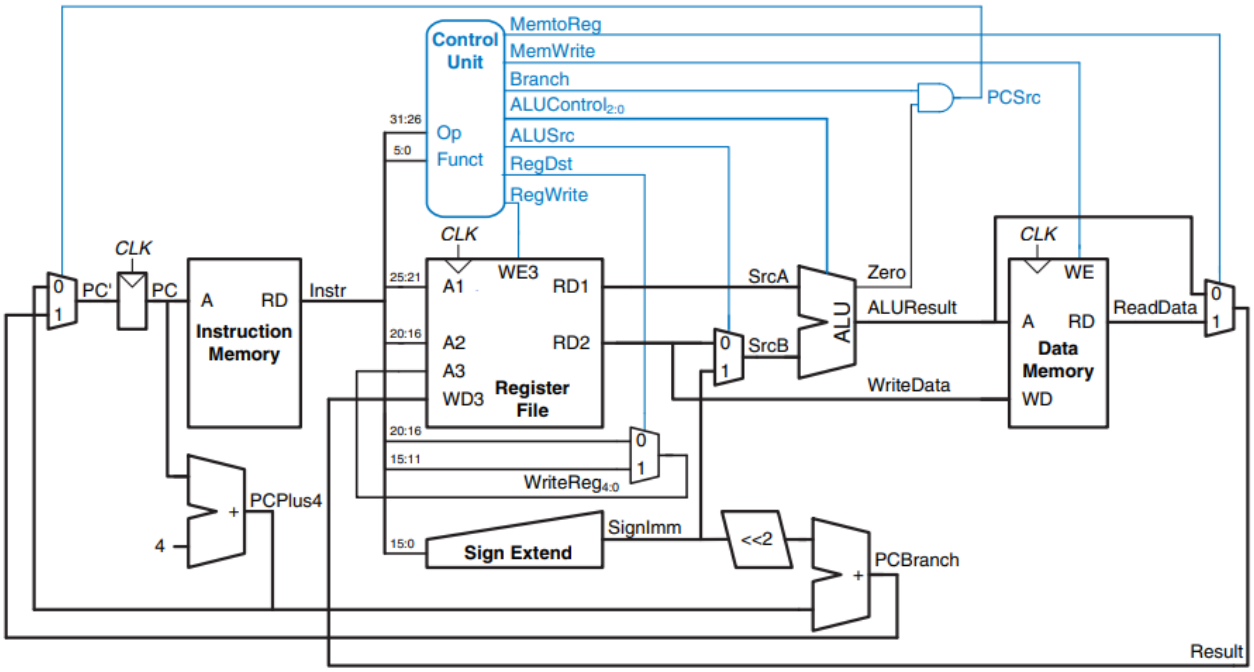
32-bit single cycle RISC-V design by Verilog

ABDELRAHMAN KHALED ELSAYED

Introduction

RISC-V (pronounced "risk-five") is an open-source instruction set architecture (ISA) that is gaining significant attention in the tech industry due to its flexibility, scalability, and growing ecosystem. Unlike traditional ISAs like x86 and ARM, which are proprietary, RISC-V is free and open, allowing developers and companies to use, modify, and implement it without licensing fees or restrictions. This open nature fosters innovation and collaboration, enabling a wide range of applications from small embedded systems to high-performance computing. RISC-V's modular design also allows for customization, making it an attractive choice for both academic research and commercial product development. As the demand for more adaptable and cost-effective computing solutions grows, RISC-V is poised to play a crucial role in the future of computing architecture.



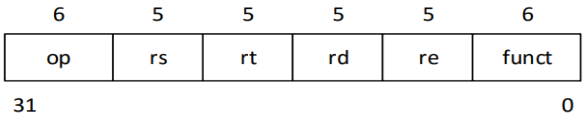
Extension Instruction Set

The RISC-V architecture features 32 registers in a register file, with x0 always set to zero. Registers x1-x31 are used for operands, supporting Boolean, signed, and unsigned integer values. It can be divided into six basic instruction formats.

- R-type: Register-Register ALU instructions.
- I-type: ALU immediate instructions, load instructions.
- S/B-types: Store instructions, comparison and branch instructions.
- U/J-types: Jump instructions, jump and link instructions.

R-Type

The R-Type format is designed for arithmetic and logical operations that involve only registers. It includes fields for the destination register, two source registers, and a function code that specifies the exact operation to be performed.

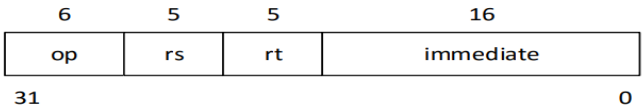


R-type instructions:

Funct	Name	Description	Operation
000000 (0)	sll rd, rt, shamt	shift left logical	[rd] = [rt] << shamt
000010 (2)	srl rd, rt, shamt	shift right logical	[rd] = [rt] >> shamt
000011 (3)	sra rd, rt, shamt	Shift right arithmetic	[rd]=[rt]>>> shamt
000100 (4)	sllv rd, rt, rs	shift left logical variable	[rd]= [rt] << [rs]4:0
000110 (6)	srlv rd, rt, rs	shift right logical variable	[rd]= [rt] >> [rs]4:0
000111 (7)	srav rd, rt, rs	shift right arithmetic variable	[rd]=[rt]>>>[rs]4:0
001000 (8)	jr rs	jump register	PC = [rs]
001001 (9)	jalr rs	jump and link register	$ra = PC + 4, PC = [rs]
010000 (16)	mfhi rd	move from hi	[rd] = [hi]
010001 (17)	mthi rs	move to hi	[hi] = [rs]
010010 (18)	mflo rd	move from lo	[rd] = [lo]
010011 (19)	mtlo rs	move to lo	[lo] = [rs]
011000 (24)	mult rs, rt	multiply	{[hi], [lo]} = [rs] × [rt]
011001 (25)	multu rs, rt	multiply unsigned	{[hi], [lo]} = [rs] × [rt]
011010 (26)	div rs, rt	divide	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
011011 (27)	divu rs, rt	divide unsigned	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
100000 (32)	add rd, rs, rt	add	[rd] = [rs] + [rt]
100001 (33)	addu rd, rs, rt	add unsigned	[rd] = [rs] + [rt]
100010 (34)	sub rd, rs, rt	subtract	[rd] = [rs] – [rt]
100011 (35)	subu rd, rs, rt	subtract unsigned	[rd] = [rs] – [rt]
100100 (36)	and rd, rs, rt	and	[rd] = [rs] & [rt]
100101 (37)	or rd, rs, rt	or	[rd] = [rs] | [rt]
100110 (38)	xor rd, rs, rt	xor	[rd] = [rs] ^ [rt]
100111 (39)	nor rd, rs, rt	nor	[rd] = ~([rs] | [rt])
101010 (42)	slt rd, rs, rt	set less than	[rs] < [rt] ? [rd] = 1 : [rd] = 0
101011 (43)	sltu rd, rs, rt	set less than unsigned	[rs] < [rt] ? [rd] = 1 : [rd] = 0

I-Type

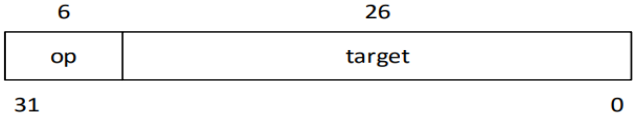
The I-Type format is used for instructions that require an immediate value, such as load instructions, arithmetic operations with an immediate operand, and some control instructions like environment calls.



Funct	Name	Description	Operation
001000 (8)	addi rt, rs, imm	add immediate	[rt]=[rs]+ SignImm
001001 (9)	addiu rt, rs, imm	add immediate unsigned	[rt]=[rs]+ SignImm
001010 (10)	slti rt, rs, imm	set less than immediate	[rs]<SignImm?[rt]=l:0
001011 (11)	sltiu rt, rs, imm	set less than immediate unsigned	[rs]<SignImm? [rt] = l : [rt] = 0
001100 (12)	andi rt, rs, imm	and immediate	[rt]=[rs]&ZeroImm
001101 (13)	ori rt, rs, imm or	immediate	[rt] = [rs]|ZeroImm
001110 (14)	xori rt, rs, imm xor	immediate	[rt]= [rs]^ZeroImm
001111 (15)	lui rt, imm	load upper immediate	[rt] = {imm, 16'b0}
100000 (32)	lb rt, imm(rs)	load byte	[rt] = SignExt ([Address]7:0)
100001 (33)	lh rt, imm(rs)	load halfword	[rt] = SignExt ([Address]15:0) 100011 (35)
100011 (35)	lw rt, imm(rs)	load word	[rt] = [Address
100100 (36)	lbu rt, imm(rs)	load byte unsigned	[rt] = ZeroExt ([Address]7:0)
100101 (37)	lhu rt, imm(rs)	load halfword unsigned	[rt] = ZeroExt ([Address]15:0)

J-Type

The J-Type format is used for jump instructions, including the jump and link (JAL) instruction. It features a destination register and a 20-bit immediate value that specifies the jump target address.



