VERILOG PROJECT

RISC-V

By:

Ahmed Ashraf El-HADY

September 9, 2023

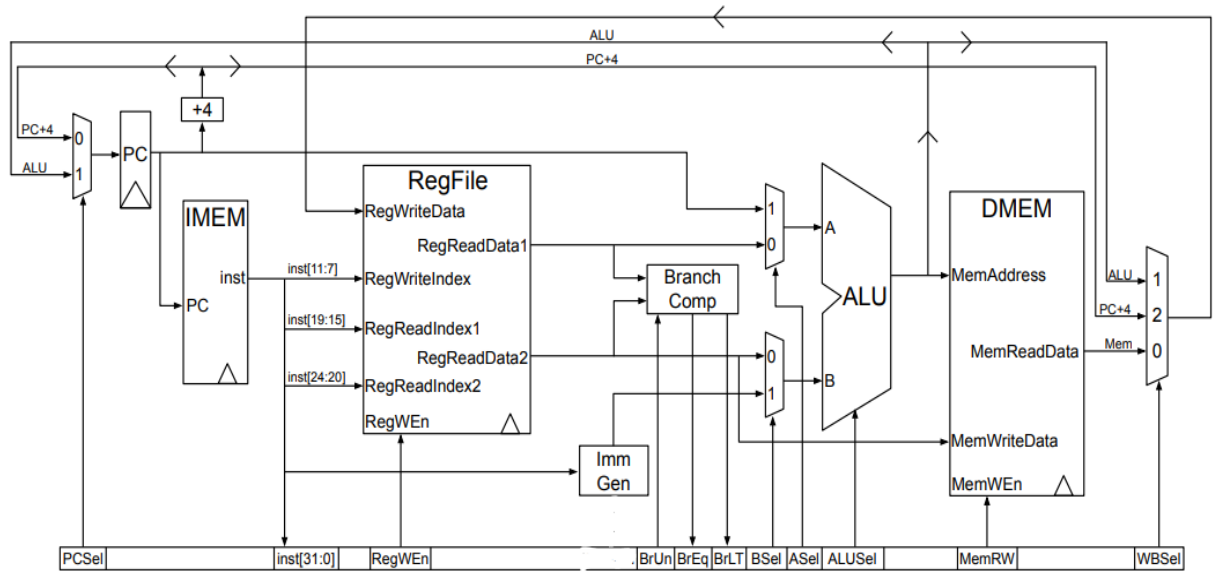
Introduction: -

RISC-V is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles.

Instruction format: -

Format	Bit																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

Design architecture: -



Covered instruction: -

	Instruction			Name	Description				Type	Opcode		Funct3	Funct7	
Arithmetic	**add**	**rd**	**rs1 rs2**	ADD	**rd**	**=**	**rs1**	**+ rs2**	R	**011**	**0011**	**000**	**000**	**0000**
	sub	**rd**	**rs1 rs2**	SUBtract	**rd**	**=**	**rs1**	**- rs2**	R	**011**	**0011**	**000**	**010**	**0000**
	and	**rd**	**rs1 rs2**	bitwise AND	**rd**	**=**	**rs1**	**& rs2**	R	**011**	**0011**	**111**	**000**	**0000**
	or	**rd**	**rs1 rs2**	bitwise OR	**rd**	**=**	**rs1**	**| rs2**	R	**011**	**0011**	**110**	**000**	**0000**
	xor	**rd**	**rs1 rs2**	bitwise XOR	**rd**	**=**	**rs1**	**^ rs2**	R	**011**	**0011**	**100**	**000**	**0000**
	sll	**rd**	**rs1 rs2**	Shift Left Logical	**rd**	**=**	**rs1**	**<< rs2**	R	**011**	**0011**	**001**	**000**	**0000**
	srl	**rd**	**rs1 rs2**	Shift Right Logical	**rd**	**=**	**rs1**	**>> rs2** (Zero-extend)	R	**011**	**0011**	**101**	**000**	**0000**
	sra	**rd**	**rs1 rs2**	Shift Right Arithmetic	**rd**	**=**	**rs1**	**>> rs2** (Sign-extend)	R	**011**	**0011**	**101**	**010**	**0000**
	slt	**rd**	**rs1 rs2**	Set Less Than (signed)	**rd**	**=**	**(rs1**	**< rs2) ? 1 : 0**	R	**011**	**0011**	**010**	**000**	**0000**
	sltu	**rd**	**rs1 rs2**	Set Less Than (Unsigned)					R	**011**	**0011**	**011**	**000**	**0000**
	addi	**rd**	**rs1 imm**	ADD Immediate	**rd**	**=**	**rs1**	**+ imm**	I	**001**	**0011**	**000**		
	andi	**rd**	**rs1 imm**	bitwise AND Immediate	**rd**	**=**	**rs1**	**& imm**	I	**001**	**0011**	**111**		
	ori	**rd**	**rs1 imm**	bitwise OR Immediate	**rd**	**=**	**rs1**	**| imm**	I	**001**	**0011**	**110**		
	xori	**rd**	**rs1 imm**	bitwise XOR Immediate	**rd**	**=**	**rs1**	**^ imm**	I	**001**	**0011**	**100**		
	slli	**rd**	**rs1 imm**	Shift Left Logical Immediate	**rd**	**=**	**rs1**	**<< imm**	I*	**001**	**0011**	**001**	**000**	**0000**
	srli	**rd**	**rs1 imm**	Shift Right Logical Immediate	**rd**	**=**	**rs1**	**>> imm** (Zero-extend)	I*	**001**	**0011**	**101**	**000**	**0000**
	srai	**rd**	**rs1 imm**	Shift Right Arithmetic Immediate	**rd**	**=**	**rs1**	**>> imm** (Sign-extend)	I*	**001**	**0011**	**101**	**010**	**0000**
	slti	**rd**	**rs1 imm**	Set Less Than Immediate (signed)	**rd**	**=**	**(rs1**	**< imm) ? 1 : 0**	I	**001**	**0011**	**010**		
	sltiu	**rd**	**rs1 imm**	Set Less Than Immediate (Unsigned)					I	**001**	**0011**	**011**		

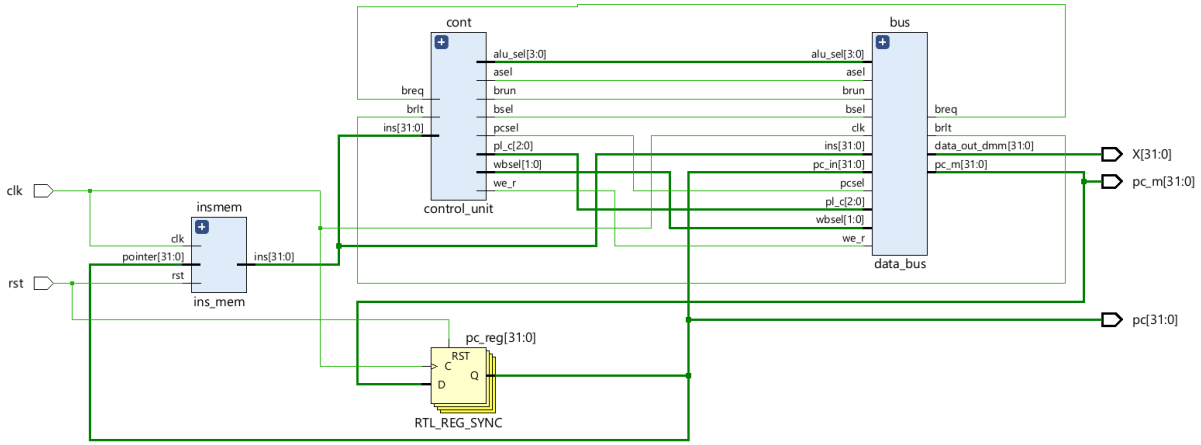
Memory	lb	rd	imm(rs1)	Load Byte	rd = 1 byte of memory at address rs1 + imm, sign-extended	I	000	0011	000	
	lbu	rd	imm(rs1)	Load Byte (Unsigned)	rd = 1 byte of memory at address rs1 + imm, zero-extended	I	000	0011	100	
	lh	rd	imm(rs1)	Load Half-word	rd = 2 bytes of memory starting at address rs1 + imm, sign-extended	I	000	0011	001	
	lhu	rd	imm(rs1)	Load Half-word (Unsigned)	rd = 2 bytes of memory starting at address rs1 + imm, zero-extended	I	000	0011	101	
	lw	rd	imm(rs1)	Load Word	rd = 4 bytes of memory starting at address rs1 + imm	I	000	0011	010	
	sb	rs2	imm(rs1)	Store Byte	Stores least-significant byte of rs2 at the address rs1 + imm in memory	S	010	0011	000	
	sh	rs2	imm(rs1)	Store Half-word	Stores the 2 least-significant bytes of rs2 starting at the address rs1 + imm in memory	S	010	0011	001	
	sw	rs2	imm(rs1)	Store Word	Stores rs2 starting at the address rs1 + imm in memory	S	010	0011	010	

