EE513 LAB
Experiment 7

Design an HDL-based 32-bit processor

Submitted by,

Pooja Kumari
ROLL NO-234102318

Contents

List of Figures

List of Tables

1 Objective :

Design an HDL-based 32-bit processor (Instruction decode (Instruction de- coder + controller) + Instruction execute (ALU)+register write (Register Bank)) which executes the following 10 instructions:
ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND

Perform post-synthesis simulations for a basic 32-bit processor which executes the aforementioned instructions. The processor will basically have the following modules:
a. A 32-bit register bank which will have 32 registers and each register can store 32-bit data.
b. An instruction decoder module which will slice the 32-bit instructions into the corresponding fieldsopcode, func and rs1, rs2 and rd values
c. A controller module which will generate a corresponding signal for the respective operations depending func and opcode.
d. 32-bit ALU for processing the data present in rs1 and rs2 registers.
e. Register write to store the result back into rd register in the register bank.
NOTE:
1. Initialize the 32 registers present in the register bank, before executing the instructions.
2. Perform ALU operations on the data present in 'rs1' and 'rs2' registers. The result is stored in 'rd' register.

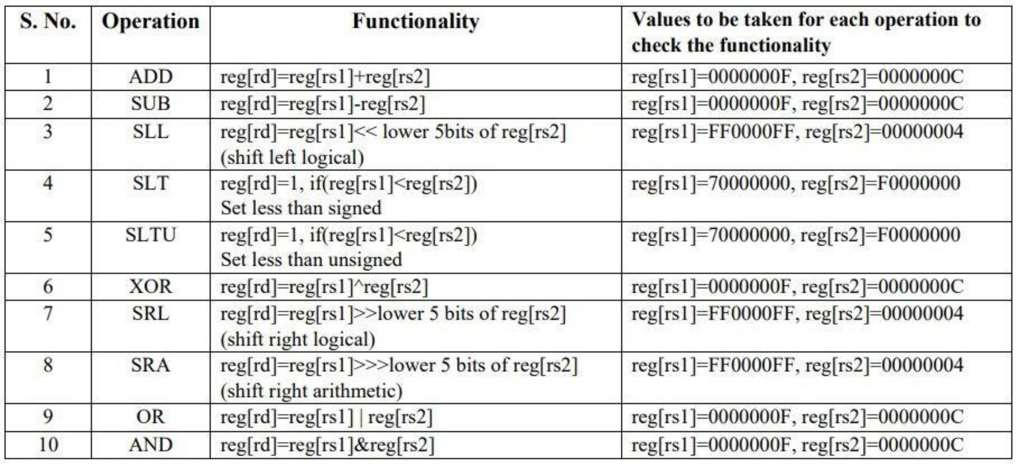
2 Theory :

2.1 Processor :

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (1/0) operations specified by the instructions. Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit (CU), distinguishing these core elements of a computer from external components such as main memory and I/O circuitry.

A microprocessor is a computer processor which incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits. The microprocessor is a multipurpose, clock driven, register based programmable electronic device which accepts digital or binary data as input, processes it according to instructions stored in its memory, and provides results as output.

2.2 Operations performed by ALU:



S. No.	Operation	Functionality	Values to be taken for each operation to check the functionality
1	ADD	reg[rd]=reg[rs1]+reg[rs2]	reg[rs1]=0000000F, reg[rs2]=0000000C
2	SUB	reg[rd]=reg[rs1]-reg[rs2]	reg[rs1]=0000000F, reg[rs2]=0000000C
3	SLL	reg[rd]=reg[rs1]<< lower 5bits of reg[rs2] (shift left logical)	reg[rs1]=FF0000FF, reg[rs2]=00000004
4	SLT	reg[rd]=1, if(reg[rs1]<reg[rs2]) Set less than signed	reg[rs1]=70000000, reg[rs2]=F0000000
5	SLTU	reg[rd]=1, if(reg[rs1]<reg[rs2]) Set less than unsigned	reg[rs1]=70000000, reg[rs2]=F0000000
6	XOR	reg[rd]=reg[rs1]^reg[rs2]	reg[rs1]=0000000F, reg[rs2]=0000000C
7	SRL	reg[rd]=reg[rs1]>>lower 5 bits of reg[rs2] (shift right logical)	reg[rs1]=FF0000FF, reg[rs2]=00000004
8	SRA	reg[rd]=reg[rs1]>>>lower 5 bits of reg[rs2] (shift right arithmetic)	reg[rs1]=FF0000FF, reg[rs2]=00000004
9	OR	reg[rd]=reg[rs1] | reg[rs2]	reg[rs1]=0000000F, reg[rs2]=0000000C
10	AND	reg[rd]=reg[rs1]&reg[rs2]	reg[rs1]=0000000F, reg[rs2]=0000000C