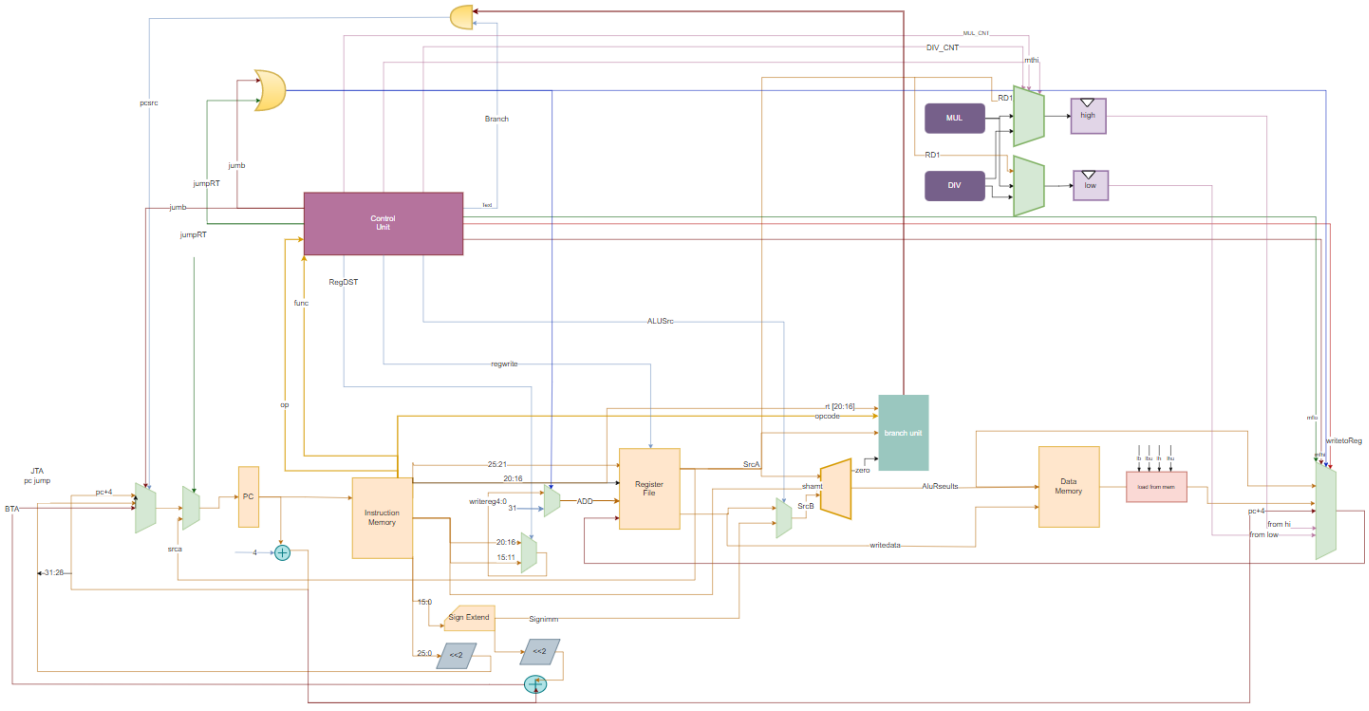
Funct	Name	Description	Operation
000011 (3)	jal label	jump and link	$ra =PC + 4, PC = JTA
000010 (2)	j label	jump	PC = JTA

B-type

The B-Type format is used for conditional branch instructions. It includes two source registers, a function code, and a sign-extended immediate value that specifies the branch target offset.

Funct	Name	Description	Operation
000001 (1) (rt = 0/1)	bltz rs, label / bgez rs, label	branch less than zero/branch greater than or equal to zero	if ([rs] < 0) PC = BTA/ if ([rs] ≥ 0) PC = BTA
000100 (4)	beq rs, rt, label	branch if equal	if ([rs] == [rt]) PC = BTA
000101 (5)	bne rs, rt, label	branch if not equal	If ([rs] != [rt]) PC = BTA
000110 (6)	blez rs, label	branch if less than or equal to zero	If ([rs] ≤ 0) PC = BTA
000111 (7)	bgtz rs, label	branch if greater than zero	if ([rs] > 0) PC = BTA

Architecture after adding instructions



Verilog code

```
module MIPS(
input clk,
input rst_n,
output wire [3:0] write_data
    );


    wire            [31:0]      pc_out;
    wire            [31:0]      pc_jumb;
    wire            [31:0]      pc_dash;
    wire            [31:0]      pcplus4;
    wire            [31:0]      pcbranch;
    wire            [31:0]      pc_dash_dash;
    wire            [31:0]      instra;
    wire            [27:0]      instra_shift;
    wire            [4:0]       des_add;
    wire            [4:0]       wr_addr3_wire;
    wire            [31:0]      result;
    wire            [31:0]      write_data3_wire;
    wire            [31:0]      srcA;
    wire            [31:0]      srcA_s_or_us;
    wire            [31:0]      read_data2_regfile;
    wire            [31:0]      sign_imm;
    wire            [33:0]      sign_imm_shift;
    wire            [31:0]      srcB;
    wire signed     [31:0]      alu_res;
    wire                        pcsrc;
    wire            [31:0]      read_data_mem;
    wire            [31:0]      hi_out;
    wire            [31:0]      lo_out;
    wire            [31:0]      div_high;
    wire            [31:0]      div_low;
    wire            [31:0]      mul_high;
    wire            [31:0]      mul_low;
    wire            [31:0]      mux_high_out;
    wire            [31:0]      mux_low_out;
    wire            [31:0]      data_out_from_mem;


    wire        reg_write;
    wire        reg_dst;
    wire        alusrc;
    wire [3:0]  alu_control;
    wire        zero;
    wire        memtoreg;
    wire        memwrite;
    wire        branch;
    wire        jump;
```

```
    wire        is_branch;
    wire        jump_RT;
    wire        is_jump;
    wire        mfhi;
    wire        mflo;
    wire        mtlo;
    wire        mthi;
    wire        mult;
    wire        multu;
    wire        div;
    wire        divu;
    wire        lb;
    wire        lbu;
    wire        lh;
    wire        lhu;

/*****************************************************************
                            data_path
*****************************************************************/


    mux4in select_pc_INS(
    .in1(pcplus4),
    .in2(pc_jumb),
    .in3(pcbranch),
    .in4(32'bx),
    .sel({pcsrc,jump}),
    .data_out(pc_dash)
    );

assign pc_dash_dash=(jump_RT==1'b1)? srcA:pc_dash;


    pc_reg PC_INST (
    .clk(clk),
    .rst_n(rst_n),
    .pc_next(pc_dash_dash),
    .pc(pc_out)
    );

    inst_mem inst_mem_INST(
    .A(pc_out),
    .RD(instra)
    );

    mux RD_MUX(
    .A(instra[20:16]),
    .B(instra[15:11]),
    .SEL(reg_dst),
    .OUT(des_add)
     );

    assign is_jump= jump_RT | jump;
    assign wr_addr3_wire = (is_jump==1'b1)? 5'b11111:des_add;


   reg_file reg_file_INST(
   .clk(clk),
   .rst_n(rst_n),
   .wr_en(reg_write),
   .rd_addr1(instra[25:21]),
   .rd_addr2(instra[20:16]),
   .wr_addr3(wr_addr3_wire),
   .wr_data3(write_data3_wire),
   .rd_data1(srcA),
   .rd_data2(read_data2_regfile)
   );
```

```
sign_ext sign_INS(
.instruction_mem(instra[15:0]),
.code(instra[31:26]),
.sign_imm(sign_imm)
);

mux srcB_MUX(
.A(read_data2_regfile),
.B(sign_imm),
.SEL(alusrc),
.OUT(srcB)
);


alu ALU_INS(
.op_code(alu_control),
.srca(srcA),
.srcb(srcB),
.shamt(instra[10:6]),
.alu_res(alu_res),
.zero_flag(zero)
);

data_mem data_mem_INS(
.clk(clk),
.rst_n(rst_n),
.wr_en(memwrite),
.addr(alu_res),
.wr_data(read_data2_regfile),
.rd_data(read_data_mem)
);


load_from_mem load_mem_INS(
.read_data(read_data_mem),
.lb(lb),  //load byte
.lbu(lbu), //unsigned
.lh(lh),
.lhu(lhu),
.data_out_from_mem(data_out_from_mem)
);




branch_unit branch_unit_INS(
.rt(instra[20:16]),
.srca(srcA),
.op_code(instra[31:26]),
.zero_flag(zero),
.is_branch(is_branch)
 );

mult mul_INS(
.in1(srcA),
.in2(read_data2_regfile),
.un_signed(multu),
.high(mul_high),
.low(mul_low)
);

div div_INS(
.in1(srcA),
.in2(read_data2_regfile),
.un_signed(divu),
.high(div_high),
.low(div_low)
);
```

```
    mux8in sel_hi_INS(
    .in1(srcA),
    .in2(mul_high),
    .in3(div_high),
    .sel({div|divu,mult|multu,mtlo}),
    .out_data(mux_high_out)
    );


    mux8in sel_lo_INS(
    .in1(srcA),
    .in2(mul_low),
    .in3(div_low),
    .sel({div|divu,mult|multu,mthi}),
    .out_data(mux_low_out)
    );

    register  HI_REG(
    .clk(clk),
    .rst_n(rst_n),
    .wr_en(mthi),
    .rd_en(mfhi),
    .D(mux_high_out),
    .Q(hi_out)
    );

    register  LO_REG(
    .clk(clk),
    .rst_n(rst_n),
    .wr_en(mtlo),
    .rd_en(mfhi),
    .D(mux_low_out),
    .Q(lo_out)
    );


    mux16in select_result_INS(
    .in1(alu_res),
    .in2(data_out_from_mem),
    .in3(pcplus4),
    .in4(hi_out),
    .in5(lo_out),
    .sel({mflo,mfhi,is_jump,memtoreg}),
    .out_data(write_data3_wire)
     );
/*****************************************************************
                        controller
/*****************************************************************/

ctrl_unit contol_INS (
.funct(instra[5:0]),
.opcode(instra[31:26]),
.mem_to_reg(memtoreg),
.mem_write(memwrite),
.branch(branch),
.alu_control(alu_control),
.alu_src(alusrc),
.reg_dst(reg_dst),
.reg_write(reg_write),
.jump(jump),
.jump_RT(jump_RT),
.mfhi(mfhi),
.mflo(mflo),
.mthi(mthi),
.mtlo(mtlo),
.mult(mult),
.multu(multu),
.div(div),
.divu(divu),
.lb(lb),  //load byte
.lbu(lbu), //unsigned
.lh(lh),
.lhu(lhu)
);
```

```
/*****************************************************************
                          pc valus
/*****************************************************************/
    assign pcplus4= pc_out+32'd4;
    assign instra_shift =instra[25:0]<<2;
    assign pc_jumb={pcplus4[31:28],instra_shift};
    assign sign_imm_shift=sign_imm<<2;
    assign pcbranch=sign_imm_shift+pcplus4;
    assign pcsrc= is_branch & branch ;


   // assign pc=pc_out;
   // assign alu_reslt=alu_res;
    //assign mem_write=memwrite;
    assign write_data=write_data3_wire[3:0];



endmodule
```

