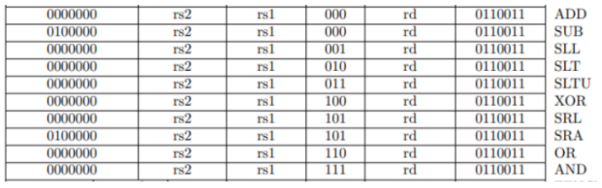
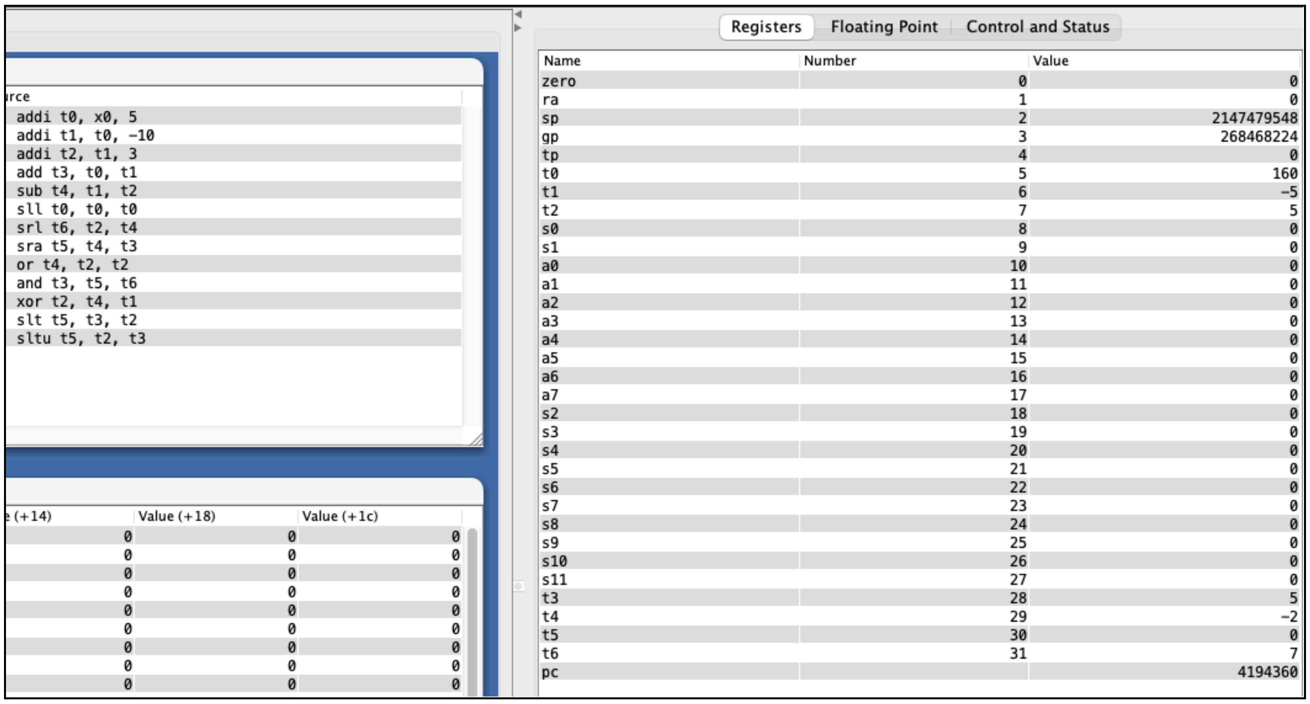
R-Type Instructions:



0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

```
.text
addi t0, x0, 5
addi t1, t0, -10
addi t2, t1, 3
add t3, t0, t1
sub t4, t1, t2
sll t0, t0, t0
srl t6, t2, t4
sra t5, t4, t3
or t4, t2, t2
and t3, t5, t6
xor t2, t4, t1
slt t5, t3, t2
sltu t5, t2, t3
```



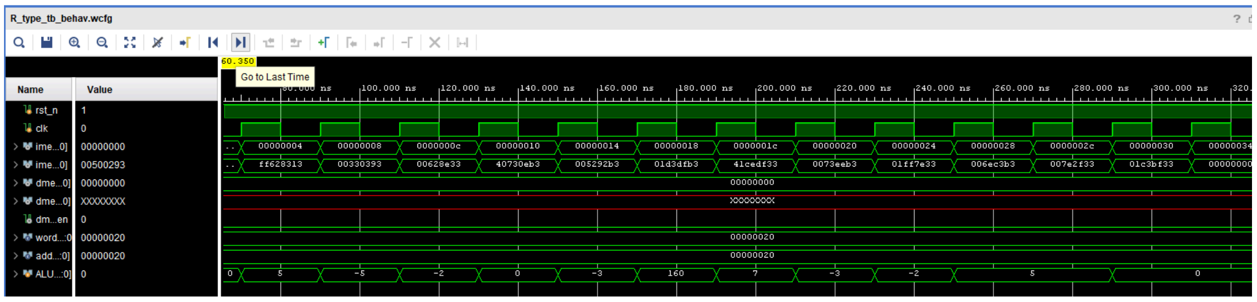
```
rce
 addi t0, x0, 5
 addi t1, t0, -10
 addi t2, t1, 3
 add t3, t0, t1
 sub t4, t1, t2
 sll t0, t0, t0
 srl t6, t2, t4
 sra t5, t4, t3
 or t4, t2, t2
 and t3, t5, t6
 xor t2, t4, t1
 slt t5, t3, t2
 sltu t5, t2, t3
```