Figure 1: Operations performed by ALU

2.3 Instruction set specification:

31 25	24 20	19 15	14 **FUNC** 12	11 7	6 **OPCODE** 0	
0000000	rs2	rs1	000	rd	0000001	ADD
0000000	rs2	rs1	001	rd	0000001	SUB
0000000	rs2	rs1	000	rd	0000011	SLL
0000000	rs2	rs1	001	rd	0000011	SRL
0000000	rs2	rs1	010	rd	0000011	SRA
0000000	rs2	rs1	000	rd	0000111	SLT
0000000	rs2	rs1	001	rd	0000111	SLTU
0000000	rs2	rs1	000	rd	0001111	XOR
0000000	rs2	rs1	001	rd	0001111	OR
0000000	rs2	rs1	010	rd	0001111	AND

Figure 2: Instruction set specification

3 32b processor :

We have implemented a 3 stage pipelined processor. The three stages are: Instruction Decode, Control and ALU.
Instruction decode module is used to separate out the rs1 number, rs2 number and rd number from the instruction bits.
The control module as given in the specification generates a control signal based on the function and opcode for each of the operation.
The ALU module does the operation based on the control signal.
We have designed a separate module for the register Bank. It has 4 input ports and 2 output ports.

3.1 Design :

```
module procsn32(
    input  clock,
    input [31:0] inst,
    output  [31:0]  result
    );

    wire [4:0] rs1_num, rs2_num, rd_num;
    reg [4:0]  rd_num_r,  rd_num_rr;
    wire [3:0] ctrl;
    wire [31:0]   rs1, rs2;
    wire [31:0] inst_r,outP;

    //Instruction  Decode  Stage
    inst_dec  ID  (inst,  rs1_num,  rs2_num,  rd_num,  inst_r,  clock);
    //Control  Signal  Stage
    control CTRL (inst_r, ctrl, clock);

    always@(posedge  clock)  begin
    rd_num_r <= rd_num;
    rd_num_rr <= rd_num_r;
    end

    //ALU  Stage
    ALU  ALU  (rs1,  rs2,  ctrl,  outP,  clock);
```

```
     //Register  Bank  module
     regBank regB(clock, rs1_num, rs2_num, rd_num_rr, outP, rs1, rs2);

     assign result = outP;

endmodule

module regBank(input clock, input [4:0] rs1_num, rs2_num, dataW_num,
                 input [31:0] dataW, output reg [31:0] data1, data2);
reg [31:0] regBank [0:31];    //register_bank
always@(posedge clock) begin
data1 <= regBank[rs1_num];
data2 <= regBank[rs2_num];
regBank[dataW_num] <= dataW;
end
endmodule

module inst_dec(input [31:0] inst,
     output [4:0] rs1_num, rs2_num, rd_num,
     output reg [31:0] inst_r, input clock);
always@(posedge clock) inst_r = inst;

assign  rs1_num  =  inst_r[19:15];  //rs1
assign  rs2_num  =  inst_r[24:20];  //rs2
assign rd_num = inst_r[11:7];  //rd

endmodule

module control(input [31:0] inst, output reg [3:0] ctrl, input clock);
parameter    ADD = 10’b0000001, SUB = 10’b0010001, SLL = 10’b0000011,
             SRL = 10’b0010011, SRA = 10’b0100011, SLT = 10’b0000111,
             SLTU= 10’b0010111, XOR = 10’b0001111, OR  = 10’b0011111,
             AND = 10’b0101111;        //func_opcode
     always@(posedge clock) begin
     case({inst[14:12],inst[3:0]})     //func_opcode
             ADD : ctrl = 4’h1;
             SUB : ctrl = 4’h2;
             SLL : ctrl = 4’h3;
```

```
            SRL : ctrl = 4’h4;
            SRA : ctrl = 4’h5;
            SLT : ctrl = 4’h6;
            SLTU: ctrl = 4’h7;
            XOR : ctrl = 4’h8;
            OR  : ctrl = 4’h9;
            AND : ctrl = 4’ha;
            default : ctrl = 4’h0;
    endcase
    end
endmodule


module ALU(input [31:0] rs1, input [31:0] rs2, input [3:0] ctrl,
            output reg [31:0] outP, input clock);
always@(posedge clock) begin//EX
case(ctrl) //func_opcode
4’h1 : outP = rs1 + rs2 ;
4’h2 : outP = rs1 – rs2 ;
4’h3 : outP = rs1 << rs2[4:0] ;
4’h4 : outP = rs1 >> rs2[4:0] ;
4’h5 : outP = $signed(rs1) >>> rs2[4:0] ;
4’h6 : outP = ($signed(rs1) < $signed(rs2))
              ? 32’h00000001 : 32’h00000000 ;
4’h7 : outP = (rs1 < rs2)
              ? 32’h00000001 : 32’h00000000 ;
4’h8 : outP = rs1 ^ rs2 ;
4’h9 : outP = rs1 | rs2 ;
4’ha : outP = rs1 & rs2 ;
default : outP = 32’hxxxxxxxx ;
endcase
end
endmodule
```

