MIPS processor

RISC Processor (Reduced Instruction Set Computer)

NAME : George Jan George Shaffik

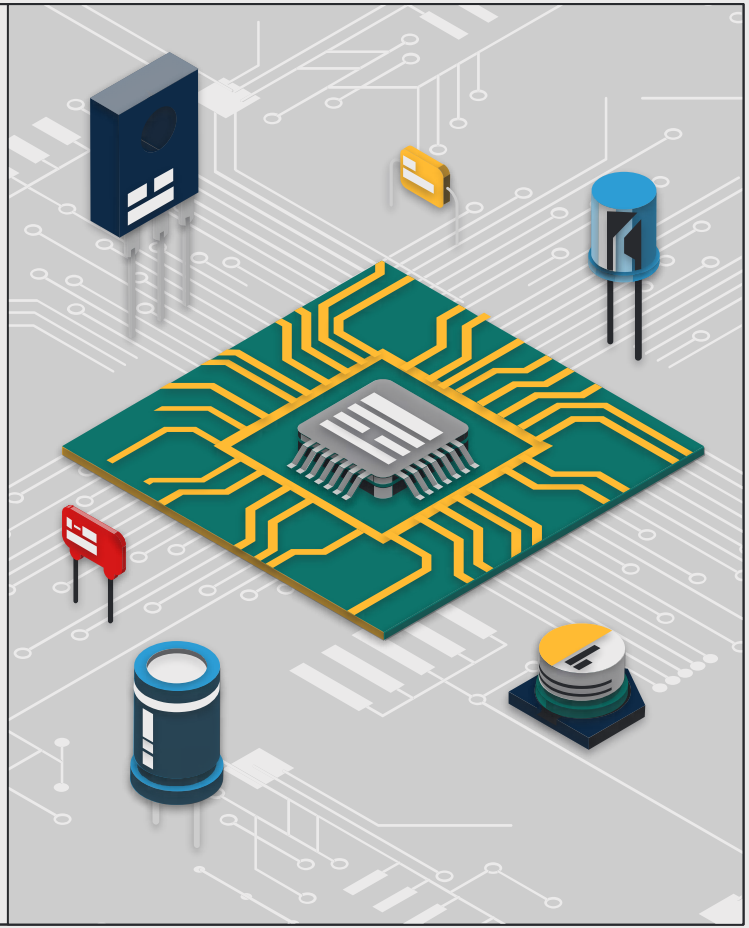
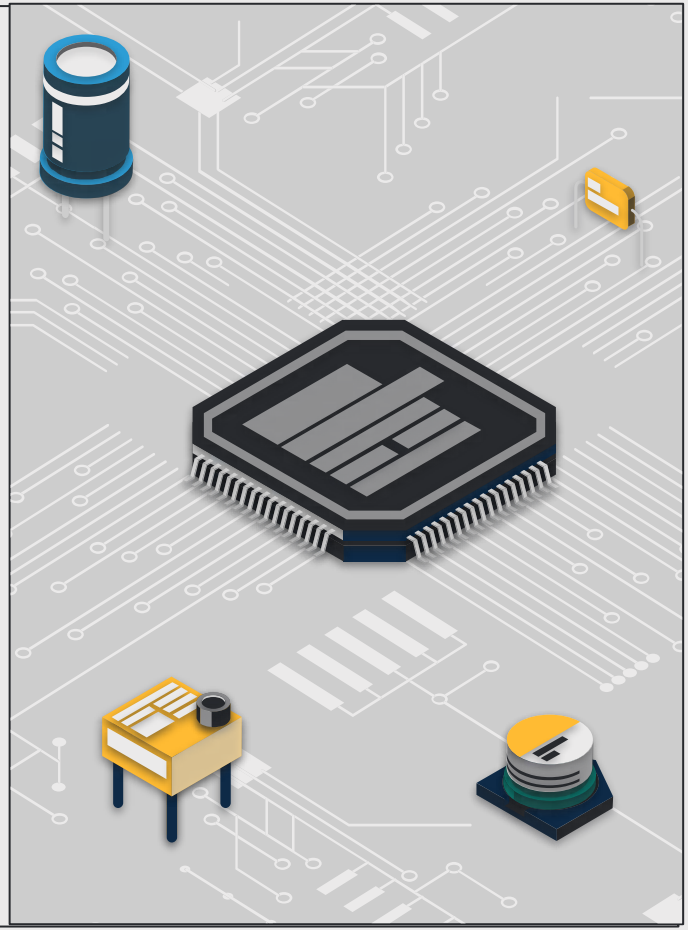
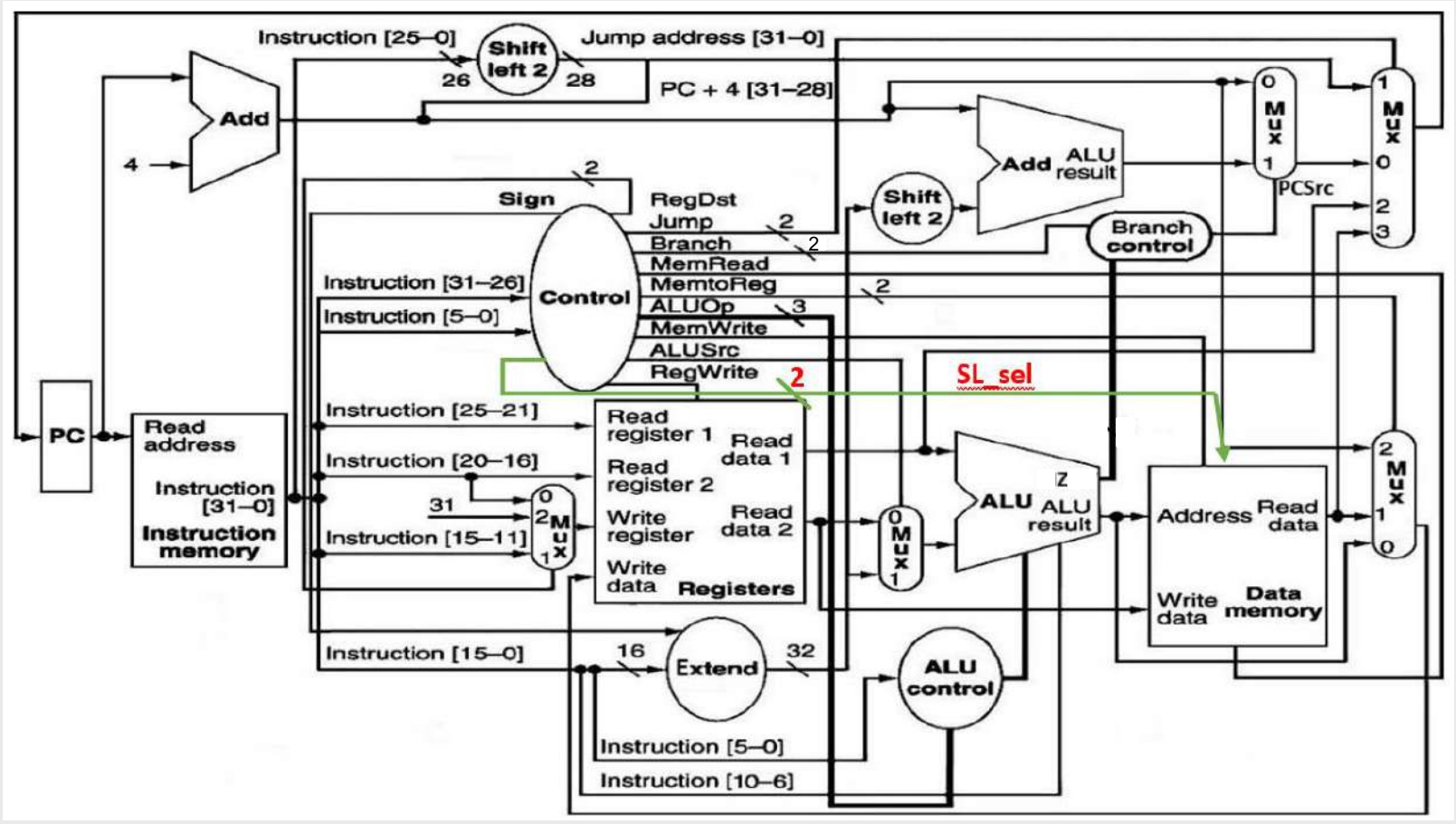


Table of contents

01

MIPS architecture

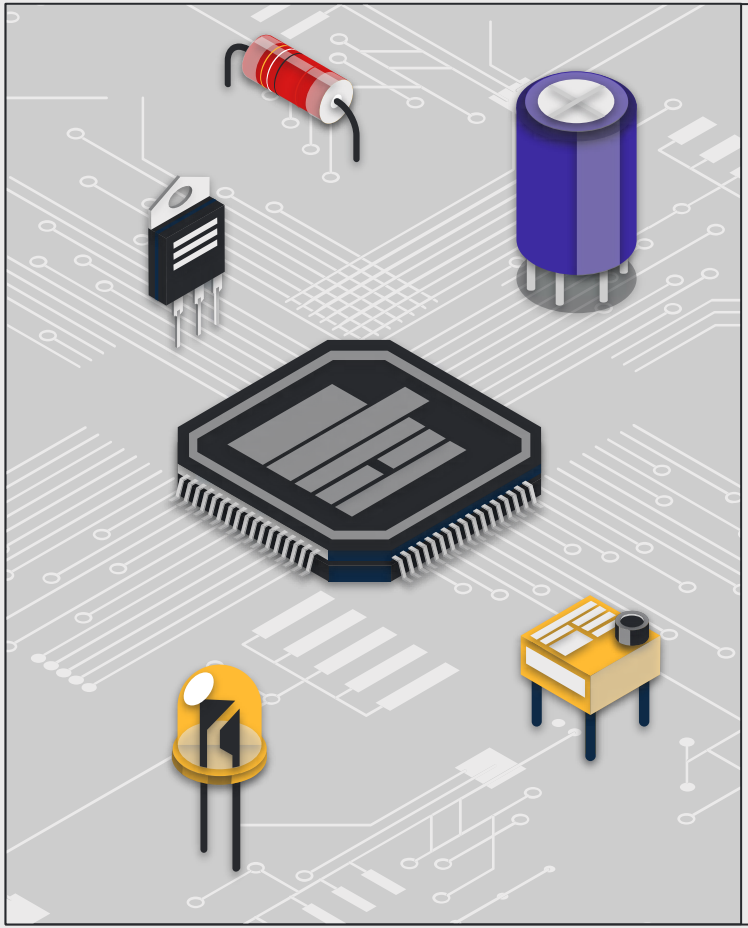



Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
Sign
2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Instruction [5–0]
Shift left 2
Add
ALU result
Branch control
PCSrc
M u x
0
1
2
3
SL_sel
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
31
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
ALU
Z
ALU result
Address
Read data
Write data
Data memory
Instruction [15–0]
16
Extend
32
ALU control
Instruction [5–0]
Instruction [10–6]

Instruction	Opcode (dec)	Funct (dec)	Instruction	Opcode (dec)	Funct (dec)	Instruction	Opcode (dec)	Funct (dec)
add	0	32	#lb	32		sll	0	0
addi	8		#lh	33		slt	0	42
and	0	36	lui	15		slti	10	
andi	12		lw	35		sra	0	3
beq	4		nor	0	39	srl	0	2
bne	5		or	0	37	sub	0	34
j	2		ori	13		sw	43	
jal	3		#sb	40		xor	0	38
jr	0	8	#sh	41		xori	14	
lwr	0	20	jm	54		jalm	52	
jalr	53		subi	51		lbit	49	
sbit	50							

Register name	Register number (dec)	Register name	Register number (dec)	Register name	Register number(dec)
$zero	0	$a0—$a3	4—7	$t8—$t9	24—25
$at	1	$t0—$t7	8—15	$sp	29
$v0—$v1	2—3	$s0—$s7	16—23	$ra	31

ALU control input signal (ALUOp) (bin)	000	001	010	011	100	101	110	111
Desired ALU action	add	sub	and	or	xor	slt	lui	Funct



Extra Instructions

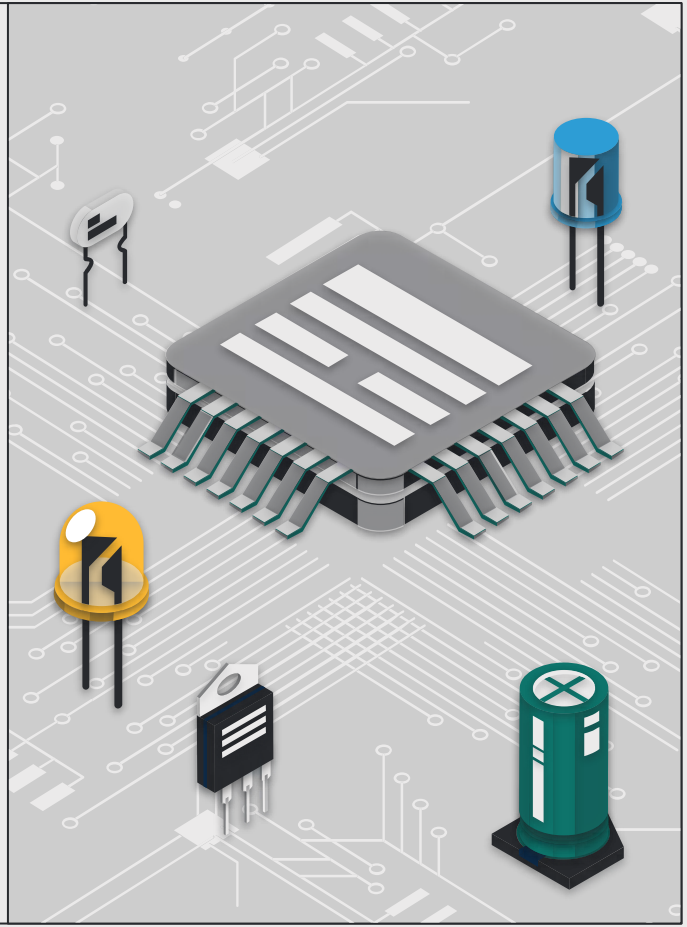
• Load word register: lwr $rd, $rs, $rt. The format of lwr is similar to that of lw, except that thetwo index registers $rs and $rt are summed to obtain the effective address of the word to be loaded, and the loaded word is saved in register $rd. lwr is an R-format instruction.
• Jump and link memory: jalm offset($rs). The format of jalm is similar to that of lw, except thatthe rt field is not used because the word loaded from memory is put in the PC instead of register $rt and jalm places the address of the following instruction in register $ra.
• Jump and link register: jalr $rs, $rd. The format of jalr is similar to that of jr, except that jalrplaces the address of the following instruction in register $rd (instruction bits 15—11).
• Jump memory: jm offset($rs). The format of jm is similar to that of lw, except that the rt field isnot used because the word loaded from memory is put in the PC instead of register $rt.
• Lbit &Sbit : load and Store the least significant bit in the byte

02

Verilog code

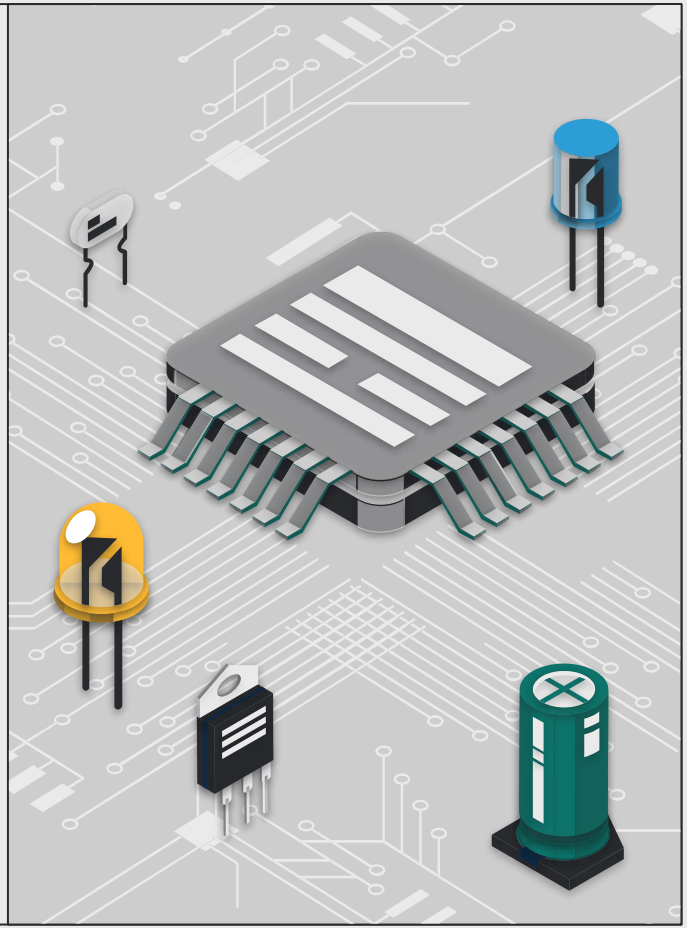
MIPS Processor with debouncer

```
module MIPS_processor_with_debouncer (
    input rst,clk_50M,i_button,
    output negative,Zero,Cout,overflow,Branch_port,
    output [4:0] ALUOUT_port,
    output [1:0] jump_port

);
wire clk;
wire [31:0]Instruction,Read_data1,Read_data2,PC_4,
OUT_shift_32,OUT_adder,PCSrc_OUT,Read_data;
wire [27:0]OUT_shift28;
wire[1:0]RegDst_2,MemtoReg_2,Branch_2,Jump_2,SL_sel;
wire ALUSrc,RegWrite,MemRead,MemWrite,Sign,PCSrc;
wire [2:0] ALUOP;
wire [3:0] ALU_sel;
wire [31:0] PC_IN,Readaddress_PC;
wire [4:0] Write_register;
wire [31:0] OUT_extend,ALU_B,ALU_result,
JumpAddress,Write_data_reg;
```



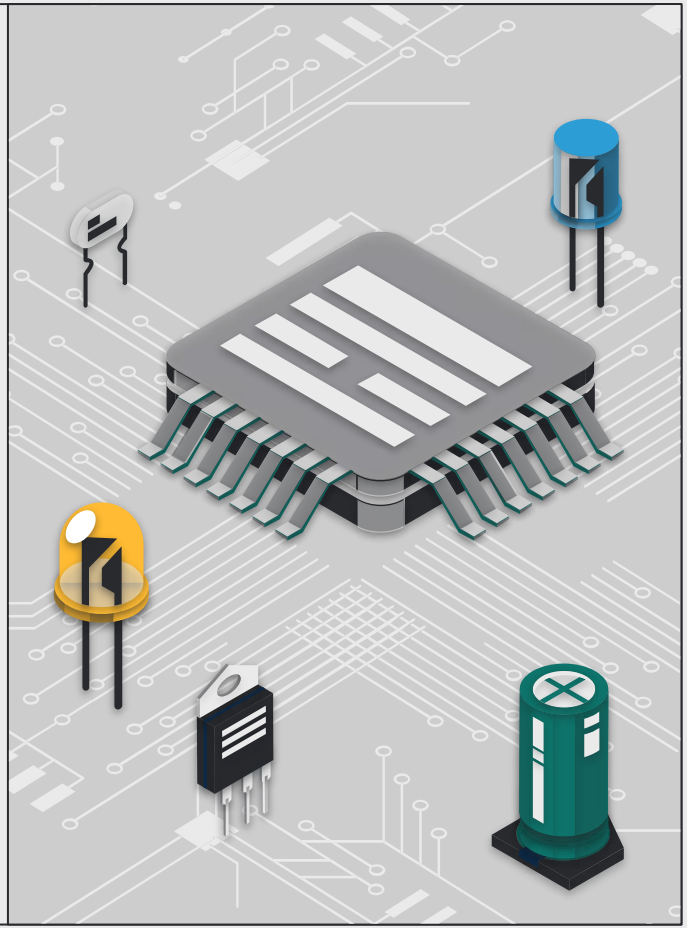
MIPS Processor with debouncer

```
////////////////////////////////
PC PC(
    .clk(clk),
    .IN(PC_IN),
    .OUT(Readaddress_PC)
);
////////////////////////////////
Instruction_memory Instruction_memory(
    .ReadAddress(Readaddress_PC[31:2]),
    .Instruction(Instruction),
    .WriteAddress(),
    .writeINS(1'b0),
    .clk(clk),
    .writeDataINS()
);
```



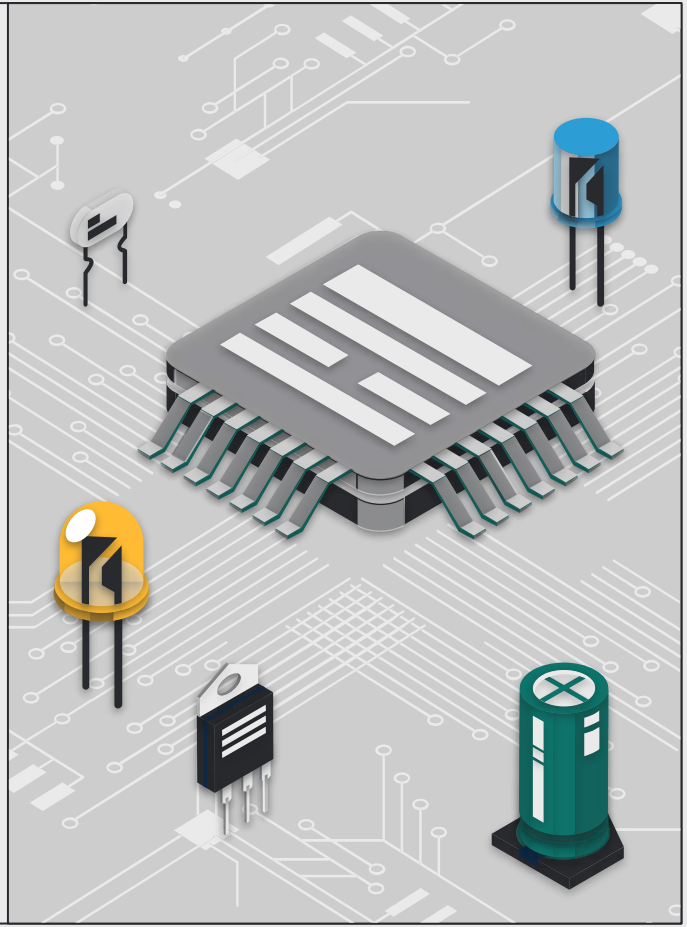
MIPS Processor with debouncer

```
////////////////////////////////
Control Control_unit (
    .OP_code(Instruction[31:26]),
    .Function_field(Instruction[5:0]),
    .RegDst_2(RegDst_2),
    .MemtoReg_2(MemtoReg_2),
    .Branch_2(Branch_2),
    .Jump_2(Jump_2),
    .ALUSrc(ALUSrc),
    .RegWrite(RegWrite),
    .MemRead(MemRead),
    .MemWrite(MemWrite),
    .Sign(Sign),
    .ALUOP(ALUOP),
    .SL_sel(SL_sel)
);
```



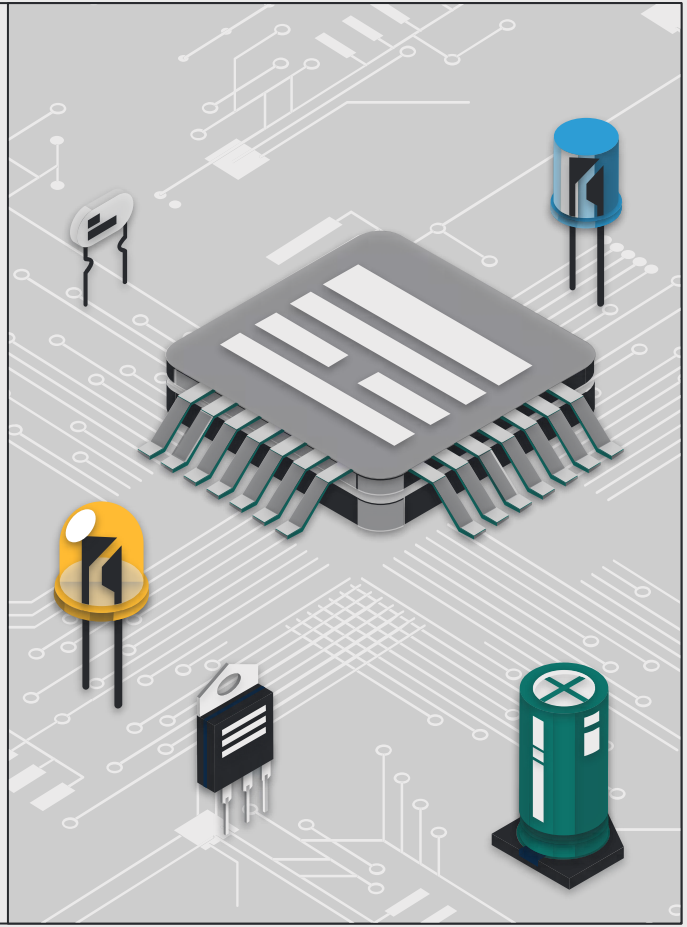
MIPS Processor with debouncer

```
//////////////////////////////
MUX_4_1 #(.N(5)) MUX0(
    .A(Instruction[20:16]),
    .B(Instruction[15:11]),
    .C(5'd31),
    .D(5'dx),
    .sel(RegDst_2),
    .Z(Write_register)
);
//////////////////////////////
Registers Registers(
    .Read_register1(Instruction[25:21]),
    .Read_register2(Instruction[20:16]),
    .Write_register(Write_register),
    .Read_data1(Read_data1),
    .Read_data2(Read_data2),
    .Write_data(Write_data_reg),
    .clk(clk),
    .RegWrite(RegWrite)
    );
```



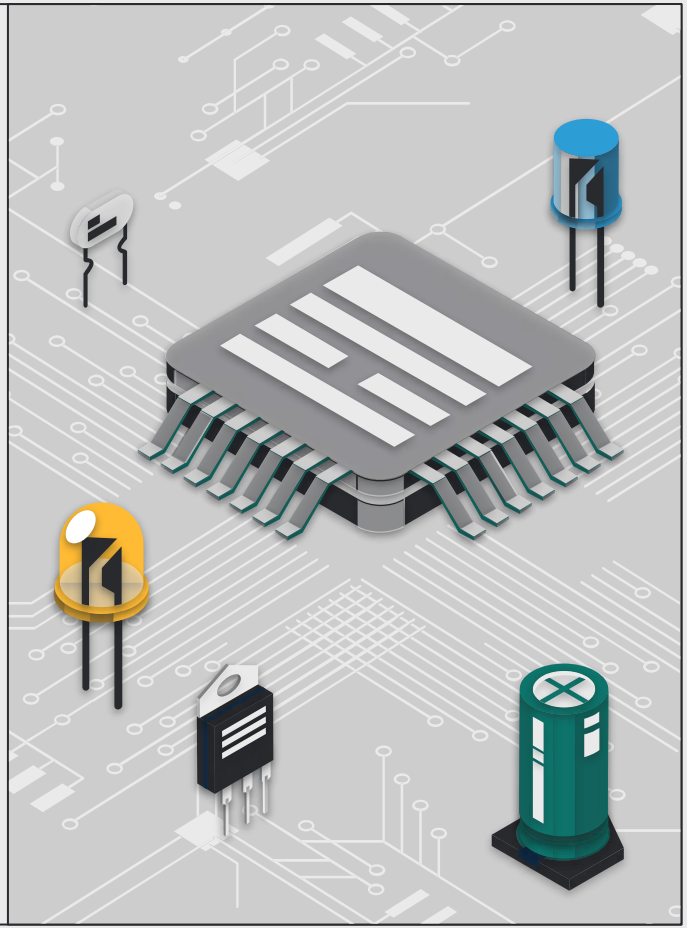
MIPS Processor with debouncer

```
////////////////////////////////
extend_unit extend_unit(
    .IN(Instruction[15:0]),
    .Sign(Sign),
    .OUT(OUT_extend)
);
////////////////////////////////
assign ALU_B=ALUSrc?OUT_extend:Read_data2;
////////////////////////////////
ALU_control ALU_control(
    .ALUOP(ALUOP),
    .Funct(Instruction[5:0]),
    .ALU_sel(ALU_sel)
);
```



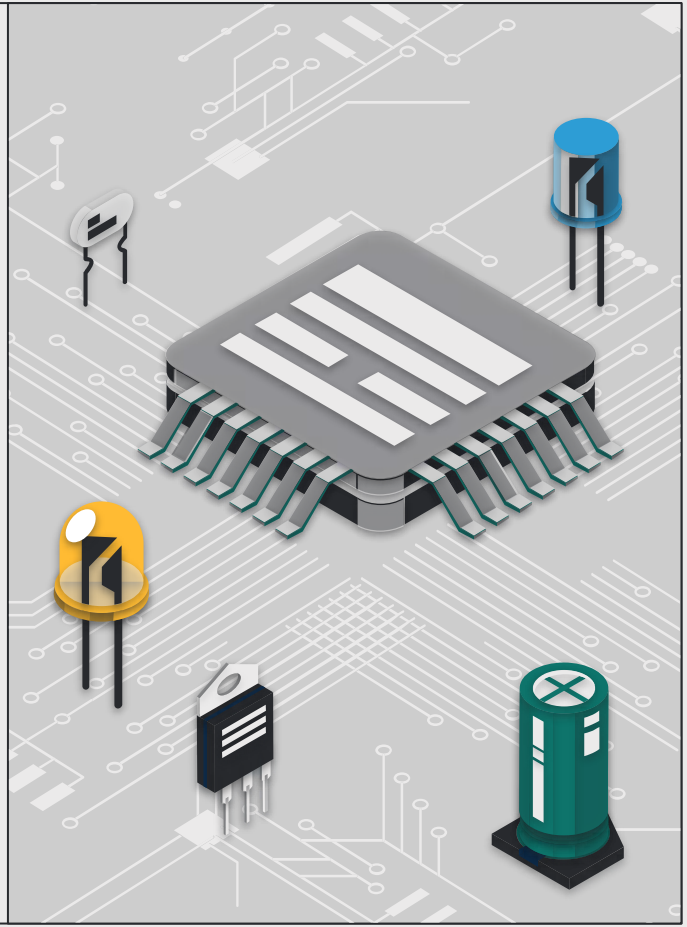
MIPS Processor with debouncer

```
////////////////////////////////
ALU #(.n(32)) ALU (
    .sel(ALU_sel),
    .A(Read_data1),
    .B(ALU_B),
    .Shamt(Instruction[10:6]),
    .OUT(ALU_result),
    .negative(negative),
    .Zero(Zero),
    .Cout(Cout),
    .overflow(overflow)
);
////////////////////////////////
Data_memory Data_memory(
    .clk(clk),
    .MemWrite(MemWrite),
    .Address(ALU_result[31:2]),
    .Write_data(Read_data2),
    .Read_data(Read_data),
    .sel(SL_sel),
    .byte_addr(ALU_result[1:0])
);
```



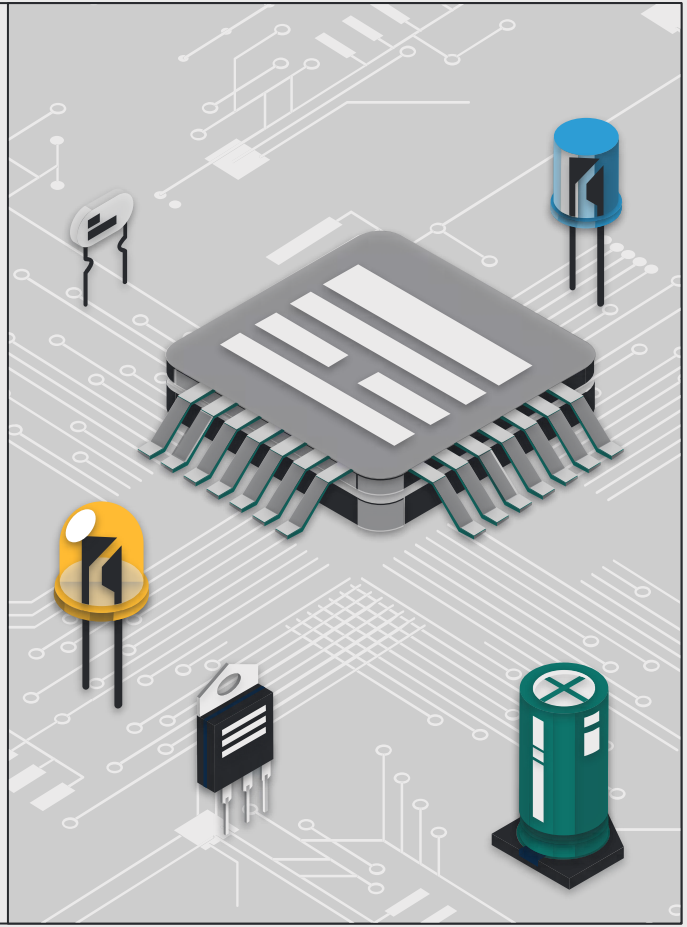
MIPS Processor with debouncer

```
////////////////////////////////
add_4 #(.n(32)) add_4(
    .PC(Readaddress_PC),
    .PC_4(PC_4)
);
////////////////////////////////
shift_left_2_32_32 shift_32(
    .IN(OUT_extend),
    .OUT(OUT_shift_32)
);
////////////////////////////////
adder #(.n(32)) adder (
    .A(PC_4),
    .B(OUT_shift_32),
    .OUT(OUT_adder)
);
////////////////////////////////
shift_left_2_26_28 shift_26(
    .IN(Instruction[25:0]),
    .OUT(OUT_shift28)
);
```



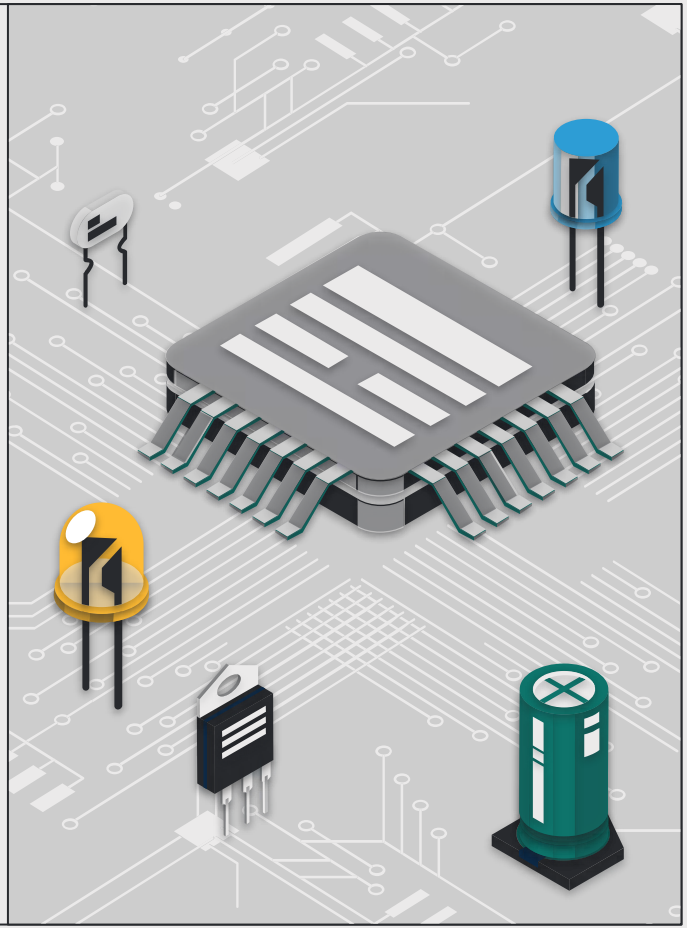
MIPS Processor with debouncer

```
////////////////////////////////
concatenation_unit concatenation_unit(
    .IN(OUT_shift28),
    .PC_in(PC_4[31:28]),
    .JumpAddress(JumpAddress)
);
////////////////////////////////
Branch_control Branch_control(
    .Branch(Branch_2),//bne beq
    .Zero(Zero),
    .OUT(PCSrc)
);
////////////////////////////////
assign PCSrc_OUT=PCSrc?OUT_adder:PC_4;
////////////////////////////////
MUX_4_1 #(.N(32)) MUX1(
    .A(PCSrc_OUT),
    .B(JumpAddress),
    .C(Read_data1),
    .D(Read_data),
    .sel(Jump_2),
    .Z(PC_IN)
```



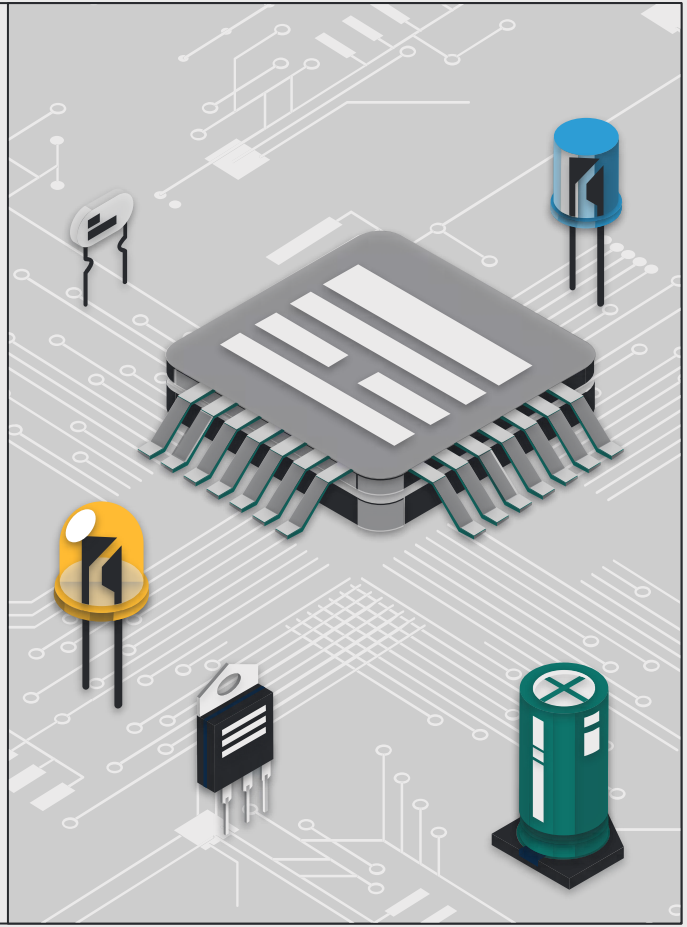
MIPS Processor with debouncer

```
);
//////////////////////////////
MUX_4_1 #(.N(32)) MUX2(
    .A(ALU_result),
    .B(Read_data),
    .C(PC_4),
    .D(32'bx),
    .sel(MemtoReg_2),
    .Z(Write_data_reg)
);
//////////////////////////////
assign ALUOUT_port=ALU_result[4:0];
assign Branch_port=PCSrc;
assign jump_port=Jump_2;
//////////////////////////////
debouncer de_inst (
    .clk(clk_50M),
    .rst(rst),
    .i_button(i_button),
    .o_button(clk)
);
endmodule
```



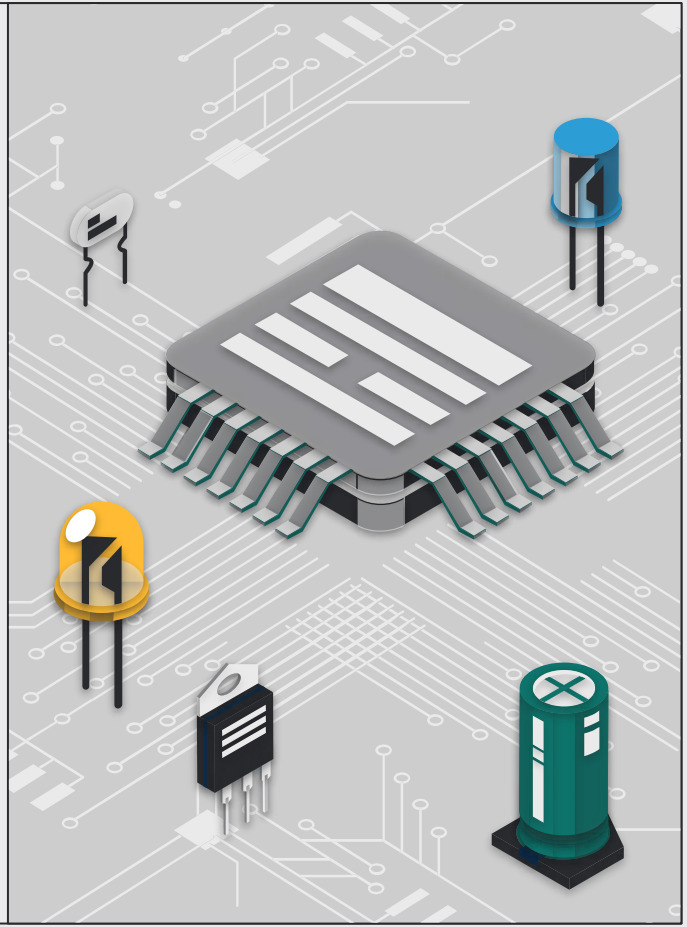
Debouncer

```
module debouncer (
    input wire clk,
    input wire rst,
    input wire i_button,
    output reg o_button
);
    reg [18:0]counter ;
    reg button_state;
    reg button_sync1,button_sync2;
    always @ (posedge clk or posedge rst)
    begin
        if (rst)
            begin
                button_sync1<=1'b0;
                button_sync2<=1'b0;
            end
        else
        begin
            button_sync1<=i_button;
            button_sync2<=button_sync1;
        end
    end
```



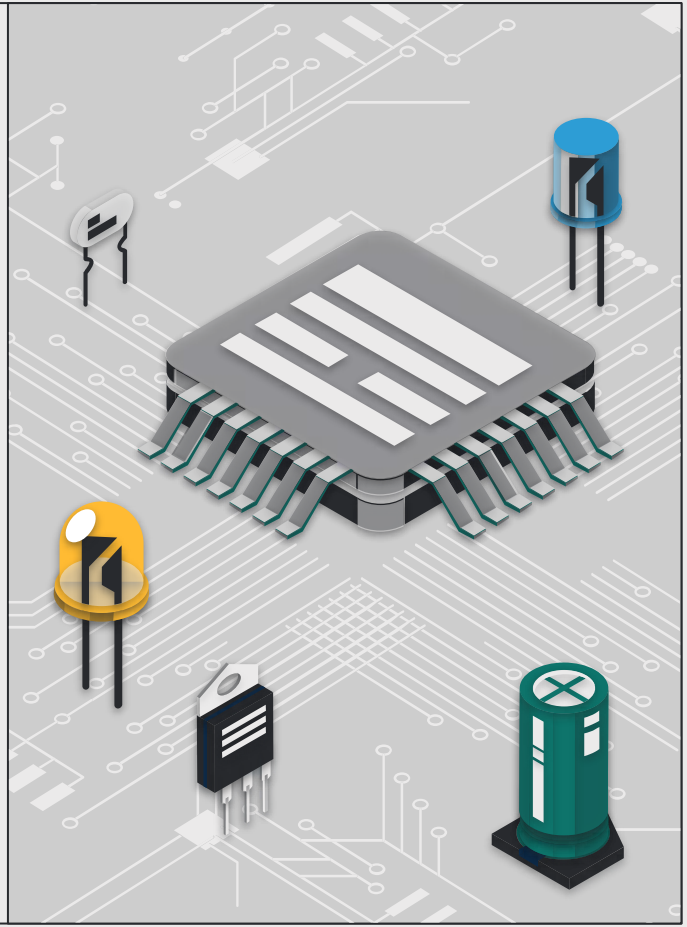
Debouncer

```
always @ (posedge clk or posedge rst)
begin
    if (rst)
    begin
    counter <=19'b0;
    button_state<=1'b0;
    o_button<=1'b0;
    end
```



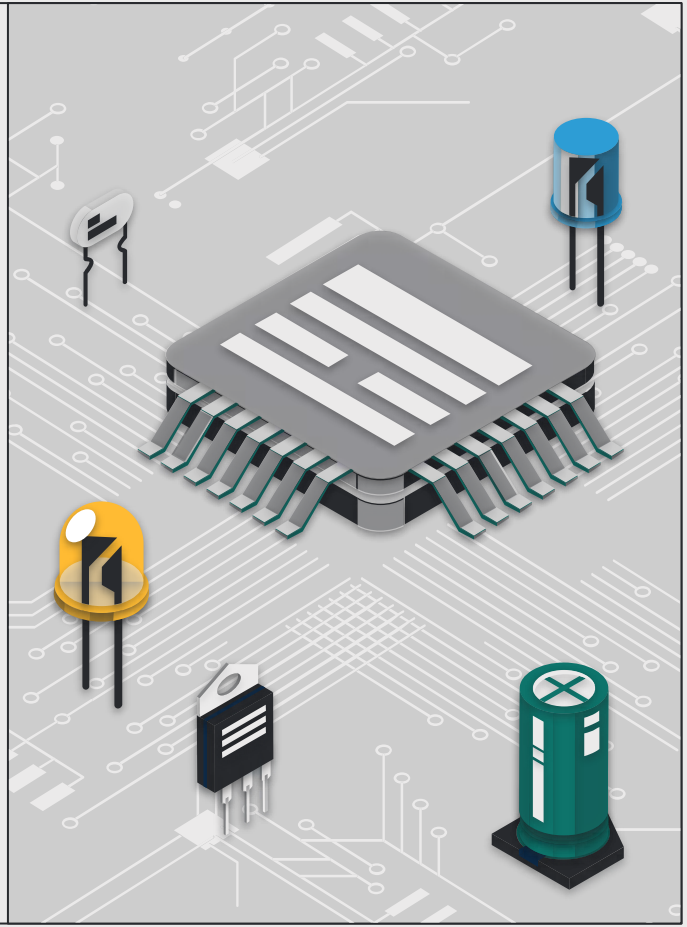
Debouncer

```
        else
        begin
            if (button_sync2 != button_state)
            begin
            counter<=counter+1'b1;
                if (counter==500_000)
                begin
                button_state<=button_sync2;
                o_button<=button_sync2;
                counter<=19'b0;
                end
                end
                else
                begin
                counter <= 19'b0;
        end
        end
    end
endmodule
```



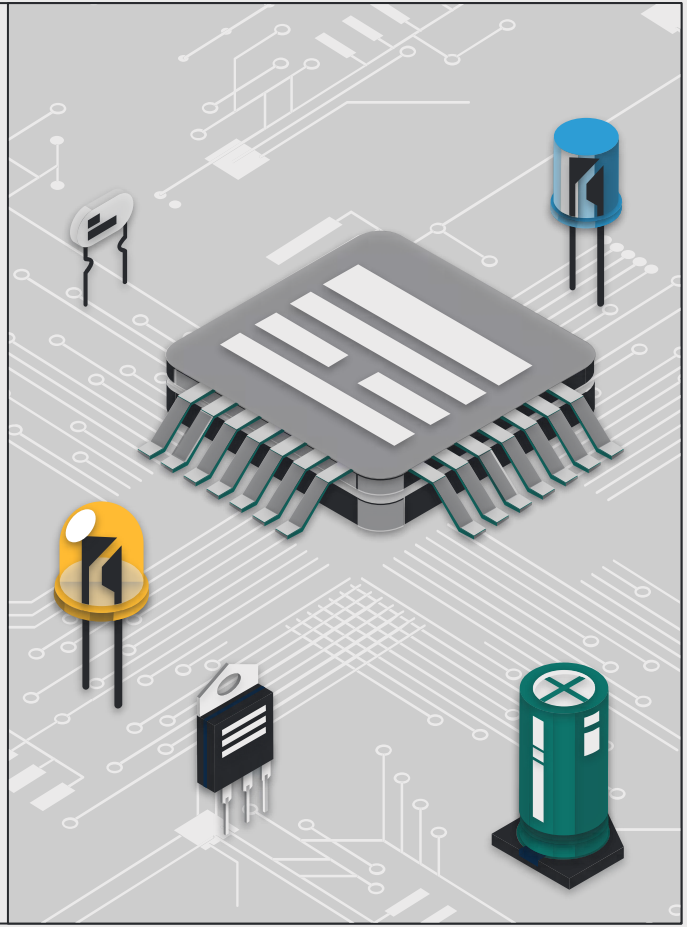
MIPS processor

```
module MIPS_processor (
    input clk,
    output negative,Zero,Cout,overflow

);
wire [31:0]Instruction,Read_data1,Read_data2,PC_4,
OUT_shift_32,OUT_adder,PCSrc_OUT,Read_data;
wire [27:0]OUT_shift28;
wire [1:0]
RegDst_2,MemtoReg_2,Branch_2,Jump_2,SL_sel;
wire ALUSrc,RegWrite,MemRead,MemWrite,Sign,PCSrc;
wire [2:0] ALUOP;
wire [3:0] ALU_sel;
wire [31:0] PC_IN,Readaddress_PC;
wire [4:0] Write_register;
wire [31:0] OUT_extend,ALU_B,ALU_result,
JumpAddress,Write_data_reg;
```



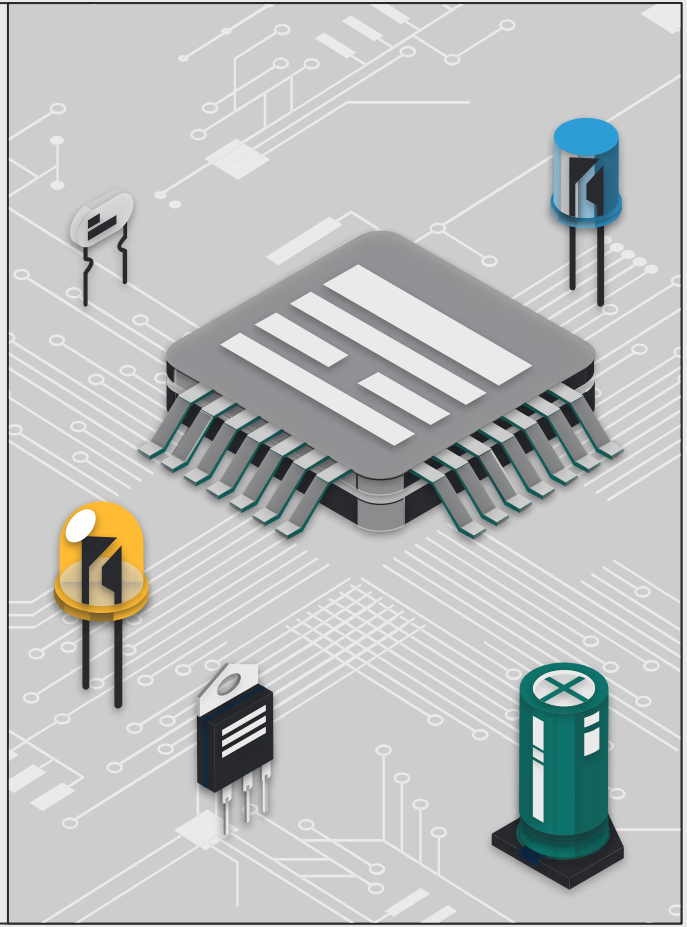
MIPS processor

```
////////////////////////////////
PC PC(
    .clk(clk),
    .IN(PC_IN),
    .OUT(Readaddress_PC)
);
////////////////////////////////
Instruction_memory Instruction_memory(
    .ReadAddress(Readaddress_PC[31:2]),
    .Instruction(Instruction),
    .WriteAddress(),
    .writeINS(1'b0),
    .clk(clk),
    .writeDataINS()
);
```



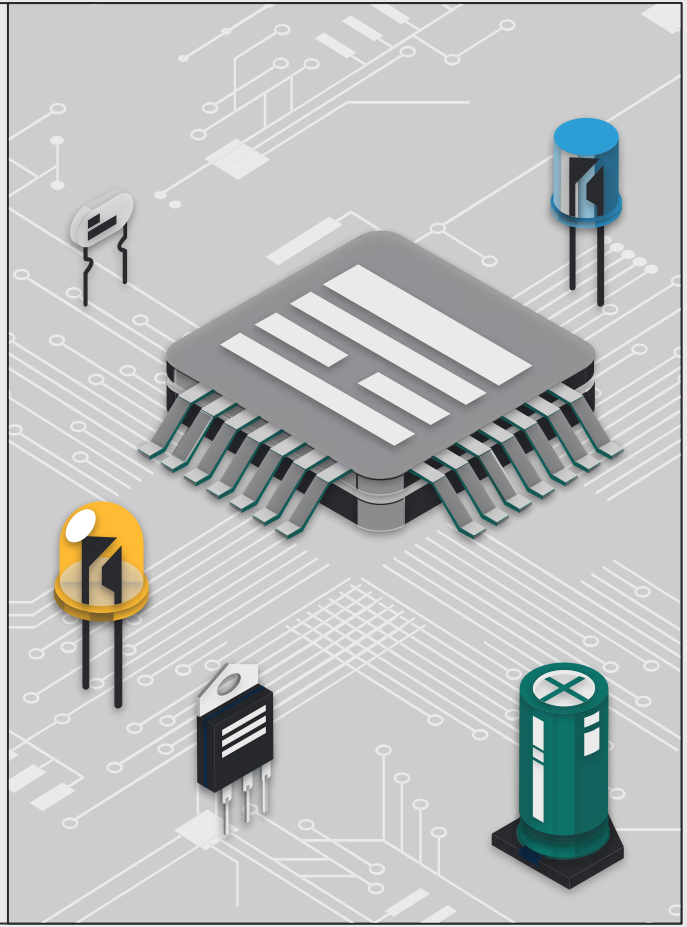
MIPS processor

```
////////////////////////////////
Control Control_unit (
    .OP_code(Instruction[31:26]),
    .Function_field(Instruction[5:0]),
    .RegDst_2(RegDst_2),
    .MemtoReg_2(MemtoReg_2),
    .Branch_2(Branch_2),
    .Jump_2(Jump_2),
    .ALUSrc(ALUSrc),
    .RegWrite(RegWrite),
    .MemRead(MemRead),
    .MemWrite(MemWrite),
    .Sign(Sign),
    .ALUOP(ALUOP),
    .SL_sel(SL_sel)
);
```



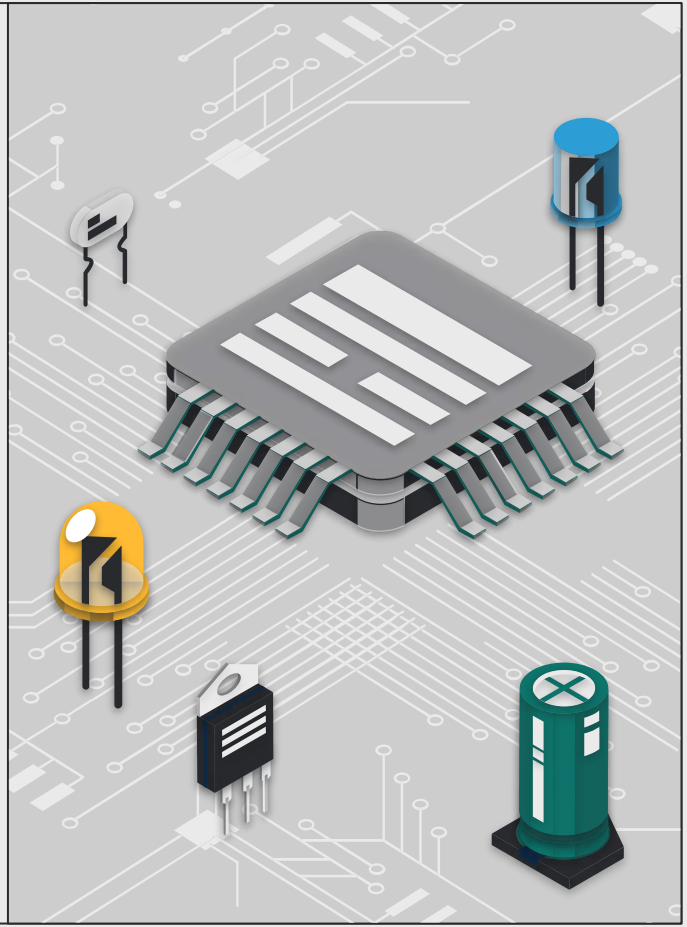
MIPS processor

```
////////////////////////////////
MUX_4_1 #(.N(5)) MUX0(
    .A(Instruction[20:16]),
    .B(Instruction[15:11]),
    .C(5'd31),
    .D(5'dx),
    .sel(RegDst_2),
    .Z(Write_register)
);
////////////////////////////////
Registers Registers(
    .Read_register1(Instruction[25:21]),
    .Read_register2(Instruction[20:16]),
    .Write_register(Write_register),
    .Read_data1(Read_data1),
    .Read_data2(Read_data2),
    .Write_data(Write_data_reg),
    .clk(clk),
    .RegWrite(RegWrite)
    );
```



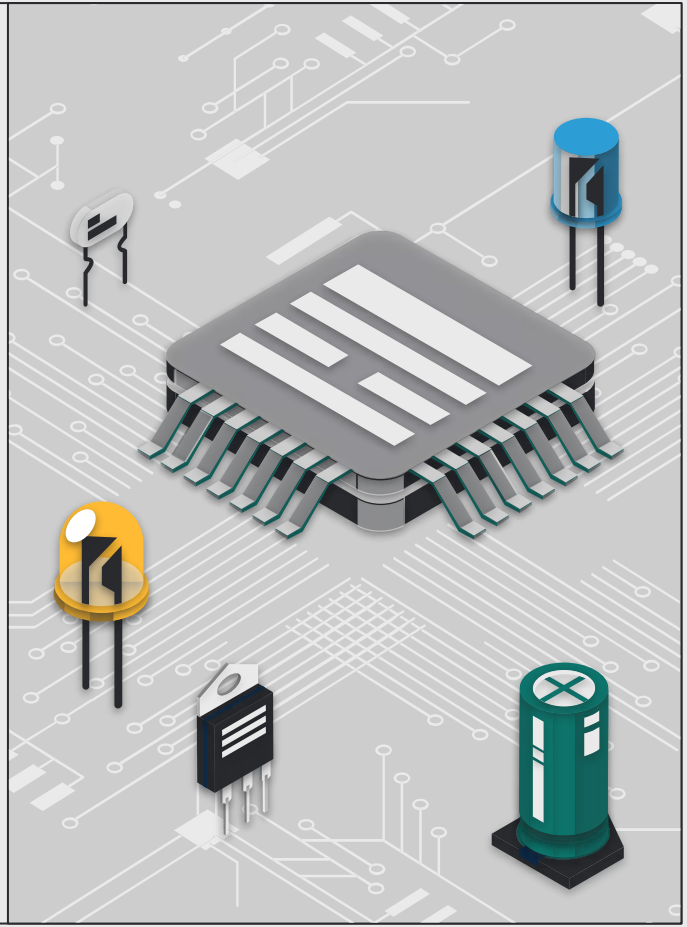
MIPS processor

```
//////////////////////////////
extend_unit extend_unit(
    .IN(Instruction[15:0]),
    .Sign(Sign),
    .OUT(OUT_extend)
);
//////////////////////////////
assign ALU_B=ALUSrc?OUT_extend:Read_data2;
//////////////////////////////
ALU_control ALU_control(
    .ALUOP(ALUOP),
    .Funct(Instruction[5:0]),
    .ALU_sel(ALU_sel)
);
```



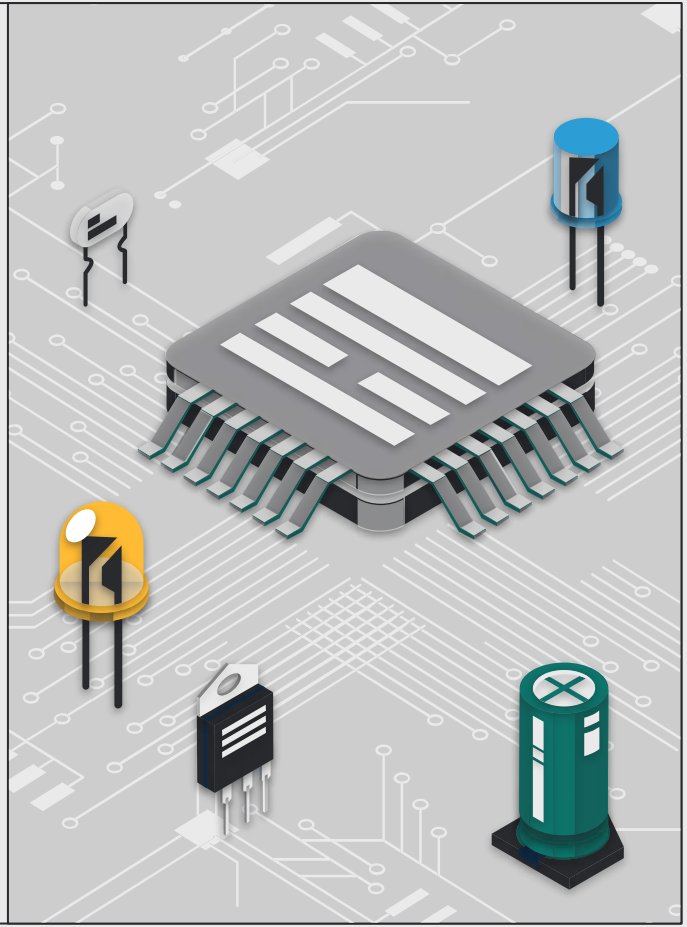
MIPS processor

```
////////////////////////////////
ALU #(.n(32)) ALU (
    .sel(ALU_sel),
    .A(Read_data1),
    .B(ALU_B),
    .Shamt(Instruction[10:6]),
    .OUT(ALU_result),
    .negative(negative),
    .Zero(Zero),
    .Cout(Cout),
    .overflow(overflow)
);
////////////////////////////////
Data_memory Data_memory(
    .clk(clk),
    .MemWrite(MemWrite),
    .Address(ALU_result[31:2]),
    .Write_data(Read_data2),
    .Read_data(Read_data),
    .sel(SL_sel),
    .byte_addr(ALU_result[1:0])
);
```



