UNIVERSITY OF MORATUWA, SRI LANKA

Faculty of Engineering

Department of Electronic and Telecommunication Engineering

Semester 5 (Intake 2020)

EN3021 DIGITAL SYSTEM DESIGN

Disanayaka R.M.R.H

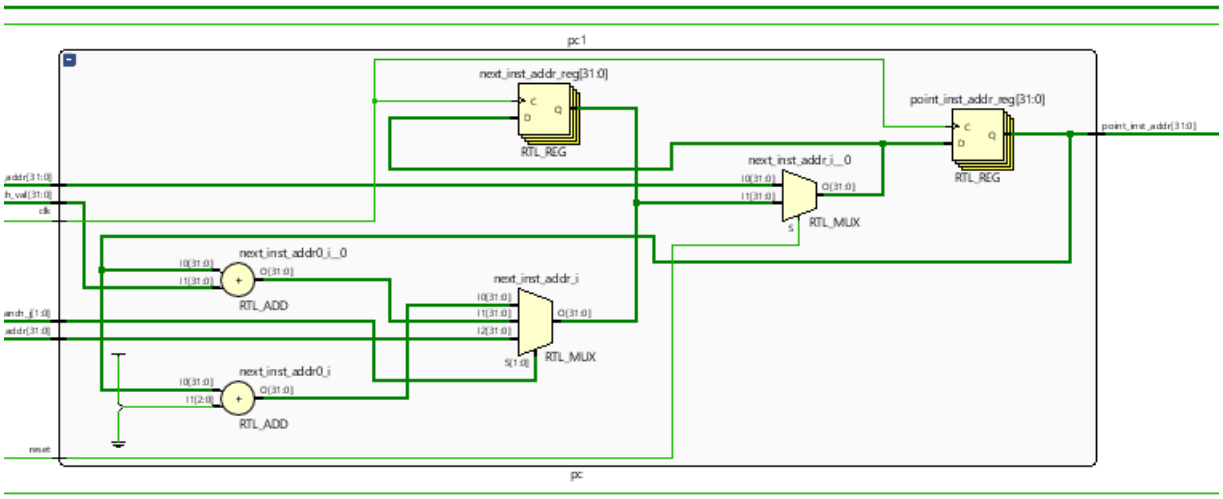
200134R

Contents

Introduction

- This is an implementation of RV32I architecture.
- Implementation was done step by step.
- First of all, modules were implemented with the required control signal to support the instruction set.
- They were independently tested using testbenches for each module.
- Finally all modules were interconnected using data path and tested.

