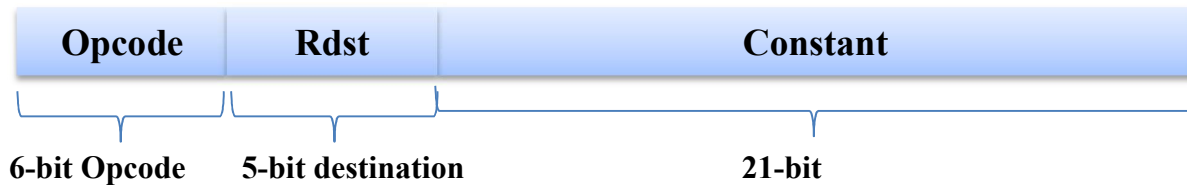


# Assignment 1: Implementation of a processor (10Marks)

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

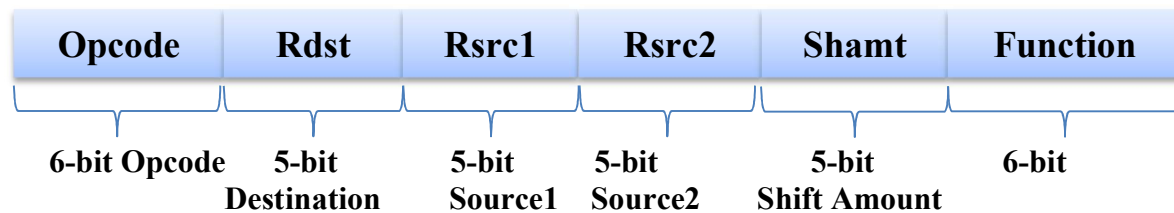
## 1. Immediate Type

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
<b>add</b>	<code>add r0, r1, r2</code>	000000	00000	00001	00010	00000	100000
<b>sub</b>	<code>sub r4, r5, r6</code>	000000	00100	00101	00110	00000	100010
<b>AND</b>	<code>and r8, r9, r10</code>	000000	01000	01001	01010	00000	100100
<b>OR</b>	<code>and r9, r8, r10</code>	000000	01001	01000	01010	00000	100101
<b>sll</b>	<code>sll r11, r6, 6</code>	000000	01011	00110	00000*	00110	000000
<b>srl</b>	<code>srl r13, r9, 10</code>	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<sup>th</sup> location of instruction memory.

As part of the assignment the following files should be submitted in zipped folder.