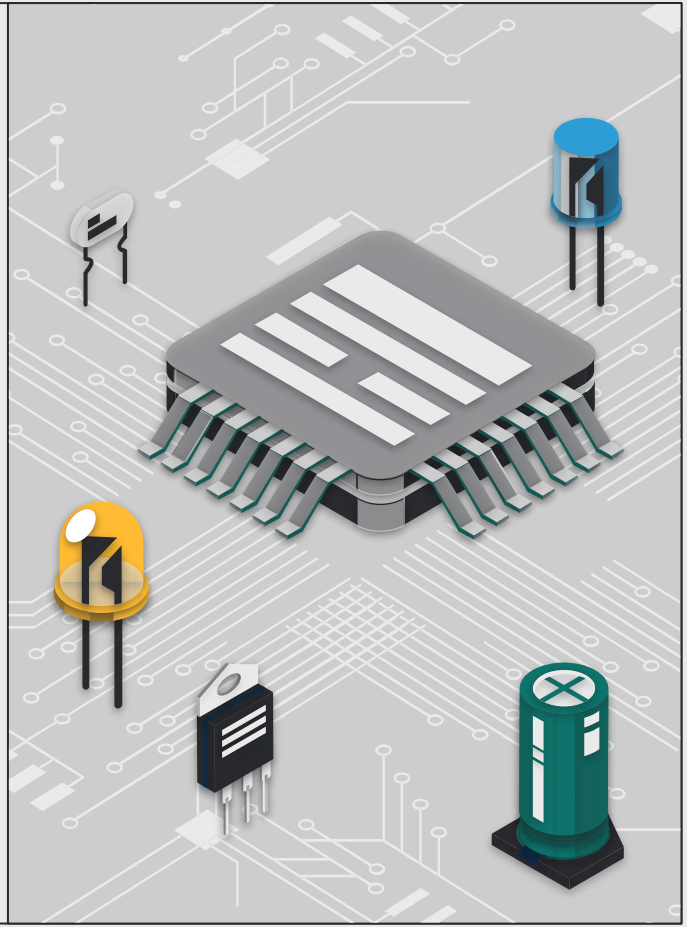
MIPS processor

```
////////////////////////////////
add_4 #(.n(32)) add_4(
    .PC(Readaddress_PC),
    .PC_4(PC_4)
);
////////////////////////////////
shift_left_2_32_32 shift_32(
    .IN(OUT_extend),
    .OUT(OUT_shift_32)
);
////////////////////////////////
adder #(.n(32)) adder (
    .A(PC_4),
    .B(OUT_shift_32),
    .OUT(OUT_adder)
);
////////////////////////////////
shift_left_2_26_28 shift_26(
    .IN(Instruction[25:0]),
    .OUT(OUT_shift28)
);
```



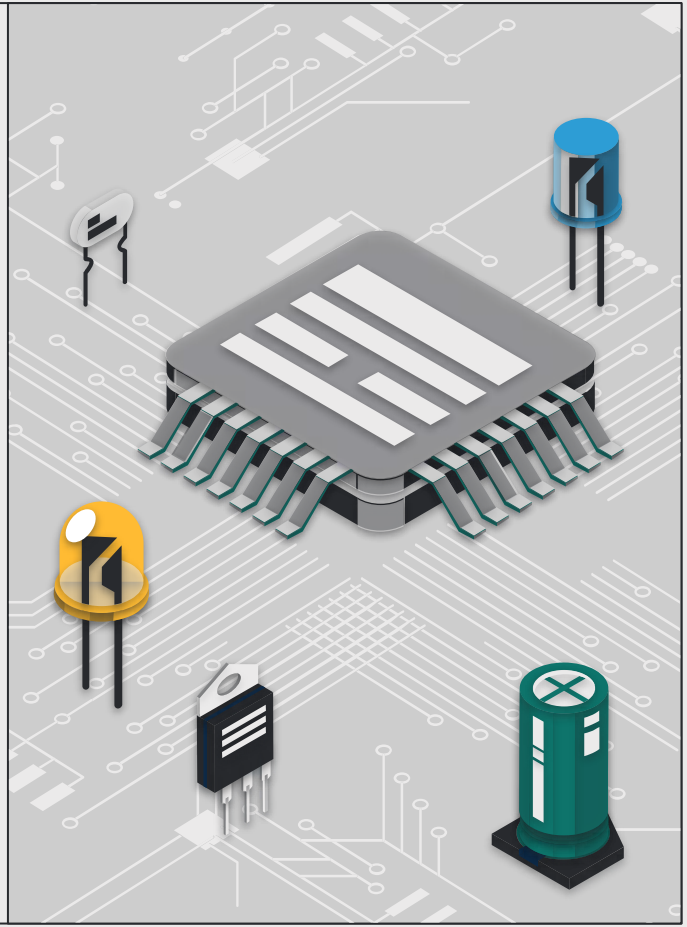
MIPS processor

```
////////////////////////////////
concatenation_unit concatenation_unit(
    .IN(OUT_shift28),
    .PC_in(PC_4[31:28]),
    .JumpAddress(JumpAddress)
);
////////////////////////////////
Branch_control Branch_control(
    .Branch(Branch_2),//bne beq
    .Zero(Zero),
    .OUT(PCSrc)
);
////////////////////////////////
assign PCSrc_OUT=PCSrc?OUT_adder:PC_4;
```



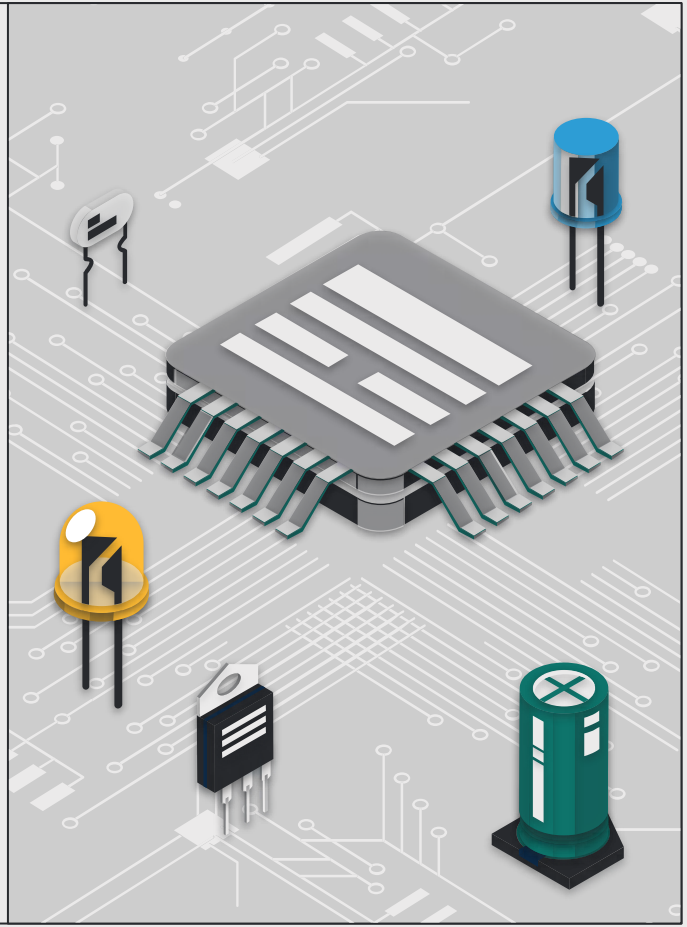
MIPS processor

```
////////////////////////////////
MUX_4_1 #(.N(32)) MUX1(
    .A(PCSrc_OUT),
    .B(JumpAddress),
    .C(Read_data1),
    .D(Read_data),
    .sel(Jump_2),
    .Z(PC_IN)
);
////////////////////////////////
MUX_4_1 #(.N(32)) MUX2(
    .A(ALU_result),
    .B(Read_data),
    .C(PC_4),
    .D(32'bx),
    .sel(MemtoReg_2),
    .Z(Write_data_reg)
);
////////////////////////////////
endmodule
```



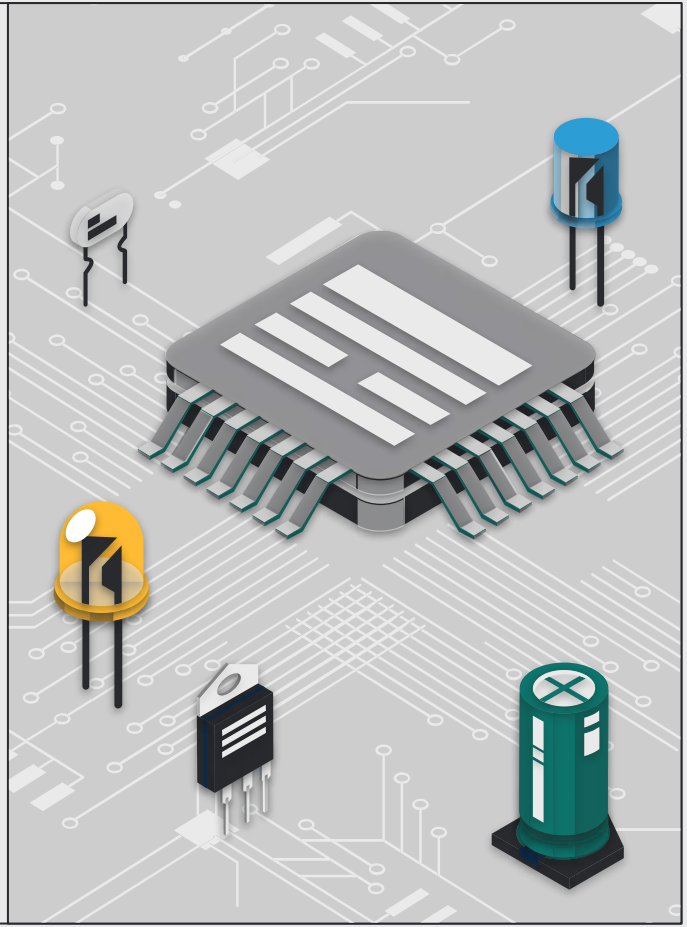
ALU

```
module ALU #(parameter n=32)(
    input [3:0] sel,
    input [n-1:0] A,B,
    input [10:6] Shamt,
    output [n-1:0] OUT,
    output negative,Zero,Cout,overflow
);
wire [n-1:0]
Arithmetic_OUT,Logic_OUT,Shift_lui_OUT,Logic_shift
;
//////////////////////////////////////
Arithmetic_unit #(.n(n))Arithmetic_unit(
    .A(A),
    .B(B),
    .add_n(sel[3]),
    .add_slt(sel[2]),
    .Cout(Cout),
    .overflow(overflow),
    .OUT(Arithmetic_OUT)
);
```



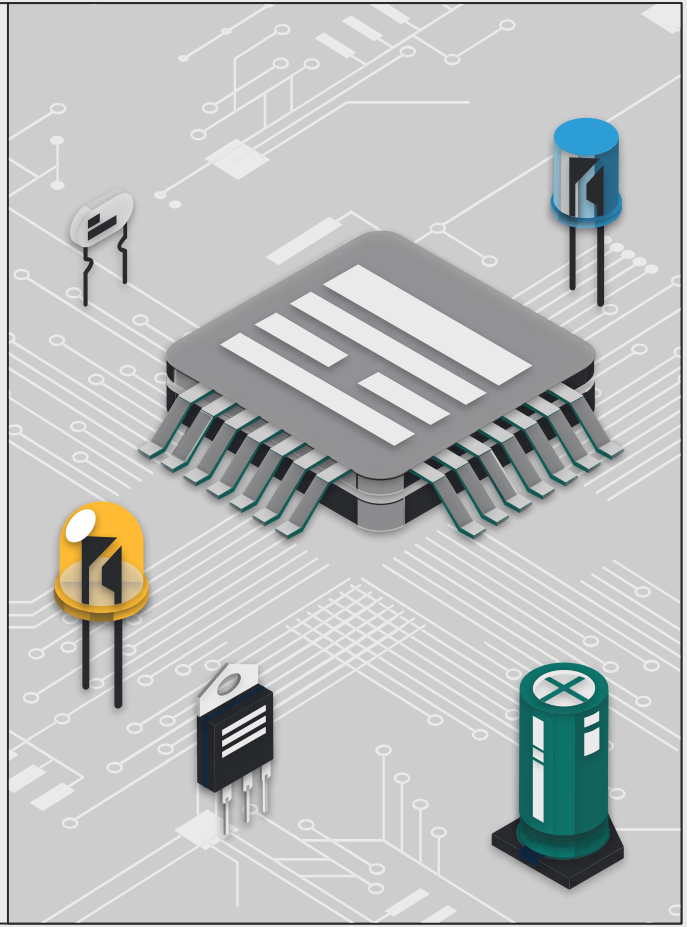
ALU

```
////////////////////////////////////
Logic_unit #(.n(n)) Logic_unit (
    .A(A),
    .B(B),
    .sel(sel[3:2]),
    .OUT(Logic_OUT)
);
////////////////////////////////////
Shift_lui_unit #(.n(n)) Shift_lui_unit (
    .A(A),
    .B(B),
    .Shamt(Shamt),
    .sel(sel[3:2]),
    .OUT(Shift_lui_OUT)
);
    assign Logic_shift = sel
[1]?Shift_lui_OUT:Logic_OUT;
    assign OUT =
sel[0]?Logic_shift:Arithmetic_OUT;
    assign negative=OUT[n-1];
    assign Zero=~(|OUT);
endmodule
```



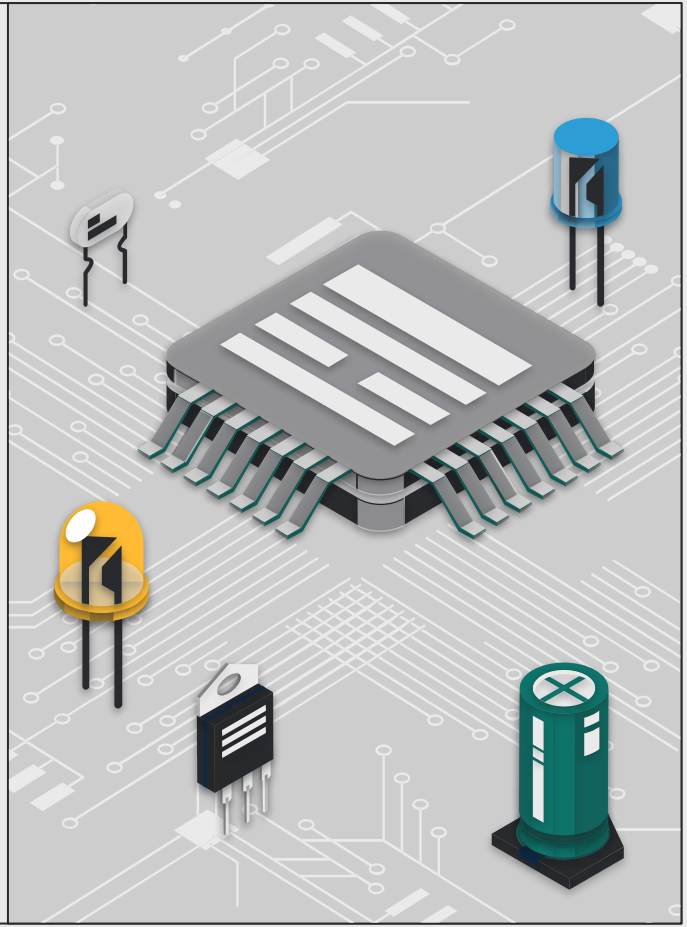
Arithmetic unit

```
module Arithmetic_unit #(parameter n = 32)(
    input [n-1:0] A,B,
    input add_n,add_slt,
    output Cout,overflow,
    output [n-1:0] OUT
);
    wire [n-1:0] B_xored,S;
    wire LessThan;
    generate
    genvar k;
        for(k=0;k<n;k=k+1)
        begin:Xored
            assign B_xored[k]=B[k]^add_n;
        end
    endgenerate
```



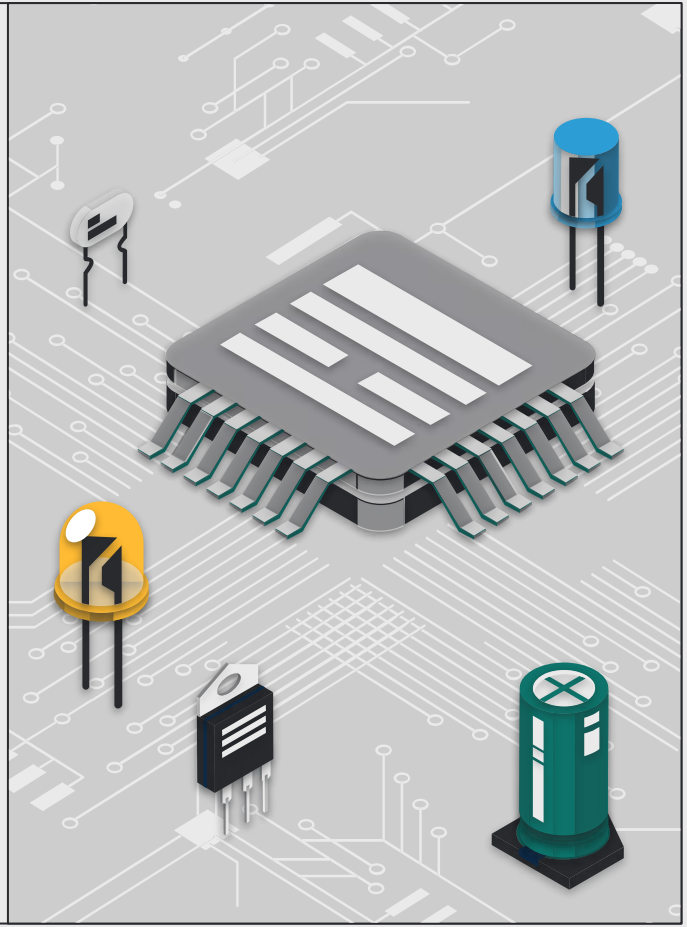
Arithmetic unit

```
    RCA #(.n(n))RCA(
    .A(A),
    .B(B_xored),
    .Cin(add_n),
    .Cout(Cout),
    .overflow(overflow),
    .LessThan(LessThan),
    .S(S)
    );
////////////////////////////////////////////
wire [n-1:0] slt = {{n-1{1'b0}},LessThan};
assign OUT=add_slt?slt:S;

endmodule
```



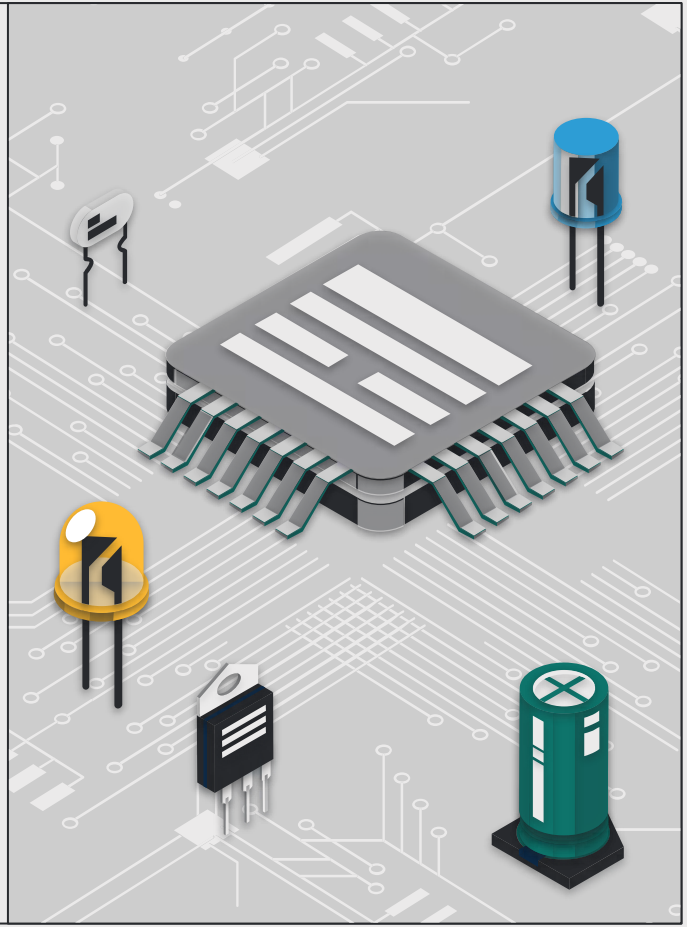
Logic unit

```
module Logic_unit #(parameter n = 32) (
    input [n-1:0] A,B,
    input [1:0] sel,
    output reg [n-1:0] OUT
);
    always @(*)
    begin
        case (sel)
            2'b00:OUT = A & B;
            2'b01:OUT = A | B;
            2'b10:OUT = ~(A | B);
            2'b11:OUT = A ^ B;
            default:OUT = A & B;
        endcase
    end
endmodule
```



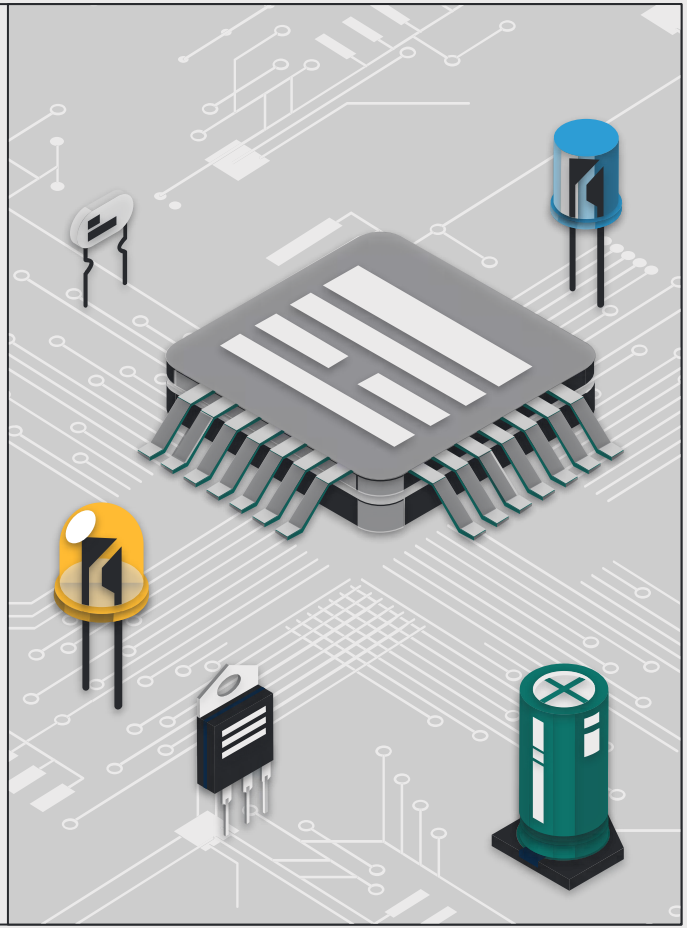
Shift lui unit

```
module Shift_lui_unit #(parameter n=32) (
    input [n-1:0] A,
    input signed [n-1:0] B,
    input [10:6] Shamt,
    input [1:0] sel,
    output reg signed [n-1:0] OUT
);
    always @ (*)
    begin
        case (sel)
        2'b00:OUT ={B[15:0],{16{1'b0}}};
        2'b01:OUT =B<<Shamt;
        2'b10:OUT =B>>Shamt;
        2'b11:OUT =B>>>Shamt;
        default:OUT ={B[15:0],{16{1'b0}}};
        endcase
    end

endmodule
```



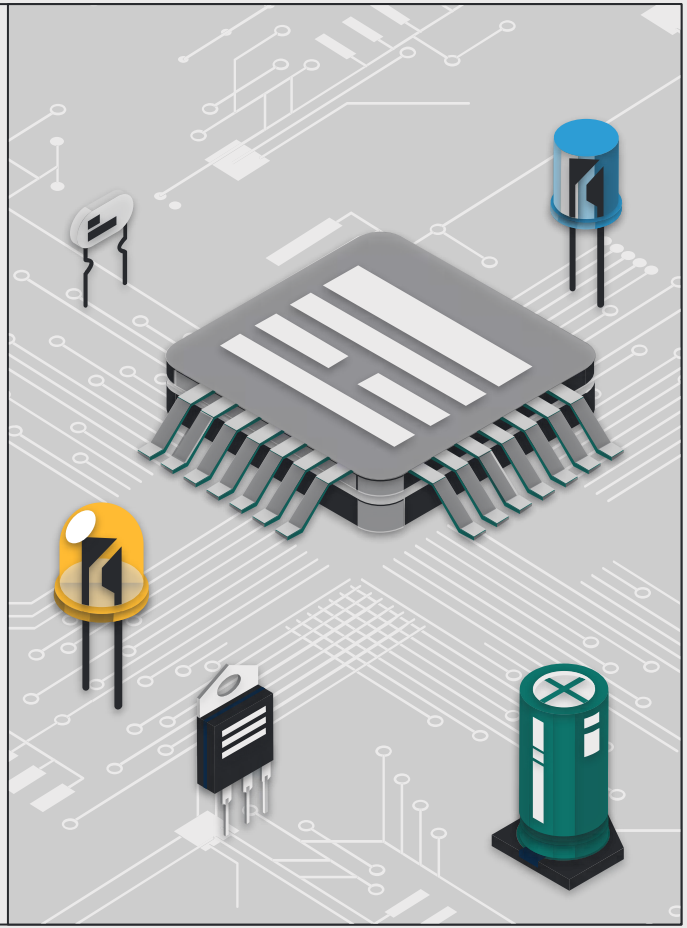
adder

```
module adder #(parameter n=32) (
    input [n-1:0] A,B,
    output [n-1:0] OUT
);
RCA #(.n(n)) ADDER (
    .A(A),
    .B(B),
    .Cin(1'b0),
    .Cout(),
    .overflow(),
    .LessThan(),
    .S(OUT)
);
endmodule
```



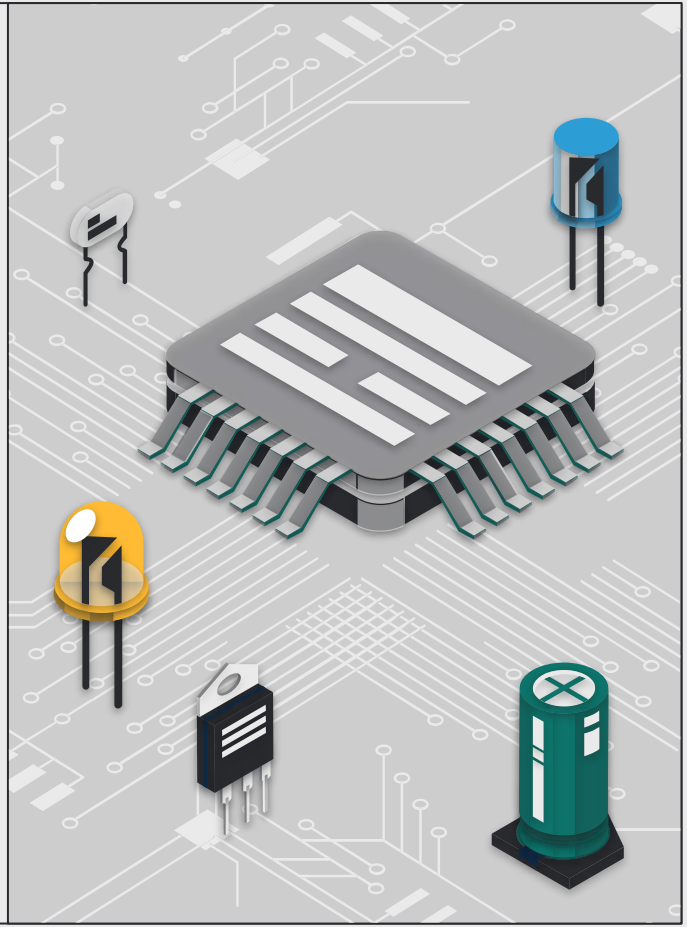
Add_4

```
module add_4 #(parameter n=32) (
    input [n-1:0] PC,
    output [n-1:0] PC_4
);
RCA #(.n(n)) ADDER (
    .A(PC),
    .B(32'd4),
    .Cin(1'b0),
    .Cout(),
    .overflow(),
    .LessThan(),
    .S(PC_4)
);
endmodule
```



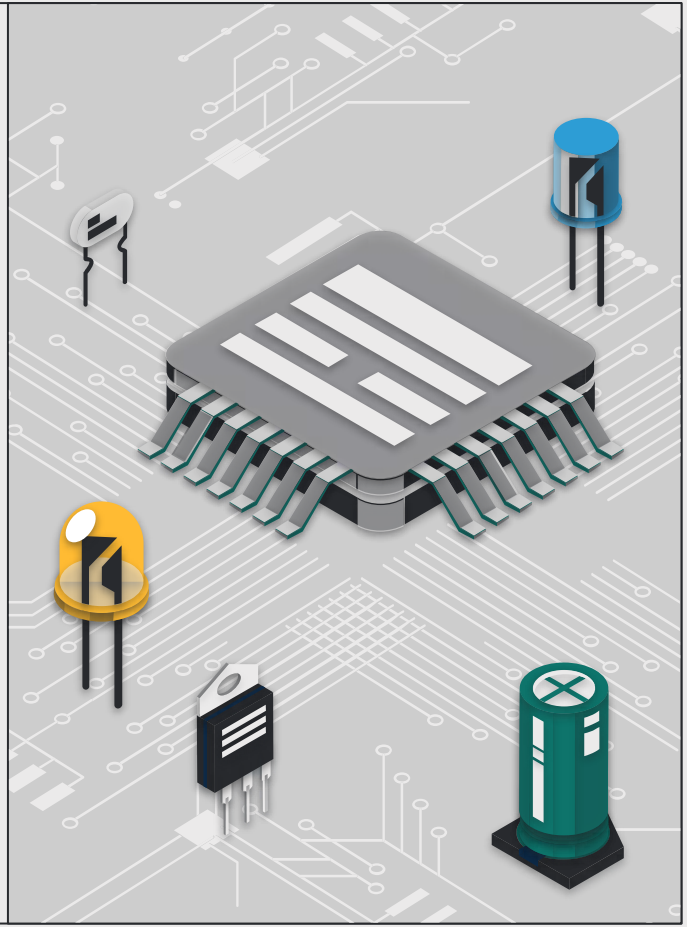
Control

```
module Control (
    input [31:26]OP_code,
    input [5:0]Function_field,
    output reg [1:0] RegDst_2,MemtoReg_2,Branch_2,Jump_2,SL_sel,
    output reg ALUSrc,RegWrite,MemRead,MemWrite,Sign,
    output reg [2:0] ALUOP
);
    always @ (*)
    begin
        case (OP_code)
        ///////////////
            6'd0:begin //jr
                if (Function_field==6'd8)
                    begin
                        RegDst_2=2'bxx;
                        MemtoReg_2=2'bxx;
                        Branch_2=2'bxx;
                        Jump_2=2'b10;
                        ALUSrc=1'bx;
                        RegWrite=1'b0;
                        MemRead=1'bx;///0
                        MemWrite=1'b0;
```



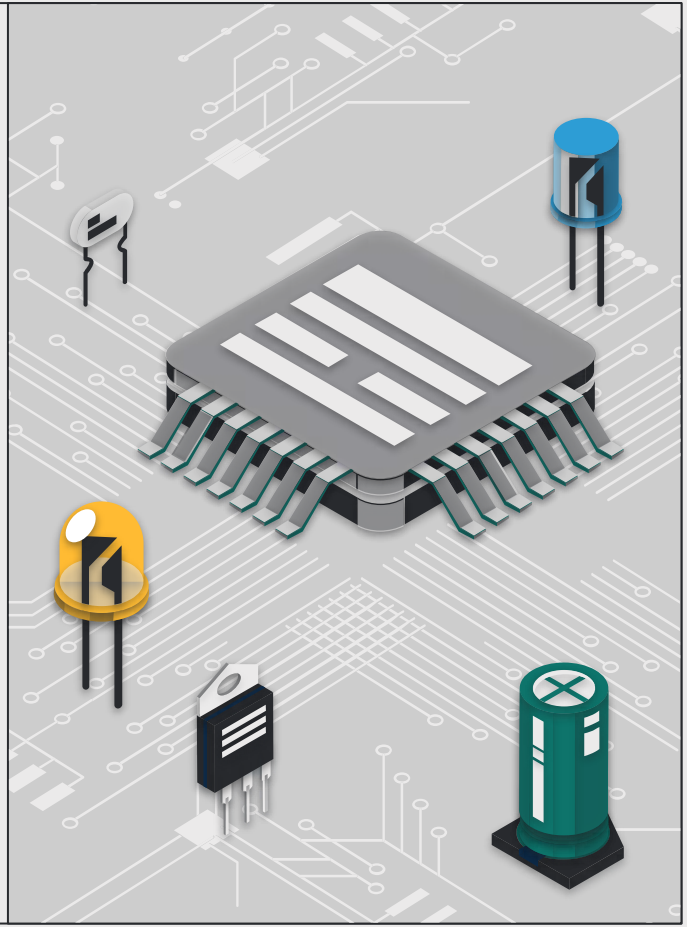
Control

```
            Sign=1'bx;
            ALUOP=3'bxxx;
            SL_sel=2'bx;
        end
    else if (Function_field==6'd20)
    begin//Lwr
            RegDst_2=2'b01;
            MemtoReg_2=2'b01;
            Branch_2=2'b00;
            Jump_2=2'b00;
            ALUSrc=1'b0;
            RegWrite=1'b1;
            MemRead=1'bx;///0
            MemWrite=1'b0;
            Sign=1'bx;
            ALUOP=3'b000;
            SL_sel=2'b00;
    end
```



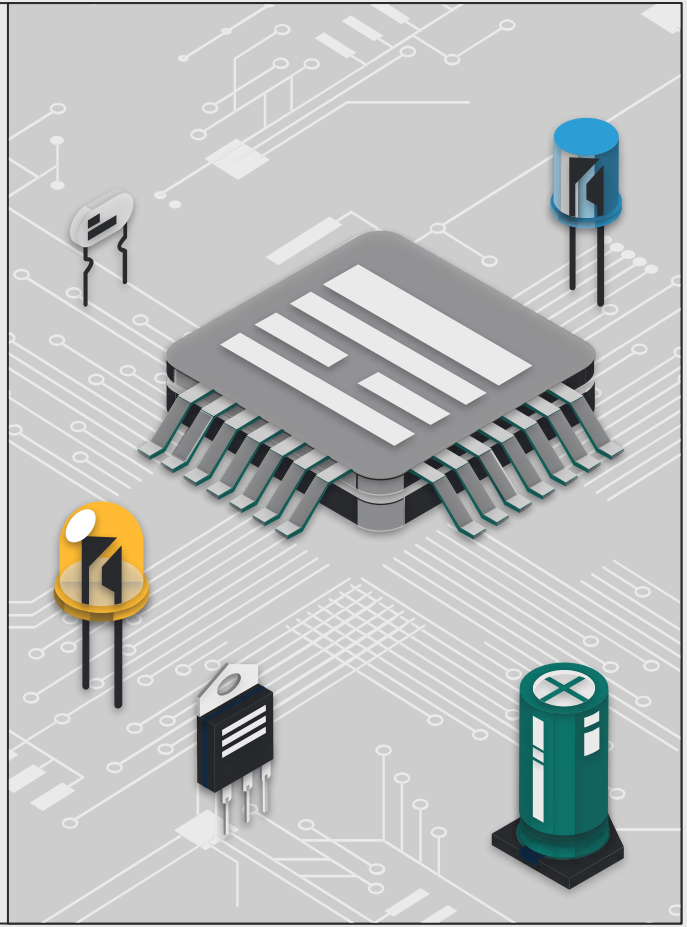
Control

```
                else
                    begin
                        RegDst_2=2'b01;
                        MemtoReg_2=2'b00;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b0;
                        RegWrite=1'b1;
                        MemRead=1'bx;///0
                        MemWrite=1'b0;
                        Sign=1'bx;
                        ALUOP=3'b111;
                        SL_sel=2'bx;
                    end
            end
        ////////////////////
            6'd2:begin  //j
                        RegDst_2=2'bxx;
                        MemtoReg_2=2'bxx;
                        Branch_2=2'bxx;
                        Jump_2=2'b01;
                        ALUSrc=1'bx;
```



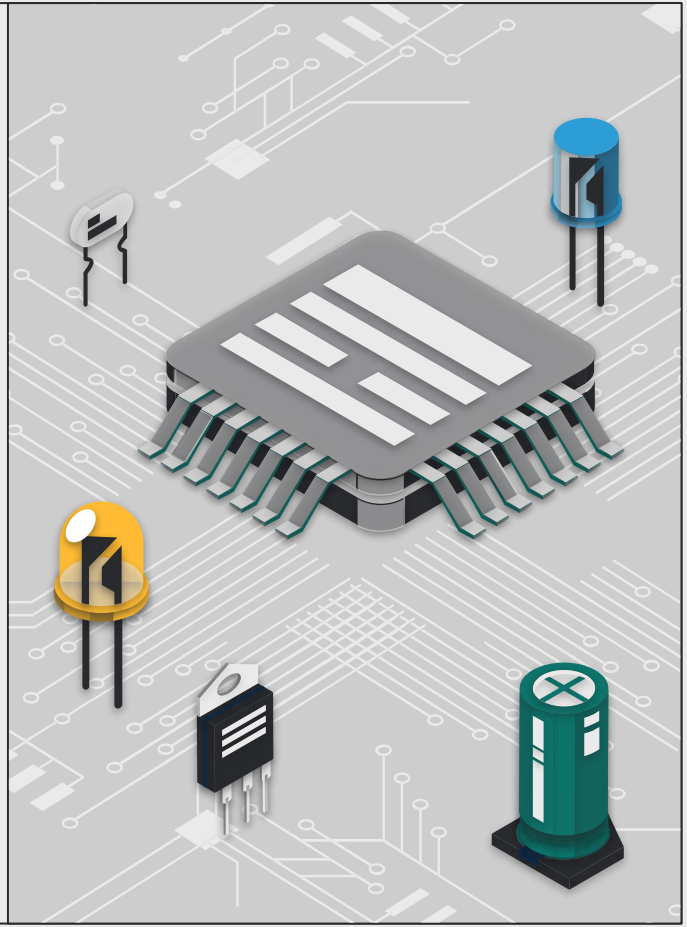
Control

```
                    RegWrite=1'b0;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'bx;
                    ALUOP=3'bxxx;
                    SL_sel=2'bx;
                end
        ///////////////
            6'd3:begin  //jal
                    RegDst_2=2'b10;
                    MemtoReg_2=2'b10;
                    Branch_2=2'bxx;
                    Jump_2=2'b01;
                    ALUSrc=1'bx;
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'bx;
                    ALUOP=3'bxxx;
                    SL_sel=2'bx;
                end
```



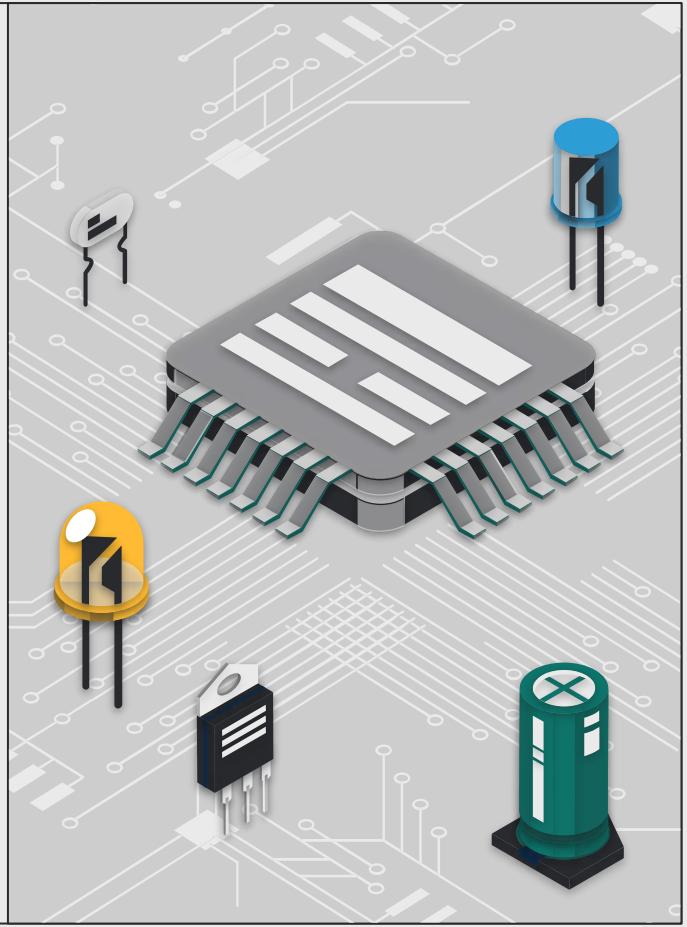
Control

```
            //////////////
                6'd4:begin  //beq
                            RegDst_2=2'bxx;
                            MemtoReg_2=2'bxx;
                            Branch_2=2'b01;
                            Jump_2=2'b00;
                            ALUSrc=1'b0;
                            RegWrite=1'b0;
                            MemRead=1'bx;///0
                            MemWrite=1'b0;
                            Sign=1'b1;
                            ALUOP=3'b001;
                            SL_sel=2'bx;
                        end
            //////////////
                6'd5:begin  //bne
                            RegDst_2=2'bxx;
                            MemtoReg_2=2'bxx;
                            Branch_2=2'b10;
                            Jump_2=2'b00;
                            ALUSrc=1'b0;
```



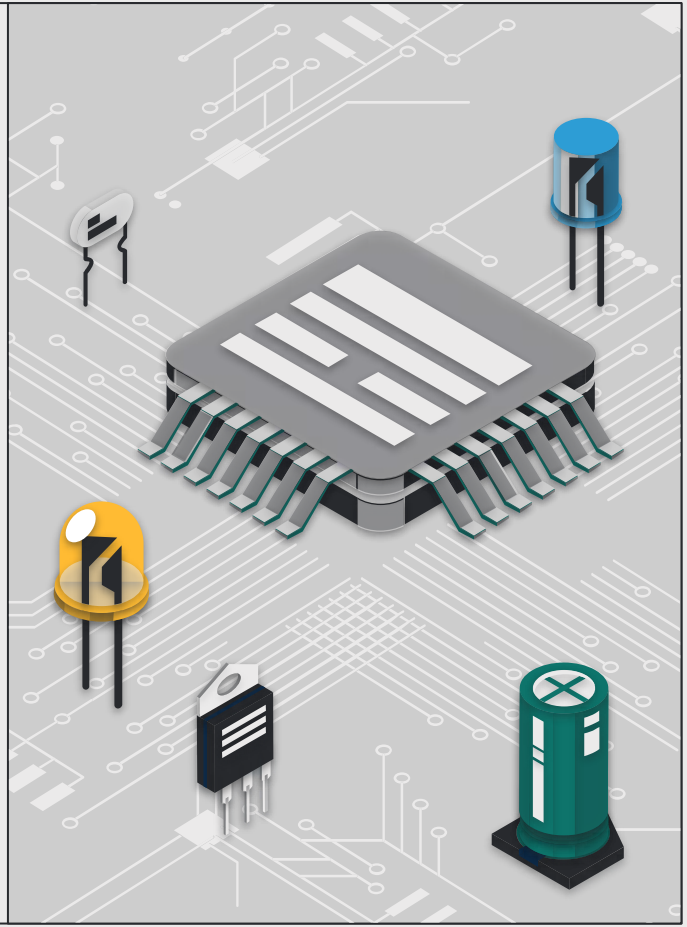
Control

```
                    RegWrite=1'b0;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'b1;
                    ALUOP=3'b001;
                    SL_sel=2'bx;
                end
        //////////////
            6'd8:begin  //addi
                    RegDst_2=2'b00;
                    MemtoReg_2=2'b00;
                    Branch_2=2'b00;
                    Jump_2=2'b00;
                    ALUSrc=1'b1;
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'b1;
                    ALUOP=3'b000;
                    SL_sel=2'bx;
                end
```



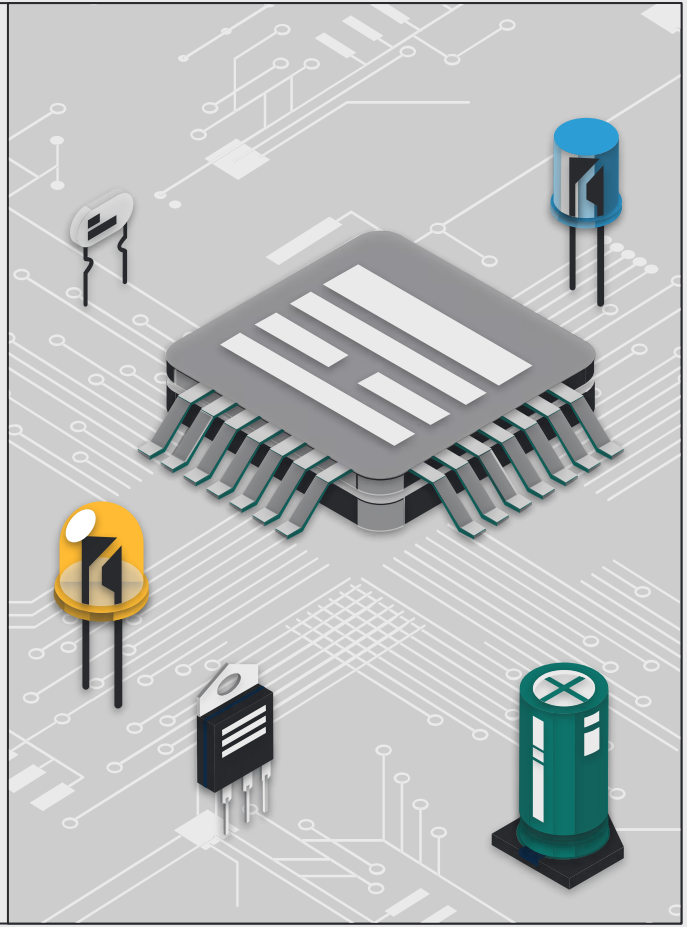
Control

```
        //////////////
            6'd10:begin  //slti
                        RegDst_2=2'b00;
                        MemtoReg_2=2'b00;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
                        RegWrite=1'b1;
                        MemRead=1'bx;///0
                        MemWrite=1'b0;
                        Sign=1'b1;
                        ALUOP=3'b101;
                        SL_sel=2'bx;
                    end
        //////////////
            6'd12:begin  //andi
                        RegDst_2=2'b00;
                        MemtoReg_2=2'b00;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
```



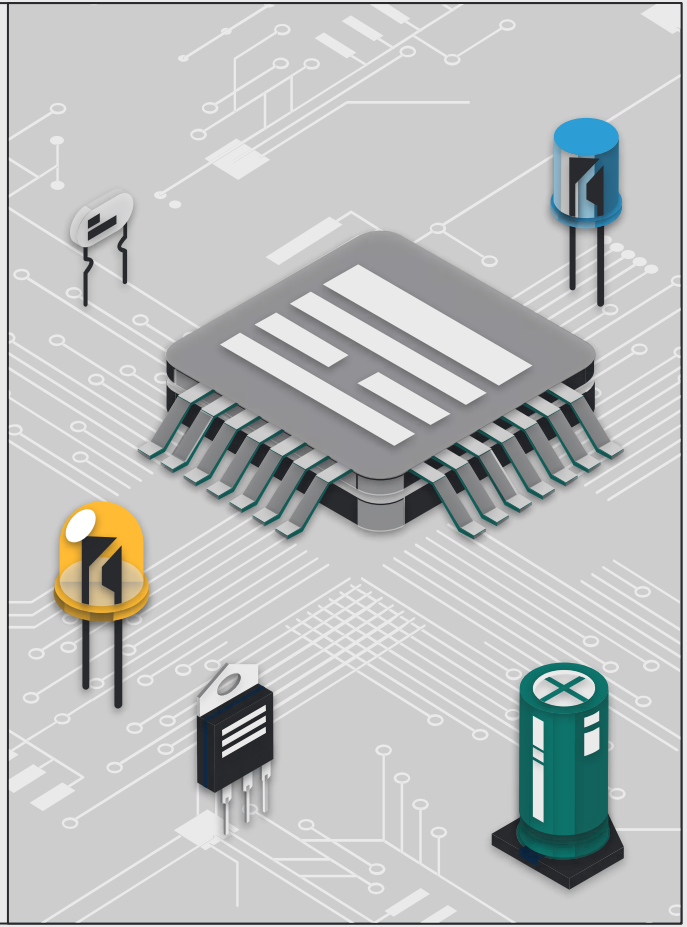
Control

```
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'b0;
                    ALUOP=3'b010;
                    SL_sel=2'bx;
                end
        ///////////////
            6'd13:begin  //ori
                    RegDst_2=2'b00;
                    MemtoReg_2=2'b00;
                    Branch_2=2'b00;
                    Jump_2=2'b00;
                    ALUSrc=1'b1;
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'b0;
                    ALUOP=3'b011;
                    SL_sel=2'bx;
                end
```



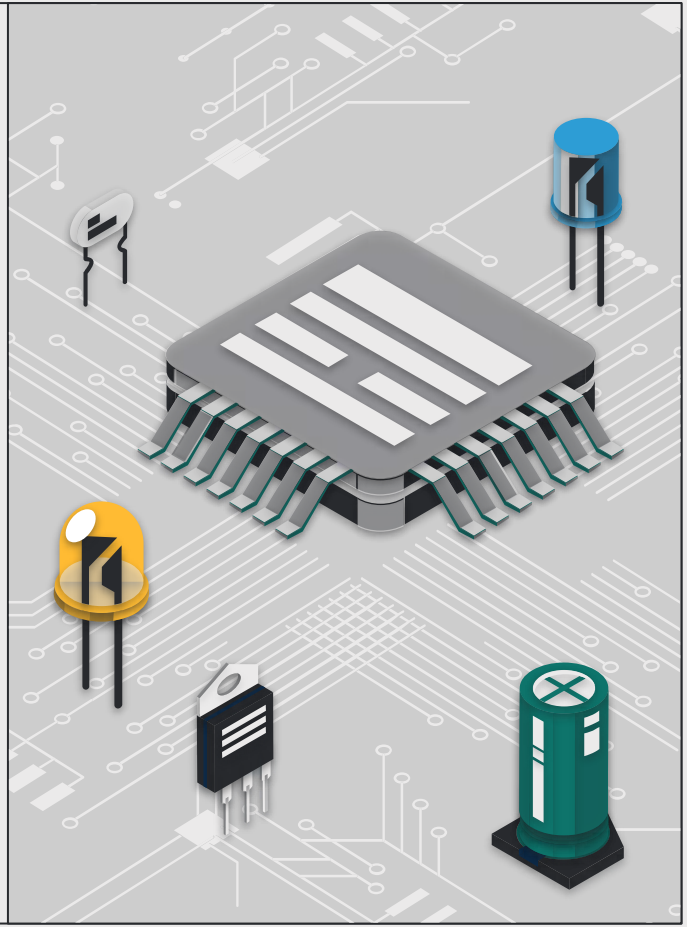
Control

```
        //////////////
            6'd14:begin  //xori
                        RegDst_2=2'b00;
                        MemtoReg_2=2'b00;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
                        RegWrite=1'b1;
                        MemRead=1'bx;///0
                        MemWrite=1'b0;
                        Sign=1'b0;
                        ALUOP=3'b100;
                        SL_sel=2'bx;
                    end
        //////////////
            6'd15:begin  //Lui
                        RegDst_2=2'b00;
                        MemtoReg_2=2'b00;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
```



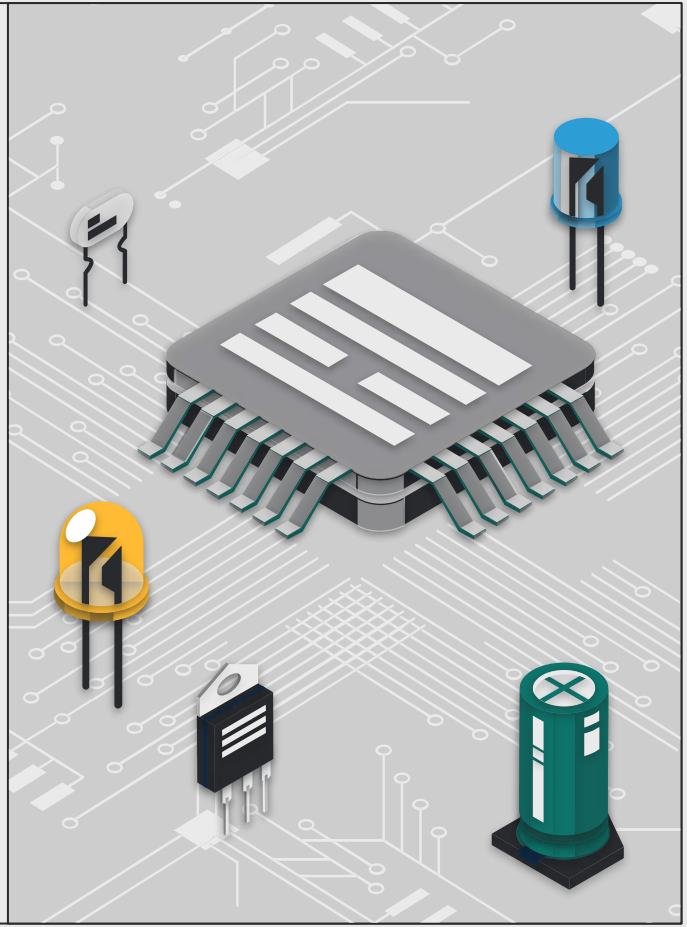
Control

```
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'bx;
                    ALUOP=3'b110;
                    SL_sel=2'bx;
                end
        ///////////////
            6'd35:begin  //LW
                    RegDst_2=2'b00;
                    MemtoReg_2=2'b01;
                    Branch_2=2'b00;
                    Jump_2=2'b00;
                    ALUSrc=1'b1;
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'b1;
                    ALUOP=3'b000;
                    SL_sel=2'b00;
                end
```



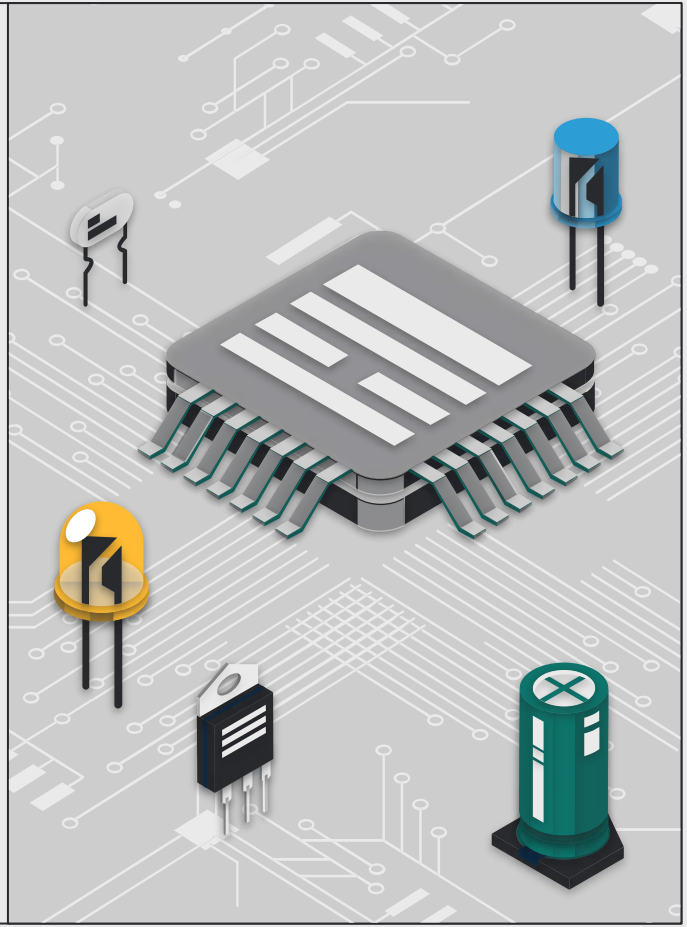
Control

```
        ///////////////
            6'd43:begin  //SW
                        RegDst_2=2'bxx;
                        MemtoReg_2=2'bxx;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
                        RegWrite=1'b0;
                        MemRead=1'bx;///0
                        MemWrite=1'b1;
                        Sign=1'b1;
                        ALUOP=3'b000;
                        SL_sel=2'b00;
                    end
        ///////////////
```



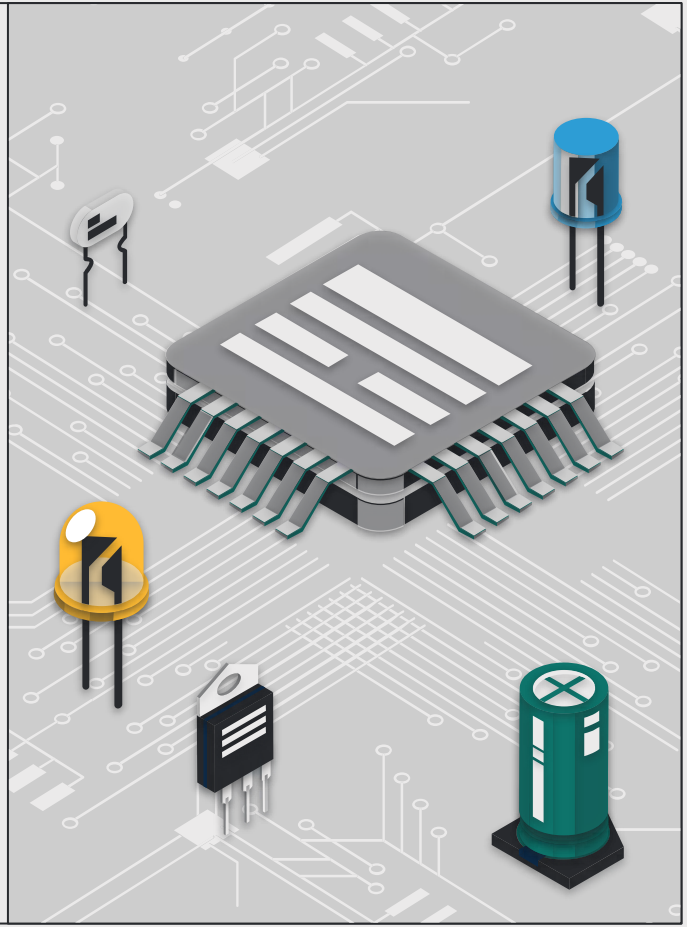
Control

```
        //////////////
            6'd33:begin  //LH
                        RegDst_2=2'b00;
                        MemtoReg_2=2'b01;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
                        RegWrite=1'b1;
                        MemRead=1'bx;///0
                        MemWrite=1'b0;
                        Sign=1'b1;
                        ALUOP=3'b000;
                        SL_sel=2'b01;
                    end
        //////////////
            6'd41:begin  //SH
                        RegDst_2=2'bxx;
                        MemtoReg_2=2'bxx;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
```



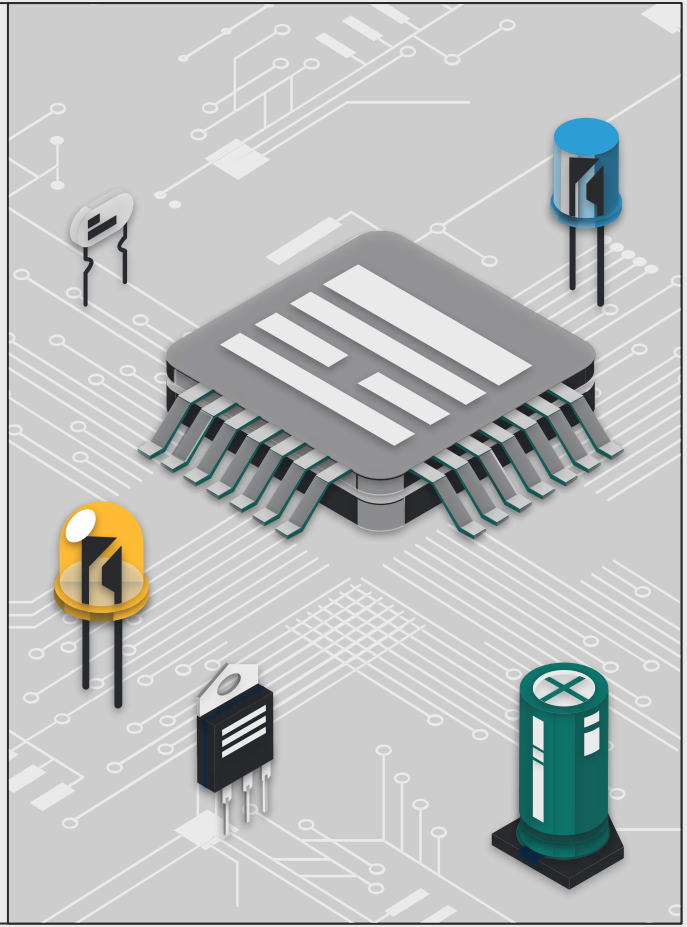
Control

```
                    RegWrite=1'b0;
                      MemRead=1'bx;///0
                      MemWrite=1'b1;
                      Sign=1'b1;
                      ALUOP=3'b000;
                      SL_sel=2'b01;
                end
        ///////////////
            6'd32:begin  //LB
                      RegDst_2=2'b00;
                      MemtoReg_2=2'b01;
                      Branch_2=2'b00;
                      Jump_2=2'b00;
                      ALUSrc=1'b1;
                      RegWrite=1'b1;
                      MemRead=1'bx;///0
                      MemWrite=1'b0;
                      Sign=1'b1;
                      ALUOP=3'b000;
                      SL_sel=2'b10;
                end
```



