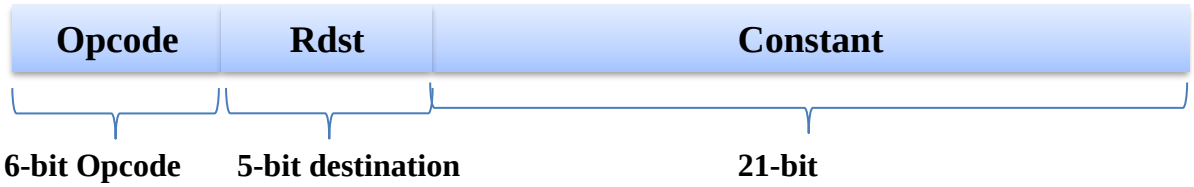
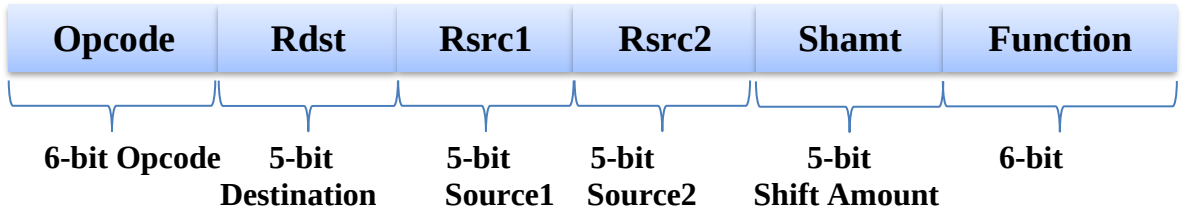
Lab 3 - Implementation of a MIPS like processor

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.

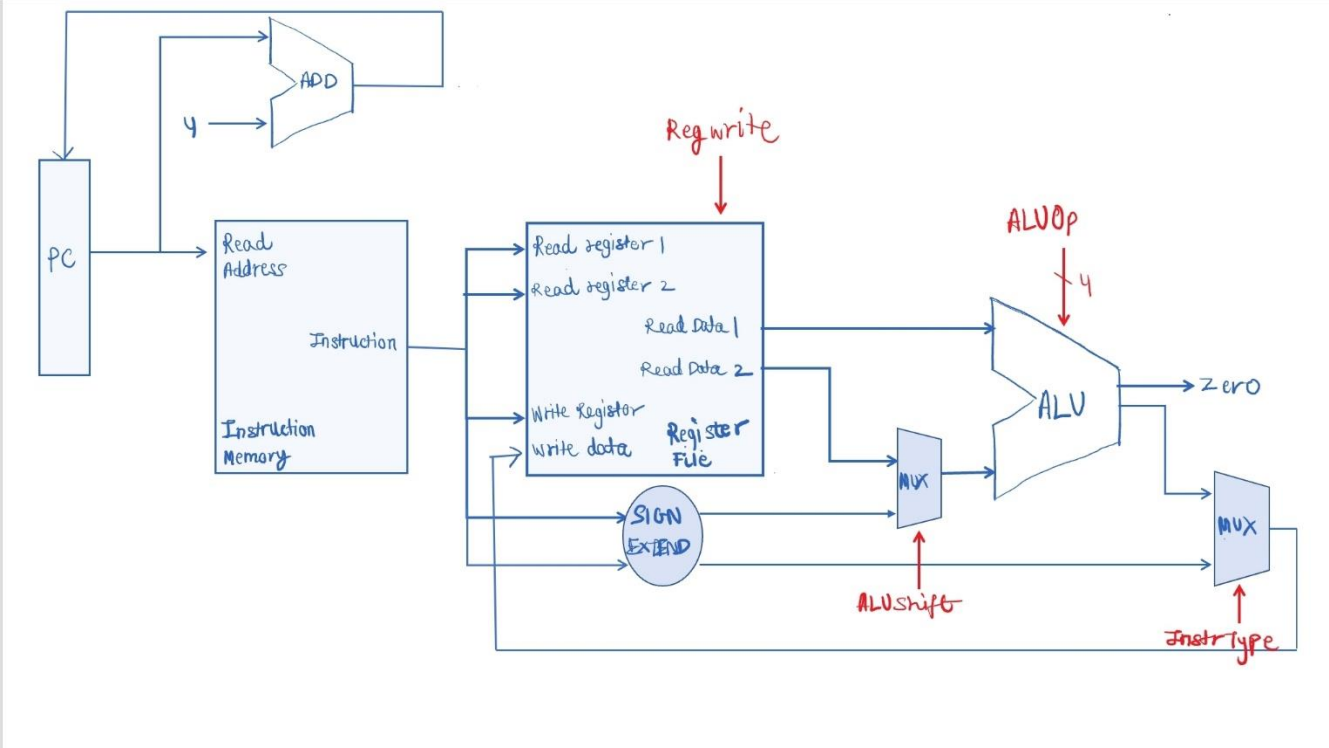
R-type Instruction	Example usage	Opcode	Rdst	Rsrc1	Rsrc2	shamt	Function
add	**add r0, r1, r2**	**000000**	**00000**	**00001**	**00010**	**00000**	**100000**
sub	**sub r4, r5, r6**	**000000**	**00100**	**00101**	**00110**	**00000**	**100010**
AND	**and r8, r9, r10**	**000000**	**01000**	**01001**	**01010**	**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.