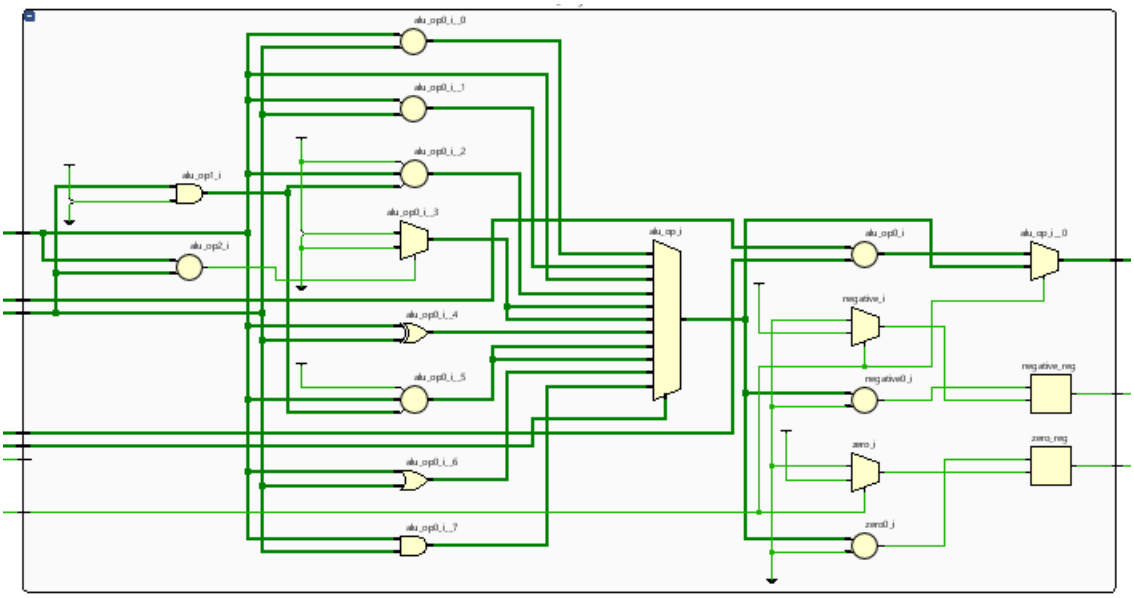
Program Counter



RTL view of the program counter is given above. System Verilog code implementation of the above Module is given below.

```
module PC(
    input logic [31:0] base_addr,branch_val,jump_addr,
    input logic [1:0] is_branch_j,
    input logic reset,clk,

    output logic  [31:0]  point_inst_addr
     );

    logic  [31:0]  next_inst_addr;

    always_comb
      unique case(is_branch_j)
        'b10 : next_inst_addr = jump_addr;
        'b01 : next_inst_addr = point_inst_addr + branch_val;
        'b00 : next_inst_addr = point_inst_addr + 32'd4;
        //default : next_inst_addr = 1;
      endcase
    always @ (posedge clk  )
    begin
        if (reset) begin
         next_inst_addr = base_addr;
         point_inst_addr = next_inst_addr;
        end
        else
        point_inst_addr = next_inst_addr;
    end

endmodule
```

Arithmetic and Logical Unit

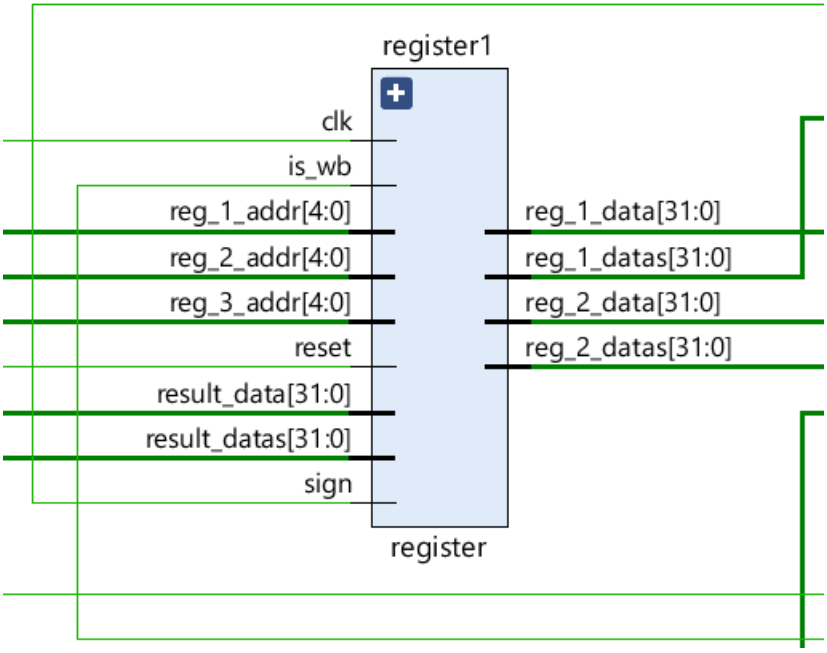


RTL view of the ALU is given above which supports operations for RV32I. System Verilog code implementation of the above Module is given below.

```
module alu_design #(
       parameter WIDTH=32 , W_ALU_SEL=4
    )(
    input logic  [WIDTH-1:0] alu_ip_1,alu_ip_2,
    input logic signed [WIDTH-1:0]  alu_ip_1s,alu_ip_2s,
    input logic  [W_ALU_SEL-1:0] alu_sel,
    input logic clk,
    output logic [WIDTH-1:0] alu_op,
    output logic zero,negative,
    input logic sign
    );

   always_comb begin

    if(sign)begin

        unique case (alu_sel)
            'b0000 : alu_op = alu_ip_1s + alu_ip_2s; //add
        endcase
        end
    else begin
```

```
else begin

unique case (alu_sel)
'b0111 : alu_op = alu_ip_1 & alu_ip_2; //and
'b0110 : alu_op = alu_ip_1 | alu_ip_2; //or
'b1101 : alu_op = alu_ip_1 >> (alu_ip_2%32); //sra
'b0101 : alu_op = alu_ip_1 >> (alu_ip_2%32); //srl !
'b0100 : alu_op = alu_ip_1 ^ alu_ip_2; //xor
'b0011 : alu_op = (alu_ip_1<alu_ip_2)? 1 : 0; //sltu !
'b0010 : alu_op = (alu_ip_1<alu_ip_2)? 1 : 0; //slt !
'b0001 : alu_op = alu_ip_1 << (alu_ip_2%32) ; //sll !
'b1000 : alu_op = alu_ip_1 - alu_ip_2;  //sub
'b0000 : alu_op = alu_ip_1 + alu_ip_2; //add


    default alu_op = alu_ip_1;
endcase
 zero = (alu_op==0);
 negative = (alu_op<0);
//assign is_wb =1;
end
end
```

