EC39004: VLSI LABORATORY
Design and Implementation of an 8-bit simple Microprocessor

Name - PAWAN KUMAR

Objective:

➢ Diagram

- We have to 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.

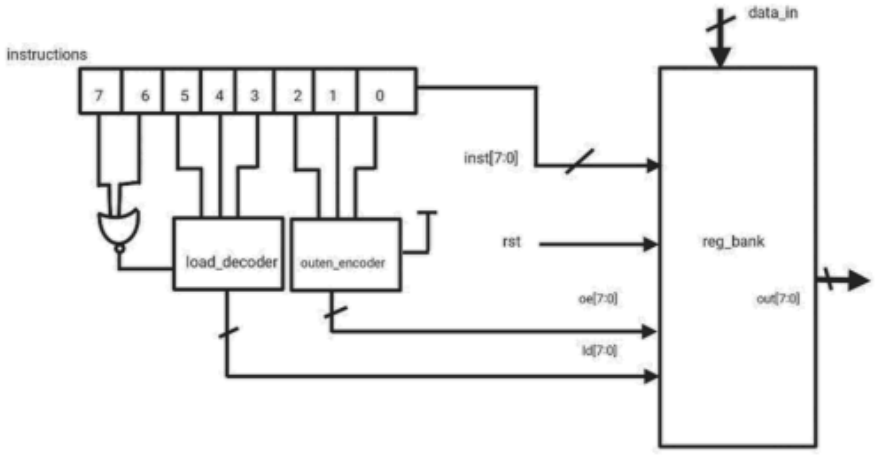
  IN R: Input an 8-bit number to register R.

  OUT R: Output the 8-bit number in register R.

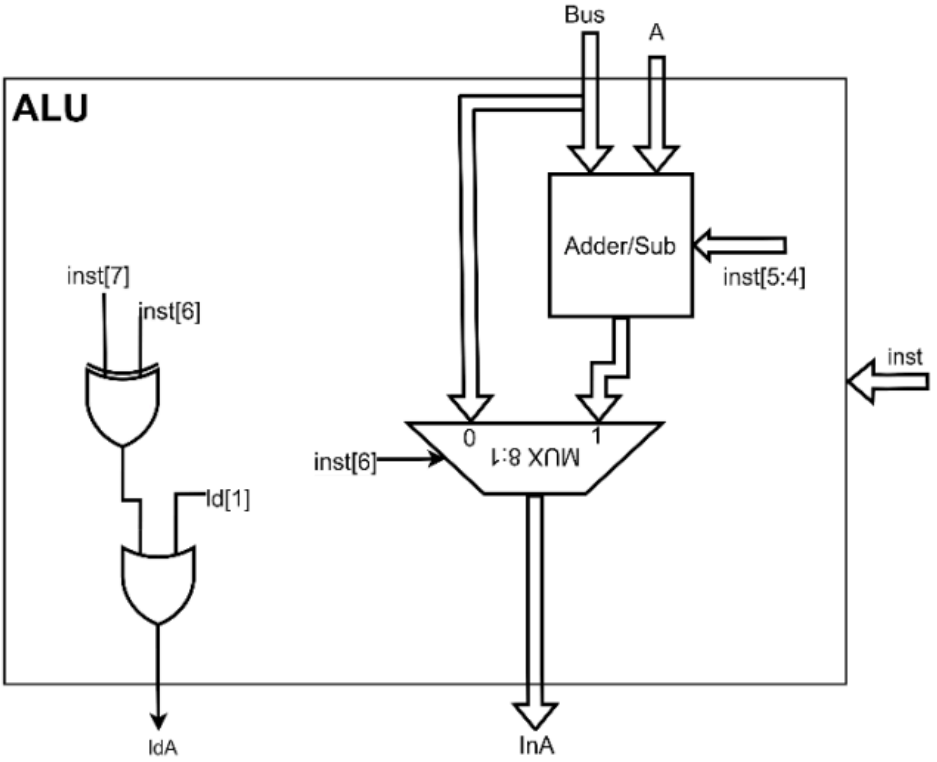
- Assign 8-bit instructions to each of the above 5 operations as per your convenience and proceed with the design. Design using Verilog Hardware Description Language (HDL) behaviourally. Design appropriate testbench and observe and report the obtained waveforms. Show the output waveforms, RTL Schematic

