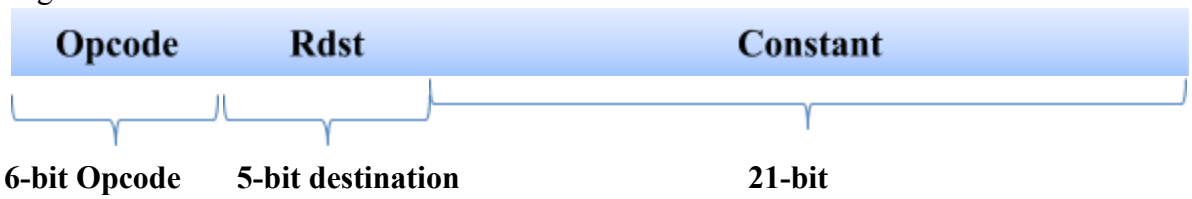
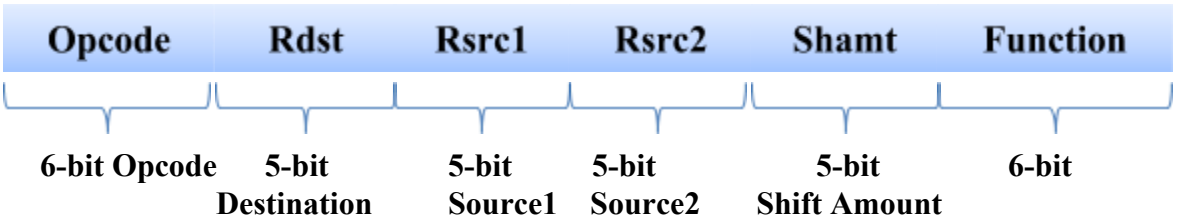
Assignment 1: Implementation of a MIPS like processor (10Marks)

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).

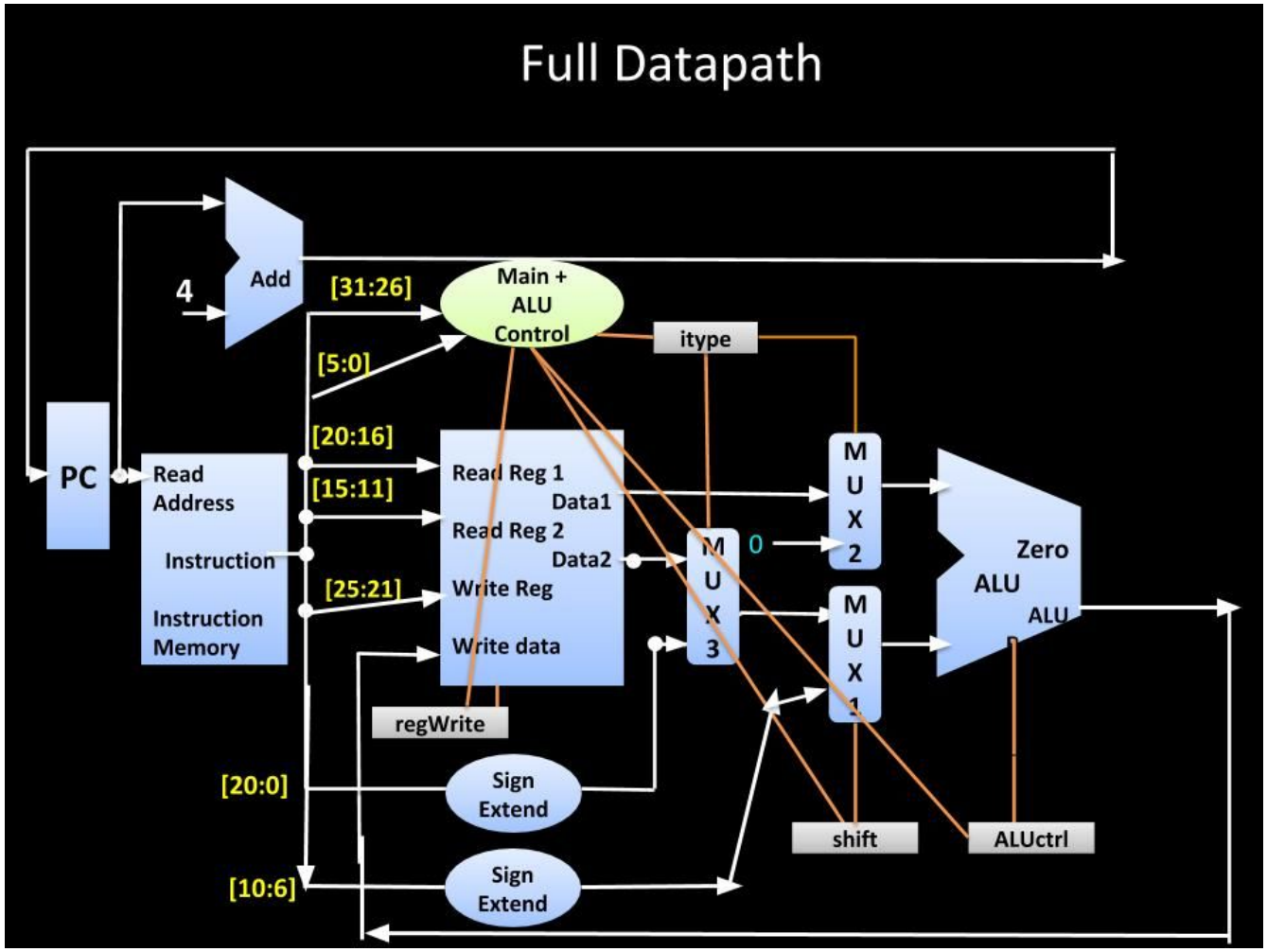
3. Design Verilog file for the main processor.