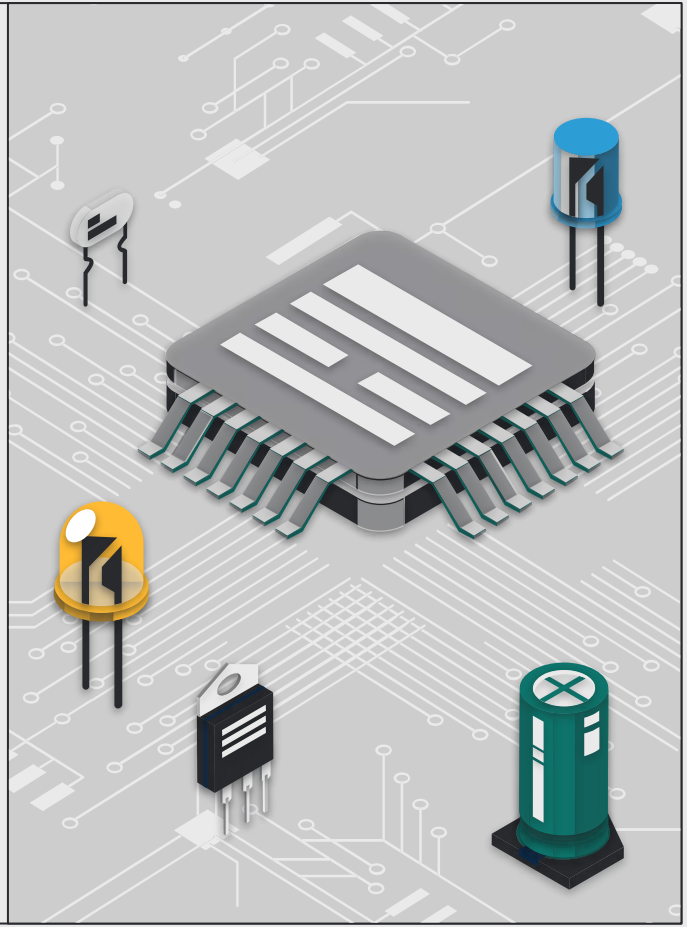
Control

```
        //////////////
            6'd40:begin  //SB
                        RegDst_2=2'bxx;
                        MemtoReg_2=2'bxx;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
                        RegWrite=1'b0;
                        MemRead=1'bx;///0
                        MemWrite=1'b1;
                        Sign=1'b1;
                        ALUOP=3'b000;
                        SL_sel=2'b10;
                    end
        //////////////

 ////////////////////////////////////////////////////////
        //////////////
```



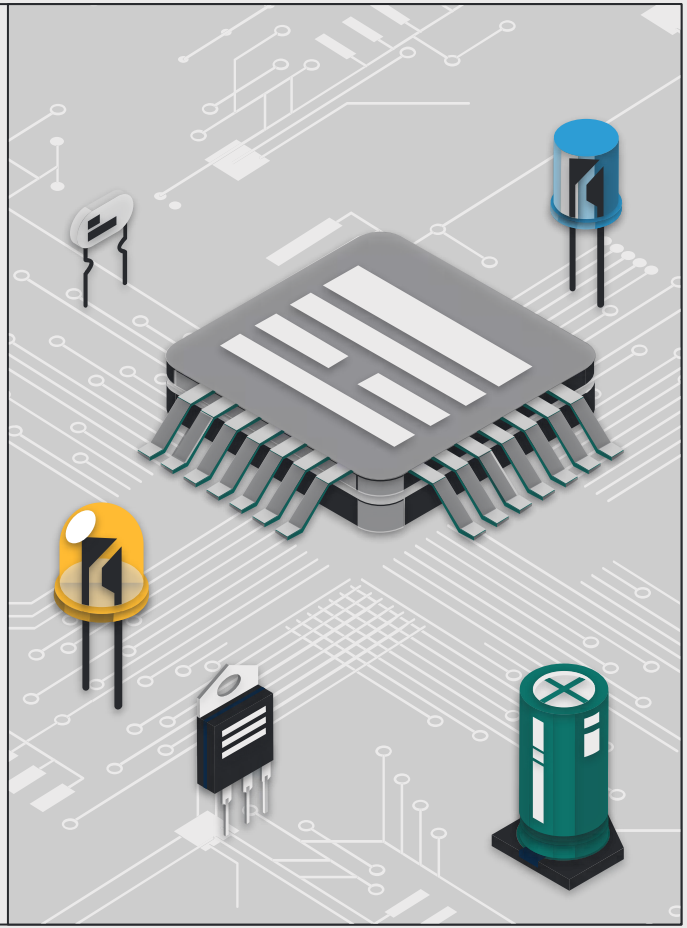
Control

```
            6'd52:begin  //jalm
                        RegDst_2=2'b10;
                        MemtoReg_2=2'b10;
                        Branch_2=2'bxx;
                        Jump_2=2'b11;
                        ALUSrc=1'b1;
                        RegWrite=1'b1;
                        MemRead=1'bx;///0
                        MemWrite=1'b0;
                        Sign=1'b1;
                        ALUOP=3'b000;
                        SL_sel=2'b00;
                    end
        ///////////////
        ///////////////
            6'd53:begin  //jalr
                        RegDst_2=2'b01;
                        MemtoReg_2=2'b10;
                        Branch_2=2'bxx;
                        Jump_2=2'b10;
                        ALUSrc=1'bx;
```



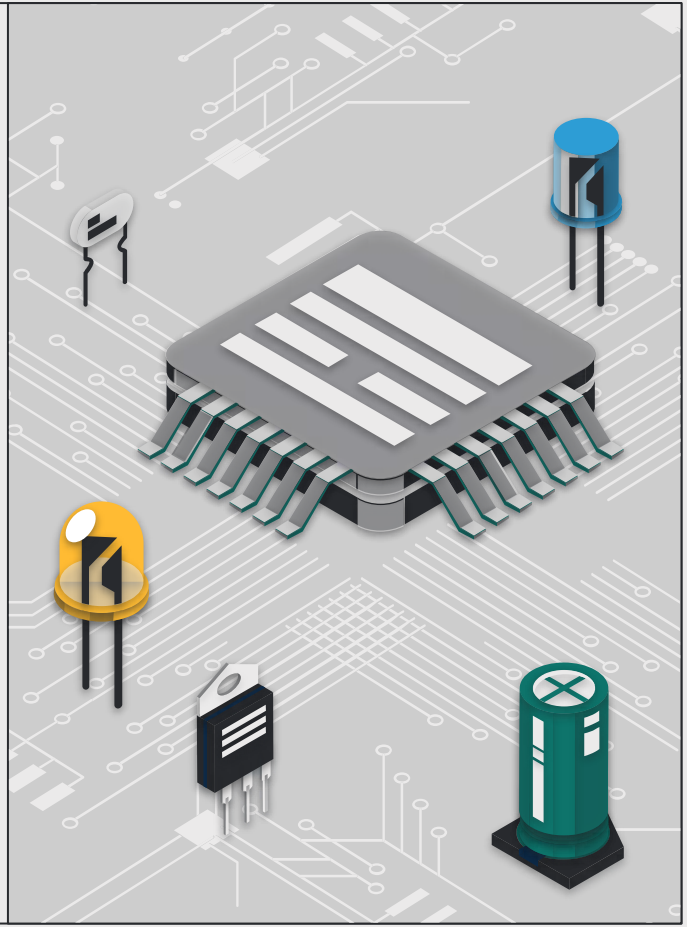
Control

```
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'bx;
                    ALUOP=3'bxxx;
                    SL_sel=2'bxx;
                end
        ///////////////
        ///////////////
            6'd54:begin  //jm
                    RegDst_2=2'bxx;
                    MemtoReg_2=2'bxx;
                    Branch_2=2'bxx;
                    Jump_2=2'b11;
                    ALUSrc=1'b1;
                    RegWrite=1'b0;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'b1;
                    ALUOP=3'b000;
                    SL_sel=2'b00;
                end
```



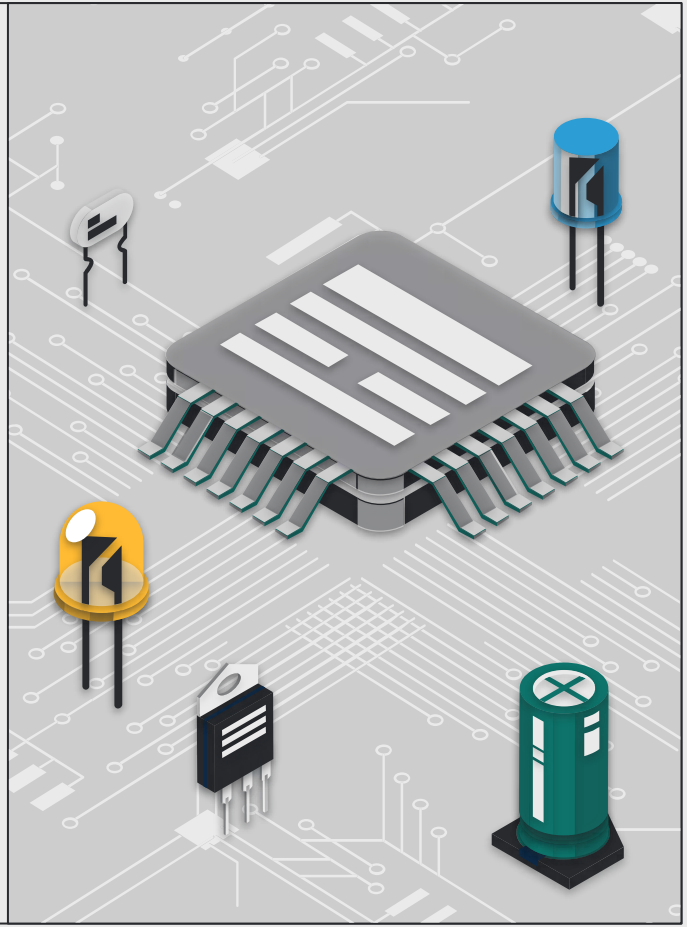
Control

```
        ///////////////
        ///////////////
            6'd51:begin  //subi
                        RegDst_2=2'b00;
                        MemtoReg_2=2'b00;
                        Branch_2=2'b00;
                        Jump_2=2'b00;
                        ALUSrc=1'b1;
                        RegWrite=1'b1;
                        MemRead=1'bx;///0
                        MemWrite=1'b0;
                        Sign=1'b1;
                        ALUOP=3'b001;
                        SL_sel=2'bx;
                    end
        ///////////////
        ///////////////
            6'd49:begin  //Lbit
                        RegDst_2=2'b00;
                        MemtoReg_2=2'b01;
                        Branch_2=2'b00;
```



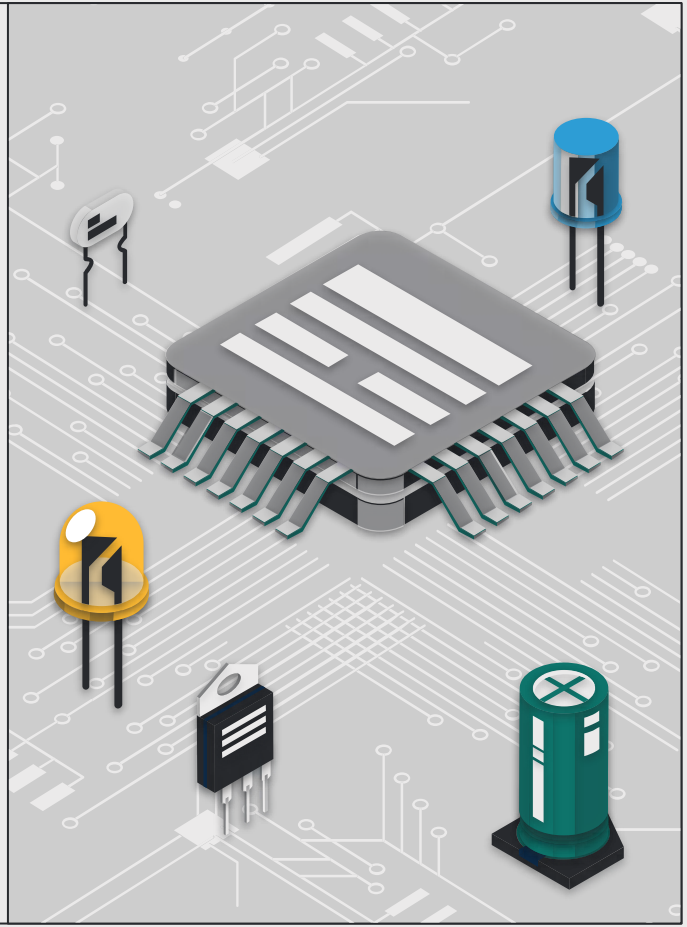
Control

```
                    Jump_2=2'b00;
                    ALUSrc=1'b1;
                    RegWrite=1'b1;
                    MemRead=1'bx;///0
                    MemWrite=1'b0;
                    Sign=1'b1;
                    ALUOP=3'b000;
                    SL_sel=2'b11;
                end
        ///////////////
            6'd50:begin  //Sbit
                    RegDst_2=2'bxx;
                    MemtoReg_2=2'bxx;
                    Branch_2=2'b00;
                    Jump_2=2'b00;
                    ALUSrc=1'b1;
                    RegWrite=1'b0;
                    MemRead=1'bx;///0
                    MemWrite=1'b1;
                    Sign=1'b1;
                    ALUOP=3'b000;
                    SL_sel=2'b11;
```



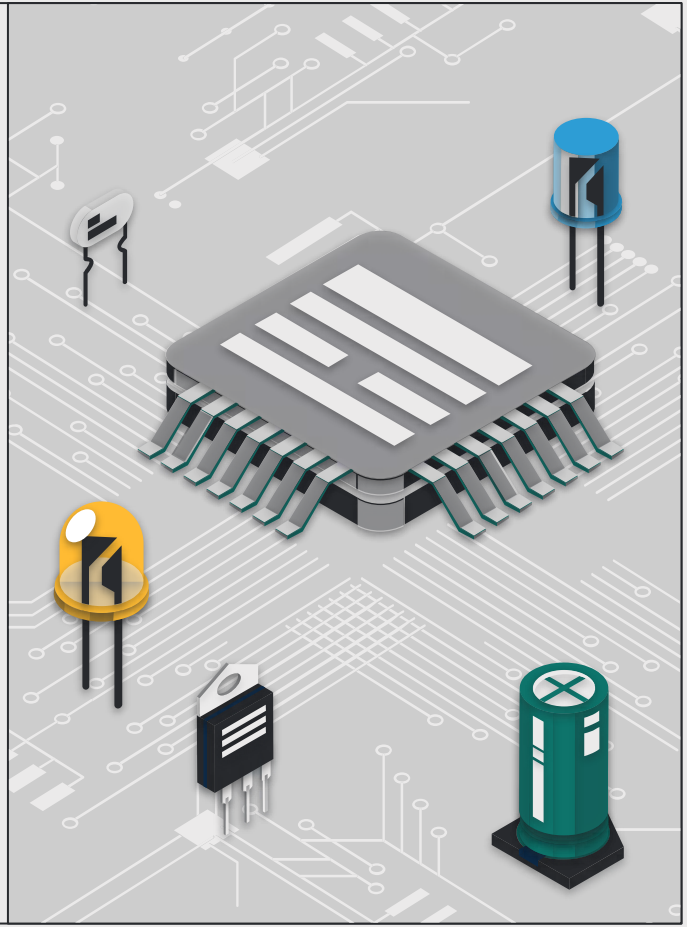
Control

```
                    end
            //////////////

                default:    begin
                            RegDst_2=2'bxx;
                            MemtoReg_2=2'bxx;
                            Branch_2=2'bxx;
                            Jump_2=2'bxx;
                            ALUSrc=1'bx;
                            RegWrite=1'bx;
                            MemRead=1'bx;///
                            MemWrite=1'bx;
                            Sign=1'bx;
                            ALUOP=3'bxxx;
                            SL_sel=2'bx;
                        end
            endcase
        end

    endmodule
```



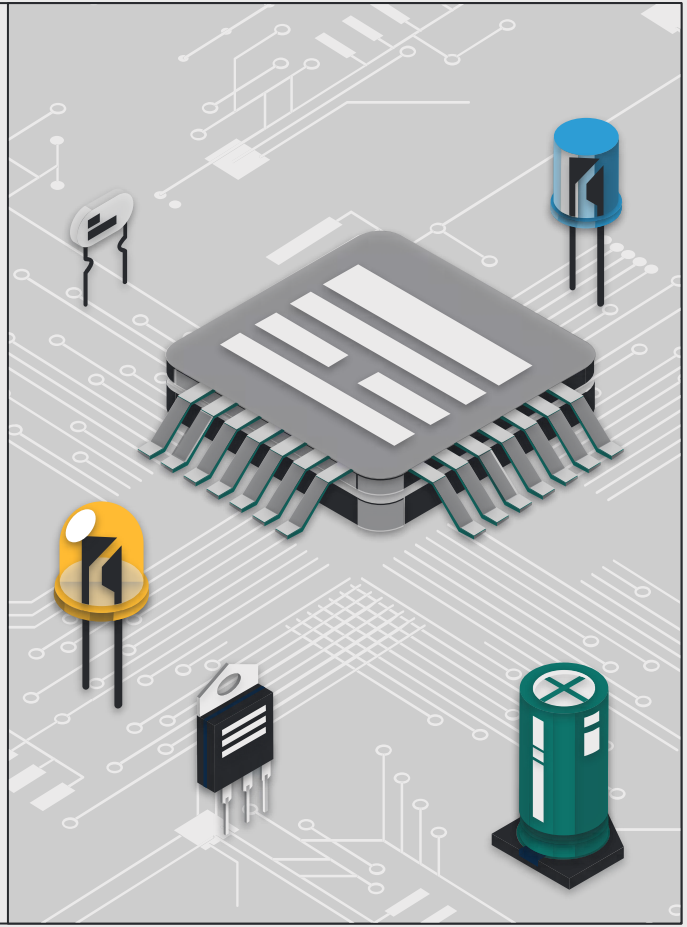
ALU control

```
module ALU_control (
    input [2:0] ALUOP,
    input [5:0] Funct,
    output reg[3:0]ALU_sel
);
    always @ (*)
    begin
        case (ALUOP)
        3'b000:ALU_sel=4'b00x0;//add
        3'b001:ALU_sel=4'b10x0;//sub
        3'b010:ALU_sel=4'b0001;//and
        3'b011:ALU_sel=4'b0101;//or
        3'b100:ALU_sel=4'b1101;//xor
        3'b101:ALU_sel=4'b11x0;//slt
        3'b110:ALU_sel=4'b0011;//lui
        3'b111://function feild
        begin
```



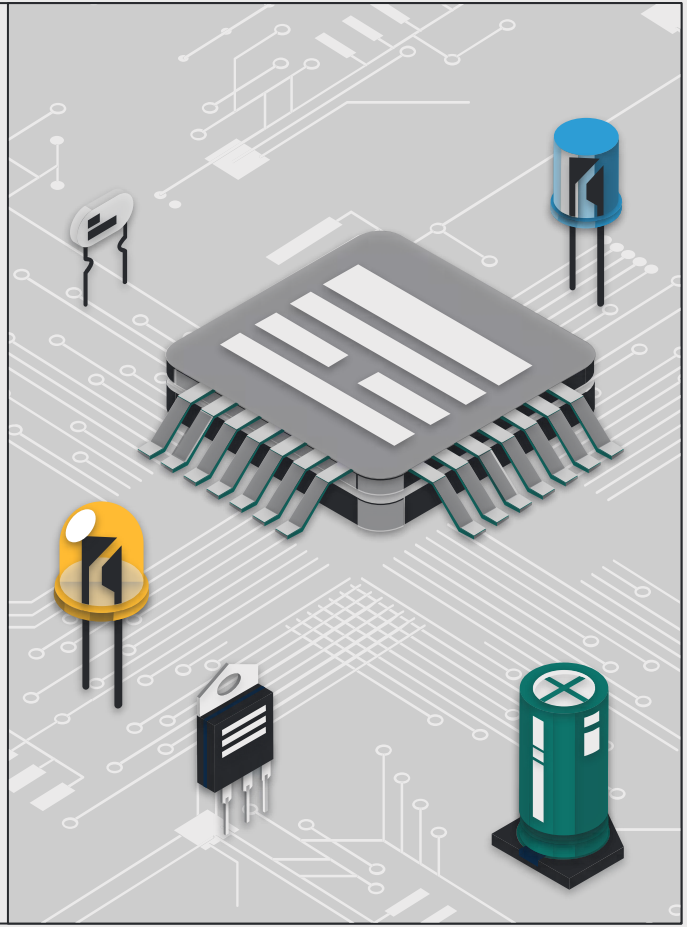
ALU control

```
            case (Funct)
            6'd0:ALU_sel=4'b0111;//sll
            6'd2:ALU_sel=4'b1011;//srl
            6'd3:ALU_sel=4'b1111;//sra
            6'd32:ALU_sel=4'b00x0;//add
            6'd34:ALU_sel=4'b10x0;//sub
            6'd36:ALU_sel=4'b0001;//and
            6'd37:ALU_sel=4'b0101;//or
            6'd38:ALU_sel=4'b1101;//xor
            6'd39:ALU_sel=4'b1001;//nor
            6'd42:ALU_sel=4'b11x0;//slt
            default :ALU_sel=4'bxxxx;
            endcase
        end
        default :ALU_sel=4'bxxxx;
        endcase
    end
endmodule
```



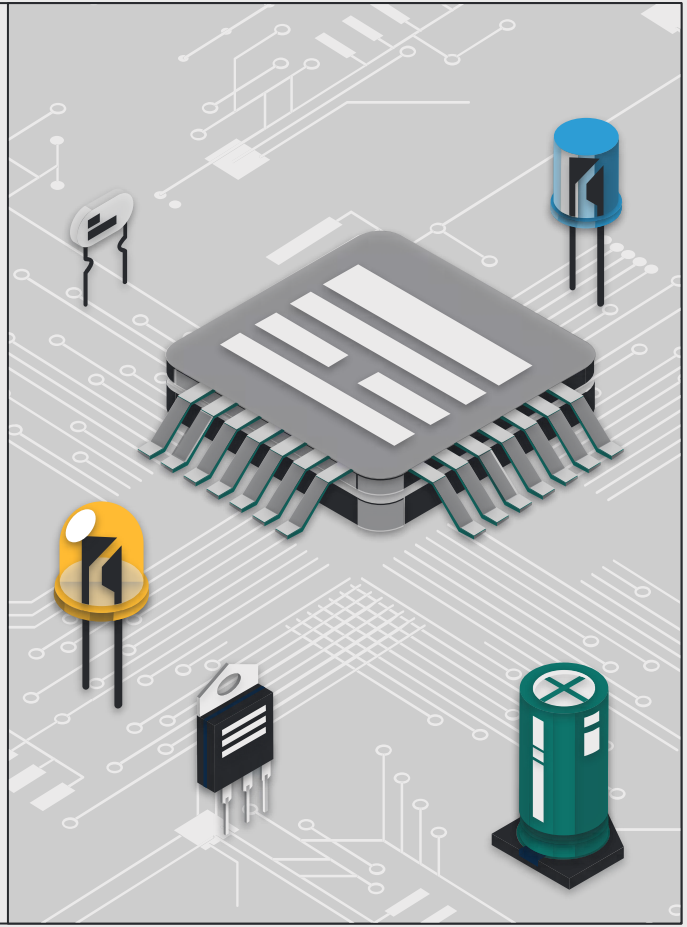
Branch control

```
module Branch_control (
    input [1:0]Branch,//bne beq
    input Zero,
    output OUT
);
    assign OUT = (Branch[0] & Zero)| (Branch[1] & ~Zero);
endmodule
```



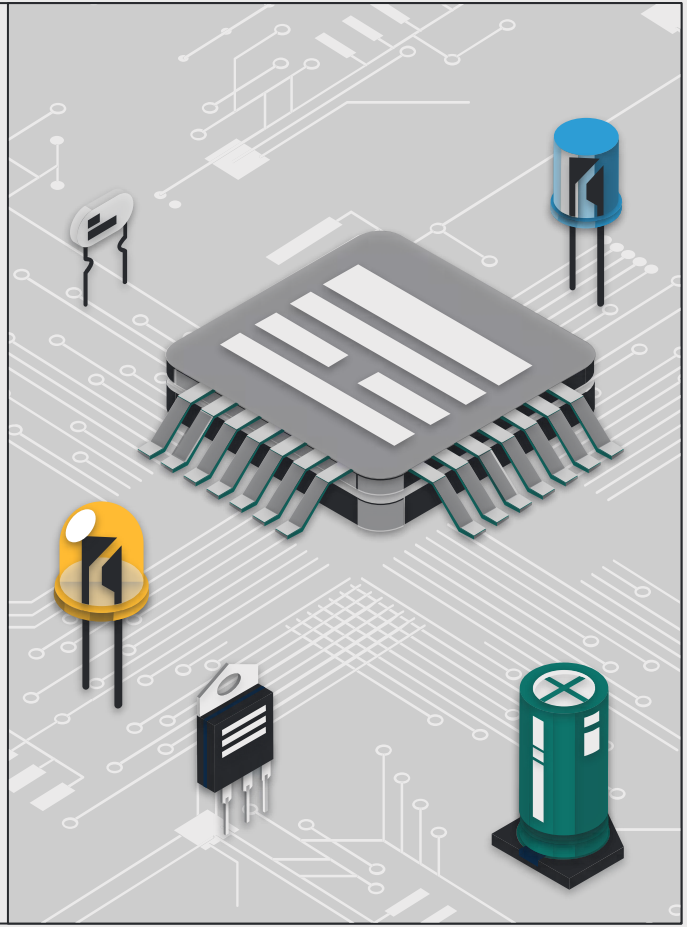
Extend unit

```
module extend_unit (
    input [15:0] IN,
    input Sign,
    output reg [31:0] OUT
);
    always @ (*)
    begin
    if (Sign)
        OUT = {{16{IN[15]}},IN};
    else
        OUT = {{16{1'b0}},IN};
    end

endmodule
```



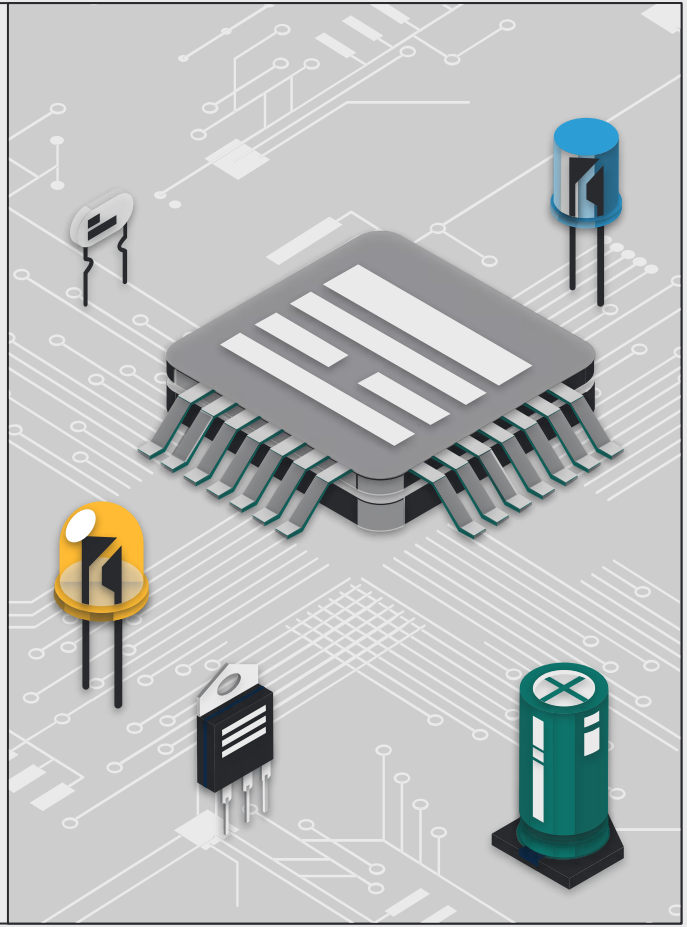
Shift left2 32→32

```
module shift_left_2_32_32 (
    input [31:0] IN,
    output [31:0] OUT
);
    assign OUT = {IN[29:0],2'b00};
endmodule
```



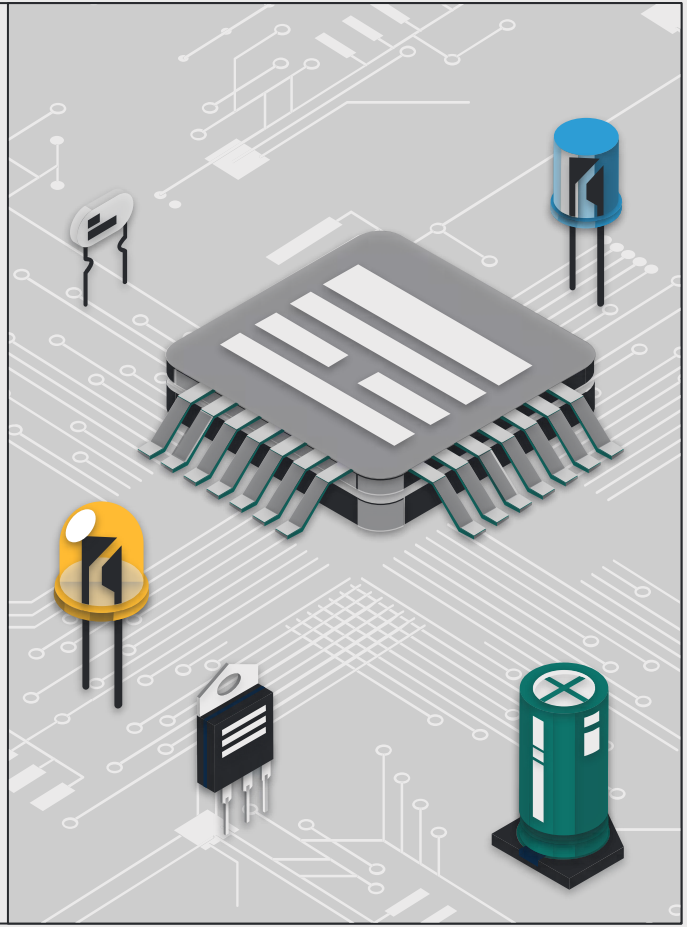
Shift left2 26→28

```
module shift_left_2_26_28 (
    input [25:0] IN,
    output [27:0] OUT
);
    assign OUT = {IN,2'b00};
endmodule
```



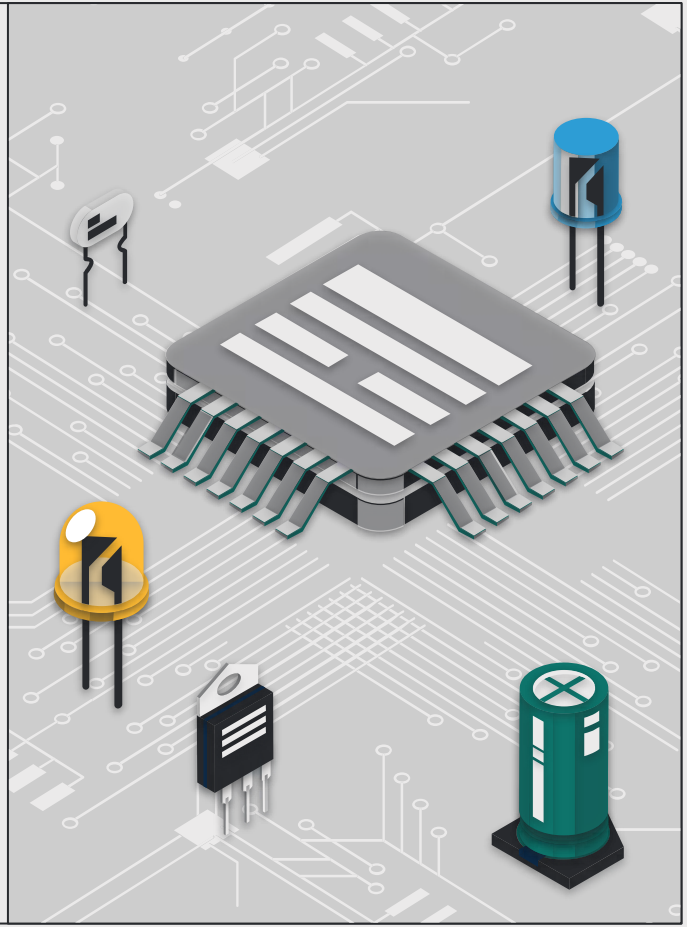
Concatenation unit

```
module concatenation_unit (
    input [27:0] IN,
    input [31:28] PC_in,
    output [31:0] JumpAddress
);
    assign JumpAddress={PC_in,IN};
endmodule
```



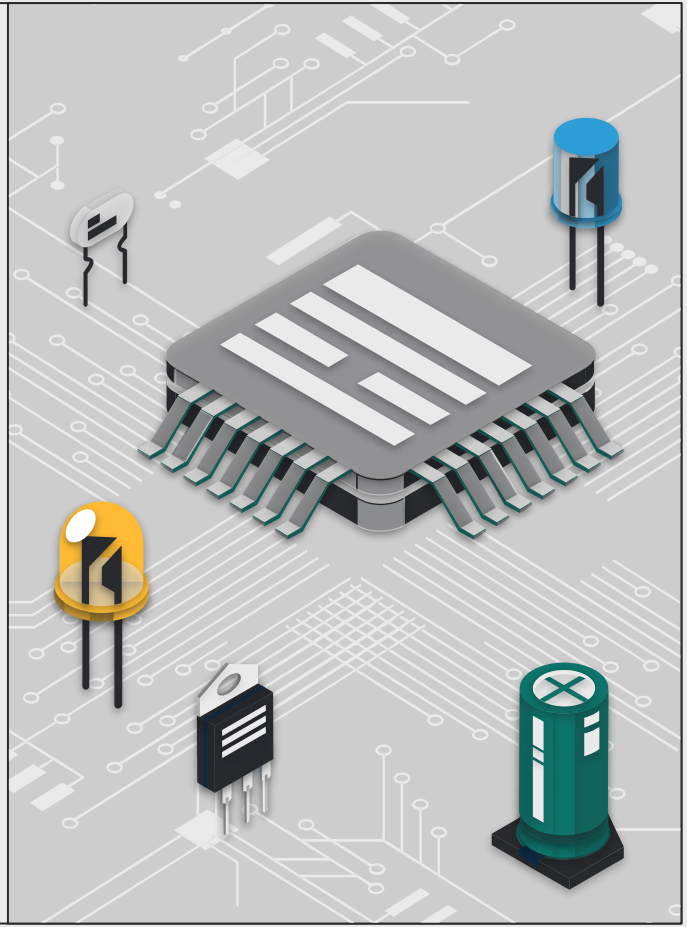
Registers

```
module Registers (
    input [4:0]Read_register1,Read_register2,
    input [4:0]Write_register,
    output [31:0]Read_data1,Read_data2,
    input [31:0] Write_data,
    input clk,RegWrite
    );
    reg [31:0] register [31:0];
        assign Read_data1= register
[Read_register1];
        assign Read_data2= register
[Read_register2];
    initial
    register [0]=0;
    always @ (posedge clk)
    begin
    if (RegWrite)
        register [Write_register]<=Write_data;
    else;
    end
endmodule
```



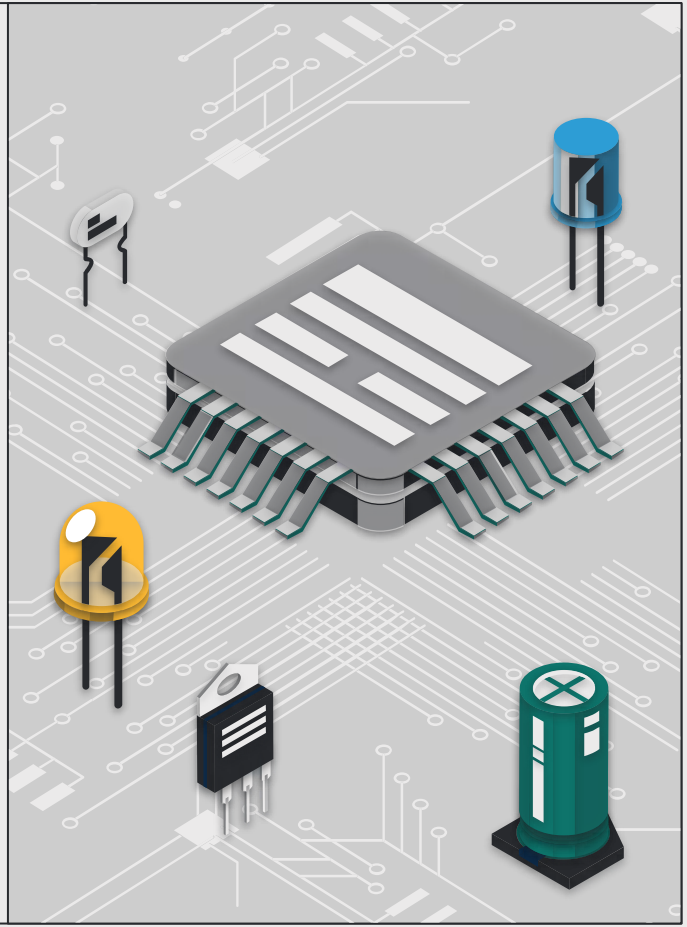
Instruction memory

```
module Instruction_memory (
    input [31:2] ReadAddress,WriteAddress,
    input writeINS,clk,
    input [31:0] writeDataINS,
    output reg [31:0] Instruction
);
    reg [31:0] mem [127:0];
    always @ (posedge clk)
    begin
        if (writeINS)
            mem[WriteAddress]<=writeDataINS;
        else;
    end
    always @ (*)
    Instruction<=mem[ReadAddress];
    initial
    begin
    mem[0]<=32'h8c080000;//lw $t0, 0($0)          # lw
    mem[1]<=32'h21090005;//addi $t1, $t0, 5       # addi
    mem[2]<=32'h00095080;//sll $t2, $t1, 2        # sll
```



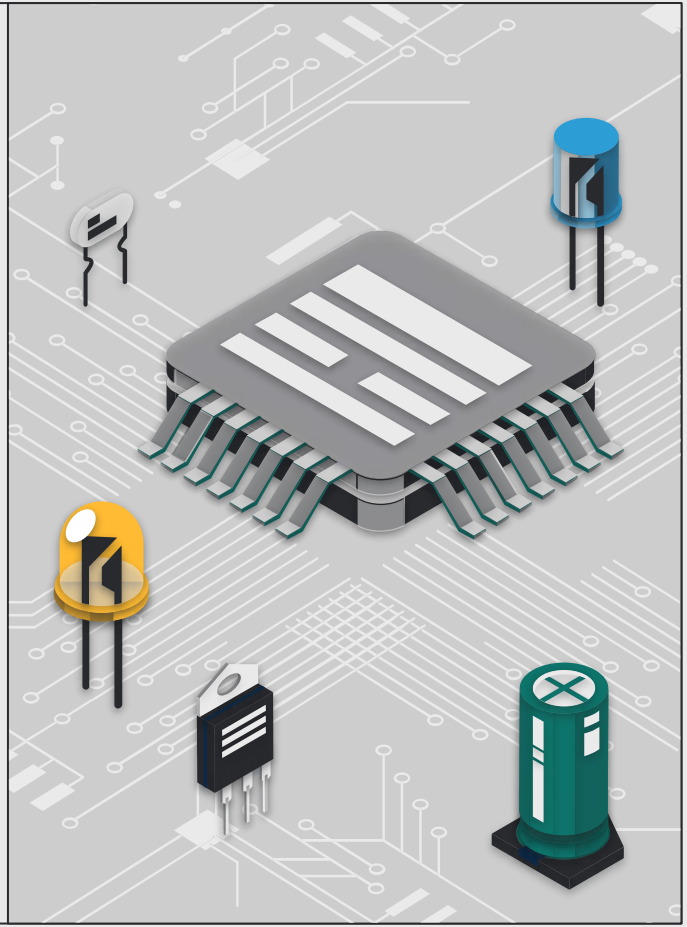
Instruction memory

```
mem[3]<=32'h3c0b1000;//lui $t3, 0x1000         # lui
mem[4]<=32'h016a6026;//xor $t4, $t3, $t2       # xor
mem[5]<=32'h800d0004;//lb $t5, 4($0)           # lb
mem[6]<=32'h840e0006;//lh $t6, 6($0)           # lh
mem[7]<=32'h01ae782a;//slt $t7, $t5, $t6       # slt
mem[8]<=32'h01aec024;//and $t8, $t5, $t6       # and
mem[9]<=32'h29b9000a;//slti $t9, $t5, 10       # slti
mem[10]<=32'h31b000ff;//andi $s0, $t5, 0xFF     # andi
mem[11]<=32'h000d8883;//sra $s1, $t5, 2         # sra
mem[12]<=32'h11ae0006;//beq $t5, $t6, label1    # beq
mem[13]<=32'h01ae9027;//nor $s2, $t5, $t6       # nor
mem[14]<=32'h000e98c2;//srl $s3, $t6, 3         # srl
mem[15]<=32'h15e00005;//bne $t7, $0, label2     # bne
mem[16]<=32'h01aea025;//or $s4, $t5, $t6        # or
mem[17]<=32'h01cda822;//sub $s5, $t6, $t5       # sub
mem[18]<=32'h08000017;//j label3                # j
mem[19]<=32'h20160001;//addi $s6, $0, 1     # addi (example)
mem[20]<=32'h08000018;//j label4              # j
mem[21]<=32'h20160002;//addi $s6, $0, 2     # addi (example)
```



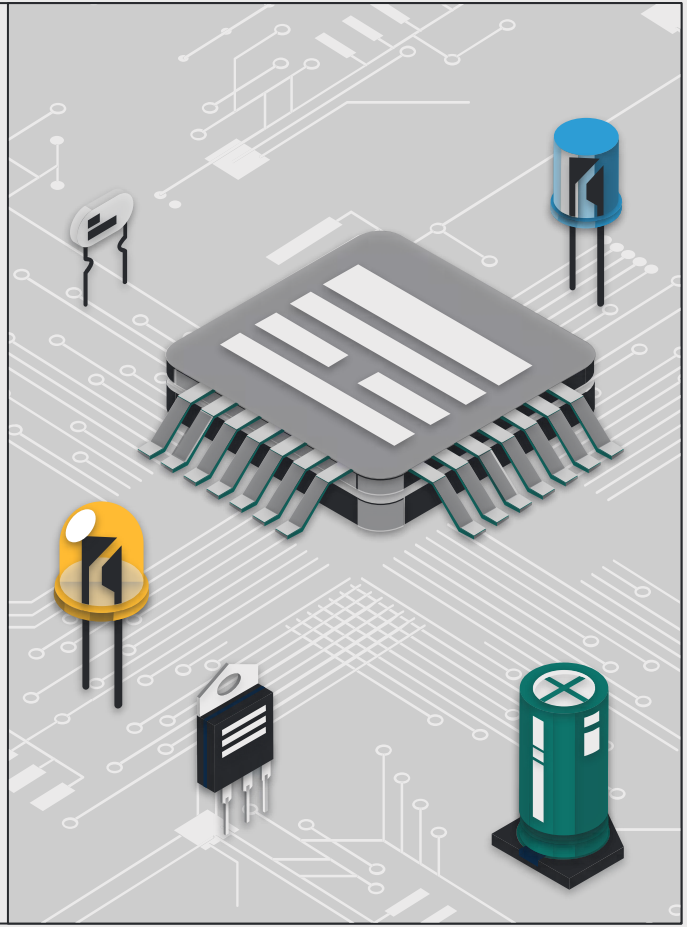
Instruction memory

```
    mem[22]<=32'h08000018;//j label4             # j
    mem[23]<=32'h20160003;//addi $s6, $0, 3  # addi (example)
    mem[24]<=32'h35b70055;//ori $s7, $t5, 0x55     # ori
    mem[25]<=32'hac170008;//sw $s7, 8($0)          # sw
    mem[26]<=32'h0c00001e;//jal subroutine          # jal
    mem[27]<=32'ha00d000a;//sb $t5, 10($0)         # sb
    mem[28]<=32'ha40e000c;//sh $t6, 12($0)         # sh
    mem[29]<=32'h39b000ff;//xori $s0, $t5, 0xFF    # xori
    mem[30]<=32'h02118020;//add $s0, $s0, $s1  # add
    mem[31]<=32'h03e00008;//jr $ra               # jr

     end
endmodule
```



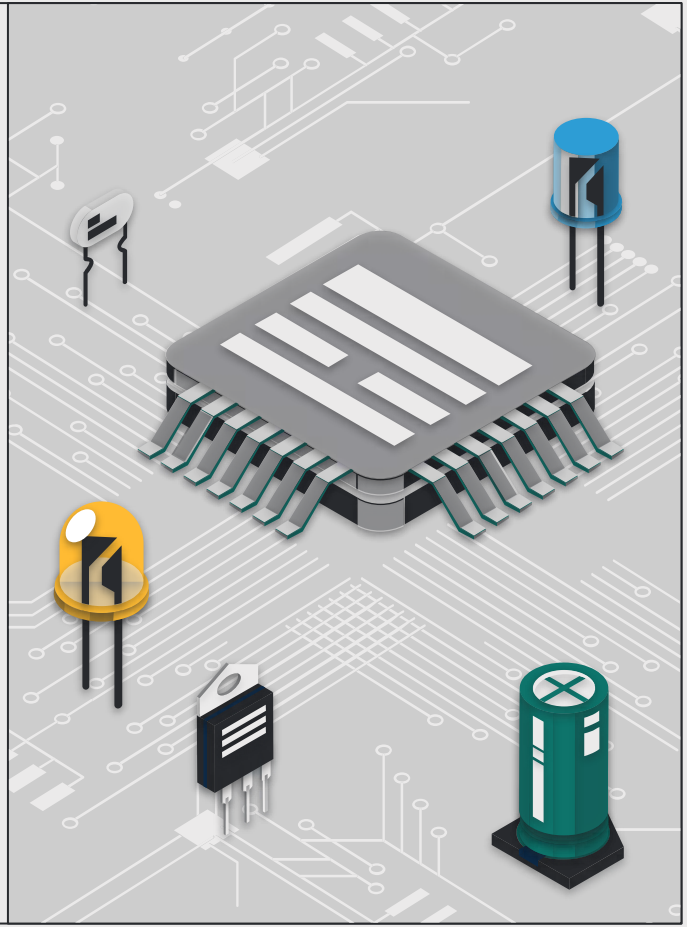
Data memory

```
module Data_memory (
    input clk,MemWrite,
    input [31:2] Address,
    input [31:0]Write_data,
    input [1:0] sel,byte_addr,
    output reg [31:0] Read_data
);
    reg [31:0] mem [127:0];
    always @ (posedge clk)
    begin
    if (MemWrite)
        begin
        case (sel)
        2'b00:mem [Address]=Write_data;//sw
        2'b01:
        begin
        case (byte_addr [1])
        1'b0:mem [Address][31:16]=Write_data[15:0];//sh
        1'b1:mem [Address][15:0]=Write_data[15:0];//sh
        default : ;
        endcase
        end
```



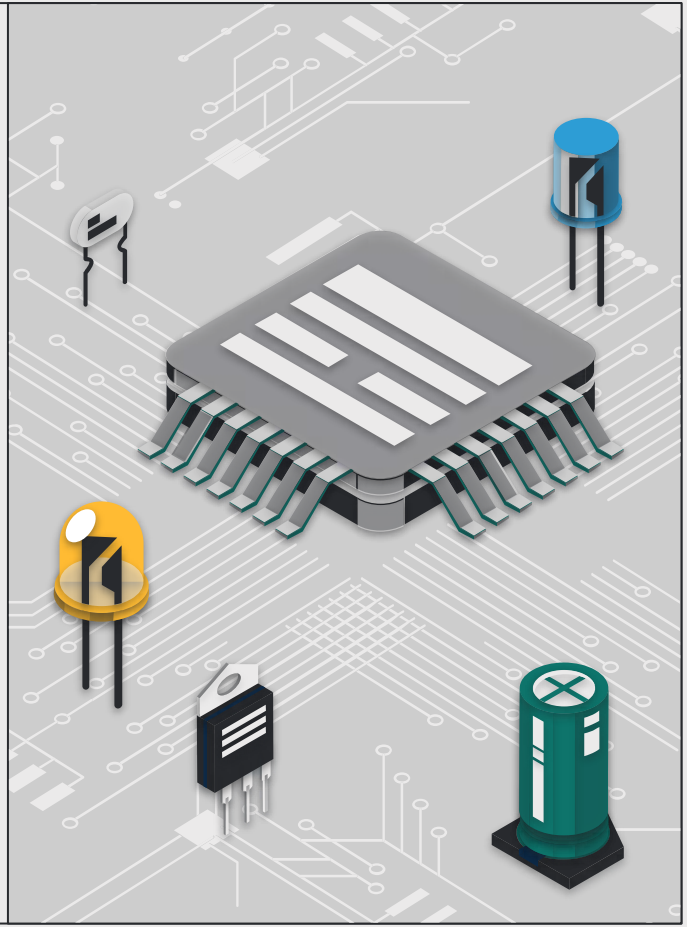
Data memory

```
        2'b10:
        begin
        case (byte_addr)//////////////////////////
        2'b00:mem [Address][31:24]=Write_data[7:0];//sb
        2'b01:mem [Address][23:16]=Write_data[7:0];//sb
        2'b10:mem [Address][15:8]=Write_data[7:0];//sb
        2'b11:mem [Address][7:0]=Write_data[7:0];//sb
        endcase
        end
        2'b11://sbit
        begin
        case (byte_addr)//////////////////////////
        2'b00:mem [Address][24]=Write_data[0];//sbit
        2'b01:mem [Address][16]=Write_data[0];//sbit
        2'b10:mem [Address][8]=Write_data[0];//sbit
        2'b11:mem [Address][0]=Write_data[0];//sbit
        endcase
        end
        default :;
        endcase
    end
```



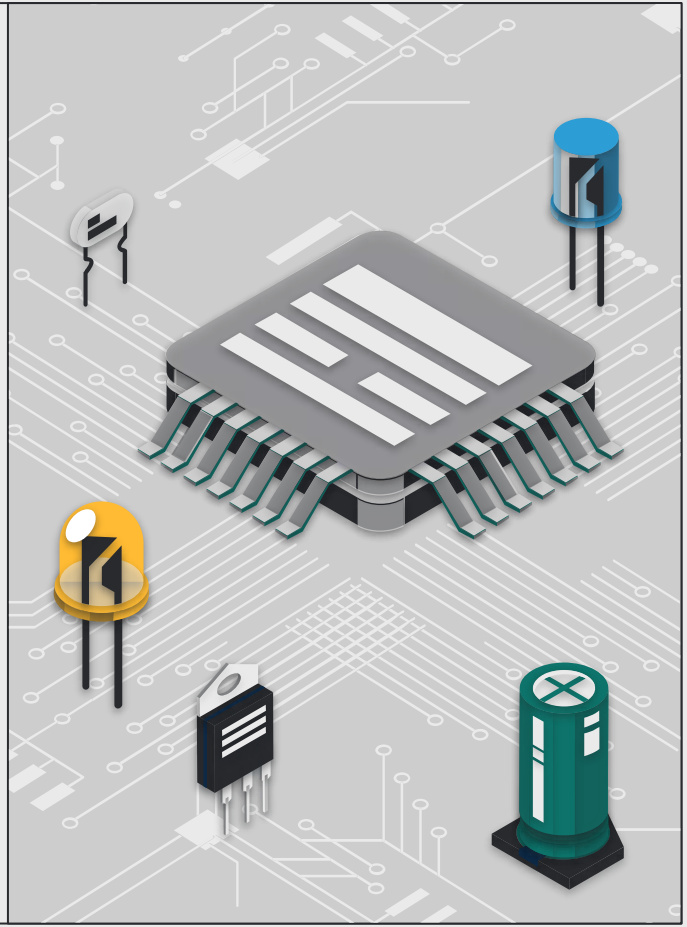
Data memory

```
else;
end
always @ (*)
begin
    case (sel)
    2'b00:Read_data=mem[Address];//lw
    2'b01:
    begin
    case (byte_addr [1])
    1'b0:Read_data={{16{mem[Address][31]}},mem[Address][31:16]};//lh
    1'b1:Read_data={{16{mem[Address][15]}},mem[Address][15:0]};//lh
    default : ;
    endcase
    end
    2'b10:
    begin
    case (byte_addr)//////////////////////////
    2'b00:Read_data={{24{mem[Address][31]}},mem[Address][31:24]};//lb
    2'b01:Read_data={{24{mem[Address][23]}},mem[Address][23:16]};//lb
    2'b10:Read_data={{24{mem[Address][15]}},mem[Address][15:8]};//lb
    2'b11:Read_data={{24{mem[Address][7]}},mem[Address][7:0]};//lb
    endcase
```



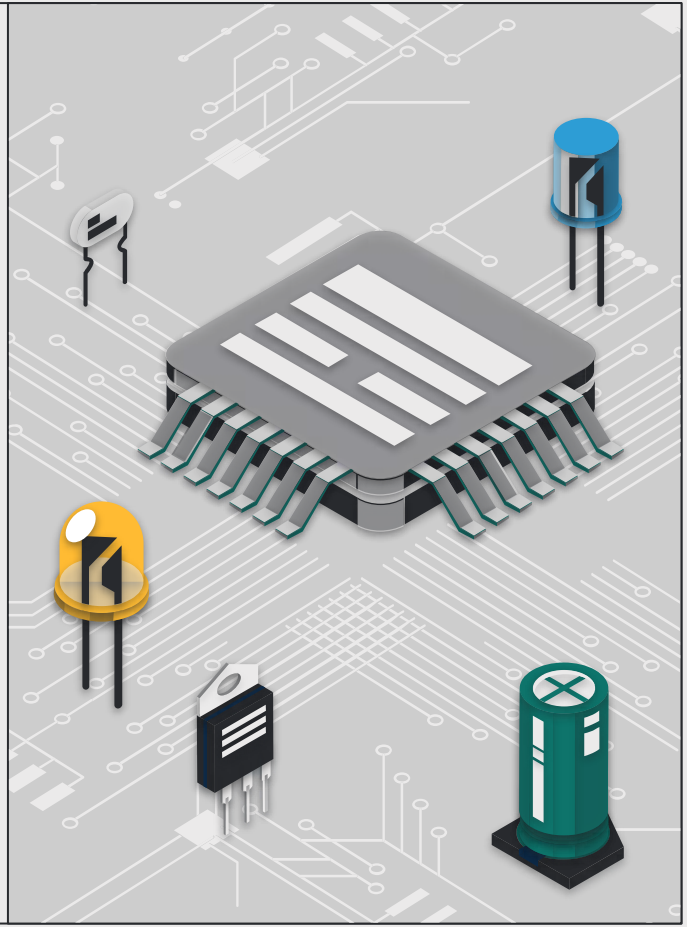
Data memory

```
    end
    2'b11:
    begin
    case (byte_addr)//////////////////////
    2'b00:Read_data={{31{mem[Address][31]}},mem[Address][24]};//1bit
    2'b01:Read_data={{31{mem[Address][23]}},mem[Address][16]};//1bit
    2'b10:Read_data={{31{mem[Address][15]}},mem[Address][8]};//1bit
    2'b11:Read_data={{31{mem[Address][7]}},mem[Address][0]};//1bit
    endcase
    end
    default :;
    endcase
end
initial
begin
mem [0]<=32'h0af00005;
mem [1]<=32'h0ba00007;
mem [2]<=32'h0ca0000C;
end
endmodule
```



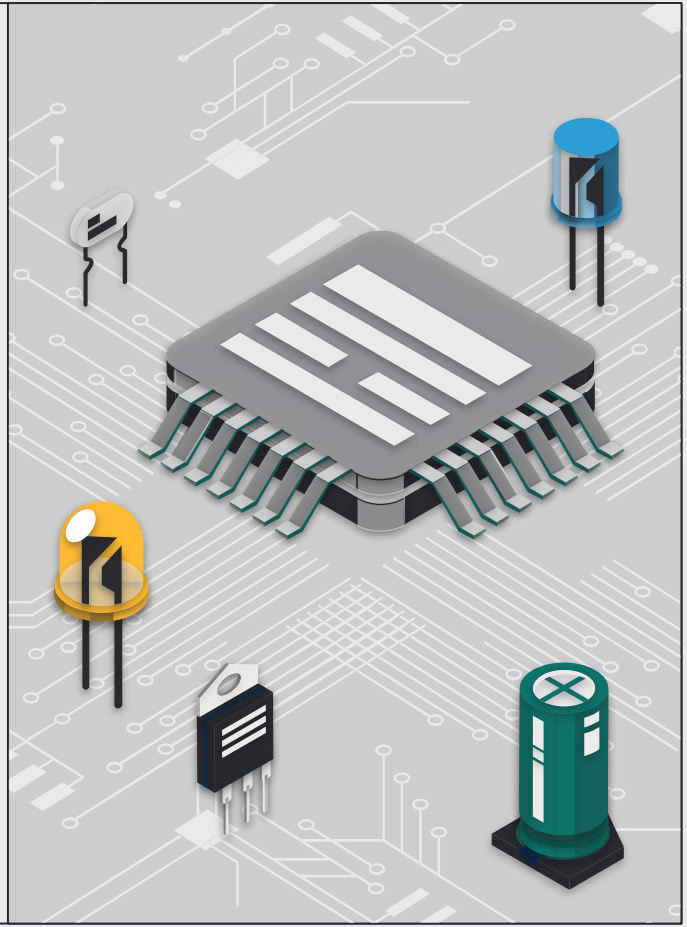
PC

```
module PC (
    input clk,
    input [31:0] IN,
    output reg [31:0] OUT
);
    initial
    OUT=0;
    always @ (posedge clk)
        OUT=IN;
endmodule
```



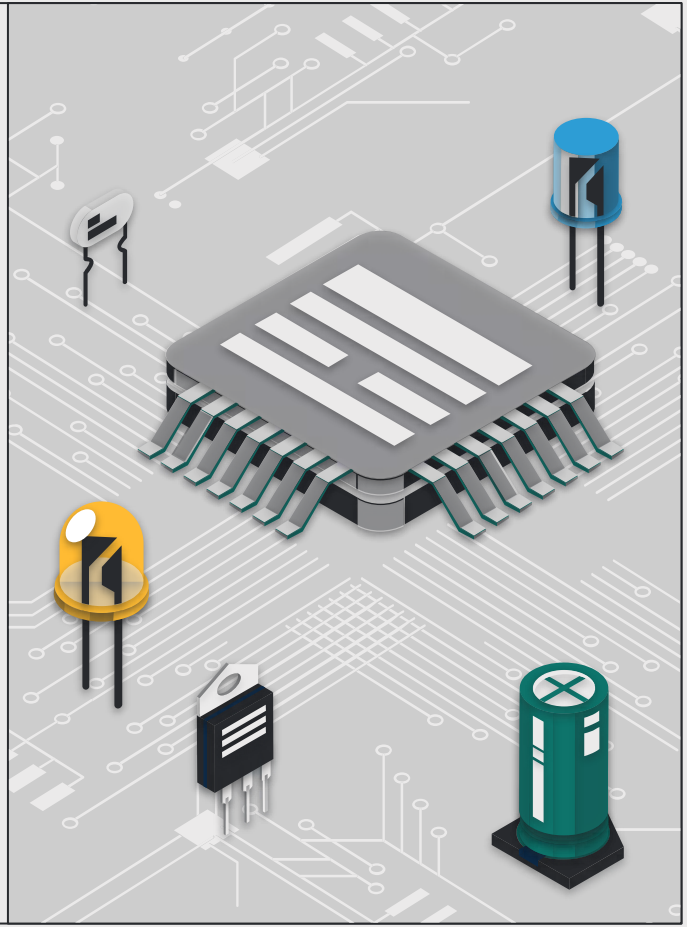
MUX 4:1

```
module MUX_4_1 #(parameter N = 32)(
    input [N-1:0] A,B,C,D,
    input [1:0]sel,
    output reg [N-1:0] Z
);
always @ (*)
begin
    case (sel)
    2'b00:Z=A;
    2'b01:Z=B;
    2'b10:Z=C;
    2'b11:Z=D;
    default:Z=A;
    endcase
end
endmodule
```