Registers

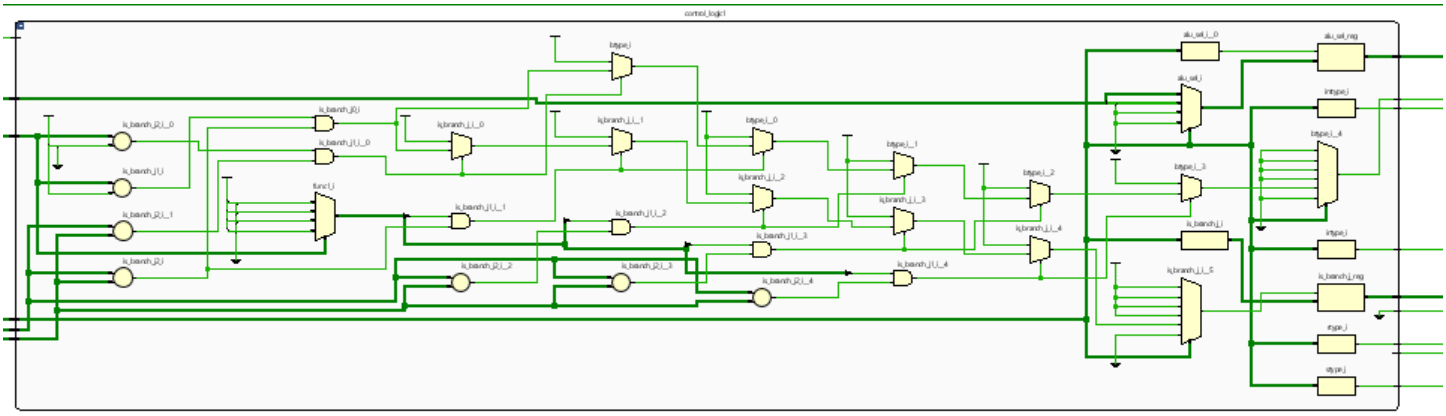


32 registers were defined while 2 of them become source registers for most instructions and one will become the destination as given in the instruction. Important parts of the code are given below.

```
module register #(
    parameter WIDTH = 5 // has only 32 registers,therefore 5 bits enough to represent them
 )(
    input logic [4:0] reg_1_addr,reg_2_addr,reg_3_addr,
    input logic [31:0] result_data,
    input logic signed [31:0] result_datas,
    input logic reset,clk,is_wb,sign,
    output logic [31:0] reg_1_data,reg_2_data,
    output logic signed [31:0] reg_1_datas,reg_2_datas
    );

    reg signed [31:0] reg1;
    reg signed [31:0] reg2;
    reg signed [31:0] reg3;


 always_comb
 unique case (reg_1_addr)
     'b00000 : reg1 = x0;
     'b00001 : reg1 = x1;
     'b00010 : reg1 = x2;
     'b00011 : reg1 = x3;
     'b00100 : reg1 = x4;
     'b00101 : reg1 = x5;
     'b00110 : reg1 = x6;
     'b00111 : reg1 = x7;
     'b01000 : reg1 = x8;
     'b01001 : reg1 = x9;
     'b01010 : reg1 = x10;
     'b01011 : reg1 = x11;
   end
   else if (is_wb) begin
       reg_1_data = reg1;
       reg_2_data = reg2;
       //reg3 = result_data;
      // wb_done = 1;
   end
   else begin
       reg_1_data = reg1;
       reg_2_data = reg2;
      // wb_done = 0;
   end
```

```
always @ (posedge clk  ) begin
if(sign)begin
    reg3 = result_datas;
end
else begin
    reg3 = result_data;
end

    unique case (reg_3_addr)
        'b00000 : x0 = reg3;
        'b00001 : x1 = reg3;
        'b00010 : x2 = reg3;
```

