EC39004: VLSI LABORATORY

ASSIGNMENT 5: LAB REPORT

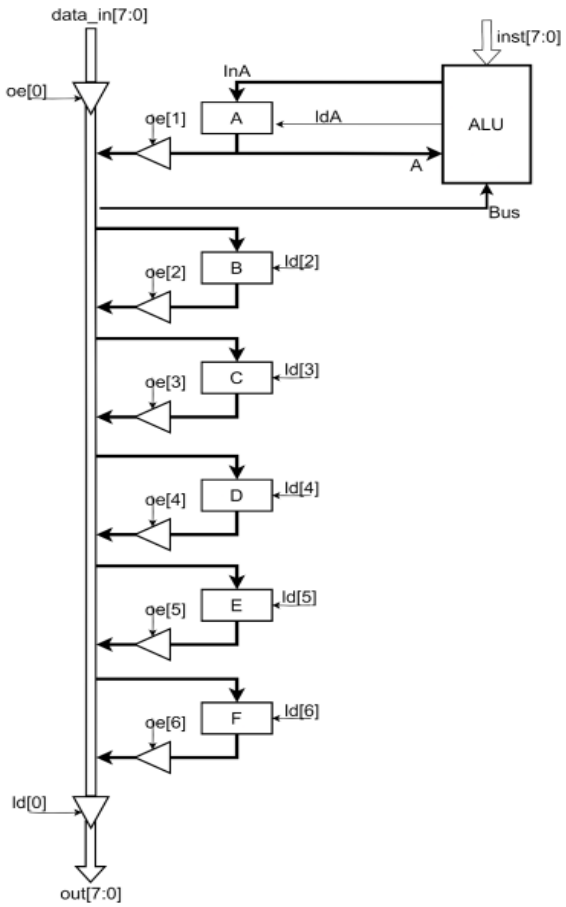
Team Members:

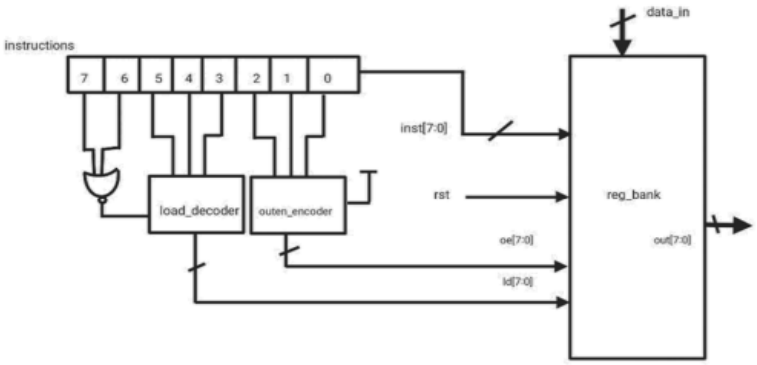
- Jadhav Aryan(21EC30027)
- Panga Sumanth Tejeswar(21EC30037)

Objective-1:

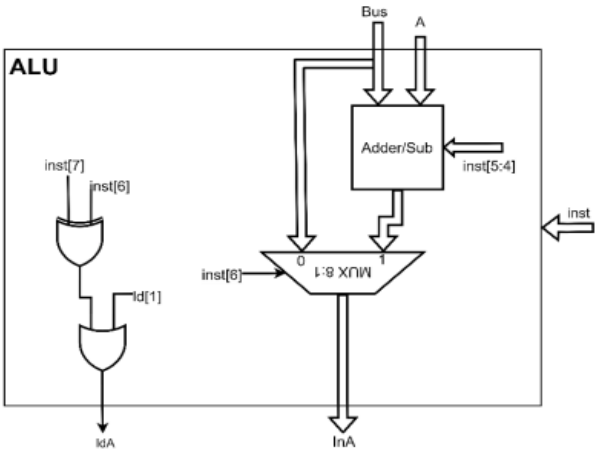
- Design a sample microprocessor with 6 registers and 5 operations. Input bus and Output bus need to be of 8-bit width. The instruction set consists of 8-bit instructions. The allowed operations are ADD, SUB, MOV, IN and OUT.
- Assume the 8 registers as A, B, C, D, E and F. Say R, R1 and R2 is used to represent one of the registers. The operations are defined as follows.
  - ➢ ADD R: Add the value in R to value in A and update A to the obtained sum.
  - ➢ MOV R1 R2: Copy the value of R2 into R1.
  - ➢ IN R: Input an 8-bit number to register R.
  - ➢ OUT R: Output the 8-bit number in register R.
- Assign 8-bit instructions to each of the above 5 operations as per your convenience and proceed with the design. Design using Verilog Hardware Description Language (HDL) behaviourally. Design appropriate testbench and observe and report the obtained waveforms. Show the output waveforms, RTL Schematic

Architecture of Microprocessor:





Main architecture



Introduction:

A **microprocessor** is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provide results as output.

