EC39004 – VLSI Laboratory Verilog Module

Name : *Rekha Lokesh*

Roll No : *19EC10052*

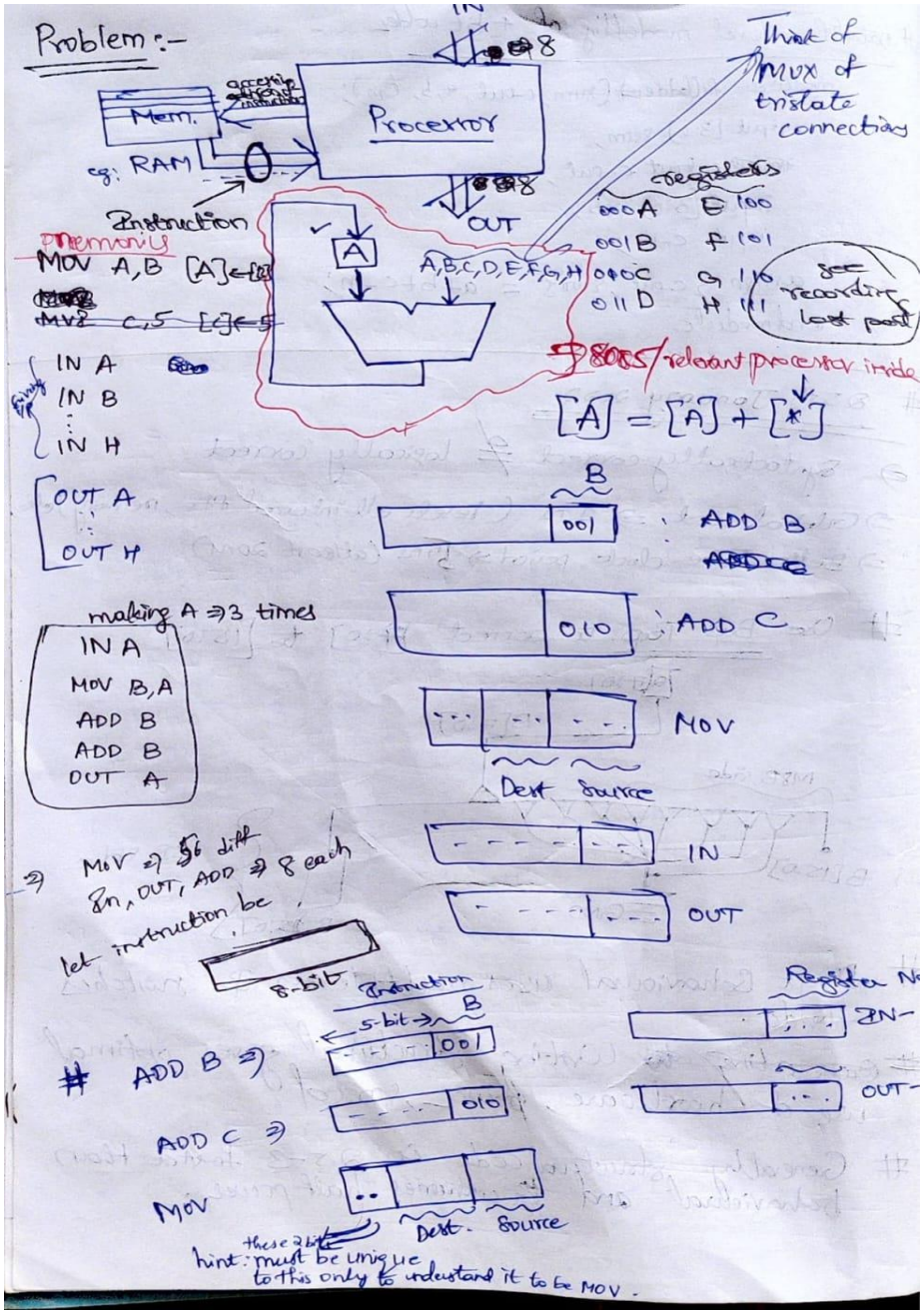
Group : 3