PC

```
module pc_reg (
input clk,
input rst_n,
input [31:0]pc_next,
output reg [31:0] pc
);

always @(posedge clk or negedge rst_n)begin
    if(!rst_n)  pc<=0;
    else    pc<=pc_next;

end



endmodule
```

Instruction Memory

```
module inst_mem(
input [31:0] A,
output  [31:0] RD

);
reg [7:0] mem_ins [0:1267];

initial

begin
    $readmemh("Test1.txt.mem",mem_ins);
end

assign RD={mem_ins[A+3],mem_ins[A+2],mem_ins[A+1],mem_ins[A]};


endmodule
```

Register File

```
module reg_file (
    input   wire    clk,
    input   wire    wr_en,
    input   wire    rst_n,
    input   wire    [4:0]   rd_addr1,
    input   wire    [4:0]   rd_addr2,
    input   wire    [4:0]   wr_addr3,
    input   wire    [31:0]  wr_data3,
    output  wire    [31:0]  rd_data1,
    output  wire    [31:0]  rd_data2
    );

    // MIPS has 32 registers with 32 bits

    reg [31:0] register_file    [0:31]; //little endian
    integer i;

    always @(posedge clk or negedge rst_n)
    begin
       if(~rst_n) begin
           for(i=0;i<32;i=i+1) begin
               register_file[i]<=32'b0;
           end
       end
       else if(wr_en==1'b1) begin
             register_file[wr_addr3]<=wr_data3;
       end
       else begin
           register_file[wr_addr3]<=register_file[wr_addr3];
       end
    end


    assign rd_data1=(rd_addr1==1'b0)? 32'b0:register_file[rd_addr1];
    assign rd_data2=(rd_addr2==1'b0)? 32'b0:register_file[rd_addr2];


endmodule
```

Sign Extend

```
module sign_ext(
input [15:0] instruction_mem,
input [5:0] code,
output reg [31:0] sign_imm
    );
    always @(*)begin
        if(code ==6'b001001 || code ==6'b001100 ||code==6'b001101 || code==6'b001110) begin //  andi  ori xori
            sign_imm = {16'b0,instruction_mem};
        end
        else if(code==6'b001111) begin //lui
            sign_imm = {instruction_mem,16'b0};
        end
        else begin
            sign_imm = {{16{instruction_mem[15]}},instruction_mem};
        end
    end


endmodule
```

Data Memory

```
module data_mem(
input clk,
input rst_n,
input wr_en,
input [9:0] addr,
input [31:0] wr_data,
output [31:0] rd_data

);
//reg [31:0] mem[0:1023];

//reg [7:0] mem[0:1023];
reg [7:0] mem[0:1267];

integer i;


always @(posedge clk or negedge rst_n) begin
    if(~rst_n)begin
        for(i=0;i<1024;i=i+1)begin
            mem[i]<=32'b0;
        end
    end
    else if(wr_en)begin
        {mem[addr+3],mem[addr+2],mem[addr+1],mem[addr]}<=wr_data;

    end
    else begin
        mem[addr]<=mem[addr];
    end
end


assign rd_data={mem[addr+3],mem[addr+2],mem[addr+1],mem[addr]};


endmodule
```

ALU

```
module alu (
    input   wire        [3:0]   op_code,
    input   wire        [31:0]  srca,
    input   wire       [31:0]   srcb,
    input   wire       [4:0]    shamt,

    output  reg signed [31:0]    alu_res,
    output  wire                 zero_flag
    );


wire   [31:0]   srca_unsigned;
wire   [31:0]   srcb_unsigned;
wire   [31:0]   alu_res_temp;

assign srca_unsigned=(srca[31]==1'b1)? (~srca + 1'b1):srca;
assign srcb_unsigned=(srcb[31]==1'b1)? (~srcb + 1'b1):srcb;
assign alu_res_temp= srca -     srcb;

    always @(*)begin
        case(op_code)
        4'b0000:    alu_res=    srca &  srcb;
        4'b0001:    alu_res=    srca |  srcb;
        4'b0010:    alu_res=    srca +  srcb;
        4'b0011:                        ;
        4'b0100:    alu_res=    srca ^  srcb;
        4'b0101:    alu_res=  ~(srca |  srcb);
        4'b0110:    alu_res=   alu_res_temp;  //sub
```

```
        4'b0111:begin//slt
                    alu_res=   alu_res_temp;
                    alu_res={{31{1'b0}},alu_res[31]};
                end
        4'b1000: begin //sltu
                    alu_res= srca_unsigned - srcb_unsigned ;
                    alu_res={{31{1'b0}},alu_res[31]};
                 end
        4'b1001:    alu_res= srcb << shamt  ;//sll
        4'b1010:    alu_res= srcb >> shamt   ;//slr
        4'b1011:    alu_res= srcb >>> shamt  ;//sra
        4'b1100:    alu_res= srcb << srca    ; //sllv
        4'b1101:    alu_res= srcb >> srca    ;//srlv
        4'b1110:    alu_res= srcb >>> srca   ;//srav
        4'b1111:                            ;
        default:alu_res=    srca ;
       endcase
    end

    assign zero_flag = (alu_res==32'b0)? 1'b1:1'b0;



endmodule
```

Branch unit

```
module branch_unit(

input [4:0]    rt,
input [31:0]   srca,
input [5:0]    op_code,
input          zero_flag,

output  reg      is_branch

    );

    wire [31:0] temp;

    assign temp=srca-32'b0;


    always@(*)begin
        case(op_code)
            6'b000100:begin //beg
                if(zero_flag ==1'b1)    is_branch=1'b1;
                else    is_branch=1'b0;
            end

            6'b000101:begin //bneg
                 if(zero_flag ==1'b0)    is_branch=1'b1;
                 else    is_branch=1'b0;
            end

            6'b000001:begin //bltz or bgez
                 if((rt==5'b0) && (temp[31]==1'b1))    is_branch=1'b1; //bltz
                 else if ((rt==5'b00001) && (temp[31]==1'b0))    is_branch=1'b1; //bgez
                 else    is_branch=1'b0;
            end
            6'b000110:begin //blez
                 if((temp==32'b0) || (temp[31]==1'b1))    is_branch=1'b1;
                 else    is_branch=1'b0;
            end

            6'b000111:begin //bgtz
                 if(temp[31]==1'b0)    is_branch=1'b1;
                 else    is_branch=1'b0;
            end
            default:         is_branch=1'b0;

        endcase


    end

endmodule
```

Multiplier

```
module mult(
input    [31:0] in1,
input    [31:0] in2,
input           un_signed,
output   [31:0] high,
output   [31:0] low
    );

    wire   [63:0]   res;

     wire    [31:0]  in1_s_un;
     wire    [31:0]  in2_s_un;

       assign in1_s_un =(un_signed==1'b1)? (~in1 + 1'b1) : in1;
       assign in2_s_un =(un_signed==1'b1)? (~in2 + 1'b1) : in2;


    assign res  =   in1_s_un *  in2_s_un;
    assign high = res[63:32];
    assign low  = res[31:0];


endmodule
```

division

```
module div(
input     [31:0]  in1,
input     [31:0]  in2,
input             un_signed,
output    [31:0]  high,
output    [31:0]  low
    );

    wire    [31:0]  in1_s_un;
    wire    [31:0]  in2_s_un;

    assign in1_s_un =(un_signed==1'b1)? (~in1 + 1'b1) : in1;
    assign in2_s_un =(un_signed==1'b1)? (~in2 + 1'b1) : in2;

    assign low =  in1_s_un / in2_s_un;
    assign high= in1_s_un % in2_s_un;

endmodule
```

High and low register

```
module register(
input               clk,
input               rst_n,
input               wr_en,
input               rd_en,
input      [31:0]   D,
output     [31:0]   Q
    );

   reg [31:0] new_reg;
    always @(posedge clk or negedge rst_n)begin
        if(~rst_n)          new_reg<=32'b0;
        else if(wr_en)      new_reg<=D;
        else                new_reg<=new_reg;

    end


    assign Q = (rd_en==1'b1)?   new_reg:32'b0;

endmodule
```