Control	beq rs1 rs2 label	Branch if EQual	if (rs1 == rs2) PC = PC + offset	B	110 0011	000
	bge rs1 rs2 label	Branch if Greater or Equal (signed)	if (rs1 >= rs2) PC = PC + offset	B	110 0011	101
	bgeu rs1 rs2 label	Branch if Greater or Equal (Unsigned)		B	110 0011	111
	blt rs1 rs2 label	Branch if Less Than (signed)	if (rs1 < rs2) PC = PC + offset	B	110 0011	100
	bltu rs1 rs2 label	Branch if Less Than (Unsigned)		B	110 0011	110
	bne rs1 rs2 label	Branch if Not Equal	if (rs1 != rs2) PC = PC + offset	B	110 0011	001
	jal rd label	Jump And Link	rd = PC + 4 PC = PC + offset	J	110 1111	
	jalr rd rs1 imm	Jump And Link Register	rd = PC + 4 PC = rs1 + imm	I	110 0111	000
	auipc rd imm	Add Upper Immediate to PC	rd = PC + (imm << 12)	U	001 0111	
	lui rd imm	Load Upper Immediate	rd = imm << 12	U	011 0111	

RTL analysis: -

```
module CPU (clk,rst,pc_m,pc,X,ins);
      input clk , rst;
      output[31:0] pc_m,X,ins;
      output reg [31:0]pc;

      wire [3:0]alu_sel;
      wire [2:0] pl_c;
      wire [1:0] wbsel;
      wire bsel,asel,we_r,brun,pcsel,brlt,breq;
```

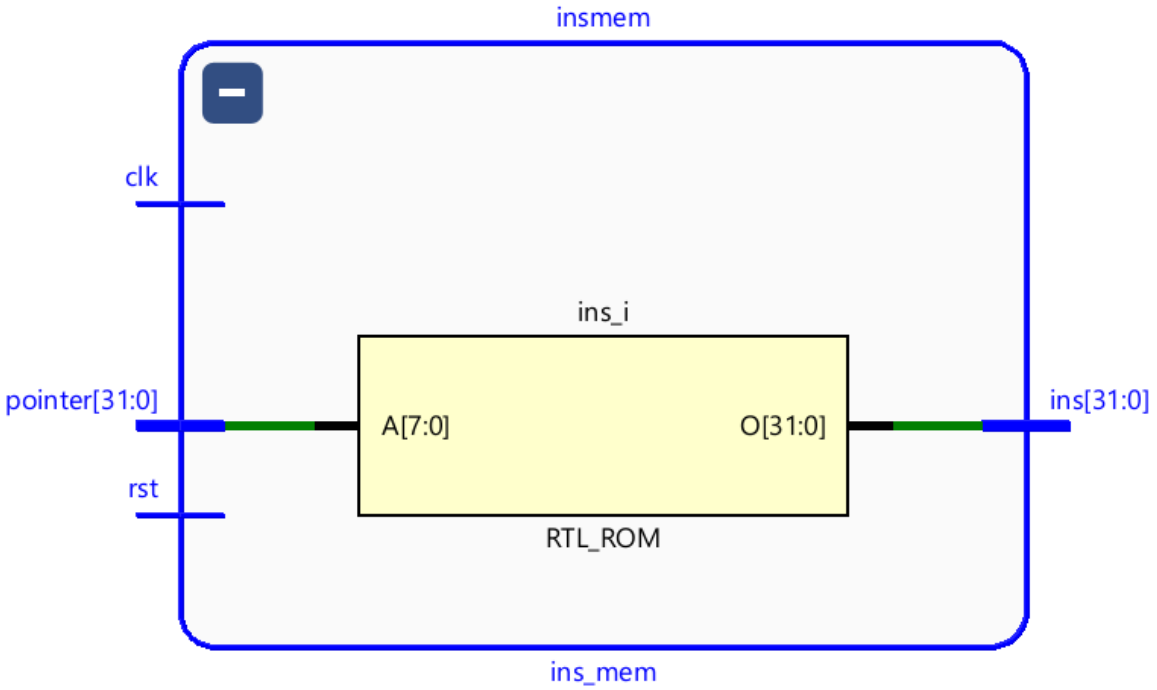
```
      control_unit
cont(ins,alu_sel,bsel,wbsel,pl_c,we_r,asel,brun,breq,brlt,pcsel);

      data_bus
bus(clk,we_r,bsel,alu_sel,ins,wbsel,pl_c,asel,pcsel,brun,breq,brlt,p
c,pc_m,X);

      ins_mem insmem(clk,rst,pc,ins);



      always @(posedge clk) begin
            if(rst) begin
                  pc<= 0;
                  $readmemh("reg_ini.mem",bus.regfile.r_f);
                  $readmemh("Dmem_ini.mem",bus.d_mem.D_mem);
            end else begin
                  pc <= pc_m;
            end
      end
endmodule
```



Instruction memory: -

```
// 1 KB ROM to keep ins
module ins_mem (clk,rst,pointer,ins);
input clk,rst;
input [31:0]pointer;
output  [31:0] ins;
reg [31:0] ins_rom [255:0];

wire [7:0] add = pointer[9:2];

assign ins =ins_rom[add];
```

```
endmodule
```



Control unit: -

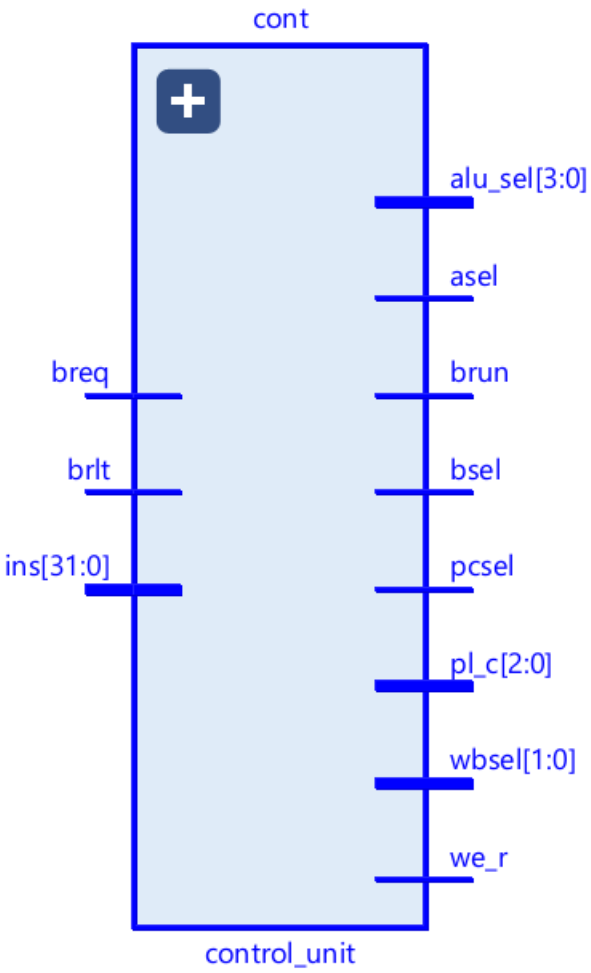
```
module control_unit
(ins,alu_sel,bsel,wbsel,pl_c,we_r,asel,brun,breq,brlt,pcsel);
////////////////////////////////////PARAMETERS//////////////////////
//////

parameter ADD = 4'h0 , AND = 4'h1 ,OR  = 4'h2 , XOR = 4'h3 ,  SUB =
4'h4 , SLT = 4'h5 , SLTU = 4'h6 ,
             SLL = 4'h7 , SRL  = 4'h8 , SRA = 4'h9 ,LUI=4'ha;
parameter LW =3'h0 ,
            LH =3'h1 ,
            LHU=3'h2 ,
            LB =3'h3 ,
            LBU=3'h4 ,
            SB =3'h5 ,
            SH =3'h6 ,
            SW =3'h7 ;
////////////////////////////////////////////////////////////////////
////
```