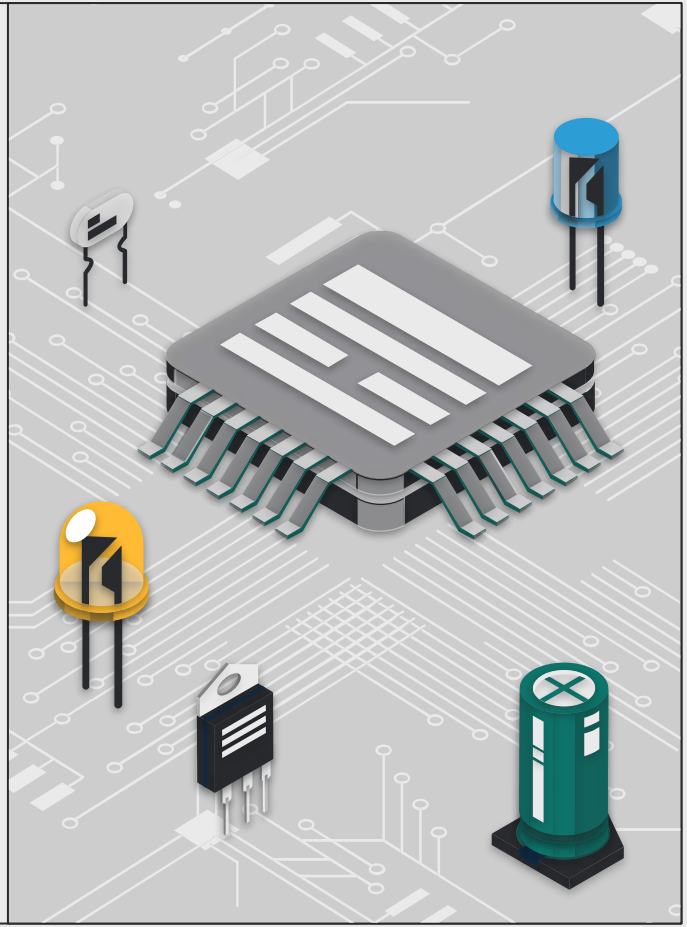
RCA

```
module RCA #(parameter n=32)(
    input [n-1:0] A,B,
    input Cin,
    output Cout,overflow,LessThan,
    output [n-1:0] S
);

    wire [n:0] C;
    assign C[0]=Cin;
    assign Cout=C[n];
    assign overflow = C[n]^C[n-1];
    assign LessThan = S[n-1]^overflow;
```



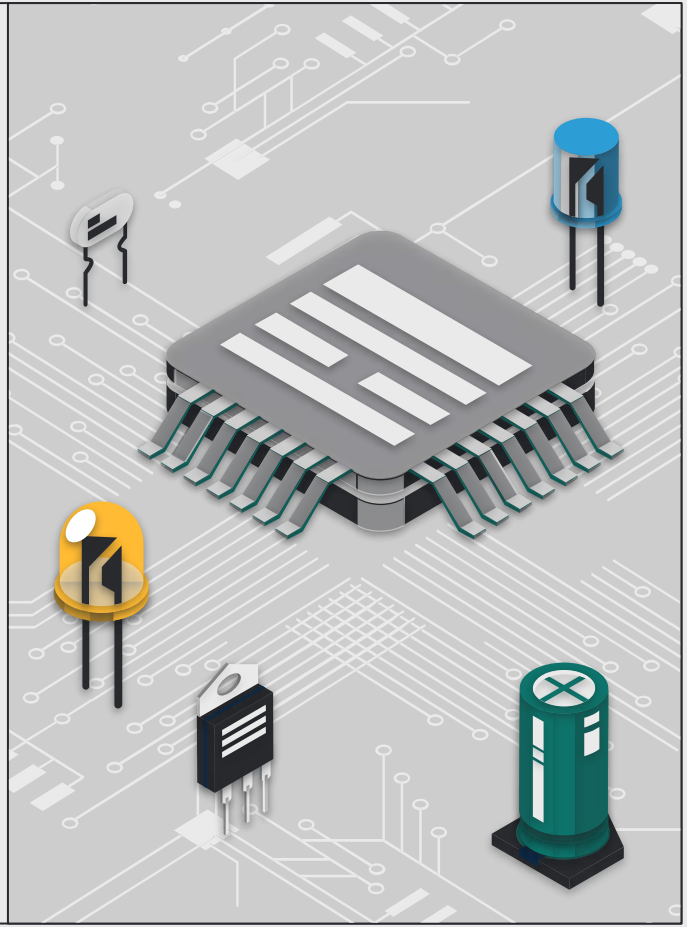
RCA

```
    generate
        genvar k;
        for (k=0;k<n;k=k+1)
        begin : FA
            FA FA(
                    .A(A[k]),
                    .B(B[k]),
                    .Cin(C[k]),
                    .Cout(C[k+1]),
                    .S(S[k])
                    );
        end
    endgenerate

endmodule
```



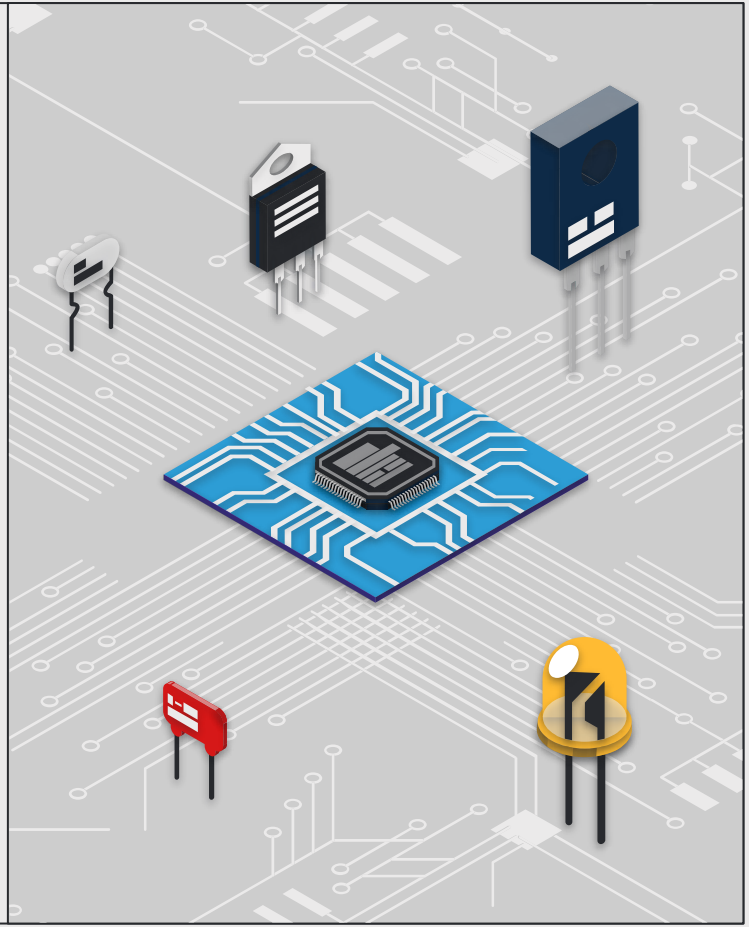
FA

```
module FA (
    input A,B,Cin,
    output Cout,S
);
    assign Cout = (A&Cin)|(B&Cin)|(A&B);
    assign S = A^B^Cin;
endmodule
```



03

Verilog testbench



Address	Code	Basic		Source
0x00000000	0x8c080000	lw $8,0x00000000($0)	1	lw $t0, 0($0) # lw
0x00000004	0x21090005	addi $9,$8,0x00000005	2	addi $t1, $t0, 5 # addi
0x00000008	0x00095080	sll $10,$9,0x00000002	3	sll $t2, $t1, 2 # sll
0x0000000c	0x3c0b1000	lui $11,0x00001000	4	lui $t3, 0x1000 # lui
0x00000010	0x016a6026	xor $12,$11,$10	5	xor $t4, $t3, $t2 # xor
0x00000014	0x800d0004	lb $13,0x00000004($0)	6	lb $t5, 4($0) # lb
0x00000018	0x840e0006	lh $14,0x00000006($0)	7	lh $t6, 6($0) # lh
0x0000001c	0x01ae782a	slt $15,$13,$14	8	slt $t7, $t5, $t6 # slt
0x00000020	0x01aec024	and $24,$13,$14	9	and $t8, $t5, $t6 # and
0x00000024	0x29b9000a	slti $25,$13,0x0000000	10	slti $t9, $t5, 10 # slti
0x00000028	0x31b000ff	andi $16,$13,0x000000f	11	andi $s0, $t5, 0xFF # andi
0x0000002c	0x000d8883	sra $17,$13,0x00000002	12	sra $s1, $t5, 2 # sra
0x00000030	0x11ae0006	beq $13,$14,0x00000006	13	beq $t5, $t6, label1 # beq
0x00000034	0x01ae9027	nor $18,$13,$14	14	nor $s2, $t5, $t6 # nor
0x00000038	0x000e98c2	srl $19,$14,0x00000003	15	srl $s3, $t6, 3 # srl
0x0000003c	0x15e00005	bne $15,$0,0x00000005	16	bne $t7, $0, label2 # bne
0x00000040	0x01aea025	or $20,$13,$14	17	or $s4, $t5, $t6 # or

Address	Code	Basic	Source
0x00000044	0x01cda822	sub $21,$14,$13	18 sub $s5, $t6, $t5 # sub
0x00000048	0x08000017	j 0x0040005c	19 j label3 # j
0x0000004c	0x20160001	addi $22,$0,0x00000001	22 addi $s6, $0, 1 # addi (example)
0x00000050	0x08000018	j 0x00400060	23 j label4 # j
0x00000054	0x20160002	addi $22,$0,0x00000002	26 addi $s6, $0, 2 # addi (example)
0x00000058	0x08000018	j 0x00400060	27 j label4 # j
0x0000005c	0x20160003	addi $22,$0,0x00000003	30 addi $s6, $0, 3 # addi (example)
0x00000060	0x35b70055	ori $23,$13,0x00000055	33 ori $s7, $t5, 0x55 # ori
0x00000064	0xac170008	sw $23,0x00000008($0)	34 sw $s7, 8($0) # sw
0x00000068	0x0c00001e	jal 0x00400078	35 jal subroutine # jal
0x0000006c	0xa00d000a	sb $13,0x0000000a($0)	36 sb $t5, 10($0) # sb
0x00000070	0xa40e000c	sh $14,0x0000000c($0)	37 sh $t6, 12($0) # sh
0x00000074	0x39b000ff	xori $16,$13,0x000000f	38 xori $s0, $t5, 0xFF # xori
0x00000078	0x02118020	add $16,$16,$17	41 add $s0, $s0, $s1 # add
0x0000007c	0x03e00008	jr $31	42 jr $ra # jr

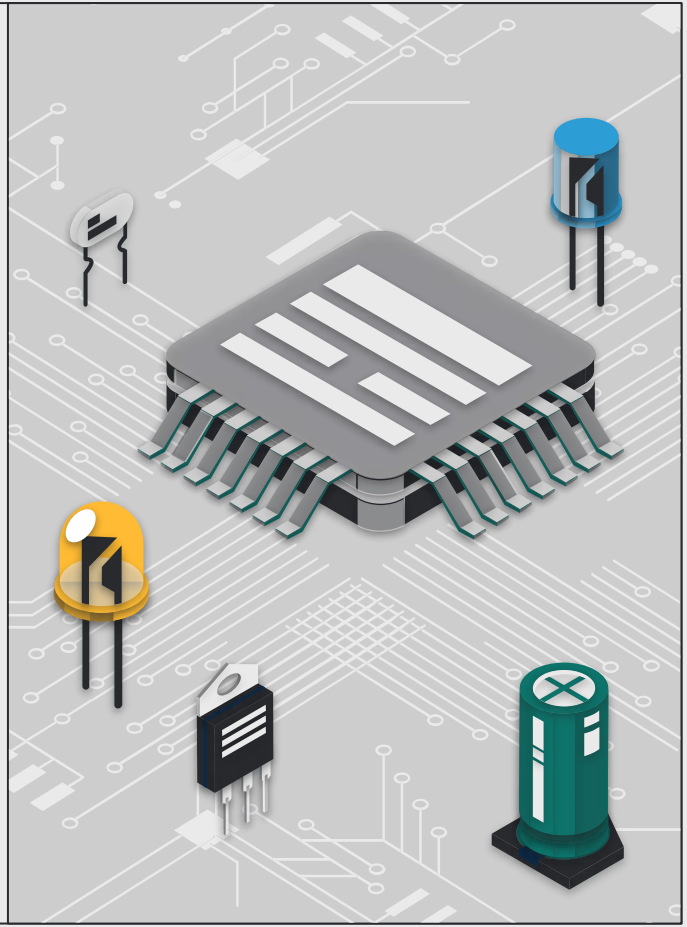
Test code

```
lw $t0, 0($0)           # lw
addi $t1, $t0, 5        # addi
sll $t2, $t1, 2         # sll
lui $t3, 0x1000         # lui
xor $t4, $t3, $t2       # xor
lb $t5, 4($0)           # lb
lh $t6, 6($0)           # lh
slt $t7, $t5, $t6       # slt
and $t8, $t5, $t6       # and
slti $t9, $t5, 10       # slti
andi $s0, $t5, 0xFF     # andi
sra $s1, $t5, 2         # sra
beq $t5, $t6, label1    # beq
nor $s2, $t5, $t6       # nor
srl $s3, $t6, 3         # srl
bne $t7, $0, label2     # bne
or $s4, $t5, $t6        # or
sub $s5, $t6, $t5       # sub
j label3                # j
```

```
label1:
    addi $s6, $0, 1     # addi (example)
    j label4             # j

label2:
    addi $s6, $0, 2     # addi (example)
    j label4             # j

label3:
    addi $s6, $0, 3     # addi (example)

label4:
ori $s7, $t5, 0x55      # ori
sw $s7, 8($0)           # sw
jal subroutine           # jal
sb $t5, 10($0)          # sb
sh $t6, 12($0)          # sh
xori $s0, $t5, 0xFF     # xori

subroutine:
    add $s0, $s0, $s1   # add
    jr $ra               # jr
```

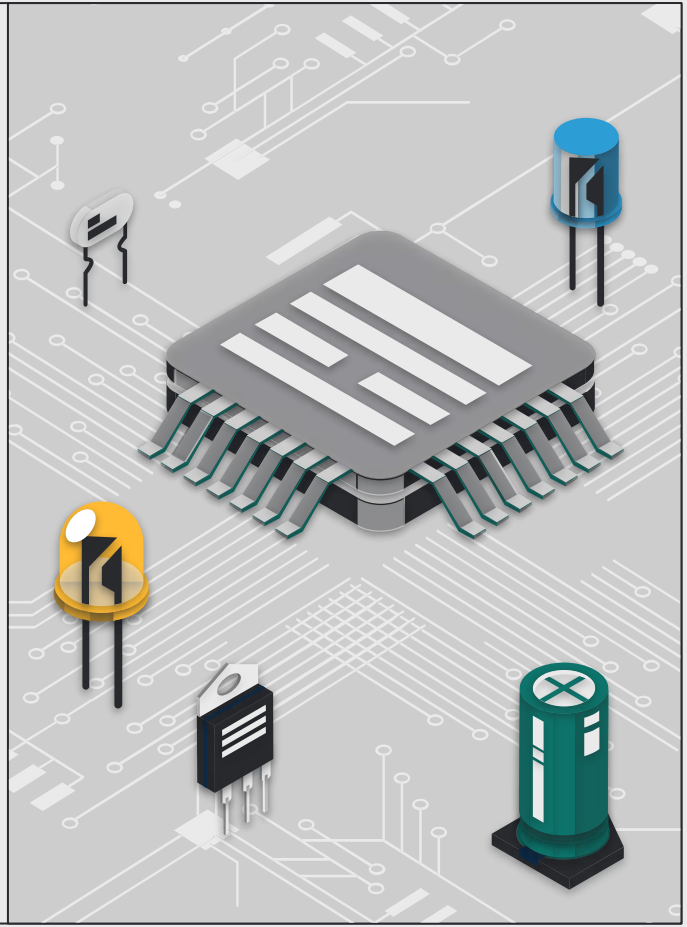
MIPS_processor_tb

```
`timescale 1ns/1ps
module MIPS_processor_tb();
    reg clk=0;
    wire negative,Zero,Cout,overflow;
    parameter clk_prd=1000000000;
    always # (clk_prd/2) clk=~clk;

MIPS_processor DUT (
    .clk(clk),
    .negative(negative),
    .Zero(Zero),
    .Cout(Cout),
    .overflow(overflow)
);
endmodule
```



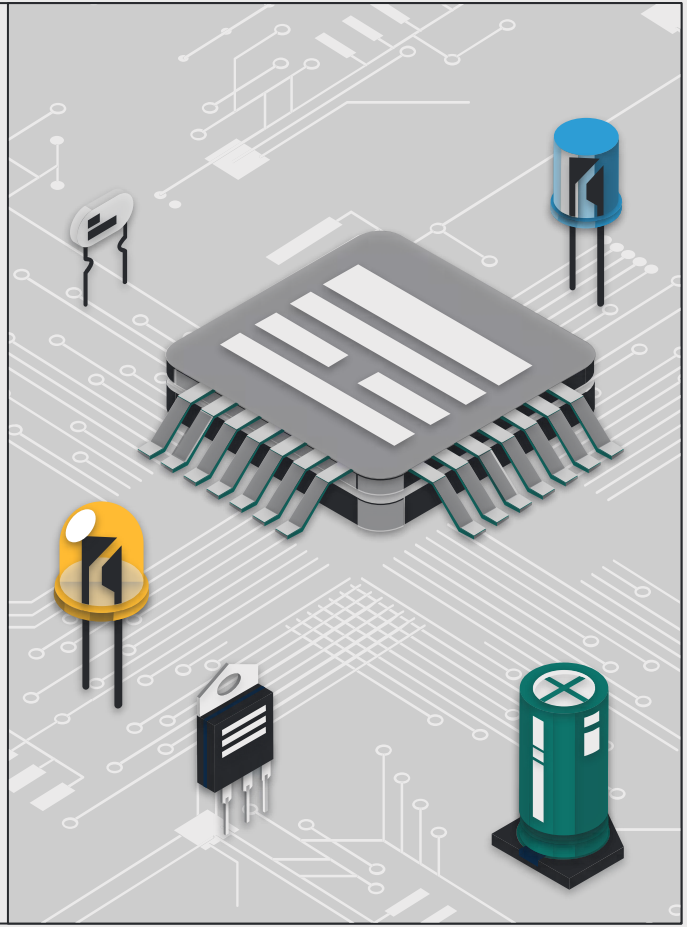
MIPS_processor_tb

```
`timescale 1ns/1ps
module MIPS_processor_tb();
    reg clk=0;
    wire negative,Zero,Cout,overflow;
    parameter clk_prd=1000000000;
    always # (clk_prd/2) clk=~clk;

MIPS_processor DUT (
    .clk(clk),
    .negative(negative),
    .Zero(Zero),
    .Cout(Cout),
    .overflow(overflow)
);
endmodule
```



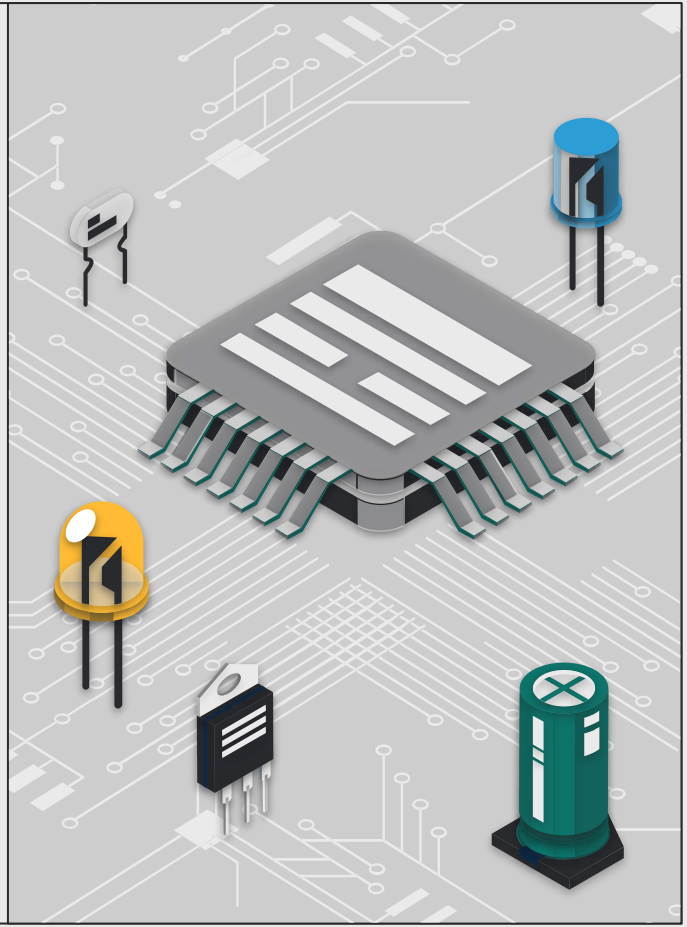
ALU tb

```
`timescale 1ns/1ps
module ALU_tb ();
    parameter n=32;
    reg [3:0] sel;
    reg [n-1:0] A,B;
    reg [10:6] Shamt;
    wire [n-1:0] OUT;
    wire negative,Zero,Cout,overflow;
ALU #(.n(n))DUT(
    .sel(sel),
    .A(A),
    .B(B),
    .Shamt(Shamt),
    .OUT(OUT),
    .negative(negative),
    .Zero(Zero),
    .Cout(Cout),
    .overflow(overflow)
);
```



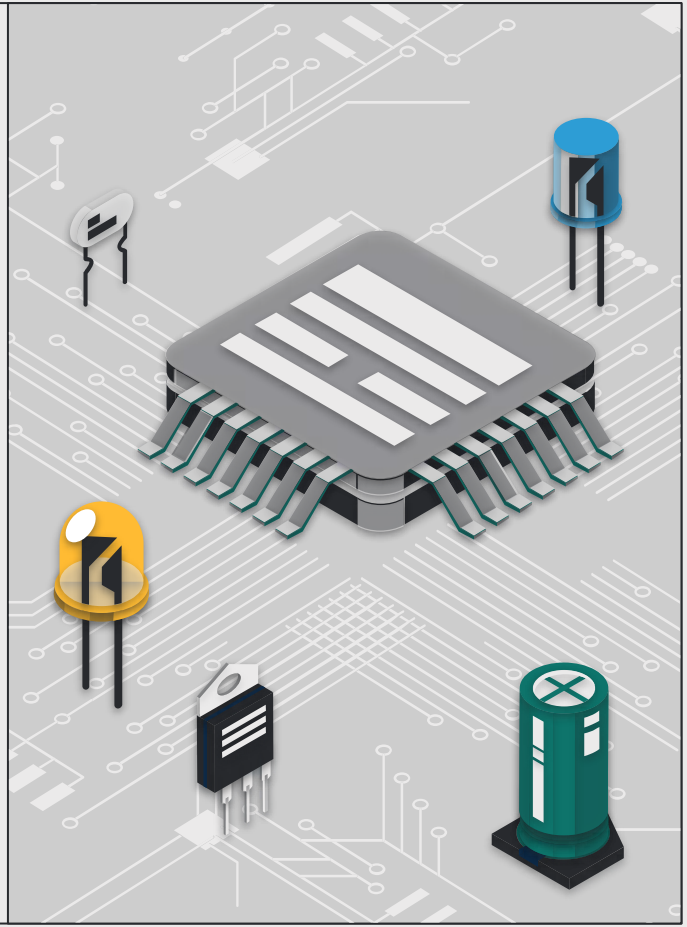
ALU tb

```
initial
begin
    A=32'd59;
    B=32'd77;
    sel=4'b00x0;
    #10;
    A=32'd528;
    B=32'd456;
    sel=4'b00x0;
    #10;
    A=32'd528;
    B=32'd456;
    sel=4'b10x0;
    #10;
    A=32'd528;
    B=32'd456;
    sel=4'b11x0;
    #10;
    A=32'd105;
    B=32'd552;
    sel=4'b11x0;
    #10;
```



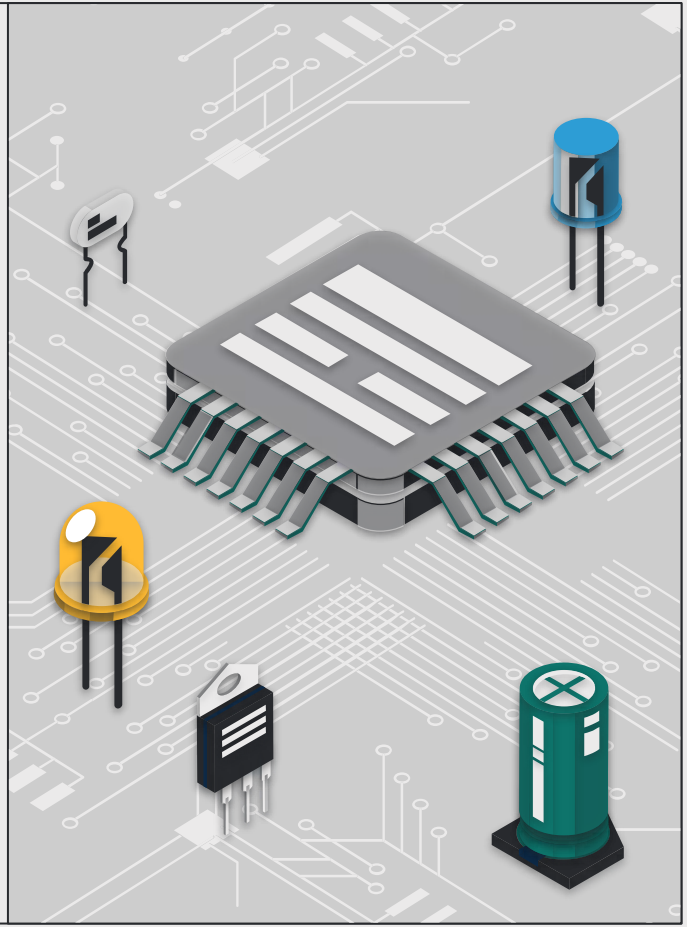
ALU tb

```
    //
    A=4152;
    B=1553;
    sel=4'b0001;
    #10;
    A=4152;
    B=1553;
    sel=4'b0101;
    #10;
    A=4152;
    B=1553;
    sel=4'b1001;
    #10;
    A=4152;
    B=1553;
    sel=4'b1101;
    #10;
    //
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=4'b0011;
```



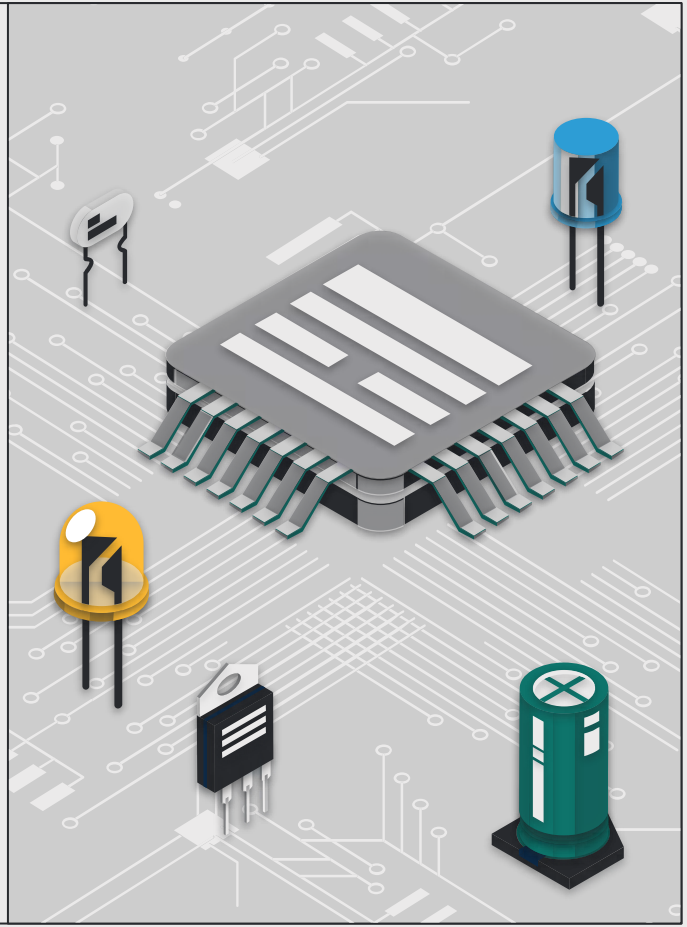
ALU tb

```
    #10;
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=4'b0111;
    #10;
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=4'b1011;
    #10;
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=4'b1111;
    #10;
    A=32'hffaa;
    B=$signed (32'hffffbbcc);
    Shamt=5'd6;
    sel=4'b1111;
    #10;
```



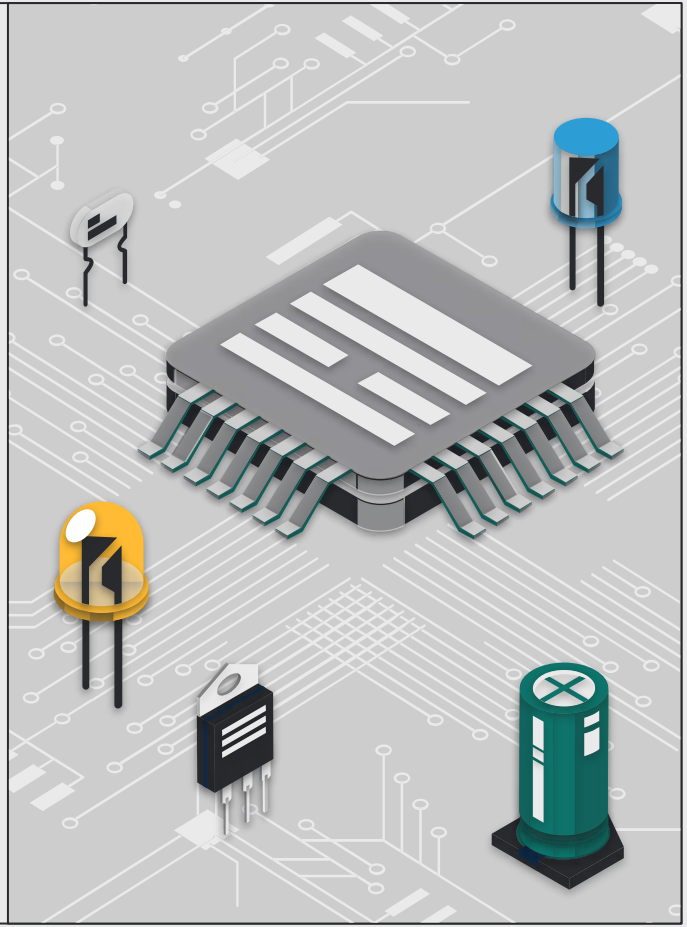
ALU tb

```
    A=32'h80000000;
    B=32'h80000000;
    sel=4'b00x0;
    #10;
    A=32'hffffffff;
    B=32'hffffffff;
    sel=4'b10x0;
    #10;
end
endmodule
```



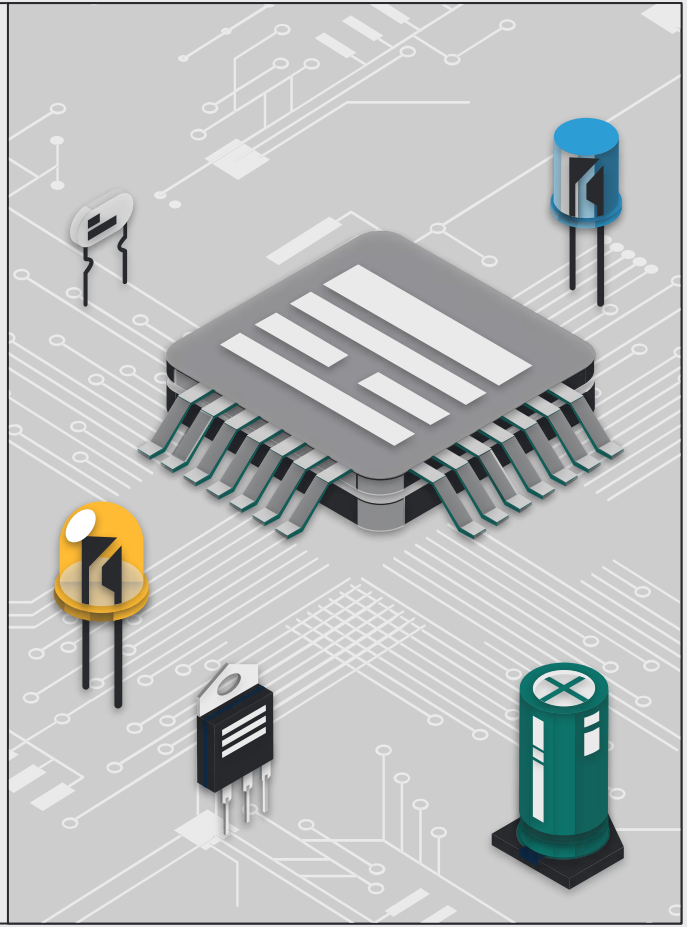
Arithmetic unit tb

```
`timescale 1ns/1ps
module Arithmetic_unit_tb ();
    parameter n=32;
    reg [n-1:0] A,B;
    reg add_n,add_slt;
    wire Cout,overflow;
    wire [n-1:0] OUT;
Arithmetic_unit #(.n(n)) DUT (
    .A(A),
    .B(B),
    .add_n(add_n),
    .add_slt(add_slt),
    .Cout(Cout),
    .overflow(overflow),
    .OUT(OUT)
);
```



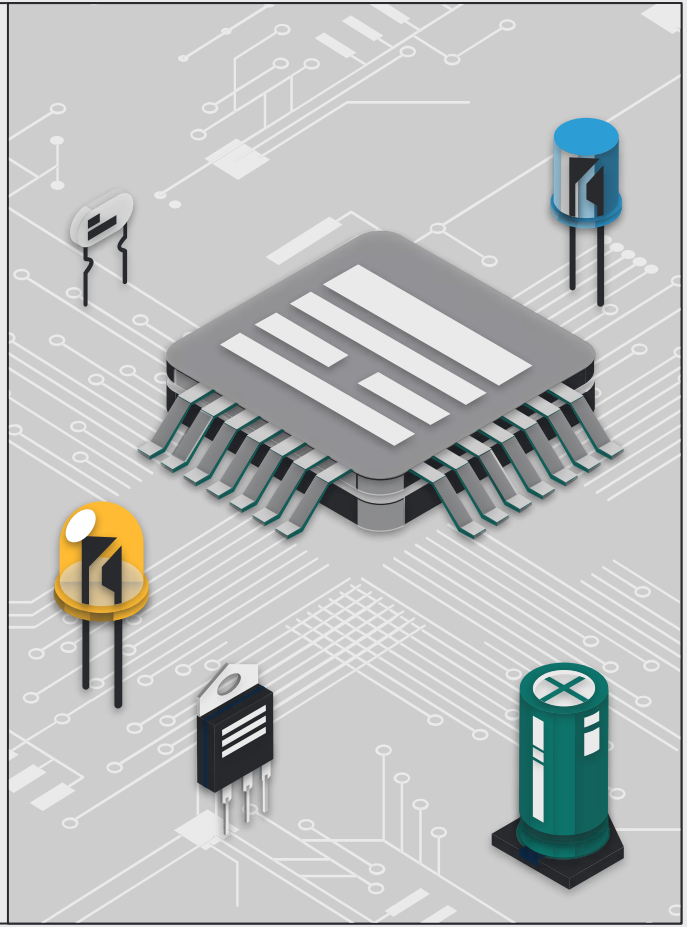
Arithmetic unit tb

```
initial
begin
A=32'd59;
B=32'd77;
add_n=1'b0;
add_slt=1'b0;
#10;
A=32'd528;
B=32'd456;
add_n=1'b0;
add_slt=1'b0;
#10;
A=32'd528;
B=32'd456;
add_n=1'b1;
add_slt=1'b0;
#10;
A=32'd528;
B=32'd456;
add_n=1'b1;
add_slt=1'b1;
```



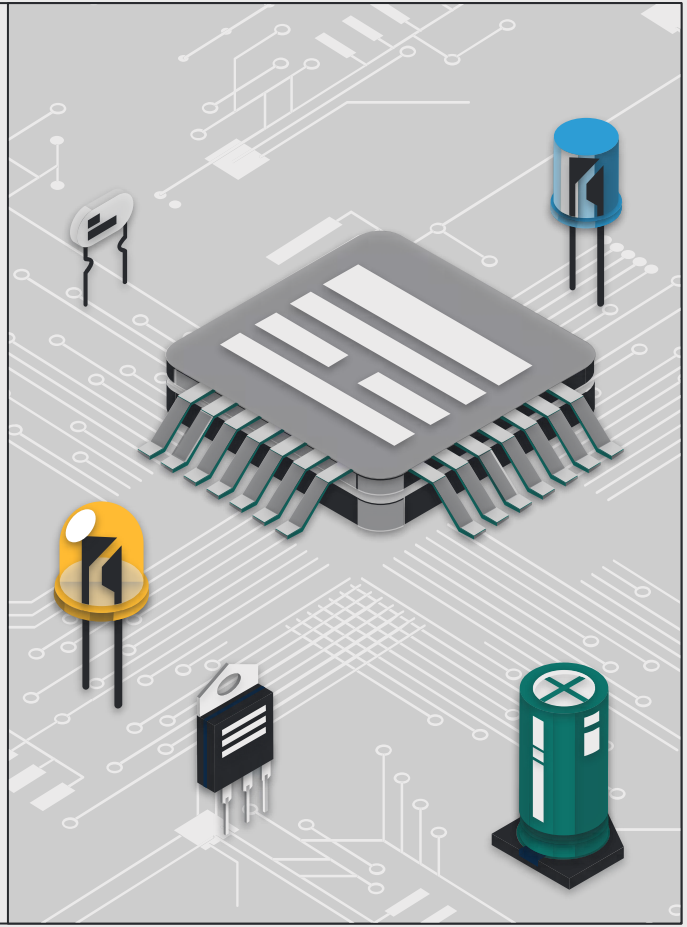
Arithmetic unit tb

```
    #10;
    A=32'd105;
    B=32'd552;
    add_n=1'b1;
    add_slt=1'b1;
    #10;
    end
endmodule
```



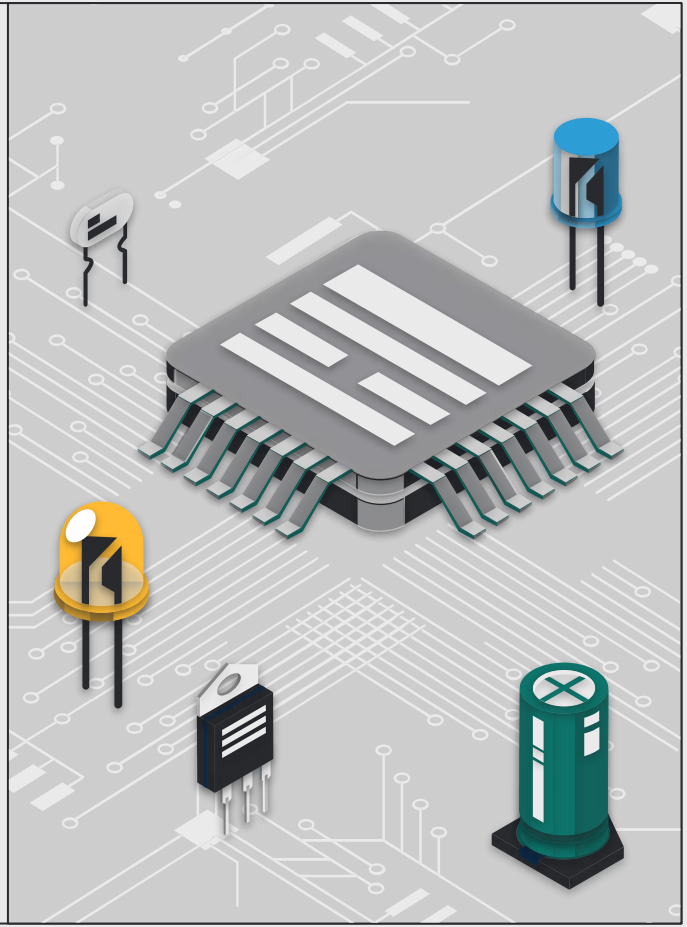
Logic unit tb

```
`timescale 1ns/1ps
module Logic_unit_tb ();
    parameter n = 32;
    reg [n-1:0] A,B;
    reg [1:0] sel;
    wire [n-1:0] OUT;
Logic_unit #(.n(n)) DUT (
    .A(A),
    .B(B),
    .sel(sel),
    .OUT(OUT)
);
    initial
    begin
    A=51;
    B=486;
    sel=2'b00;
    #10;
    A=51;
    B=486;
    sel=2'b01;
```



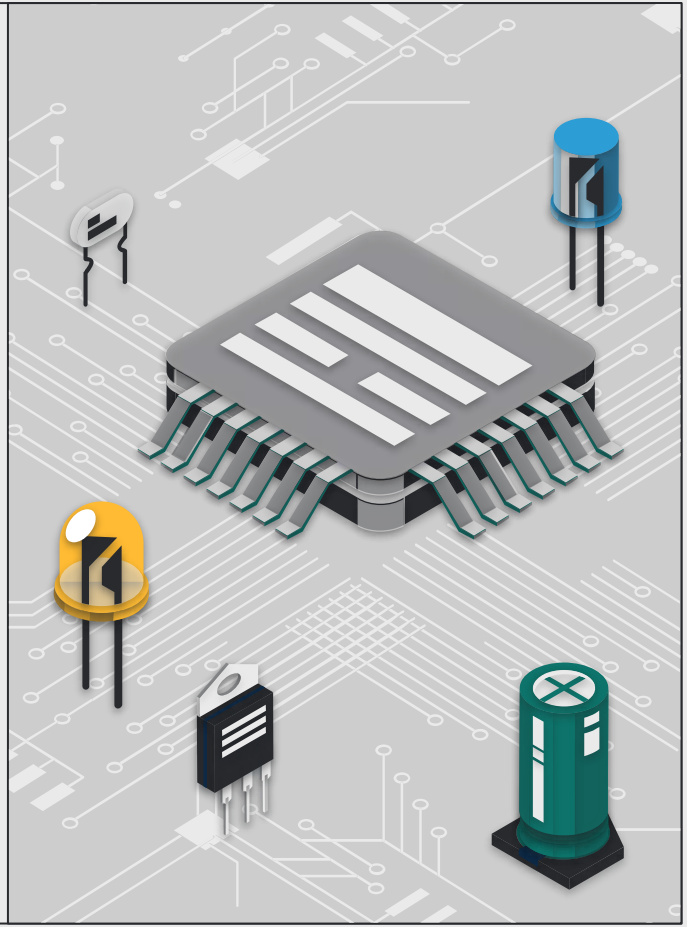
Logic unit tb

```
    #10;
    A=51;
    B=486;
    sel=2'b10;
    #10;
    A=51;
    B=486;
    sel=2'b11;
    #10;
    A=4152;
    B=1553;
    sel=2'b00;
    #10;
    A=4152;
    B=1553;
    sel=2'b01;
    #10;
    A=4152;
    B=1553;
    sel=2'b10;
```



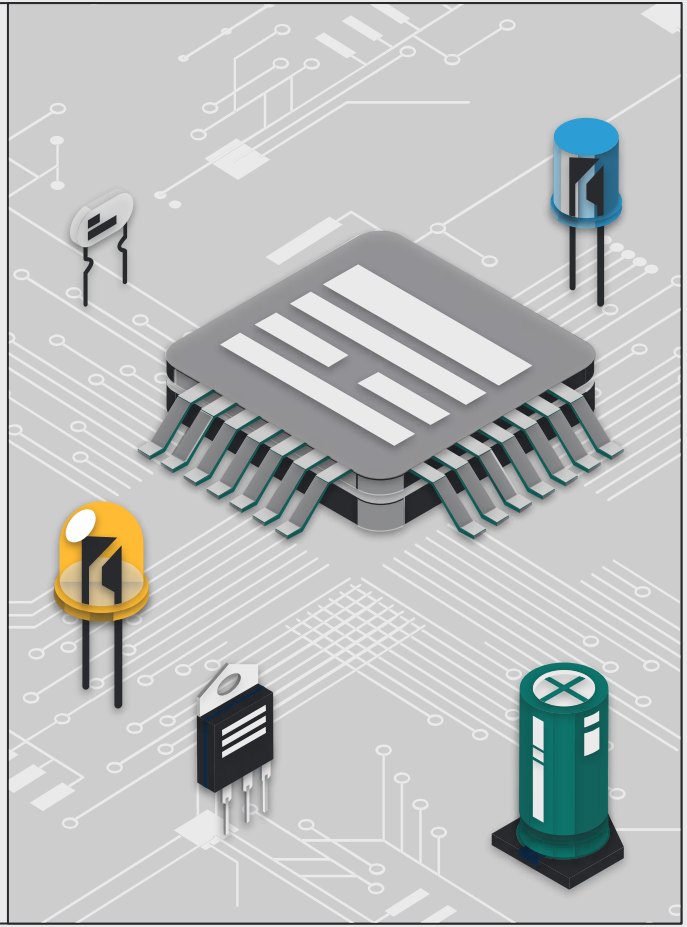
Logic unit tb

```
    #10;
    A=4152;
    B=1553;
    sel=2'b11;
    #10;
    end

endmodule
```



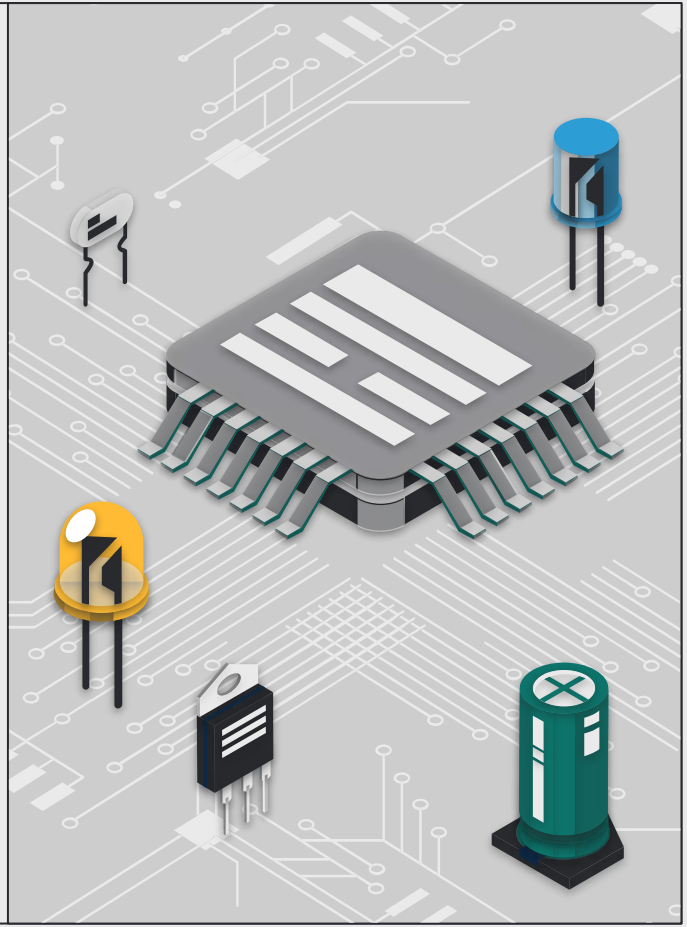
Shift lui unit tb

```
`timescale 1ns/1ps
module Shift_lui_unit_tb();
    parameter n=32;
    reg [n-1:0] A;
    reg signed [n-1:0] B;
    reg [10:6] Shamt;
    reg [1:0] sel;
    wire signed [n-1:0] OUT;
Shift_lui_unit #(.n(n)) DUT (
    .A(A),
    .B(B),
    .Shamt(Shamt),
    .sel(sel),
    .OUT(OUT)
);
    initial
    begin
    A=32'hfab;
    B=32'habc;
    Shamt=5'd6;
    sel=2'b00;
```



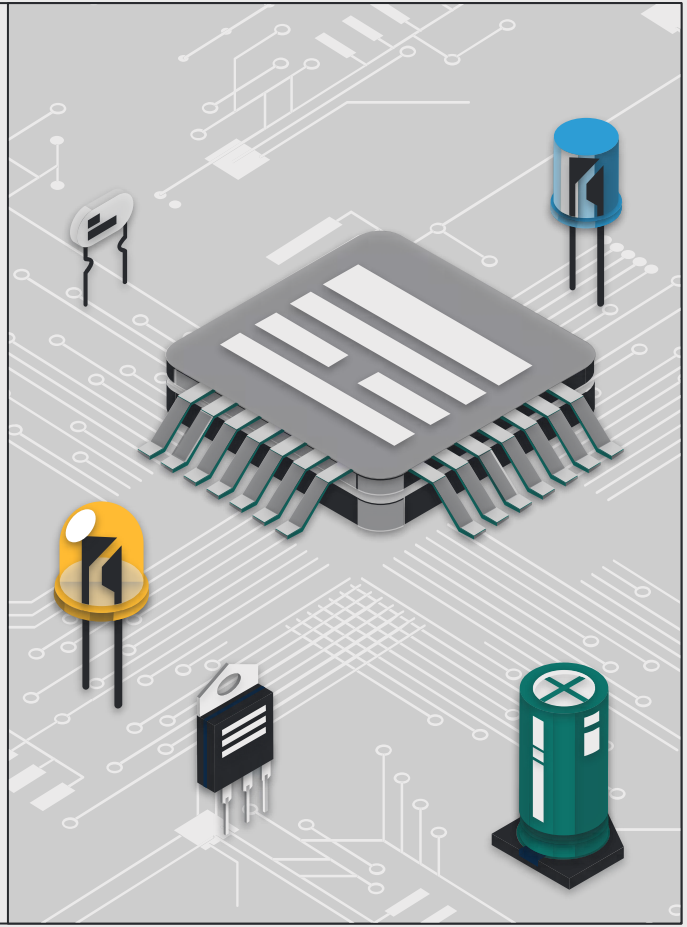
Shift lui unit tb

```
    #10;
    A=32'hfab;
    B=32'habc;
    Shamt=5'd6;
    sel=2'b01;
    #10;
    A=32'hfab;
    B=32'habc;
    Shamt=5'd6;
    sel=2'b10;
    #10;
    A=32'hfab;
    B=32'habc;
    Shamt=5'd6;
    sel=2'b11;
    #10;
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=2'b00;
    #10;
```



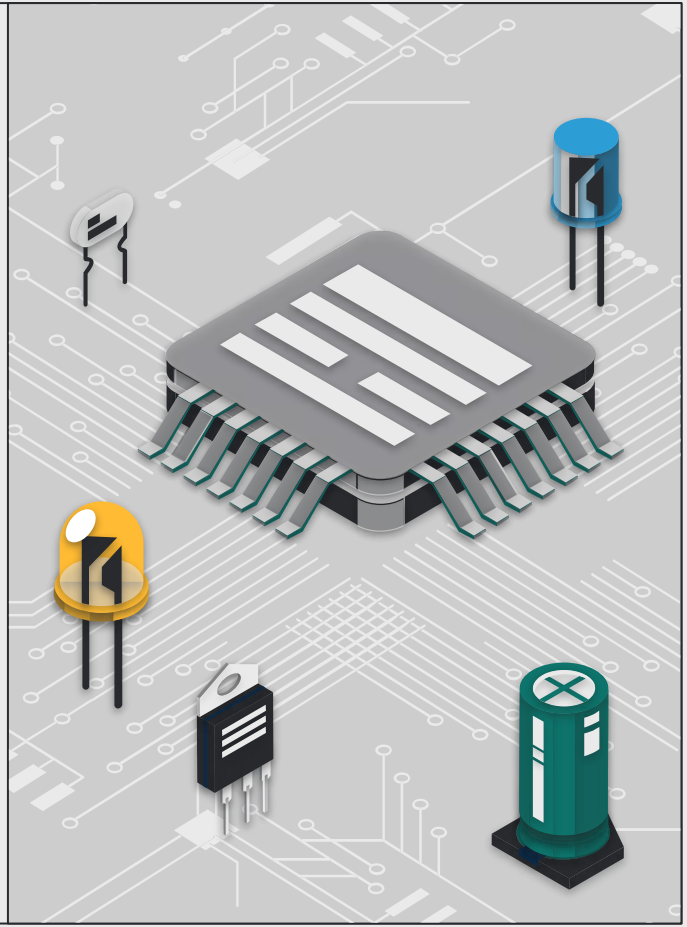
Shift lui unit tb

```
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=2'b01;
    #10;
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=2'b10;
    #10;
    A=32'hffaa;
    B=32'hbbcc;
    Shamt=5'd6;
    sel=2'b11;
    #10;
    A=32'hffaa;
    B=32'hffffbbcc;
    Shamt=5'd6;
    sel=2'b11;
    #10;
    end
endmodule
```



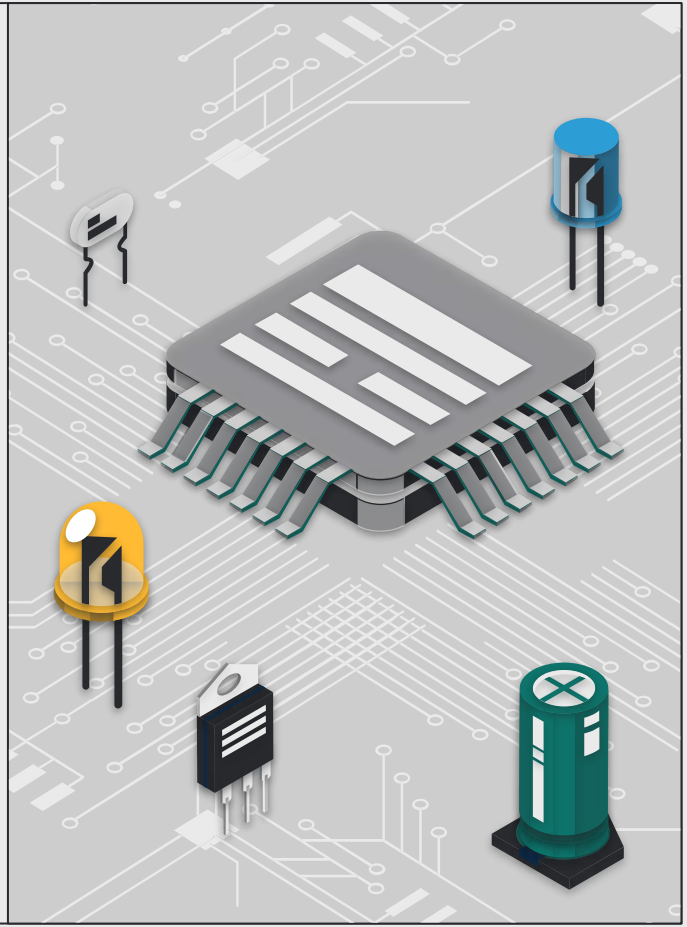
Adder tb

```
`timescale 1ns/1ps
module adder_tb ();
    parameter n=32;
    reg [n-1:0] A,B;
    wire [n-1:0] OUT;
adder #(.n(n)) DUT (
    .A(A),
    .B(B),
    .OUT(OUT)
);
initial
begin
    A=32'd4542;
    B=32'd5482;
    #10;
    A=32'd4253;
    B=32'd415;
    #10;
    A=32'd785;
    B=32'd5985;
```



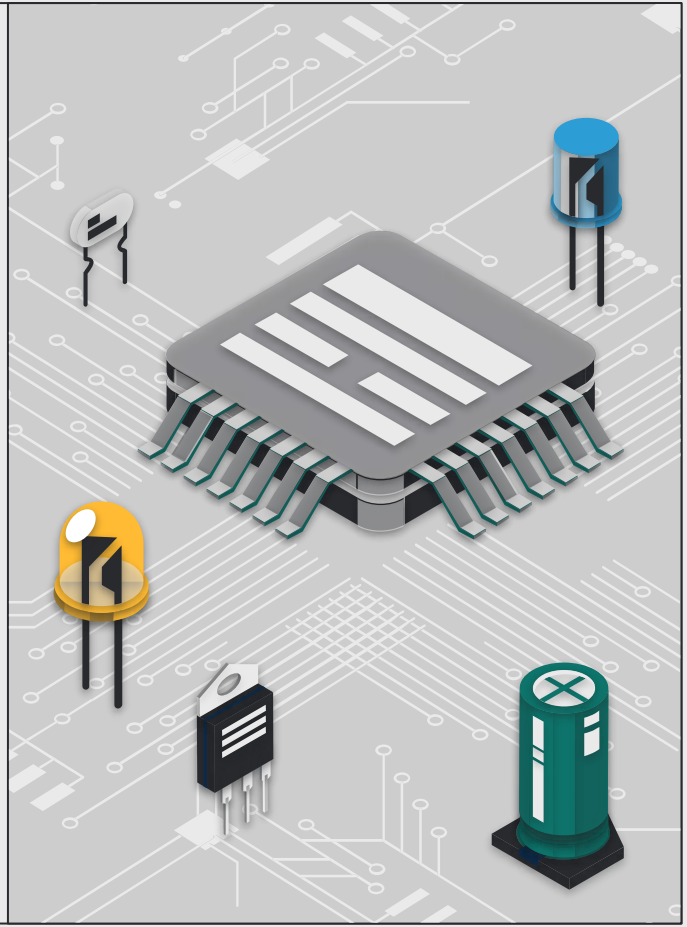
Adder tb

```
    #10;
    A=32'd1023;
    B=32'd248;
    #10;
    A=32'd759;
    B=32'd7458;
    #10;
    A=32'd42566;
    B=32'd15211;
    #10;
    A=32'd425;
    B=32'd258;
    #10;
    A=32'd5698;
    B=32'd7958;
    #10;
end
endmodule
```



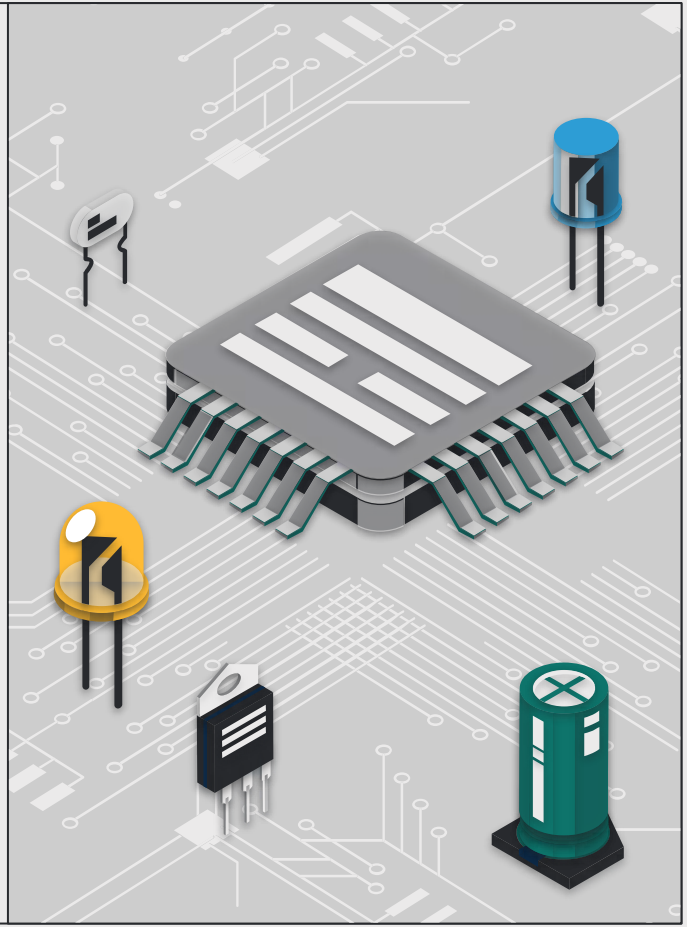
Add 4 tb

```
`timescale 1ns/1ps
module add_4_tb ();
    parameter n=32;
    reg [n-1:0] PC;
    wire [n-1:0] PC_4;
add_4 #(.n(n)) DUT (
    .PC(PC),
    .PC_4(PC_4)
);
initial
begin
    PC=32'd4542;
    #10;
    PC=32'd4253;
    #10;
    PC=32'd785;
    #10;
    PC=32'd1023;
    #10;
    PC=32'd759;
```



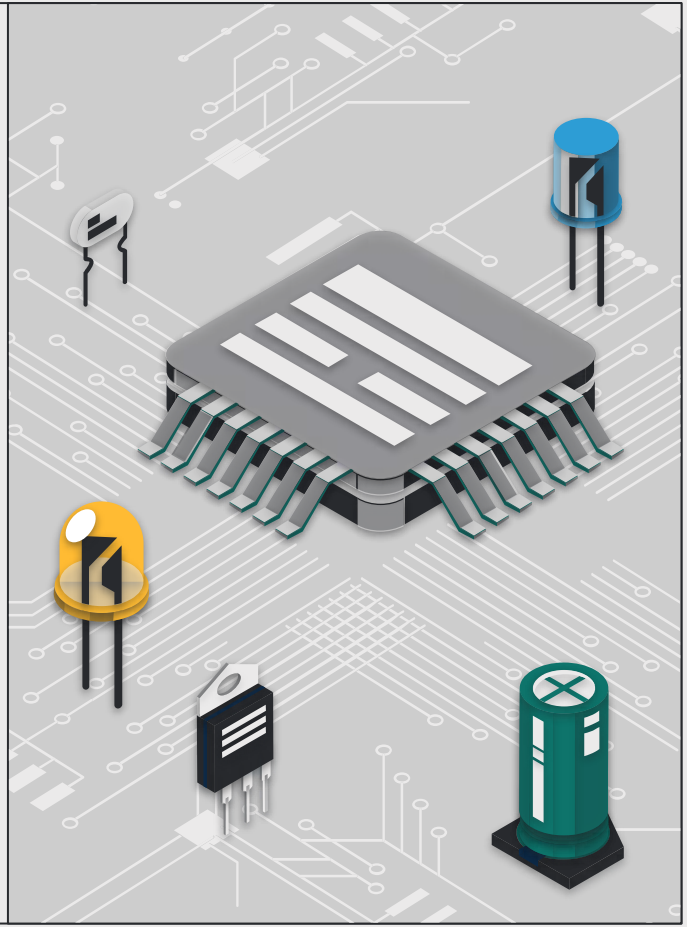
Add 4 tb

```
    #10;
    PC=32'd42566;
    #10;
    PC=32'd425;
    #10;
    PC=32'd5698;
    #10;
end
endmodule
```