Control Unit

```
module ctrl_unit (
// inputs
input wire [5:0] funct,
input wire [5:0] opcode,
// outputs
output reg mem_to_reg,  //select out from mem or not
output reg mem_write,   //control to write in mem
output reg branch,      //control for B type
output reg [3:0] alu_control,
output reg alu_src,
output reg reg_dst,     //select rt,rd
output reg reg_write,   //control to write in RF
output reg jump,        //jumb for i type
output reg jump_RT,     //jumb for R type
output reg mfhi,        //move from high signal
output reg mflo,        //move from low signal
output reg div,         //division signal cntrol
output reg mthi,        //move to high signal
output reg mtlo,        //move to low signal
output reg divu,        //division unsigned cntrol
output reg mult,        //multiplier control signal
output reg multu,       //multiplier unsigned control signal
output reg lb,          //load byte signal control
output reg lbu,         //load unsigned byte signal control
output reg lh,          //load half word signal control
output reg lhu          //load unsigned half word  signal control

);

reg [1:0] alu_op;
reg [2:0] extra_control; //for any type not R type
always @(*)begin

    case(opcode)
    /*************************************************************************************

                                    R TYPE

    *************************************************************************************/
        6'b000000:begin
                    reg_write       =1'b1;
                    reg_dst         =1'b1;
                    alu_src         =1'b0;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b10;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                case(funct)
                    6'b001001: begin //JALR
                                jump_RT         =1'b1;
                                reg_write       =1'b1;
                    end

                    6'b001000 : begin //jr i don't write to reg file only read ra
                                 reg_write      =1'b0;
                                 jump_RT        =1'b1;
                                 end
                    6'b010000 : begin //mfhi
                                  mfhi          =1'b1;
                                 end
                   6'b010010 : begin //mflo
                                  mflo          =1'b1;
                                 end
```

```
                    6'b010001 : begin //mthi
                                mthi          =1'b1;
                                end
                    6'b010011 : begin //mtlo
                                mtlo          =1'b1;
                                end
                    6'b011000 : begin //mult
                                mult          =1'b1;
                                mthi          =1'b1;
                                mtlo          =1'b1;
                                end
                    6'b011001 : begin //multu
                                multu         =1'b1;
                                mthi          =1'b1;
                                mtlo          =1'b1;
                                end
                    6'b011010 : begin //div
                                div           =1'b1;
                                mthi          =1'b1;
                                mtlo          =1'b1;
                                end
                    6'b011011 : begin //divu
                                divu          =1'b1;
                                mthi          =1'b1;
                                mtlo          =1'b1;
                                end
                endcase
        end
/*************************************************************************************************

                                Load Store

*************************************************************************************************/
        6'b100011:begin //lw
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b1;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end

        6'b101011:begin//sw
                    reg_write       =1'b0;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b1;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
```

```
/*****************************************************************************************
                                I TYPE
*****************************************************************************************/

        6'b001000:begin//addi
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end

        6'b001001:begin//addiu unsigned
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
        6'b001100 :begin //andi
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b001; //for using and
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
        6'b001101 :begin //ori
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00; //sub in alu
                    jump_RT         =1'b0;
                    extra_control   =3'b010; //for using or
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
```

```
        6'b001110 :begin //xori
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00; //sub in alu
                    jump_RT         =1'b0;
                    extra_control   =3'b011; //for using or
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end


        6'b001010 :begin //slti
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b100;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
            end
         6'b001011 :begin //sltiu
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b101;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end

    /*****************************************************************************************
                                    Branch
    *****************************************************************************************/
         6'b000100:begin//beq
                    reg_write       =1'b0;
                    reg_dst         =1'b0;
                    alu_src         =1'b0;
                    branch          =1'b1;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b01;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
```

```
        6'b000101:begin//bneq
                    reg_write       =1'b0;
                    reg_dst         =1'b0;
                    alu_src         =1'b0;
                    branch          =1'b1;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b01;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end

        6'b000001 :begin //bltz or bgez
                    reg_write       =1'b0;
                    reg_dst         =1'b0;
                    alu_src         =1'b0;
                    branch          =1'b1;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b01;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
        6'b000110 :begin //blez
                    reg_write       =1'b0;
                    reg_dst         =1'b0;
                    alu_src         =1'b0;
                    branch          =1'b1;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b01;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
        6'b000111 :begin //bgtz
                    reg_write       =1'b0;
                    reg_dst         =1'b0;
                    alu_src         =1'b0;
                    branch          =1'b1;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
```

```
/*****************************************************************************************

                                J TYPE

*****************************************************************************************/

        6'b000010:begin//j
                    reg_write       =1'b0;
                    reg_dst         =1'b0;
                    alu_src         =1'b0;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b1;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
        6'b000011:begin//jal
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b0;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b1;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b0;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end

/*****************************************************************************************

                                load

*****************************************************************************************/

        6'b001111 :begin //lui
                    reg_write       =1'b1;
                    reg_dst         =1'b0;
                    alu_src         =1'b1;
                    branch          =1'b0;
                    mem_write       =1'b0;
                    mem_to_reg      =1'b0;
                    jump            =1'b0;
                    alu_op          =2'b00;
                    jump_RT         =1'b0;
                    extra_control   =3'b000;
                    mfhi            =1'b0;
                    mflo            =1'b0;
                    mthi            =1'b0;
                    mtlo            =1'b0;
                    mult            =1'b0;
                    multu           =1'b0;
                    div             =1'b0;
                    divu            =1'b0;
                    lb              =1'b0;
                    lbu             =1'b0;
                    lh              =1'b0;
                    lhu             =1'b0;
                end
```

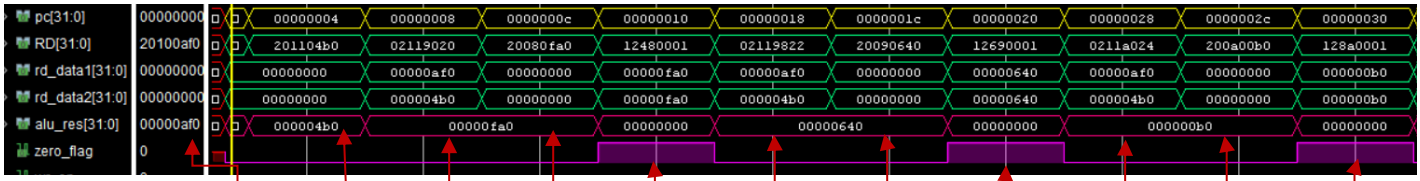
```
6'b100000 :begin //lb
            reg_write       =1'b1;
            reg_dst         =1'b0;
            alu_src         =1'b1;
            branch          =1'b0;
            mem_write       =1'b0;
            mem_to_reg      =1'b1;
            jump            =1'b0;
            alu_op          =2'b00;
            jump_RT         =1'b0;
            extra_control   =3'b000;
            mfhi            =1'b0;
            mflo            =1'b0;
            mthi            =1'b0;
            mtlo            =1'b0;
            mult            =1'b0;
            multu           =1'b0;
            div             =1'b0;
            divu            =1'b0;
            lb              =1'b1;
            lbu             =1'b0;
            lh              =1'b0;
            lhu             =1'b0;

          end

6'b100100 :begin //lbu
            reg_write       =1'b1;
            reg_dst         =1'b0;
            alu_src         =1'b1;
            branch          =1'b0;
            mem_write       =1'b0;
            mem_to_reg      =1'b1;
            jump            =1'b0;
            alu_op          =2'b00;
            jump_RT         =1'b0;
            extra_control   =3'b000;
            mfhi            =1'b0;
            mflo            =1'b0;
            mthi            =1'b0;
            mtlo            =1'b0;
            mult            =1'b0;
            multu           =1'b0;
            div             =1'b0;
            divu            =1'b0;
            lb              =1'b0;
            lbu             =1'b1;
            lh              =1'b0;
            lhu             =1'b0;
          end
6'b100001 :begin //lh
             reg_write       =1'b1;
             reg_dst         =1'b0;
             alu_src         =1'b1;
             branch          =1'b0;
             mem_write       =1'b0;
             mem_to_reg      =1'b1;
             jump            =1'b0;
             alu_op          =2'b00;
             jump_RT         =1'b0;
             extra_control   =3'b000;
             mfhi            =1'b0;
             mflo            =1'b0;
             mthi            =1'b0;
             mtlo            =1'b0;
             mult            =1'b0;
             multu           =1'b0;
             div             =1'b0;
             divu            =1'b0;
             lb              =1'b0;
             lbu             =1'b0;
             lh              =1'b1;
             lhu             =1'b0;
         end


6'b100101 :begin //lhu
             reg_write       =1'b1;
             reg_dst         =1'b0;
             alu_src         =1'b1;
             branch          =1'b0;
             mem_write       =1'b0;
             mem_to_reg      =1'b1;
             jump            =1'b0;
             alu_op          =2'b00;
             jump_RT         =1'b0;
             extra_control   =3'b000;
             mfhi            =1'b0;
             mflo            =1'b0;
             mthi            =1'b0;
             mtlo            =1'b0;
             mult            =1'b0;
             multu           =1'b0;
             div             =1'b0;
             divu            =1'b0;
             lb              =1'b0;
             lbu             =1'b0;
             lh              =1'b0;
             lhu             =1'b1;
          end
```

```
            6'b100011 :begin //
                        reg_write       =1'b1;
                        reg_dst         =1'b0;
                        alu_src         =1'b1;
                        branch          =1'b0;
                        mem_write       =1'b0;
                        mem_to_reg      =1'b1;
                        jump            =1'b0;
                        alu_op          =2'b00;
                        jump_RT         =1'b0;
                        extra_control   =3'b000;
                        mfhi            =1'b0;
                        mflo            =1'b0;
                        mthi            =1'b0;
                        mtlo            =1'b0;
                        mult            =1'b0;
                        multu           =1'b0;
                        div             =1'b0;
                        divu            =1'b0;
                        lb              =1'b0;
                        lbu             =1'b0;
                        lh              =1'b0;
                        lhu             =1'b0;
                    end




            default:begin
                        reg_write       =1'b0;
                        reg_dst         =1'b0;
                        alu_src         =1'b0;
                        branch          =1'b0;
                        mem_write       =1'b0;
                        mem_to_reg      =1'b0;
                        jump            =1'b0;
                        alu_op          =2'b00;
                        jump_RT         =1'b0;
                        extra_control   =3'b0;
                        mfhi            =1'b0;
                        mflo            =1'b0;
                        mthi            =1'b0;
                        mtlo            =1'b0;
                        mult            =1'b0;
                        multu           =1'b0;
                        div             =1'b0;
                        divu            =1'b0;
                        lb              =1'b0;
                        lbu             =1'b0;
                        lh              =1'b0;
                        lhu             =1'b0;
                end
    endcase
end
```

```
  always @(*) begin

  casex({alu_op ,funct,extra_control})

  11'b00_xxxxxx_000 :begin alu_control = 4'b0010;end //add
  11'bx1_xxxxxx_000 :begin alu_control = 4'b0110;end //subtract
  11'b1x_100000_000 :begin alu_control = 4'b0010;end //add
  11'b1x_100010_000 :begin alu_control = 4'b0110;end //subtract
  11'b1x_100100_000 :begin alu_control = 4'b0000;end //and
  11'b1x_100101_000 :begin alu_control = 4'b0001;end //or
  11'b1x_101010_000 :begin alu_control = 4'b0111;end //slt
  11'b10_100001_000 :begin alu_control = 4'b0010;end //add for addu
  11'b10_100011_000 :begin alu_control = 4'b0110;end //sub for subu
  11'b10_100110_000 :begin alu_control = 4'b0100;end //xor
  11'b10_100111_000 :begin alu_control = 4'b0101;end //nor
  11'b1x_101011_000 :begin alu_control = 4'b1000;end //sltu
  11'b10_000000_000 :begin alu_control = 4'b1001;end //sll
  11'b10_000010_000 :begin alu_control = 4'b1010;end //srl
  11'b10_000011_000 :begin alu_control = 4'b1011;end //sra
  11'b10_000100_000 :begin alu_control = 4'b1100;end //sllv
  11'b10_000110_000 :begin alu_control = 4'b1101;end //srlv
  11'b10_000111_000 :begin alu_control = 4'b1110;end //srav
  11'bxx_xxxxxx_001 :begin alu_control = 4'b0000;end //andi
  11'bxx_xxxxxx_010 :begin alu_control = 4'b0001;end //ori
  11'bxx_xxxxxx_011 :begin alu_control = 4'b0100;end //xori
  11'bxx_xxxxxx_100 :begin alu_control = 4'b0111;end //slti
  11'bxx_xxxxxx_101 :begin alu_control = 4'b1000;end //sltiu

  default : alu_control = 4'b0010; //default

  endcase
  end

  endmodule
```