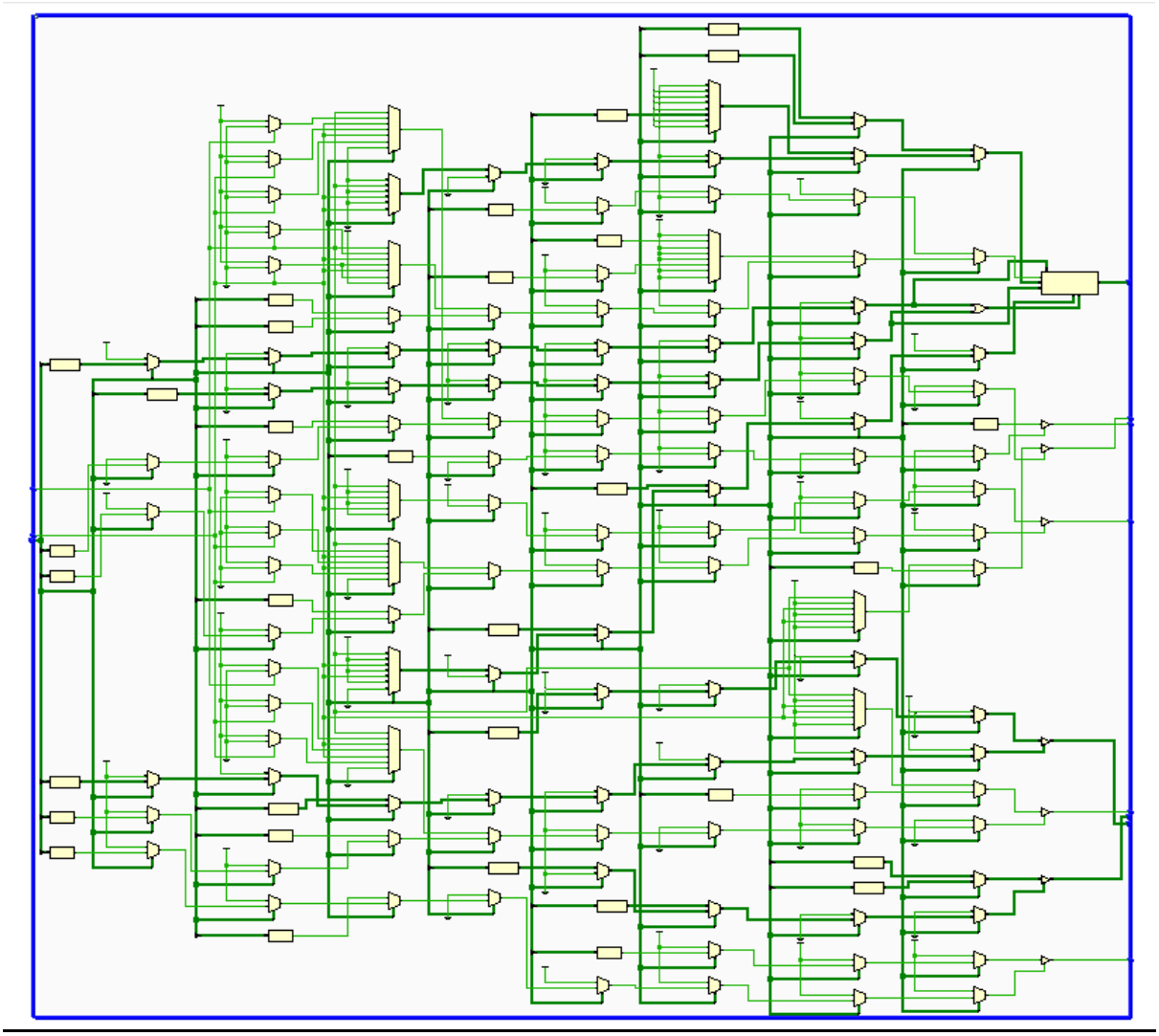
```
input brlt,breq;
input [31:0] ins;
output reg [3:0] alu_sel;
output reg bsel,asel,we_r,brun,pcsel ;
output reg [1:0] wbsel;
output reg [2:0] pl_c;

wire [6:0] opcode;
wire [2:0] funct3;
wire [6:0] funct7;
assign opcode =ins[6:0];
assign funct3 =ins[14:12];
assign funct7 =ins[31:25];

//alu_sel
always @(*) begin
     bsel='bz;
     wbsel='bz;
     pl_c='bz;
     we_r='bz;
     asel='bz;
     brun='bz;
     pcsel='b0;
     if (opcode==7'h33)// Rtype Arth
          begin
               asel=0;
               pcsel=0;
               we_r=1;
               bsel=0;
               wbsel=2'h1;
               if (funct7==0)begin
                    case (funct3)
                         'b000   : alu_sel= ADD;
                         'b001   : alu_sel= SLL;
                         'b010   : alu_sel= SLT;
                         'b011   : alu_sel=SLTU;
                         'b100   : alu_sel= XOR;
                         'b101   : alu_sel= SRL;
                         'b110   : alu_sel= OR ;
                         'b111   : alu_sel= AND;
                         default : alu_sel= 'bz;
                    endcase
               end
               else if(funct7=='h20)begin
                    if (funct3=='b101)
                         alu_sel= SRA;
                    else if(funct3=='b000)
                         alu_sel= SUB;
               end
```

```
          end
     else if(opcode==7'h13)// Itype Arth
               begin
                    asel=0;
                    pcsel=0;
                    we_r=1;
                    bsel=1;
                    wbsel=2'h1;
                    case (funct3)
                         'b000    : alu_sel= ADD;
                         'b001    : if(funct7==0)
                                        alu_sel= SLL;
     /////i*
                         'b010    : alu_sel= SLT;
                         'b011    : alu_sel=SLTU;
                         'b100    : alu_sel= XOR;
                         'b101    :if(funct7==0)
/////i*
                                        alu_sel= SRL;
                                       else
                                       if(funct7=='h20)
     /////i*
                                        alu_sel= SRA;
                         'b110    : alu_sel= OR ;
                         'b111    : alu_sel= AND;
                         default : alu_sel= 'bz;
                    endcase

               end
     else if(opcode==7'h03)// Itype MEM
               begin
                    asel=0;
                    pcsel=0;
                    we_r=1;
                    wbsel=2'h0;
                    bsel=1;
                    case (funct3)
                         'b000 : begin alu_sel= ADD; pl_c =LB ;
end
                         'b001 : begin alu_sel= ADD; pl_c =LH ;
end
                         'b010 : begin alu_sel= ADD; pl_c =LW ;
end
                         'b100      : begin alu_sel= ADD; pl_c
=LBU; end
                         'b101      : begin alu_sel= ADD; pl_c
=LHU; end
                         default : alu_sel= 'bz;
```

```
                    endcase

                end
     else if(opcode==7'h23)// Stype MEM
                begin
                    asel=0;
                    pcsel=0;
                    we_r=0;
                    bsel=1;
                    case (funct3)
                        'b000 : begin alu_sel= ADD; pl_c =SB ;
end
                        'b001 : begin alu_sel= ADD; pl_c =SH ;
end
                        'b010 : begin alu_sel= ADD; pl_c =SW ;
end
                        default : alu_sel= 'bz;
                    endcase

                end

     else if(opcode==7'h63)// Btype MEM
                begin
                    case (funct3)
                        'b000 : begin               if( breq
)begin alu_sel= ADD;asel=1; bsel=1;pcsel=1; end  end
                        'b001 : begin               if(!breq
)begin alu_sel= ADD;asel=1; bsel=1;pcsel=1; end  end
                        'b100 : begin brun=0; if( brlt )begin
alu_sel= ADD;asel=1; bsel=1;pcsel=1; end    end
                        'b101 : begin    brun=0; if(!brlt )begin
alu_sel= ADD;asel=1; bsel=1;pcsel=1; end    end
                        'b110 : begin    brun=1; if( brlt )begin
alu_sel= ADD;asel=1; bsel=1;pcsel=1; end    end
                        'b111 : begin    brun=1; if(!brlt )begin
alu_sel= ADD;asel=1; bsel=1;pcsel=1; end    end
                        default : begin  brun=1'bz;  alu_sel=
'bz;asel=1'bz; bsel=1'bz;pcsel=1'bz;  end
                    endcase

                end
     else if(opcode==7'h67)// I type  JALR
                begin
                    if(funct3==0)begin
                    wbsel=2'h2;
                    we_r =1;
                    bsel=1;
                    asel=0;
                    pcsel=0;
```

```
                        alu_sel=ADD;
                        end
                  end
      else if(opcode==7'h6f)// J type  JAL
                  begin
                        pcsel=1;
                        wbsel=2'h2;
                        we_r =1;
                        bsel=1;
                        asel=1;
                        alu_sel=ADD;

                  end
    else if(opcode==7'h37)// U type         LUI
                  begin

                        pcsel=0;
                        wbsel=2'h1;
                        we_r =1;
                        bsel=1;
                        alu_sel=LUI;

                  end
      else if(opcode==7'h17)// U type       AUIPC
                  begin
                        pcsel=0;
                        wbsel=2'h1;
                        we_r =1;
                        bsel=1;
                        asel=1;
                        alu_sel=ADD;

                  end
end


endmodule
```