Controller

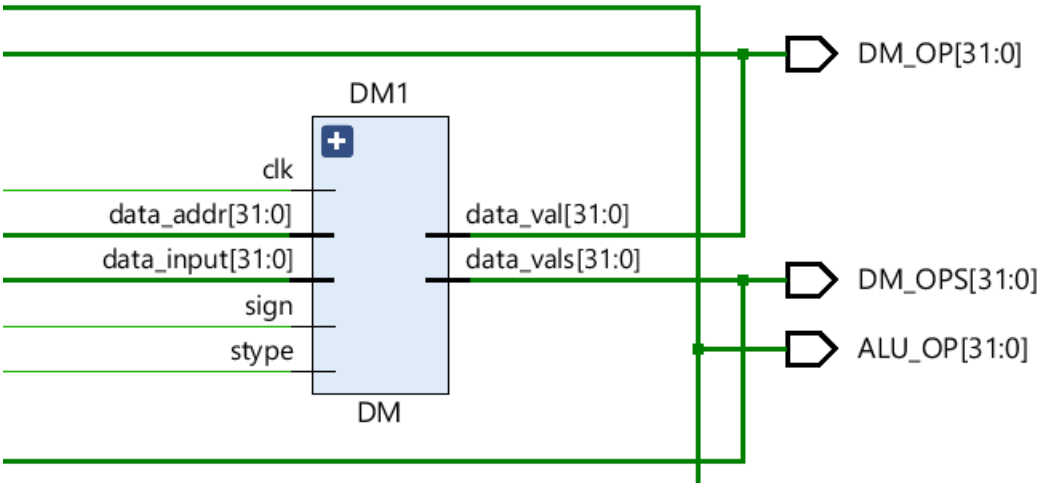


This outputs the control signals according to the instructions.

```
always_comb begin

    rtype = 0;
    irtype = 0;
    imtype = 0;
    stype = 0;
    btype = 0;
    is_wb = 0;

    case (opcode)

        'b0110011 : begin
            rtype = 1; // R+R type
            alu_sel = {func[8],func[2:0]};
            is_branch_j = 'b00;
            //sign = 1;
         end

        'b0010011 : begin
            irtype = 1; // I+R type
            is_branch_j = 'b00;
            if(func[2:0] != 101) begin
                alu_sel = {0,func[2:0]};
            end
            else begin
                alu_sel = {func[8],func[2:0]};
            end
         end
```

```
        'b1100011 :begin
            //btype = 1;
            if((func1== 'b000) && (rs1==rs2)) begin
                btype = 1; // Branch type
                is_branch_j = 'b01;
             end
             else if((func1 == 'b001) && (rs1!=rs2)) begin
                btype = 1; // Branch type
                is_branch_j = 'b01;
             end
             else if((func1 == 'b100) && (rs1<rs2)) begin
                btype = 1; // Branch type
                is_branch_j = 'b01;
             end
             else if((func1 == 'b101) && (rs1>=rs2)) begin
                btype = 1; // Branch type
                is_branch_j = 'b01;
             end
             else if((func1 != 'b110) && (rs1<=rs2)) begin
                btype = 1; // Branch type
                is_branch_j = 'b01;
             end
             else if((func1 != 'b111) && (rs1>=rs2)) begin
                btype = 1; // Branch type
                is_branch_j = 'b01;
             end
             else begin
                btype = 0;
             end

    end
```

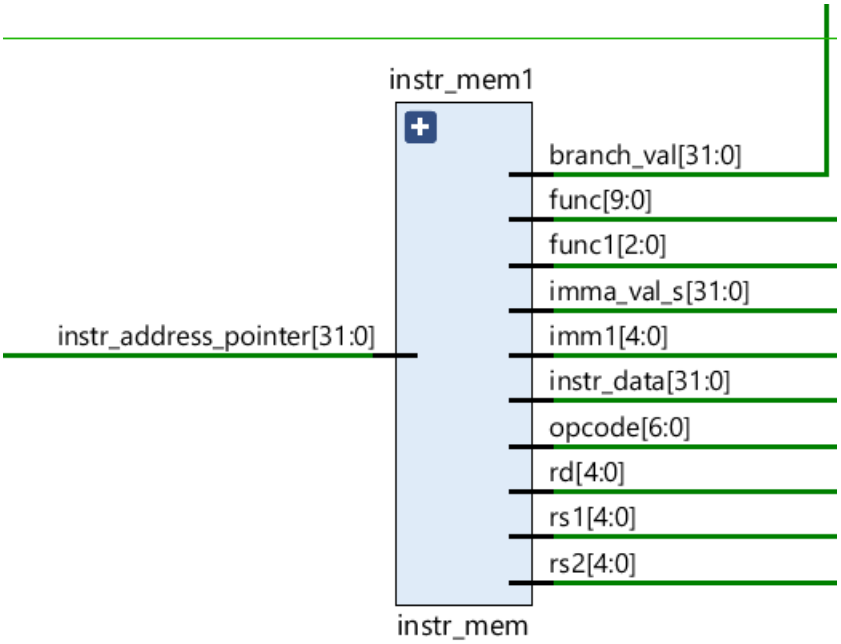
Data Memory




```
module DM(
    input logic [31:0] data_addr,data_input,
    output logic  [31:0] data_val,
    output logic signed [31:0] data_vals,
    input stype,clk,sign
    );

    reg [31:0] d0=100;
    reg [31:0] d1=20;
    reg [31:0] d2=30;
    reg [31:0] d3=40;
    reg [31:0] d4;
    reg [31:0] d5;
    reg [31:0] d6;
    reg [31:0] d7;


    always_comb

    if(sign) begin
    unique case (data_addr)
        'b00000000000000000000000000000000 : data_vals = d0;
        'b00000000000000000000000000000100 : data_vals = d1;
        'b00000000000000000000000000001100 : data_vals = d2;
        'b00000000000000000000000000010000 : data_vals = d3;
        'b00000000000000000000000000010100 : data_vals = d4;
        default : data_val=17;
    endcase
    end
    else begin
    unique case (data_addr)
        'b00000000000000000000000000000000 : data_val = d0;
        'b00000000000000000000000000000100 : data_val = d1;
        'b00000000000000000000000000001100 : data_val = d2;
        'b00000000000000000000000000010000 : data_val = d3;
        'b00000000000000000000000000010100 : data_val = d4;
        default : data_val=17;
    endcase
```

```
    always @ (posedge clk) begin

        if(stype==1) begin
         unique case (data_addr)
            'b00000000000000000000000000000000 : d0 = data_input;
            'b00000000000000000000000000000100 : d1 = data_input;
            'b00000000000000000000000000001100 : d2 = data_input;
            'b00000000000000000000000000010000 : d3 = data_input;
            'b00000000000000000000000000010100 : d4 = data_input;
            default : d0=data_input;
         endcase
         end
    end

endmodule
```