Test bench simulation results

Tast 1:




```
addi $s0, $0, 0x0AF0 //2800
addi $s1, $0, 0x04B0 //1200
add $s2, $s0, $s1
addi $t0, $0, 0x0FA0 //Result
beq $s2, $t0, NEXT1
j ERROR
NEXT1: sub $s3, $s0, $s1
addi $t1, $0, 0x0640 //Result
beq $s3, $t1, NEXT2
j ERROR
NEXT2: and $s4, $s0, $s1
addi $t2, $0, 0x00B0 //Result
beq $s4, $t2, NEXT3
j ERROR
NEXT3: or $s5, $s0, $s1
addi $t3, $0, 0x0EF0 //Result
beq $s5, $t3, NEXT4
j ERROR
NEXT4: addi $t4, $0, 0x0064
sw $s0, 50($t4)
lw $t5, 50($t4)
beq $s0, $t5, DONE
ERROR: addi $s0, $0, 0XDEAD
j SKIP
DONE: addi $s0, $0, 0XD08E
SKIP: nop
```

Tast 2

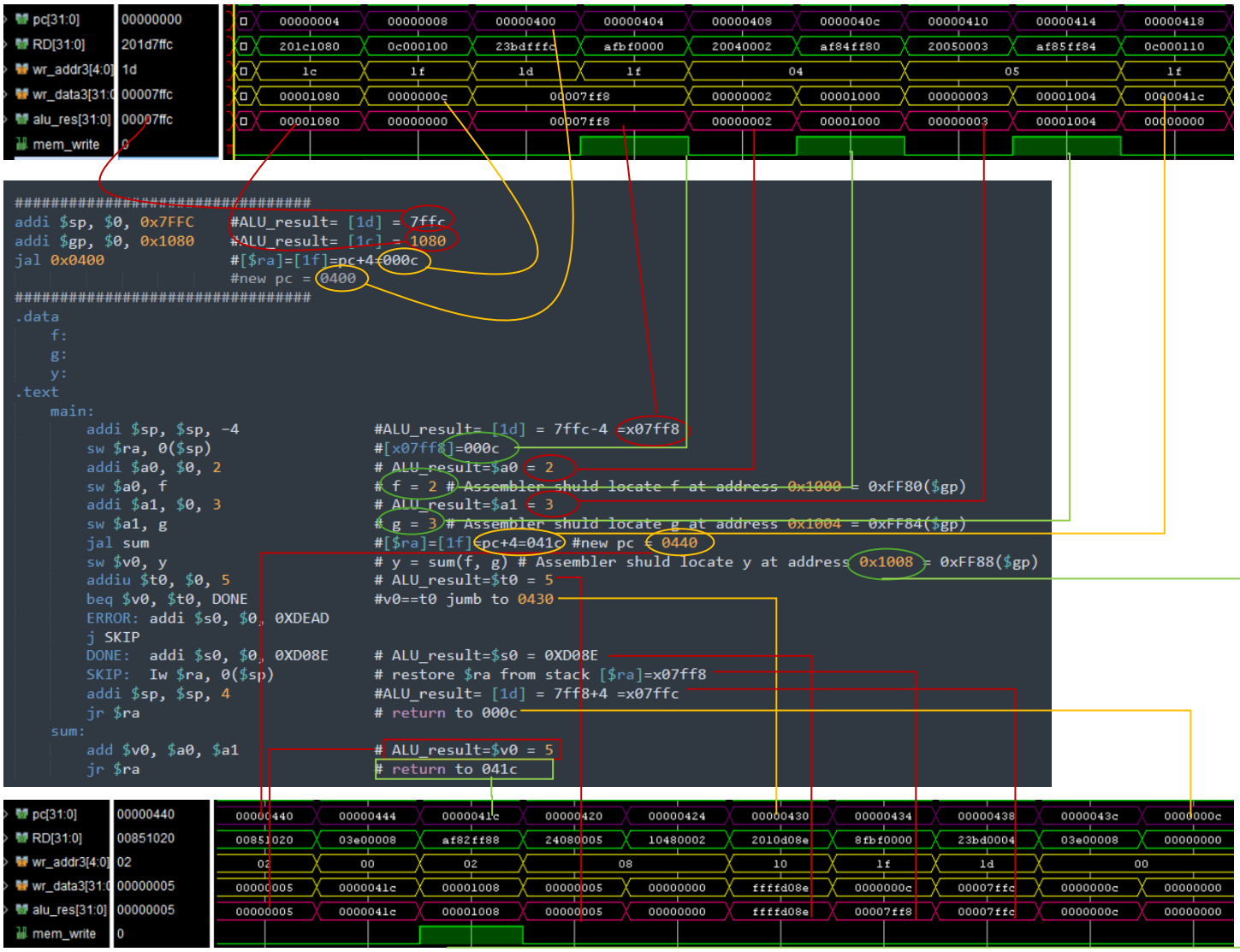
To test the Function Calls and Returns by using instructions like JAL, JR.

```c
int f, g, y; // global variables
int main(void)
{
f = 2;
g = 3;
y = sum(f, g);
return y;
}
int sum(int a, int b) {
return (a + b);
}
```

Machine code

```
201d7ffc    ;0x0000_0000: addi $sp, $0, 0x7FFC
201c1080    ;0x0000_0004: addi $gp, $0, 0x1080
0c000100    ;0x0000_0008: jal 0x0400
00000000    ;0x0000_000c:
00000000    ;0x0000_03e8:
00000000    ;0x0000_03ec:
00000000    ;0x0000_03f0:
00000000    ;0x0000_03f4:
00000000    ;0x0000_03f8:
00000000    ;0x0000_03fc:
23bdfffc    ;0x0000_0400: main:    addi $sp, $sp, -4
afbf0000    ;0x0000_0404:          sw $ra, 0($sp)
20040002    ;0x0000_0408:          addi $a0, $0, 2
af84ff80    ;0x0000_040c:          sw $a0, f
20050003    ;0x0000_0410:          addi $a1, $0, 3
af85ff84    ;0x0000_0414:          sw $a1, g
0c000110    ;0x0000_0418:          jal sum
af82ff88    ;0x0000_041c:          sw $v0, y
24080005    ;0x0000_0420:          addiu $t0, $0, 5
10480002    ;0x0000_0424:          beq $v0, $t0, DONE
2010dead    ;0x0000_0428:          ERROR: addi $s0, $0, 0XDEAD
0800010d    ;0x0000_042c:          j SKIP
2010d08e    ;0x0000_0430:          DONE:  addi $s0, $0, 0XD08E
8fbf0000    ;0x0000_0434:          SKIP:  Iw $ra, 0($sp)
23bd0004    ;0x0000_0438:          addi $sp, $sp, 4
03e00008    ;0x0000_043c:          jr $ra
00851020    ;0x0000_0440: sum:     add $v0, $a0, $a1
03e00008    ;0x0000_0444:          jr $ra
```

