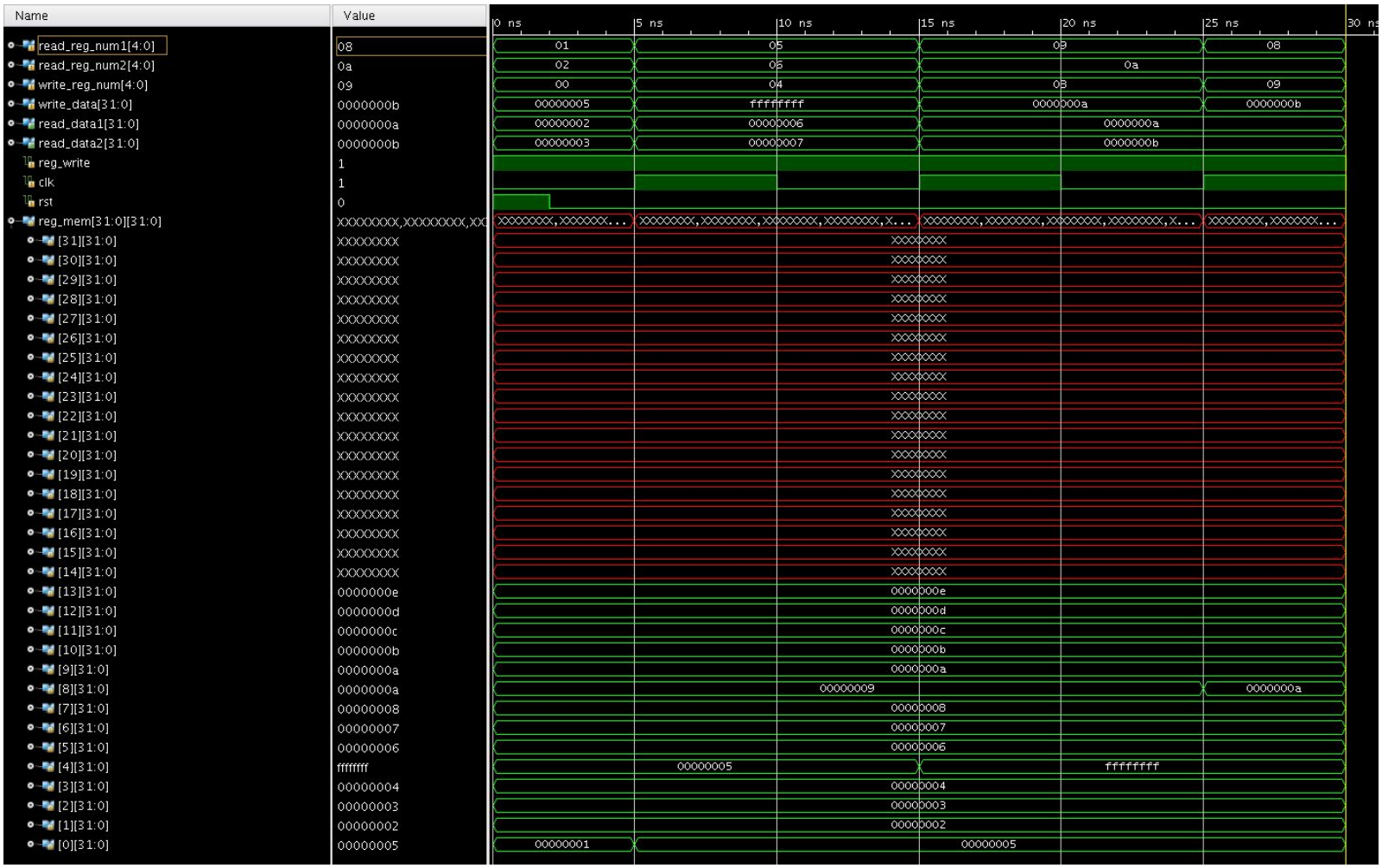
and r9, r8, r10

sll r11, r6, 6

srl r13, r9, 10

Q6.8. Verify if the register file is getting updated according to your sequence 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: The main problem I faced was the implementation of the control unit. Especially the way the control signals are assigned. It also took me a while to figure out the way to implement muxes properly in order to cover all cases of instructions.

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 the processor on my own. Although I did take help from my friend to understand the control signals.

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: Rohan Tammara **Date:** 3/4/19
ID No.: 2016A8PS0332H