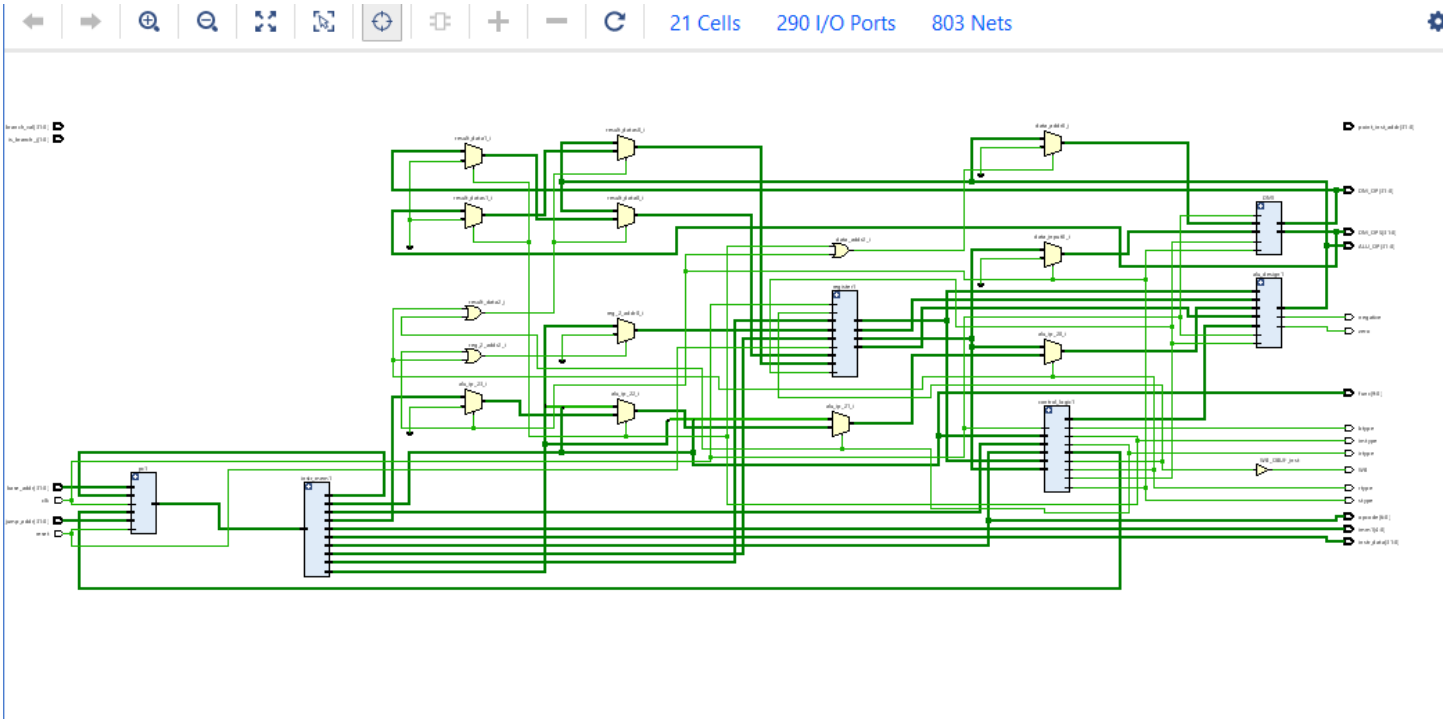
Instruction Memory



```
module instr_mem(
    input logic [31:0] instr_address_pointer,
    output logic [31:0] instr_data,
    output logic [6:0] opcode,
    output logic [4:0] rd,rs1,rs2,imm1,
    output logic [9:0] func,
    logic [2:0] func1,
    output logic  [31:0] imma_val_s,branch_val

    );

    logic [6:0] func2;
    //logic [2:0] func1;

    logic [20:0] make_neg = 20'b11111111111111111111;
    logic [20:0] make_pos = 20'b00000000000000000000;

    logic rtype,irtype,imtype,stype,btype,endbit;


assign endbit=0;
assign opcode = instr_data[6:0];
assign func1 = instr_data[14:12];
assign func2 = instr_data[31:25];
assign rs1 = instr_data[19:15];
assign rs2 = instr_data[24:20];
assign rd = instr_data[11:7];
assign func = {func2,func1};
//assign imma_val_s = (opcode != 0100011 ?(instr_data[31]==0? {make_pos,func2,rs2}:{make_neg,func2,rs2} ):(i
//assign imma_val_s = (opcode == 0100011? {func2,rd} : 0 );
assign imma_val_s = {func2,rd};

assign branch_val = ( rs1==rs2?(instr_data[31]==0? {make_pos,instr_data[7],instr_data[30:25],instr_data[11:7

always_comb
unique case(instr_address_pointer)


    'b00000000000000000000000000000000 : instr_data = 'b000000000000_00000_010_00001_0000011;//lw x1 <-- da
    'b00000000000000000000000000000100 : instr_data = 'b000000000100_00000_010_00010_0000011;//lw x2 <-- da

    'b00000000000000000000000000001000 : instr_data = 'b0000000_00010_00000_010_00000_0100011;//sw data_mem
    'b00000000000000000000000000001100 : instr_data = 'b0000000_00001_00000_010_00100_0100011;//sw data_mem

    'b00000000000000000000000000010000 : instr_data = 'b000000000000_00000_010_00001_0000011;//lw x1 <-- da
    'b00000000000000000000000000010100 : instr_data = 'b000000000100_00000_010_00010_0000011;//lw x2 <-- da

    'b00000000000000000000000000011000 : instr_data = 'b0000000_00010_00001_000_00011_0110011;//addi x3 <--

    default    :    instr_data = 'b11111111111111111111111111111111;
endcase

odule
```