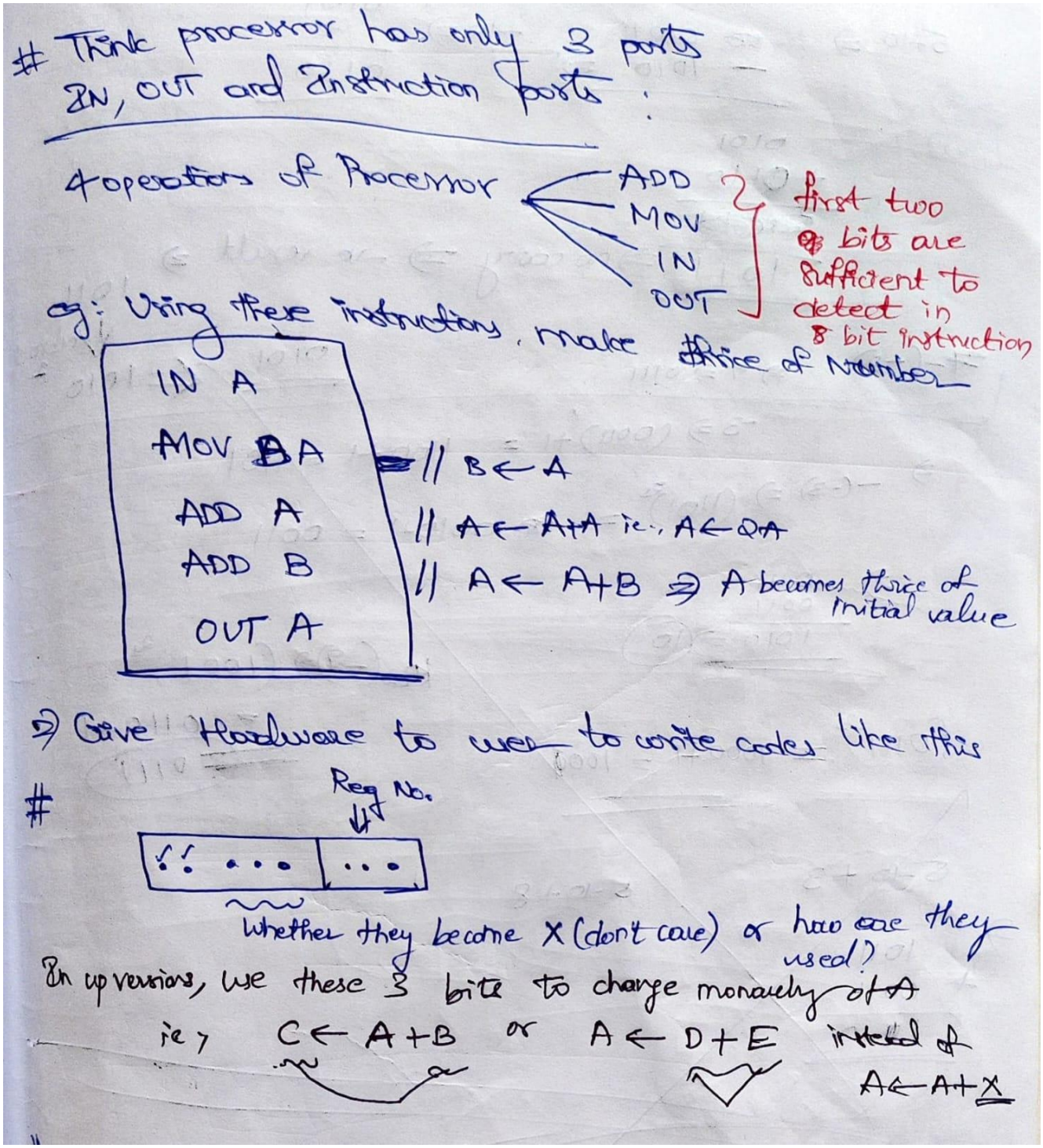
Expt 4: Basic Processor Structural

Objective:

- Write a Structural Verilog code to synthesize the given Basic Processor Operations.