3.2 Test bench ;

```
module testbench(

    );

    reg clock;
    reg [31:0] inst;

    always #10 clock = ~clock;

    procsn32 DUT(clock, inst);

    initial begin
    clock <= 1;
        DUT.regB.regBank[0] = 0;
        DUT.regB.regBank[1] = 32’h0000000f;
        DUT.regB.regBank[2] = 32’h0000000c;
        DUT.regB.regBank[3] = 32’hff0000ff;
        DUT.regB.regBank[4] = 32’h00000004;
        DUT.regB.regBank[5] = 32’h70000000;
        DUT.regB.regBank[6] = 32’hf0000000;

    #20 inst = {7’h00,5’h02,5’h01,3’h0,5’h0a,7’h01};     //ADD
    #20 inst = {7’h00,5’h02,5’h01,3’h1,5’h0b,7’h01};     //SUB
    #20 inst = {7’h00,5’h04,5’h03,3’h0,5’h0c,7’h03};     //SLL
    #20 inst = {7’h00,5’h04,5’h03,3’h1,5’h0d,7’h03};     //SRL
    #20 inst = {7’h00,5’h04,5’h03,3’h2,5’h0e,7’h03};     //SRA
    #20 inst = {7’h00,5’h06,5’h05,3’h0,5’h0f,7’h07};     //SLT
    #20 inst = {7’h00,5’h06,5’h05,3’h1,5’h10,7’h07};     //SLTU
    #20 inst = {7’h00,5’h02,5’h01,3’h0,5’h11,7’h0f};     //XOR
    #20 inst = {7’h00,5’h02,5’h01,3’h1,5’h12,7’h0f};     //OR
    #20 inst = {7’h00,5’h02,5’h01,3’h2,5’h13,7’h0f};     //AND
    end

endmodule
```

In the test bench instructions have been given in a way such that each field is distinctly visible and it is easy to separate each instruction. The register bank is initially storing the values that are needed for computation.