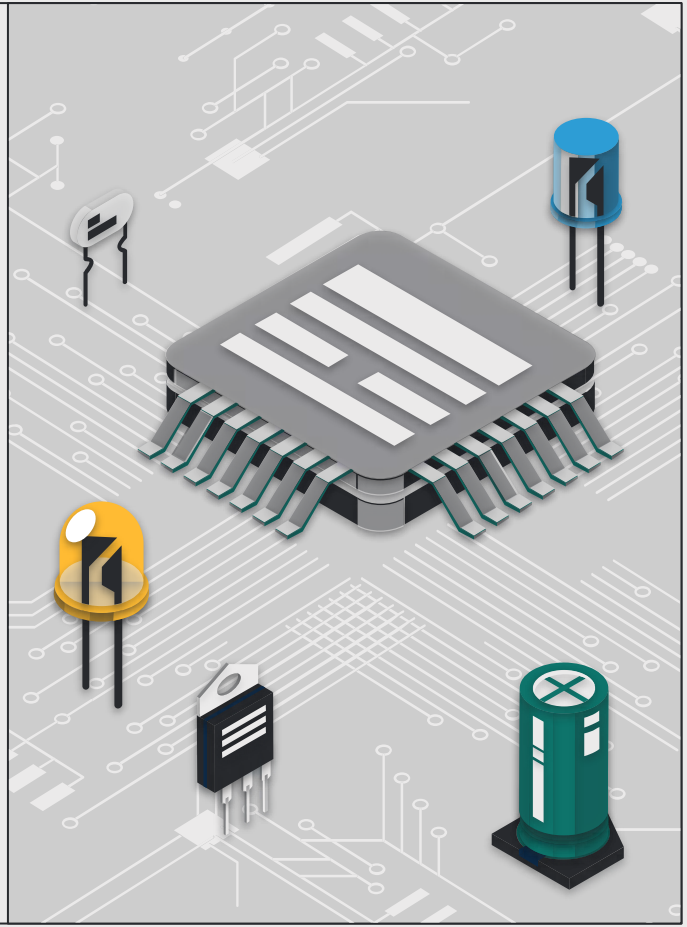
Control tb

```
`timescale 1ns/1ps
module Control_tb ();
    reg [31:26]OP_code;
    reg [5:0]Function_field;
    wire [1:0] RegDst_2,MemtoReg_2,Branch_2,Jump_2,SL_sel;
    wire ALUSrc,RegWrite,MemRead,MemWrite,Sign;
    wire [2:0] ALUOP;
Control DUT(
    .OP_code(OP_code),
    .Function_field(Function_field),
    .RegDst_2(RegDst_2),
    .MemtoReg_2(MemtoReg_2),
    .Branch_2(Branch_2),
    .Jump_2(Jump_2),
    .ALUSrc(ALUSrc),
    .RegWrite(RegWrite),
    .MemRead(MemRead),
    .MemWrite(MemWrite),
    .Sign(Sign),
    .ALUOP(ALUOP),
    .SL_sel(SL_sel)
);
```



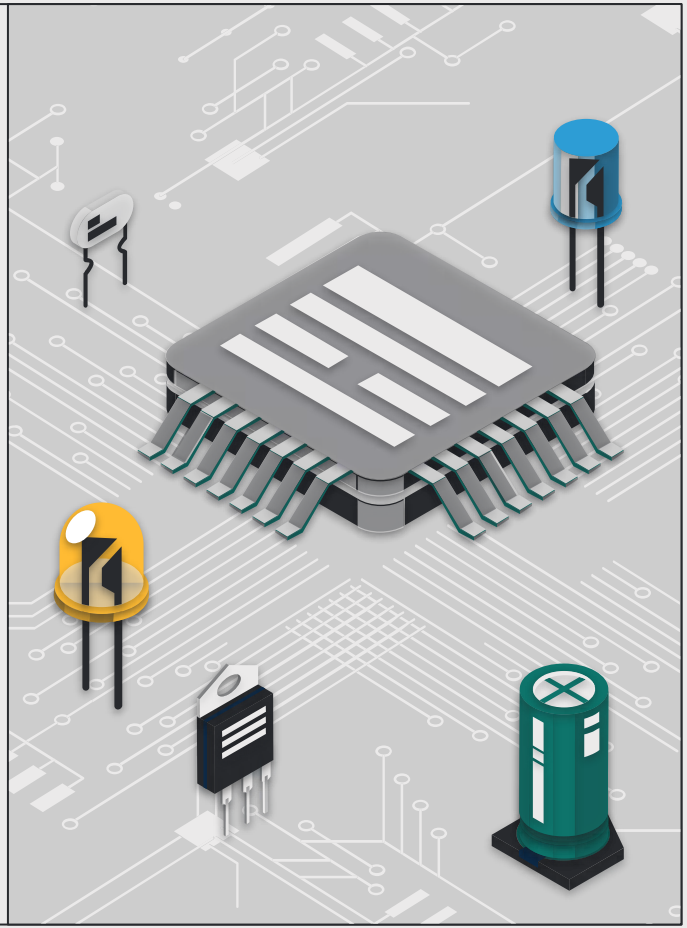
Control tb

```
initial
begin
    OP_code=6'd0;
    Function_field=6'd8;
    #10;
    OP_code=6'd0;
    Function_field=6'd32;
    #10;
    OP_code=6'd2;
    Function_field=6'dx;
    #10;
    OP_code=6'd3;
    Function_field=6'dx;
    #10;
    OP_code=6'd4;
    Function_field=6'dx;
    #10;
    OP_code=6'd5;
    Function_field=6'dx;
    #10;
    OP_code=6'd8;
    Function_field=6'dx;
```



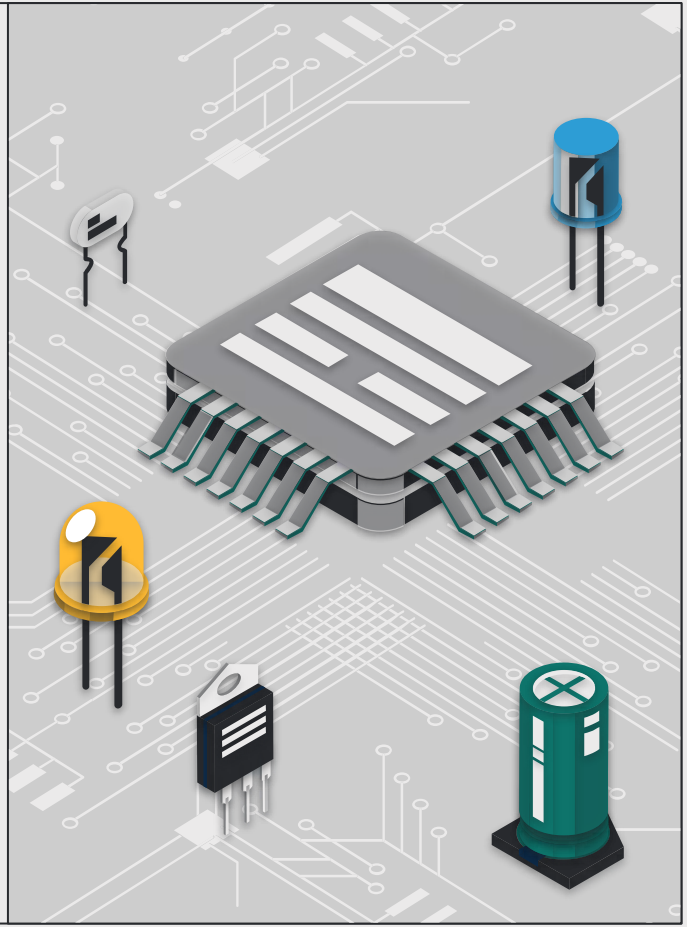
Control tb

```
        #10;
        OP_code=6'd10;
        Function_field=6'dx;
        #10;
        OP_code=6'd12;
        Function_field=6'dx;
        #10;
        OP_code=6'd13;
        Function_field=6'dx;
        #10;
        OP_code=6'd14;
        Function_field=6'dx;
        #10;
        OP_code=6'd15;
        Function_field=6'dx;
        #10;
        OP_code=6'd35;
        Function_field=6'dx;
        #10;
        OP_code=6'd43;
        Function_field=6'dx;
        #10;
    end
    endmodule
```



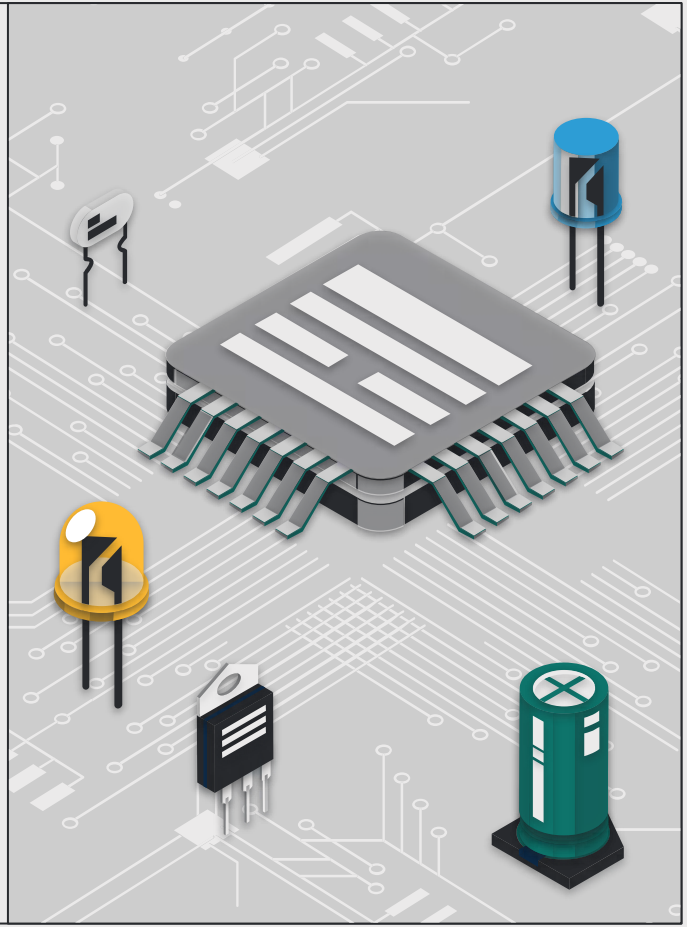
ALU control tb

```
`timescale 1ns/1ps
module ALU_control_tb ();
    reg [2:0]ALUOP;
    reg [5:0]Funct;
    wire [3:0]ALU_sel;

ALU_control DUT(
    .ALUOP(ALUOP),
    .Funct(Funct),
    .ALU_sel(ALU_sel)
);

initial
begin
    ALUOP=3'b000;
    Funct=6'bx;
    #10;
    ALUOP=3'b001;
    Funct=6'bx;
    #10;
    ALUOP=3'b010;
    Funct=6'bx;
```



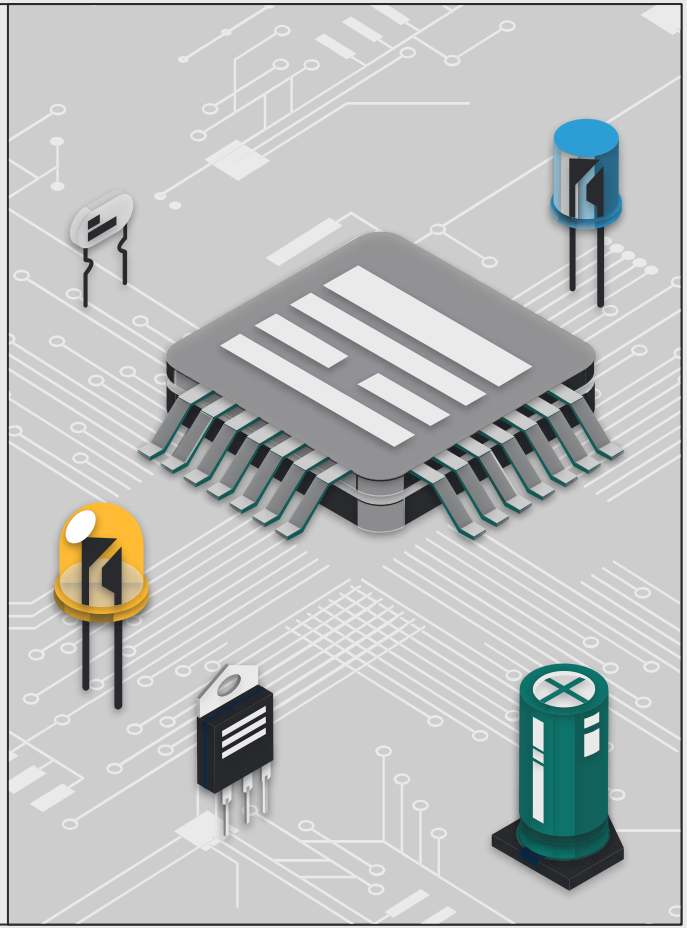
ALU control tb

```
    #10;
    ALUOP=3'b011;
    Funct=6'bx;
    #10;
    ALUOP=3'b100;
    Funct=6'bx;
    #10;
    ALUOP=3'b101;
    Funct=6'bx;
    #10;
    ALUOP=3'b110;
    Funct=6'bx;
    #10;
    ALUOP=3'b111;
    Funct=6'bx;
    #10;
    ALUOP=3'b111;
    Funct=6'd0;
    #10;
    ALUOP=3'b111;
    Funct=6'd2;
```



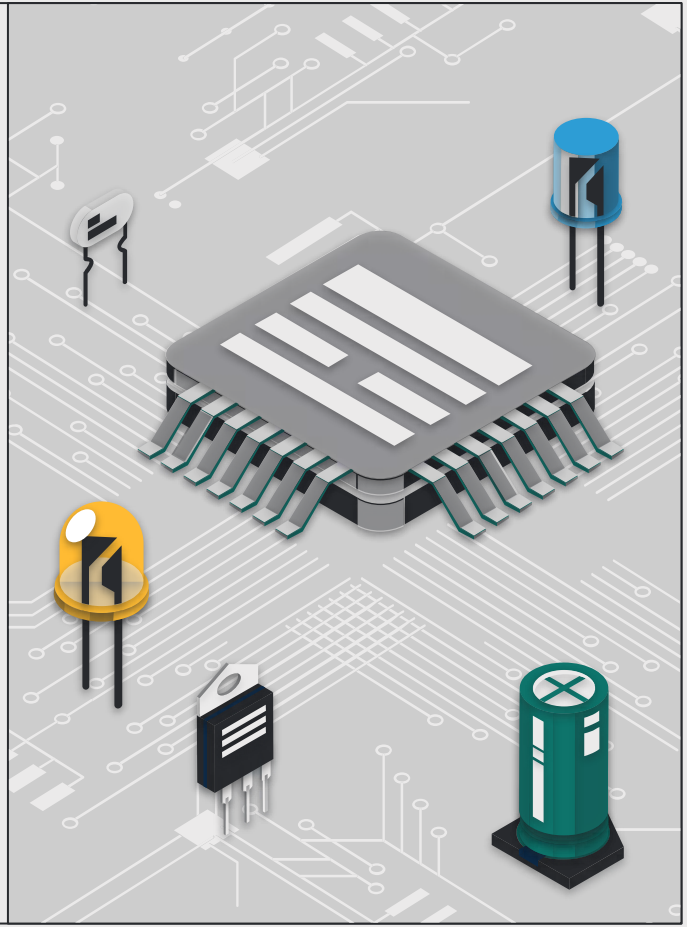
ALU control tb

```
    #10;
    ALUOP=3'b111;
    Funct=6'd3;
    #10;
    ALUOP=3'b111;
    Funct=6'd32;
    #10;
    ALUOP=3'b111;
    Funct=6'd34;
    #10;
    ALUOP=3'b111;
    Funct=6'd36;
    #10;
    ALUOP=3'b111;
    Funct=6'd37;
    #10;
    ALUOP=3'b111;
    Funct=6'd38;
    #10;
    ALUOP=3'b111;
    Funct=6'd39;
```



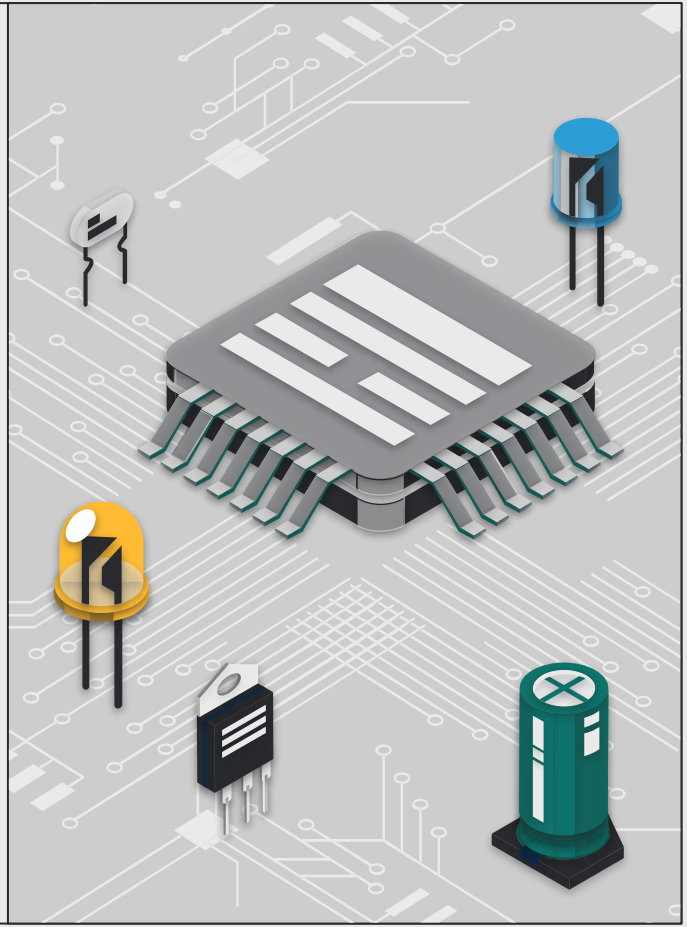
ALU control tb

```
    #10;
    ALUOP=3'b111;
    Funct=6'd42;
    #10;
end
endmodule
```



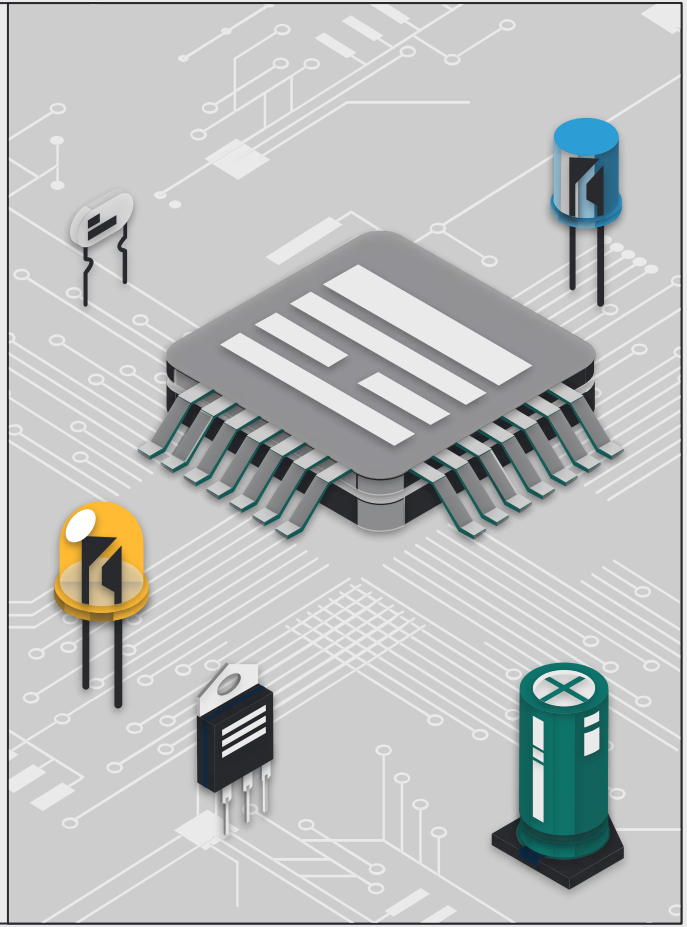
Branch control tb

```
`timescale 1ns/1ps
module Branch_control_tb ();
    reg [1:0]Branch;//bne beq
    reg Zero;
    wire OUT;
Branch_control DUT (
    .Branch(Branch),//bne beq
    .Zero(Zero),
    .OUT(OUT)
);
initial
begin
    Branch=2'b00;
    Zero=1'b0;
    #10;
    Branch=2'b00;
    Zero=1'b1;
    #10;
    Branch=2'b01;
    Zero=1'b0;
```



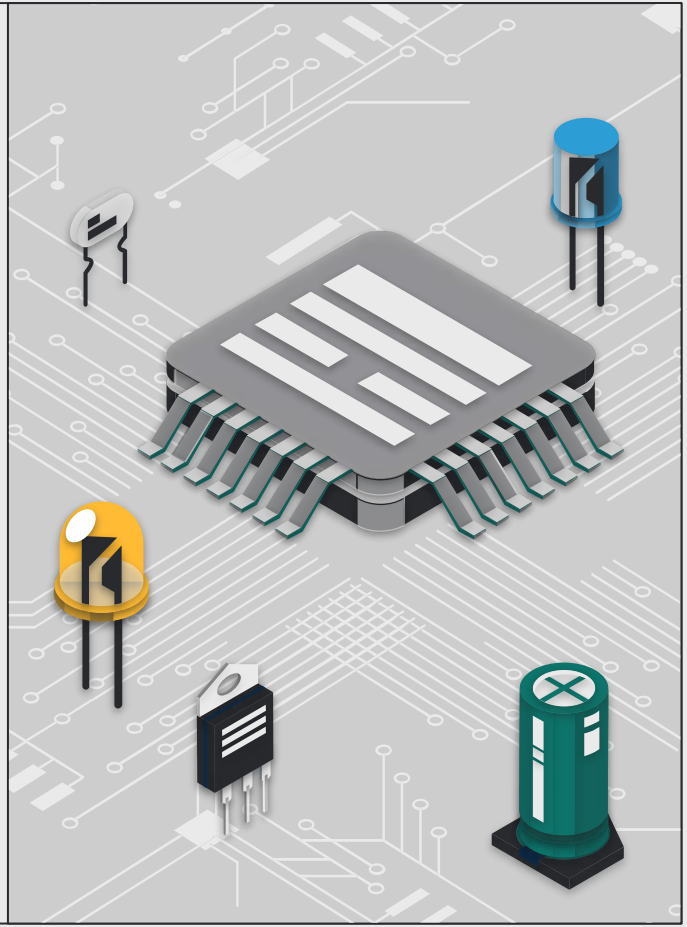
Branch control tb

```
    #10;
    Branch=2'b01;
    Zero=1'b1;
    #10;
    Branch=2'b10;
    Zero=1'b0;
    #10;
    Branch=2'b10;
    Zero=1'b1;
    #10;
end
endmodule
```



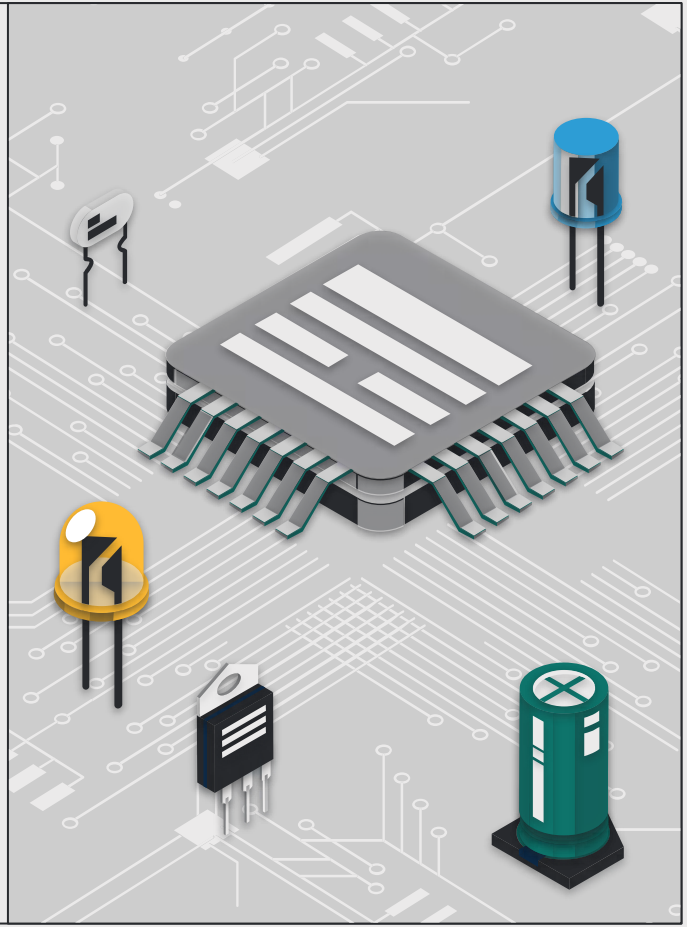
Extend unit tb

```
`timescale 1ns/1ps
module extend_unit_tb ();
    reg [15:0] IN;
    reg Sign;
    wire [31:0] OUT;
extend_unit DUT(
    .IN(IN),
    .Sign(Sign),
    .OUT(OUT)
);
initial
begin
    Sign=1'b0;
    IN=16'd2616;
    #10;
    IN=16'hffff;
    #10;
    IN=16'd2616;
    #10;
    IN=16'd2616;
```



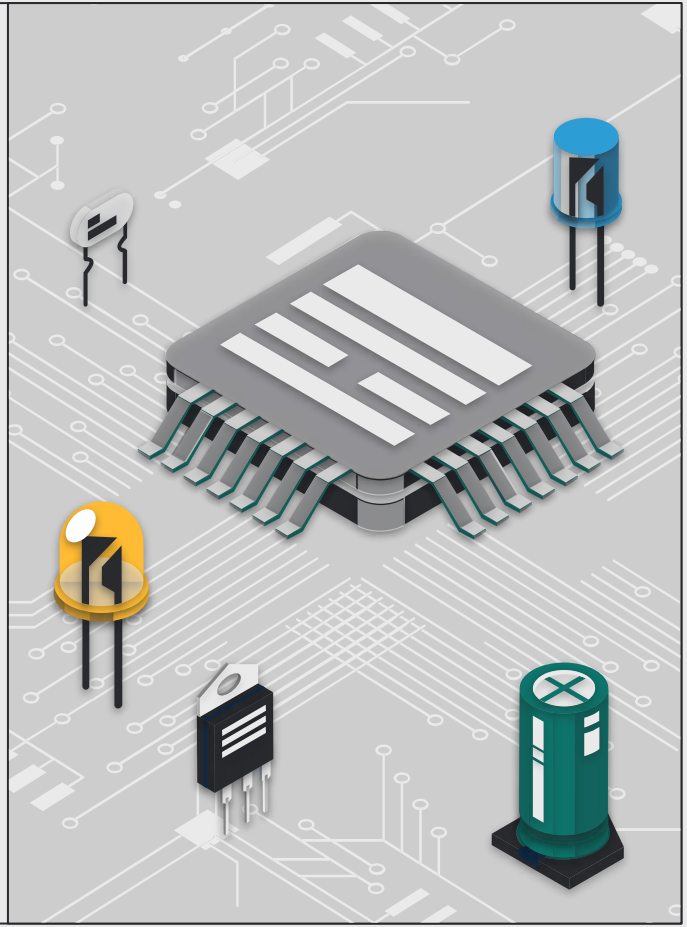
Extend unit tb

```
    #10;
    Sign=1'b1;
    #10;
    IN=16'hffab;
    #10;
    IN=16'hffff;
    #10;
    IN=16'habc;
    #10;
    IN=16'hbc;
    #10;
end
endmodule
```



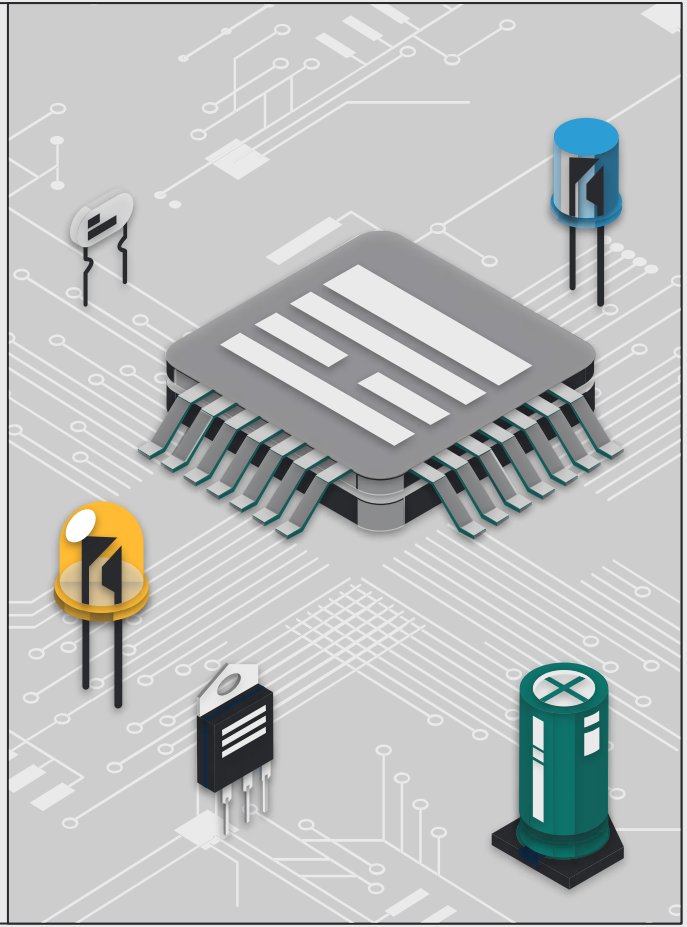
Shift left2 32→32 tb

```
`timescale 1ns/1ps
module shift_left_2_32_32_tb ();
    reg [31:0] IN;
    wire [31:0] OUT;
shift_left_2_32_32 DUT (
    .IN(IN),
    .OUT(OUT)
);
initial
begin
    IN=32'd1556;
    #10;
    IN=32'd1518;
    #10;
    IN=32'd7856;
    #10;
    IN=32'd5962;
    #10;
```



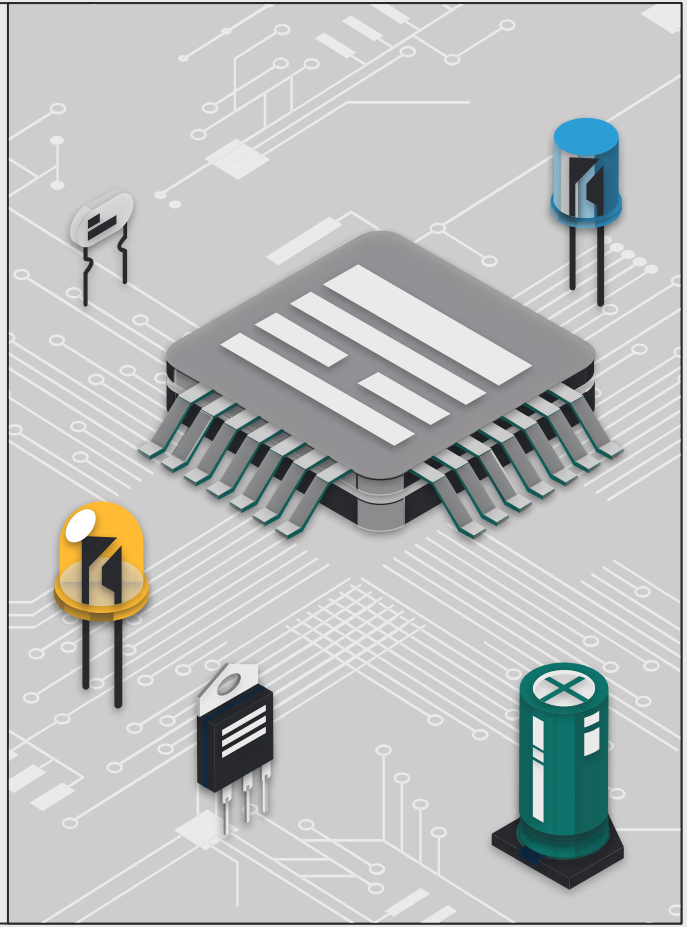
Shift left2 32→32 tb

```
    IN=32'd4896;
    #10;
    IN=32'd7598;
    #10;
    IN=32'd4568;
    #10;
    IN=32'd7851;
    #10;
    IN=32'd8748;
    #10;
    IN=32'd4896;
    #10;
end
endmodule
```



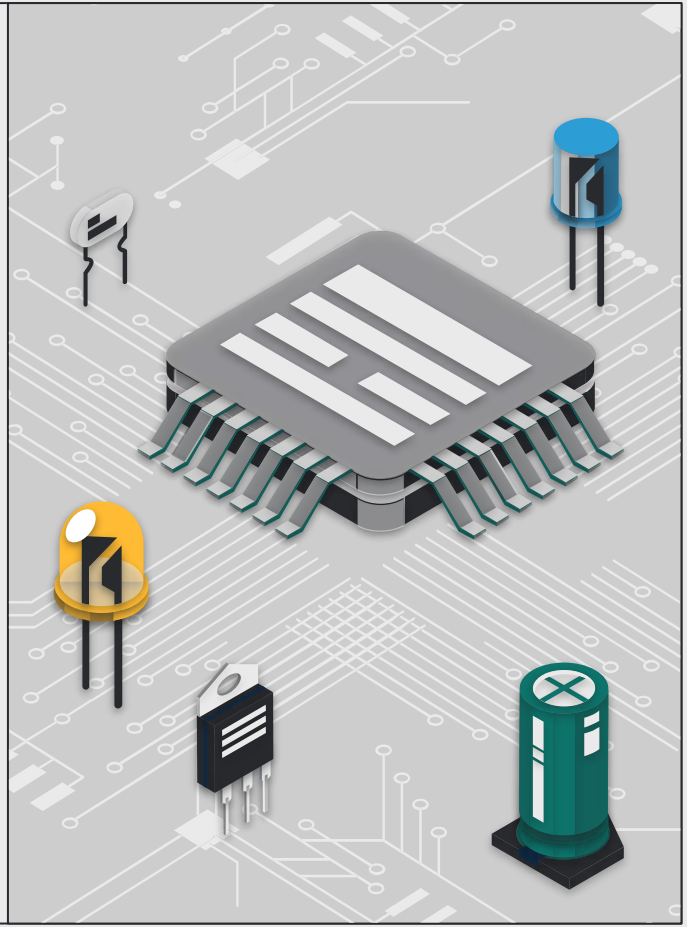
Shift left2 26→28 tb

```
`timescale 1ns/1ps
module shift_left_2_26_28_tb ();
    reg [25:0] IN;
    wire [27:0] OUT;
shift_left_2_26_28 DUT (
    .IN(IN),
    .OUT(OUT)
);
initial
begin
    IN=26'd1556;
    #10;
    IN=26'd1518;
    #10;
    IN=26'd7856;
    #10;
    IN=26'd5962;
    #10;
    IN=26'd4896;
```



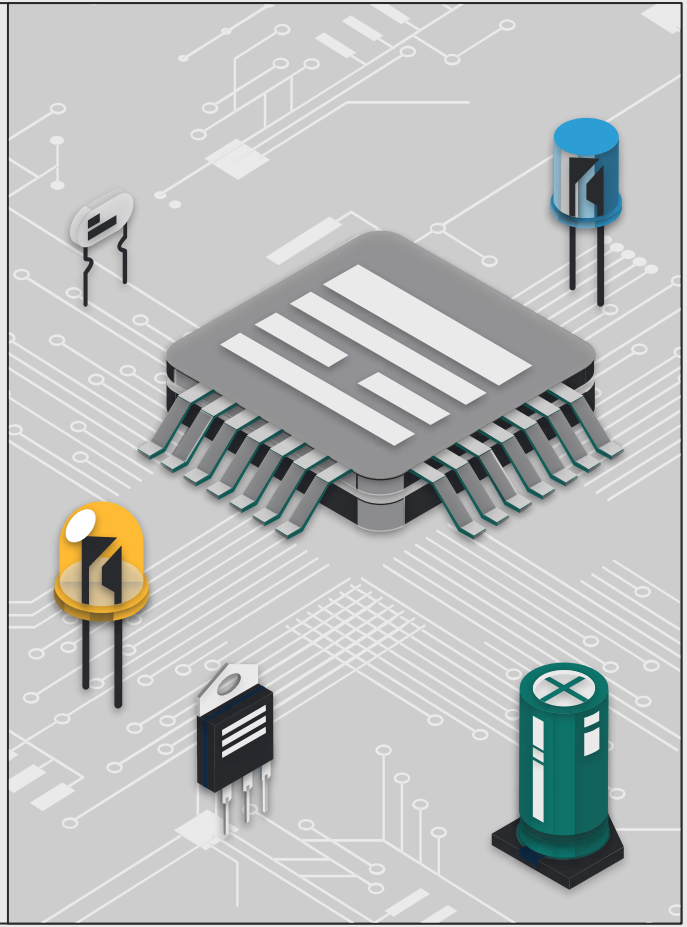
Shift left2 26→28 tb

```
    #10;
    IN=26'd7598;
    #10;
    IN=26'd4568;
    #10;
    IN=26'd7851;
    #10;
    IN=26'd8748;
    #10;
    IN=26'd4896;
    #10;
end
endmodule
```



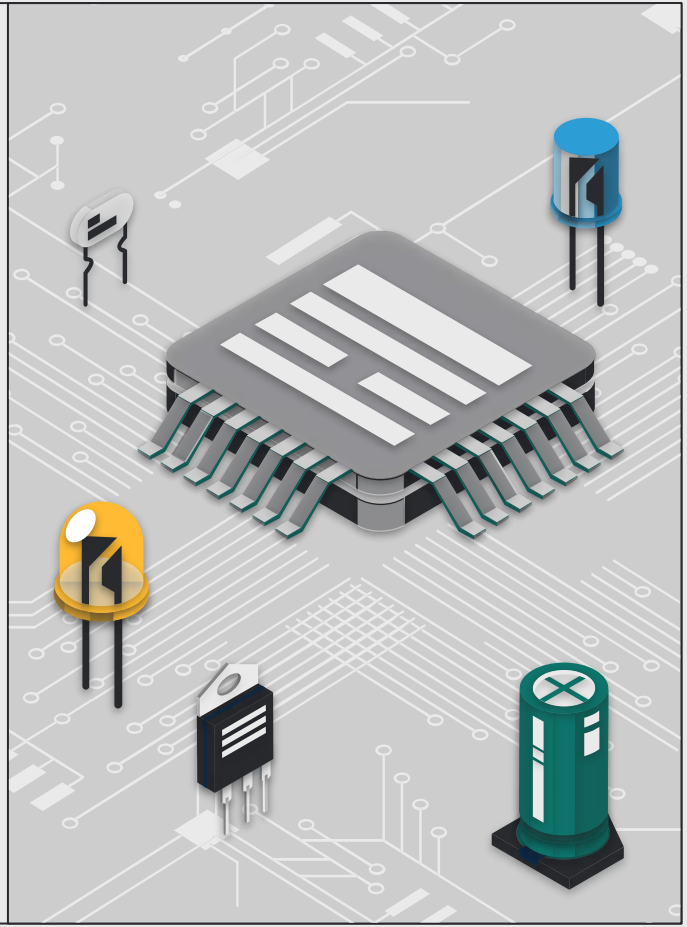
Registers_tb

```
`timescale 1ns/1ps
module Registers_tb ();
    reg [4:0]Read_register1,Read_register2;
    reg [4:0]Write_register;
    wire [31:0]Read_data1,Read_data2;
    reg [31:0] Write_data;
    reg clk=0,RegWrite;
    parameter clk_prd=100;
    always # (clk_prd/2) clk=~clk;
Registers DUT(
    .Read_register1(Read_register1),
    .Read_register2(Read_register2),
    .Write_register(Write_register),
    .Read_data1(Read_data1),
    .Read_data2(Read_data2),
    .Write_data(Write_data),
    .clk(clk),
    .RegWrite(RegWrite)
    );
initial
begin
```



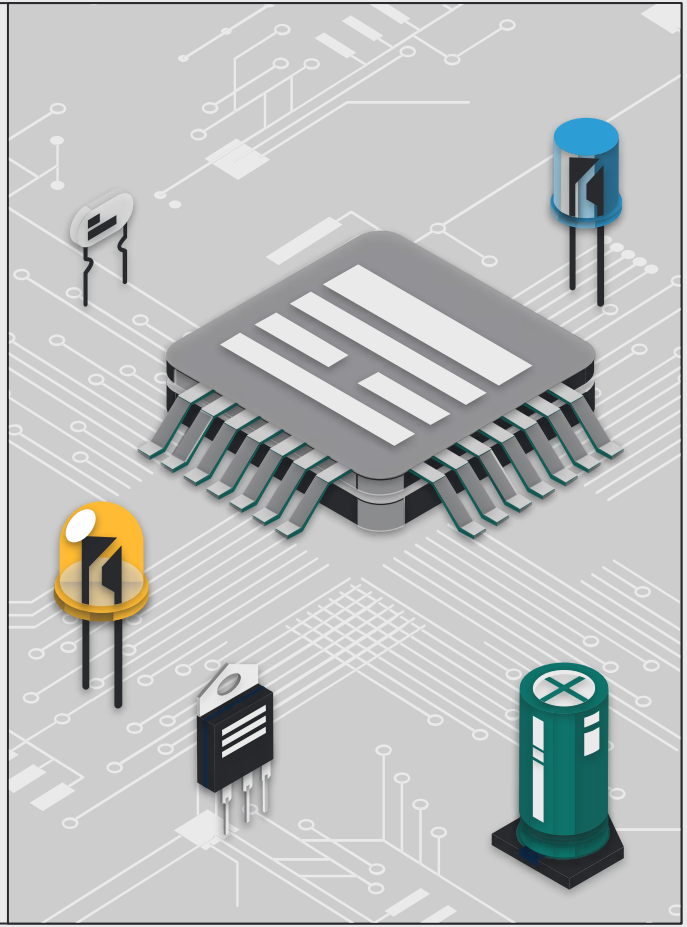
Registers tb

```
Write_register=5'd1;
Read_register1=5'd1;
Read_register2=5'd2;
Write_data=32'hfffaff;
RegWrite=1'b1;
#70;
Write_register=5'd2;
Read_register1=5'd1;
Read_register2=5'd2;
Write_data=32'habcdef;
RegWrite=1'b1;
#100;
Write_register=5'd3;
Read_register1=5'd2;
Read_register2=5'd3;
Write_data=32'hfafafaf;
RegWrite=1'b1;
#100;
```



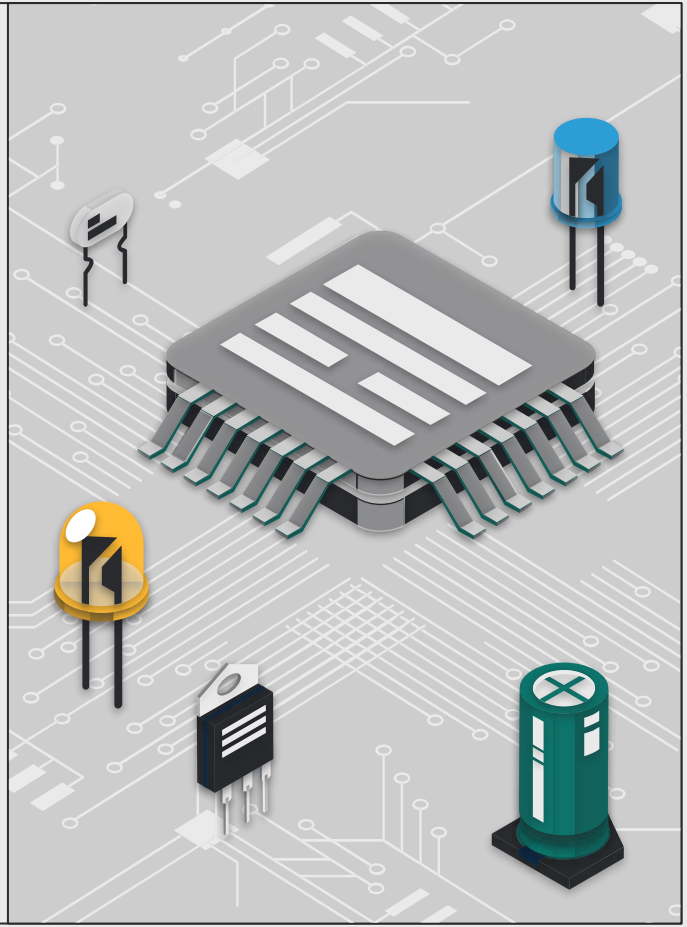
Registers tb

```
    Write_register=5'd4;
    Read_register1=5'd2;
    Read_register2=5'd3;
    Write_data=32'haddfa;
    RegWrite=1'b1;
    #100;
    Write_register=5'd3;
    Read_register1=5'd3;
    Read_register2=5'd4;
    Write_data=32'habda;
    RegWrite=1'b1;
    #100;
end
endmodule
```



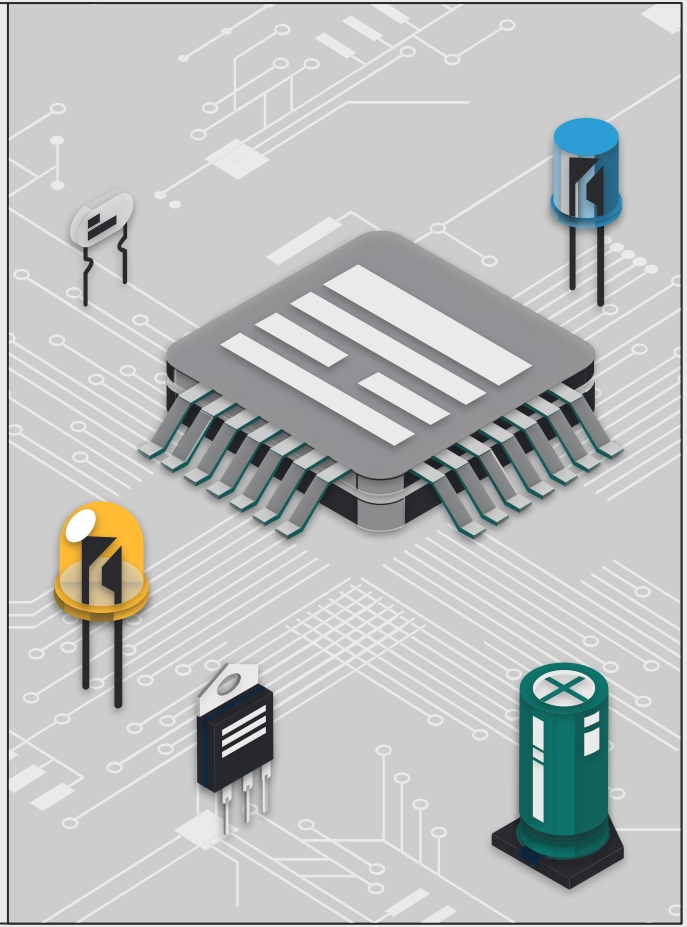
Instruction memory tb

```
`timescale 1ns/1ps
module Instruction_memory_tb();
    reg [31:2] ReadAddress,WriteAddress;
    reg [31:0] writeDataINS;
    wire [31:0] Instruction;
    reg writeINS,clk;
    parameter clk_prd=100;
    always # (clk_prd/2)clk=~clk;
Instruction_memory DUT(
    .ReadAddress(ReadAddress),
    .Instruction(Instruction),
    .WriteAddress(WriteAddress),
    .writeINS(writeINS),
    .clk(clk),
    .writeDataINS(writeDataINS)
);
    initial
    begin
    writeINS=1'b0;
    ReadAddress=30'd0;
```



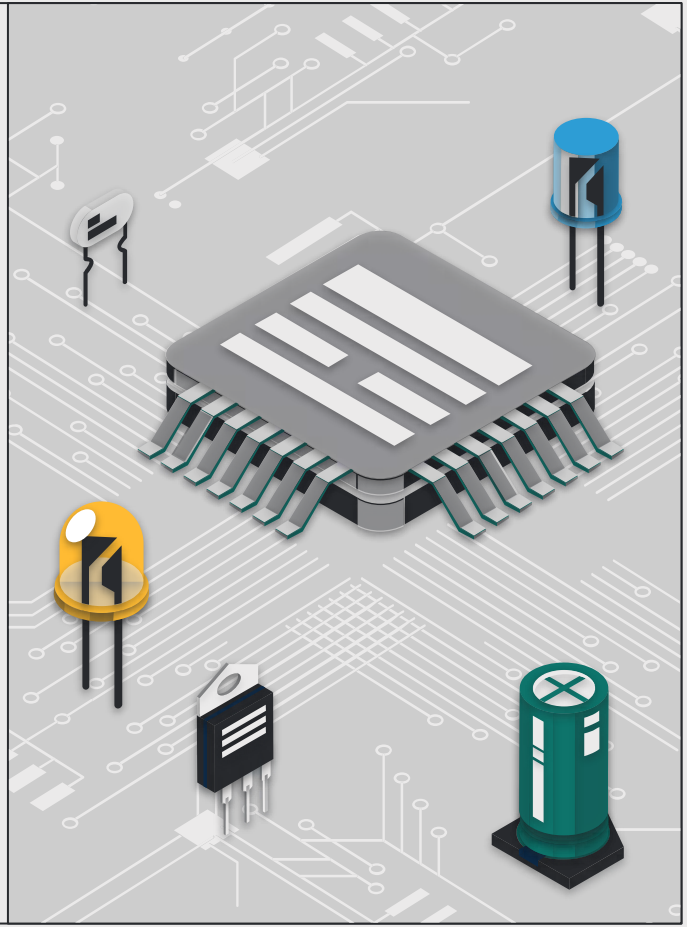
Instruction memory tb

```
    #10;
    ReadAddress=30'd1;
    #10;
    ReadAddress=30'd2;
    #10;
    ReadAddress=30'd3;
    #10;
    ReadAddress=30'd4;
    #10;
    ReadAddress=30'd5;
    #10;
    ReadAddress=30'd6;
    #10;
```



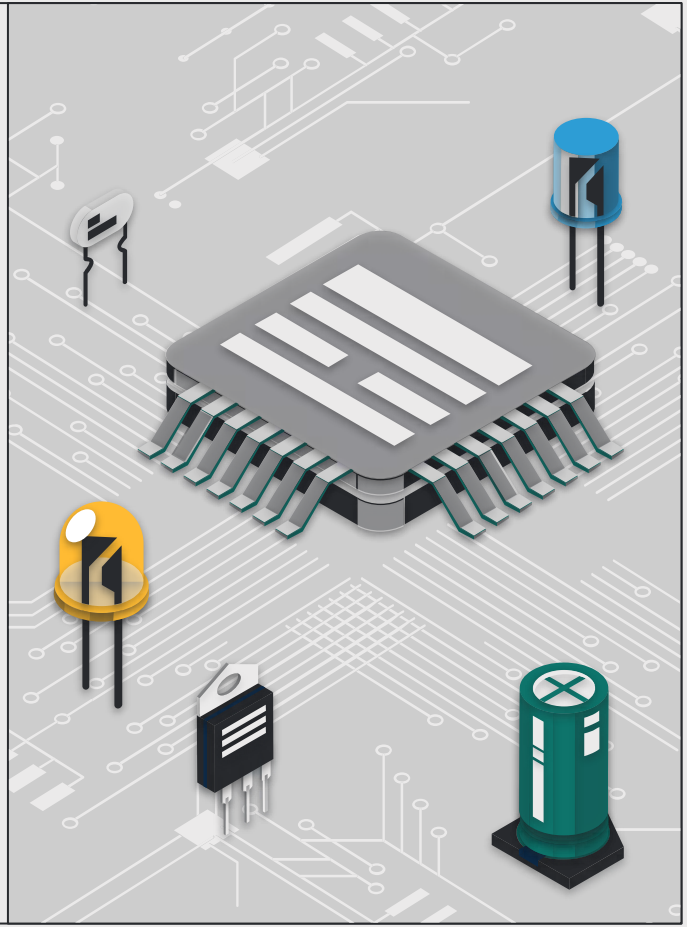
Instruction memory tb

```
    ReadAddress=30'd7;
    #10;
    ReadAddress=30'd8;
    #10;
    ReadAddress=30'd9;
    #10;
    ReadAddress=30'd10;
    #10;
    ReadAddress=30'd11;
    #10;
    end
endmodule
```



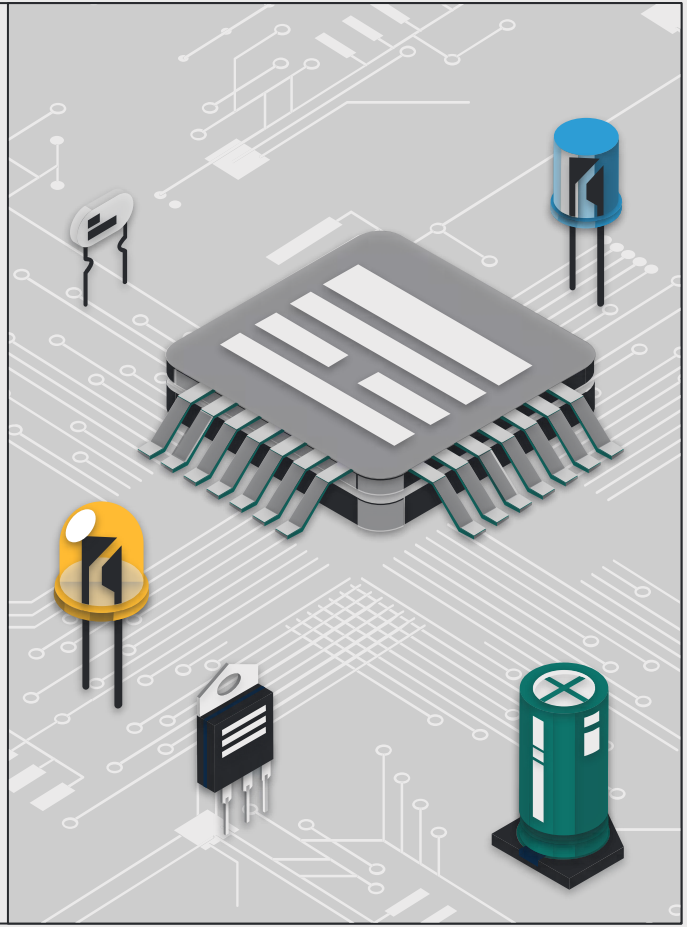
Data memory tb

```
`timescale 1ns/1ps
module Data_memory_tb ();
    reg MemWrite;
    reg [1:0]sel,byte_addr;
    reg [31:2] Address;
    reg [31:0]Write_data;
    wire [31:0] Read_data;
    reg clk=0;
    parameter clk_Prd=100;
    always # (clk_Prd/2) clk=~clk;

Data_memory DUT (
    .clk(clk),
    .sel(sel),
    .MemWrite(MemWrite),
    .Address(Address),
    .Write_data(Write_data),
    .Read_data(Read_data),
    .byte_addr(byte_addr)
);
```



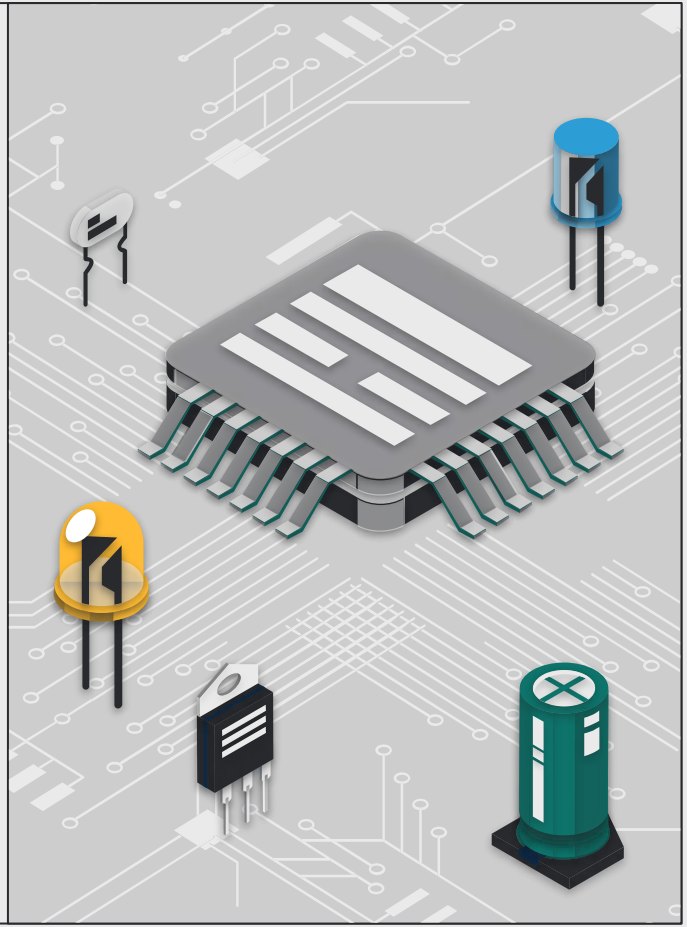
Data memory tb

```
initial
begin
sel=2'b00;
byte_addr=2'b00;
MemWrite=1'b1;
Address=30'd127;
Write_data=32'hfafa;
#80;
sel=2'b00;
byte_addr=2'b00;
MemWrite=1'b1;
Address=30'd126;
Write_data=32'hbaba;
#100;
sel=2'b01;
byte_addr=2'b00;
MemWrite=1'b1;
Address=30'd125;
Write_data=32'habcd;
```



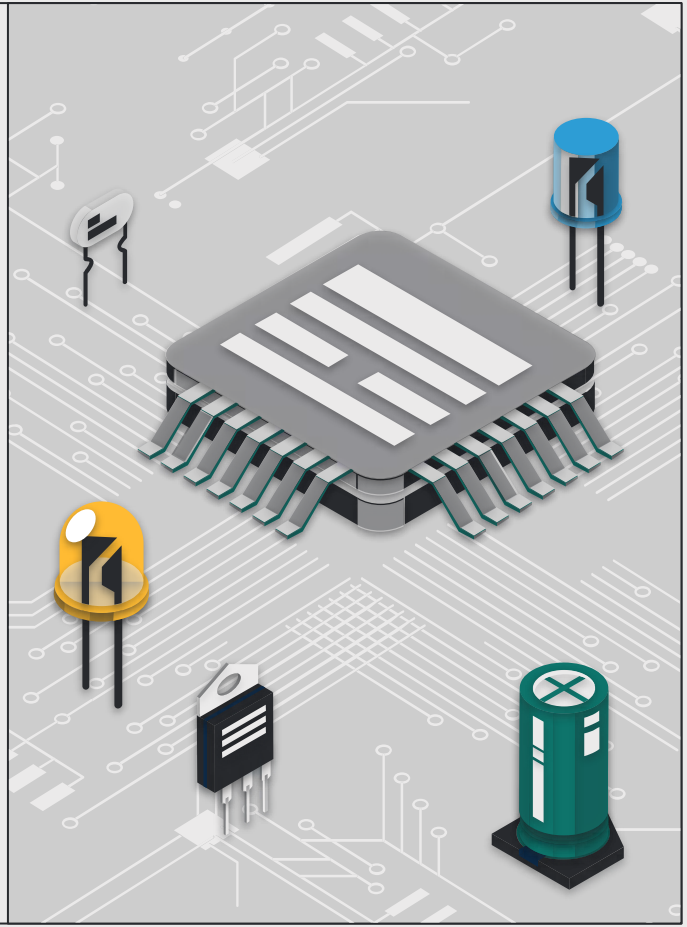
Data memory tb

```
    #100;
    sel=2'b10;
    byte_addr=2'b00;
    MemWrite=1'b1;
    Address=30'd124;
    Write_data=32'hadbf;
    #100;
    sel=2'b11;
    byte_addr=2'b00;
    MemWrite=1'b1;
    Address=30'd123;
    Write_data=32'hfabc;
    #100;
////////////byte_addr_01
    sel=2'b00;
    byte_addr=2'b01;
    MemWrite=1'b1;
    Address=30'd127;
    Write_data=32'hfafa;
```