Test bench simulation results



Comments:

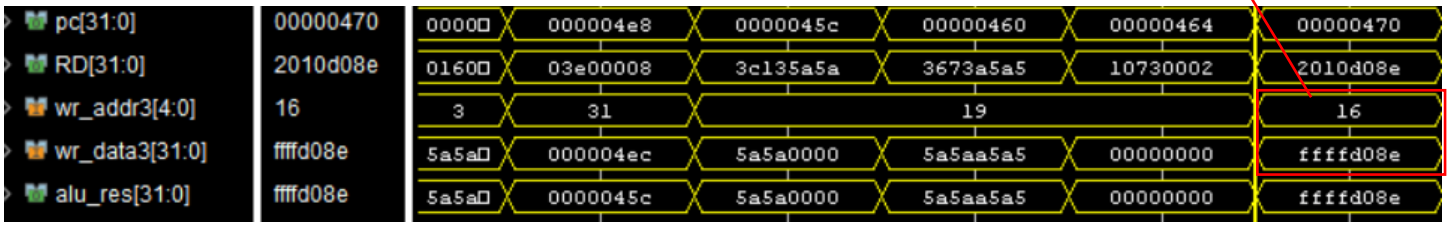
- Jal affects the pc as store the pc+4 in $ra and jump to new location x0440.
- Jr return to old location by using stored location in $ra x041c.
- At the end of program, the values 2,3,5 will write in memory.

Tast 3

This Test is created to: Verify Conditional & Unconditional Branch Instructions: bltz, bgez, blez, bgtz, beq, bne, j, jal, jr, jalr

2-Verify the Arithmetic-Logical Instructions: add, addu, sub, subu, and, or, xor, nor, slt, sltu

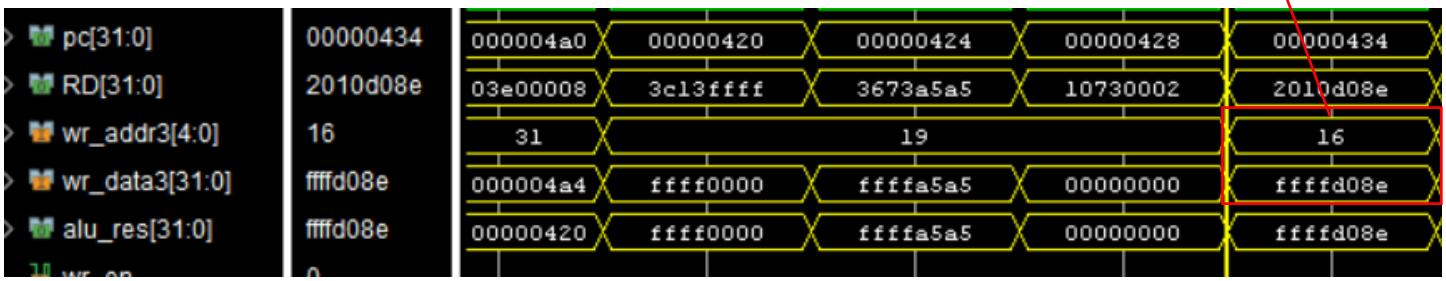
```
addi $sp, $0, 0x7FFC            # $sp is located in the Reg file at address 29
addi $gp, $0, 0x1080            # $gp is located in the Reg file at address 28
jal main                        # main is located at 0x0400
################################
main:
    addi $sp, $sp, -4                   # Move $sp one location to store thre return address.
    sw $ra, 0($sp)                      # store $ra (return address of OS before starting the main funtion) on stack
    ################################
    addi $a0, $0, 0xFFFC                # $a0=-4
    addi $a1, $0, 0xFFF0                # $a1=-16
    addi $a2, $0, 0xFFF6                # $a2=-10
    addi $a3, $0, 0xFFD8                # $a3=-40
    jal signed_diffofsums               # $v0=[(-4)+(-16)]-[(-10)+(-40)]=30
    addi $s1, $0, 0x001E
    beq $v0, $s1, Next1
    j ERROR
    ################################
    Next1: addi $a0, $0, 0x2710         # $a0=10,000
    addi $a1, $0, 0x07D0                # $a1=2,000
    addi $a2, $0, 0x1B58                # $a2=7,000
    addi $a3, $0, 0x1B58                # $a3=7,000
    addi $s7, $0, unsigned_diffofsums   # $v0=[(10,000)+(2,000)]-[(7,000)+(7,000)]=-2,000
    jalr $s7
    addi $s2, $0, 0xF830
    bne $v0, $s2, ERROR
    ################################
    lui $a0, $0, 0xA5A5
    ori $a0, $a0, 0x5A5A                # $a0 = 0xA5A55A5A
    lui $a1, $0, 0xAAAA
    ori $a1, $a1, 0x5555                # $a1 = 0xAAAA5555
    jal logical_op
    lui $s3, $0, 0x5A5A
    ori $s3, $s3, 0xA5A5                # $s3 = 0x5A5AA5A5
    beq $v1, $s3, DONE
    ################################
    ERROR: addi $s0, $0, 0XDEAD
    j SKIP
    DONE:  addi $s0, $0, 0XD08E
    SKIP:  lw $ra, 0($sp)               # restore $ra from stack
    addi $sp, $sp, 4                    # restore stack pointer
    jr $ra                              # return to operating system
################################
signed_diffofsums:
    addi $sp, $sp, -4           # make space on stack to store one register
    sw $s0, 0($sp)              # save $s0 on stack
    add $t0, $a0, $a1           # $t0 = f + g
    add $t1, $a2, $a3           # $t1 = h + i
    sub $s0, $t0, $t1           # result = (f + g) - (h + i)
    add $v0, $s0, $0            # put return value in $v0
    lw $s0, 0($sp)              # restore $s0 from stack
    addi $sp, $sp, 4            # deallocate stack space
    jr $ra                      # return to caller
unsigned_diffofsums:
    addu $t0, $a0, $a1          # $t0 = f + g
    addu $t1, $a2, $a3          # $t1 = h + i
```



Test4

This Test is created to verify the Arithmatic-Logical with Immediate Instructions: slti, sltiu, andi, xori, ori, addi, addiu

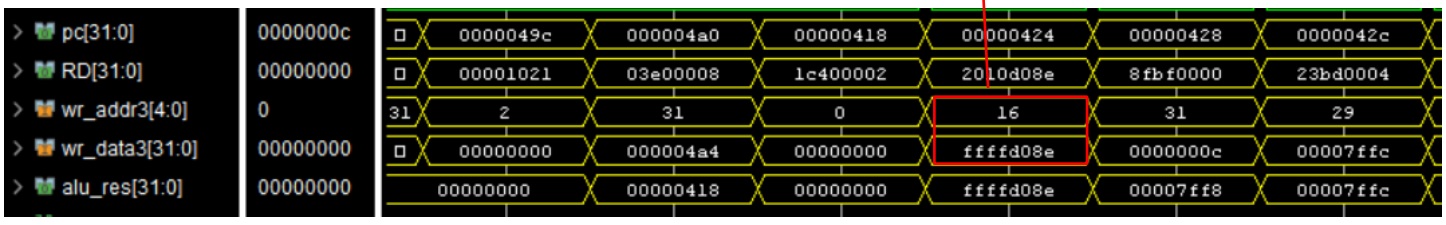
```
addi $sp, $0, 0x7FFC            # $sp is located in the Reg file at address 29
addi $gp, $0, 0x1080            # $gp is located in the Reg file at address 28
jal main                        # main is located at 0x0400
################################
main:
    addi $sp, $sp, -4                   # Move $sp one location to store thre return address.
    sw $ra, 0($sp)                      # store $ra (return address of OS before starting the main funtion) on stack
    ################################
    lui $a0, 0xA5A5
    ori $a0, $a0, 0x5A5A                # $a0 = 0xA5A55A5A (unsinged=+2,779,077,210) or (singed=-0x5A5A_A5A6=-1,515,890,086)
    jal arith_add_imm
    bgtz $v0, NEXT
    j ERROR
    NEXT:jal logical_op_imm
    lui $s3, 0x5A5A
    ori $s3, $s3, 0xA5A5                # $s3 = 0x5A5AA5A5
    beq $v1, $s3, DONE
    ################################
    ERROR: addi $s0, $0, 0XDEAD
    j SKIP
    DONE:  addi $s0, $0, 0XD08E
    SKIP:  lw $ra, 0($sp)               # restore $ra from stack
    addi $sp, $sp, 4                    # restore stack pointer
    jr $ra                              # return to operating system
################################
arith_add_imm:
    addi  $t0, $a0, 0xAAAA          # $t0 = 0xA5A6_0504,    # $t0 = (-1,515,890,086+43,690 = 1,515,846,396 = -0x5A59_FAFC)      (-0x0504_FAFC = -84,212,476)
    addiu $t1, $a0, 0xAAAA          # $t1 = 0xA5A6_0504,    # $t1 = (+2,779,077,210+43,690 = 2,779,120,900)
    lui $t2, 0xA5A6
    ori $t2, $t2, 0x0504
    beq $t0, $t2, NEXT_ADD
    j END_ADD
    NEXT_ADD: beq $t1, $t2, NEXT2_ADD
    j END_ADD
    NEXT2_ADD: addiu $v0, $t3, 0x0001
    jr $ra
    END_ADD:addiu $v0, $0, 0x0000
    jr $ra
logical_op_imm:
    andi $t0, $a0, 0x5555           # $a0 = 0xA5A55A5A, $t0 = 0x00005050
    xori $t1, $t0, 0x5A5A           # $t1 = 0x00000A0A
    ori $t2, $t1, 0x5050            # $t2 = 0x00005A5A
    nor $t3, $t1, $t2               # $t3 = 0xFFFFA5A5
    lui $t3, 0x5A5A                 # $t3 = 0x5A5AA5A5
    slti $t4, $t0, 0x0A0A           # $t4 = 0x1
    bgtz $t4, OK
    j END
    OK: sltiu $t5, $t0, 0xFFFF      # $t5 = 0x0
    bgez $t5, OK2
    j END
    OK2: addu $v1, $t3, $0
    jr $ra
    END:addu $v1, $0, $0
    jr $ra
```



