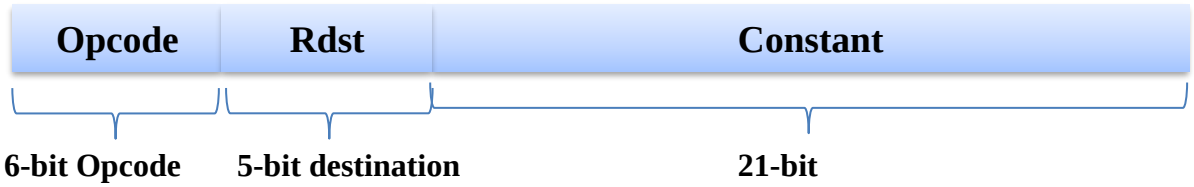
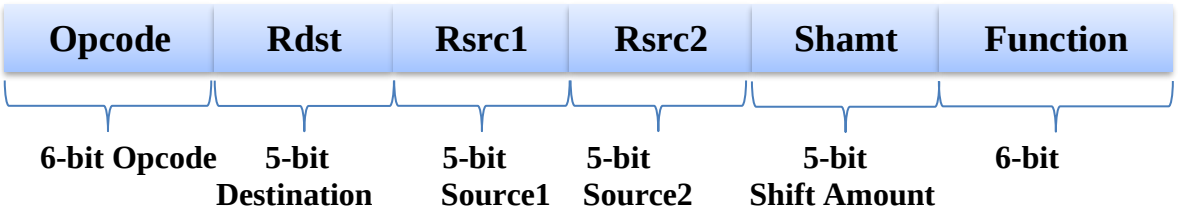
Assignment 1: Implementation of a processor (10Marks)

Design and implement (in Verilog) datapath and control unit for a single cycle 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).

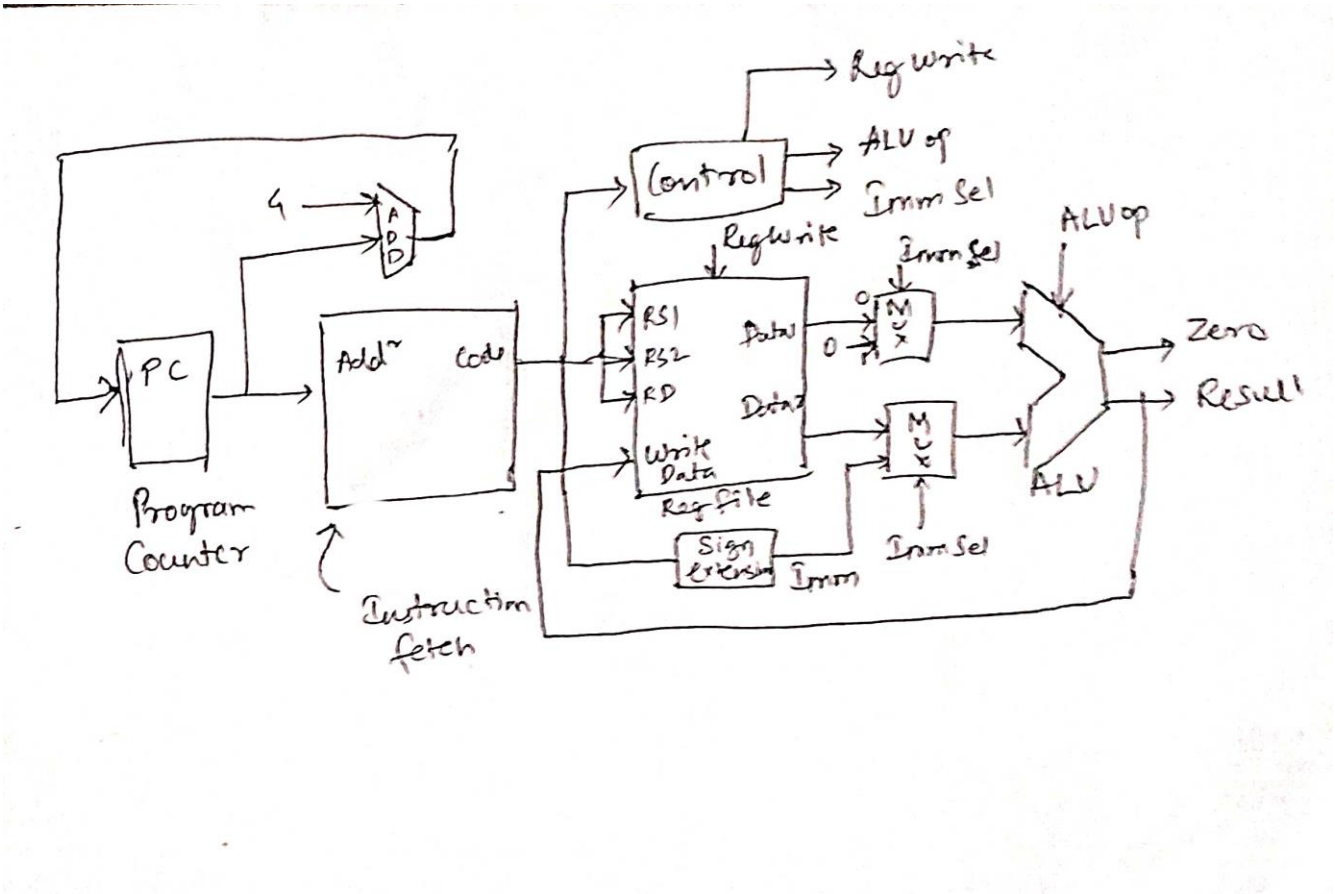
3. Design Verilog file for the main processor.