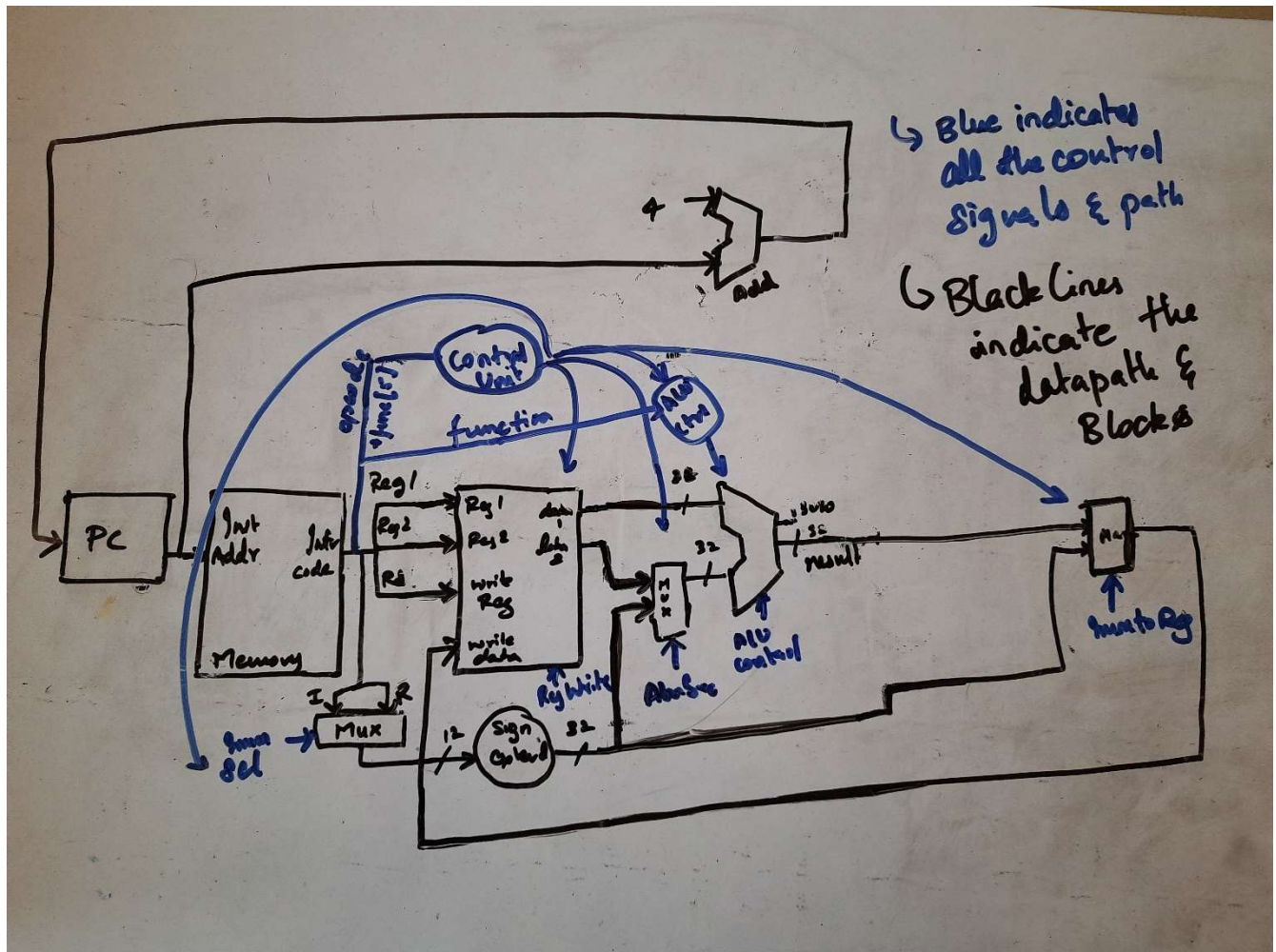
1. PDF version of this Document with all the Questions below answered with file name as IDNO\_NAME.pdf.
2. Design Verilog Files for all the Sub-modules (including control unit).
3. Design Verilog file for the main processor.

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: Program Counter, Instruction Memory (Together form instruction fetch unit)  
Register bank, ALU, Write-back stage (just a mux), main control unit, ALU Control unit,  
immediate generator.

(Data memory is added but not required for the current implementation)

**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.

Answer:

**Immediate Generation unit:**



```
module immGen(
    input [31:0] instr,
    input [1:0] immSel,
    output reg [31:0] immOut
);

parameter [2:0] I_type = 2'b00, S_type = 2'b01, B_type = 2'b10, R_type = 2'b11;
// The S_type and B_type are not being used, have just added it for future extension.
// Moreover, R-type now has new immediate values in the shift left and shift right instructions

always @(*)
    begin
        case(immSel)
            I_type:
                immOut <= {{11{instr[20]}}, instr[20:0]};
            R_type:
                immOut <= {27'b0, instr[11:6]};
            default:
                immOut <= 32'bx;
        endcase
    end
endmodule
```