e (+14)	Value (+18)	Value (+1c)
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	160
t1	6	-5
t2	7	5
s0	8	0
s1	9	0
a0	10	0
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	5
t4	29	-2
t5	30	0
t6	31	7
pc		4194360

Above are ALU instructions with R-type format that were tested. The expected output of the instructions are shown on the right.

Waveform:


R_type_tb_behav.wcfg
60.350
Go to Last Time
Name
Value
rst_n 1
clk 0
ime...0] 00000000 .. 00000004 00000008 0000000c 00000010 00000014 00000018 0000001c 00000020 00000024 00000028 0000002c 00000030 00000034
ime...0] 00500293 .. ff628313 00330393 00628e33 40730eb3 005292b3 01d3dfb3 41cedf33 0073eeb3 01ff7e33 006ec3b3 007e2f33 01c3bf33 00000000
dme...0] 00000000 00000000
dme...0] XXXXXXXX XXXXXXXX
dm...en 0
word...:0 00000020 00000020
add...:0] 00000020 00000020
ALU...:0] 0 0 5 -5 -2 0 -3 160 7 -3 -2 5 0
80.000 ns 100.000 ns 120.000 ns 140.000 ns 160.000 ns 180.000 ns 200.000 ns 220.000 ns 240.000 ns 260.000 ns 280.000 ns 300.000 ns 320.

Trace files:

```
//trace files
    integer file1, file2, clear1, clear2, CC;
    //pc
    always_comb begin
        file1 = $fopen("C:/Users/mc100/Documents/Coding
Repositories/Classwork/CMPE
140/CMPE140_lab5_brandonV/CMPE140_lab5_bv/CMPE140_lab5_bv.srcs/sim_1/imports/CM
PE140_lab5_bv/pc.txt", "a");
        if (program_counter <= 0) begin
            clear1 = $fopen("C:/Users/mc100/Documents/Coding
Repositories/Classwork/CMPE
140/CMPE140_lab5_brandonV/CMPE140_lab5_bv/CMPE140_lab5_bv.srcs/sim_1/imports/CM
PE140_lab5_bv/pc.txt", "w");
            $fwrite(clear1, "");
        end
        if (file1) begin
            $display("Opening file!");
            $fwrite(file1, "PC: %0h\n", program_counter);
            $fclose(file1);
        end else begin
            $display("Error opening file!");
        end
    end
    //cc, register number, and value
    always_comb begin
        file2 = $fopen("C:/Users/mc100/Documents/Coding
Repositories/Classwork/CMPE
140/CMPE140_lab5_brandonV/CMPE140_lab5_bv/CMPE140_lab5_bv.srcs/sim_1/imports/CM
PE140_lab5_bv/result.txt", "a");
        CC = program_counter / 4;
        if (program_counter <= 0) begin
            clear2 = $fopen("C:/Users/mc100/Documents/Coding
Repositories/Classwork/CMPE
140/CMPE140_lab5_brandonV/CMPE140_lab5_bv/CMPE140_lab5_bv.srcs/sim_1/imports/CM
PE140_lab5_bv/result.txt", "w");
            $fwrite(clear2, "");
        end
        if (file2) begin
            $display("Opening file!");
            $fwrite(file2, "CC: %0d, Register: x%0d, Value: %0d\n", CC,
MEM_WB[11:7], $signed(ALU_RESULT));
            $fclose(file2);
        end else begin
            $display("Error opening file!");
        end
    end
```