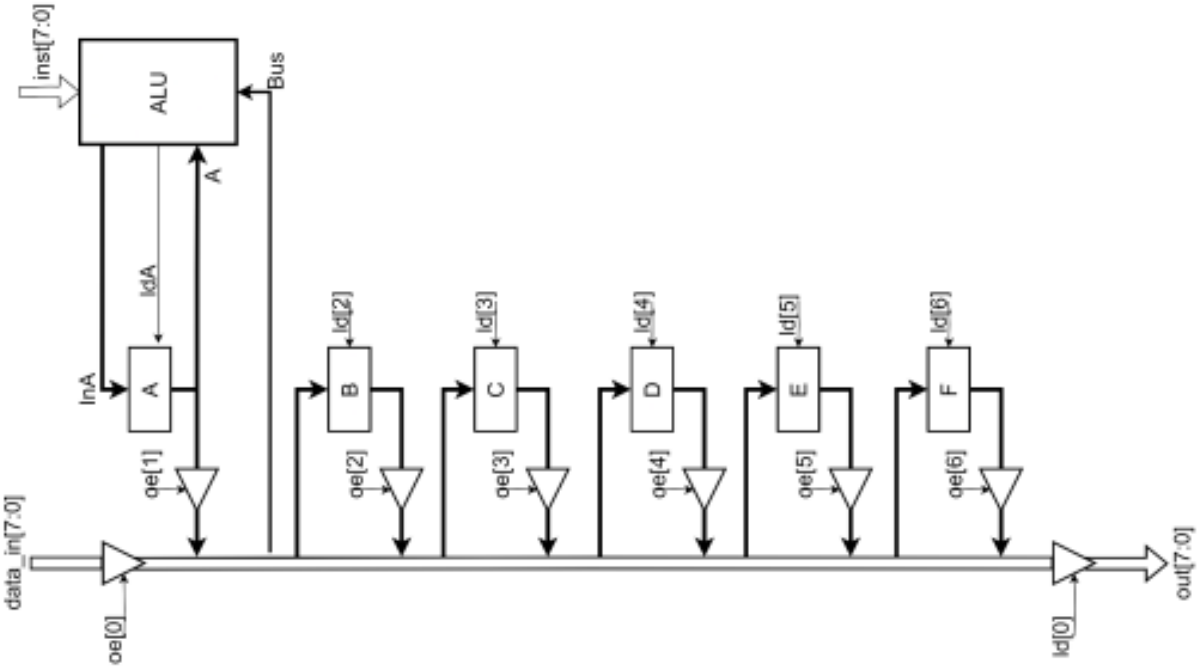
Inst[7:6]	Inst[5:3]	Inst[2:0]	Operation	
00	DDD	000	IN D	D ← input
00	DDD	SSS	Mov D,S	D ← S
00	000	SSS	OUT S	output ← S
01	0XX	S	ADD S	A ← A+S
01	1XX	S	SUB S	A ← A-S

Architecture:



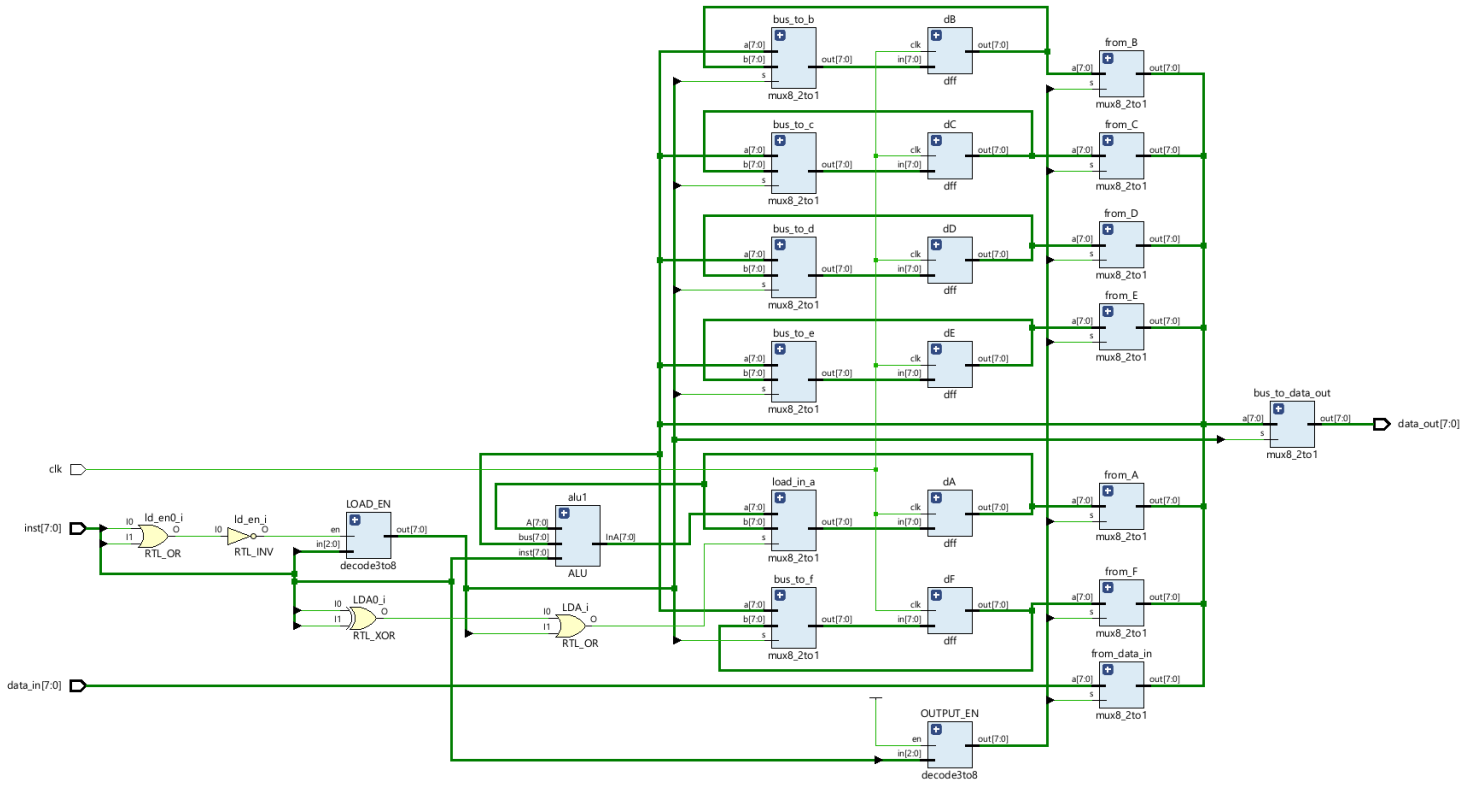
Main architecture





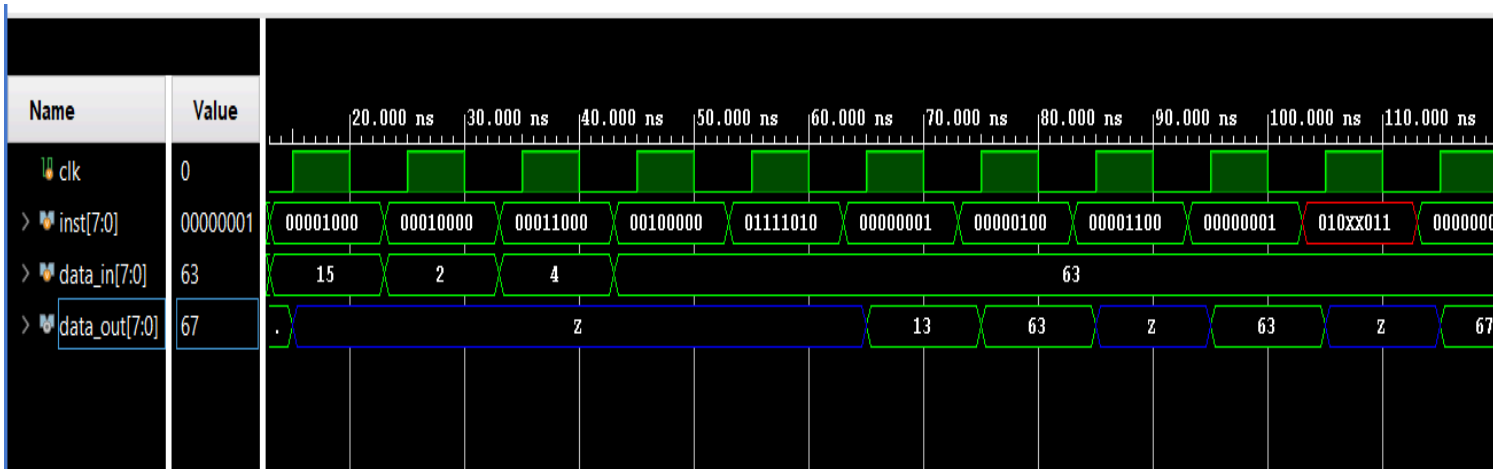
Reg bank architecture

Schematic diagram



➢ Plots and Observations:

- **Plotting the output of the microprocessor**




```
  0 data_in=xxxxxxxx inst=xxxxxxxx  data_out=xxxxxxxx
  5 data_in=10101010 inst=00000000  data_out=10101010
 15 data_in=00001111 inst=00001000  data_out=zzzzzzzz
 23 data_in=00000010 inst=00010000  data_out=zzzzzzzz
 33 data_in=00000100 inst=00011000  data_out=zzzzzzzz
 43 data_in=00111111 inst=00100000  data_out=zzzzzzzz
 53 data_in=00111111 inst=01111010  data_out=zzzzzzzz
 65 data_in=00111111 inst=00000001  data_out=00001101
 75 data_in=00111111 inst=00000100  data_out=00111111
 85 data_in=00111111 inst=00001100  data_out=zzzzzzzz
 95 data_in=00111111 inst=00000001  data_out=00111111
105 data_in=00111111 inst=010xx011  data_out=zzzzzzzz
115 data_in=00111111 inst=00000001  data_out=01000011
```

➢ Codes:

```
`timescale 1ns / 1ps
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
module decode3to8 (input en,                    //module for 3 to 8 decoder
                   input [2:0]in,
                   output [7:0]out);
  assign
    out[0]= en&~in[2]&~in[1]&~in[0],
    out[1]= en&~in[2]&~in[1]& in[0],
    out[2]= en&~in[2]& in[1]&~in[0],
    out[3]= en&~in[2]& in[1]& in[0],
    out[4]= en& in[2]&~in[1]&~in[0],
    out[5]= en& in[2]&~in[1]& in[0],
    out[6]= en& in[2]& in[1]&~in[0],
    out[7]= en& in[2]& in[1]& in[0];
endmodule

  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

module add_sub(input [7:0]inst,                 //module for adder and subtractor
               input [7:0] bus,
               input [7:0] A,
               output [7:0] add_sub_out);
  assign add_sub_out=inst[5]?(A-bus):(A+bus);
endmodule

 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