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:

The different blocks used in the implementation of the datapath are:

1. Instruction fetch block
    - Instruction memory block
2. Control Block
3. Register File
4. ALU
5. MUX_1
6. MUX_2

7. MUX_3

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.

Answer: The elements which were not implemented in the lab are :

1. The control signals for immediate operations - shown in control module
2. The immediate operation in ALU - control signal for which is shown in control_module and
3. Shifting operation in ALU - shown in mux_1 module
4. Control signal for shifting operation - shown in control_module

```
21  module control_module(
22      input [5:0] opcode,
23      input [5:0] funct,
24      output regWrite,
25      output itype,
26      output shift,
27      output reg [2:0] ALUctrl
28      );
29
30      assign itype = (opcode==6'b000000)?1'b0:1'b1;
31      assign regWrite = 1'b1;
32      assign shift = ((funct == 6'b000000) | (funct == 6'b000010))?1:0;
33
34      // ALU control :
35      always@(*)
36      begin
37
38       if(opcode == 6'b111111)
39           ALUctrl <= 3'b000;
40       else
41       begin
42           case(funct)
43              6'b100000 : ALUctrl <= 3'b000;   // ADD
44              6'b100010 : ALUctrl <= 3'b001;   //SUB
45              6'b100100 : ALUctrl <= 3'b010;   //AND
46              6'b100101 : ALUctrl <= 3'b011;   //OR
47              6'b000000 : ALUctrl <= 3'b100;   //sll
48              6'b000010 : ALUctrl <= 3'b101;   //srl
49           endcase
50        end
51      end
52  endmodule
```

```
module ALU_main(
    input [31:0] data1,
    input [31:0] data2,
    input [2:0] ALUctrl,
    output reg [31:0] alu_result,
    output reg zero
    );

    always@(*)
    begin
      case(ALUctrl)
         3'b000: alu_result = data1 + data2;
         3'b001: alu_result = data1 - data2;
         3'b010: alu_result = data1 & data2;
         3'b011: alu_result = data1 | data2;
         3'b100: alu_result = data1 << data2;
         3'b101: alu_result = data1 >> data2;
      endcase
      if(alu_result == 0)
         zero = 1'b1;
      else
         zero = 1'b0;
    end



endmodule
```

```
module mux_1(
    input [31:0] mux_3_out,
    input [4:0] shamt,
    input shift,
    output [31:0] mux_1_out
    );

    reg [31:0] sign_extended_shamt;

    always@(*)
    begin
      sign_extended_shamt[31:0] <= { {27{shamt[4]}} , shamt[4:0]};
    end

    assign mux_1_out = (shift == 1)?sign_extended_shamt:mux_3_out;


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	**RegWrite**	**itype**	**ALUctrl**	**shift**	
li r1, 8	1	1	000	0	
add r0, r1, r2	1	0	000	0	
sub r4, r5, r6	1	0	001	0	
and r8, r9, r10	1	0	010	0	
and r9, r8, r10	1	0	010	0	
sll r11, r6, 6	1	0	100	1	
srl r13, r9, 10	1	0	101	1	

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

Answer:

