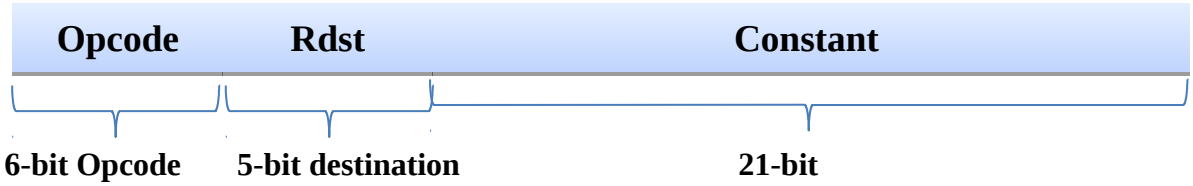
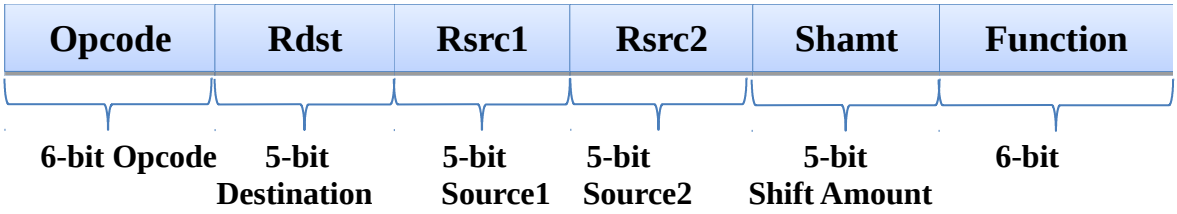
Assignment 1: Implementation of a MIPS like processor (16Marks)

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.

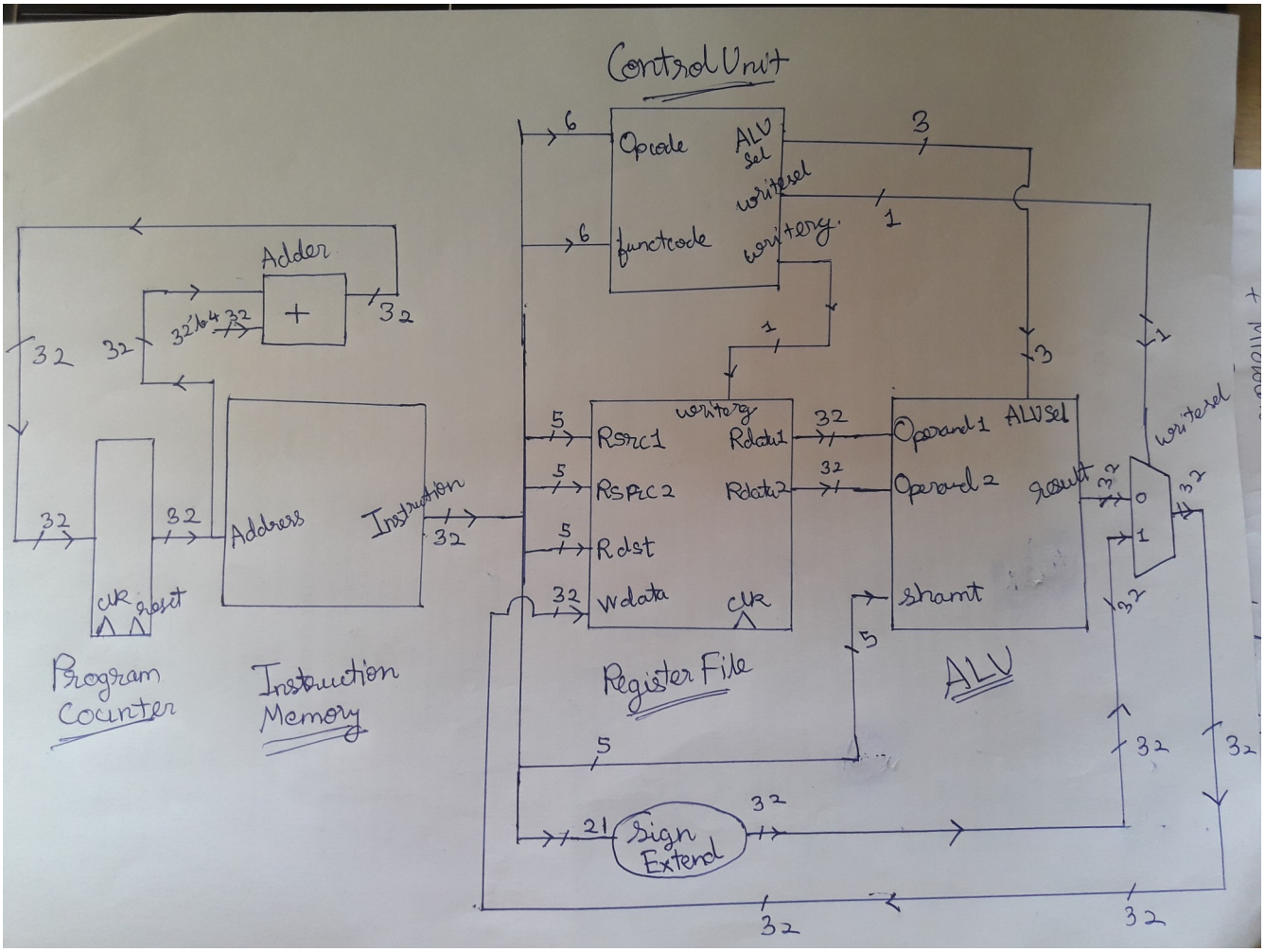
1. PDF version of this Document with all the Questions below answered with file name as **IDNO_NAME.pdf**.
2. Design Verilog Files for all the Sub-modules (including control unit).
3. Design Verilog file for the main processor.