Class Discussion and Problem Statement



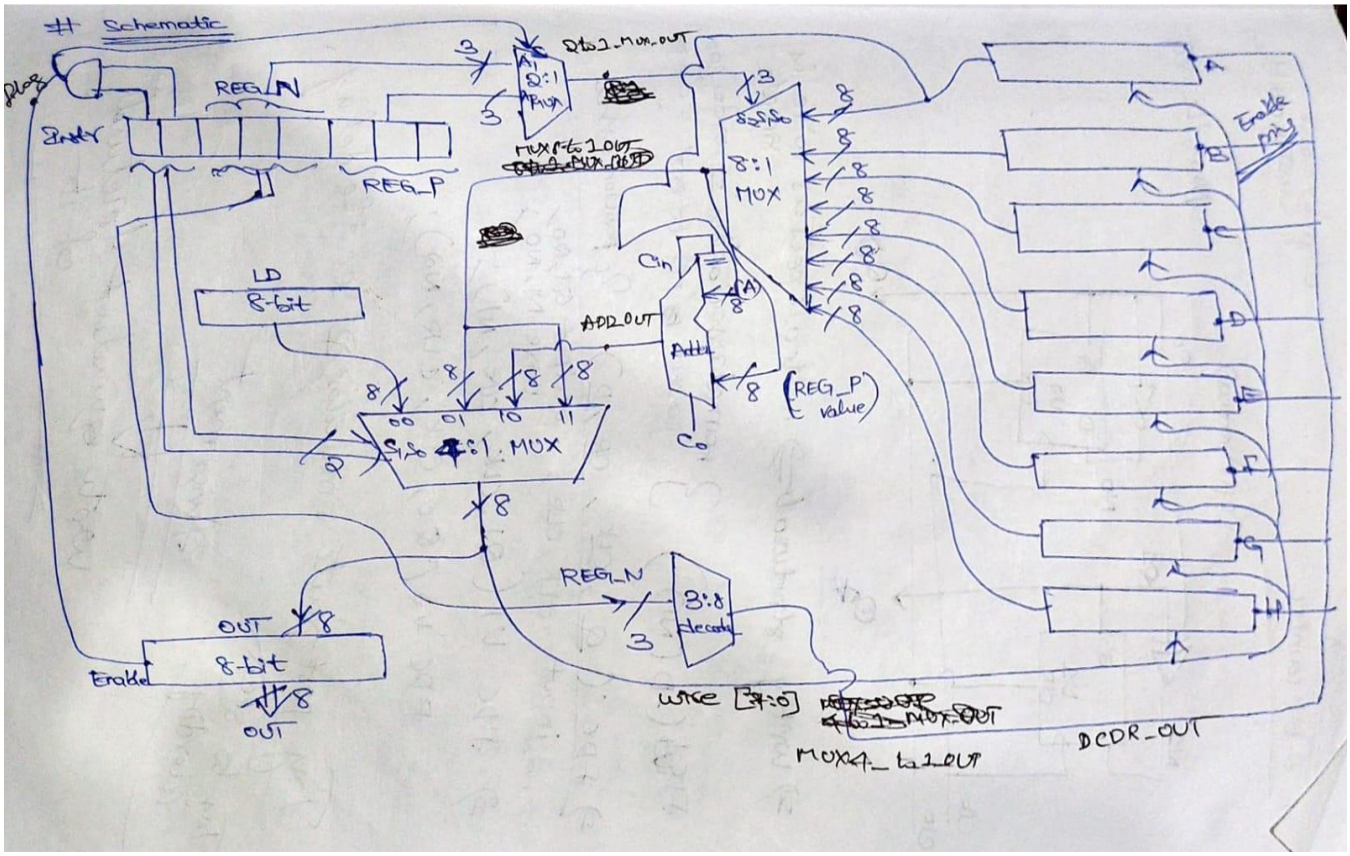


Design approach/steps:

- The instruction, load, clock and Reset are 4 input ports and 'out' is an output port to the system.
- ***Instruction is 8 bits wide*** where the ***two most significant bits are used as select lines*** to multiplex for the operation and ***the next three bits are for destination register*** and ***next three for source register***.