Processor

Every module (pre tested) was interconnected to make the processor. Hierarchical design was used where this is the top module while above modules are sub modules of this.



Code is given Below.

```
include "pc.sv";
include "instr_mem.sv";
include "register.sv";
include "alu_design.sv";
//include "data_memory.sv";
include "control_logic.sv";
include "DM.sv";
module processor(
    input logic [31:0] base_addr,branch_val,jump_addr,
    input logic clk,reset,
    input logic [1:0] is_branch_j,
    output logic  [31:0]  point_inst_addr,ALU_OP,DM_OP,DM_OPS,
    output logic [31:0] instr_data,
    output logic [6:0] opcode,
    output logic [4:0] imm1,
    output logic [9:0] func,
    output logic negative,zero,rtype,irtype,imtype,stype,btype,WB
    );
```

```
wire [31:0] ADDRESS_INSTR,ALU_2,ALU_1,ALU_1s,ALU_2s;
wire [4:0] RD,RS1,RS2;
wire[3:0] ALU_SEL;
wire [31:0] IMM_VAL_S,BRANCH_VAL;
wire SIGN;
wire [1:0] IS_BRANCH_J;
wire [2:0] FUNC1;
//wire WB;

pc pc1(
    .base_addr(base_addr),
    .branch_val(BRANCH_VAL),
    .jump_addr(jump_addr),
    .reset(reset),
    .clk(clk),
    .is_branch_j(IS_BRANCH_J),
    .point_inst_addr(ADDRESS_INSTR)
);



control_logic control_logic1(
    .opcode(opcode),
    .clk(clk),
    .func(func),
    .alu_sel(ALU_SEL),
    .rtype(rtype),
    .irtype(irtype),
    .imtype(imtype),
    .stype(stype),
    .btype(btype),
    .is_wb(WB),
    .sign(SIGN),
    .is_branch_j(IS_BRANCH_J),
    .rs1(ALU_1),
    .rs2(ALU_2),
    .func1(FUNC1)
);



 instr_mem instr_mem1(
     .instr_address_pointer(ADDRESS_INSTR),
     .instr_data(instr_data),
     .opcode(opcode),
     .rd(RD),
     .rs1(RS1),
     .rs2(RS2),
     .imm1(imm1),
     .func(func),
     .func1(FUNC1),
     .imma_val_s(IMM_VAL_S),
     .branch_val(BRANCH_VAL)
 );
```

```
 alu_design alu_design1(
     .alu_ip_1(ALU_1),
     .alu_ip_1s(ALU_1s),
     .alu_ip_2s(ALU_2s),
     .alu_ip_2(rtype ? ALU_2: (irtype? {func[9:3],RS2}: (imtype?{func[9:3],RS2}:(stype ? IMM_VAL_S :0) )
     .alu_sel(ALU_SEL),
     .clk(clk),
     .alu_op(ALU_OP),
     .zero(zero),
     .negative(negative),
     .sign(SIGN)
 );


    DM DM1(
        .data_addr(( imtype || stype )? ALU_OP:0),
        .data_val(DM_OP),
        .data_vals(DM_OPS),
        .data_input(stype ? ALU_2 : 0),
        .stype(stype),
        .clk(clk),
        .sign(SIGN)
    );


    register register1(
        .reg_1_addr(RS1),
        .reg_2_addr((stype||rtype) ? RS2 : 0),
        .reg_3_addr(RD),
        .result_data((rtype|irtype)?ALU_OP:(imtype?DM_OP:0)),
        .result_datas((rtype|irtype)?ALU_OP:(imtype?DM_OPS:0)),
        .reset(reset),
        .clk(clk),
        .is_wb(WB),
        .reg_1_data(ALU_1),
        .reg_1_datas(ALU_1s),
        .reg_2_data(ALU_2),
        .reg_2_datas(ALU_2s),
        .sign(SIGN)
    );

endmodule
```

Testing & Results

Here are some simulation results which were done to verify the proper functioning.