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

The due date for submission is 30-March-2018, 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:

control unit

Instruction Memory Module

Sign Extend Module

Register File

ALU

Multiplexer to select write back data

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:

Sign Extend Module

```
module signextend(input[20:0] constantIn,output reg[31:0] constantOut);

always @ ( * )
begin
  constantOut = { {11{constantIn[20]}} , constantIn };
end

endmodule
```

Instruction Memory Module

```
module Imem(input[31:0] Address,output reg[31:0] Instruction);

  reg[31:0] memory[0:(((32'b1)<<10)-1)];

always @ ( * )
begin
        Instruction = memory[Address>>2];
end

// The test machine code that I am using
initial
begin
  memory[0] <= 32'b11111100000000000000000000000011;
  memory[1] <= 32'b11111100001000000000000000000101;
  memory[2] <= 32'b11111100010000000000000000000110;
  memory[3] <= 32'b00000000011000000000100000100100;
  memory[4] <= 32'b00000000010000000000100000100101;
  memory[5] <= 32'b00000000010100001000100000100000;
  memory[6] <= 32'b00000000010100101000000001000000;
  memory[7] <= 32'b00000000010100101000000000100000;
end

endmodule
```

Register File

```
module regfile( input[4:0] Rdst,input[4:0] Rsrc1,input[4:0] Rsrc2,
                input[31:0] wdata,output reg [31:0] Rdata1,output reg [31:0] Rdata2,
                input writereg,input clk,
                output reg[31:0] registers[0:31] );

always @ (posedge clk)
begin
  registers[Rdst] <= ( writereg ? wdata : registers[Rdst] );
end

always @ ( * )
begin
        Rdata1 = registers[Rsrc1];
        Rdata2 = registers[Rsrc2];
end


// I am Initializing all registers to 0 on startup
  integer i=0;
  initial
  begin
    repeat(32)
      begin
        registers[i] <= 0;
        i=i+1;
      end
  end

endmodule
```

ALU

```
module alu(input[31:0] operand1,input[31:0] operand2,input[4:0] shamt,input[2:0] select,output reg[31:0] result );

always @ ( * )
begin
        case (select)
        3'b000 :        result = operand1 + operand2;
        3'b001 :        result = operand1 - operand2;
        3'b010 :        result = operand1 & operand2;
        3'b011 :        result = operand1 | operand2;
        3'b100 :        result = operand1 << shamt;
        3'b101 :        result = operand1 >> shamt;
        default:        result = 32'bz;
        endcase
end

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	Write sel	alusel[2]	alusel[1]	alusel[0]	writereg
li r1, 8	1	x	x	x	1
add r0, r1, r2	0	0	0	0	1
sub r4, r5, r6	0	0	0	1	1
and r8, r9, r10	**0**	**0**	**1**	**0**	**1**
and r9, r8, r10	**0**	**0**	**1**	**0**	**1**
sll r11, r6, 6	**0**	1	**0**	**0**	**1**
srl r13, r9, 10	**0**	**1**	**0**	**1**	**1**

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

Answer:

```
module controlunit(input[5:0] opcode,input[5:0] functcode,
output reg[2:0] alusel,output reg writesel,output reg writereg);

always @ ( * )
begin
        case (opcode)
                6'b000000:
                begin
                        writereg <= 1'b1;
                        writesel <= 1'b0;
                        case (functcode)
                        6'b100000: alusel <= 3'b000;// add
                        6'b100010: alusel <= 3'b001;// sub
                        6'b100100: alusel <= 3'b010;// and
                        6'b100101: alusel <= 3'b011;// or
                        6'b000000: alusel <= 3'b100;// sll
                        6'b000010: alusel <= 3'b101;// srl
                        default:   alusel <= 3'bxxx;
            endcase
                end
                6'b111111:
                begin
                        writereg <= 1'b1;
                        writesel <= 1'b1;
                        alusel   <= 3'bxxx;
                end
                default :
                begin
                        writereg <= 1'b0;
                        writesel <= 1'bx;
                        alusel   <= 3'bxxx;
                end
        endcase
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.