- The 8 storage registers are defined as ***A, B, C,…., H*** and their 3 bit binary values are ***000, 001,….,111*** respectively.
- The operations according to instr[7:6] are:
  - 00 -> Input Load into REGN (source register) -> LD REGN
  - 01 -> Move REGP to REGN -> MOV REGN REGP
  - 10 -> Add A to REGP and Store in REGN -> ADD REGN REGP
  - 11 -> Output REGP (destination register) -> OUT REGP
- The ADDER is designed using a macro and the rest individual components like 8:1 MUX, 4:1 MUX, 2:1 MUX, 3to8 Decoder and Registers, individual modules are written in structural manner (without primitives) and all of them are ***instantiated in the top module*** according to the Circuit Followed below.
- As visible from the circuit, ***4:1 MUX handles the operation***, ***8:1 MUX handles the source register***, ***3:8 Decoder handles the destination register*** and the ***AND gate is used to detect the OUT Case*** where the Source Register is the Destination Register (handled by 2:1 MUX).
- After every given instruction, give ***one cycle to the clock*** so that any ***edge triggered components*** in the system move forward.
- For testcase, take the case of making the value of A thrice using the algorithm LD A -> MOV B A -> OUT B -> ADD A B -> ADD A B -> OUT A

THE CIRCUIT DIAGRAM FOLLOWED



Structural Verilog Code:

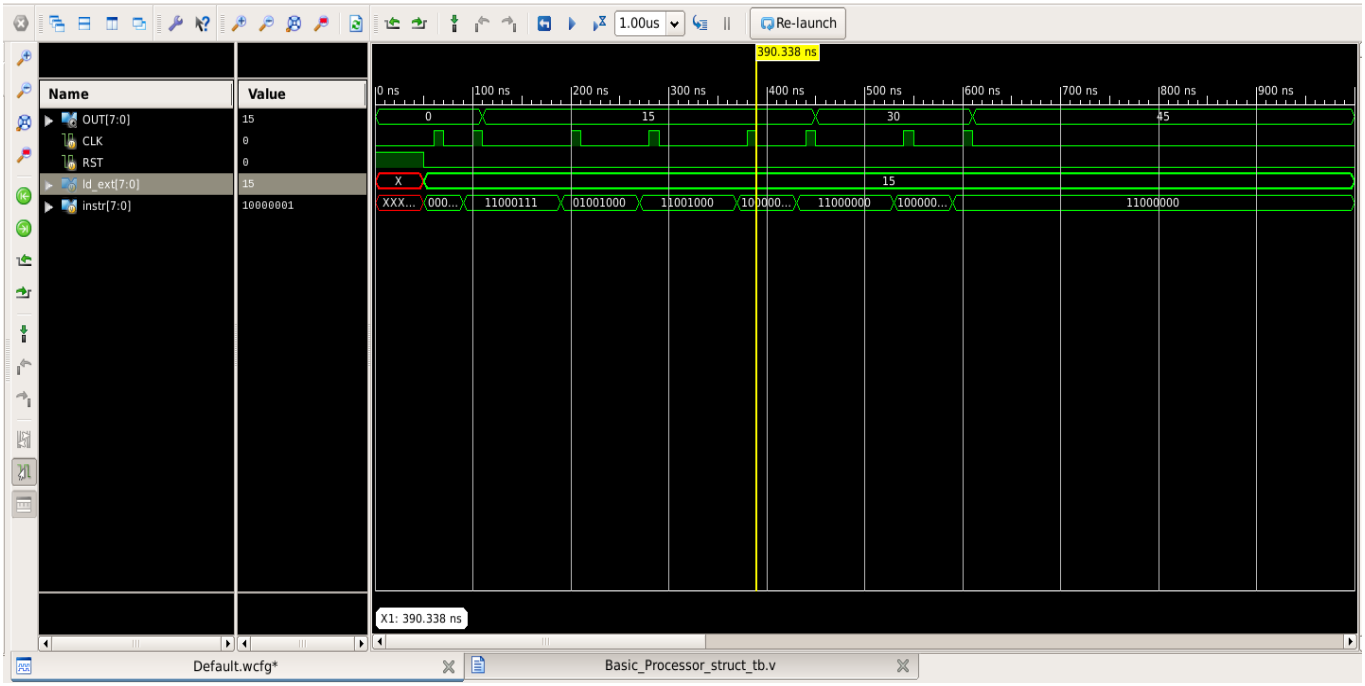
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:43:26 02/06/2022
// Design Name:
// Module Name:    Basic_Processor_Struct
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module Basic_Processor_Struct(
  input [7:0] INSTR,
  input CLK, input RST,
  input [7:0] Load,
  output [7:0] OUT
  );

        wire [2:0] MUX_2to1_OUT;
        wire [7:0] MUX_8to1_OUT, MUX_4to1_OUT, ADD_OUT, DCDR;
        wire [7:0] A_out, B_out, C_out, D_out, E_out, F_out, G_out, H_out;

        wire flag, Car_out;