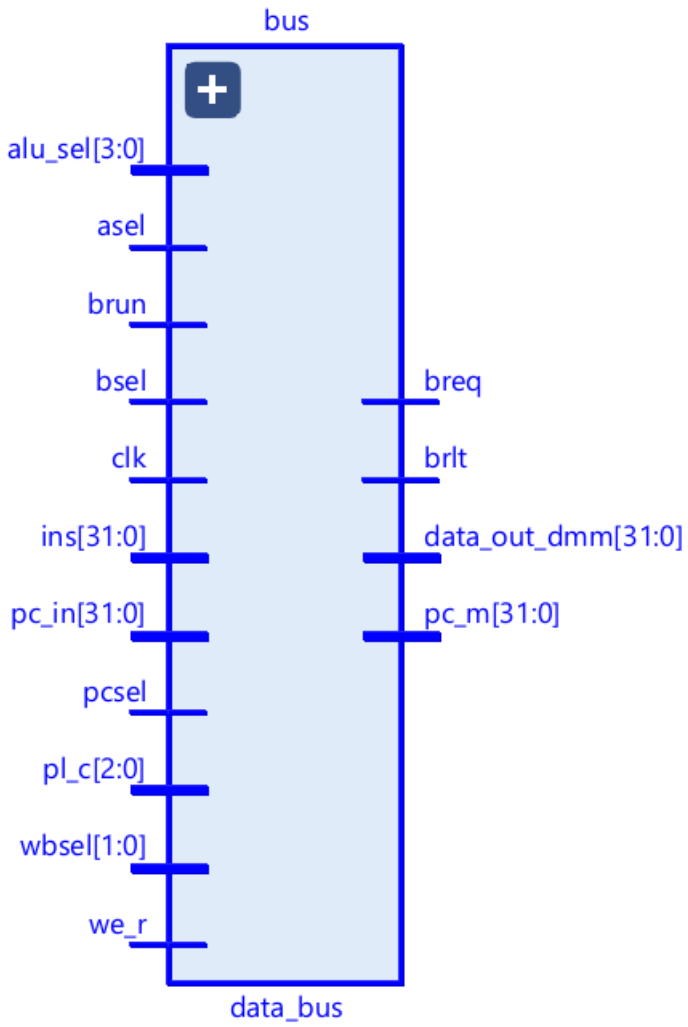

cont
alu_sel[3:0]
asel
breq
brun
brlt
bsel
ins[31:0]
pcsel
pl_c[2:0]
wbsel[1:0]
we_r
control_unit

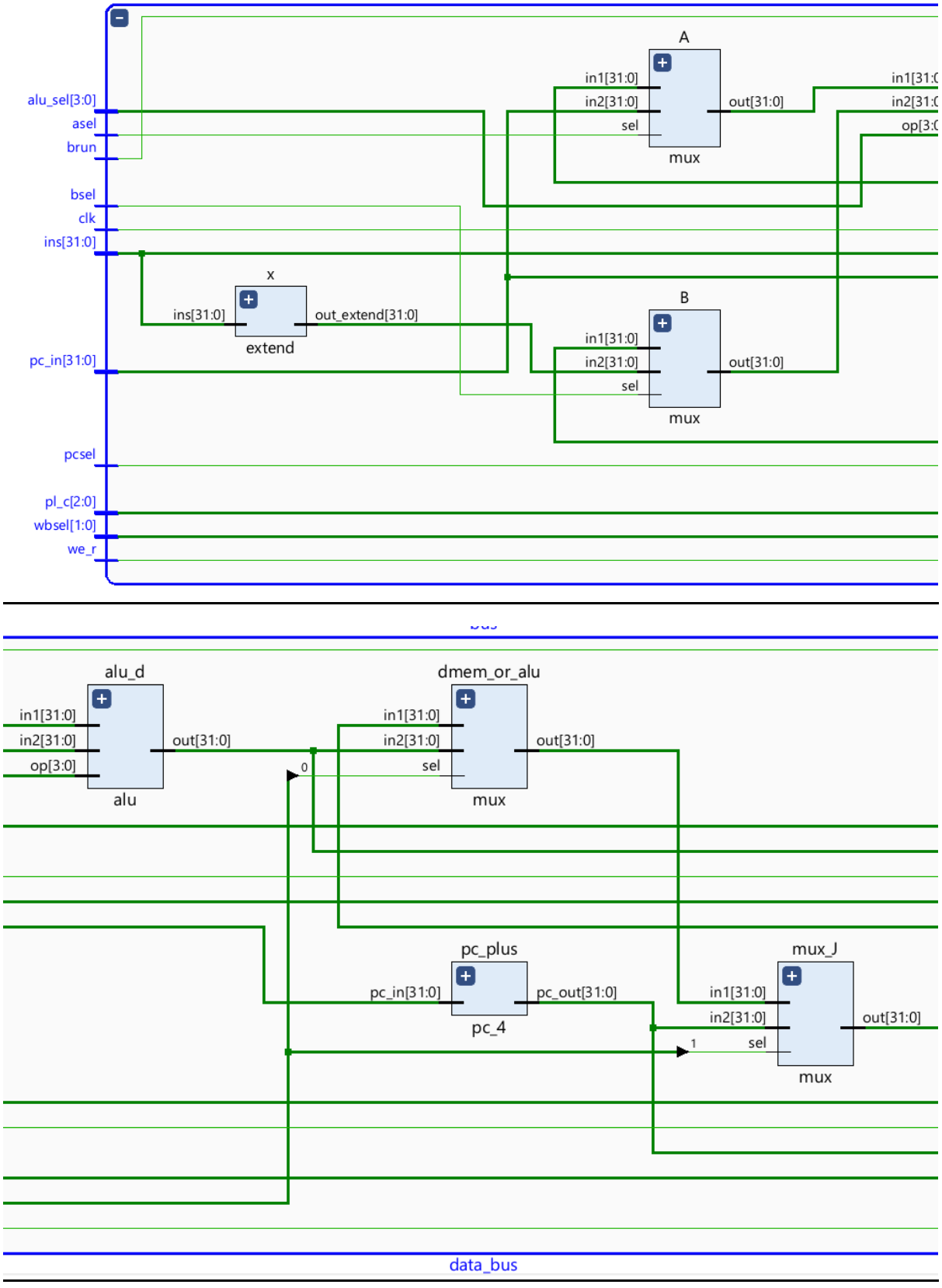


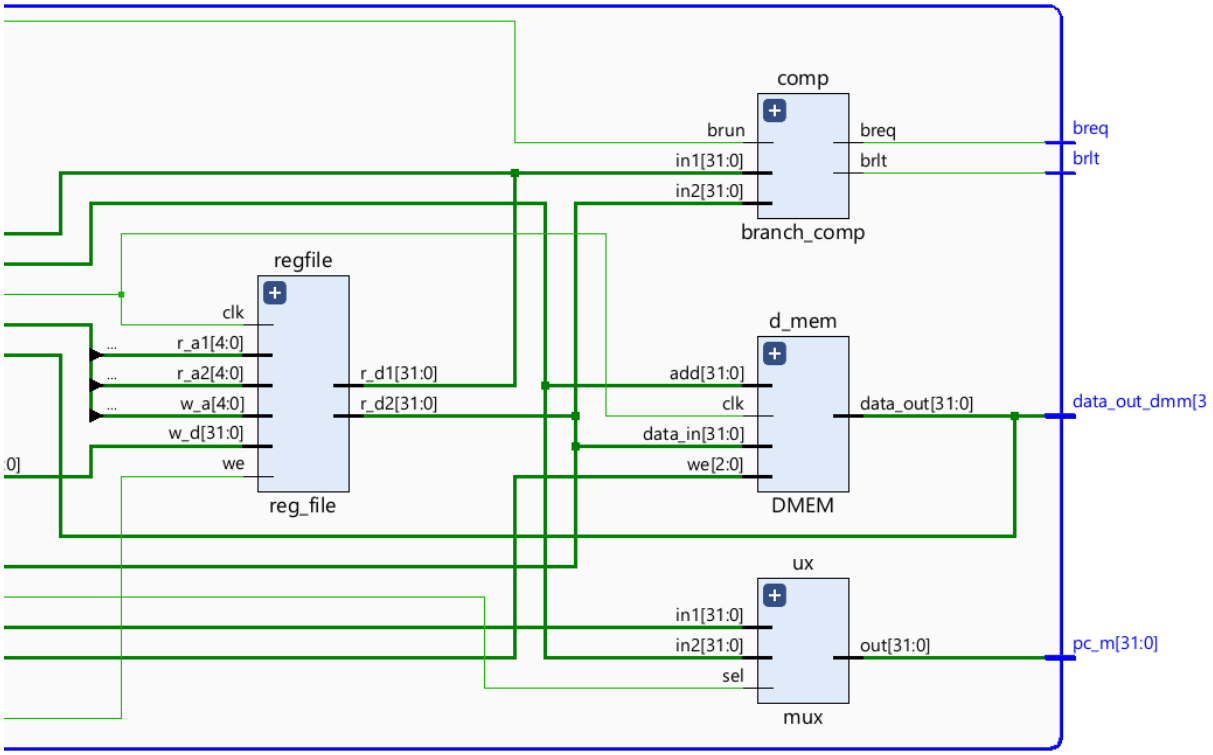
Data bus: -

```
module data_bus
(clk,we_r,bsel,alu_sel,ins,wbsel,pl_c,asel,pcsel,brun,breq,brlt,pc_i
n,pc_m,data_out_dmm);
      input clk,we_r,bsel,asel,brun,pcsel;
      input[1:0] wbsel;
      input [2:0] pl_c;
      input [3:0]alu_sel;
      input [31:0]ins,pc_in;
      output breq,brlt;
      output [31:0]pc_m,data_out_dmm;

      wire
[31:0]alu_out,r_d1,r_d2,r2_m,imm,reg_alu_out,lp_out,r1_m,pc_out,reg_
alu_out_m;
      pc_4 pc_plus(pc_in,pc_out);
      mux ux(pc_out,alu_out,pcsel,pc_m);
      reg_file
regfile(clk,we_r,ins[11:7],ins[19:15],ins[24:20],reg_alu_out,r_d1,r_
```

```
d2);
      alu alu_d(r1_m,r2_m,alu_sel,alu_out);
      mux B(r_d2,imm,bsel,r2_m);
      mux A(r_d1,pc_in,asel,r1_m);
      extend x(ins,imm);
      DMEM d_mem(clk,pl_c,alu_out,r_d2,data_out_dmm);
      //partial_load lp(data_out_dmm,pl_c,lp_out); // done inside
mem
      mux dmem_or_alu(data_out_dmm,alu_out,wbsel[0],reg_alu_out_m);
      mux mux_J(reg_alu_out_m,pc_out,wbsel[1],reg_alu_out);
      branch_comp comp(r_d1,r_d2,brun,breq,brlt);


endmodule
```