```
21  module control_module(
22      input [5:0] opcode,
23      input [5:0] funct,
24      output regWrite,
25      output itype,
26      output shift,
27      output reg [2:0] ALUctrl
28      );
29
30      assign itype = (opcode==6'b000000)?1'b0:1'b1;
31      assign regWrite = 1'b1;
32      assign shift = ((funct == 6'b000000) | (funct == 6'b000010))?1:0;
33
34      // ALU control :
35      always@(*)
36      begin
37
38       if(opcode == 6'b111111)
39           ALUctrl <= 3'b000;
40       else
41       begin
42           case(funct)
43              6'b100000 : ALUctrl <= 3'b000;   // ADD
44              6'b100010 : ALUctrl <= 3'b001;   //SUB
45              6'b100100 : ALUctrl <= 3'b010;   //AND
46              6'b100101 : ALUctrl <= 3'b011;   //OR
47              6'b000000 : ALUctrl <= 3'b100;   //sll
48              6'b000010 : ALUctrl <= 3'b101;   //srl
49           endcase
50        end
51      end
52  endmodule
```

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

Answer:

```
21  module main(
22      input clk,
23      input rst
24      );
25
26      wire [31:0] instruction_code, mux_2_out, mux_3_out, data1, data2, alu_result, data1_final, data2_final, mux_1_out;
27      wire [4:0] read_reg_1, read_reg_2, write_reg;
28      wire itype, regWrite, zero, shift;
29      wire [2:0] ALUctrl;
30      wire [20:0] constant;
31      wire [5:0] opcode, funct;
32      wire [4:0] rs, rt, rd, shamt;
33
34      instr_fetch instr_f(
35      .clk(clk),
36      .rst(rst),
37      .instruction_code(instruction_code)
38      );
39
40        assign opcode = instruction_code[31:26];
41        assign rd = instruction_code[25:21];
42        assign rs = instruction_code[20:16];
43        assign rt = instruction_code[15:11];
44        assign shamt = instruction_code[10:6];
45        assign funct = instruction_code[5:0];
46        assign constant = instruction_code[20:0];
47
```

```
48      control_module ctrl_mod(
49      opcode,
50      funct,
51      regWrite,
52      itype,
53      shift,
54      ALUctrl
55      );
56
57      register_file reg_file(
58      .read_reg_1(rs),
59      .read_reg_2(rt),
60      .write_reg(rd),
61      .write_data(alu_result),
62      .regWrite(regWrite),
63      .rst(rst),
64      .clk(clk),
65      .data1(data1),
66      .data2(data2)
67      );
68
69      mux_2 m2(
70      .data1(data1),
71      .itype(itype),
72      .mux_2_out(data1_final)
73      );
74
75      mux_3 m3(
76      .data2(data2),
77      .constant(constant),
78      .itype(itype),
79      .mux_3_out(data2_final)
80      );
```

```
81
82      mux_1 m1(
83      .mux_3_out(data2_final),
84      .shamt(shamt),
85      .shift(shift),
86      .mux_1_out(mux_1_out)
87      );
88
89      ALU_main alu(
90      data1_final,
91      mux_1_out,
92      ALUctrl,
93      alu_result,
94      zero
95      );
96
97
98
99  endmodule
100
```

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:

1. add r0, r0, r0
2. sub r2, r2, r1
3. and r3, r3, r5
4. sll r5, r5, 2
5. li r6, 2

Initial contents:

r0 = 1, r2 = 3, r3 = 4, r5 = 6, r6 = 7

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: "FATAL ERROR" showed up a few times because of the presence of blocking statements inside a sequential block of code. It was resolved by converting these statements to non-blocking.

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: Help was taken from Mr. Kunal Gulati to solve the "FATAL ERROR" case. Everything else was self implemented.

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: Ravi Bharadwaj C **Date:** 31/03/2019
ID No.: 2016AAPS0244H