        RGSTR8CE A(A_out, CLK, DCDR[0], RST, MUX_4to1_OUT);
        RGSTR8CE B(B_out, CLK, DCDR[1], RST, MUX_4to1_OUT);
        RGSTR8CE C(C_out, CLK, DCDR[2], RST, MUX_4to1_OUT);
        RGSTR8CE D(D_out, CLK, DCDR[3], RST, MUX_4to1_OUT);
        RGSTR8CE E(E_out, CLK, DCDR[4], RST, MUX_4to1_OUT);
        RGSTR8CE F(F_out, CLK, DCDR[5], RST, MUX_4to1_OUT);
        RGSTR8CE G(G_out, CLK, DCDR[6], RST, MUX_4to1_OUT);
        RGSTR8CE H(H_out, CLK, DCDR[7], RST, MUX_4to1_OUT);

        AND2 A1(flag, INSTR[7], INSTR[6]);
        MUX2_1 M1(MUX_2to1_OUT, INSTR[2:0], INSTR[5:3], flag);
        MUX8_1 M2(MUX_8to1_OUT, A_out, B_out, C_out, D_out, E_out, F_out, G_out, H_out, MUX_2to1_OUT);

        ADDSUB_MACRO #(
                .DEVICE("VIRTEX6"), // Target Device: "VIRTEX5", "VIRTEX6", "SPARTAN6"
                .LATENCY(0), // Desired clock cycle latency, 0-2
                .WIDTH(8) // Input / output bus width, 1-48
                ) ADDSUB_MACRO_inst (
                .CARRYOUT(Car_out), // 1-bit carry-out output signal
                .RESULT(ADD_OUT), // Add/sub result output, width defined by WIDTH parameter
                .A(MUX_8to1_OUT), // Input A bus, width defined by WIDTH parameter
                .ADD_SUB(1'b1), // 1-bit add/sub input, high selects add, low selects subtract
                .B(A_out), // Input B bus, width defined by WIDTH parameter
                .CARRYIN(1'b0), // 1-bit carry-in input
                .CE(1'b1), // 1-bit clock enable input
                .CLK(CLK), // 1-bit clock input
                .RST(RST) // 1-bit active high synchronous reset
                );

        MUX4_1 M3(MUX_4to1_OUT, Load, MUX_8to1_OUT, ADD_OUT, MUX_8to1_OUT, INSTR[7:6]);

        DCDR3_8 D1(DCDR, INSTR[5:3]);

        RGSTR8CE OTPT(OUT, CLK, flag, RST, MUX_4to1_OUT);

endmodule
```

```
module RGSTR8CE(output reg [7:0] OUT, input CLK, input CE, input CLR, input [7:0] D);
	always @(negedge CLK or posedge CLR)
		begin
		if(CLR)
			OUT = 8'do;
		else
			if(CE)
				begin
					OUT = D;
				end
		end
endmodule

module MUX2_1(output reg [2:0] OUT, input [2:0] I0, input [2:0] I1, input S);
	always @(S or I0 or I1)
	begin
		case(S)
		1'b0:	OUT = I0;
		1'b1:	OUT = I1;
		default: OUT = 3'dx;
		endcase
	end
endmodule


//8 to 1 MUX
module MUX8_1 (output reg [7:0] OUT, input [7:0] I0, input [7:0] I1, input [7:0] I2, input [7:0] I3, input [7:0] I4, input [7:0] I5, input [7:0] I6, input [7:0] I7, input [2:0] S);
	always @(S or I0 or I1 or I2 or I3 or I4 or I5 or I6 or I7)
	begin
		case(S)
		3'b000:	OUT = I0;
		3'b001:	OUT = I1;
		3'b010:	OUT = I2;
		3'b011:	OUT = I3;
		3'b100:	OUT = I4;
		3'b101:	OUT = I5;
		3'b110:	OUT = I6;
		3'b111:	OUT = I7;
		default: OUT = 8'dx;
		endcase
	end
endmodule

