Microprocessor Design using Verilog HDL

Objective:

- 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.

Operations:

- ADD R1 // Add value of R1 to A and update the value of A
- SUB R1 // Subtract the value of R1 to A and update the value of A
- MOV R1, R2 // Move the value of R2 to R1
- IN R1 // Input an 8-bit number to Register R1
- OUT R1 // Output the 8-bit number in Register R1

Code:

1) Microprocessor:

```
module MicroProc(clk, rst, inst, datain, out);
    input clk, rst;
    input[7:0] inst, datain;
    output[7:0] out;
    wire in76;
    wire[7:0] ld, oe;
    assign in76 = ~(inst[7] | inst[6]);
     RegisterBank SD1(clk, datain, inst, rst, oe, ld, out);
LdDec SD2(in76, inst[5:3], ld);
OEenc SD3(inst[2:0], oe);
        endmodule
```

A) Register Bank:

```
module RegisterBank(clk, datain, inst, rst, oe, ld, out);
input[7:0] datain, inst, oe, ld;
input clk, rst;
output reg[7:0] out;
reg[7:0] bus;
wire[7:0] outA, outB, outC, outD, outE, outF, lnA;
wire ldA;
always @(posedge clk) begin if(ld[0] == 1'b1) begin
 out <= bus; end else begin out <= 8'hxx;
 end
end
always @(*) begin if(oe[0] == 1'b1) begin
bus = datain;
end
else if(oe[1] == 1'b1) begin
bus = outA;
end
else if(oe[2] == 1'b1) begin
 bus = outB;
end
 else if(oe[3] == 1'b1) begin
 bus = outC;
end
 else if(oe[4] == 1'b1) begin
bus = outD;
end
else if(oe[5] == 1'b1) begin
bus = outE;
end
 else if(oe[6] == 1'b1) begin
bus = outF;
end
else begin
bus = 8'hxx;
end
end
ALU AddSub(outA, bus, ld[1], inst, lnA, ldA);
Register RA(clk, rst, lnA, ldA, outA);
Register RB(clk, rst, bus, ld[2], outB);
Register RC(clk, rst, bus, ld[3], outC);
Register RD(clk, rst, bus, ld[4], outD);
Register RE(clk, rst, bus, ld[5], outE);
Register RF(clk, rst, bus, ld[6], outF);
endmodule
```

i) ALU:

```
module ALU(A, Bus, ld1, inst, lnA, ldA);

input[7:0] A, Bus, inst;

input ld1;

output[7:0] lnA;

output ldA;

wire[7:0] Sum;

RipAdd uut(A, Bus, inst[5], Sum, Cout);

Mux2to1 M1(Sum[0], Bus[0], inst[6], lnA[0]);

Mux2to1 M2(Sum[1], Bus[1], inst[6], lnA[1]);

Mux2to1 M3(Sum[2], Bus[2], inst[6], lnA[2]);

Mux2to1 M4(Sum[3], Bus[3], inst[6], lnA[3]);

Mux2to1 M5(Sum[4], Bus[4], inst[6], lnA[4]);

Mux2to1 M6(Sum[5], Bus[5], inst[6], lnA[5]);

Mux2to1 M7(Sum[6], Bus[6], inst[6], lnA[6]);

Mux2to1 M8(Sum[7], Bus[7], inst[6], lnA[7]);

assign ldA = (ld1 | (inst[7]^inst[6]));

endmodule
```

a) Adder-Subtractor:

```
module RipAdd(A, B, in, Sum, Cout);
input[7:0] A, B; input in; output[7:0] Sum;
output Cout; wire[6:0] Cin; wire[7:0] C;
assign C[0] = B[0]^in;
assign C[1] = B[1]^in;
assign C[2] = B[2]^in;
assign C[3] = B[3]^in;
assign C[4] = B[4]^in;
assign C[5] = B[5]^in;
assign C[6] = B[6]^in;
assign C[7] = B[7]^in;
FullAdd utt1(A[0], C[0], in, Sum[0], Cin[0]);
FullAdd utt2(A[1], C[1], Cin[0], Sum[1], Cin[1]);
FullAdd utt3(A[2], C[2], Cin[1], Sum[2], Cin[2]);
FullAdd utt4(A[3], C[3], Cin[2], Sum[3], Cin[3]);
FullAdd utt5(A[4], C[4], Cin[3], Sum[4], Cin[4]);
FullAdd utt6(A[5], C[5], Cin[4], Sum[5], Cin[5]);
FullAdd utt7(A[6], C[6], Cin[5], Sum[6], Cin[6]);
```

```
FullAdd utt8(A[7], C[7], Cin[6], Sum[7], Cout);
endmodule
```

