COMPUTER SYSTEM ARCHITECTURE

LAB REPORT #06

RISC-V Datapath Implementation+ Control Logic

Prepared by:
SHAZIL 532263

Presented To:
LE USAMA SHAUKAT

NUST CEME
DEPT. COMPUTER ENGINEERING

TABLE OF CONTENT

RISC-V Datapath Implementation (I-Type and R-Type Instructions) + Control Logic

Objective

The aim of this lab is to Check our last implemented RISC-V Datapath either capable of executing both I-type and R-type instruction and secondly we will be implementing Control logic and lastly Top module.

Tools Required

Xilinix Vivado

RISC-V Datapath Overview

The **microarchitecture** of the RISC-V Datapath includes the following key components:

- **Program Counter (PC)**
- **Instruction Memory (IMEM)**
- **Data Memory (DMEM)**
- **Register File**
- **Arithmetic Logic Unit (ALU)**
- **Immediate Generation Unit (ImmGen)**
- **Control Logic**

Control Unit Module

control unit.v

```
1  `timescale 1ns / 1ps
2  //
      //////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 10/16/2025 02:40:59 PM
7  // Design Name:
8  // Module Name: control_unit
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File ted
18 // Additional :
19
20
21
22 module control_unit(
23     input logic [6:0] opcode,
24     output logic regwrite,
25     output logic alusrc,
26     output logic memread,
27     output logic memwrite,
28     output logic memtoreg,
29     output logic branch,
30     output logic [1:0] aluop
31 );
32     always_comb begin
33         regwrite = 1'b0;
34         alusrc   = 'b0;
35         memread  = 1'b0;
36         memwrite = 1'b0;
37         memtoreg = 1'b0;
38         branch   = 'b0;
39         aluop    = 'b00;
40
41         case(opcode)
42           7'b0000011:   begin
```

```
                alusrc    =  'b1;
                memread   = 1'b0;
                memwrite  = 1'b0;
                memtoreg  = 1'b0;
                branch    =  'b0;
                aluop     =  'b10;
              end

              7'b0110011:   begin
                alusrc    =  'b1;
                memread   = 1'b1;
                memwrite  = 1'b0;
                memtoreg  = 1'b0;
                branch    =  'b0;
                aluop     =  'b10;
              end

              7'b0100011:   begin
                alusrc    =  'b0;
                memread   = 1'b1;
                memwrite  = 1'b0;
                memtoreg  = 1'b1;
                branch    =  'b0;
                aluop     =  'b00;
              end

              7'b1100011:   begin
                alusrc    =  'b0;
                memread   = 1'b0;
                memwrite  = 1'b0;
                memtoreg  = 1'b0;
                branch    =  'b1;
                aluop     =  'b01;
              end
            endcase
        end
    endmodule
```

ALU Control Module

```
//
// Create Date: 10/16/2025 03:48:31 PM
// Design Name:
// Module Name: alucontrol
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//



module alucontrol(
    input logic [1:0] op,
    input logic [2:0] x,
    input logic y,
    output logic [3:0] out
);
    logic [5:0] a;

    always_comb begin
        out = 4'b0000;
        a = {x, op, y};
        case(a)
            6'b000100: out = 4'b0010;
            6'b000101: out = 4'b0110;
            6'b111100: out = 4'b0000;
            6'b110100: out = 4'b0001;
        endcase
    end
endmodule
```

1 Top Module

alucontrol.v

top module.v

```
`timescale 1ns / 1ps

module top_module(
    input logic [6:0] opcode,
```

```
    input logic [2:0] x,
    input logic y,
    output logic [3:0] alu_out
);
    logic regwrite, alusrc, memread, memwrite, memtoreg,
       branch;
    logic [1:0] aluop;

    control_unit CU(
        .opcode(opcode),
        .regwrite(regwrite),
        .alusrc(alusrc),
        .memread(memread),
        .memwrite(memwrite),
        .memtoreg(memtoreg),
        .branch(branch),
        .aluop(aluop)
    );

    alucontrol AC(
        .op(aluop),
        .x(x),
        .y(y),
        .out(alu_out)
    );
endmodule
```