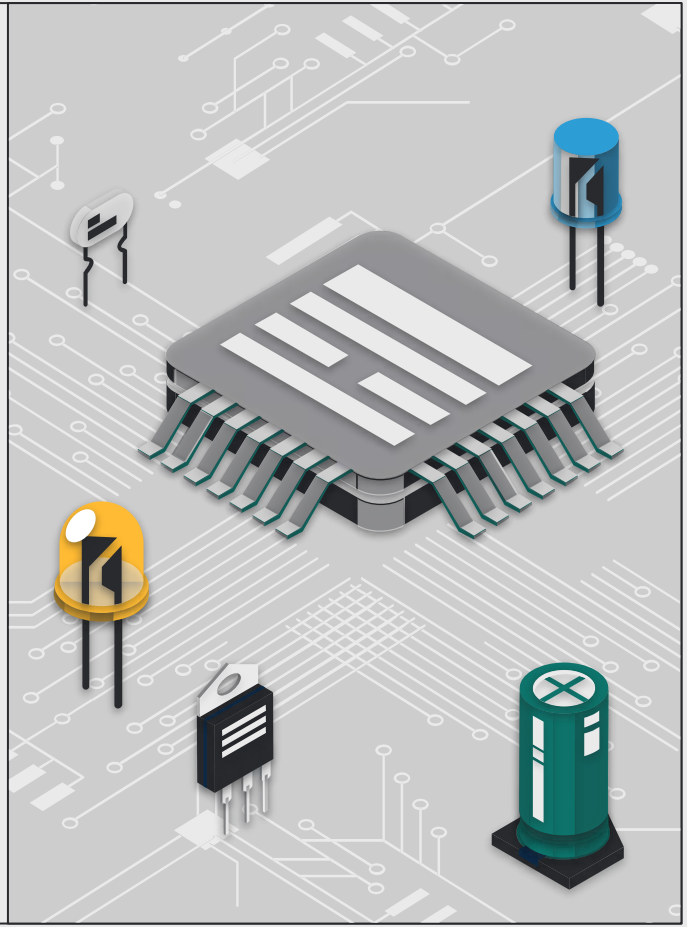
Data memory tb

```
#80;
sel=2'b00;
byte_addr=2'b01;
MemWrite=1'b1;
Address=30'd126;
Write_data=32'hbaba;
#100;
sel=2'b01;
byte_addr=2'b01;
MemWrite=1'b1;
Address=30'd125;
Write_data=32'habcd;
#100;
sel=2'b10;
byte_addr=2'b01;
MemWrite=1'b1;
Address=30'd124;
Write_data=32'hadbf;
```



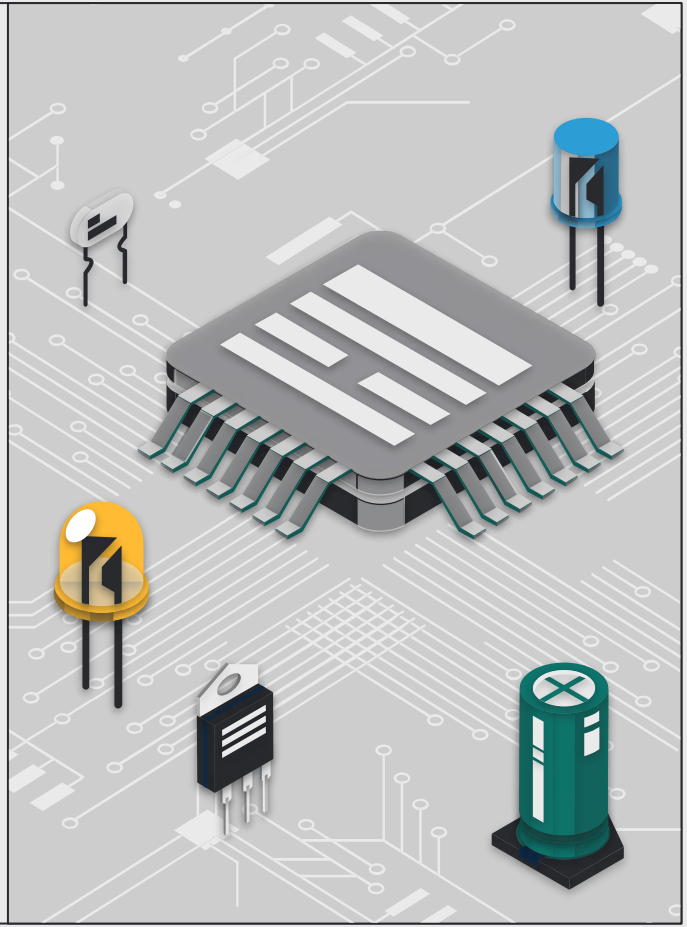
Data memory tb

```
    #100;
    sel=2'b11;
    byte_addr=2'b01;
    MemWrite=1'b1;
    Address=30'd123;
    Write_data=32'hfabc;
    #100;
/////byte_addr_10
    sel=2'b00;
    byte_addr=2'b10;
    MemWrite=1'b1;
    Address=30'd127;
    Write_data=32'hfafa;
    #80;
    sel=2'b00;
    byte_addr=2'b10;
    MemWrite=1'b1;
    Address=30'd126;
    Write_data=32'hbaba;
```



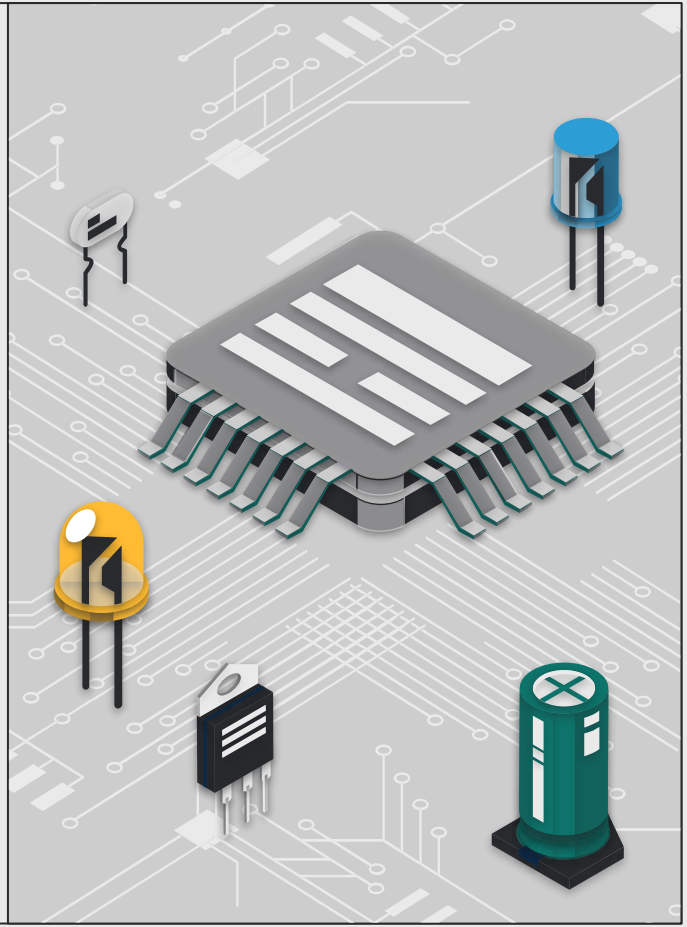
Data memory tb

```
    #100;
    sel=2'b01;
    byte_addr=2'b10;
    MemWrite=1'b1;
    Address=30'd125;
    Write_data=32'habcd;
    #100;
    sel=2'b10;
    byte_addr=2'b10;
    MemWrite=1'b1;
    Address=30'd124;
    Write_data=32'hadbf;
    #100;
    sel=2'b11;
    byte_addr=2'b10;
    MemWrite=1'b1;
    Address=30'd123;
    Write_data=32'hfabc;
    #100;
```



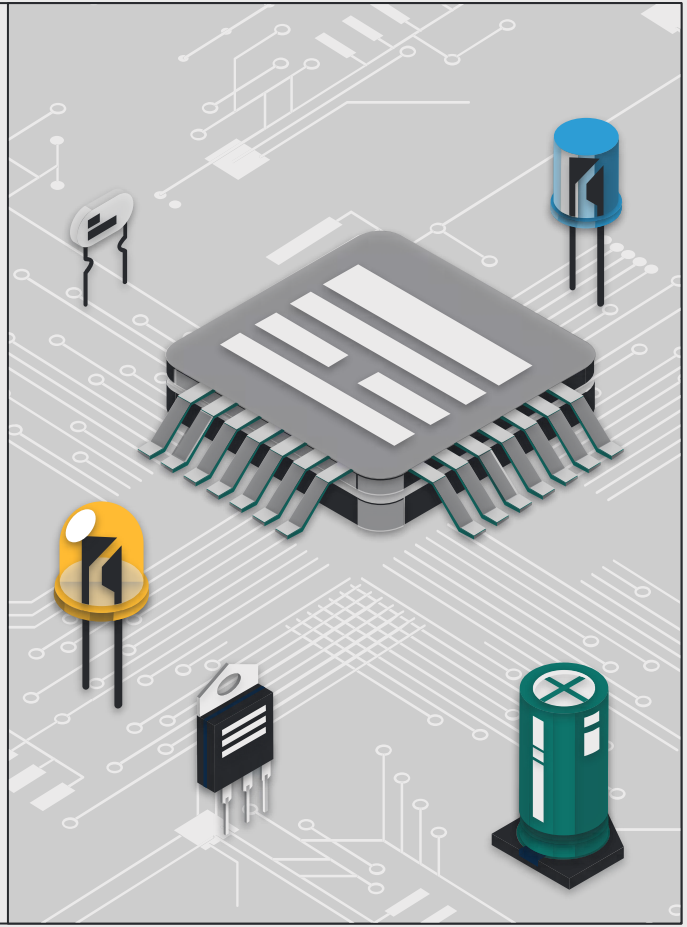
Data memory tb

```
///////byte_addr_11
    sel=2'b00;
    byte_addr=2'b11;
    MemWrite=1'b1;
    Address=30'd127;
    Write_data=32'hfafa;
    #80;
    sel=2'b00;
    byte_addr=2'b11;
    MemWrite=1'b1;
    Address=30'd126;
    Write_data=32'hbaba;
    #100;
    sel=2'b01;
    byte_addr=2'b11;
    MemWrite=1'b1;
    Address=30'd125;
    Write_data=32'habcd;
    #100;
```



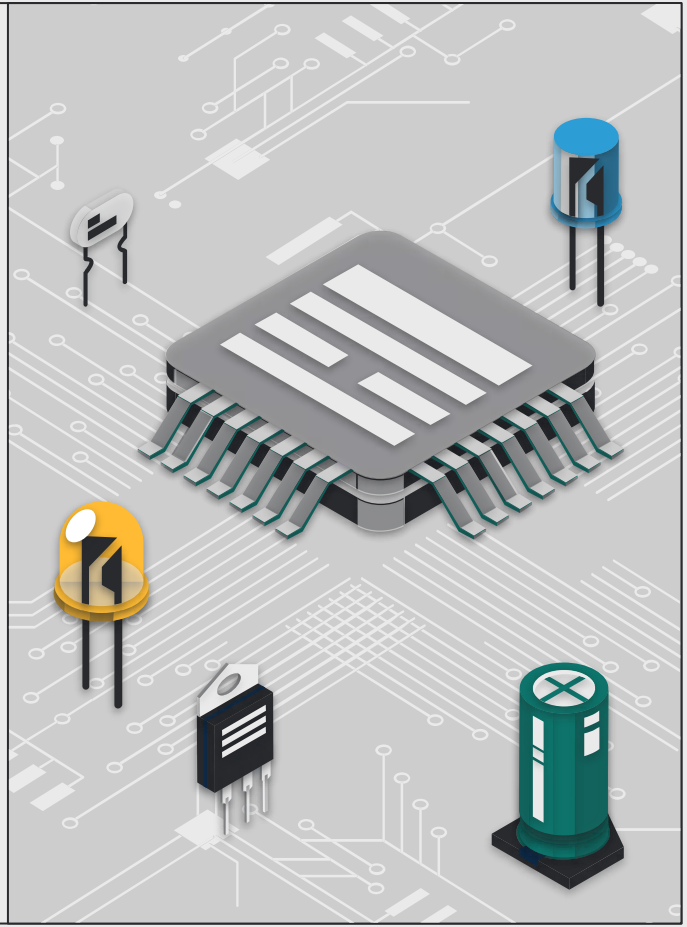
Data memory tb

```
    sel=2'b10;
    byte_addr=2'b11;
    MemWrite=1'b1;
    Address=30'd124;
    Write_data=32'hadbf;
    #100;
    sel=2'b11;
    byte_addr=2'b11;
    MemWrite=1'b1;
    Address=30'd123;
    Write_data=32'hfabc;
    #100;
    end
endmodule
```



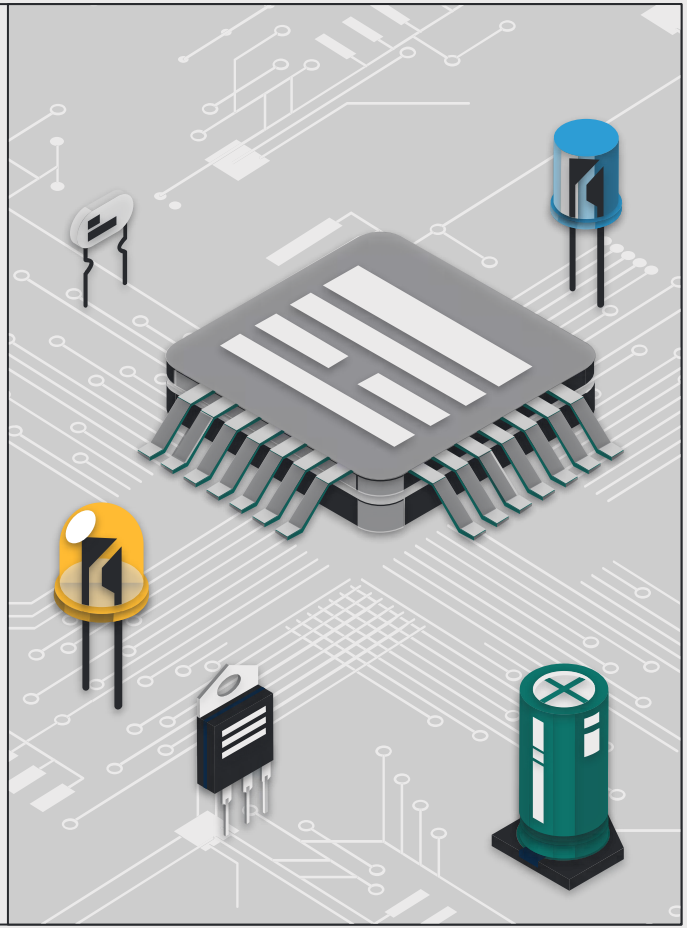
PC_tb

```
`timescale 1ns/1ps
module PC_tb ();
    reg clk=0;
    reg [31:0] IN;
    wire [31:0] OUT;
    parameter clk_prd=100;
    always # (clk_prd/2)clk=~clk;
PC DUT(
    .clk(clk),
    .IN(IN),
    .OUT(OUT)
);
```



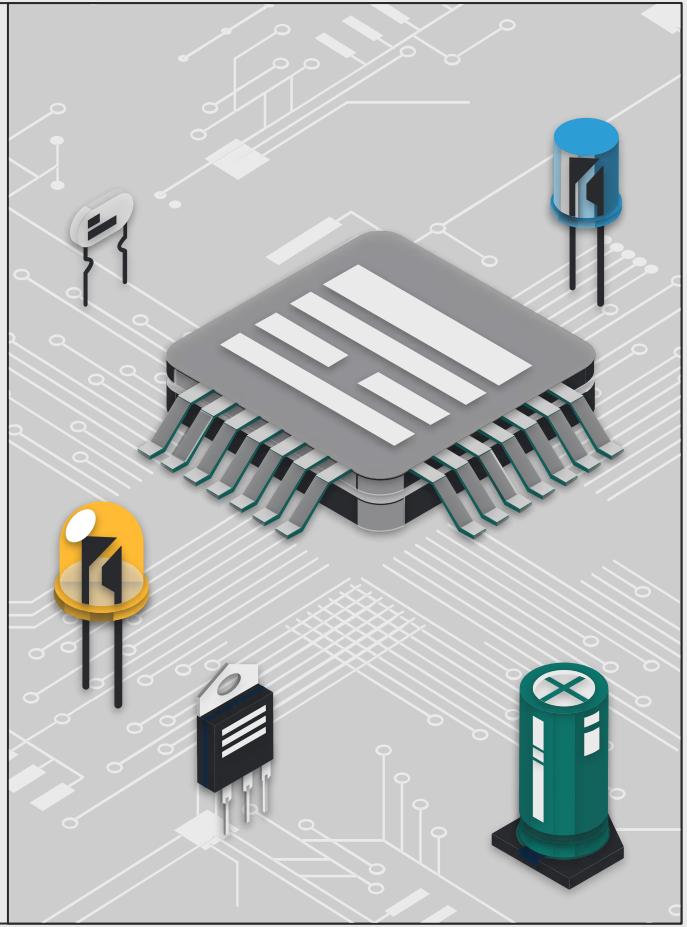
PC tb

```
initial
begin
    IN=32'd4535;
    #70;
    IN=32'd7811;
    #100;
    IN=32'd4277;
    #100;
    IN=32'd1535;
    #100;
    IN=32'd5343;
    #100;
    IN=32'd83434;
    #100;
end
endmodule
```



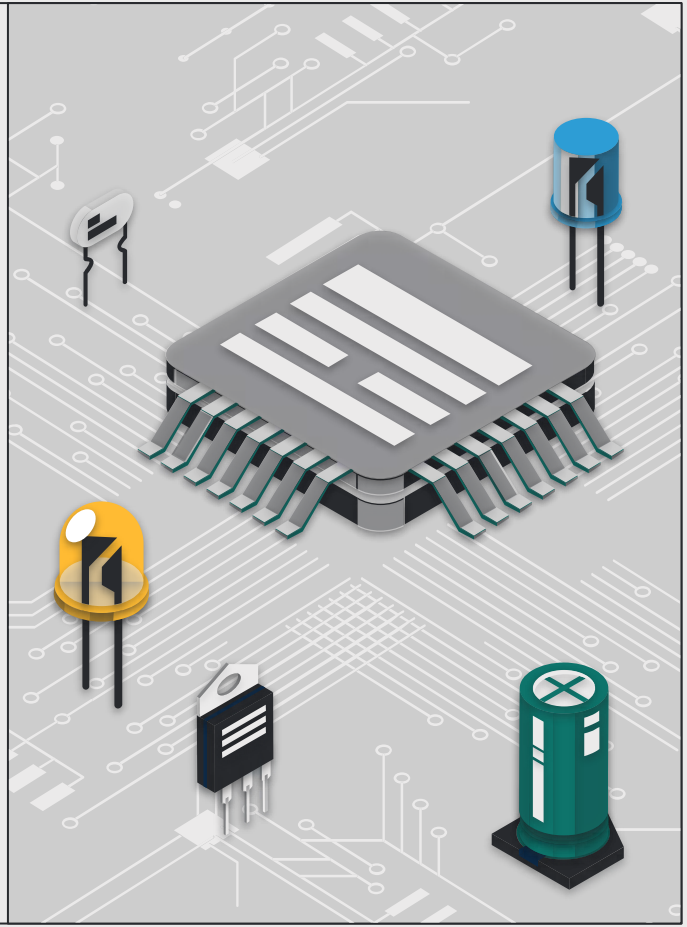
MUX 4:1 tb

```
`timescale 1ns/1ps
module MUX_4_1_tb();
    parameter N = 32;
    reg [N-1:0] A,B,C,D;
    reg [1:0]sel;
    wire [N-1:0] Z;
MUX_4_1 #(.N(N))DUT(
    .A(A),
    .B(B),
    .C(C),
    .D(D),
    .sel(sel),
    .Z(Z)
);
initial
begin
    A=32'd4526;
    B=32'd5659;
    C=32'd745;
    D=32'd2156;
    sel=2'b00;
```



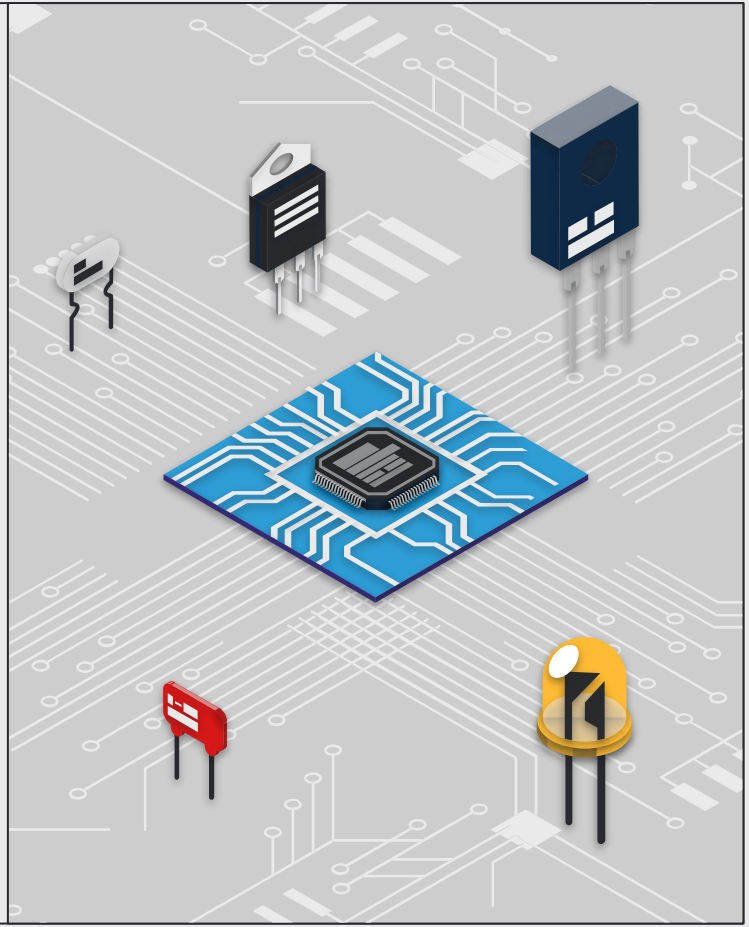
MUX 4:1 tb

```
    #10;
    sel=2'b01;
    #10;
    sel=2'b10;
    #10;
    sel=2'b11;
    #10;
    A=32'd4548;
    B=32'd1568;
    C=32'd78515;
    D=32'd1558;
    sel=2'b00;
    #10;
    sel=2'b01;
    #10;
    sel=2'b10;
    #10;
    sel=2'b11;
    #10;
end
endmodule
```

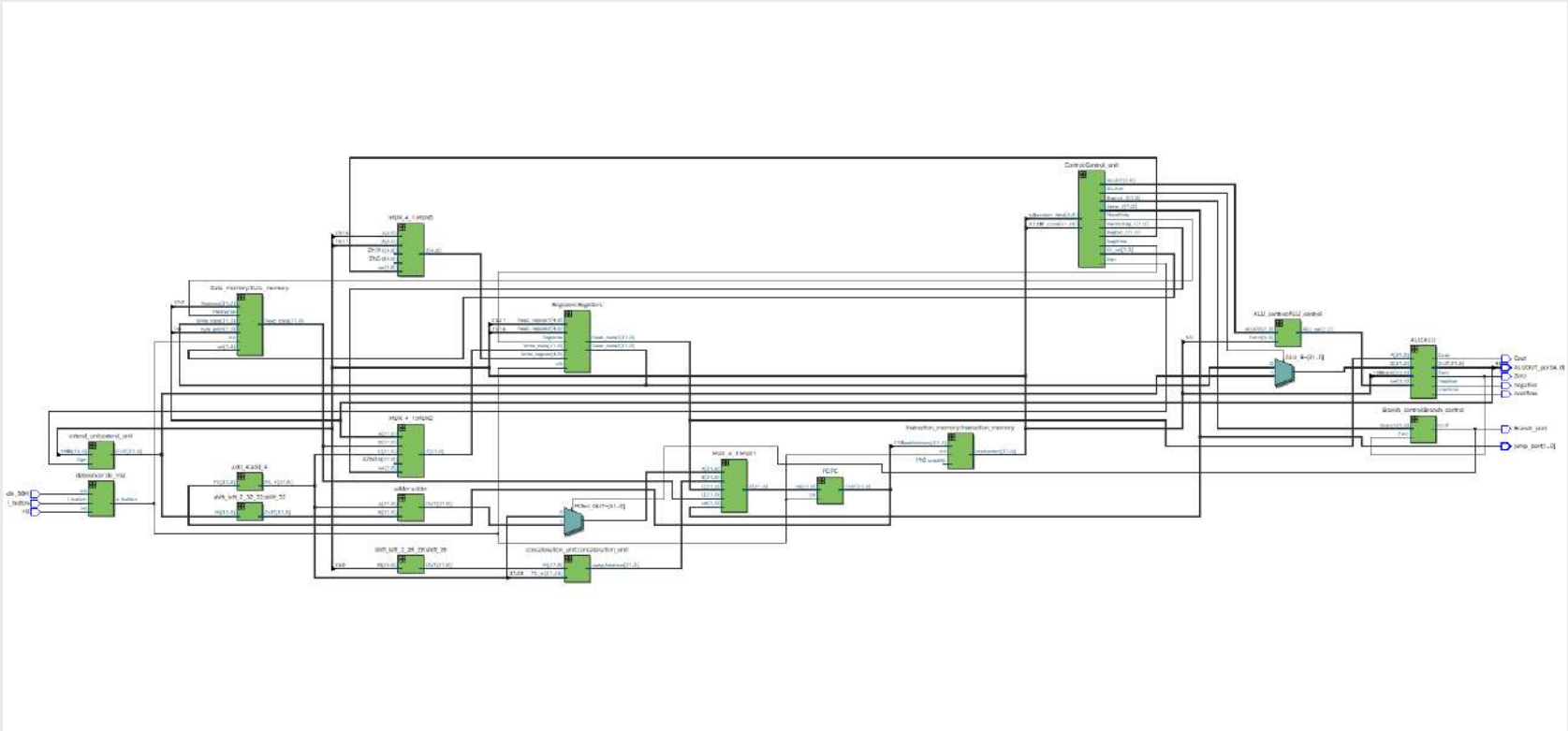


04

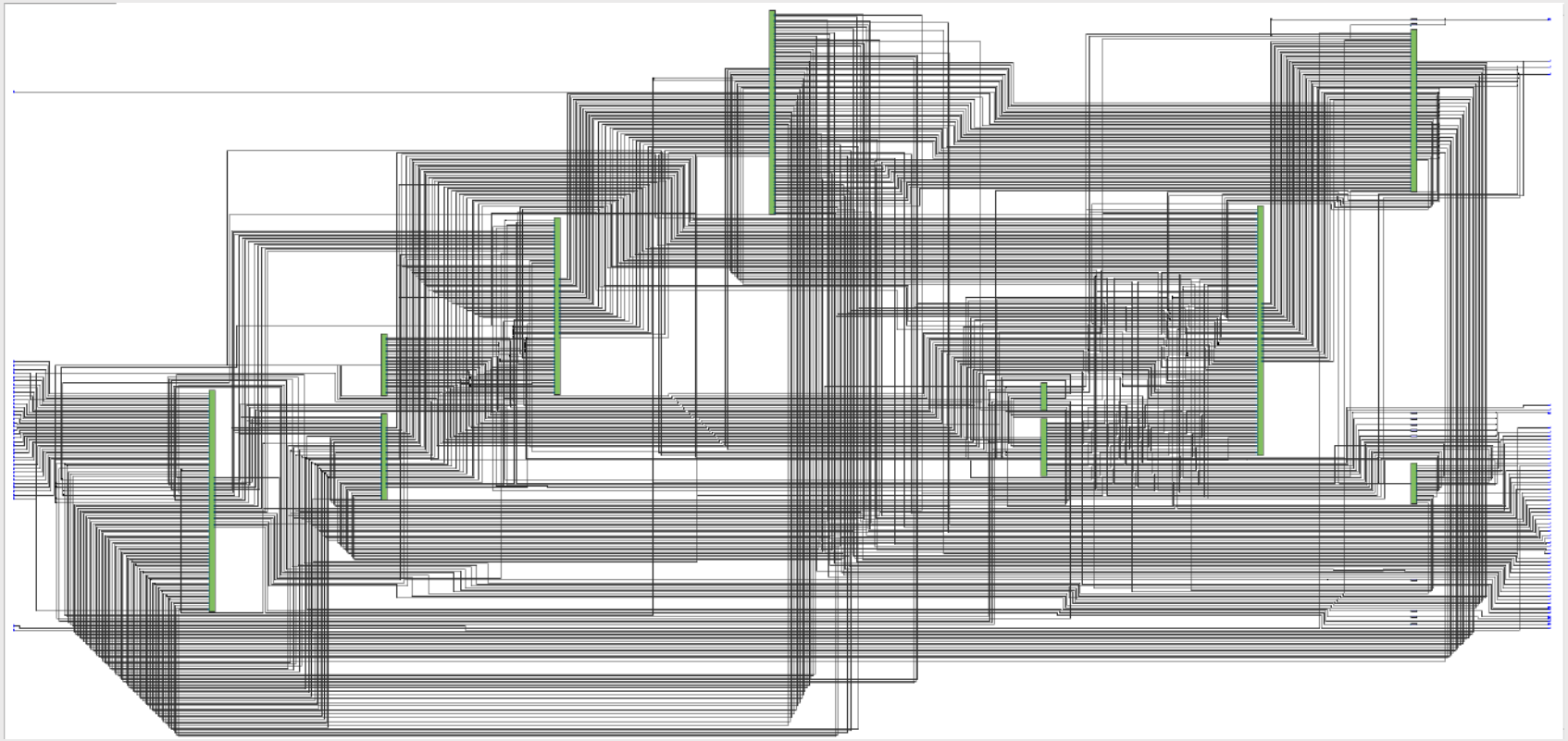
RTL & ModelSIM simulation



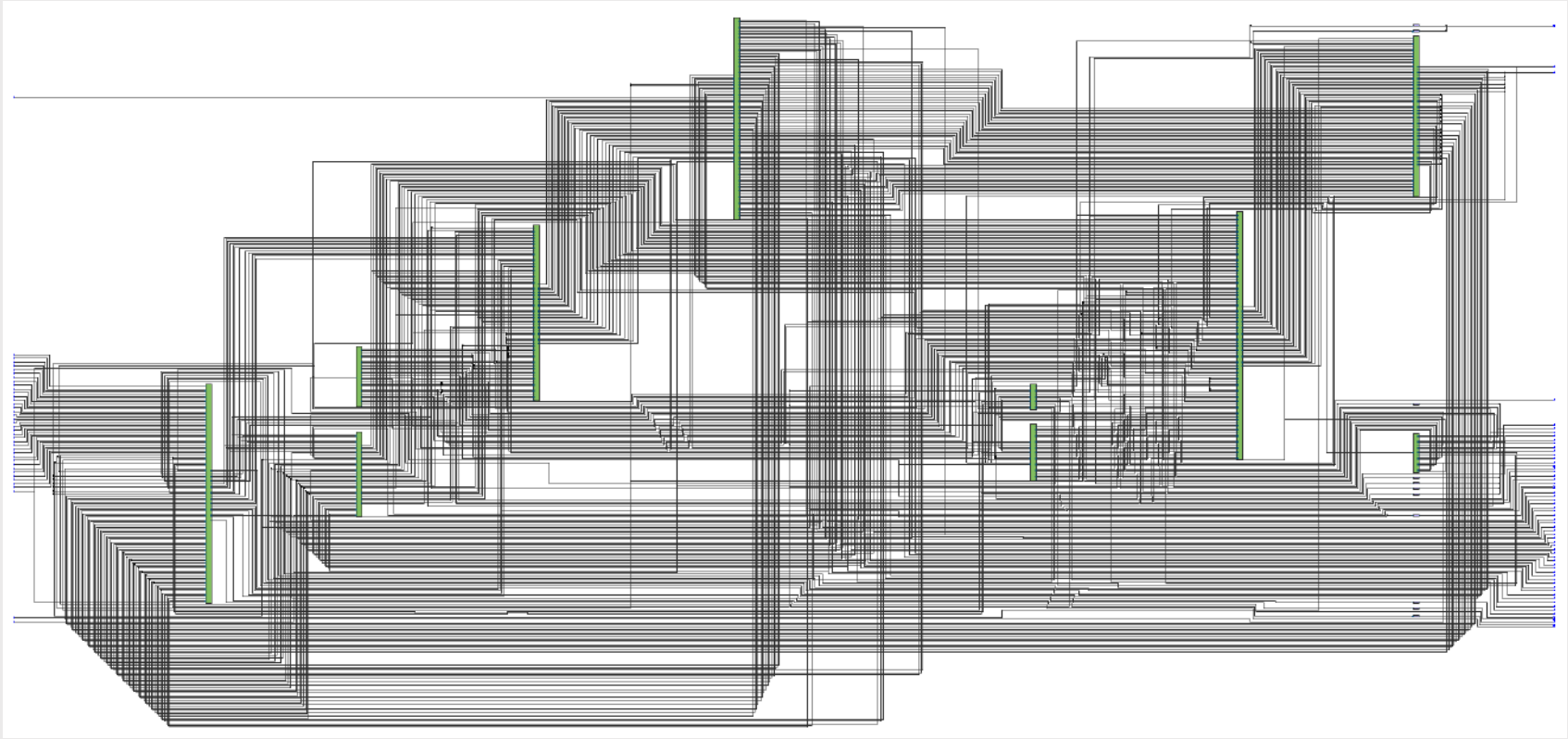
MIPS processor with debouncer (RTL)



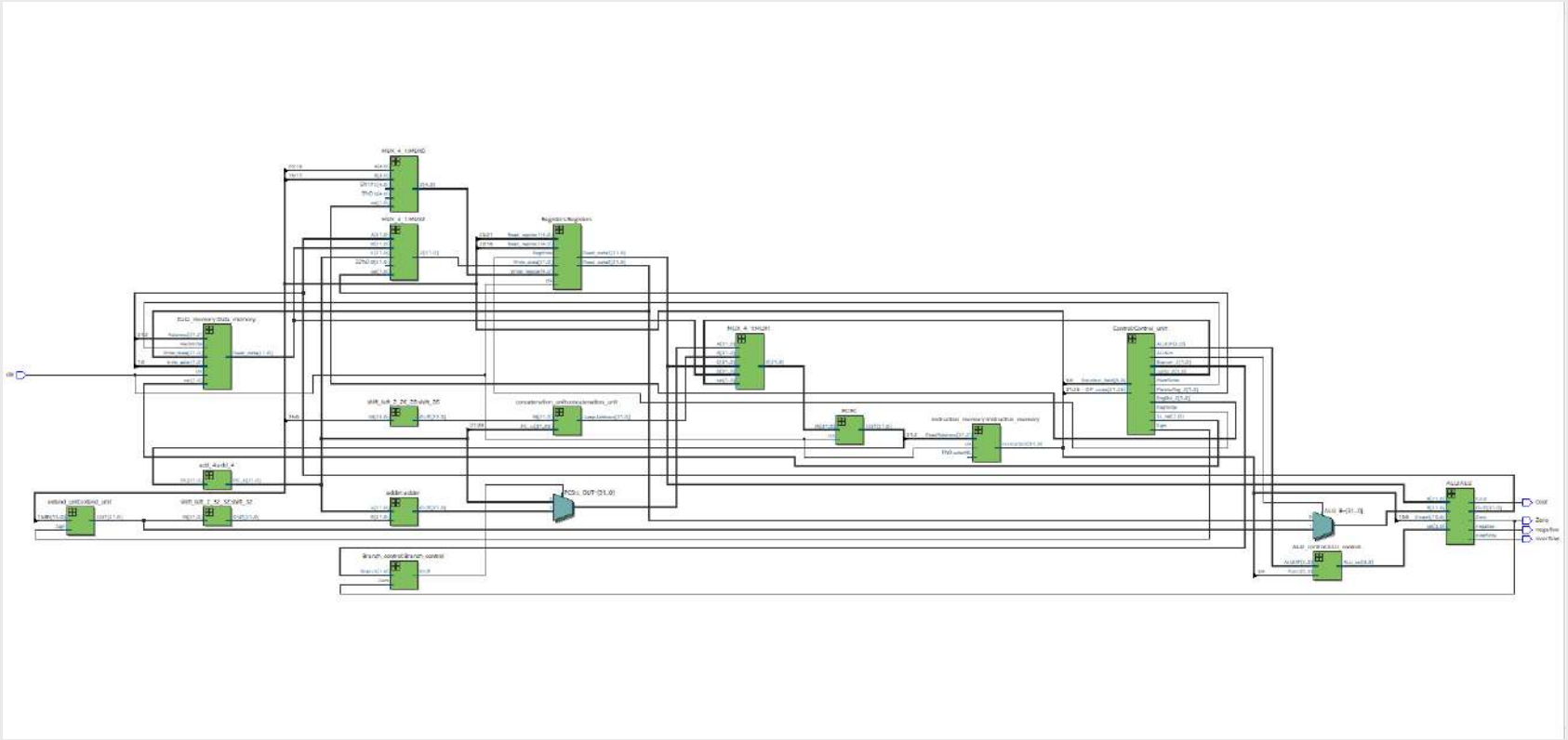
MIPS processor with debouncer (Technology Map Viewer (Post-mapping))



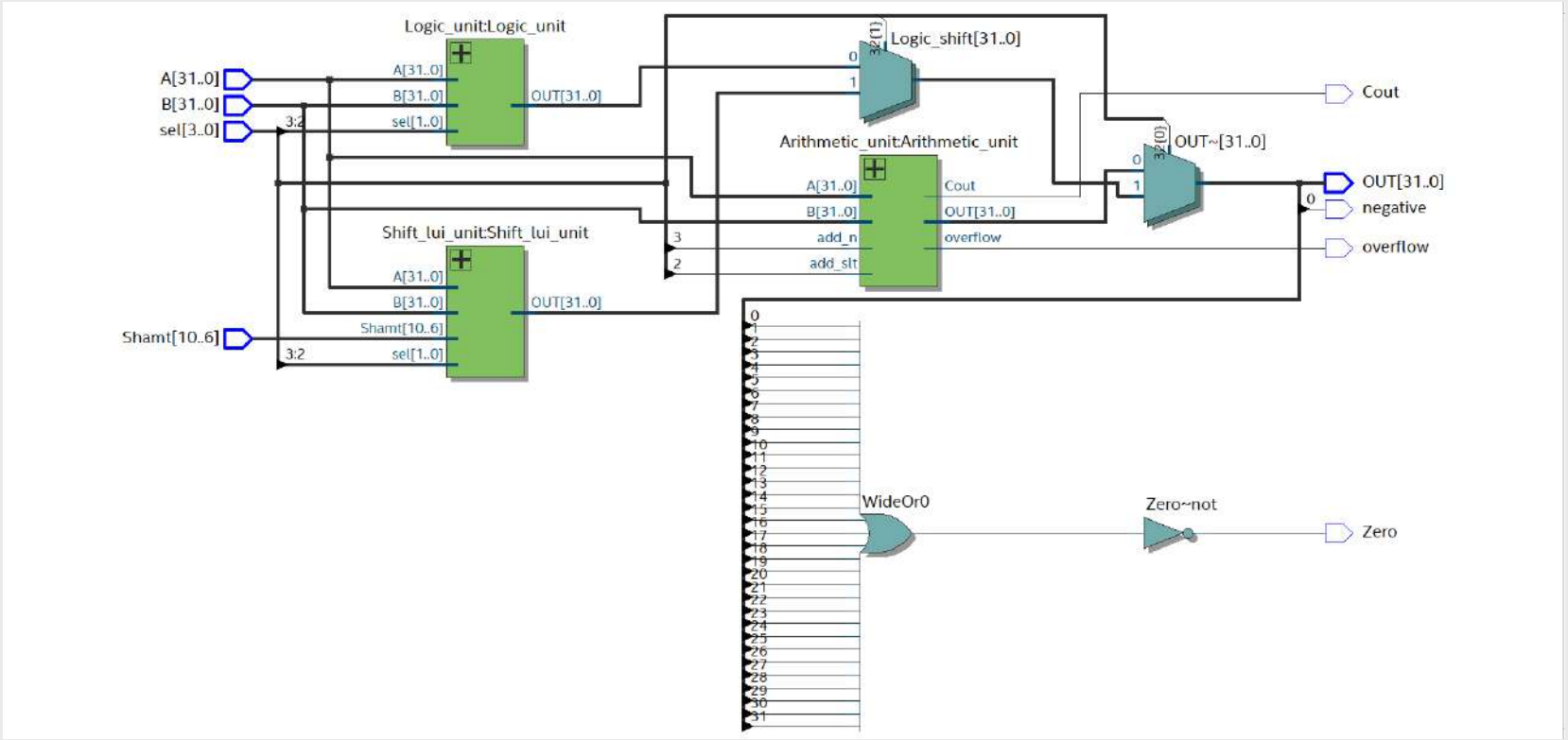
MIPS processor with debouncer (Technology Map Viewer (Post-Fitting))



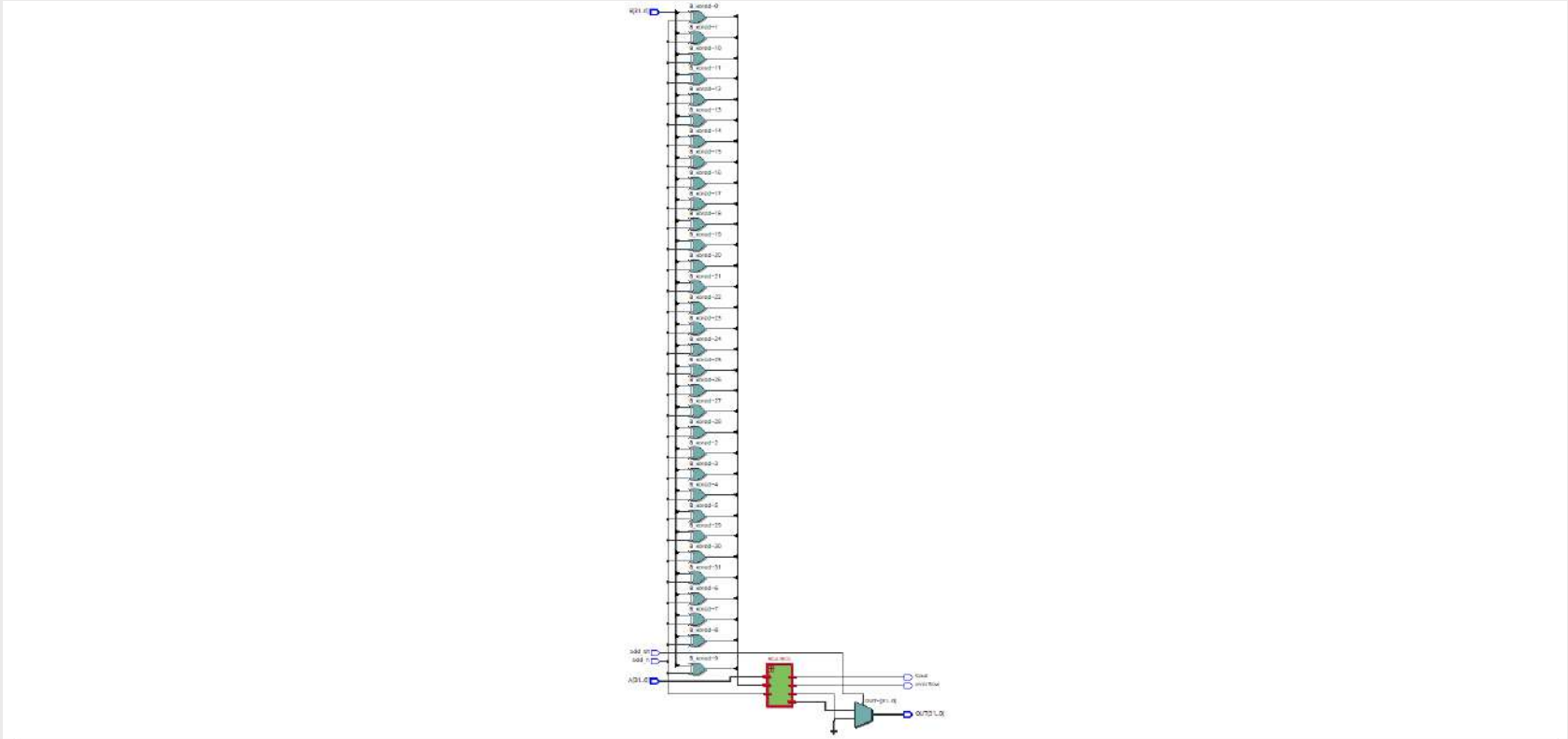
MIPS processor(RTL)



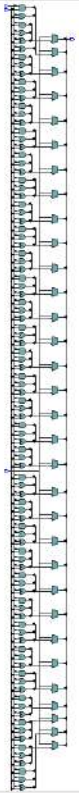
ALU (RTL)


Logic_unit:Logic_unit
A[31..0]
B[31..0]
sel[3..0]
3:2
A[31..0]
B[31..0]
sel[1..0]
OUT[31..0]
Logic_shift[31..0]
32 [1]
0
1
Arithmetic_unit:Arithmetic_unit
A[31..0]
B[31..0]
add_n
add_slt
3
2
Cout
OUT[31..0]
overflow
OUT~[31..0]
32 [0]
0
1
Cout
OUT[31..0]
negative
overflow
Shift_lui_unit:Shift_lui_unit
A[31..0]
B[31..0]
Shamt[10..6]
sel[1..0]
3:2
OUT[31..0]
Shamt[10..6]
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
WideOr0
Zero~not
Zero

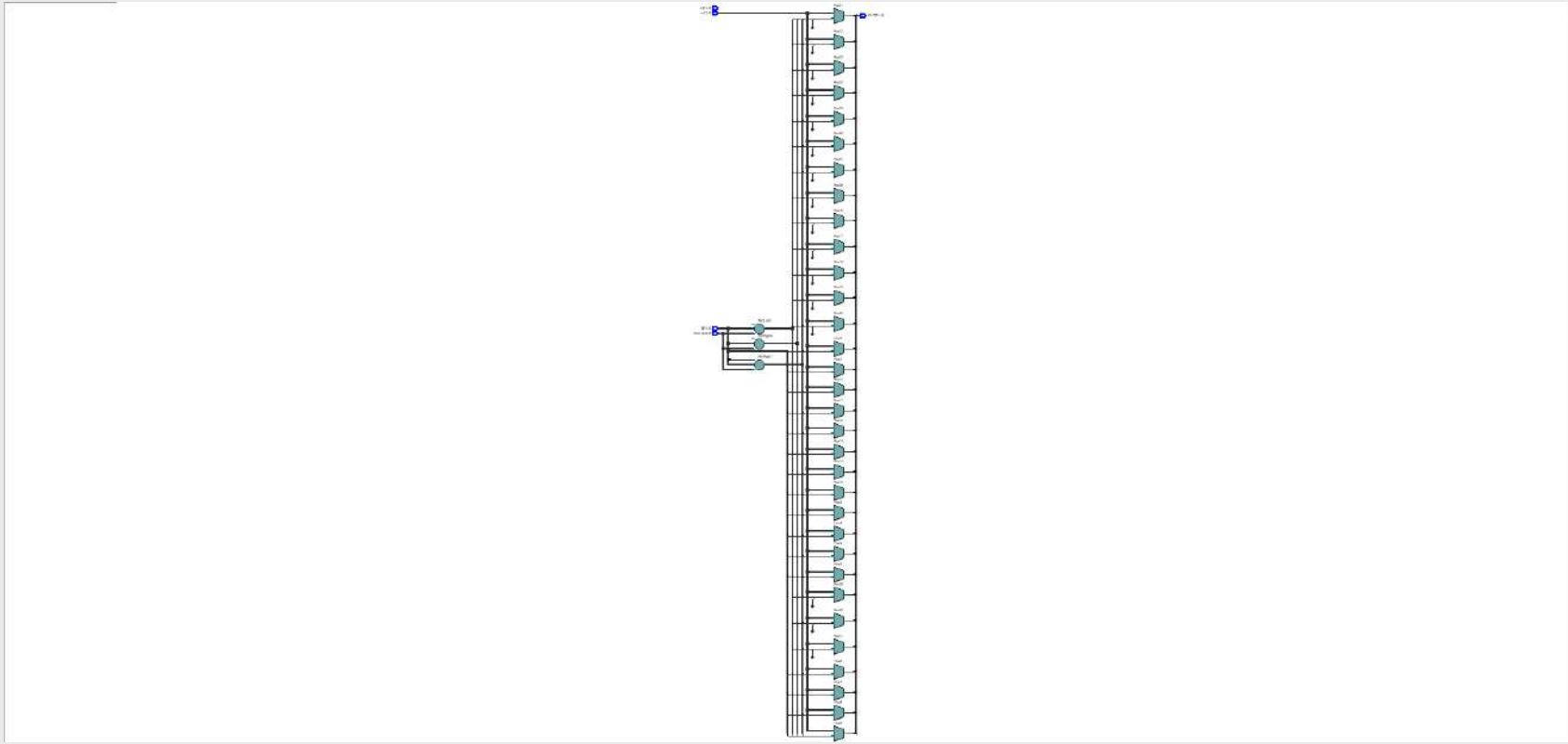
Arithmetic unit (RTL)



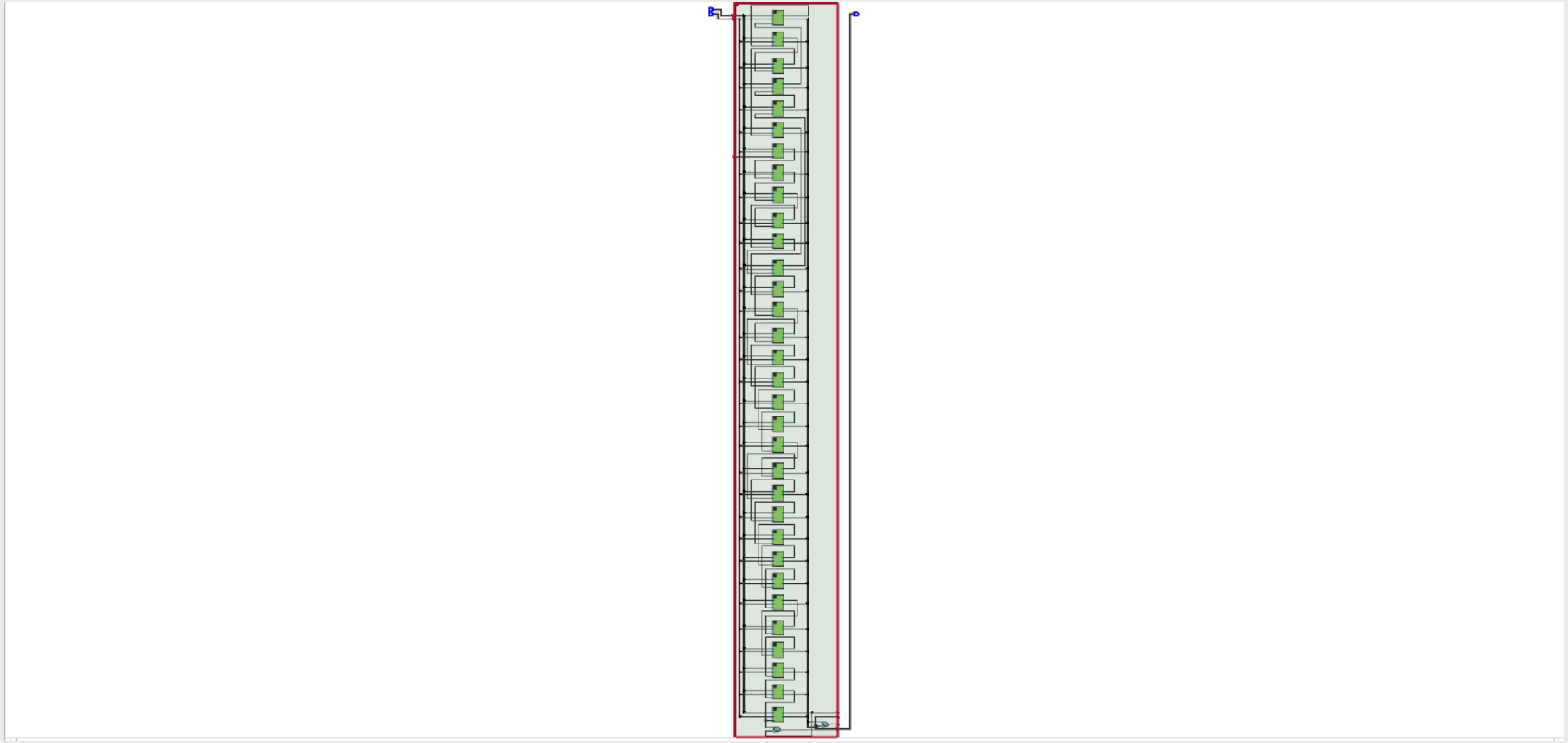
Logic unit(RTL)



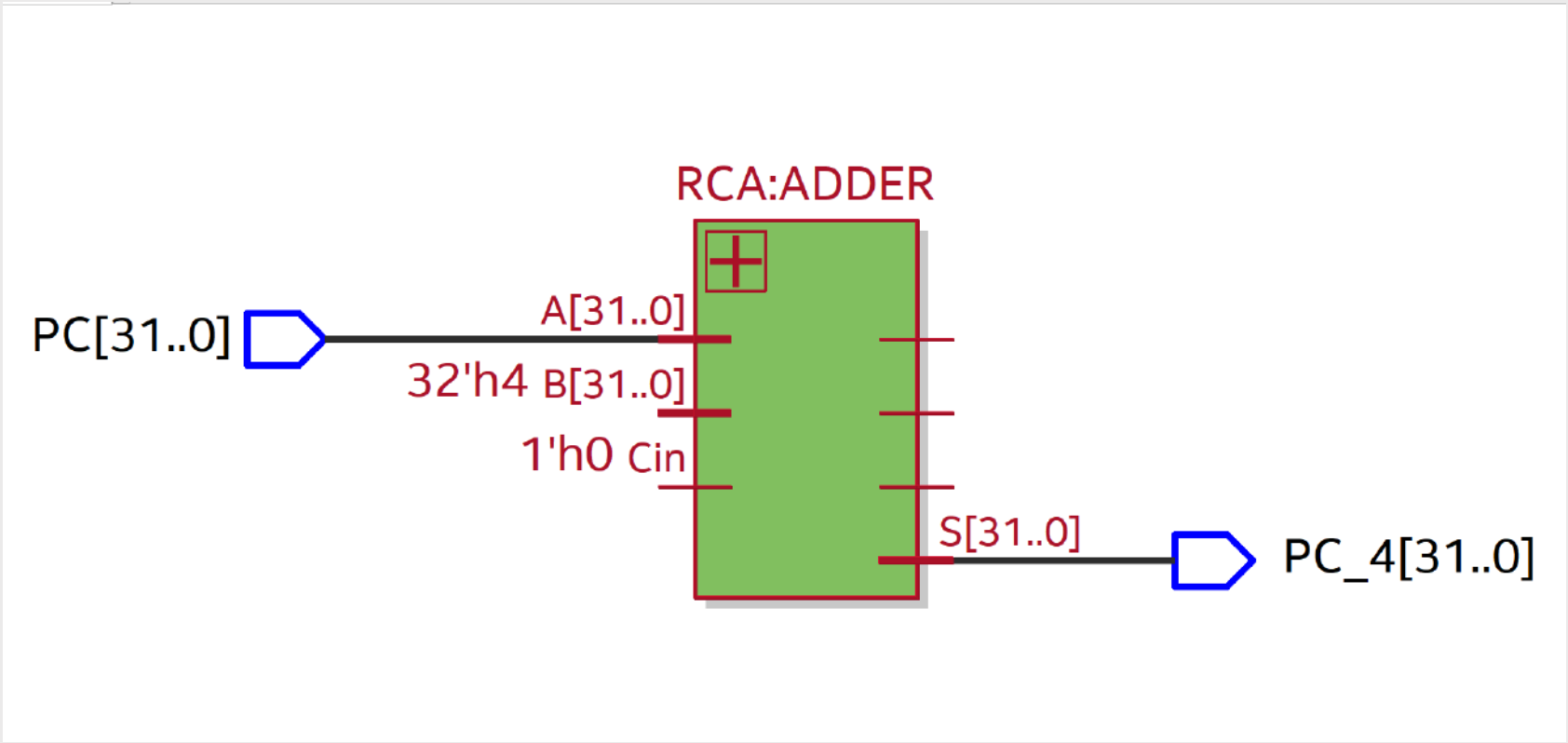
Shift lui unit(RTL)



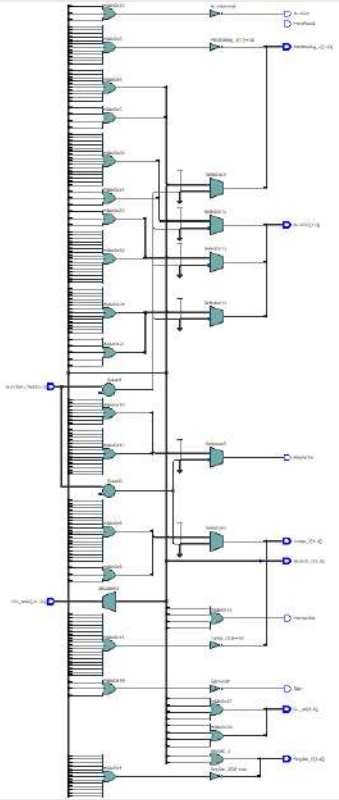
Adder (RTL)



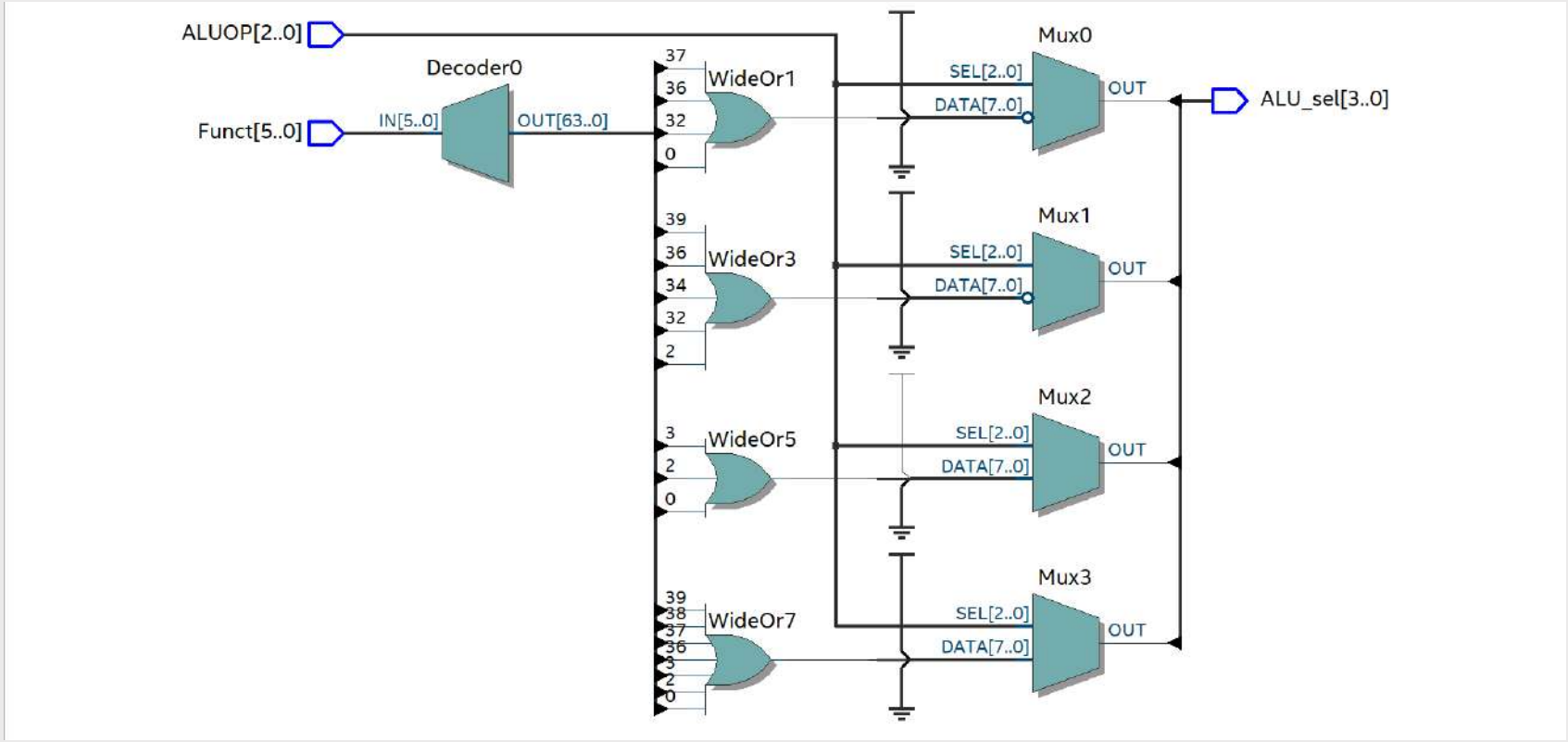
Add 4 (RTL)



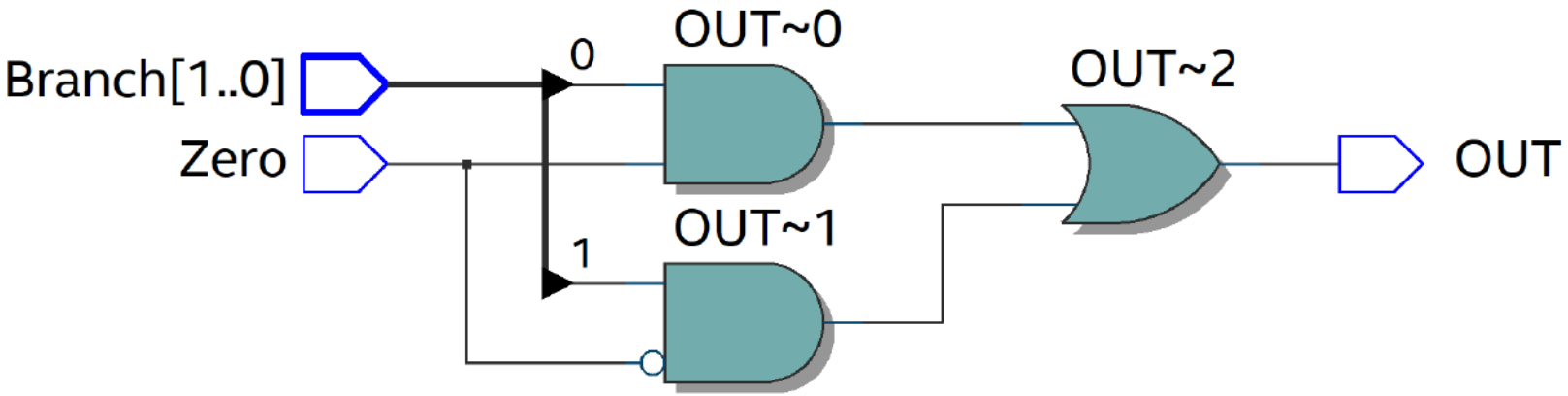
Control (RTL)