A **register** is a small and temporary storage unit inside a computer's central processing unit (CPU). It plays a vital role in holding the data required by the CPU for immediate processing and is made up of flip-flops. It usually holds a limited amount of data ranging from 8 to 64 bits, depending on the processor architecture.

An **arithmetic logic unit (ALU)** is a major component of the central processing unit of a computer system. It does all processes related to arithmetic and logic operations that need to be done on instruction words.

Here, we have designed a sample Microprocessor that consists of ALU, Registers, buffers, Multiplexer, Decoder, and various other gates. We made sure to execute the 5 types of instructions namely **IN**, **OUT**, **Mov**, **ADD**, **SUB** which are 8-bit instructions.

Below are the Instruction set:

Inst[7:6]	Inst[5:3]	Inst[2:0]	Operation	
00	DDD	000	IN D	D ← input
00	DDD	SSS	Mov D,S	D ← S
00	000	SSS	OUT S	output ← S
01	0XX	S	ADD S	A ← A+S
01	1XX	S	SUB S	A ← A-S

.

Code:

```
module Decoder(input [2:0] A, input en, output reg [7:0] out);

always @( A or en)
    begin
      if (en)
        begin
          out=8'b0;
          case (A)
              3'b000: out[0]=1'b1;
              3'b001: out[1]=1'b1;
              3'b010: out[2]=1'b1;
              3'b011: out[3]=1'b1;
              3'b100: out[4]=1'b1;
              3'b101: out[5]=1'b1;
              3'b110: out[6]=1'b1;
              3'b111: out[7]=1'b1;
              default: out=8'b0;
          endcase
      end
else
out=8'b0;
end
endmodule
```

```
    module Buffer(output [7:0] out, input [7:0] A, input oe);

    bufif1 B1(out[0], A[0], oe);
    bufif1 B2(out[1], A[1], oe);
    bufif1 B3(out[2], A[2], oe);
    bufif1 B4(out[3], A[3], oe);
    bufif1 B5(out[4], A[4], oe);
    bufif1 B6(out[5], A[5], oe);
    bufif1 B7(out[6], A[6], oe);
    bufif1 B8(out[7], A[7], oe);

    endmodule
```

```
module Register(input [7:0] A, input clk, input ld, output reg [7:0] out);
  always @(posedge clk)
  begin
    if(ld)
     out=A;
  end
endmodule
```

```
module Add_sub(input [7:0] A, input [7:0] B, input sel, output [7:0] S);   //sel=0--> ADD    sel=1-->SUB
wire [7:0] t;

Full_adder A0(A[0], B[0]^sel, sel, t[0], S[0]);
Full_adder A1(A[1], B[1]^sel, t[0], t[1], S[1]);
Full_adder A2(A[2], B[2]^sel, t[1], t[2], S[2]);
Full_adder A3(A[3], B[3]^sel, t[2], t[3], S[3]);
Full_adder A4(A[4], B[4]^sel, t[3], t[4], S[4]);
Full_adder A5(A[5], B[5]^sel, t[4], t[5], S[5]);
Full_adder A6(A[6], B[6]^sel, t[5], t[6], S[6]);
Full_adder A7(A[7], B[7]^sel, t[6], t[7], S[7]);

endmodule
```