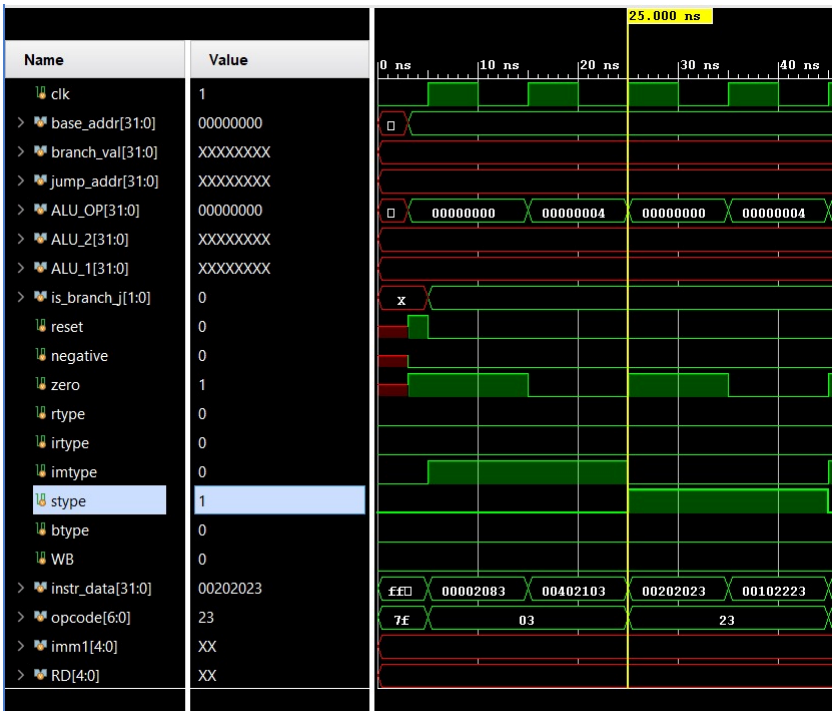
```
address of the 1st 00000000000000000000000000000000:          0
data of the 1st instr      8323
output of the alu:         0
reg x0 value          0
reg x1 value          0
reg x2 value          0
reg x3 value          0
reg x4 value          0
reg x5 value          0
DM output value :       100IMMEDIATE VAL :          1   reg value 2          0
address of the RS2 0
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
address of the 7th 00000000000000000000000000011100:         28
data of the 7th instr4294967295
output of the alu:         0
reg x0 value          0
reg x1 value         20
reg x2 value        100
reg x3 value        120
reg x4 value          0
reg x5 value          0
DM output value :        20IMMEDIATE VAL :       4095   reg value 2          0
address of the RS231