Q3.1. 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:



The red signals are the control signals generated by the Main Control unit. The signals are RegWrite, ALUShift, ALUOp, and InstrType.

Q3.2. List the different blocks that will be required for implementation of datapath of the above processor.

Answer: The different blocks that are required for datapath are:

- Instruction Memory
- Register File
- ALU
- Multiplexers
- Register Memory
- Adder
- Sign Extender

Q3.3. 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 data path blocks that are being used are Instruction Memory/Fetch unit, the Register File unit, ALU unit, multiplexers and Sign Extension Unit. The Instruction, Register and ALU unit have already been implemented as part of previous labs. The codes of multiplexers and sign-extension units are of one line only, and hence have been included in the Processor module itself, and can be referred to from there.

Q3.4. 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 →	ALUOp	InstrType	RegWrite	ALUShift
li r1, 8	XXXX	1	1	0
add r0, r1, r2	0010	0	1	0
sub r4, r5, r6	0100	0	1	0
and r8, r9, r10	0000	0	1	0
and r9, r8, r10	0000	0	1	0
sll r11, r6, 6	0011	0	1	1
srl r13, r9, 10	0111	0	1	1

Q3.5. Implement the main control unit and copy the image of Verilog code of Main control unit here.

Answer:

```
`timescale 1ns / 1ps

module ALUControl(
    output reg [3:0] ALUop,
    output reg ALUShift,
    output reg InstrType,
    output RegWrite,
    input [5:0] funct,
    input op
    );

    // WE GENERATE A TOTAL OF 4 CONTROL SIGNALS

    // This block generates ALU control lines
    always @(*) begin
    case (funct)

        6'b100000: ALUop = 4'b0010;  //ADD
        6'b100010: ALUop = 4'b0100;  //SUB
        6'b100100: ALUop = 4'b0000;  //AND
        6'b100101: ALUop = 4'b0001; //OR
        6'b000000: ALUop = 4'b0011; //shift left logical
        6'b000010: ALUop = 4'b0111; //shift right logical

    endcase
    end

    always @(*) begin
    ALUShift = (funct == 6'b000000) || (funct == 6'b000010); //For choosing the shamt field as second operand in ALU
    InstrType = op; //For differentiating between R and I type instruction
    end

    assign RegWrite = 1'b1; //This will remain 1 for both R-type or I-Type instructions

endmodule
```

Q3.6. Implement complete processor in Verilog (using all the datapath blocks and main control unit as modules). Copy the image of Verilog code of the processor here.