a1) Full Adder:

```
module FullAdd(A, B, Cin, S, Cout);

input A, B, Cin;

output S, Cout;

wire t1, t2, t3;

xor G1(t1, A, B);

and G2(t2, A, B);

xor G3(S, t1, Cin);

and G4(t3, t1, Cin);

or G5(Cout, t2, t3);

endmodule
```

b) MUX:

```
module Mux2to1(in1, in0, sel, out);

input in0, in1, sel;

output out;

wire t1, t2, t3;

and G1(t1, sel, in1);

not G2(t2, sel);

and G3(t3, t2, in0);

or G4(out, t1, t3);
```

ii) Register:

```
module Register(clk, rst, in, ld, out);

input clk, rst, ld;

input[7:0] in;

output reg[7:0] out;

always @(posedge clk) begin

if(rst == 1'b0) begin

out <= 8'b00;

end
```

```
else begin
if(ld == 1'b1) begin
out <= in;
end
else begin
out <= out;
end
end
end
endmodule
```

B) Load Decoder:

```
module LdDec(in76, in, ld);
input in76; input[2:0] in;
output[7:0] ld;
assign ld[0] = (in76)&(~in[2])&(~in[1])&(~in[0]);
assign ld[1] = (~in76) | ((in76)&(~in[2])&(~in[1])&(in[0]));
assign ld[2] = (in76)&(~in[2])&(in[1])&(~in[0]);
assign ld[3] = (in76)&(~in[2])&(in[1])&(in[0]);
assign ld[4] = (in76)&(in[2])&(~in[1])&(~in[0]);
assign ld[5] = (in76)&(in[2])&(~in[1])&(in[0]);
assign ld[6] = (in76)&(in[2])&(in[1])&(~in[0]);
endmodule
```

C) Output Enable Encoder:

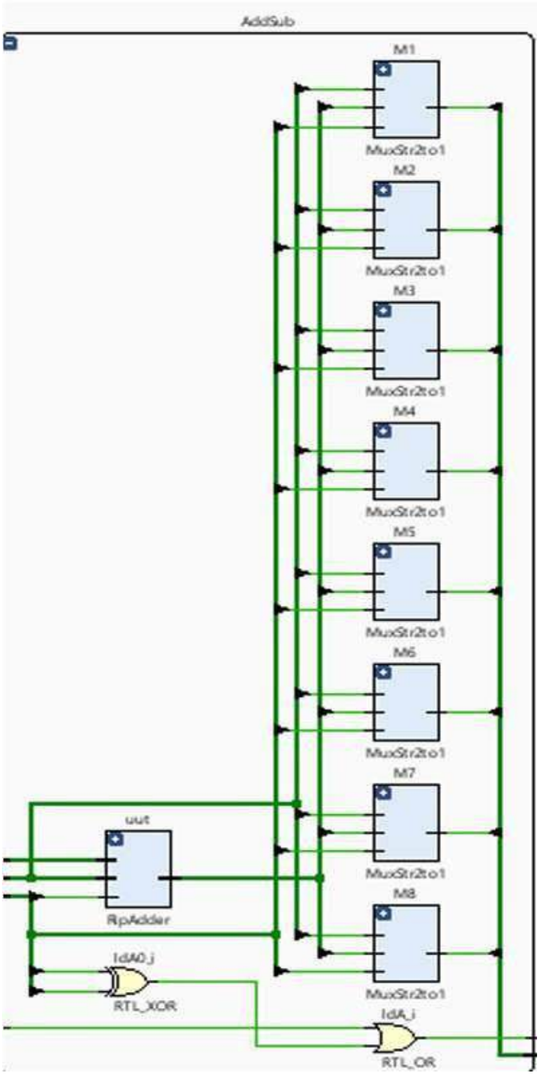
```
module OEenc(in, oe);
input[2:0] in;
output[7:0] oe;
assign oe[0] = (~in[2])&(~in[1])&(~in[0]);
assign oe[1] = (~in[2])&(~in[1])&(in[0]);
assign oe[2] = (~in[2])&(in[1])&(~in[0]);
assign oe[3] = (~in[2])&(in[1])&(in[0]);
assign oe[4] = (in[2])&(~in[1])&(~in[0]);
assign oe[5] = (in[2])&(~in[1])&(in[0]);
assign oe[6] = (in[2])&(in[1])&(~in[0]);
endmodule
```