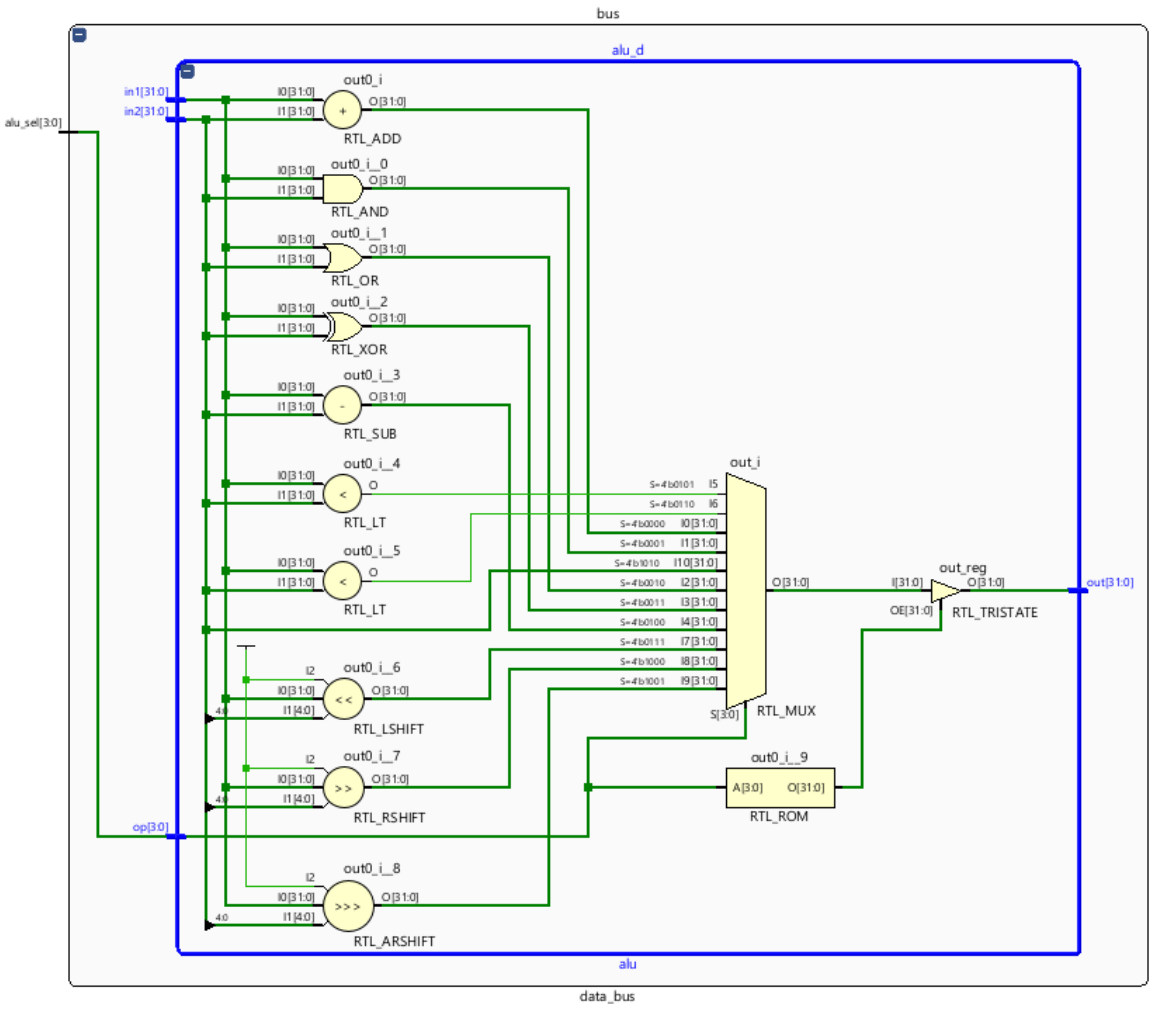

A
in1[31:0]
in2[31:0]
sel
out[31:0]
mux
alu_sel[3:0]
asel
brun
bsel
clk
ins[31:0]
x
ins[31:0]
out_extend[31:0]
extend
B
in1[31:0]
in2[31:0]
sel
out[31:0]
mux
pc_in[31:0]
pcsel
pl_c[2:0]
wbsel[1:0]
we_r
alu_d
in1[31:0]
in2[31:0]
op[3:0]
out[31:0]
alu
dmem_or_alu
in1[31:0]
in2[31:0]
sel
out[31:0]
mux
0
pc_plus
pc_in[31:0]
pc_out[31:0]
pc_4
mux_J
in1[31:0]
in2[31:0]
sel
out[31:0]
mux
1
data_bus



ALU: -

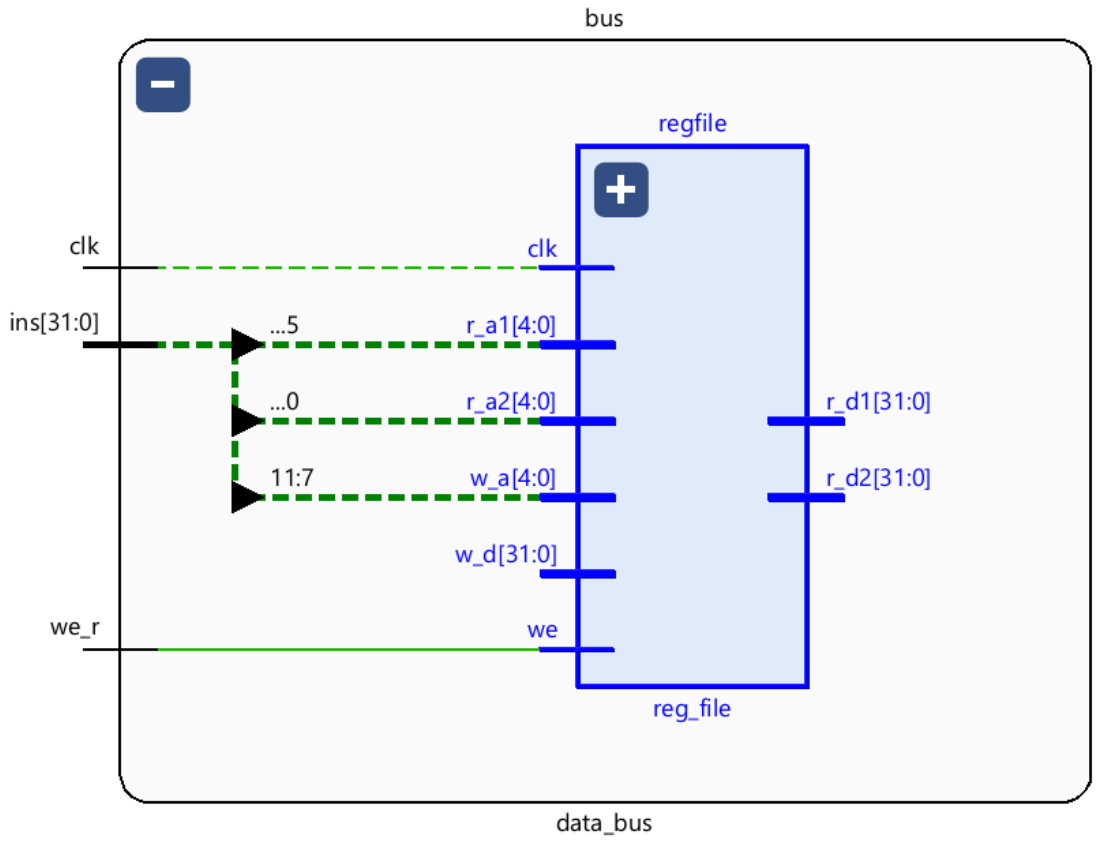
```
//operation can be performed : ADD SUB SLT SLTU _comapre signed and
unsigned _ AND OR XOR SLL SRL SRA
module alu (in1,in2,op,out);
////////////////////////////////////PARAMETERS//////////////////////
//////
parameter ADD = 4'h0 , AND = 4'h1 ,OR  = 4'h2 , XOR = 4'h3 ,  SUB =
4'h4 , SLT = 4'h5 , SLTU = 4'h6 ,
              SLL = 4'h7 , SRL  = 4'h8 , SRA = 4'h9 ,LUI=4'ha;

////////////////////////////////////////////////////////////////////
////
input signed[31:0] in1, in2;
input [3:0]op;
output reg [31:0] out;

wire   [31:0] in1_1, in2_1; // unsigned input for SLU
assign in1_1=in1;
assign in2_1=in2;

always @(*) begin

     case (op)
          ADD     : begin out = in1 + in2 ;
```