''''''''''''''''''''''''''''''''''''''''''''''''''''''''

address of the 7th 00000000000000000000000000011100:         28
data of the 7th instr4294967295
output of the alu:         0
reg x0 value          0
reg x1 value         20
reg x2 value        100
reg x3 value        120
reg x4 value          0
reg x5 value          0
DM output value :        20IMMEDIATE VAL :       4095   reg value 2          0
address of the RS231
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

In the above picture addition and store of the result was carried out. (add x3 , x1 ,x2)

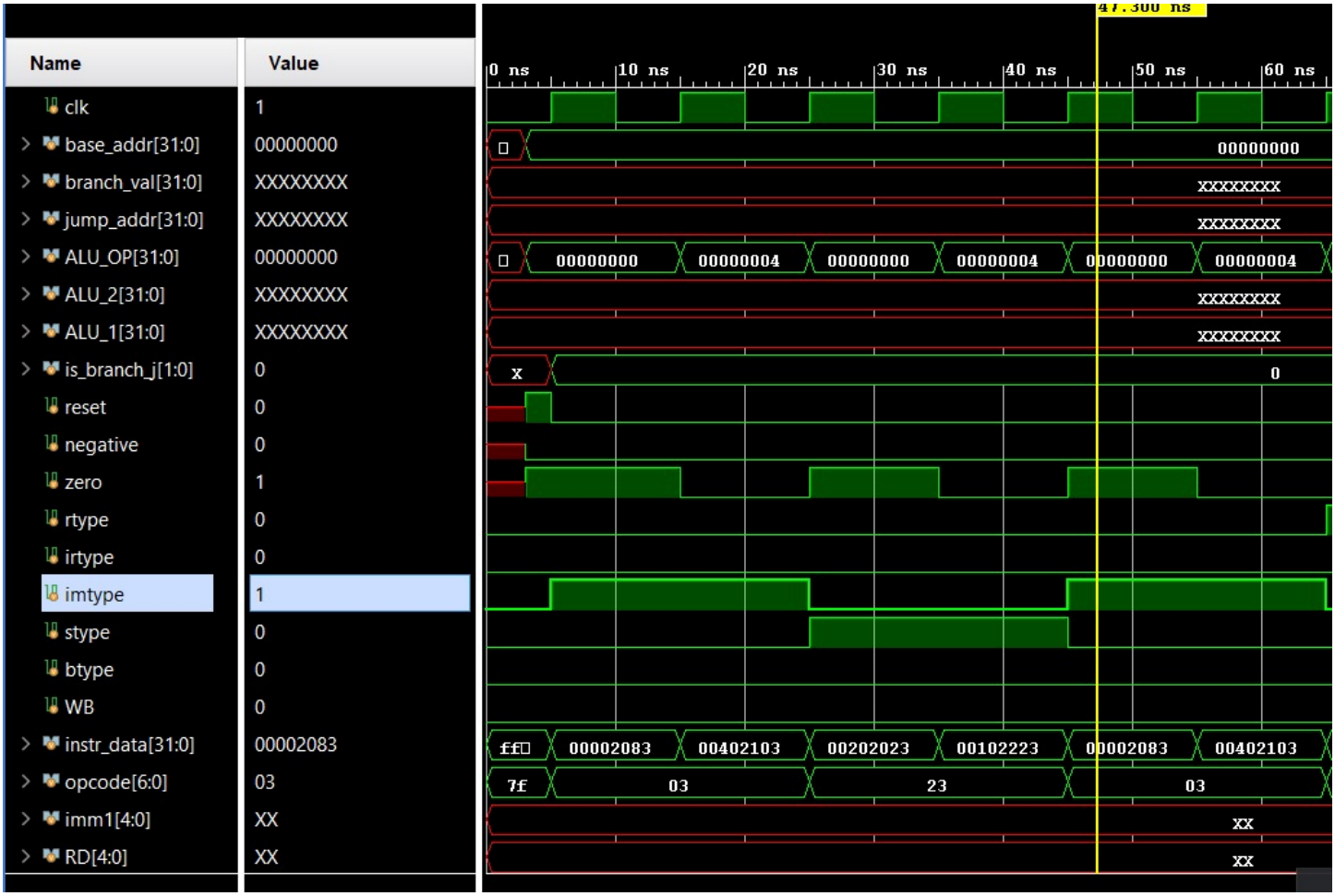
Simulation results for instruction types

R- type

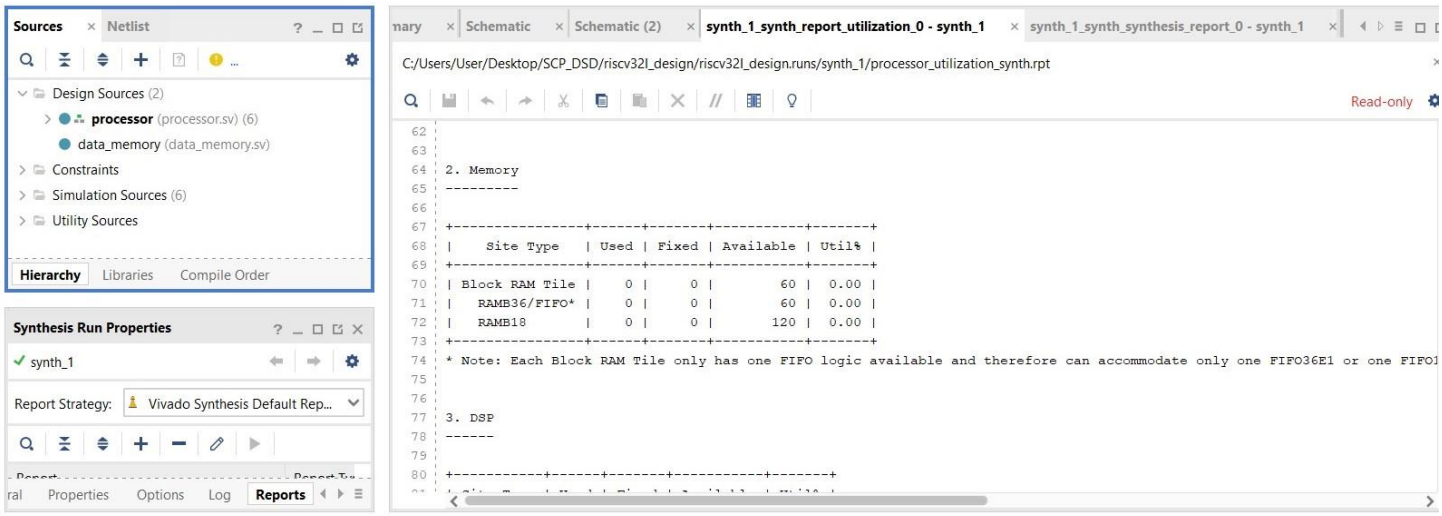
S – Type



Im – Type



After synthesizing the design, the following utilization report was taken.



End of the Report.