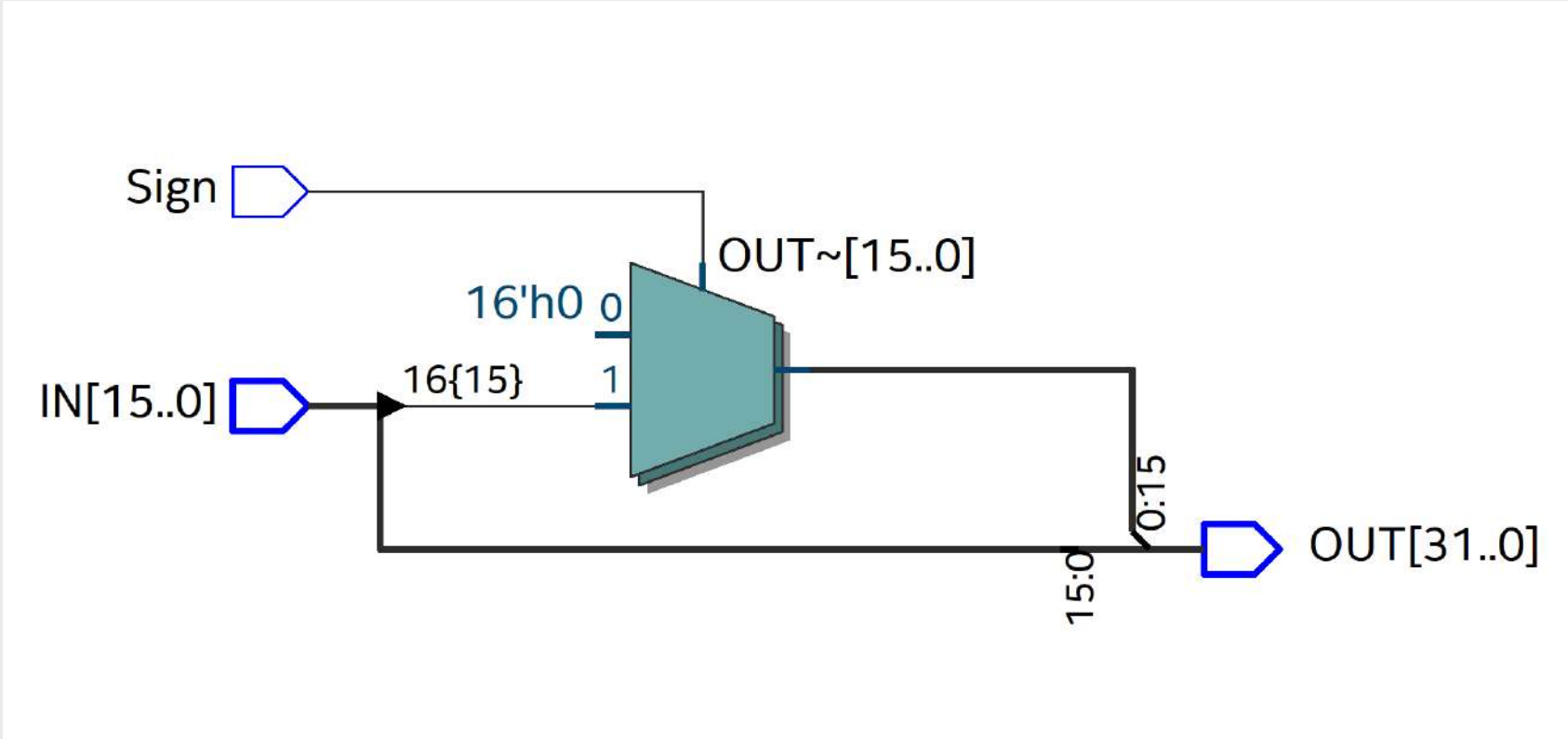
ALU control (RTL)



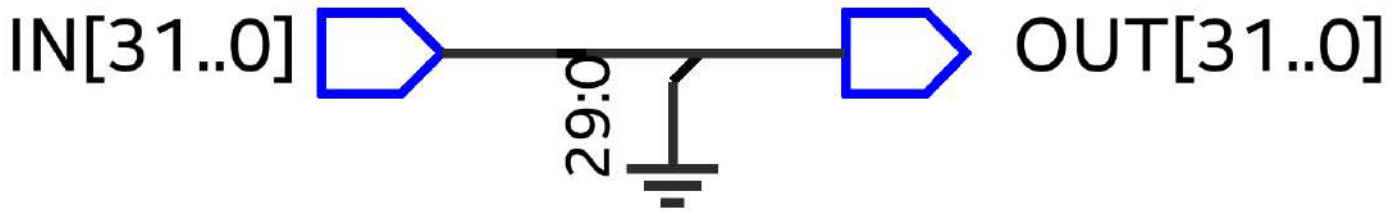
Branch control (RTL)



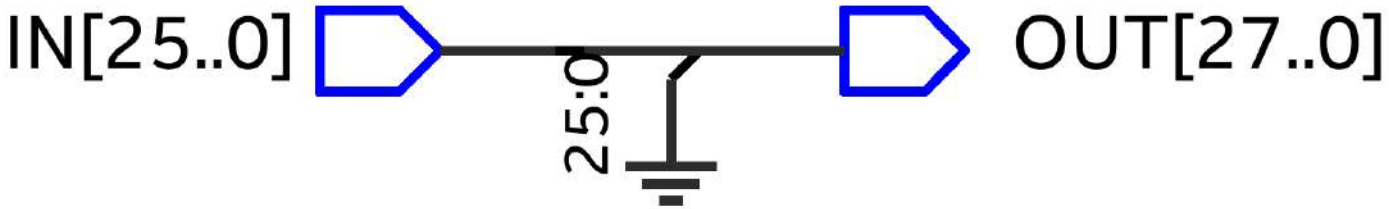
Extend unit (RTL)



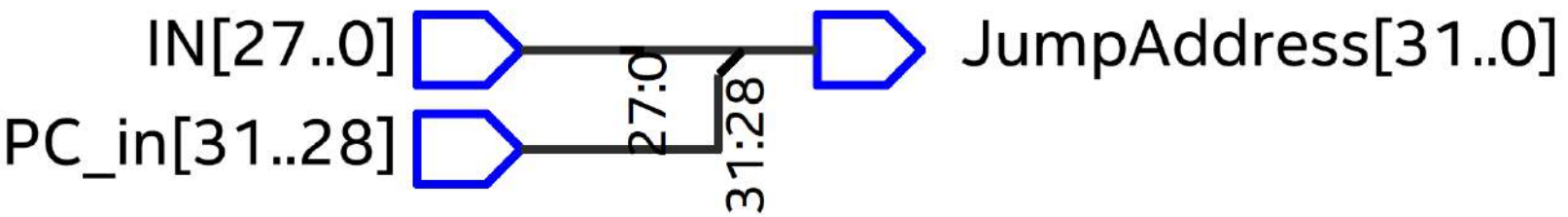
Shift left2 32:32(RTL)



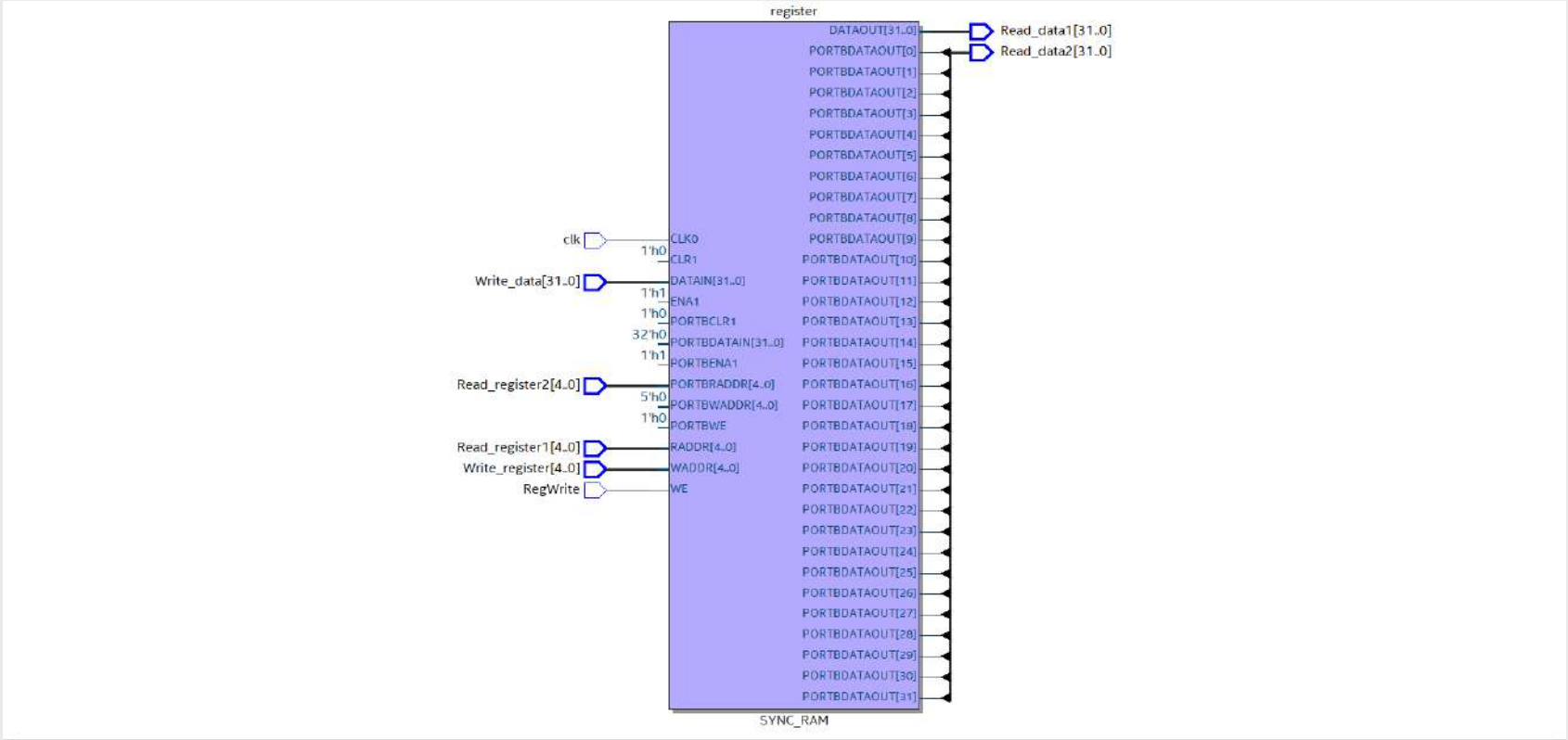
Shift left2 26:28(RTL)



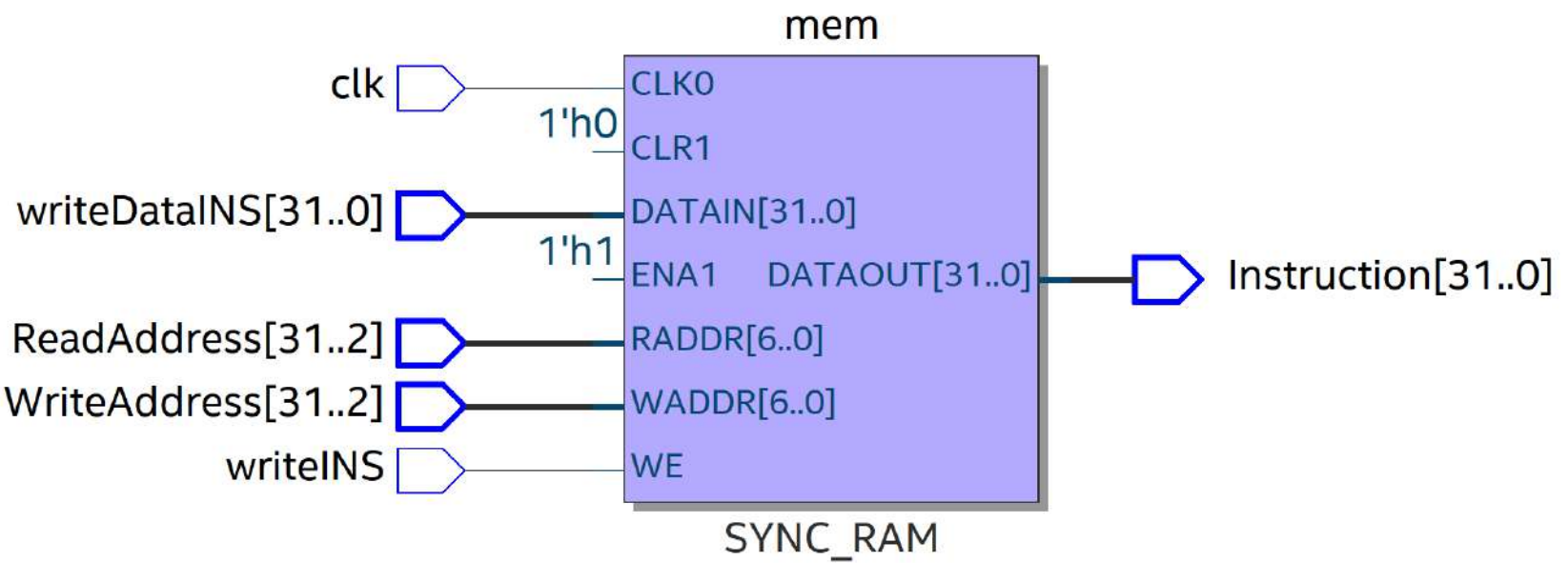
Concatenation unit (RTL)



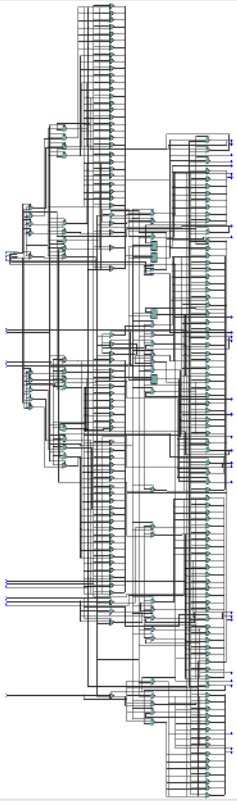
Registers (RTL)



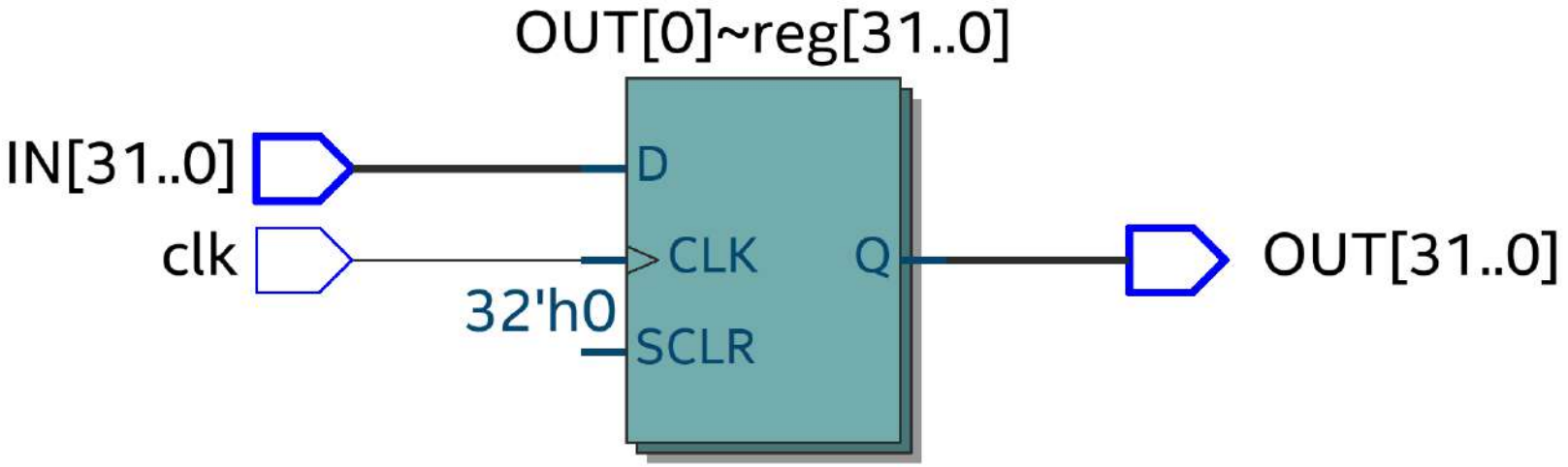
Instruction memory (RTL)



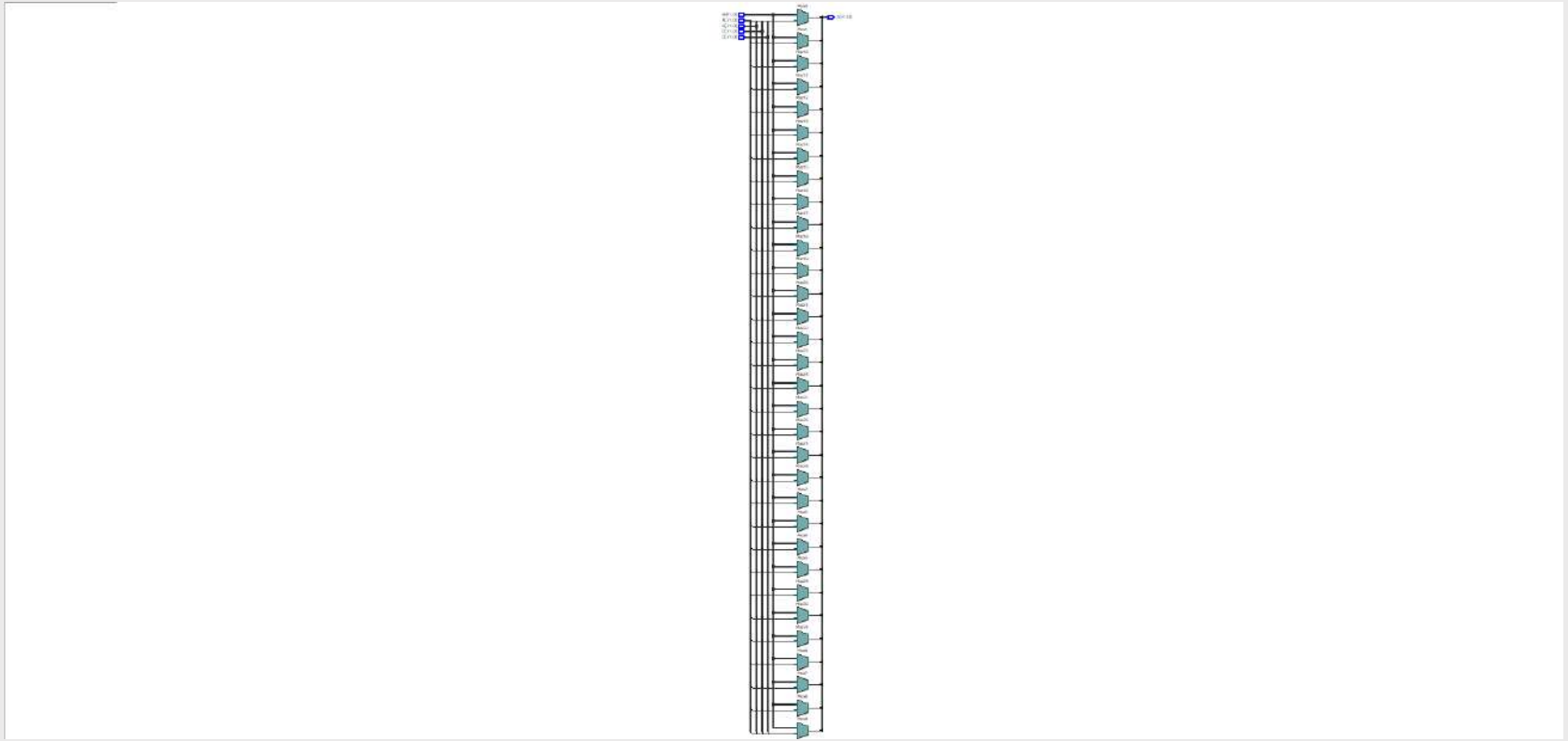
Data memory(RTL)



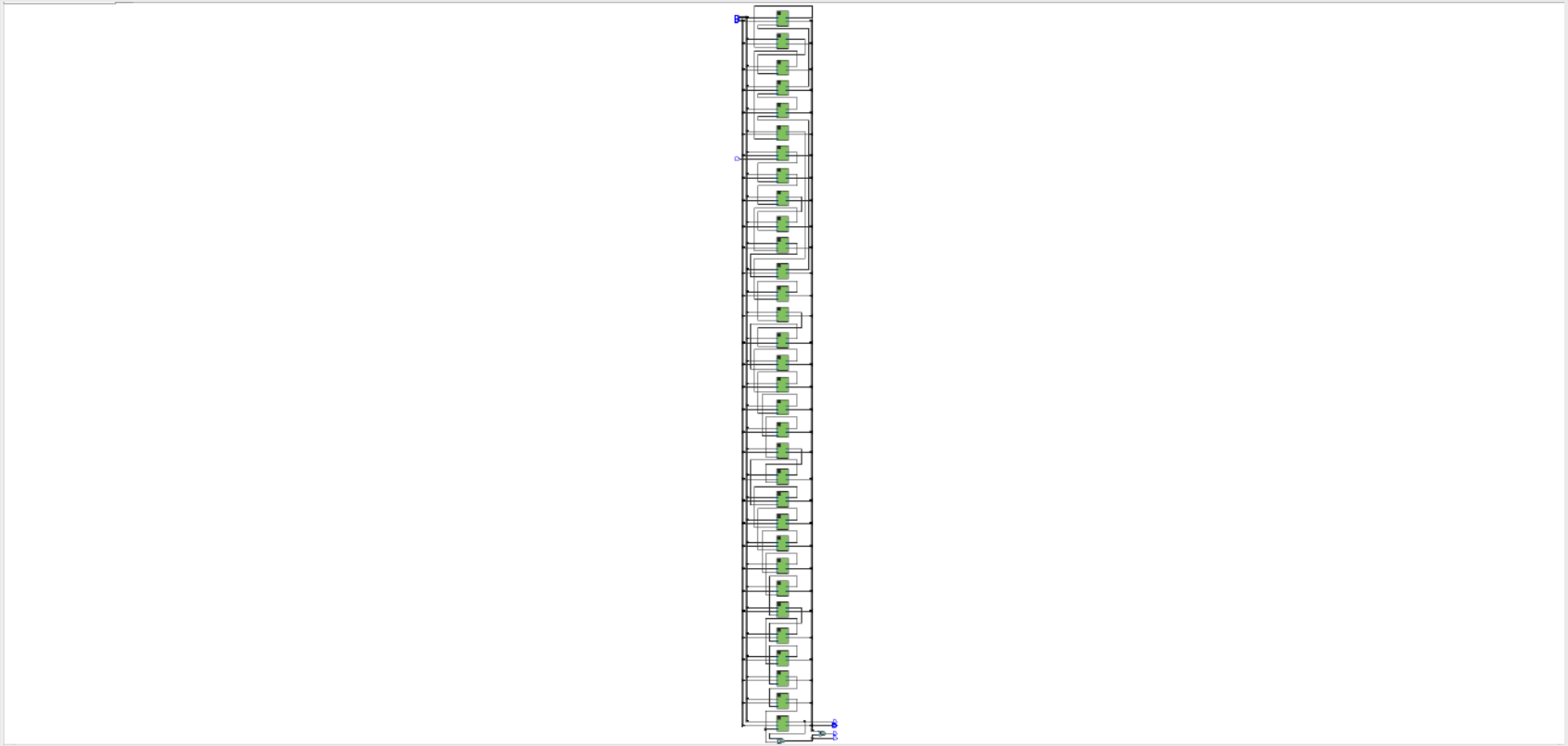
PC (RTL)



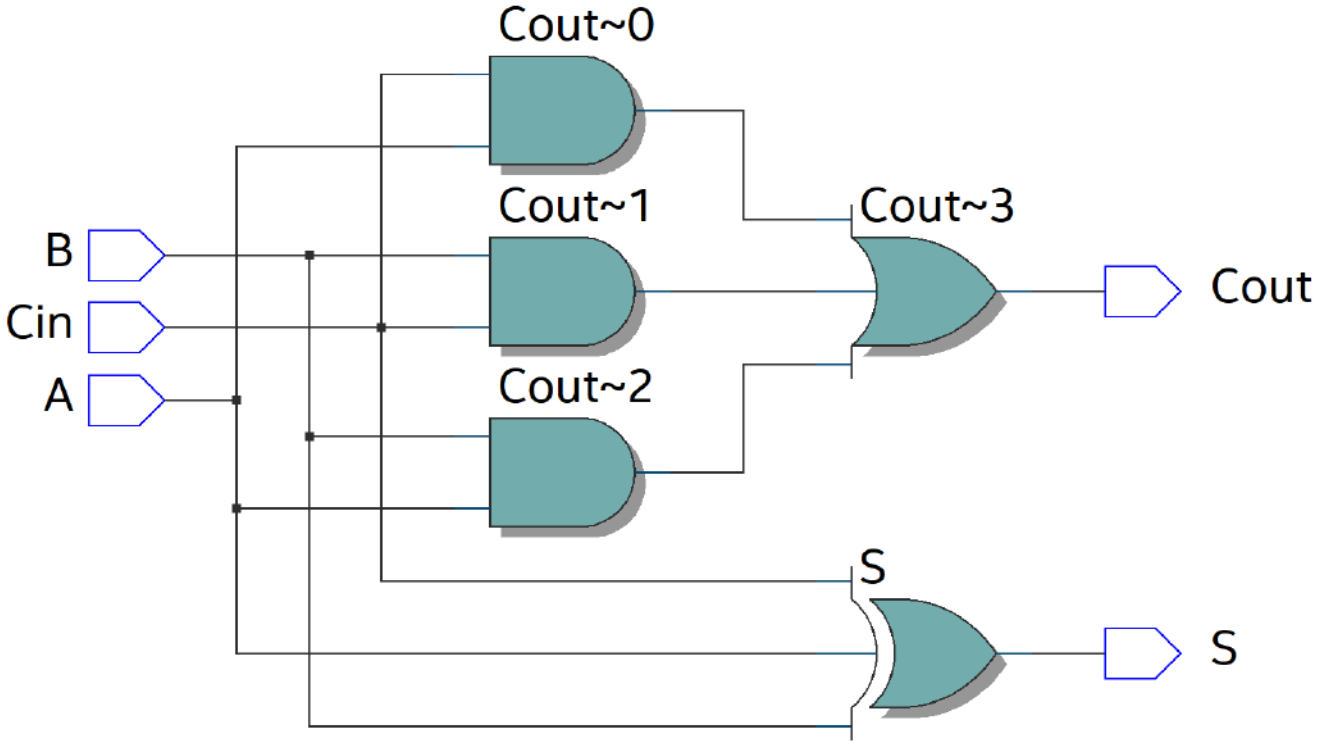
MUX 4:1 (RTL)



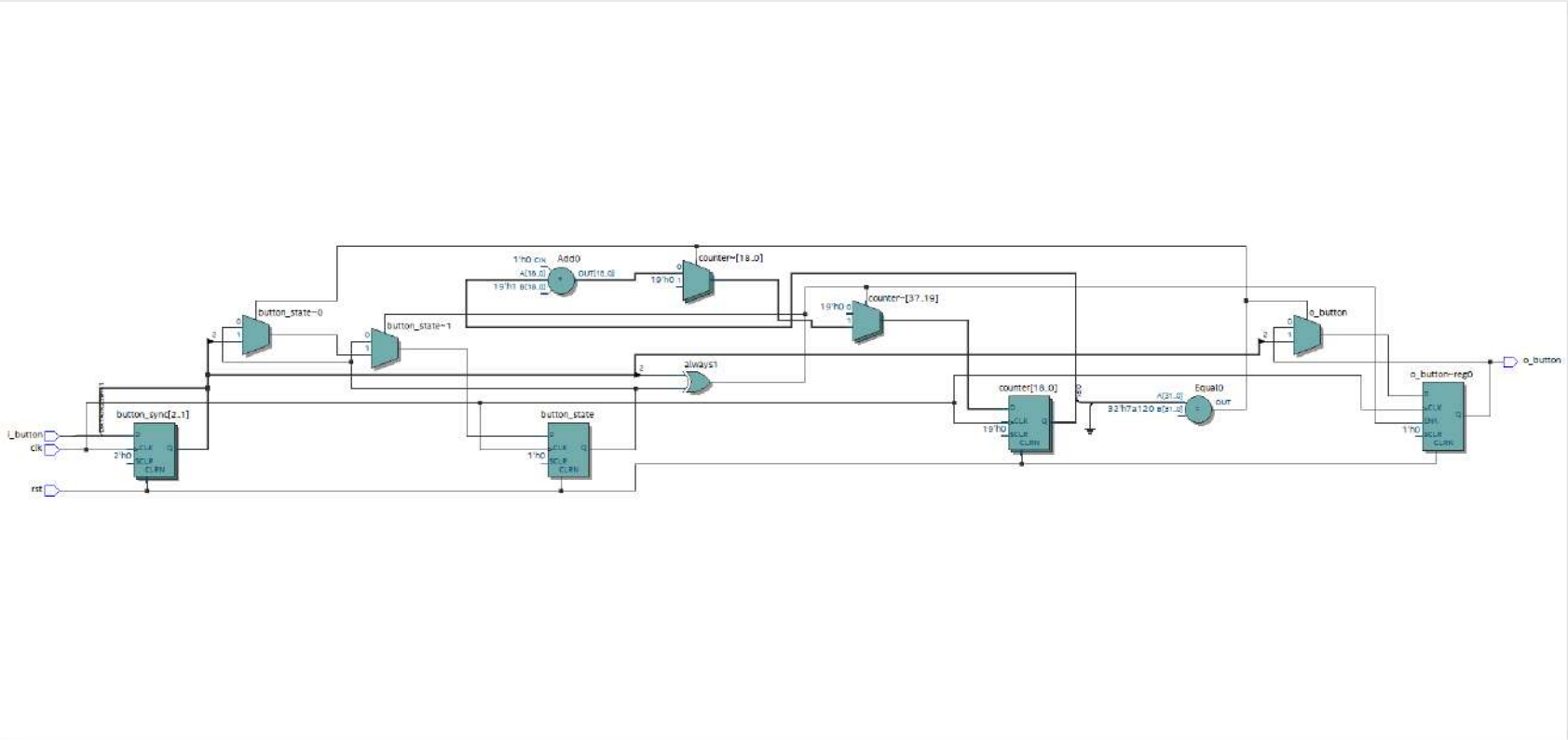
RCA (RTL)



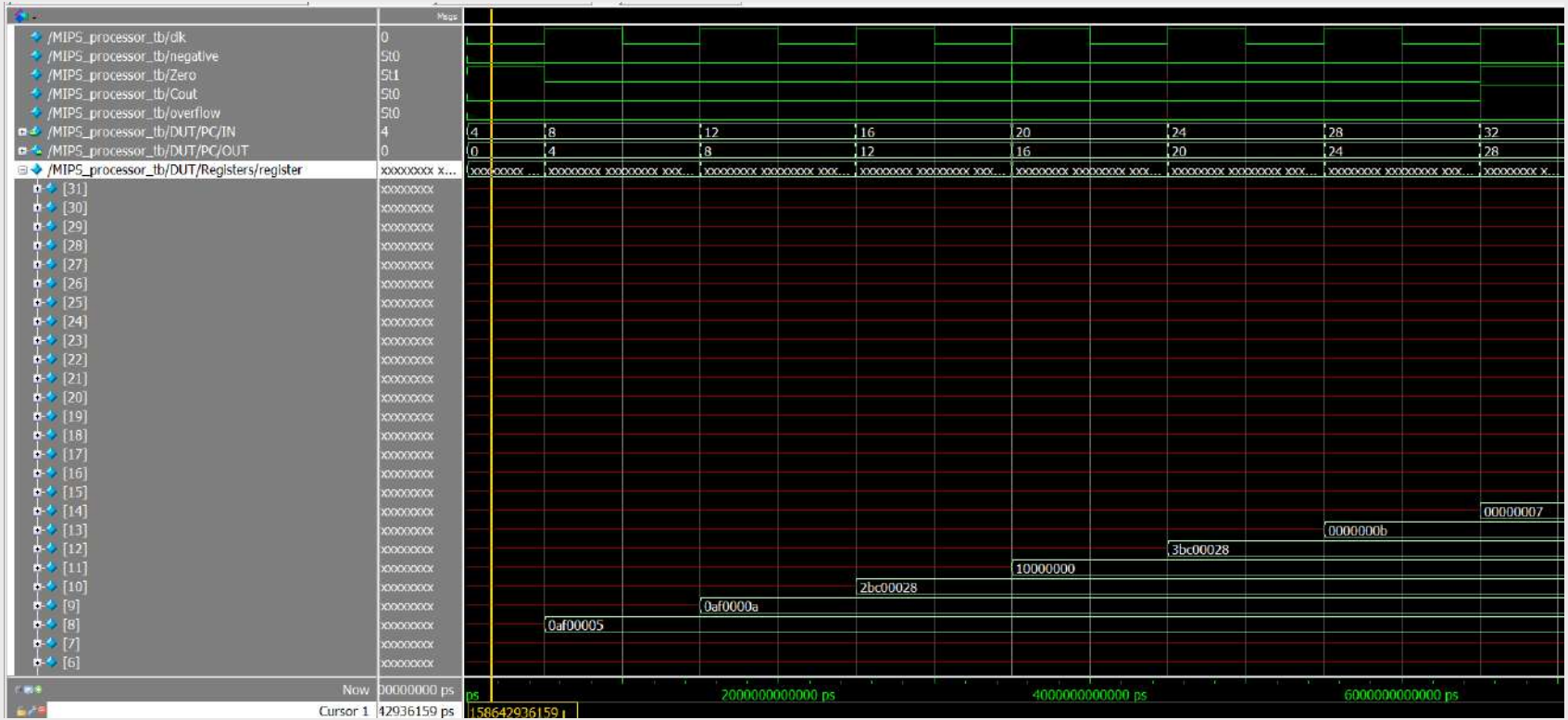
FA (RTL)



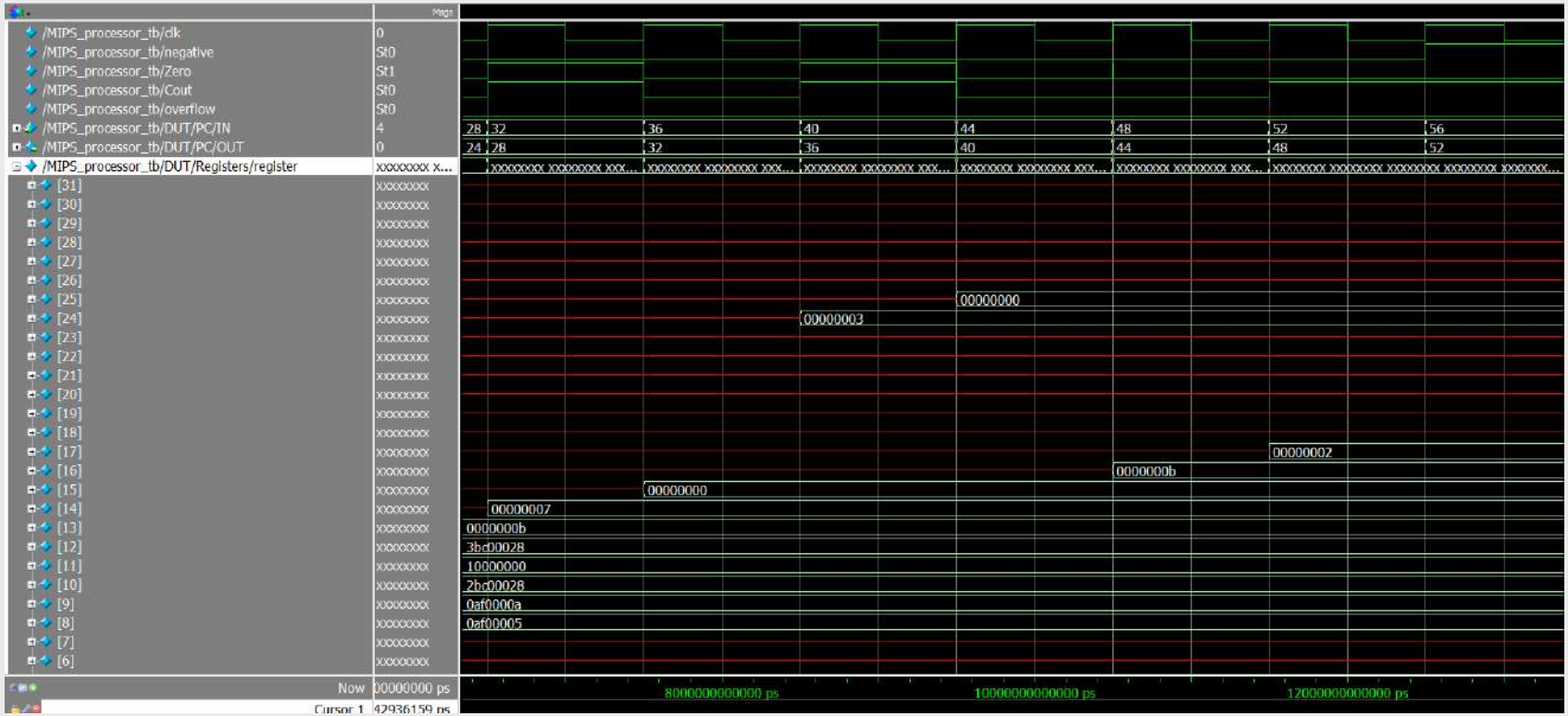
Debouncer (RTL)



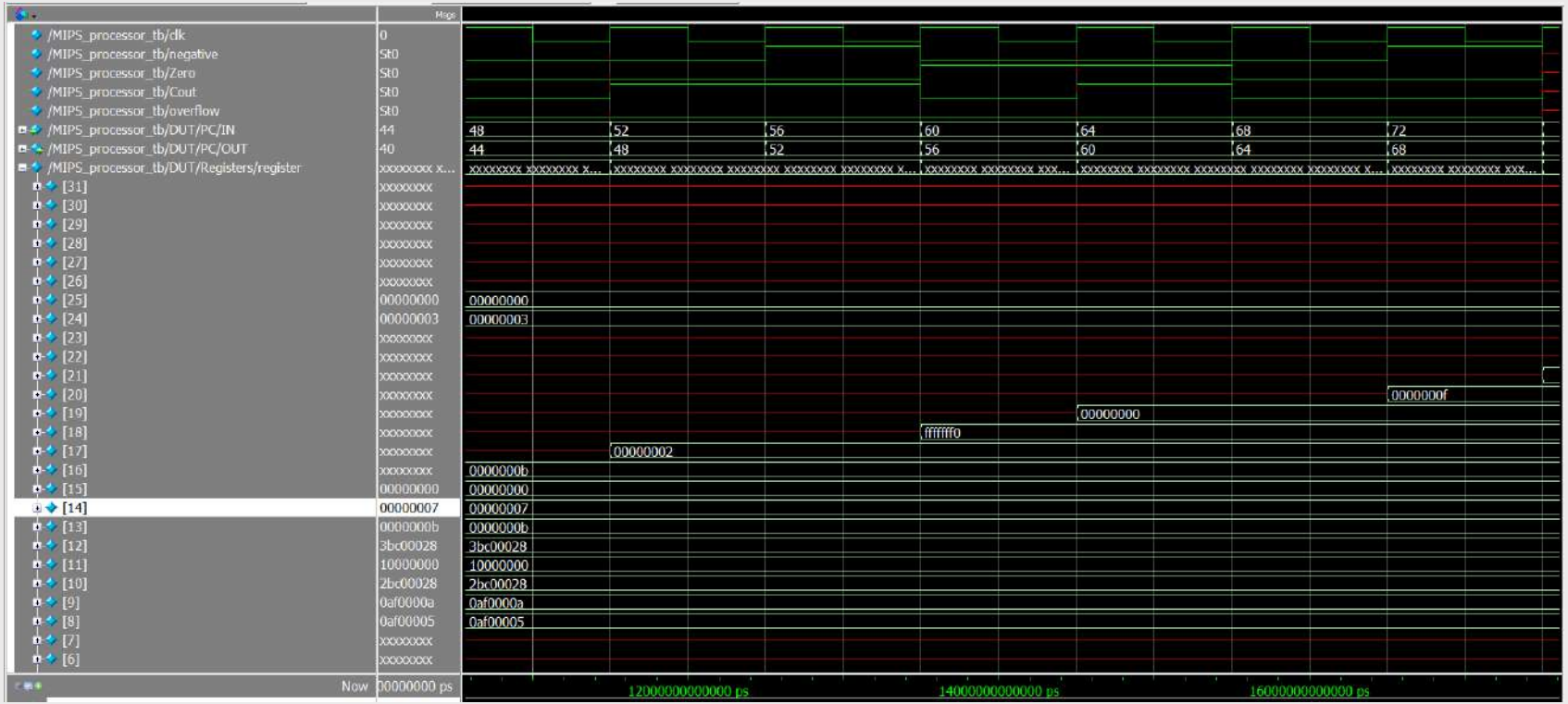
MIPS_processor_tb (ModelSIM simulation)



MIPS_processor_tb (ModelSIM simulation)

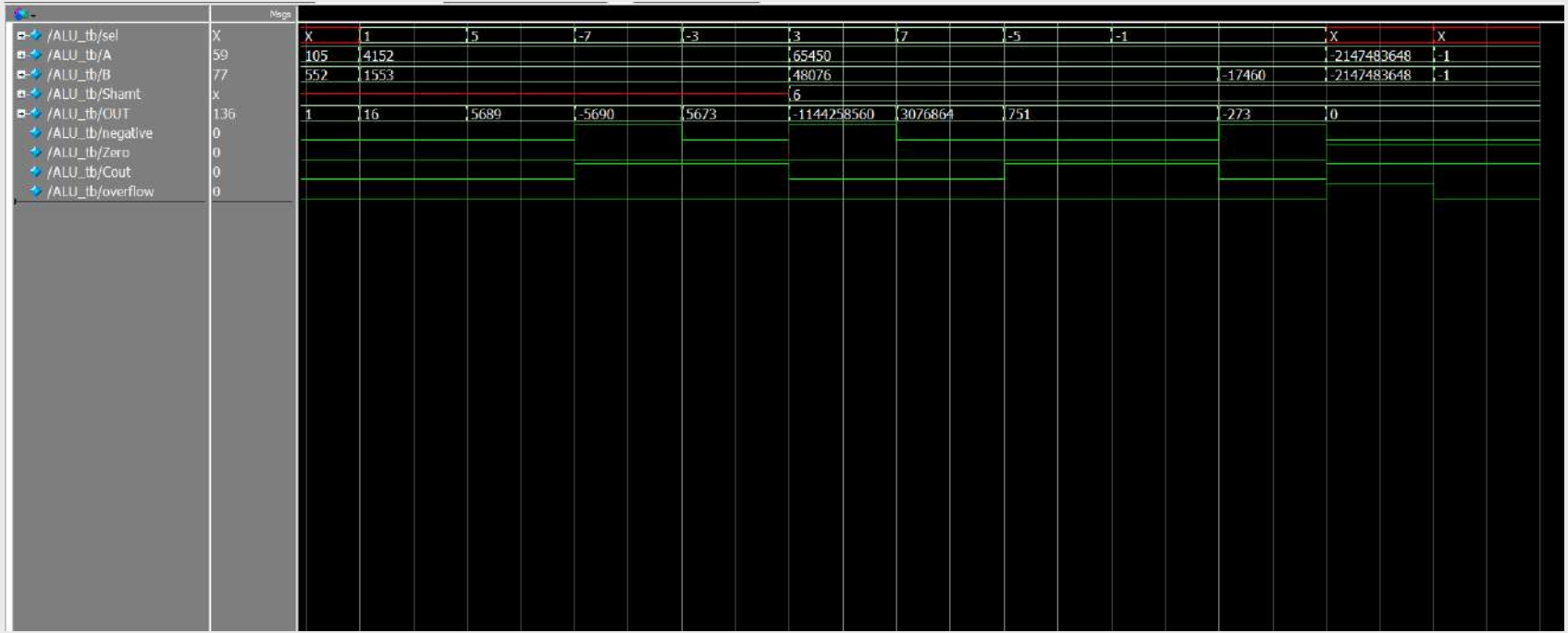


MIPS_processor_tb (ModelSIM simulation)

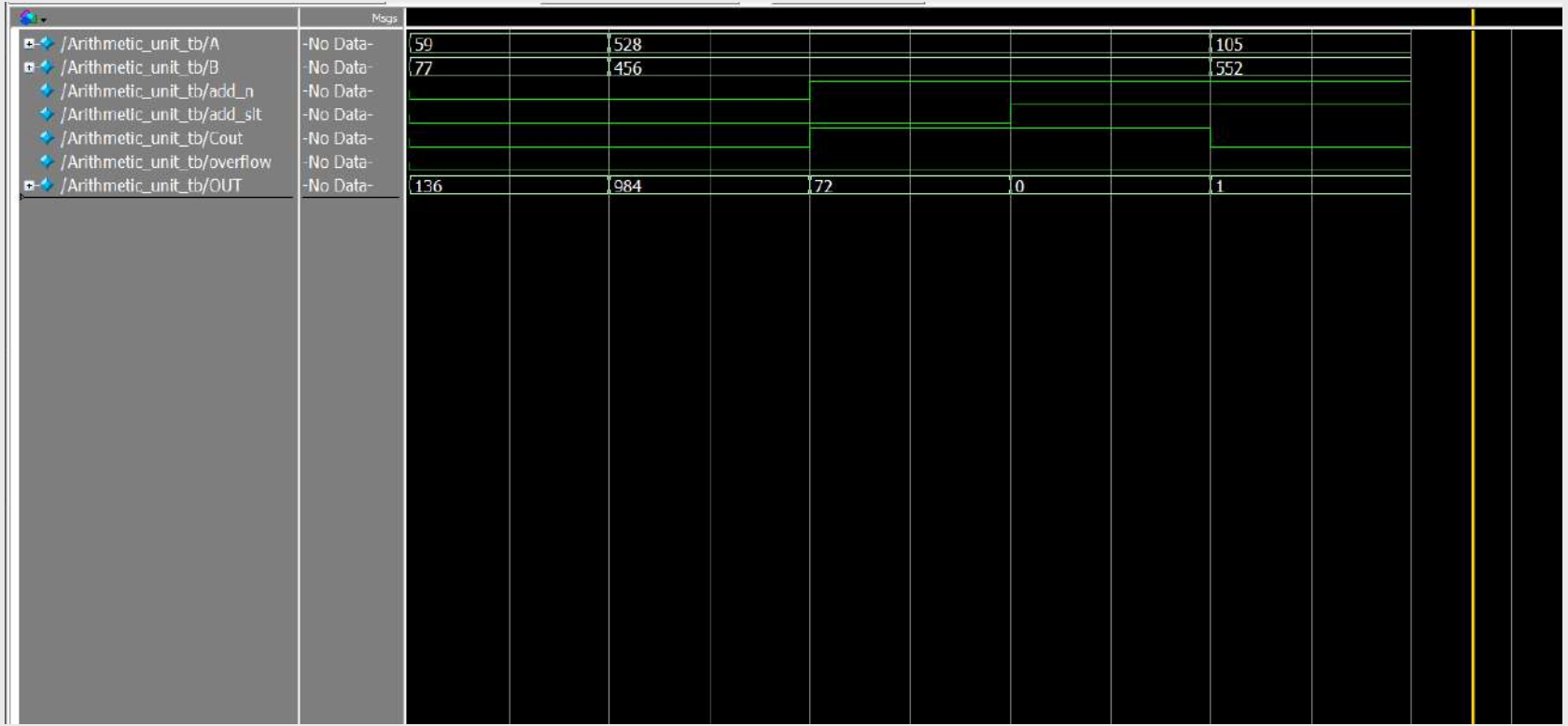


MIPS processor tb (ModelSIM simulation)

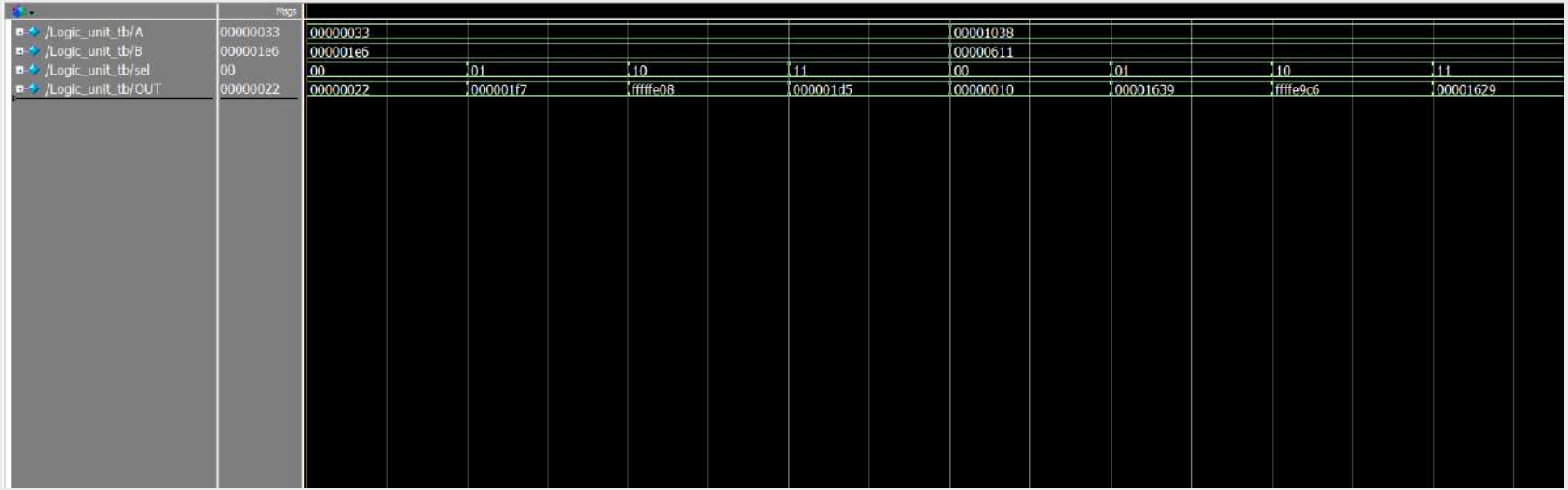
ALU_tb (ModelSIM simulation)



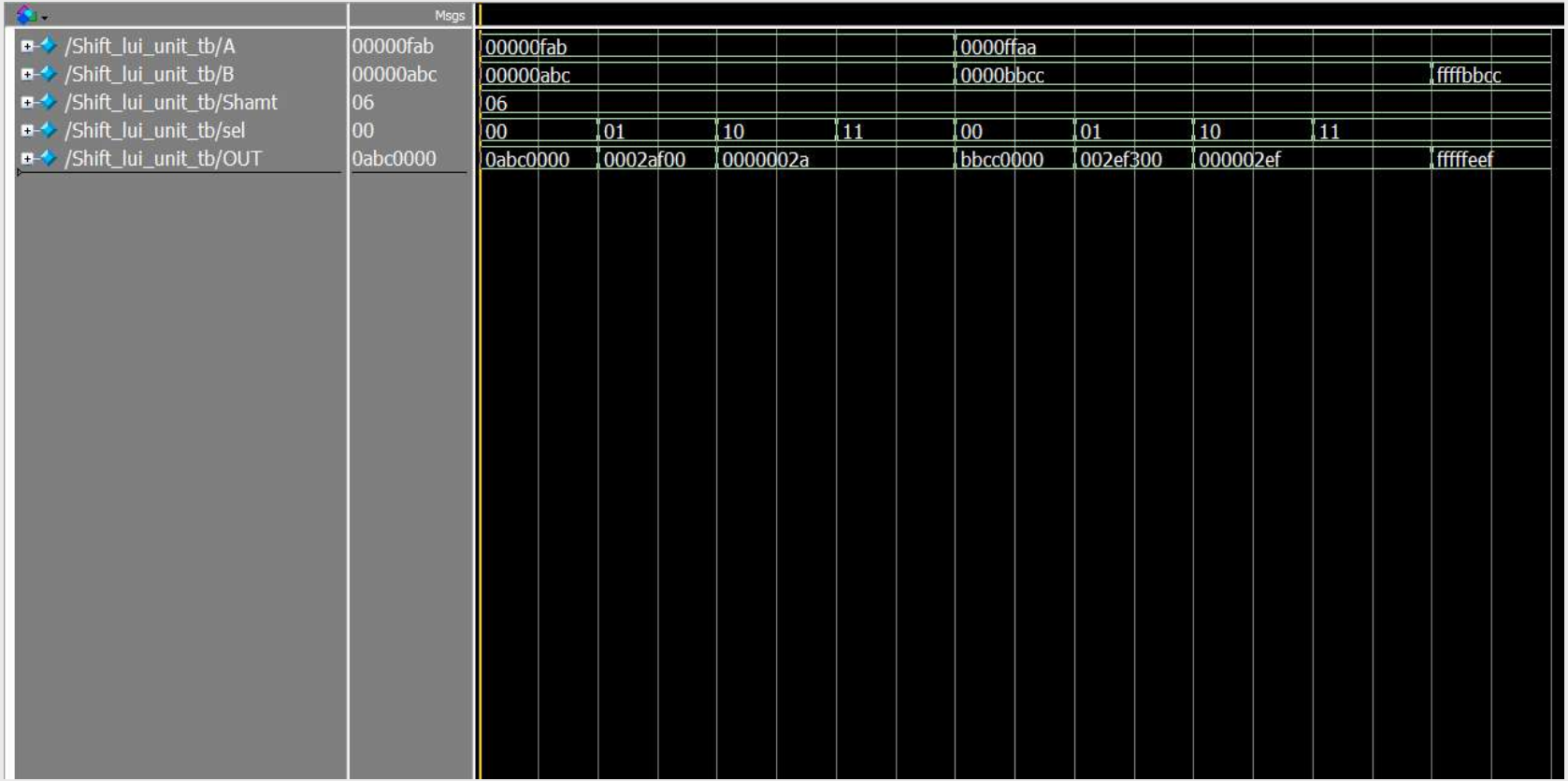
Arithmetic unit tb(ModelSIM simulation)



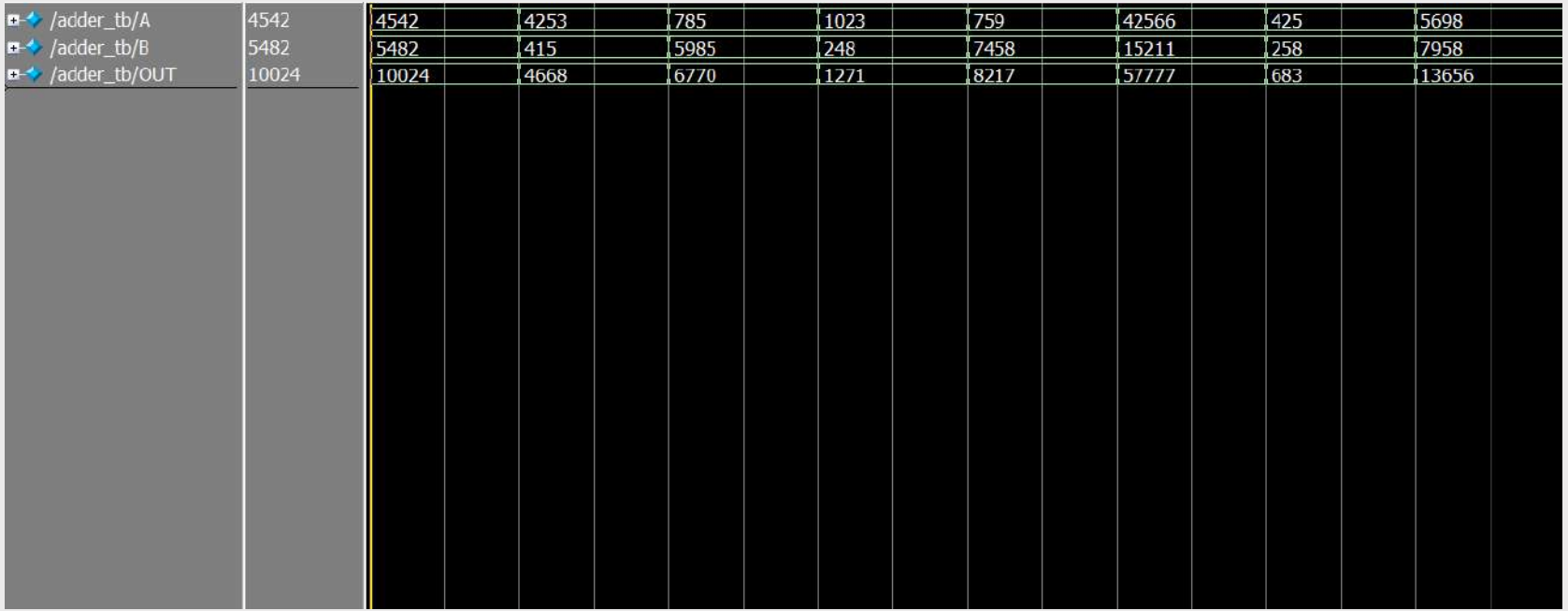
Logic unit tb(ModelSIM simulation)



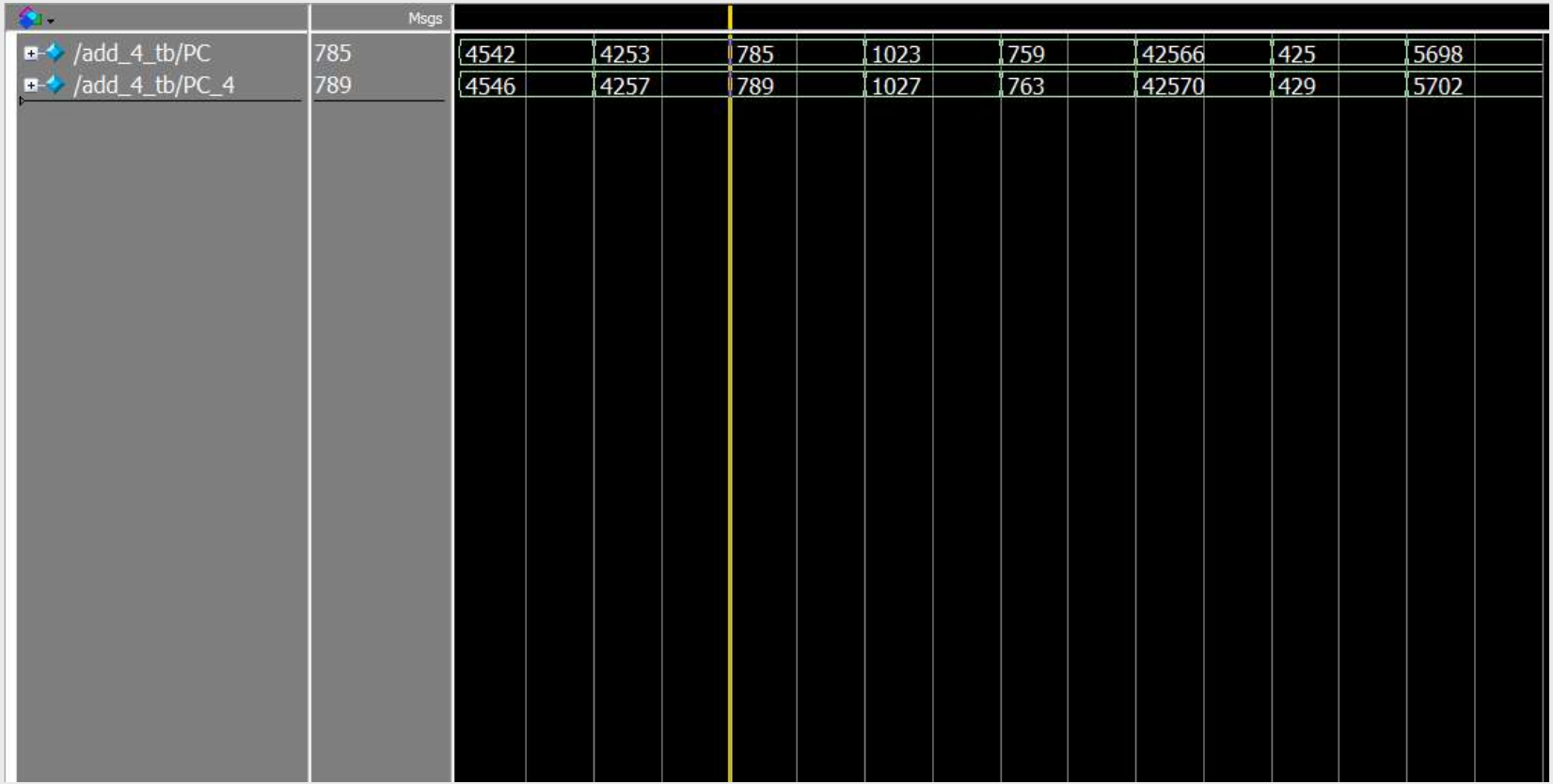
Shift_lui_unit_tb (ModelSIM simulation)