**ALU Control:**

```

module aluControl(
    input [1:0] aluOp,
    input [5:0] func,

    output reg [3:0] aluCtrl
);

wire [3:0] funct;

assign funct = {func[5], func[2:0]}; // Not all bits are required

parameter AND = 4'b1100, OR = 4'b1101, ADD = 4'b1000, SUB = 4'b1010, LEFT = 4'b0000, RIGHT = 4'b0010;
parameter I_type = 2'b00, R_type = 2'b01;

always @(*)
begin
    case(aluOp)
        I_type:
            aluCtrl = 4'bx; // You don't need an alucontrol signal for this

        R_type:
            begin
                case(func)
                    AND:
                        aluCtrl = 4'b0000;
                    OR:
                        aluCtrl = 4'b0001;
                    ADD:
                        aluCtrl = 4'b0010;
                    SUB:
                        aluCtrl = 4'b0100;
                    LEFT:
                        aluCtrl = 4'b0011;
                    RIGHT:
                        aluCtrl = 4'b0111;
                endcase
            end
        default:
            aluCtrl = 4'bx;
    endcase
end
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: The different control signals needed are: immSel, aluOp, regWrite (always 1), aluSrc, immToReg (tells if immediate value or alu result is to be written to the destination register). In this implementation even R-type instruction needs an immediate select because of sll and srl.

<b>Control Signal Name →</b>	<b>ImmSel</b>	<b>aluOp</b>	<b>aluSrc</b>	<b>immToReg</b>	<b>regWrite</b>
<b>li r1, 8</b>	00	xx	x	1	1
<b>add r0, r1, r2</b>	11	01	0	0	1
<b>sub r4, r5, r6</b>	11	01	0	0	1
<b>and r8, r9, r10</b>	11	01	0	0	1
<b>and r9, r8, r10</b>	11	01	0	0	1
<b>sll r11, r6, 6</b>	11	01	1	0	1
<b>srl r13, r9, 10</b>	11	01	1	0	1

**Q1.5. 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 func,           //6th bit of the function tag in the instruction encoding

    output reg [1:0] immSel,
    output reg [1:0] aluOp,
    output reg regWrite,
    output reg memToReg, immToReg, //Another mux that indicates that immediate value needs to be
    written to writeRegister
    output reg aluSrc,
    output reg memRead, memWrite,
    output reg branch
);

parameter [5:0] I_type = 6'b111111, R_type = 7'b000000;

always @(*)
begin
    case(opcode)
        R_type:
        begin
            immSel <= 2'b11;
            aluOp <= 2'b01;
            regWrite <= 1'b1;
            memToReg <= 1'b0;
            immToReg <= 1'b0;
            if(func==0)
                aluSrc <= 1'b1;
            else
                aluSrc <= 1'b0;
            memRead <= 1'b0;
            memWrite <= 1'b0;
            branch <= 1'b0;
        end

        I_type:
        begin
            immSel <= 2'b00;
            aluOp <= 2'b00;
            regWrite <= 1'b1;
            memToReg <= 1'bx;
            immToReg <= 1'b1;
            aluSrc <= 1'bx;
            memRead <= 1'b0;
            memWrite <= 1'b0;
            branch <= 1'b0;
        end

        default:
        begin
            immSel <= 2'b11;
            aluOp <= 2'b00;
            regWrite <= 1'b1;
            memToReg <= 1'b0;
            immToReg <= 1'b0;
            aluSrc <= 1'b0;
            memRead <= 1'b0;
            memWrite <= 1'b0;
            branch <= 1'b0;
        end
    endcase
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 topUnit(
    input clk,
    input rst
);

wire [31:0] instrCode;

// INSTRUCTION FETCH STAGE - IF
instrFetch IF(.clk(clk), .rst(rst), .instrCode(instrCode));

wire [6:0] func, opcode;
wire [5:0] shamt;
wire [4:0] rs1, rs2, rd;

assign func = instrCode[5:0];
assign shamt = instrCode[10:6];
assign rs1 = instrCode[20:16];
assign rs2 = instrCode[15:11];
assign rd = instrCode[25:21];
assign opcode = instrCode[31:26];

wire [1:0] aluOp, immSel;
wire regWrite, memToReg, immToReg, aluSrc, memRead, memWrite, branch;
wire [31:0] readRegData1, readRegData2, writeData1, writeData2, imm;

// INSTRUCTION DECODE STAGE - ID
controlUnit ctrl(.opcode(opcode), .func(func[5]), .immSel(immSel), .aluOp(aluOp), .regWrite(regWrite),
    .memToReg(memToReg), .immToReg(immToReg), .aluSrc(aluSrc), .memRead(memRead), .memWrite(memWrite),
    .branch(branch));
registerBank regs(.clk(clk), .rst(rst), .regWrite(regWrite), .readReg1(rs1), .readReg2(rs2),
    .writeReg(rd), .writeData(writeData2), .readRegData1(readRegData1), .readRegData2(readRegData2));
immGen immGen(.instr(instrCode), .immSel(immSel), .immOut(imm));

wire [31:0] aluInput, aluResult;
assign aluInput = aluSrc? imm: readRegData2; //Deciding the second input to ALU;
wire [3:0] aluCtrl;
wire zero;

// EXECUTION STAGE - EX
aluControl ctrl2(.aluOp(aluOp), .func(func), .aluCtrl(aluCtrl));
alu compute(.A(readRegData1), .B(aluInput), .op(aluCtrl), .result(aluResult), .zero(zero));

// MEMORY STAGE - MEM (PS: NOT NEEDED FOR THE CURRENT IMPLEMENTATION OF INSTRS. ADDED ONLY FOR FUTURE
EXTENSIONS)
wire [31:0] memData;
dataMemory mem(.clk(clk), .rst(rst), .addr(aluResult), .writeData(readRegData2), .readData(memData),
    .memWrite(memWrite), .memRead(memRead));

// WRITE BACK STAGE - WB
/*assign writeData1 = memToReg ? memData : aluResult;*/
assign writeData2 = immToReg? imm: aluResult;

endmodule
```