module mux8_2to1(input s,               //module for 8 2to1 mux
            input [7:0] a,
            input [7:0] b,
            output [7:0] out);
 assign out=s?a:b;
endmodule

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

module ALU(input [7:0]inst,                 //module for ALU
           input [7:0] bus,
           input [7:0] A,
           output [7:0] InA,
           output LDA );
  wire [7:0] add_sub_out;
  wire LD1;

  assign LD1=(~inst[7])&(~inst[6])&(~inst[5])&(~inst[4])&(inst[3]);
  assign LDA=(inst[7] ^ inst[6]) | LD1;

   add_sub A_S (inst, bus, A, add_sub_out);
  mux8_2to1 for_add_sub  (inst[6],add_sub_out,bus, InA);

endmodule
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~microcontroller~~~~~~~~~~~~~~~~~~~~~~

module micro(input clk,
                    input [7:0]inst,
                    input [7:0] data_in,
                    output reg [7:0] data_out  );
  reg [7:0] A,B,C,D,E,F;
  wire [7:0] bus,LD,OE,InA;
  wire ld_en=~(inst[7] | inst[6]);
  parameter oe_en=1'b1;

  wire LDA ;

   ALU alu1(inst, bus,A,InA, LDA );
  decode3to8 outen(ld_en,inst[5:3],LD);
  decode3to8 load(oe_en,inst[2:0],OE);

  always @(posedge clk)
    begin
      if(LDA) A=InA;
    end

  mux8_2to1 from_data_in (OE[0], data_in,8'bz,bus);  //from x to bus
  mux8_2to1 from_A (OE[1], A,8'bz,bus);
  mux8_2to1 from_B (OE[2], B,8'bz,bus);
  mux8_2to1 from_C (OE[3], C,8'bz,bus);
  mux8_2to1 from_D (OE[4], D,8'bz,bus);
  mux8_2to1 from_E (OE[5], E,8'bz,bus);
  mux8_2to1 from_F (OE[6], F,8'bz,bus);

  always @(posedge clk)
    begin
      if(LD[0]) data_out=bus;
      else data_out=8'bz;

       if(LD[2]) B=bus;
      else if(LD[3]) C=bus;
      else if(LD[4]) D=bus;
      else if(LD[5]) E=bus;
      else if(LD[6]) F=bus;
    end

endmodule

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~end~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

• **TestBench**