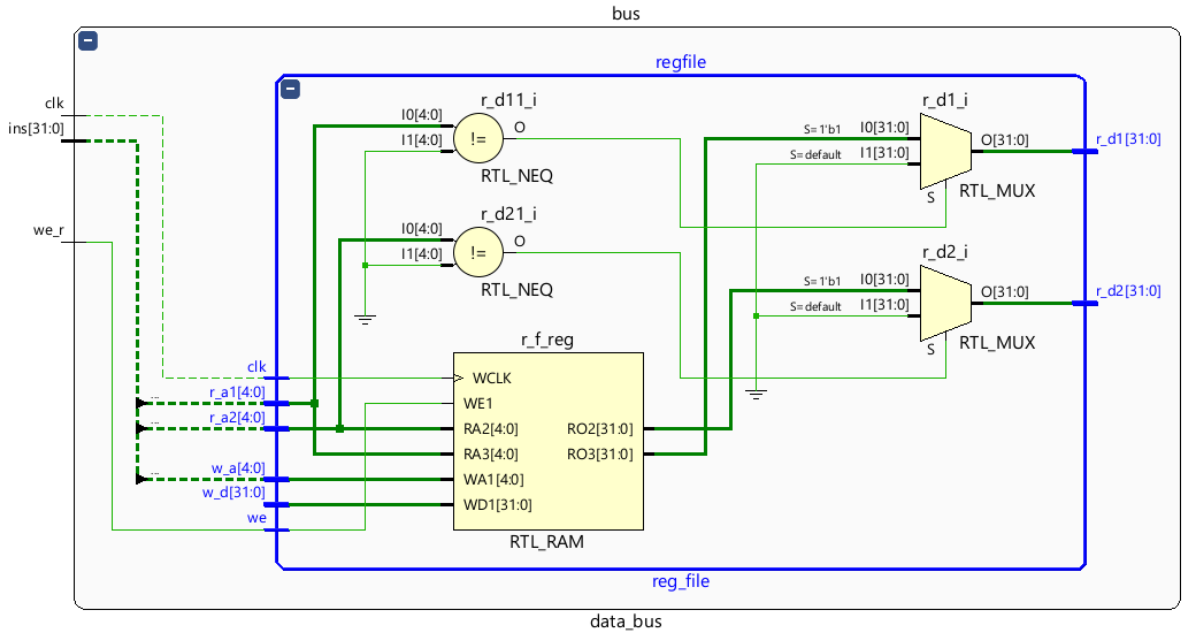
```
      end      // ADD
            AND      : begin out = in1 & in2 ;
      end      // AND
            OR       : begin out = in1 | in2 ;
      end      // OR
            XOR      : begin out = in1 ^ in2 ;
      end      // XOR
            SUB      : begin out = in1 - in2 ;
      end      //SUB
            SLT      : begin if (in1<in2) out=1; else out =0;
      end      //SLT
            SLTU     : begin if ( in1_1 < in2_1 ) out=1; else out =0;
      end      // SLTU
            SLL      : begin out = in1   <<  in2[4:0] ;
      end      // SLL
            SRL      : begin out = in1_1   >>  in2[4:0] ;
end      // SRL
            SRA      : begin out = in1 >>> in2[4:0] ;
      end      // SRA
            LUI         : begin out = in2;
                  end   // LUI
            default : begin out = 'bz;
            end   // default

      endcase

end



endmodule
```



Reg file: -

```
module reg_file (clk,we,w_a,r_a1,r_a2,w_d,r_d1,r_d2);
      input clk,we;
      input [4:0] w_a,r_a1,r_a2;
      input [31:0] w_d;
      output [31:0] r_d1,r_d2;

      reg [31:0] r_f [31:0];
      initial
      r_f[0]=0;
      always@(posedge clk)
      if (we)
            r_f[w_a]<=w_d;

      assign r_d1=(r_a1!=0)? r_f[r_a1] :0;
      assign r_d2=(r_a2!=0)? r_f[r_a2] :0;


endmodule
```



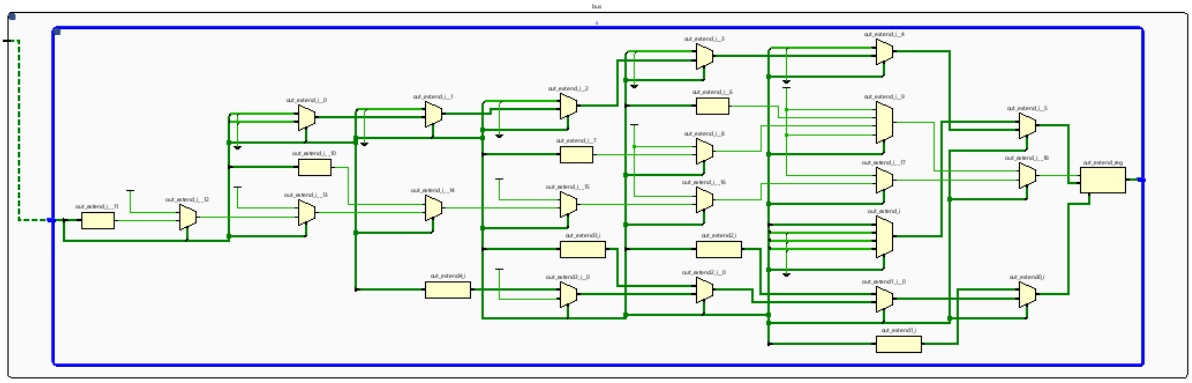


Extend (imm generation):-

```
module extend(ins,out_extend);
input signed [31:0]ins;
output  reg signed [31:0] out_extend;
reg  [31:0] out;

wire [6:0] opcode;
wire [2:0] funct3;
wire [6:0] funct7;
assign opcode =ins[6:0];
assign funct3 =ins[14:12];
assign funct7 =ins[31:25];

wire signed[11:0] imm_i;
assign imm_i=ins[31:20];

always @(*) begin
     out = 'bz;
     if (opcode=='h13)begin
          case (funct3)
                          'b000   ,
                          'b010      : out_extend=imm_i; // addi ,
slti

                          'b001   : if(funct7==0)    // slli
                                         begin
                                         out=ins[24:19];
                                         out_extend=out;
                                         end

                          'b101   :begin if(funct7==0)
     //srli                 /////i*
                                        begin
                                         out=ins[24:19];
                                         out_extend=out;
                                        end
                                        else
                                        if(funct7=='h20)
          //srai     ////i* signed
                                        out_extend=ins[24:19]; end

                          'b011 ,
                          'b100 ,
          /// sltiu ,xori, ori,andi
                          'b110   ,
                          'b111   :begin
                                         out=ins[31:20];
                                         out_extend=out;
                                    end
                          default : out_extend='bz;
```

```
                endcase
     end
     else if (opcode == 'h03)begin ////// i MEM
          case (funct3)
                              'b000,
                              'b001,
                              'b010,
                              'b100,
                              'b101 : out_extend=ins[31:20];

                              default : out_extend='bz;
                endcase
     end
     else if (opcode == 'h23)begin ////// S  MEM
          case (funct3)
                              'b000 ,
                              'b001 ,
                              'b010 :
out_extend={{20{1'b0}},ins[31:25],ins[11:7]};

                              default : out_extend='bz;
                endcase
     end

     else if(opcode==7'h63)// Btype MEM
                begin
                     case (funct3)
                              'b000 ,
                              'b001 ,
                              'b100 ,
                              'b101 ,
                              'b110 ,
                              'b111 : out_extend =
{{20{ins[31]}},ins[31],ins[7],ins[30:25],ins[11:8],1'b0};
                              default : out_extend = 'bz;
                     endcase

                end


     else if(opcode==7'h67)// I type  JALR
                begin
                     if(funct3==0)begin
                     out=ins[24:19];
                     out_extend=out;
                     end
                end

     else if(opcode==7'h6f)// J type  JAL
```