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

The due date for submission is 27-March-2022, 5:00 PM.

Q1.1. 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:



Q1.2. List the different blocks that will be required for implementation of datapath of the above processor.

Answer: Instruction Fetch, Instruction Memory, Register File, Arithmetic Logic Unit, Control Path

Q1.3. 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 IF(
    input clk,
    input reset,
    output [31:0] Instr
    );

reg [31:0] PC;
IMem M1(PC,reset,Instr);

always@(posedge clk,posedge reset)
begin
   if(reset) PC <= 32'b0;
   else PC <= PC+4;
end

endmodule
```

Answer:

```
module IMem(
    input [31:0] Addr,
    input reset,
    output reg [31:0] InstrCode
    );

reg [7:0] Mem [35:0];

always@(posedge reset)
begin
   if(reset)
   begin
      $readmemh("code.mem",Mem);
   end
end

always @(Addr) begin
   InstrCode = {Mem[Addr+3],Mem[Addr+2],Mem[Addr+1],Mem[Addr]};
   end

endmodule
```

```
module Register_File(
    input [4:0] Read_Reg_Num1,
    input [4:0] Read_Reg_Num2,
    input [4:0] Write_Reg_Num,
    input [31:0] Write_Data,
    output reg [31:0] Read_Data1,
    output reg [31:0] Read_Data2,
    input Reg_Write,
    input Clk,
    input Reset
    );
   reg [31:0] Reg_files [31:0];
   integer i;


   always @(posedge Clk, posedge Reset)
      begin
         Read_Data1 = Reg_files[Read_Reg_Num1];
         Read_Data2 = Reg_files[Read_Reg_Num2];
         if(Reset)
            begin
               for(i=0;i<=31;i=i+1)
               Reg_files[i]=i+1;
            end
         else if(Reg_Write)
            begin
               Reg_files[Write_Reg_Num] <= Write_Data;
            end
      end