```
module micro_tb;

   reg clk=1'b0;
   reg [7:0] inst;
   reg [7:0] data_in;
   wire [7:0] data_out;

  micro uut (clk ,inst,data_in,data_out);
 always #5 clk=~clk;

  initial begin
     $dumpfile("tb.vcd");
      $dumpvars(0, micro_tb);
  $monitor ($time," data_in=%b inst=%b  data_out=%b",data_in,inst, data_out);

      #3  inst = 8'b00000000;       // data_out enable, data_in enable
       data_in = 8'b10101010;

      #10 inst = 8'b00001000;       // LD in A, data_in enable
       data_in = 8'b00001111;

     #10 inst = 8'b00010000;       // LD in B, data_in enable
      data_in = 8'b00000010;

    #10 inst = 8'b00011000;        // LD in C, data_in enable
     data_in = 8'b00000100;

    #10 inst = 8'b00100000;        // LD in D, data_in enable
     data_in = 8'b00111111;

   #10 inst = 8'b01111010;         // A=A-B

   #10 inst = 8'b0000001;          // data_out enable, OE of A

   #10 inst = 8'b0000100;          //out D

     #10 inst = 8'b0001100;        //move D in A

      #10 inst = 8'b0000001;       //out A

     #10 inst = 8'b010xx011;       //A=A+C
    #10 inst = 8'b0000001;         // data_out enable, OE of A

      // Finish simulation
     #10 $finish;
   end

endmodule
```

DISCUSSIONS:

- Initially, we encountered an issue where the microprocessor was latching off the last data input and instruction to the current data output, which was resolved by implementing a positive edge-triggered clock signal. This ensures that data and instructions are properly synchronized and latched at the rising edge of the clock, enhancing the reliability and stability of the microprocessor operation.

- The code utilizes a pipelining approach in microprocessing, allowing sequential calling of various modules, a task impossible within an always block. Comprising two stages, the pipeline initially fetches control signals and data, while the subsequent stage handles calculations, if any, and returns data, optimizing the processing workflow.

.

- Each unit is designed differently, the Control Unit, the ALU. The data bus is kept a simple net variable.
- The Output Enable and load of the Registers is implemented by 2 to 1 byte MUX.
- The input of A is further connected to another MUX with output of this MUX and of ALU as inputs.

- A problem related to data hazard is faced with this design. As the values are loaded in the data bus in the first cycle of the pipeline, the same value should not be accessed in the immediate next step. Instead, one should wait for one clock cycle to use that specific register.