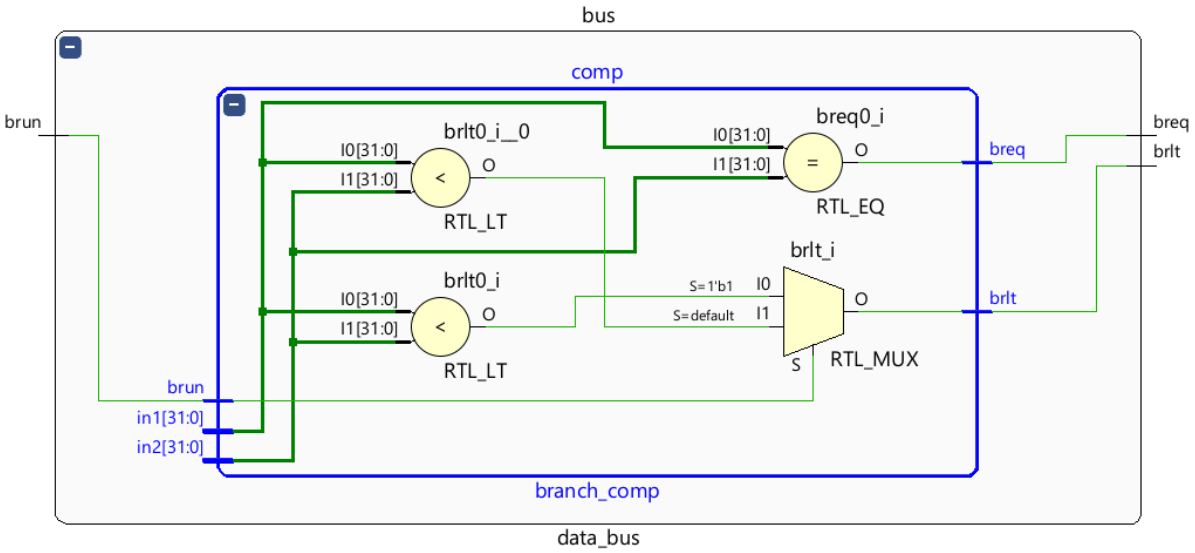
```
                begin

     out_extend={{11{ins[31]}},ins[31],ins[19:12],ins[20],ins[30:21
],1'b0};
                end
     else if(opcode==7'h37)// U type       LUI
                begin
                     out_extend={ins[31:12],12'b0};
                end
     else if(opcode==7'h17)// U type       AUIPC
                begin
                     out_extend={ins[31:12],12'b0};
                end
end

endmodule
```



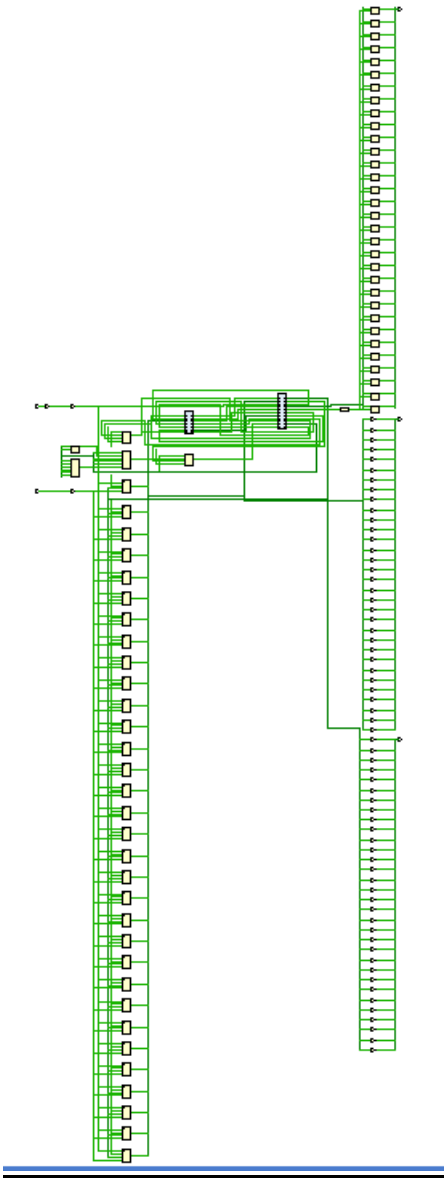
Branch comparator: -

```
module branch_comp (in1,in2,brun,breq,brlt);
input [31:0]in1,in2;
input brun;
output reg breq,brlt;

     always @(*) begin
     breq=1'bz;
     brlt=1'bz;
          if (in1==in2)
                breq=1;
          else
                breq=0;
          if(brun)begin
                if(in1<in2)
                     brlt=1;
                else
                     brlt=0;
```

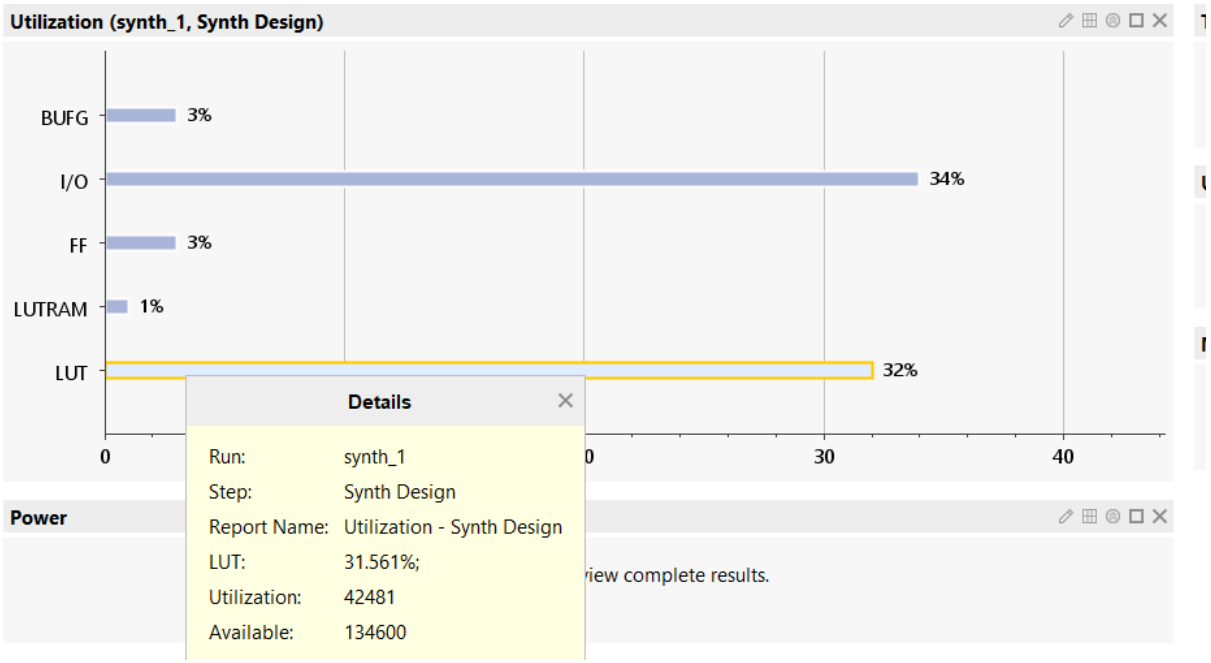
```
            end
            else begin
                  if($signed(in1)<$signed(in2))
                        brlt=1;
                  else
                        brlt=0;
            end

      end

endmodule
```