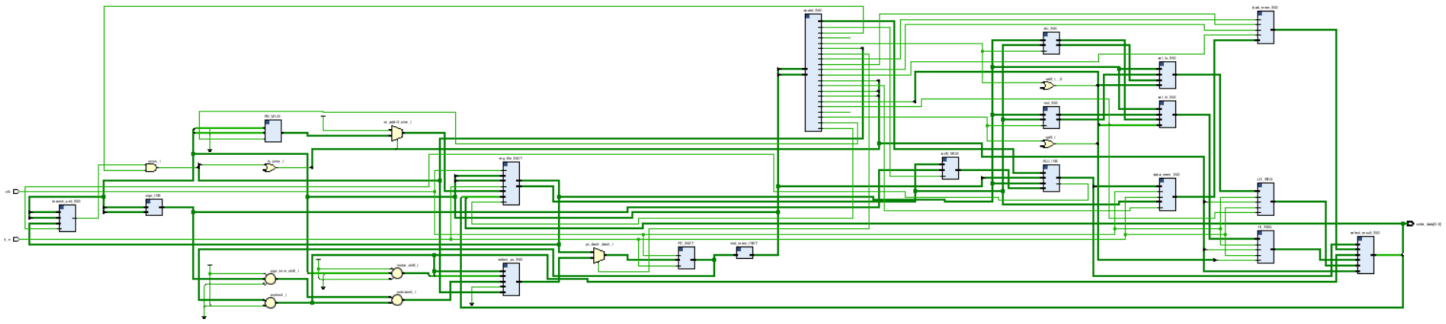
Test5

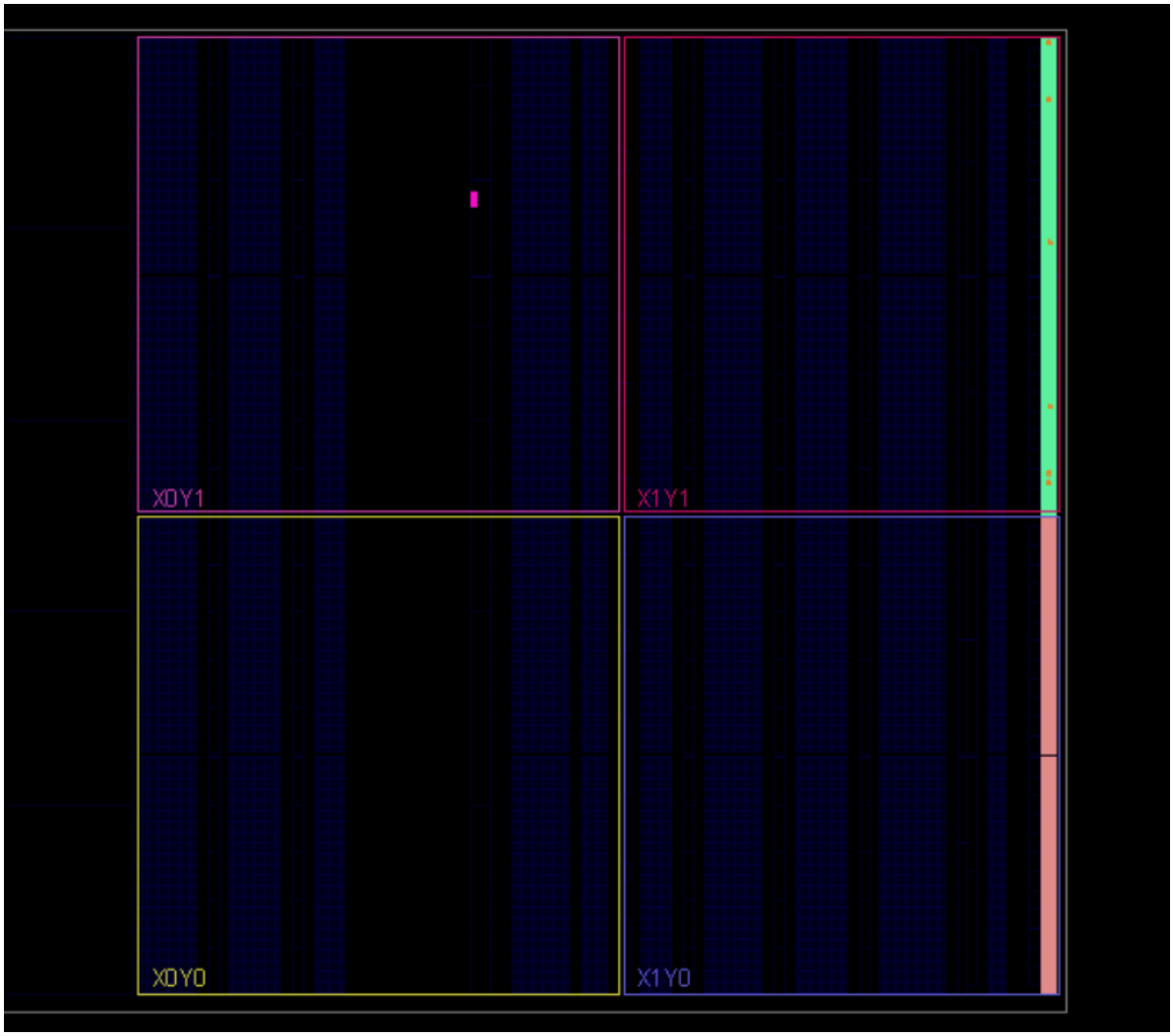
This Test is created to verify the Shift Instructions: sll, srl, sra, sllv, srlv, srav

```
addi $sp, $0, 0x7FFC            # $sp is located in the Reg file at address 29
addi $gp, $0, 0x1080            # $gp is located in the Reg file at address 28
jal main                        # main is located at 0x0400
################################
main:
    addi $sp, $sp, -4                   # Move $sp one location to store thre return address.
    sw $ra, 0($sp)                      # store $ra (return address of OS before starting the main funtion) on stack
    ################################
    lui $a0, 0xA5A5
    ori $a0, $a0, 0x5A5A                # $a0 = 0xA5A55A5A
    addi $a1, $0, 0xACAC                # $a1 = 0x0000ACAC  =>  $a1[4:0] = 12
    jal shift_instr
    bgtz $v0, DONE
    ################################
    addi $s0, $0, 0XDEAD
    j SKIP
    DONE:  addi $s0, $0, 0XD08E
    SKIP:  lw $ra, 0($sp)               # restore $ra from stack
    addi $sp, $sp, 4                    # restore stack pointer
    jr $ra                              # return to operating system
################################
shift_instr:
    sll  $t0, $a0, 12           # $t0 = 0xA5A5_5A5A <<  12 = 0x55A5_A000
    sllv $t1, $a0, $a1          # $t1 = 0xA5A5_5A5A <<  12 = 0x55A5_A000
    lui  $t2, 0x55A5
    ori  $t2, $t2, 0xA000       # $t2 = 0x55A5_A000
    beq  $t2, $t0, SH_NEXT1
    j END
    SH_NEXT1: beq  $t2, $t1, SH_NEXT2
    j END
    ################################
    SH_NEXT2:
    srl  $t0, $a0, 12           # $t0 = 0xA5A5_5A5A >>  12 = 0x000A_5A55
    srlv $t1, $a0, $a1          # $t1 = 0xA5A5_5A5A >>  12 = 0x000A_5A55
    lui  $t2, 0x000A
    ori  $t2, $t2, 0x5A55       # $t2 = 0x000A_5A55
    beq  $t2, $t0, SH_NEXT3
    j END
    SH_NEXT3: beq  $t2, $t1, SH_NEXT4
    j END
    ################################
    SH_NEXT4:
    sra  $t0, $a0, 12           # $t0 = 0xA5A5_5A5A >>> 12 = 0xFFFA_5A55
    srav $t1, $a0, $a1          # $t1 = 0xA5A5_5A5A >>> 12 = 0xFFFA_5A55
    lui  $t2, 0xFFFA
    ori  $t2, $t2, 0x5A55       # $t2 = 0xFFFA_5A55
    beq  $t2, $t0, SH_NEXT5
    j END
    SH_NEXT5: beq  $t2, $t1, SH_DONE
    j END
    ################################
    SH_DONE: addiu $v0, $0, 0x0001
    jr $ra
    END:addu $v0, $0, $0
    jr $ra
```