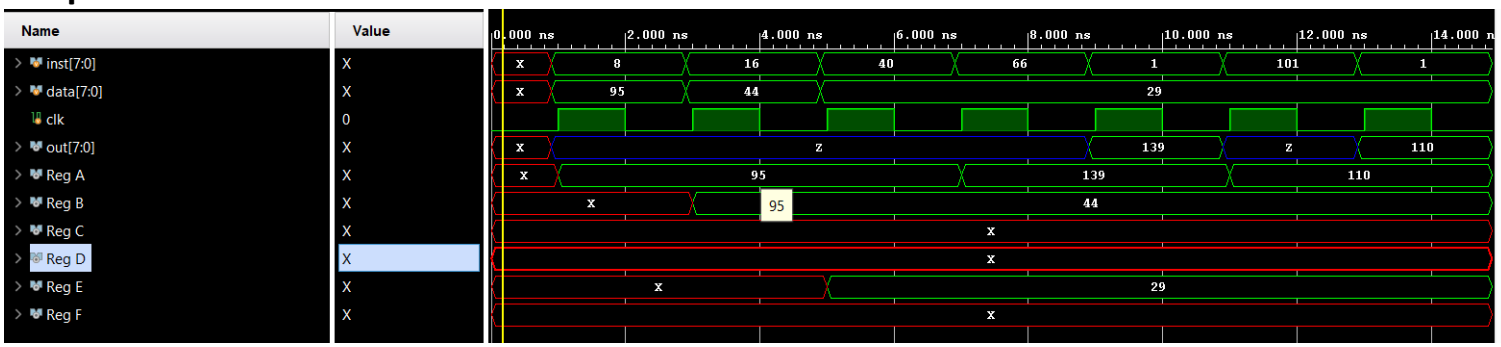
```
module MUX8bit(input [7:0] in1, input [7:0] in2, input sl, output [7:0] out);

MUX2to1 M0(in1[0], in2[0], sl, out[0]);
MUX2to1 M1(in1[1], in2[1], sl, out[1]);
MUX2to1 M2(in1[2], in2[2], sl, out[2]);
MUX2to1 M3(in1[3], in2[3], sl, out[3]);
MUX2to1 M4(in1[4], in2[4], sl, out[4]);
MUX2to1 M5(in1[5], in2[5], sl, out[5]);
MUX2to1 M6(in1[6], in2[6], sl, out[6]);
MUX2to1 M7(in1[7], in2[7], sl, out[7]);

endmodule
```

```
module Microprocessor(input [7:0] inst, input [7:0] data, input clk, output [7:0] out, output [7:0] A_out, output [7:0] B_out, output [7:0] C_out, output [7:0] D
wire ldA;
wire [7:0] ld, oe, Bus, sum, lnA;// A_out, B_out, C_out, D_out, E_out, F_out;
wire en1, en2;
assign en1=~(inst[6]||inst[7]);
assign en2=1;

Decoder load_decoder(inst [5:3] , en1, ld [7:0]);
Decoder outen_encoder(inst [2:0] , en2, oe [7:0]);

assign ldA=(inst[7]^inst[6])||(ld[1]);
Buffer B0(Bus, data, oe[0]);
Buffer B1(Bus, A_out, oe[1]);
Buffer B2(Bus, B_out, oe[2]);
Buffer B3(Bus, C_out, oe[3]);
Buffer B4(Bus, D_out, oe[4]);
Buffer B5(Bus, E_out, oe[5]);
Buffer B6(Bus, F_out, oe[6]);

Register Ra(lnA, clk, ldA, A_out);
Register Rb(Bus, clk, ld[2], B_out);
Register Rc(Bus, clk, ld[3], C_out);
Register Rd(Bus, clk, ld[4], D_out);
Register Re(Bus, clk, ld[5], E_out);
Register Rf(Bus, clk, ld[6], F_out);

Buffer B10(out, Bus, ld[0]);
Add_sub AS0(A_out, Bus, inst[5], sum);
MUX8bit M8b(Bus, sum, inst[6], lnA);
endmodule
```