3.3 RTL Schematic :

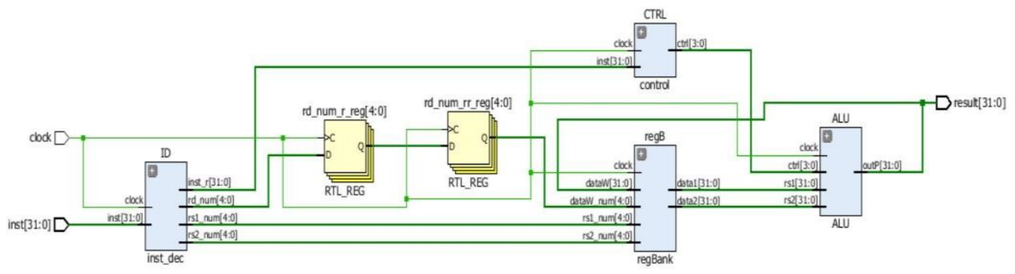


Figure 3: Schematic of the processor

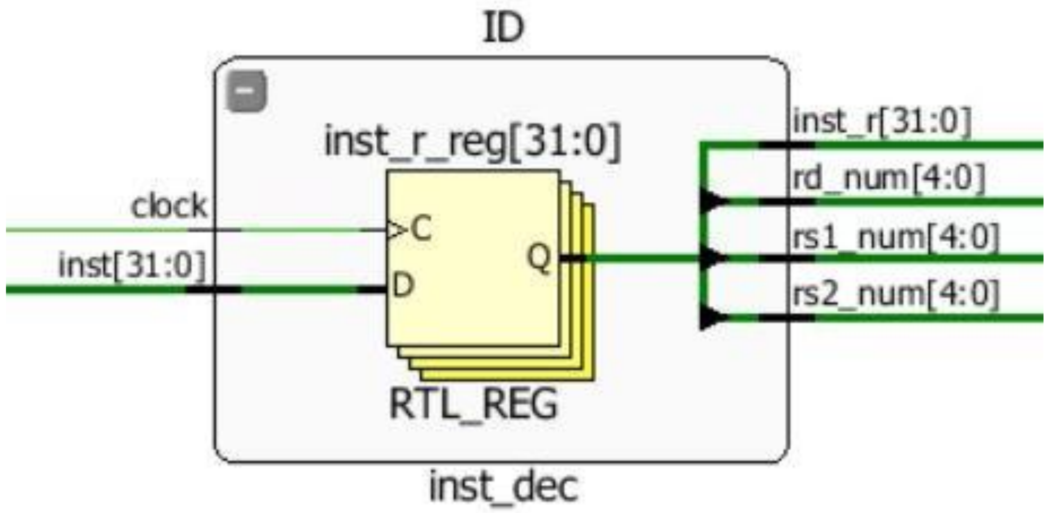


Figure 4: Schematic of Instruction Decode

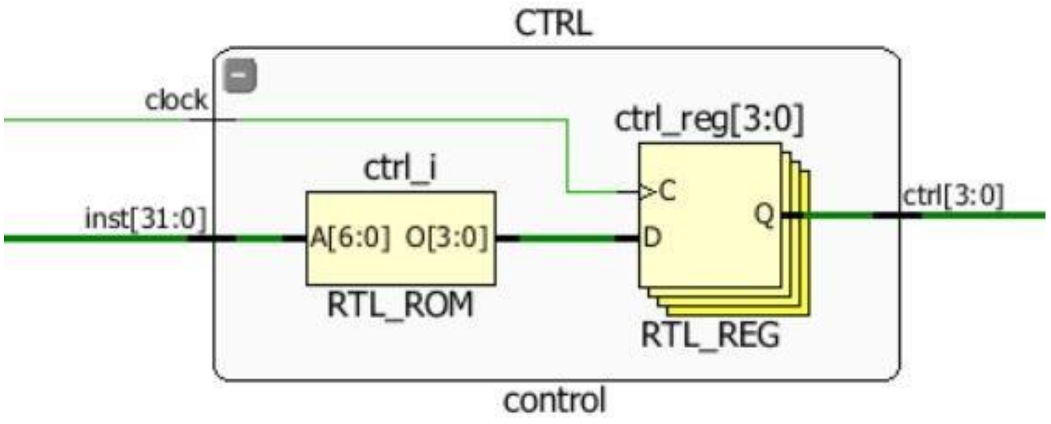


Figure 5: Schematic of Control

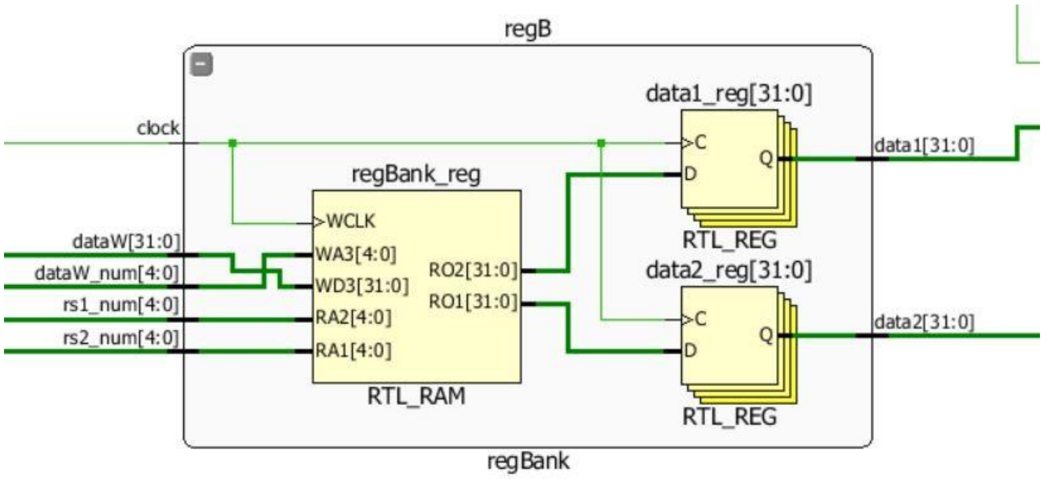


Figure 6: Schematic of Register Bank

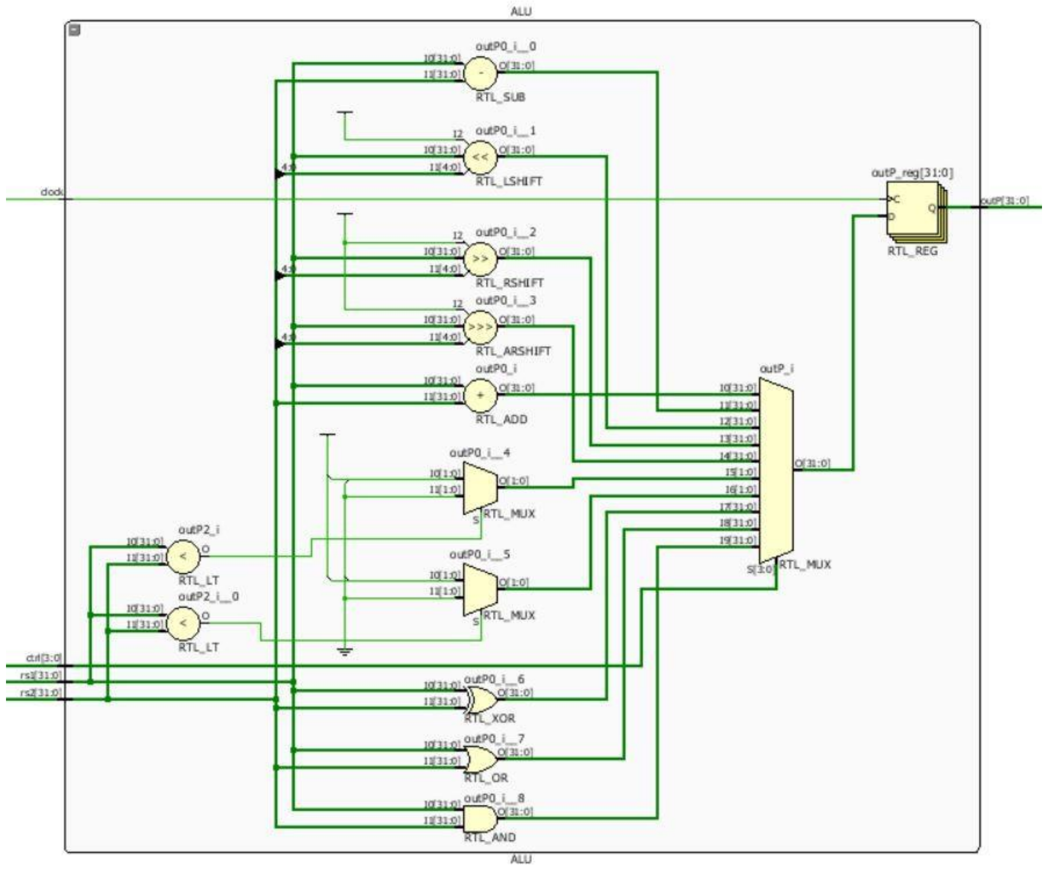


Figure 7: Schematic of ALU

3.4 Simulation :

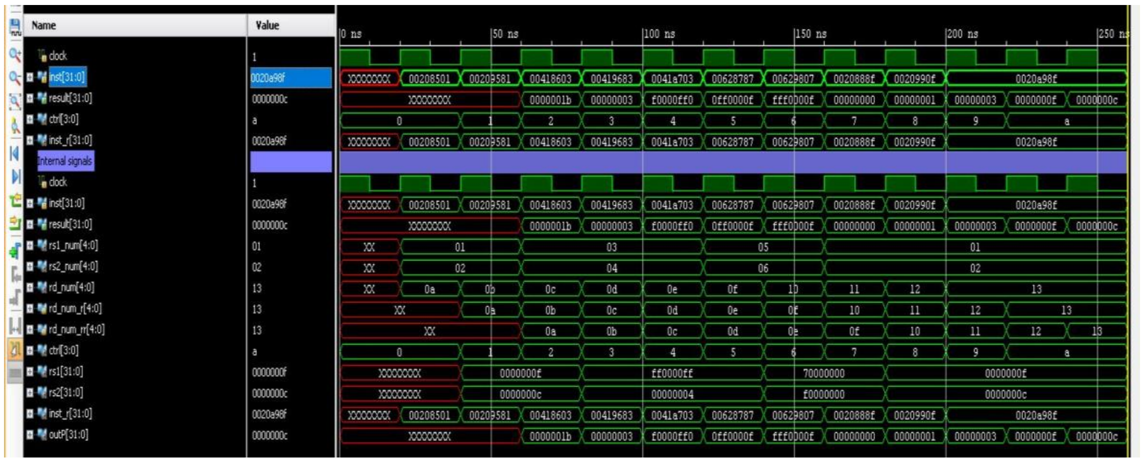
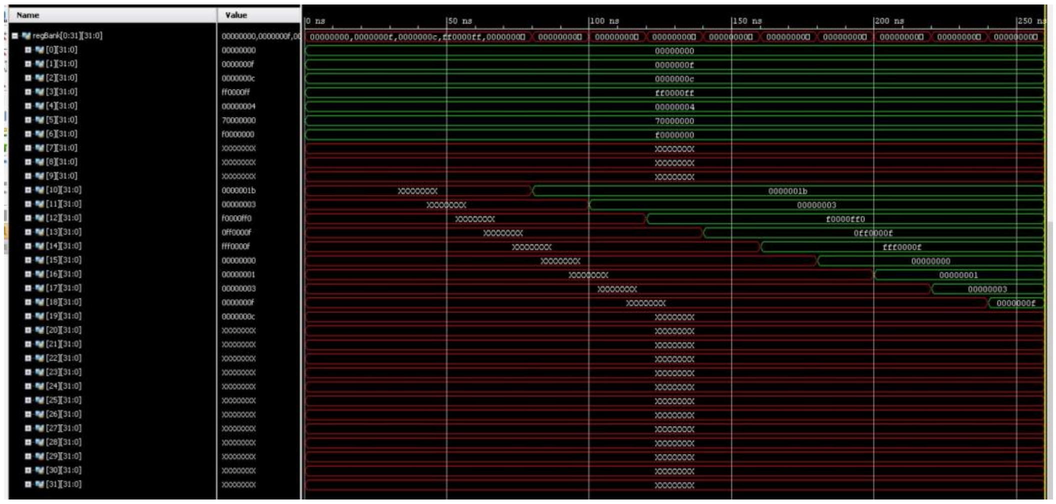


Figure 8: Simulation waveform of the processor

The functionality is as expected from the specification. All the given instruc- tions have been fed serially and the corresponding output is observed. The output is matching with theoretical calculation. The outP signal and the result signal are delayed because of the pipelining in the processor.