Elaborate and synthesis





Summary

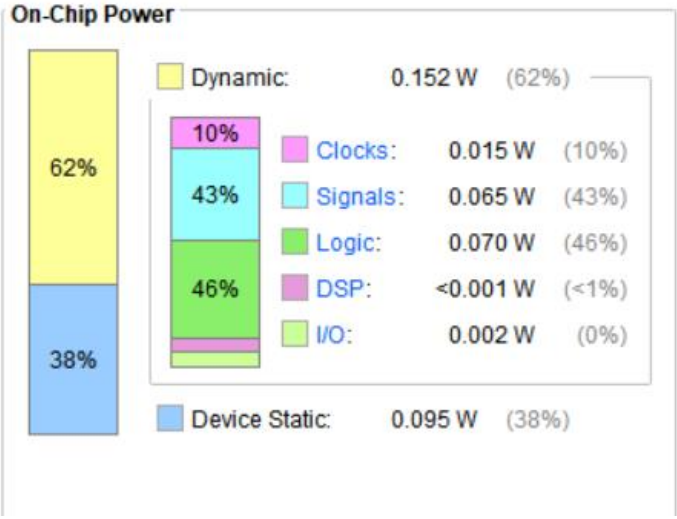
Settings Edit

Project name:	project_1
Project location:	D:/projects/MIPS/project_1
Product family:	Zynq-7000
Project part:	xc7z010clg400-1
Top module name:	MIPS
Target language:	Verilog
Simulator language:	Verilog

Power report

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.247 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**27.8°C**
Thermal Margin:	57.2°C (4.9 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium



Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Logic (W)	DSP (W)	I/O (W)
0.152 W (62% of total)	MIPS							
<0.001 W (<1% of total)	select_pc_INS (mux4in)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	load_mem_INS (load_from_mem)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	LO_REG (register_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	HI_REG (register)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	srcB_MUX (mux)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	sel_lo_INS (mux8in_1)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	sel_hi_INS (mux8in)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	select_result_INS (mux16in)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	mul_INS (mult)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.002 W (1% of total)	Leaf Cells (8)							
0.005 W (2% of total)	inst_mem_INST (inst_mem)	<0.001	0.002	0.002	<0.001	0.003	<0.001	<0.001
0.011 W (4% of total)	ALU_INS (alu)	<0.001	0.005	0.005	<0.001	0.006	<0.001	<0.001
0.012 W (5% of total)	PC_INST (pc_reg)	<0.001	0.007	0.007	<0.001	0.005	<0.001	<0.001
0.031 W (13% of total)	data_mem_INS (data_mem)	0.013	0.015	0.015	<0.001	0.003	<0.001	<0.001
0.038 W (16% of total)	div_INS (div)	<0.001	0.017	0.017	<0.001	0.021	<0.001	<0.001
0.051 W (21% of total)	reg_file_INST (reg_file)	0.002	0.019	0.019	<0.001	0.03	<0.001	<0.001

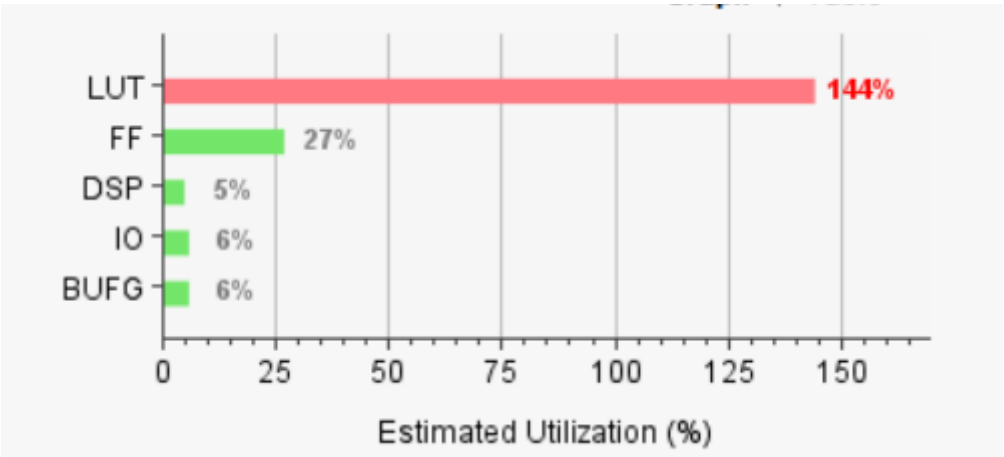
Utilization

Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	25415	17600	144.40
FF	9336	35200	26.52
DSP	4	80	5.00
IO	6	100	6.00
BUFG	2	32	6.25

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	DSPs (80)	Bonded IOB (100)	BUFGCTRL (32)
N MIPS	25415	9483	4825	2224	4	6	2
ALU_INS (alu)	4656	147	16	0	0	0	0
data_mem_INS (data_...	9074	8216	4352	2176	0	0	0
div_INS (div)	1450	0	0	0	0	0	0
HI_REG (register)	0	32	0	0	0	0	0
inst_mem_INST (inst_...	331	0	41	3	0	0	0
LO_REG (register_0)	0	32	0	0	0	0	0
load_mem_INS (load_...	1	0	0	0	0	0	0
mul_INS (mult)	47	0	0	0	4	0	0
PC_INST (pc_reg)	717	32	32	0	0	0	0
reg_file_INST (reg_file)	8994	1024	384	45	0	0	0
sel_hi_INS (mux8in)	53	0	0	0	0	0	0
sel_lo_INS (mux8in_1)	32	0	0	0	0	0	0
select_pc_INS (mux4in)	1	0	0	0	0	0	0
select_result_INS (mu...	53	0	0	0	0	0	0
srcB_MUX (mux)	14	0	0	0	0	0	0



Timing report

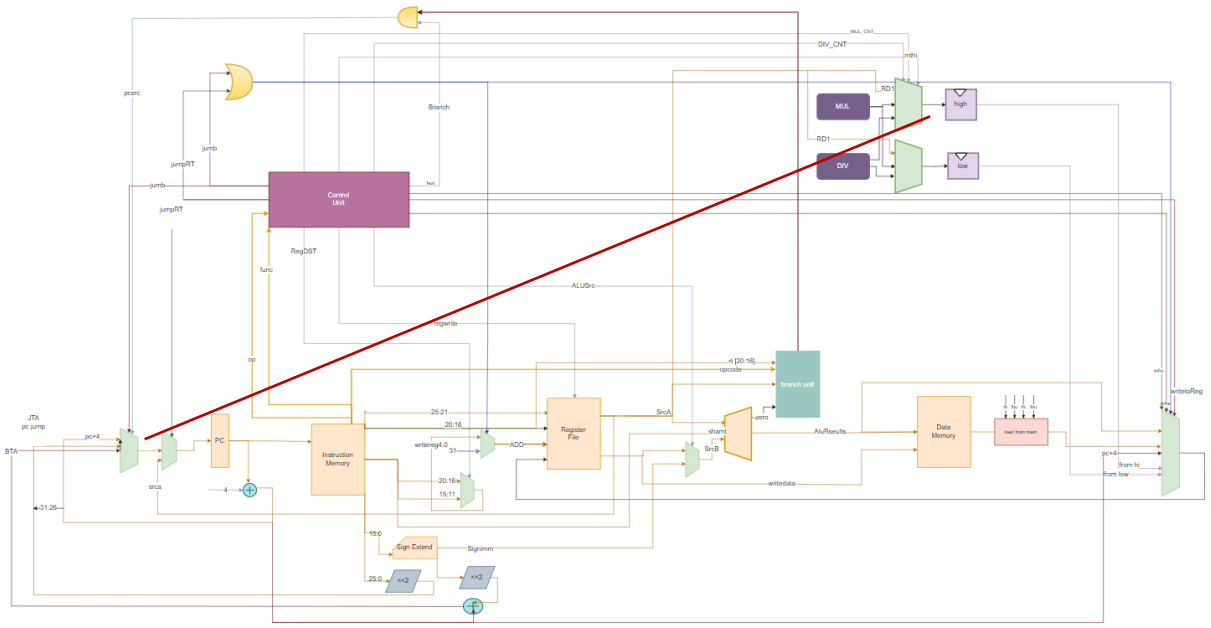
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	80.530- ns	Worst Hold Slack (WHS):	0.386 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	6041.786- ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	2224	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	18632	Total Number of Endpoints:	18632	Total Number of Endpoints:	9337

Timing constraints are not met.

WNS

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	-80.530	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[28]/D	90.379	61.502	28.877	10.0
Path 2	-80.530	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[29]/D	90.379	61.502	28.877	10.0
Path 3	-80.530	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[30]/D	90.379	61.502	28.877	10.0
Path 4	-80.530	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[31]/D	90.379	61.502	28.877	10.0
Path 5	-80.077	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[23]/D	89.926	61.502	28.424	10.0
Path 6	-80.054	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[25]/D	89.903	61.502	28.401	10.0
Path 7	-80.054	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[26]/D	89.903	61.502	28.401	10.0
Path 8	-80.036	308	309	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[27]/D	89.885	61.502	28.383	10.0
Path 9	-79.730	307	308	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[15]/D	89.579	61.388	28.191	10.0
Path 10	-79.611	306	307	257	PC_INST/pc_reg[5]/C	HI_REG/new_reg_reg[19]/D	89.460	61.128	28.332	10.0



And When working on 100ns instead of 10ns the timing report:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	9.677 ns	Worst Hold Slack (WHS):	0.397 ns	Worst Pulse Width Slack (WPWS):	49.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	18638	Total Number of Endpoints:	18638	Total Number of Endpoints:	9343