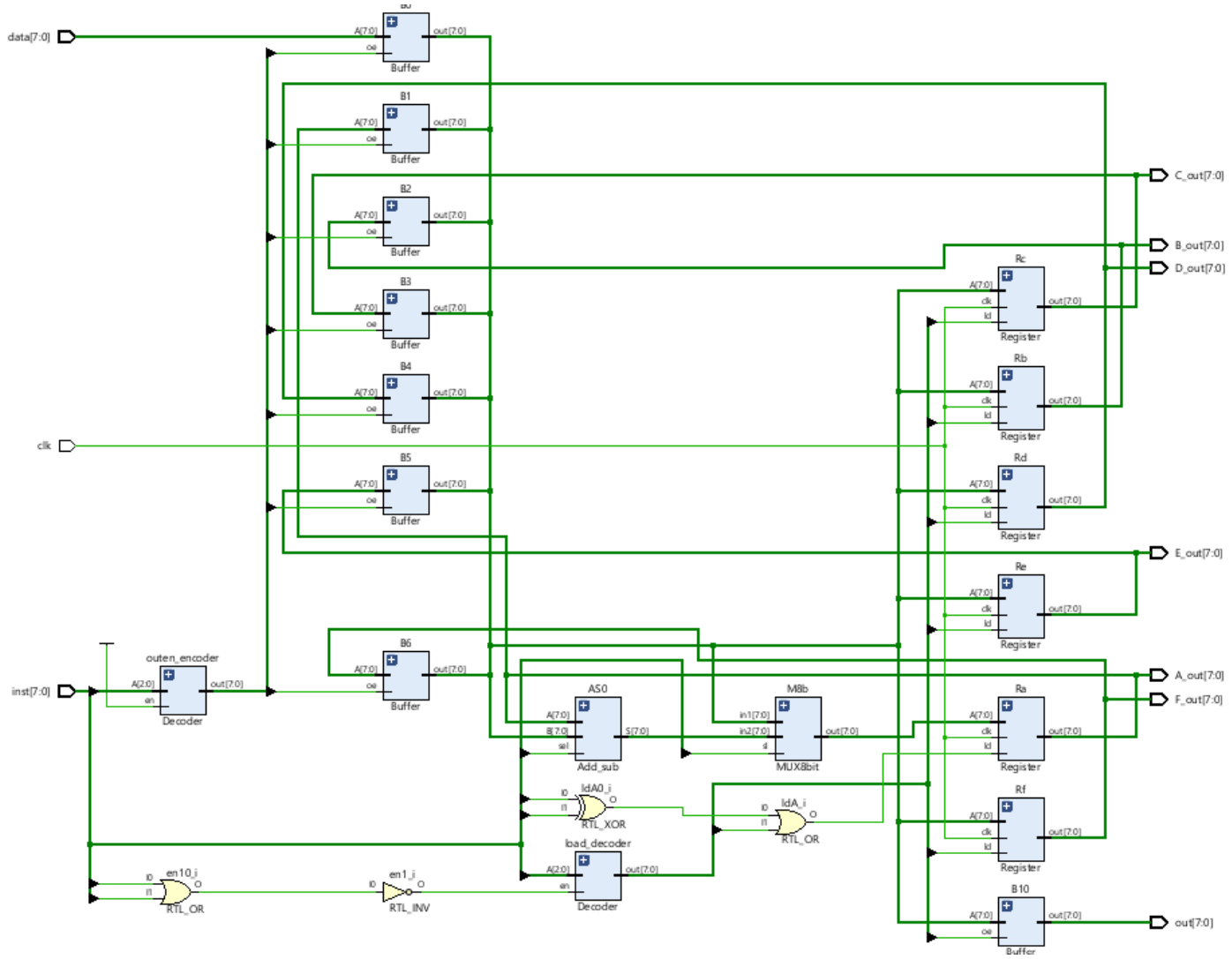
Test Bench:

```
module Microprocessor_tb;
reg [7:0] inst;
reg [7:0] data;
reg clk=1'b0;
wire [7:0] out, A_out, B_out, C_out, D_out, E_out, F_out;
Microprocessor dut(inst, data, clk, out, A_out, B_out, C_out, D_out, E_out, F_out);

always #1 clk=~clk;
 initial
  begin
  #0.9
   inst=8'b00001000; data=8'b01011111; #2  //data-->Reg A
   inst=8'b00010000; data=8'b00101100; #2  //data-->Reg B
   inst=8'b00101000; data=8'b00011101; #2  //data-->Reg E
   inst=8'b01000010; #2 //ADD reg A+B
   inst=8'b00000001; #2 //output reg A (added output)
   inst=8'b01100101; #2 //subtract Reg A-E
   inst=8'b00000001; #2 //output reg A (subtracted output)


   $finish;
  end

initial
    begin
      $dumpfile("wave91.vcd");
      $dumpvars(0, Microprocessor_tb);
    end
endmodule
```

Output Waveforms:



Schematic:



Discussions:

1. In this experiment we implemented a sample microprocessor in Verilog(HDL) which performs 5- types of Instructions namely:
   IN D
   Mov D, S
   OUT S
   ADD S
   SUB S
2. For implementation, we have used/built ALU, 2x1 Multiplexer(8-bit), 8-bit registers and buffer which are connected according to the architectural design.
3. Here, we have used Buffers so that only one data is available in the Bus at a time which avoids data clash.

4. A simple microprocessor is a central processing unit (CPU) designed to execute basic instructions and perform arithmetic and logical operations

5. The type of operation depends on the 8-bit instruction. The 1st two most significant bits of the instruction determine whether it will be an input/output/move operation, or it will be addition/subtraction operation. The remaining 6 bits determine the load decoder and output enable encoder.

6. Register Bank contains all the registers along with the ALU unit. The ALU unit is responsible for the addition and subtraction operations. All the registers are connected to the bus. The output enable when activated transfers data from a register or input to the bus and, the load when activated loads data from the bus to a register.
7. When output enable is activated the data remains in the bus throughout the clock period. However, when the load is activated, data gets loaded only at the positive edge of the clock.