Code implemented:

```
`timescale 1ns / 1ps

module cpu(
    input rst_n, clk,
    output reg [31:0] imem_addr,
    output reg [31:0] dmem_addr,
    input [31:0] imem_insn,
    inout reg [31:0] dmem_data,
    output reg dmem_wen
);

    // Register File
    reg [31:0] regfile [31:0];

    // Program Counter
    reg [31:0] program_counter = 0;

    // Pipeline Registers
    reg [31:0] IF_ID, ID_EX, EX_MEM, MEM_WB;
    reg [31:0] immgen, rs1_data, rs2_data, ALU, ALU_RESULT;

    // Control Signals
    reg stall;

    integer i;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // **Reset Logic**
            for (i = 0; i < 32; i = i + 1)
                regfile[i] = 0;

            program_counter = 0;
            IF_ID = 0;
            ID_EX = 0;
            EX_MEM = 0;
            MEM_WB = 0;
            ALU = 0;
            ALU_RESULT = 0;
            stall = 0;

            // Initialize memory control signals to prevent X values
            dmem_wen = 0;
            dmem_addr = 0;
        end
        else begin
            // **Instruction Fetch (IF)**
            if (!stall) begin
                imem_addr = program_counter;
                program_counter = program_counter + 4;
                IF_ID = imem_insn;
            end

            // **Instruction Decode (ID)**
            ID_EX = IF_ID;
            rs1_data = regfile[IF_ID[19:15]];
            rs2_data = regfile[IF_ID[24:20]];

            // Immediate Generation for I-Type
            case (IF_ID[6:0])
                7'b0010011: begin // I-Type ALU Instructions
                    immgen = {{20{IF_ID[31]}}, IF_ID[31:20]};
                    if (IF_ID[14:12] == 3'b001 || IF_ID[14:12] == 3'b101)
                        immgen = {27'b0, IF_ID[24:20]};
                end
                default:
                    immgen = 0;
            endcase

            // **Execution (EX)**
            EX_MEM = ID_EX;
            case (ID_EX[6:0])
                7'b0010011: begin // I-Type ALU
                    case (ID_EX[14:12])
                        3'b000: ALU = rs1_data + immgen;  // ADDI
                        3'b111: ALU = rs1_data & immgen;  // ANDI
                        3'b110: ALU = rs1_data | immgen;  // ORI
                        3'b100: ALU = rs1_data ^ immgen;  // XORI
                        3'b010: ALU = ($signed(rs1_data) < $signed(immgen)) ? 1 : 0; // SLTI
                        3'b001: ALU = rs1_data << immgen[4:0]; // SLLI
                        3'b101: begin
                            if (ID_EX[30] == 1'b0)
                                ALU = rs1_data >> immgen[4:0]; // SRLI
                            else
                                ALU = $signed(rs1_data) >>> immgen[4:0]; // SRAI
                        end
                    endcase
                end
                7'b0110011: begin // R-Type ALU
                   case (ID_EX[14:12])
                      3'b000: begin // ADD/SUB
                         if (ID_EX[30] == 1'b0)
                            ALU = rs1_data + rs2_data; // ADD
                         else
                            ALU = rs1_data - rs2_data; // SUB
                      end
                      3'b111: ALU = rs1_data & rs2_data; // AND
                      3'b110: ALU = rs1_data | rs2_data; // OR
                      3'b100: ALU = rs1_data ^ rs2_data; // XOR
                      3'b010: ALU = ($signed(rs1_data) < $signed(rs2_data)) ? 1 : 0; // SLT
                      3'b001: ALU = rs1_data << rs2_data[4:0]; // SLL
                      3'b101: begin
                         if (ID_EX[30] == 1'b0)
                            ALU = rs1_data >> rs2_data[4:0]; // SRL
                         else
                            ALU = $signed(rs1_data) >>> rs2_data[4:0]; // SRA
                      end
                   endcase
                end
             endcase

             // **Memory (MEM)**
             ALU_RESULT = ALU;
             MEM_WB = EX_MEM;

             dmem_wen = 0;
             dmem_addr = 0;

             case (EX_MEM[6:0])
                7'b0100011: begin // Store (sw)
                   dmem_addr = ALU_RESULT;
                   dmem_wen = 1;
                   $display("📝 Storing Data: Mem[%h] <= %h", dmem_addr,
regfile[EX_MEM[24:20]]);
            end
         endcase

         // **Writeback (WB)**
         if (MEM_WB[6:0] == 7'b0010011 || MEM_WB[6:0] == 7'b0110011) begin
            regfile[MEM_WB[11:7]] = ALU_RESULT;
            $display("✅ Writeback: x%0d <= %0h", MEM_WB[11:7], ALU_RESULT);
         end

         // **Debugging Information**
         $display("🔍 PC: %0h, Instruction: %0h, ALU Result: %0d", program_counter,
imem_insn, ALU_RESULT);
      end
```

```
    end

    // **Memory Write Handling**
    assign dmem_data = (dmem_wen) ? regfile[EX_MEM[24:20]] : 32'bz;

endmodule
```

What did we learn from this lab design?

This lab was an extension of lab 4 and lab 3. In these past two labs we implemented the ADDI instruction and other I type instructions to the cpu. In this lab we were tasked to implement the ALU instructions with R-type format r_type.dat to lead the imem. The primary challenge was having to figure out how to modify the ALU and encoding position of three register indexes such as index of Rd is at 11-7, index of rs1 is at 19-15, and index of rs 2 is at 24-20.

This was able to be achieved with the use of case statements and time spent on researching and many multiple mistake trials. The outputs were tested and verified with the results expected. Using the $display statements, it is easy to debug when there are errors within the code.