module MUX4_1 (output reg [7:0] OUT, input [7:0] I0, input [7:0] I1, input [7:0] I2, input [7:0] I3, input [1:0] S);  //4 to 1 MUX
	always @(S or I0 or I1 or I2 or I3)
	begin
		case(S)
		2'd0: OUT = I0;			//INPUT LOAD into OUT
		2'd1: OUT = I1;			//MOVE Register(INSTR[2:0]) to Register(INSTR[5:3])
		2'd2: OUT = I2;			//ADD Register(INSTR[2:0]) with A to store in Register(INSTR[5:3])
		2'd3: OUT = I3;			//OUTPUT Register(INSTR[5:3])
		default: OUT = 8'dx;
		endcase
	end
endmodule

module DCDR3_8(output reg [7:0] OUT, input [2:0] IN);		//3 to 8 Decoder
	always @(IN)
	begin
		case(IN)
		3'd0: OUT = 8'b00000001;
		3'd1: OUT = 8'b00000010;
		3'd2: OUT = 8'b00000100;
		3'd3: OUT = 8'b00001000;
		3'd4: OUT = 8'b00010000;
		3'd5: OUT = 8'b00100000;
		3'd6: OUT = 8'b01000000;
		3'd7: OUT = 8'b10000000;
		default: OUT = 8'do;
		endcase
	end
endmodule
```

Corresponding testbench Code

```
`timescale 1ns / 1ps
module Basic_Processor_struct_tb();
        reg CLK, RST;
        reg [7:0] ld_ext, instr;
        wire [7:0] OUT;
        Basic_Processor_Struct stimulus(instr, CLK, RST, ld_ext, OUT);
        initial begin
                CLK = 1'b0;
                RST = 1'b1;
//              forever #10 CLK = ~CLK; //time period of clock is 20 time units
        end
        initial begin
                #50
                begin
                        RST = 1'b0;
                        ld_ext = 8'd15;         //load external value is 15
                        instr = 8'b00000000; //Load value of 15 into A
                        #10 CLK = ~CLK;
                        #10 CLK = ~CLK;
                end

                #20 instr = 8'b11000111;        //output value of A = 15                        -----       15
                #10 CLK = ~CLK;
                #10 CLK = ~CLK;

                #80 instr = 8'b01001000;//load value of A into B
                #10 CLK = ~CLK;
                #10 CLK = ~CLK;

                #60 instr = 8'b11001000; //output value of B = 15                               --- 15
                #10 CLK = ~CLK;
                #10 CLK = ~CLK;

                #80 instr = 8'b10000001; //add value of A to B and store in A
                #10 CLK = ~CLK;
                #10 CLK = ~CLK;

                #40 instr = 8'b11000000; //output value of A -- ----- 15+15 = 30               ------30
                #10 CLK = ~CLK;
                #10 CLK = ~CLK;

                #80 instr = 8'b10000001; //add value of A to B and store in A
                #10 CLK = ~CLK;
                #10 CLK = ~CLK;

                #40 instr = 8'b11000000; //output value of A -- ----- 30+15 = 45       ----45
                #10 CLK = ~CLK;
                #10 CLK = ~CLK;

        end
        initial $monitor("Time: ", $time, " and Output = %d", OUT); //Printing the Output
        initial #(1000) $finish ;

endmodule
```

Simulation



Discussion:

- The sample case handled here is loading the value of A and making it thrice using the LD 15 -> IN A -> OUT A -> MOV B A -> OUT B -> ADD A B -> OUT A -> ADD A B -> OUT A. So, for given ***input of 15***, ***Output*** value is ***45***.
- Initially, all the values are RESET so as to reset any floating/unknown values.
- ***After every instruction***, ***only one period of clock*** is given (nothing more nor less) so that any edge triggered components in the system complete only one operation. ***If less***, they ***wouldn't have moved to next state*** and ***if more***, they would have ***moved forward by more than 1 state***. Hence, the ***clock*** in this experiment is ***handled manually*** unlike previous experiments.
- In the testbench, the inputs are registers type and outputs are wire type since the registers should go on supplying the same value until given another value (so ***Input Port should have memory***) while the output keeps on changing according to any of the inputs (***Output Port has no memory***).
- In the simulated waveform, use the radix as unsigned decimal instead of binary (default) for easy viewing for decimal numbers.

DRIVE LINK:

Structural File link is *here* (this file consists of only the Verilog code)

Corresponding Testbench is *here* (this file consists of only the testbench code)

Both the files are separated to easily synthesize the RTL schematic and do any other analysis