TEST BENCH:

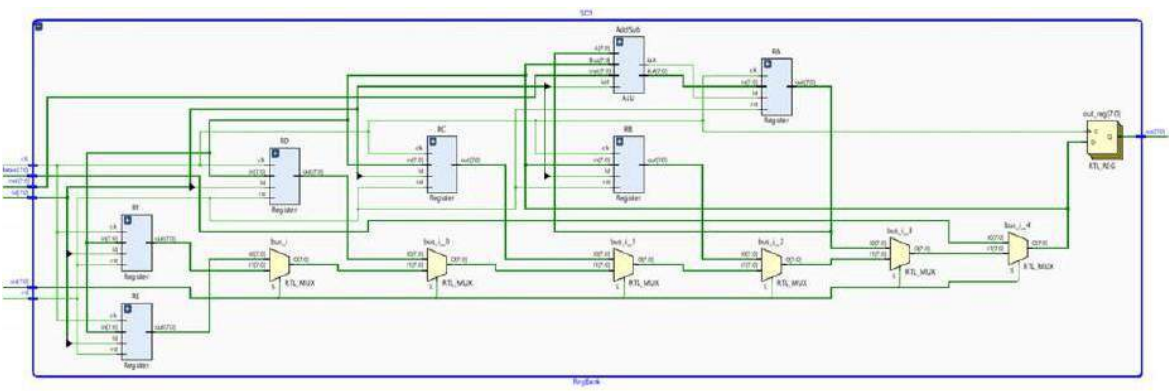
```
module test_bench();
reg clk, rst;
reg[7:0] inst, datain;
wire[7:0] out;
parameter IN = 2'b00, MOV = 2'b00, OUT = 5'b00000, ADD = 5'b01000, SUB = 5'b01100, A =
3'b001, B = 3'b010, C = 3'b011, D = 3'b100, E = 3'b101, F = 3'b110;
pro SD(clk, rst, inst, datain, out);
initial begin
clk = 1'b0;
forever #2 clk = ~clk;
end
initial begin
rst = 1'b0; inst = {IN, B, 3'b000};
datain = 8'h32;
#5 rst = 1'b1;
#8 datain = 8'h32; inst = {IN, B, 3'b000};
#4 datain = 8'h43; inst = {IN,C ,3'b000};
#4 inst = {MOV, A, B};
#4 inst = {OUT, A};
#4 inst = {ADD, C};
#4 inst = {IN, D, 3'b000}; datain = 8'h44;
#4 inst = {OUT, D};
#4 inst = {SUB, D};
#4 inst={MOV,E,A};
#4 inst = {OUT, E};
#20
$finish;
end
endmodule\
```

1) Schematic:

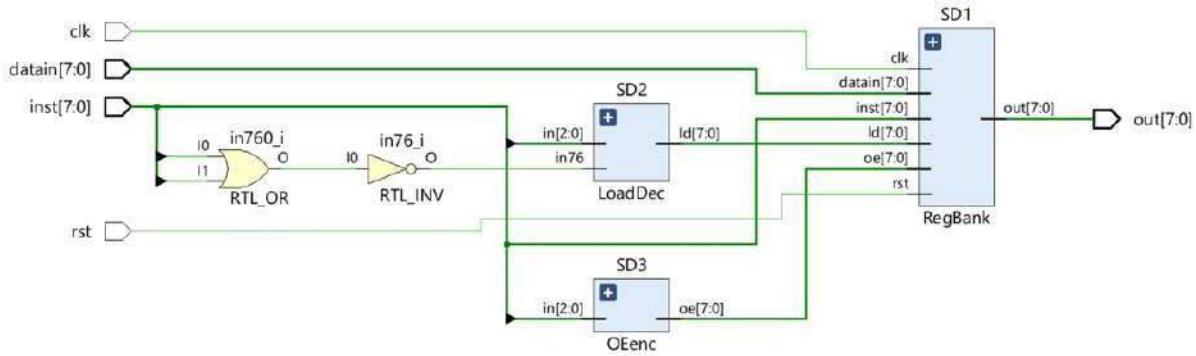
A) ALU:



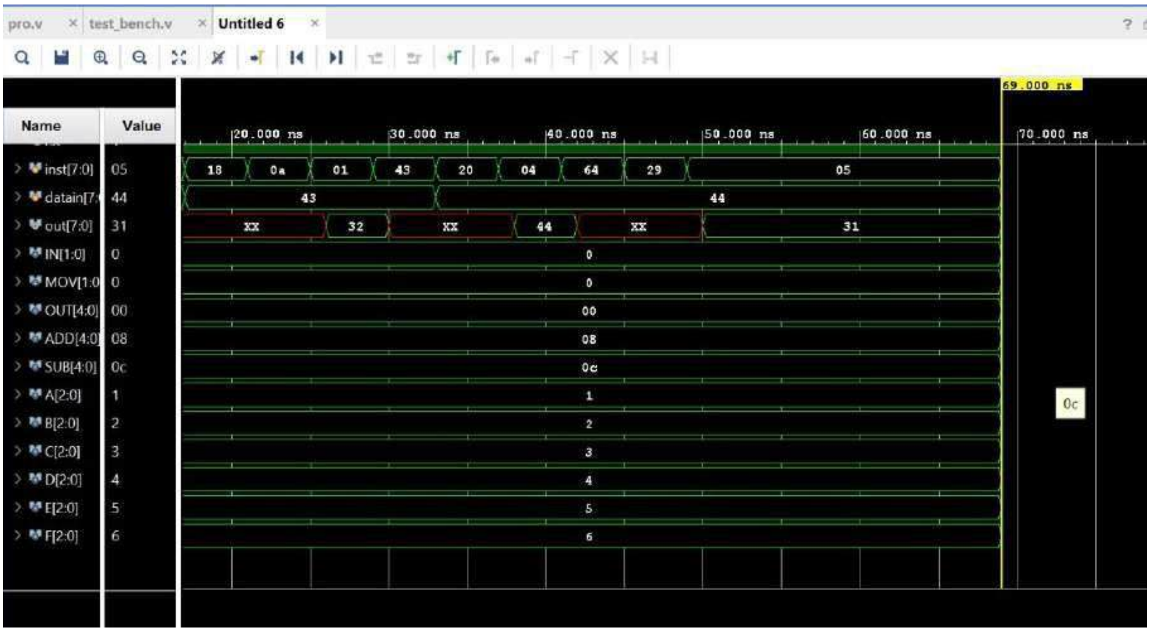
B) Register Bank:



C) Main Architecture:



Output Waveform:



DISCUSSION:

- A basic microprocessor functions as the brain of a computer, executing fundamental commands and handling mathematical and logical tasks.
- Its operations are dictated by 8-bit instructions, with the first two bits determining whether it performs input/output/movement actions or addition/subtraction calculations. The subsequent 6 bits specify the operations for loading and outputting data.
- The Register Bank is a repository of registers and the Arithmetic Logic Unit (ALU). The ALU handles addition and subtraction tasks. Registers are interconnected with the bus, a pathway for data transmission.
- Activating the output enable facilitates data transfer from a register or input to the bus, while activating the load initiates the transfer of data from the bus to a register. During output enable activation, data remains on the bus throughout the clock cycle, whereas during load activation, data is only loaded at the positive edge of the clock signal.