endmodule
```

```
//////////////////////////////////////////////////////////////
module Control_Unit(
    input [31:0] Instruction_Code,
    output reg [3:0] ALU_op,
    output reg imm_signal,
    output reg Reg_Write
    );
    wire [5:0] funct6;
    assign funct6 = Instruction_Code[5:0];
    always @(*) begin
       //Reg_Write = 0;
       Reg_Write = 1'b1;
       imm_signal = 1'b0;
       if(Instruction_Code[31:26] == 6'b000000)
         begin
             case(funct6)
                6'b100000: ALU_op = 4'b0010;/*add*/
                6'b100010: ALU_op = 4'b0110;/*sub*/
                6'b100100: ALU_op = 4'b0000;/*AND*/
                6'b100101: ALU_op = 4'b0001;/*OR*/
                6'b000000: ALU_op = 4'b1110;/*SLL*/
                6'b000010: ALU_op = 4'b1100;/*SRL*/
             endcase

       end
       if(Instruction_Code[31:26] == 6'b111111)
         begin
            ALU_op = 4'b0010;
            imm_signal = 1'b1;
         end
    end
endmodule
```

```
module ALU(
    input [31:0] A,
    input [31:0] B,
    input [3:0] ALU_Control,
    input imm_signal,
    input [20:0] imm,
    output reg[31:0] ALU_Result,
    output reg Zero
    );
   wire [4:0] shift_amt;
   assign shift_amt = imm[10:6];

   always @(A,B,ALU_Control)
   begin
   case(ALU_Control)
      4'b0000: ALU_Result = A & B;
      4'b0001: ALU_Result = A | B;
      4'b0010: ALU_Result = A + B;
      4'b0110: ALU_Result = A - B;
      4'b1110: ALU_Result = A<<shift_amt;/*SLL*/
      4'b1100: ALU_Result = A>>shift_amt;/*SRL*/
   endcase
   end
   always @(ALU_Result)
   begin
      if(ALU_Result==0) Zero = 1;
      else Zero = 0;
   end
endmodule
```

```
module Processor(
    input Clk,
    input Reset
    );
    wire [31:0] Instr_Code;
    wire [31:0] result;
    wire [31:0] Read_Data1;
    wire [31:0] Read_Data2;
    wire Reg_Write,Zero;
    wire [3:0] ALU_op;
    wire [31:0] result1;
    wire imm_signal;

    wire [20:0] imm;
    assign imm = Instr_Code[21:0];

    IF i1(Clk,Reset,Instr_Code);

    Register_File rf1(Instr_Code[20:16],Instr_Code[15:11], Instr_Code[25:21], result1, Read_Data1, Read_Data2, Reg_Write, Clk, Reset);

    Control_Unit cu1(Instr_Code,ALU_op,imm_signal,Reg_Write);
    reg [31:0] A,B;
    always @(*)
      begin
         if(imm_signal==1)
            begin
               A = 32'b0;
               B = {{11{imm[20]}}, imm};
            end
         else
            begin
               A = Read_Data1;
               B = Read_Data2;
            end
      end

    ALU alu1(A,B, ALU_op, imm_signal, imm, result, Zero);
    assign result1 = result;
endmodule
```

Q1.4. 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: Click here to enter text.

Control Signal Name →	**ALU op**	RegWrite	ImmSel		
li r1, 8	0010	1	1		
add r0, r1, r2	0010	1	0		
sub r4, r5, r6	0110	1	0		
and r8, r9, r10	0000	1	**0**		
and r9, r8, r10	0000	**1**	0		
sll r11, r6, 6	1000	1	0		
srl r13, r9, 10	1001	1	0		

Q1.5. Implement the main control unit and copy the image of Verilog code of Main control unit here.

Answer:

```
//////////////////////////////////////////////////////////////////
module Control_Unit(
    input [31:0] Instruction_Code,
    output reg [3:0] ALU_op,
    output reg imm_signal,
    output reg Reg_Write
    );
    wire [5:0] funct6;
    assign funct6 = Instruction_Code[5:0];
    always @(*) begin
       //Reg_Write = 0;
       Reg_Write = 1'b1;
       imm_signal = 1'b0;
       if(Instruction_Code[31:26] == 6'b000000)
         begin
             case(funct6)
                6'b100000: ALU_op = 4'b0010;/*add*/
                6'b100010: ALU_op = 4'b0110;/*sub*/
                6'b100100: ALU_op = 4'b0000;/*AND*/
                6'b100101: ALU_op = 4'b0001;/*OR*/
                6'b000000: ALU_op = 4'b1110;/*SLL*/
                6'b000010: ALU_op = 4'b1100;/*SRL*/
             endcase