Synthesis design: -



Synthesis report: -

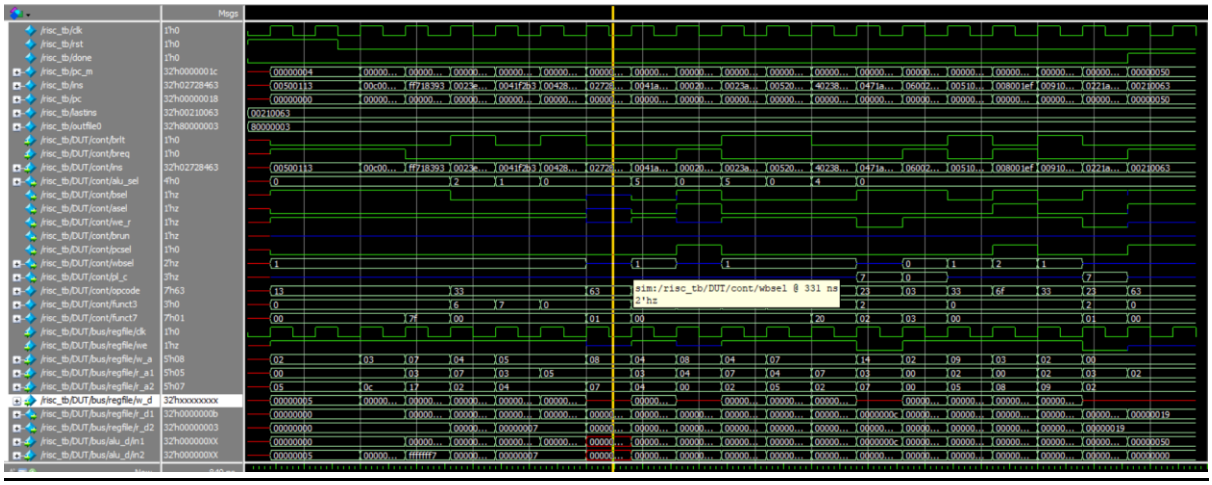


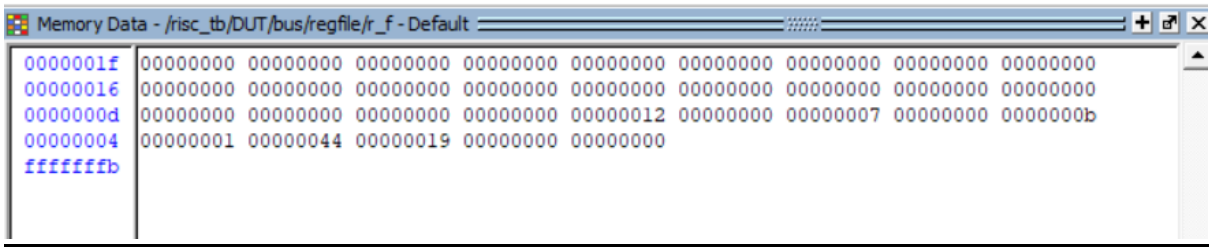
Input sample: -

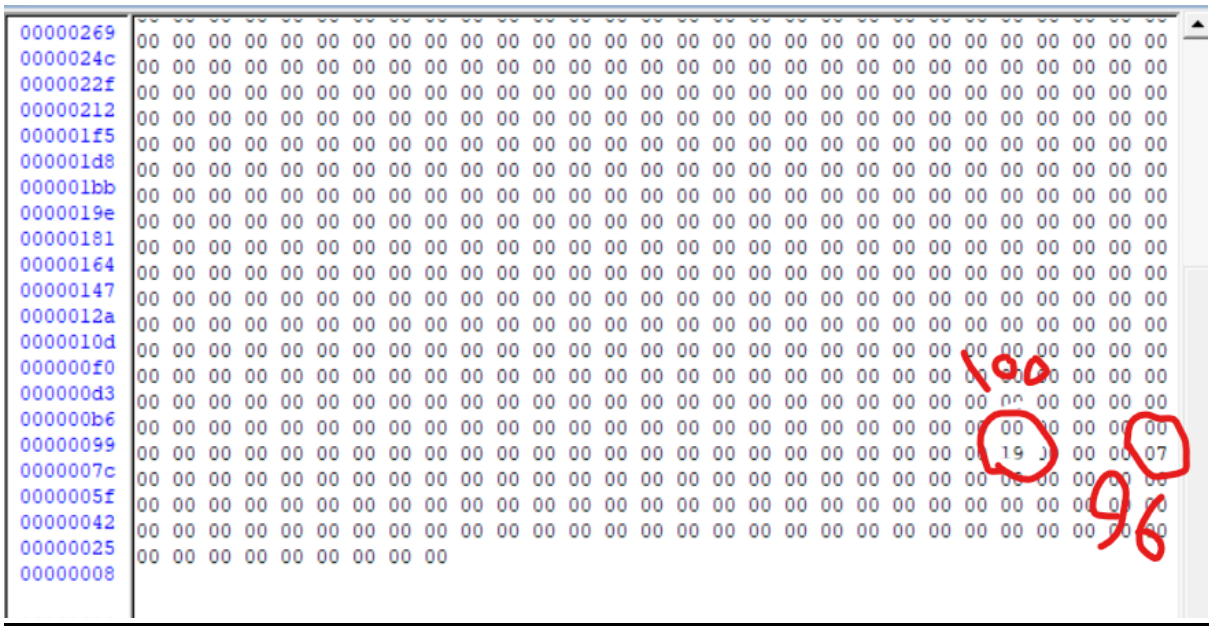
```
00500113
00C00193
FF718393
0023E233
0041F2B3
004282B3
02728463
0041A233
00020463
00000293
0023A233
005203B3
402383B3
0471AA23
06002103
005104B3
008001EF
00100113
00910133
0221A023
00210063
```

```
main: addi x2, x0, 5    # x2 = 5 0 00500113
addi x3, x0, 12         # x3 = 12 4 00C00193
addi x7, x3, -9         # x7 = (12 - 9) = 3 8 FF718393
or x4, x7, x2           # x4 = (3 OR 5) = 7 C 0023E233
and x5, x3, x4          # x5 = (12 AND 7) = 4 10 0041F2B3
add x5, x5, x4          # x5 = 4 + 7 = 11 14 004282B3
beq x5, x7, end         # shouldn't be taken 18 02728863
#slt x4, x3, x4         # x4 = (12 < 7) = 0 1C 0041A233
beq x4, x0, around      # should be taken 20 00020463
addi x5, x0, 0          # shouldn't execute 24 00000293
around: #slt x4, x7, x2  # x4 = (3 < 5) = 1 28 0023A233
add x7, x4, x5          # x7 = (1 + 11) = 12 2C 005203B3
sub x7, x7, x2          # x7 = (12 - 5) = 7 30 402383B3
sw x7, 84(x3)           # [96] = 7 34 0471AA23
lw x2, 96(x0)           # x2 = [96] = 7 38 06002103
add x9, x2, x5          # x9 = (7 + 11) = 18 3C 005104B3
jal x3, end             # jump to end, x3 = 0x44 40 008001EF
addi x2, x0, 1          # shouldn't execute 44 00100113
end: add x2, x2, x9     # x2 = (7 + 18) = 25 48
sw x2, 0x20(x3)         # [100] = 25 0221A023
done: beq x2, x2, done  # infinite loop 50 00210063
```

Snippets: -


/risc_tb/clk
/risc_tb/rst
/risc_tb/done
/risc_tb/pc_m
/risc_tb/ins
/risc_tb/pc
/risc_tb/lastins
/risc_tb/outfile0
/risc_tb/DUT/cont/brlt
/risc_tb/DUT/cont/breq
/risc_tb/DUT/cont/ins
/risc_tb/DUT/cont/alu_sel
/risc_tb/DUT/cont/bsel
/risc_tb/DUT/cont/asel
/risc_tb/DUT/cont/we_r
/risc_tb/DUT/cont/brun
/risc_tb/DUT/cont/pcsel
/risc_tb/DUT/cont/wbsel
/risc_tb/DUT/cont/pl_c
/risc_tb/DUT/cont/opcode
/risc_tb/DUT/cont/funct3
/risc_tb/DUT/cont/funct7
/risc_tb/bus/regfile/clk
/risc_tb/bus/regfile/we
/risc_tb/bus/regfile/w_a
/risc_tb/bus/regfile/r_a1
/risc_tb/bus/regfile/r_a2
/risc_tb/DUT/bus/regfile/w_d
/risc_tb/DUT/bus/regfile/r_d1
/risc_tb/DUT/bus/regfile/r_d2
/risc_tb/DUT/bus/alu_d/in1
/risc_tb/DUT/bus/alu_d/in2
sim:/risc_tb/DUT/cont/wbsel @ 331 ns
2'hz


Memory Data - /risc_tb/DUT/bus/regfile/r_f - Default
0000001f 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000016 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000000d 00000000 00000000 00000000 00000000 00000012 00000000 00000007 00000000 0000000b
00000004 00000001 00000044 00000019 00000000 00000000
fffffffb


00000269
0000024c
0000022f
00000212
000001f5
000001d8
000001bb
0000019e
00000181
00000164
00000147
0000012a
0000010d
000000f0
000000d3
000000b6
00000099
0000007c
0000005f
00000042
00000025
00000008