Answer:

```
`timescale 1ns / 1ps
module Processor(
    input clk,
    input reset
    );

    wire [31:0] Instruction_Code;
    Instruction_Fetch proc0 (clk, reset, Instruction_Code);
    wire [5:0] opcode;
    wire [4:0] rdst;
    wire [4:0] rsrc1, rsrc2, shamt;
    wire [5:0] funct;
    wire [20:0] constant;
    wire [31:0] write_data;
    wire [31:0] read_src1, read_src2;
    wire RegWrite;
    wire Zero;
    wire ALUShift, InstrType;
    wire [3:0] ALUOp;
    wire [31:0] Result;

    // THIS IS THE INSTRUCTION DECODE UNIT
    assign opcode = Instruction_Code[31:26];
    assign rdst = Instruction_Code[25:21];
    assign rsrc1 = Instruction_Code[20:16];
    assign rsrc2 = Instruction_Code[15:11];
    assign shamt = Instruction_Code[10:6];
    assign funct = Instruction_Code[5:0];
    assign constant = Instruction_Code[20:0]; //SPLIT THE INSTRUCTION CODE FOR R-TYPE and I-TYPE OPERATIONS

   ALUControl proc1 (ALUOp, ALUShift, InstrType, RegWrite, funct, opcode[5]);
   Register_file proc2 (rsrc1, rsrc2, rdst, write_data, read_src1, read_src2, RegWrite, clk, reset);
   wire [31:0] input_A, input_B; //INPUTS TO ALU
   assign input_A = read_src1;
   assign input_B = (ALUShift)? ( { {27{shamt[4]}}, shamt } ):(read_src2);  //MUX FUNCTIONALITY FOR ALU's SECOND OPERAND
   //This mux checks if the function corresponds to shift operation, and replaces the second ALU line with the shift amount
   alu proc3 (input_A, input_B, ALUOp, Zero, Result);
   assign write_data = (InstrType)?( { {11{constant[20]}}, constant } ):(Result); //MUX FUNCTIONALITY FOR WRITING INTO THE REGISTORS
   //This mux will store the data into the registor or store the ALU result into the registor

endmodule
```

Q3.7. 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).

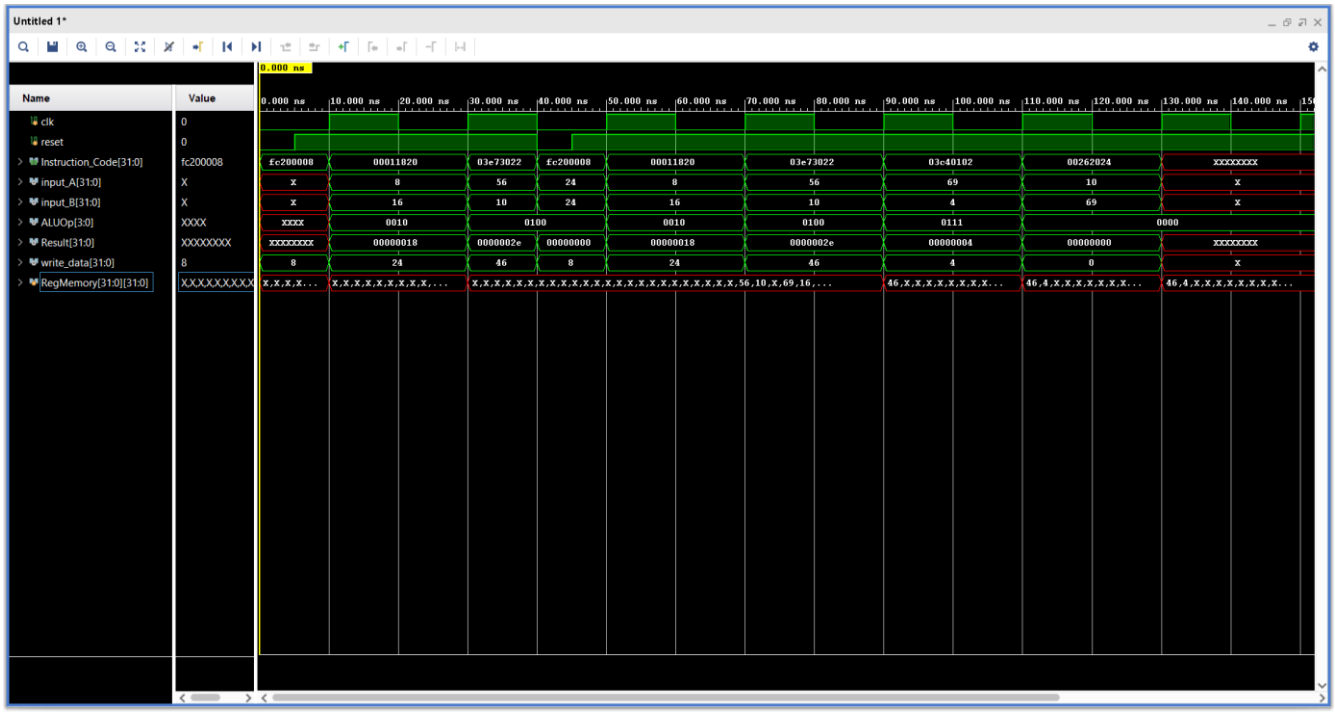
Answer: The instructions that I have used to initialize the instruction memory with are:

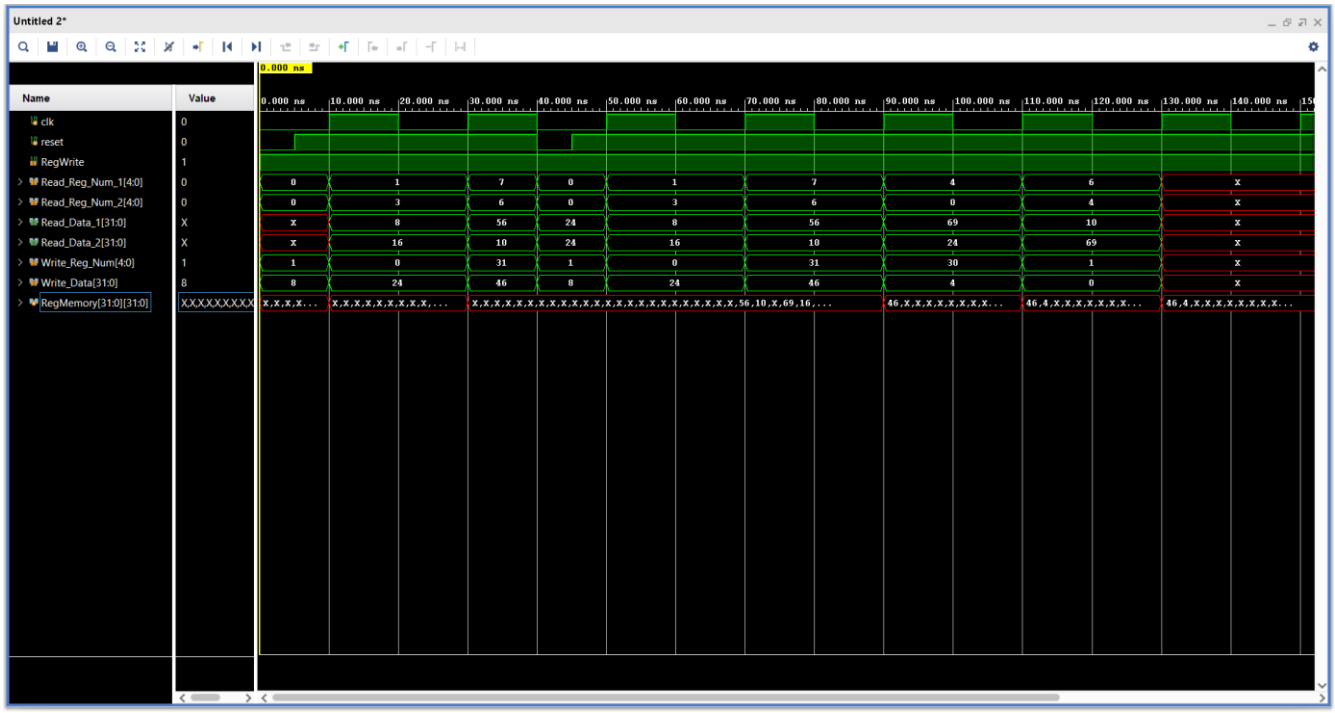
- li R1, 8
- add R0, R1, R3
- sub R31, R7, R6
- srl R30, R4, 4
- AND R1, R6, R4

It has been verified that the register file also changes according to changes in the instructions.

Q3.8. Once design, test and verification are

Copy verified **Register file** waveform here:





The first waveform window shows the ALU operations waveforms. The second one shows the verified Register File waveforms.