       end
       if(Instruction_Code[31:26] == 6'b111111)
         begin
            ALU_op = 4'b0010;
            imm_signal = 1'b1;
         end
    end
endmodule
```

Q1.6. 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:

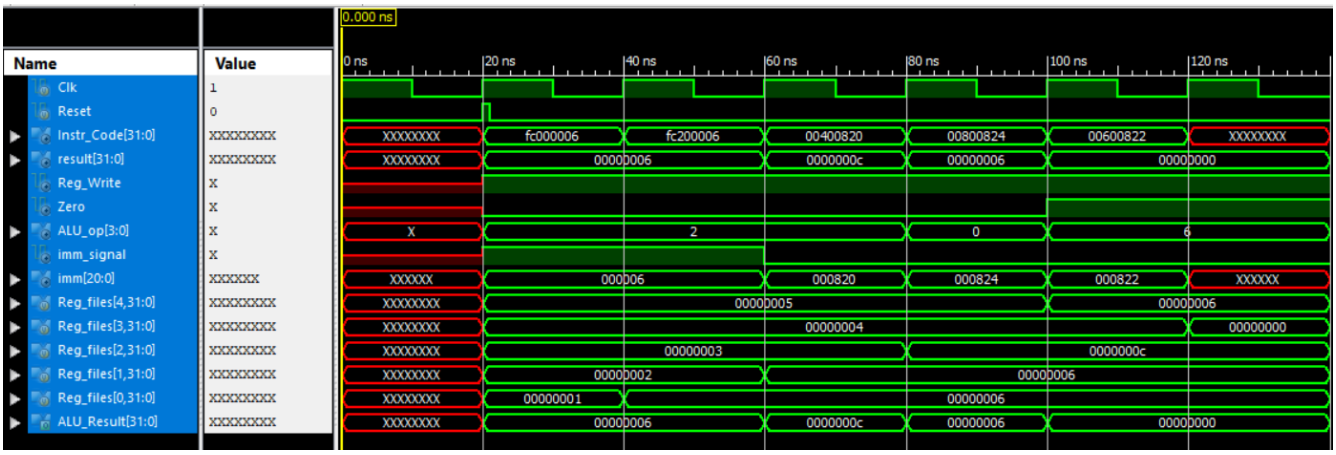
```
module Processor(
    input Clk,
    input Reset
    );
    wire [31:0] Instr_Code;
    wire [31:0] result;
    wire [31:0] Read_Data1;
    wire [31:0] Read_Data2;
    wire Reg_Write,Zero;
    wire [3:0] ALU_op;
    wire [31:0] result1;
    wire imm_signal;

    wire [20:0] imm;
    assign imm = Instr_Code[21:0];

    IF i1(Clk,Reset,Instr_Code);

    Register_File rf1(Instr_Code[20:16],Instr_Code[15:11], Instr_Code[25:21], result1, Read_Data1, Read_Data2, Reg_Write, Clk, Reset);

    Control_Unit cu1(Instr_Code,ALU_op,imm_signal,Reg_Write);
    reg [31:0] A,B;
    always @(*)
      begin
         if(imm_signal==1)
            begin
               A = 32'b0;
               B = {{11{imm[20]}}, imm};
            end
         else
            begin
               A = Read_Data1;
               B = Read_Data2;
            end
      end

    ALU alu1(A,B, ALU_op, imm_signal, imm, result, Zero);
    assign result1 = result;
endmodule
```

Q1.7. 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 r0,6**
li r1,6
add r2,r0,r1
sub r3,r0,r1
and r4,r0,r1

Q1.8. 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: problems with reg and wire and assignment and storing register values

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

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: ANANTHA SAI SATWIK VYSYARAJU **Date:** 27-03-2022

ID No.: 2019A3PS1323H