Name	Value
regBank[0:31][31:0]	00000000,0000000f,00
[0][31:0]	00000000
[1][31:0]	0000000f
[2][31:0]	0000000c
[3][31:0]	ff0000ff
[4][31:0]	00000004
[5][31:0]	70000000
[6][31:0]	f0000000
[7][31:0]	XXXXXXXX
[8][31:0]	XXXXXXXX
[9][31:0]	XXXXXXXX
[10][31:0]	0000001b
[11][31:0]	00000003
[12][31:0]	f0000ff0
[13][31:0]	0ff0000f
[14][31:0]	fff0000f
[15][31:0]	00000000
[16][31:0]	00000001
[17][31:0]	00000003
[18][31:0]	0000000f
[19][31:0]	0000000c
[20][31:0]	XXXXXXXX
[21][31:0]	XXXXXXXX
[22][31:0]	XXXXXXXX
[23][31:0]	XXXXXXXX
[24][31:0]	XXXXXXXX
[25][31:0]	XXXXXXXX
[26][31:0]	XXXXXXXX
[27][31:0]	XXXXXXXX
[28][31:0]	XXXXXXXX
[29][31:0]	XXXXXXXX
[30][31:0]	XXXXXXXX
[31][31:0]	XXXXXXXX

Figure 9: Simulation waveform of the values in the register Bank

The register bank is initially storing the values that are needed for computation. The result values are stored in different register numbers as each instruction is processed.

4 Observation Table :

The values have been found after synthesis of the corresponding designs. I have done the experiment on Vivado 2014.1. The LUTs and Flops have been found from the utilization report. The delay has been found from the timing report and the power has been found from the power report.

	LUTs	Flip-Flops	Power (W)	Delay (ns)
Processor	382	68	0.164	1.156

Table 1: Observation table for Processor

5 Results :

- We have designed the circuits as mentioned.
- The functionality of our design have been verified according to the specification provided. The functionality are showing as expected.
- The different parameters of the design such as LUTs, delay and power have been calculated from the synthesis and tabulated.