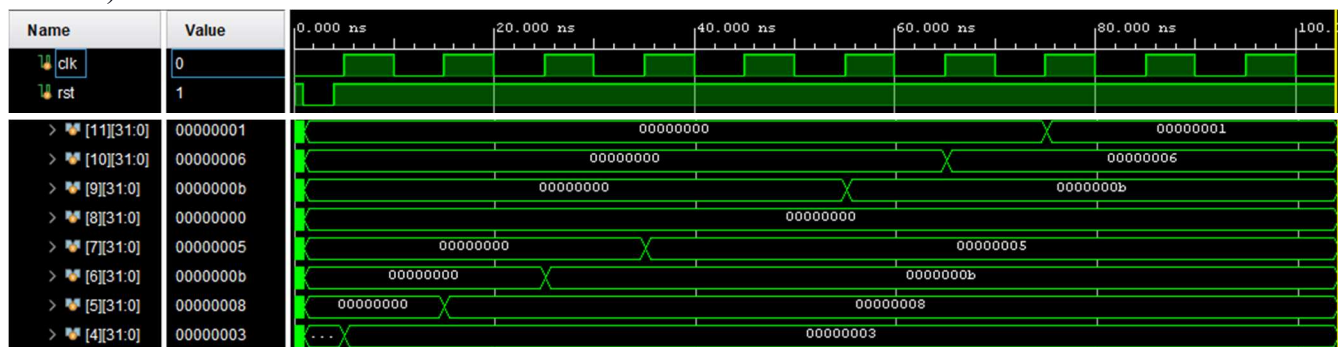
**Q1.7. Test the processor design by initializing the instruction memory with a set of instructions (at least 5 instructions). List below the instructions you have used to initialize the instruction memory. Verify if the register file is changing according to the instructions. (Register file contains unknowns, you can initialize the register or you can load values into the register file using li instruction specified earlier).**

Sequence of Instructions Implemented:

- 1) li tp, 3 (tp is the 4<sup>th</sup> register)
- 2) li t0, 8 (t0 is the 5<sup>th</sup> register)
- 3) add t1, tp, t0 (t1 is the 6<sup>th</sup> register)
- 4) sub t2, t0, tp (t2 is the 7<sup>th</sup> register)
- 5) and s0, tp, t0 (s0 is the 8<sup>th</sup> register)
- 6) or s1, tp, t0 (s1 is the 9<sup>th</sup> register)
- 7) sll a0, tp, 1 (a0 is the 10<sup>th</sup> register)
- 8) srl a1, tp, 1 (a1 is the 11<sup>th</sup> register)

**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: No problems faced as such

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: Had healthy discussion with Suraj and Naren during the development of the assignment.



**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:** VISHWAS VASUKI GAUTAM**Date:** 23/03/2022**ID No.:** 2019A3PS0443H