Answer:

```
module processor(input clk,input reset,

                  // port to processor registers
                  output wire[31:0] register[0:31],

                  // programcounter register
                 output reg[31:0] programcounter

                 );



wire[31:0] Instruction;

// Instruction fields
wire[5:0] opcode;
wire[4:0] Rdst;
wire[4:0] Rsrc1;
wire[4:0] Rsrc2;
wire[4:0] shamt;
wire[5:0] functcode;

// SignExtend module connections
wire[20:0] constantIn;
wire[31:0] constantOut;

// ALU wire connections
wire[31:0] Operand1;
wire[31:0] Operand2;
wire[31:0] result;

wire[31:0] wdata;

// control signals from control unit
wire[2:0] alusel;
wire writereg;
wire writesel;
```

```
// Instruction fetch
Imem InstructionMemory (programcounter,Instruction);

// decode instruction in to fields
assign {opcode,Rdst,Rsrc1,Rsrc2,shamt,functcode} = Instruction;
assign constantIn = Instruction[20:0];

// signextend the constant value for li
signextend SignExtend (constantIn,constantOut);

// generate appropriate control signals for ALU and register file
controlunit ControlUnit (opcode,functcode,alusel,writesel,writereg);

// register file to store temporary data and retrieve operands
  regfile RegisterFile  (Rdst,Rsrc1,Rsrc2,wdata,Operand1,Operand2,writereg,clk,register);

// ALU for R type instructions namely, +,-,&,|,<<,>>
alu ALU (Operand1,Operand2,shamt,alusel,result);

// multiplexer to select what data to write back
assign wdata = (writesel ? constantOut : result );

  always @ ( posedge clk or negedge reset )
begin
  if( clk )
        begin
                programcounter <= programcounter + 32'b100;
        end
        else
        begin
                programcounter <= 32'b0;
        end
end

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:

In assembly code

li r0,3

li r1,5

li r2,6

and r3,r0,r1

or r4,r0,r1

add r5,r1,r2

sll r5,r5,1

add r5,r5,r0

In machine code

111111 00000 00000000000000000011

111111 00001 00000000000000000101

111111 00010 00000000000000000110

000000 00011 00000 00001 00000 100100

000000 00100 00000 00001 00000 100101

000000 00101 00001 00010 00000 100000

000000 00101 00101 00000 00001 000000

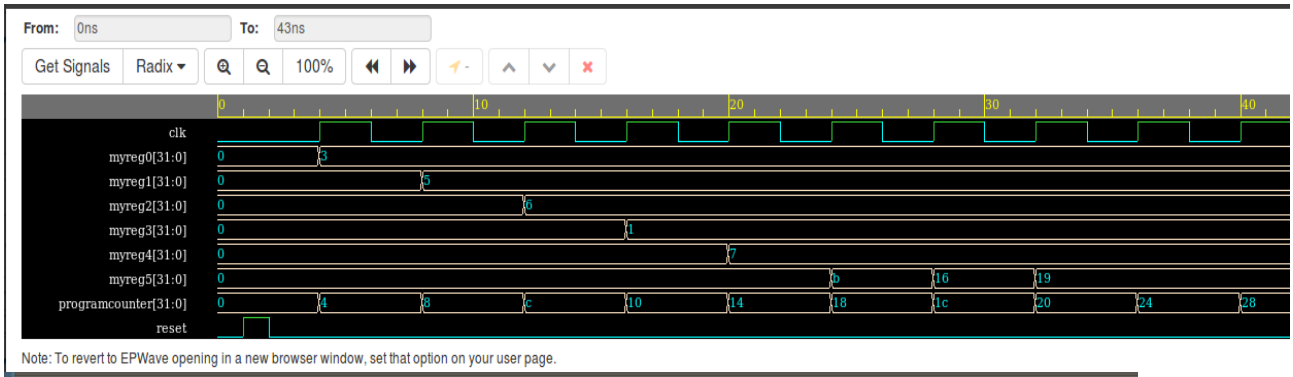
000000 00101 00101 00000 00000 100000

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):

I used a different compiler/simulator toolkit (Provided online by EDA Playground **https://www.edaplayground.com**), since I am working on an Linux ubuntu 16.04 distros.

Also the code used for Testbench of processor is given below.




```
module testbench;
  reg clk;
  reg reset;

  wire[31:0] register[0:31];
  wire[31:0] myreg0;  assign myreg0 = register[0];
  wire[31:0] myreg1;  assign myreg1 = register[1];
  wire[31:0] myreg2;  assign myreg2 = register[2];
  wire[31:0] myreg3;  assign myreg3 = register[3];
  wire[31:0] myreg4;  assign myreg4 = register[4];
  wire[31:0] myreg5;  assign myreg5 = register[5];

  wire[31:0] programcounter;

  processor processorP (clk,reset,register,programcounter);

initial
begin
    // Dump waves
     $dumpfile("dump.vcd");
     $dumpvars(1);
  reset=0;clk=0;
        #1;
  reset=1;clk=0;
        #1;
  reset = 0;
    #1;
  repeat(20)
    begin
        #1;
        clk=~clk;
        #1;
    end

        $finish;
end

endmodule
```

Unrelated Questions

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

Answer: No, problems Sir, except the availability of simulation software, But It's okay since there are many online simulation softwares available.

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 Sir, I implemented the processor on my own, I used ASICworld.com website for referencing syntax.

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: **Date: 29th March 2018**
ID No.: 2014A4TS0456H