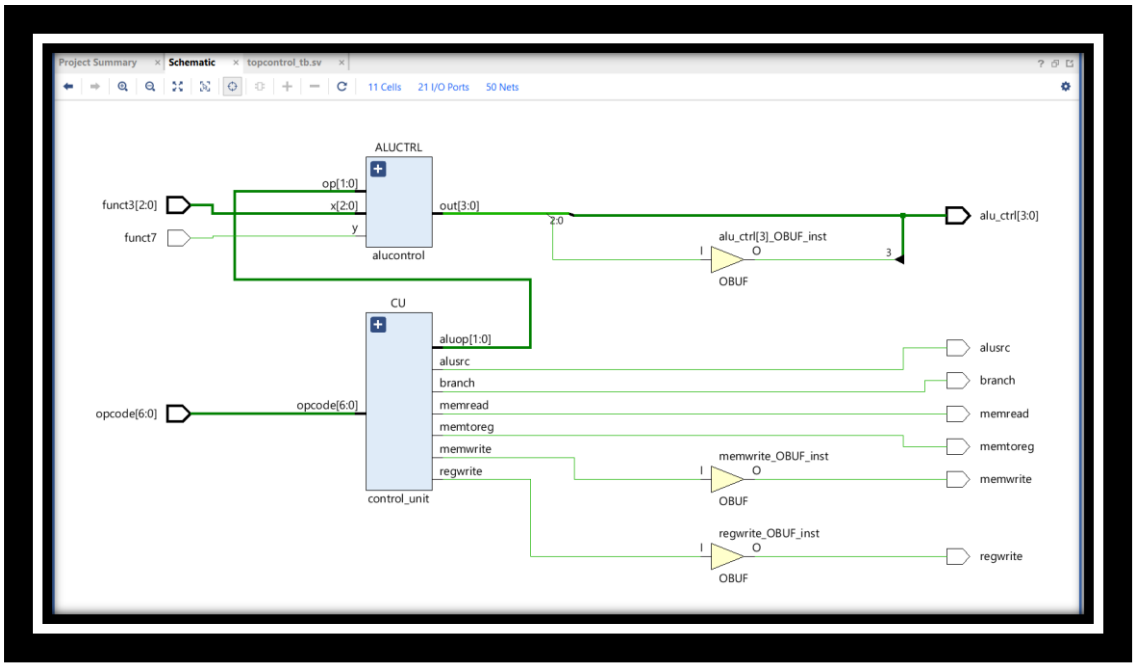
2 Testbench Module

tb top module.v

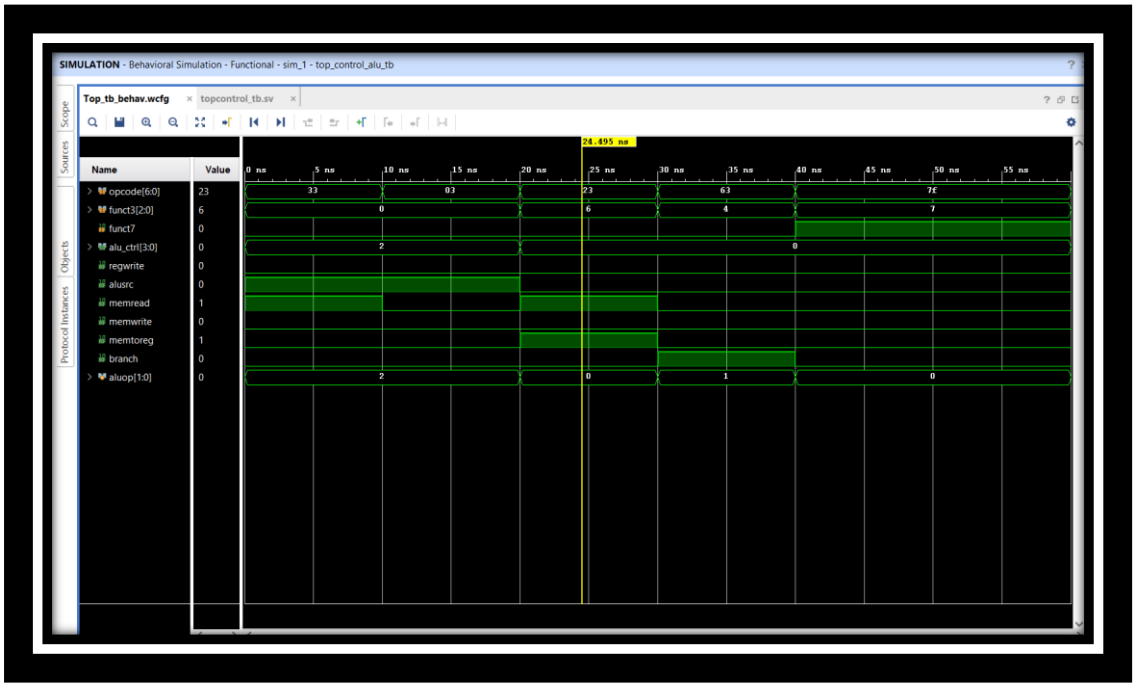
```
`timescale 1ns / 1ps

module tb_top_module;
    logic [6:0] opcode;
    logic [2:0] x;
    logic y;
    logic [3:0] alu_out;

    top_module uut(
        .opcode(opcode),
        .x(x),
        .y(y),
        .alu_out(alu_out)
    );

    initial begin
        $monitor("Time=%0t|Opcode=%b|X=%b|Y=%b|
```

```
            ALU_Out=%b",
18                  $time, opcode, x, y, alu_out);
19
20          opcode = 7'b0110011; x = 3'b000; y = 0; #10;
21          opcode = 7'b0110011; x = 3'b000; y = 1; #10;
22          opcode = 7'b1100011; x = 3'b111; y = 0; #10;
23          opcode = 7'b1100011; x = 3'b110; y = 0; #10;
24          $finish;
25      end
26  endmodule
```

SCHEMATICS DIAGRAM



Simulations:



Conclusion:

The Control Unit and ALU Control modules were successfully designed and verified through simulation. The Control Unit correctly generated the control signals based on the input opcode, demonstrating how instruction decoding drives processor operation. The ALU Control module accurately translated the ALU operation codes into corresponding control signals, ensuring proper arithmetic and logical function selection. The Top Module integrated both units effectively, confirming signal communication and functional behavior. Simulation results from the testbench validated the design logic, timing, and correctness. This lab reinforced understanding of instruction decoding, control signal generation, and modular hardware design in Verilog.