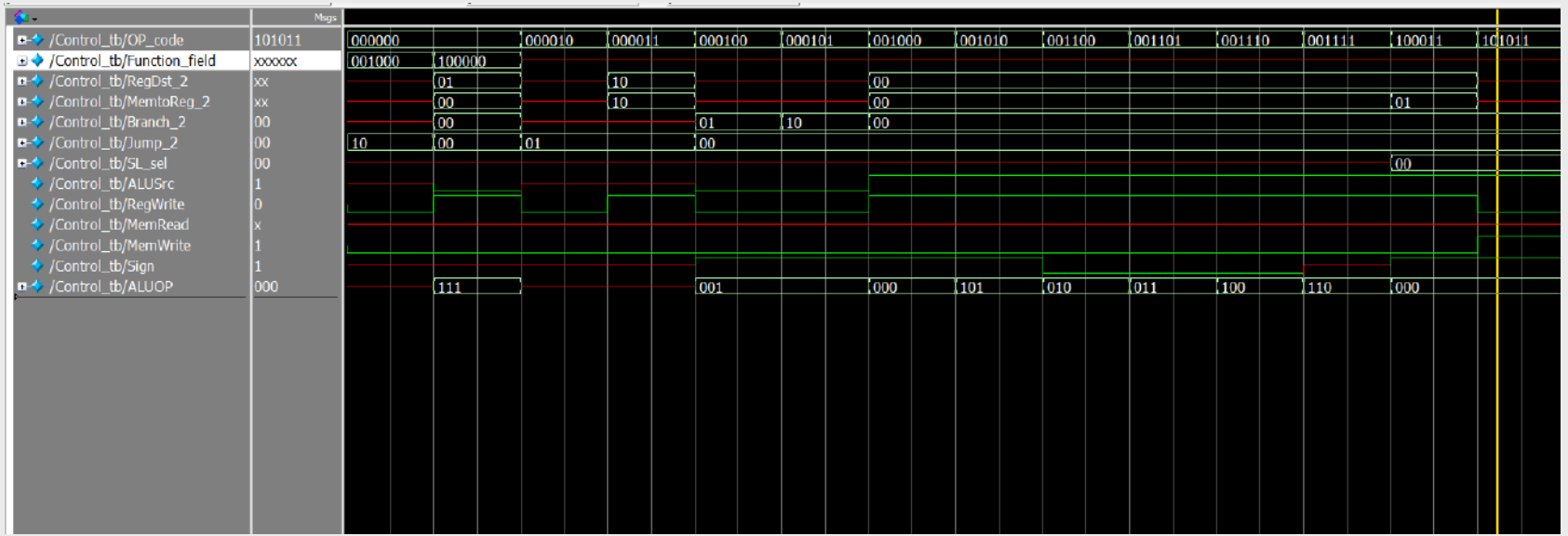
Adder tb (ModelSIM simulation)



Add 4 tb(ModelSIM simulation)

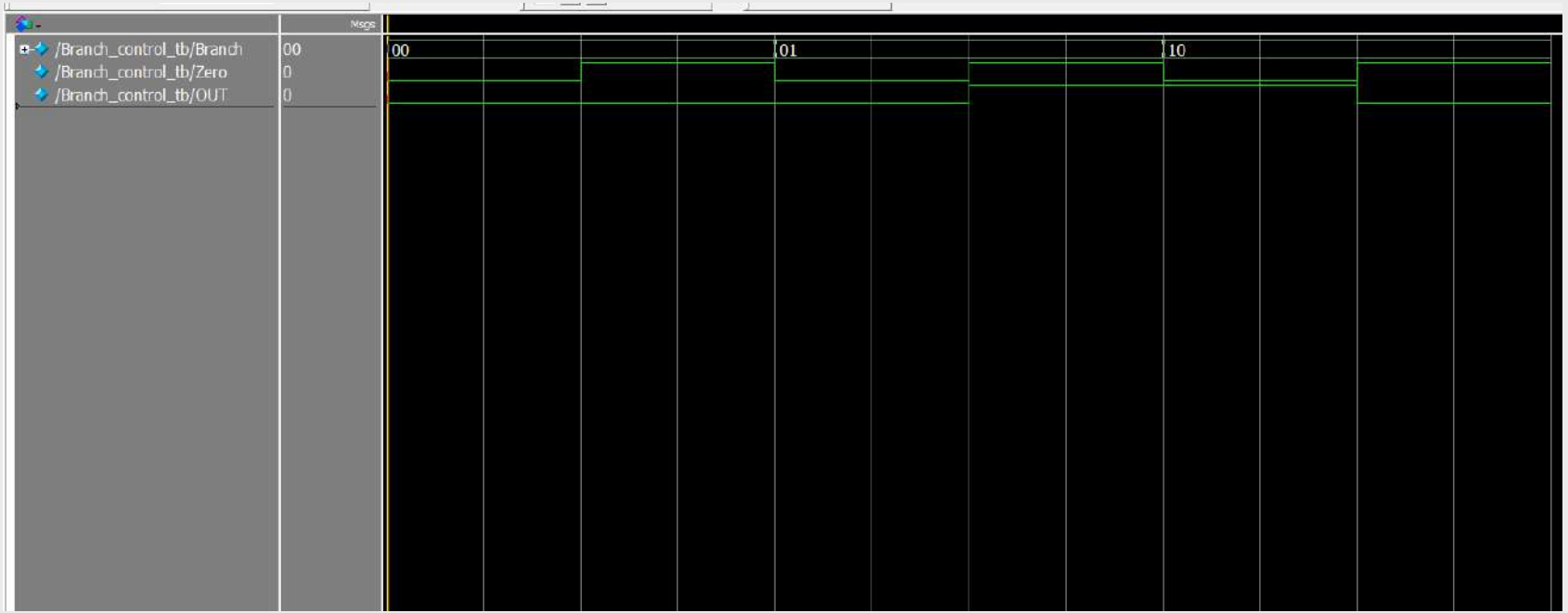


Control_tb (ModelSIM simulation)

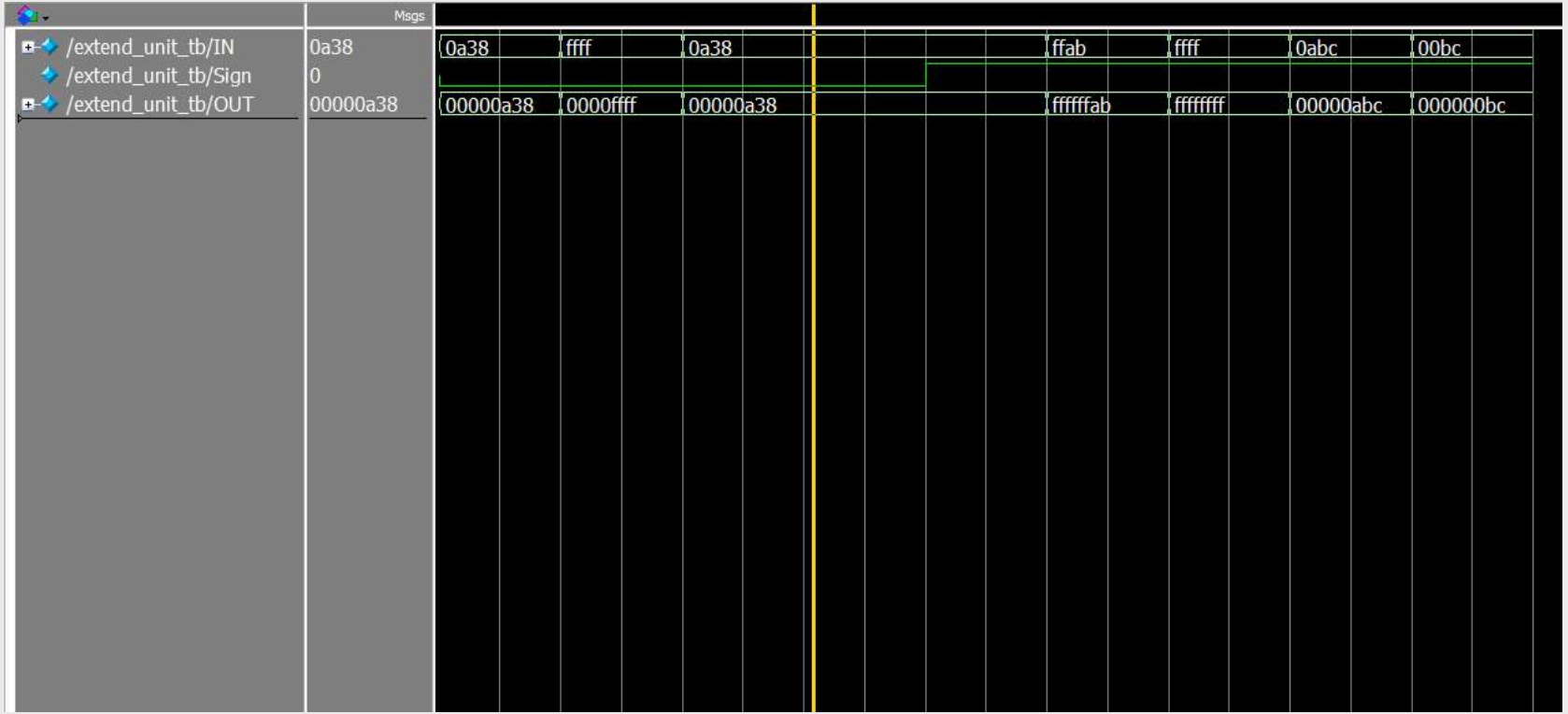


ALU control tb (ModelSIM simulation)

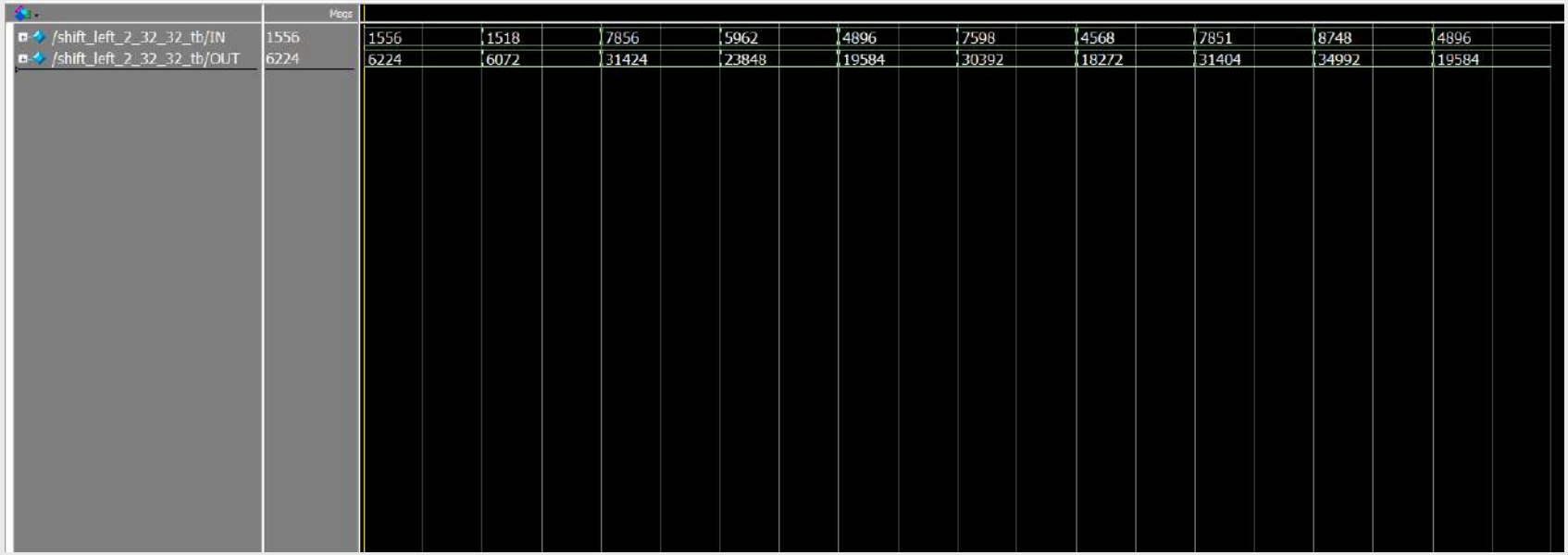
Branch_control_tb (ModelSIM simulation)



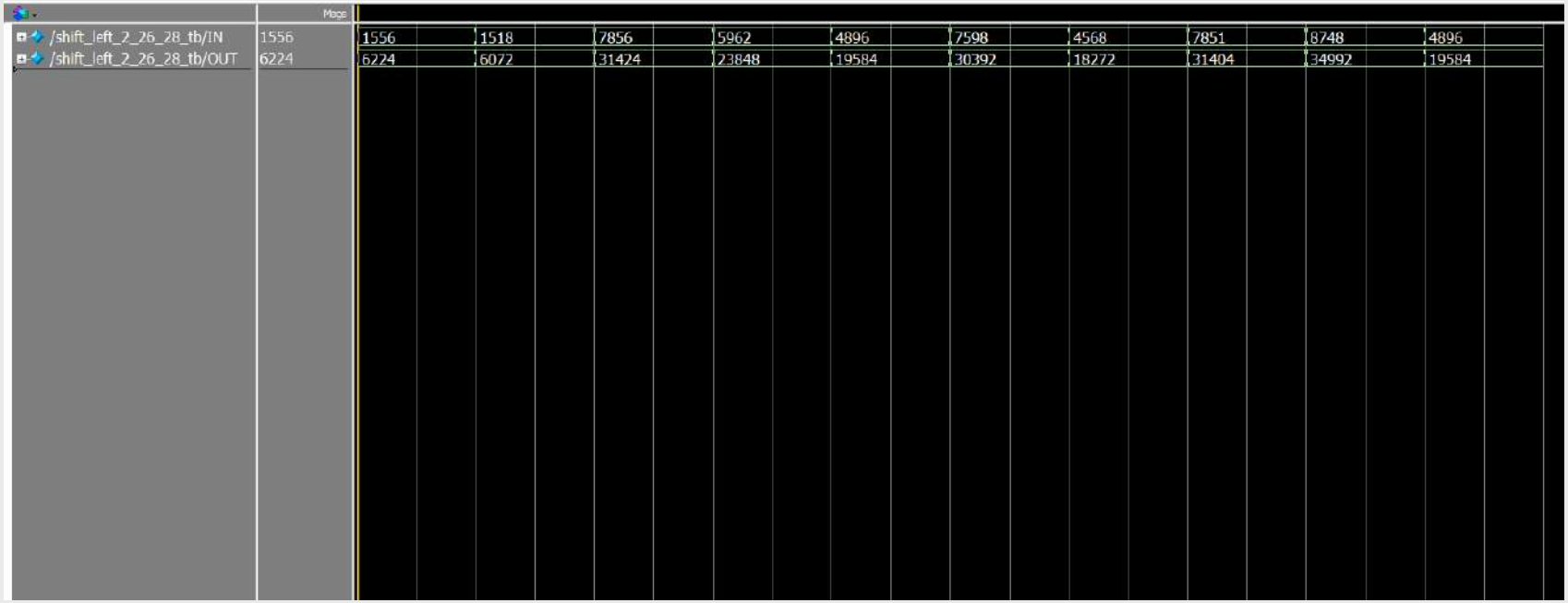
Extend unit tb (ModelSIM simulation)



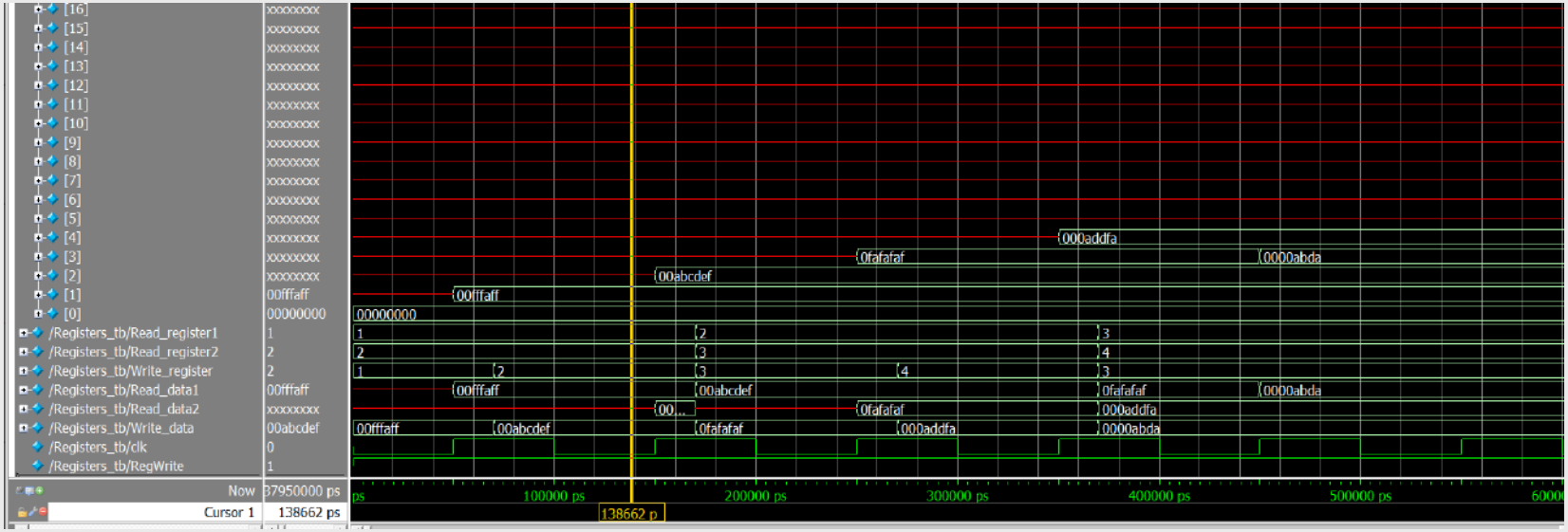
Shift_left2_32:32_tb (ModelSIM simulation)



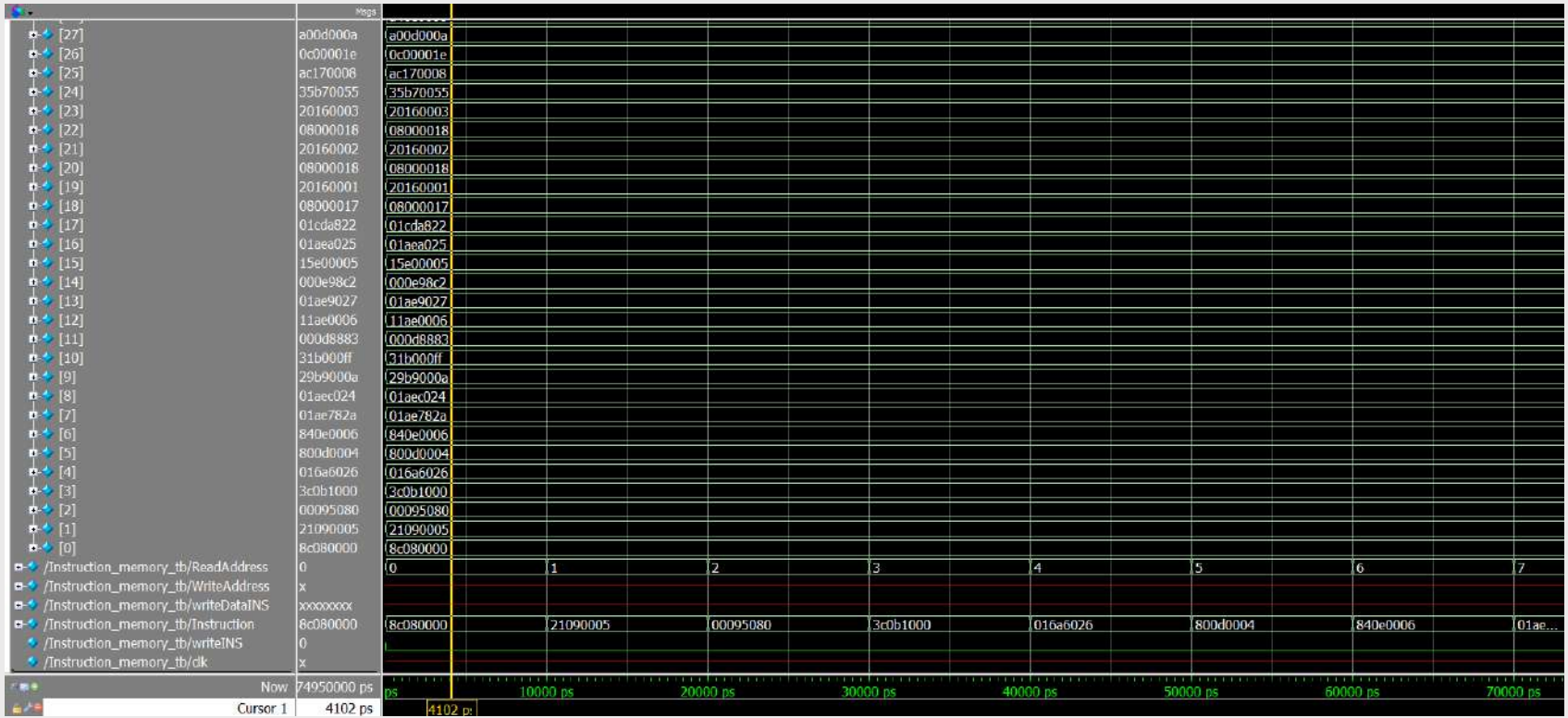
Shift_left2_26:28_tb(ModelSIM simulation)



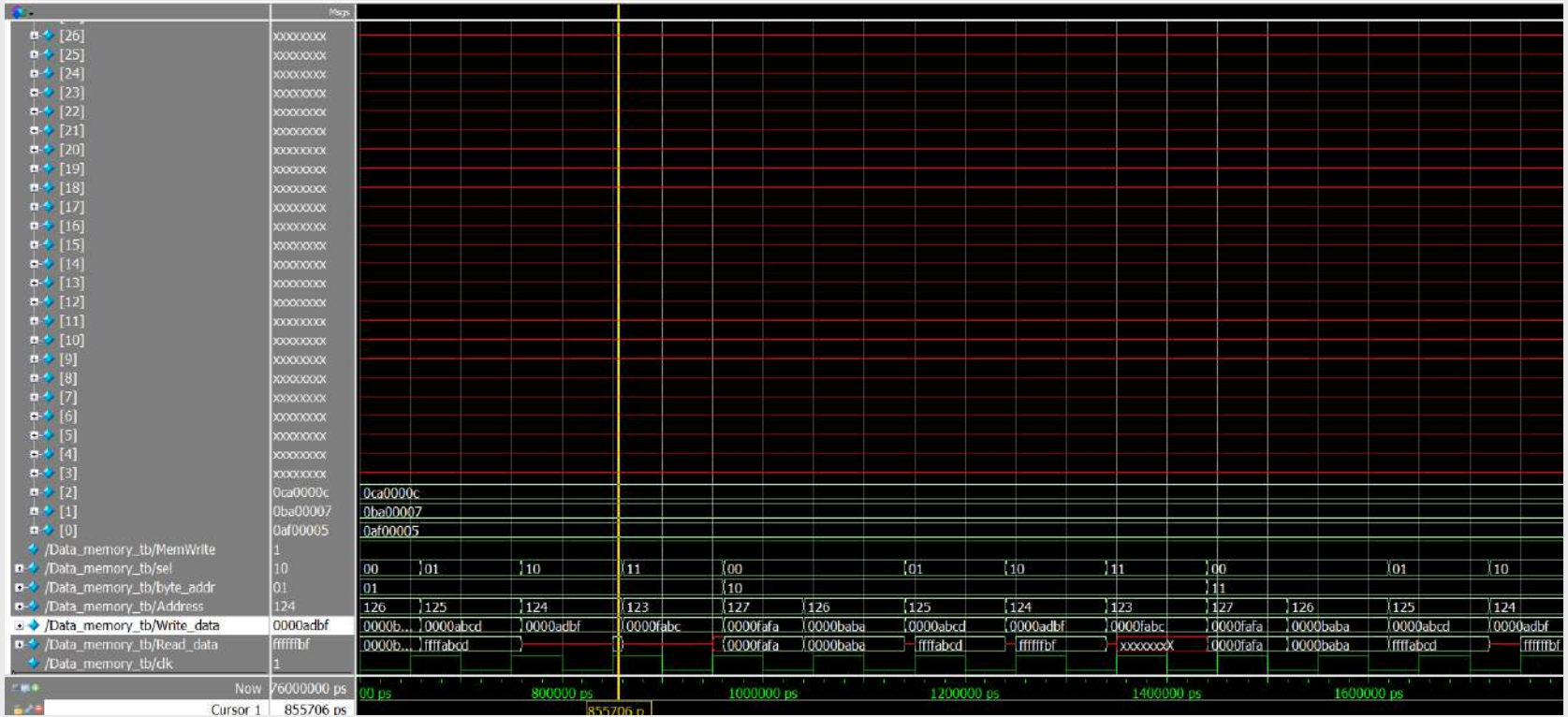
Registers_tb (ModelSIM simulation)



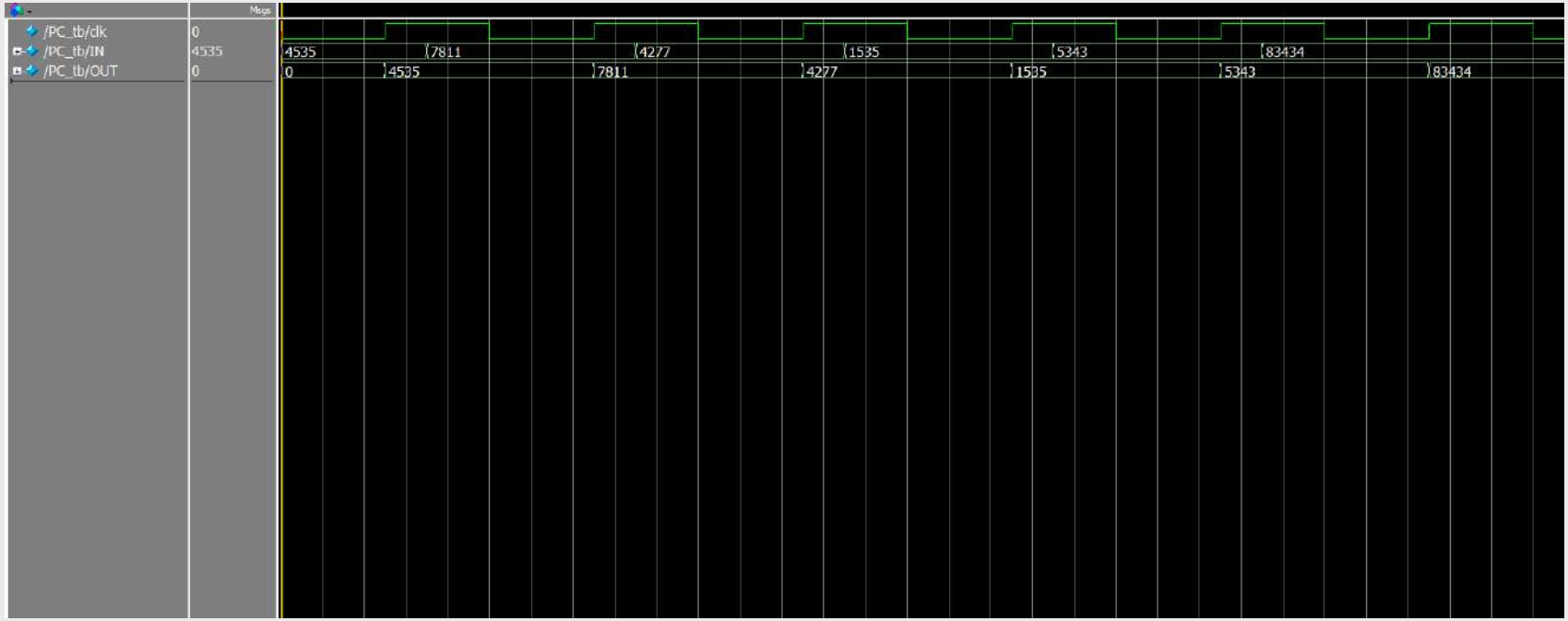
Instruction memory tb (ModelSIM simulation)



Data memory tb (ModelSIM simulation)

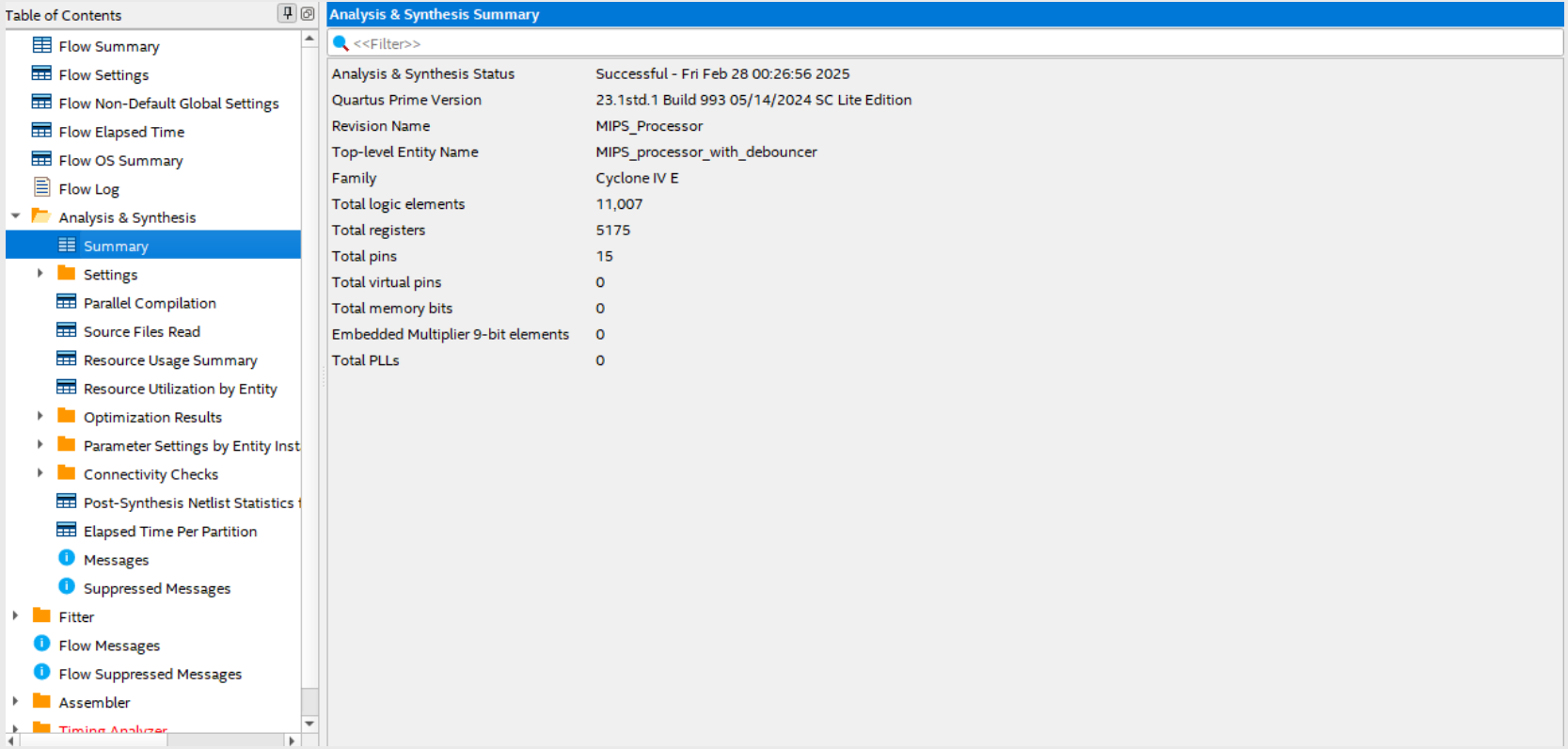


PC_tb (ModelSIM simulation)



MUX 4:1 tb (ModelSIM simulation)

synthesis results



synthesis results

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
  - Summary
  - Settings
  - Parallel Compilation
  - Source Files Read
  - Resource Usage Summary
  - Resource Utilization by Entity
  - Optimization Results
  - Parameter Settings by Entity Inst
  - Connectivity Checks
  - Post-Synthesis Netlist Statistics
  - Elapsed Time Per Partition
  - Messages
  - Suppressed Messages
- Fitter
- Flow Messages
- Flow Suppressed Messages
- Assembler
- Timing Analyzer

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins
1	|MIPS_processor_with_debouncer	6456 (33)	5175 (0)	0	0	0	0	15
1	|ALU:ALU|	357 (172)	0 (0)	0	0	0	0	0
2	|ALU_control:ALU_control|	13 (13)	0 (0)	0	0	0	0	0
3	|Branch_control:Branch_control|	2 (2)	0 (0)	0	0	0	0	0
4	|Control:Control_unit|	29 (29)	0 (0)	0	0	0	0	0
5	|Data_memory:Data_memory|	4263 (4263)	4096 (4096)	0	0	0	0	0
6	|Instruction_memory:Instruction_memory|	80 (80)	0 (0)	0	0	0	0	0
7	|MUX_4_1:MUX0|	3 (3)	0 (0)	0	0	0	0	0
8	|MUX_4_1:MUX1|	68 (68)	0 (0)	0	0	0	0	0
9	|MUX_4_1:MUX2|	65 (65)	0 (0)	0	0	0	0	0
10	|PC:PC|	28 (28)	32 (32)	0	0	0	0	0
11	|Registers:Registers|	1384 (1384)	1024 (1024)	0	0	0	0	0
12	|add_4:add_4|	52 (0)	0 (0)	0	0	0	0	0
13	|adder:adder|	51 (0)	0 (0)	0	0	0	0	0
14	|debouncer:de_inst|	27 (27)	23 (23)	0	0	0	0	0
15	|extend_unit:extend_unit|	1 (1)	0 (0)	0	0	0	0	0

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

synthesis results

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
  - Summary
  - Settings
  - Parallel Compilation
  - Source Files Read
  - Resource Usage Summary
  - Resource Utilization by Entity
  - Optimization Results
  - Parameter Settings by Entity Inst
  - Connectivity Checks
  - Post-Synthesis Netlist Statistics
  - Elapsed Time Per Partition
  - Messages
  - Suppressed Messages
- Fitter
- Flow Messages
- Flow Suppressed Messages
- Assembler
- Timing Analyzer

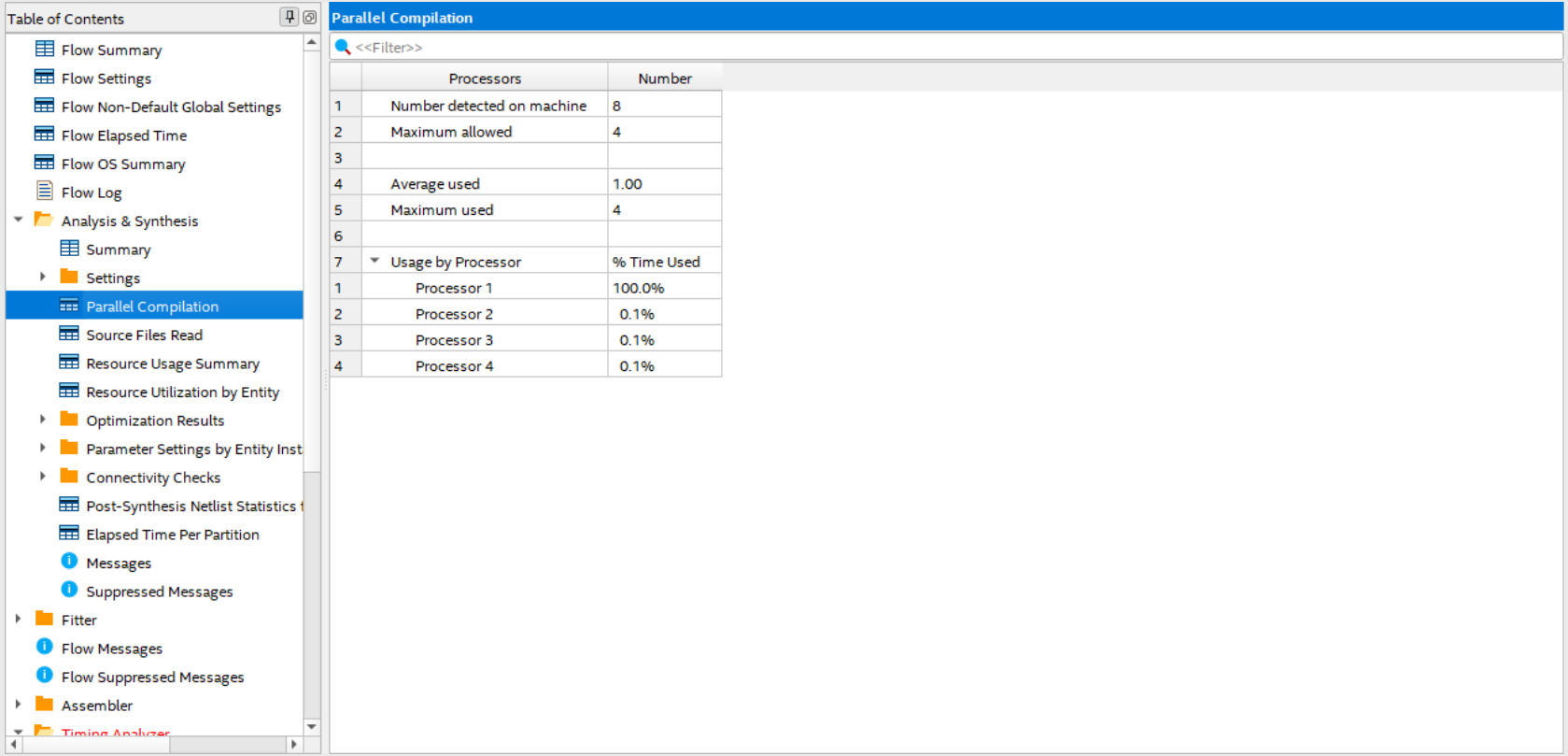
Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

ents	DSP 9x9	DSP 18x18	Pins	Virtual Pins	Full Hierarchy Name	Entity Name	Library Name
	0	0	15	0	|MIPS_processor_with_debouncer	MIPS_proc...debouncer	work
	0	0	0	0	|MIPS_processor_with_debouncer|ALU:ALU	ALU	work
	0	0	0	0	|MIPS_processor_with_debouncer|ALU_control:ALU_control	ALU_control	work
	0	0	0	0	|MIPS_processor_with_debouncer|Branch_control:Branch_control	Branch_control	work
	0	0	0	0	|MIPS_processor_with_debouncer|Control:Control_unit	Control	work
	0	0	0	0	|MIPS_processor_with_debouncer|Data_memory:Data_memory	Data_memory	work
	0	0	0	0	|MIPS_processor_with_debouncer|Instruction_memory:Instruction_memory	Instruction_memory	work
	0	0	0	0	|MIPS_processor_with_debouncer|MUX_4_1:MUX0	MUX_4_1	work
	0	0	0	0	|MIPS_processor_with_debouncer|MUX_4_1:MUX1	MUX_4_1	work
	0	0	0	0	|MIPS_processor_with_debouncer|MUX_4_1:MUX2	MUX_4_1	work
	0	0	0	0	|MIPS_processor_with_debouncer|PC:PC	PC	work
	0	0	0	0	|MIPS_processor_with_debouncer|Registers:Registers	Registers	work
	0	0	0	0	|MIPS_processor_with_debouncer|add_4:add_4	add_4	work
	0	0	0	0	|MIPS_processor_with_debouncer|adder:adder	adder	work
	0	0	0	0	|MIPS_processor_with_debouncer|debouncer:de_inst	debouncer	work
	0	0	0	0	|MIPS_processor_with_debouncer|extend_unit:extend_unit	extend_unit	work

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

synthesis results



synthesis results

Table of Contents

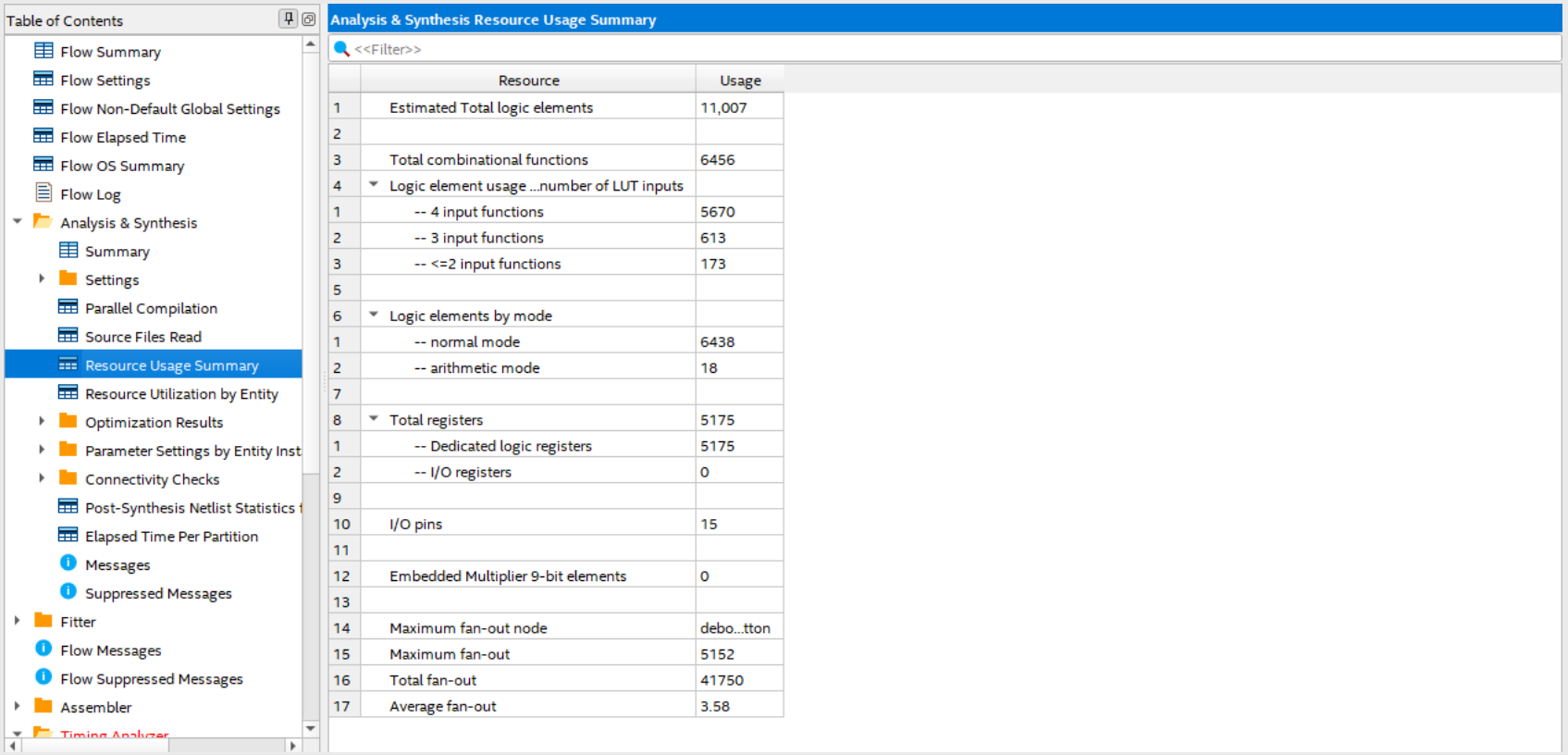
- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
  - Summary
  - Settings
  - Parallel Compilation
  - Source Files Read
  - Resource Usage Summary
  - Resource Utilization by Entity
  - Optimization Results
  - Parameter Settings by Entity Inst
  - Connectivity Checks
  - Post-Synthesis Netlist Statistics f
  - Elapsed Time Per Partition
  - Messages
  - Suppressed Messages
- Fitter
- Flow Messages
- Flow Suppressed Messages
- Assembler
- Timing Analyzer

Analysis & Synthesis Source Files Read

<<Filter>>

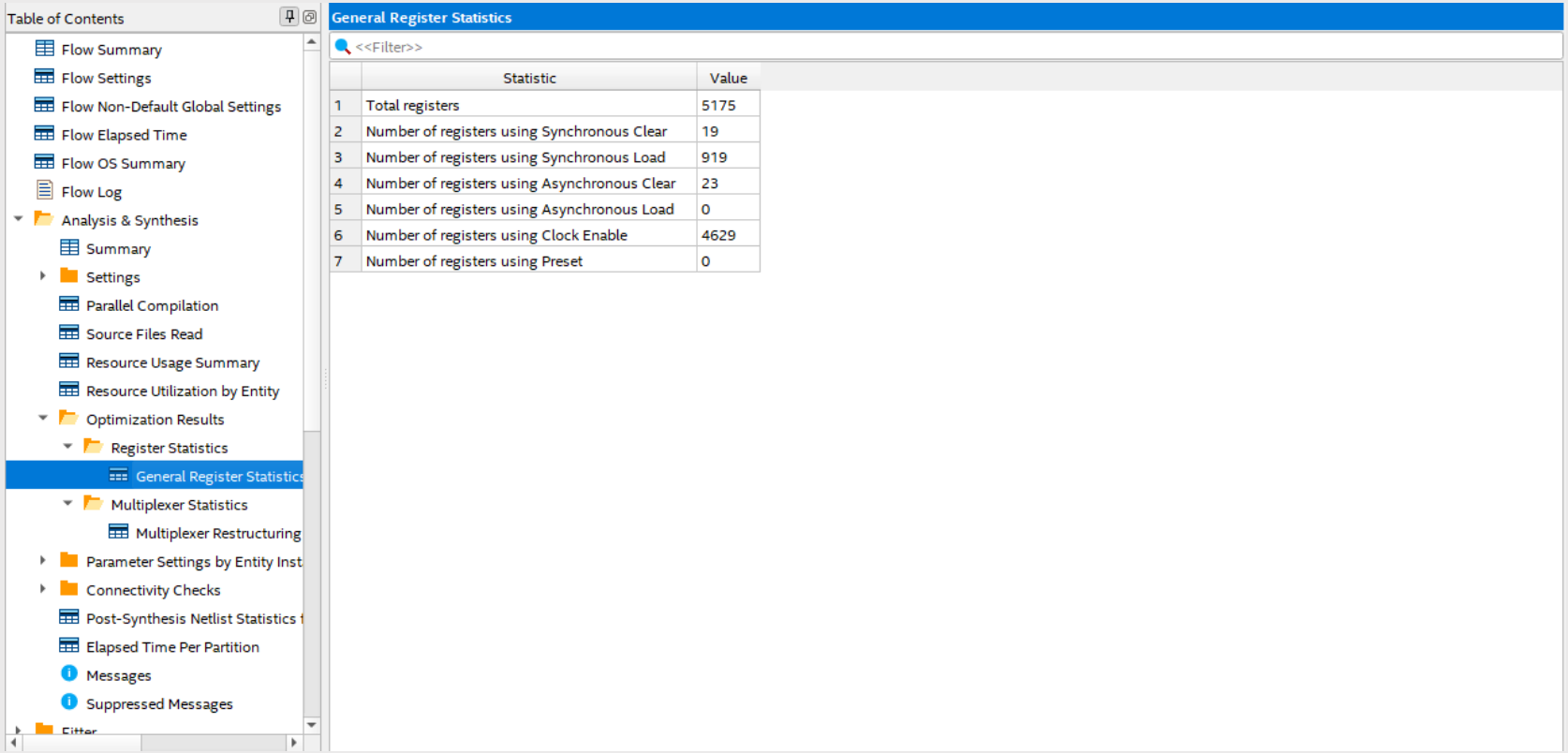
	File Name with User-Entered Path	Used in Netlist	File Type	File Name with Absolute Path	Library
1	ALU.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/ALU.v	
2	Arithmetic_unit.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Arithmetic_unit.v	
3	Logic_unit.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Logic_unit.v	
4	Shift_lui_unit.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Shift_lui_unit.v	
5	adder.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/adder.v	
6	add_4.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/add_4.v	
7	Control.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Control.v	
8	ALU_control.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/ALU_control.v	
9	Branch_control.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Branch_control.v	
10	extend_unit.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/extend_unit.v	
11	shift_left_2_32_32.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/shift_left_2_32_32.v	
12	shift_left_2_26_28.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/shift_left_2_26_28.v	
13	concatenation_unit.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/concatenation_unit.v	
14	Registers.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Registers.v	
15	Instruction_memory.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Instruction_memory.v	
16	Data_memory.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/Data_memory.v	
17	PC.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/PC.v	
18	MUX_4_1.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/MUX_4_1.v	
19	RCA.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/RCA.v	
20	FA.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/FA.v	
21	debouncer.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/debouncer.v	
22	MIPS_processor_with_debouncer.v	yes	User Verilog HDL File	D:/Verilog_courses/MIPS_Processor/MIPS_processor_with_debouncer.v	

synthesis results



	Resource	Usage
1	Estimated Total logic elements	11,007
2		
3	Total combinational functions	6456
4	Logic element usage ...number of LUT inputs	
1	-- 4 input functions	5670
2	-- 3 input functions	613
3	-- <=2 input functions	173
5		
6	Logic elements by mode	
1	-- normal mode	6438
2	-- arithmetic mode	18
7		
8	Total registers	5175
1	-- Dedicated logic registers	5175
2	-- I/O registers	0
9		
10	I/O pins	15
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	debo...tton
15	Maximum fan-out	5152
16	Total fan-out	41750
17	Average fan-out	3.58

synthesis results



synthesis results

Table of Contents

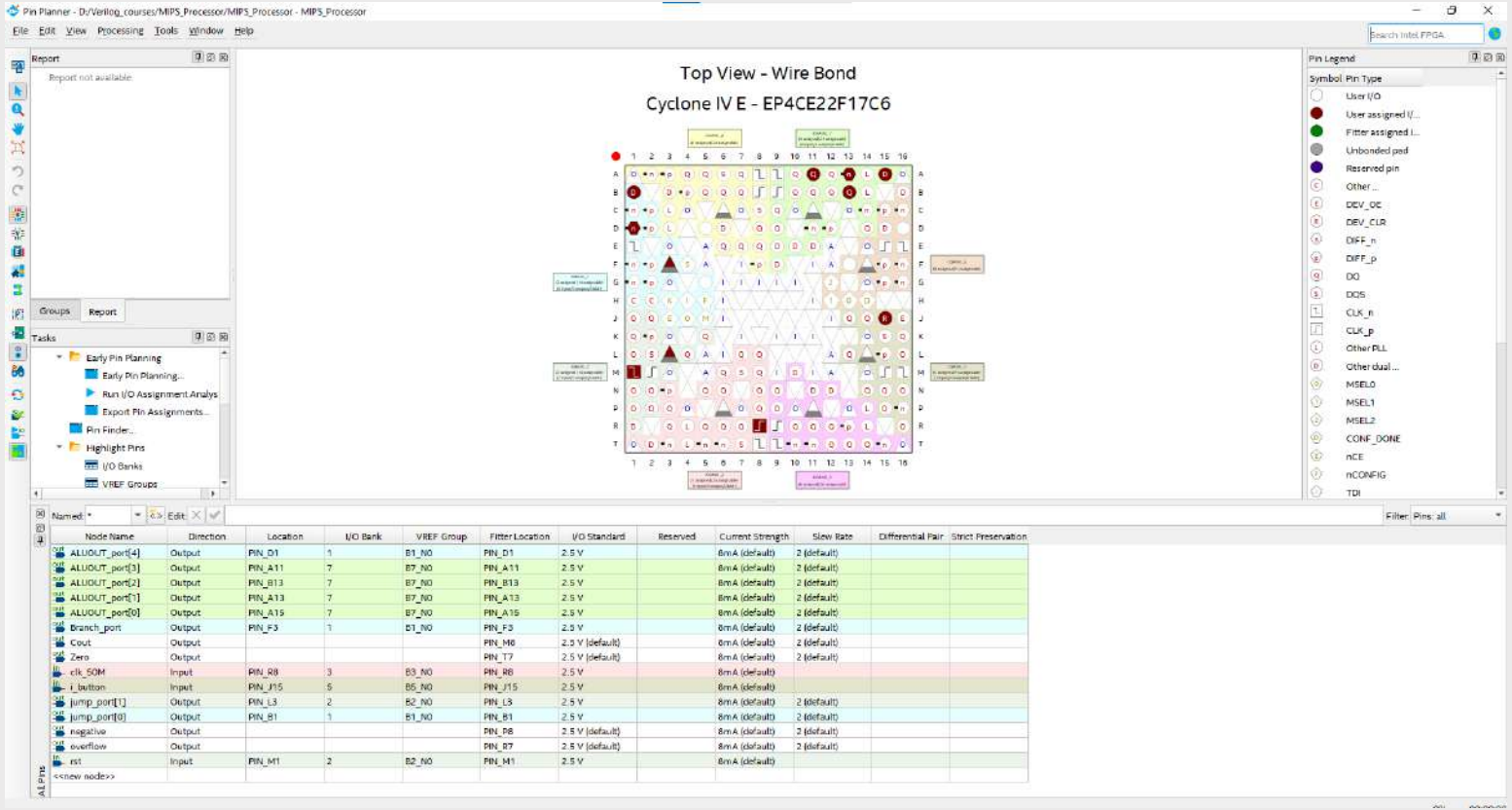
- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
  - Summary
  - Settings
  - Parallel Compilation
  - Source Files Read
  - Resource Usage Summary
  - Resource Utilization by Entity
  - Optimization Results
    - Register Statistics
      - General Register Statistics
    - Multiplexer Statistics
      - Multiplexer Restructuring
  - Parameter Settings by Entity Inst
  - Connectivity Checks
  - Post-Synthesis Netlist Statistics f
  - Elapsed Time Per Partition
  - Messages
  - Suppressed Messages
- Fitter

Multiplexer Restructuring Statistics (Restructuring Performed)

<<Filter>>

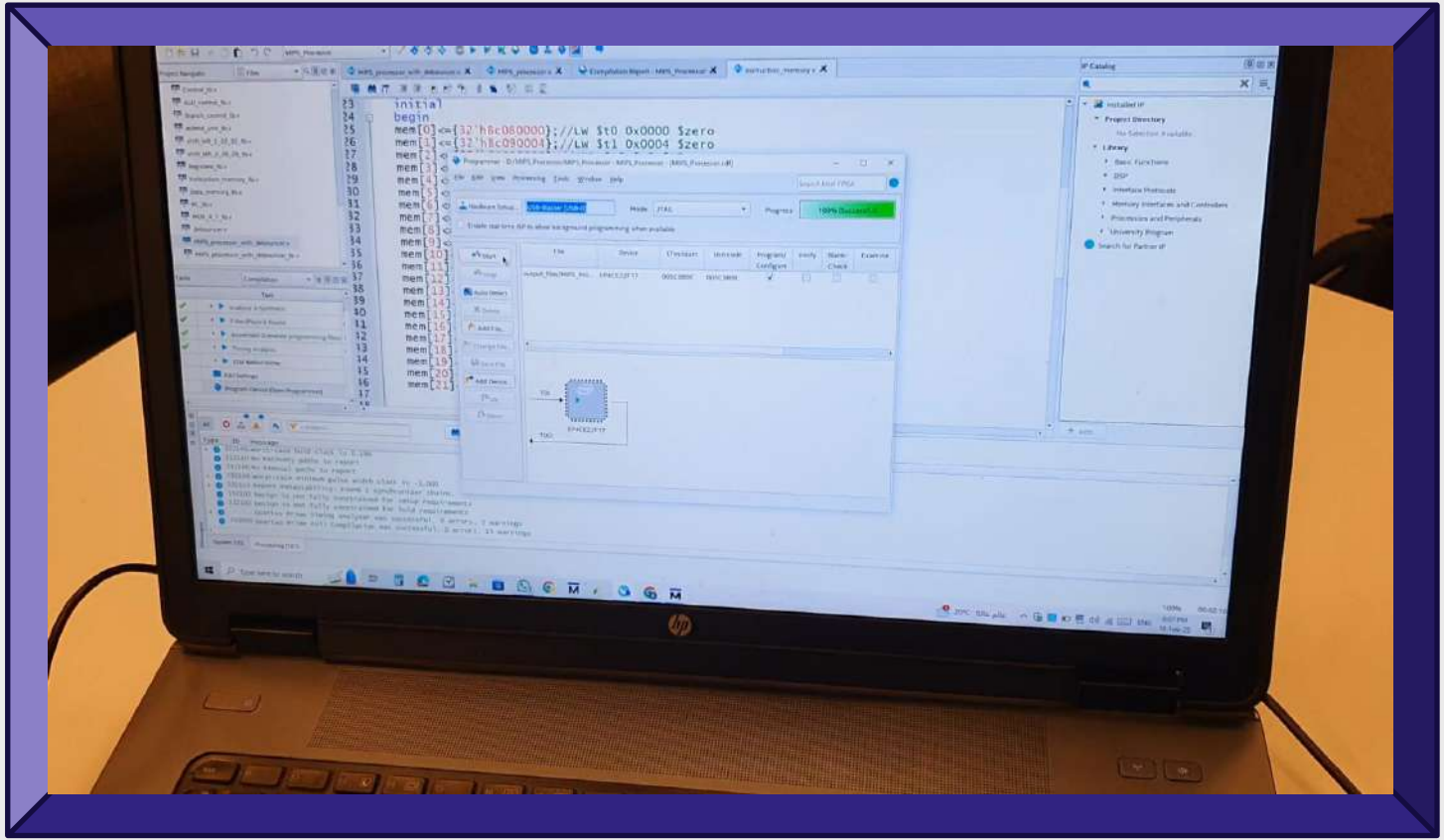
	Multiplexer Inputs	Bus Width	Baseline Area	Area if Restructured	Saving if Restructured	Registered	Example Multiplexer Output
1	3:1	19 bits	38 LEs	19 LEs	19 LEs	Yes	|MIPS_processor_with_debouncer|debouncer:de_inst|cou
2	4:1	2 bits	4 LEs	4 LEs	0 LEs	Yes	|MIPS_processor_with_debouncer|PC:PC|OUT[0]
3	5:1	3 bits	9 LEs	9 LEs	0 LEs	Yes	|MIPS_processor_with_debouncer|PC:PC|OUT[10]
4	5:1	4 bits	12 LEs	8 LEs	4 LEs	Yes	|MIPS_processor_with_debouncer|PC:PC|OUT[28]
5	5:1	23 bits	69 LEs	69 LEs	0 LEs	Yes	|MIPS_processor_with_debouncer|PC:PC|OUT[19]
6	5:1	7 bits	21 LEs	0 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
7	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
8	7:1	7 bits	28 LEs	14 LEs	14 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
9	5:1	7 bits	21 LEs	0 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
10	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
11	7:1	7 bits	28 LEs	14 LEs	14 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
12	5:1	7 bits	21 LEs	0 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
13	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
14	7:1	7 bits	28 LEs	14 LEs	14 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
15	5:1	7 bits	21 LEs	0 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
16	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
17	7:1	7 bits	28 LEs	14 LEs	14 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
18	5:1	7 bits	21 LEs	0 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
19	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
20	7:1	7 bits	28 LEs	14 LEs	14 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
21	5:1	7 bits	21 LEs	0 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
22	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
23	7:1	7 bits	28 LEs	14 LEs	14 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
24	5:1	7 bits	21 LEs	0 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m
25	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|MIPS_processor_with_debouncer|Data_memory:Data_m

Pin Planner



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
ALUOUT_port[4]	Output	PIN_D1	1	B1_N0	PIN_D1	2.5 V		8mA (default)	2 (default)		
ALUOUT_port[3]	Output	PIN_A11	7	B7_N0	PIN_A11	2.5 V		8mA (default)	2 (default)		
ALUOUT_port[2]	Output	PIN_B13	7	B7_N0	PIN_B13	2.5 V		8mA (default)	2 (default)		
ALUOUT_port[1]	Output	PIN_A13	7	B7_N0	PIN_A13	2.5 V		8mA (default)	2 (default)		
ALUOUT_port[0]	Output	PIN_A15	7	B7_N0	PIN_A15	2.5 V		8mA (default)	2 (default)		
Branch_port	Output	PIN_F3	1	B1_N0	PIN_F3	2.5 V		8mA (default)	2 (default)		
Cout	Output				PIN_M6	2.5 V (default)		8mA (default)	2 (default)		
Zero	Output				PIN_T7	2.5 V (default)		8mA (default)	2 (default)		
clk_50M	Input	PIN_R8	3	B3_N0	PIN_R8	2.5 V		8mA (default)			
i_button	Input	PIN_J15	5	B5_N0	PIN_J15	2.5 V		8mA (default)			
jump_port[1]	Output	PIN_L3	2	B2_N0	PIN_L3	2.5 V		8mA (default)	2 (default)		
jump_port[0]	Output	PIN_B1	1	B1_N0	PIN_B1	2.5 V		8mA (default)	2 (default)		
negative	Output				PIN_P8	2.5 V (default)		8mA (default)	2 (default)		
overflow	Output				PIN_R7	2.5 V (default)		8mA (default)	2 (default)		
rst	Input	PIN_M1	2	B2_N0	PIN_M1	2.5 V		8mA (default)			
<<new node>>											

FPGA implementation using (Cyclone IV E: EP4CE22F17C6)



MIPS processor

RISC Processor (Reduced Instruction Set Computer)

NAME : George Jan George Shaffik

historical background

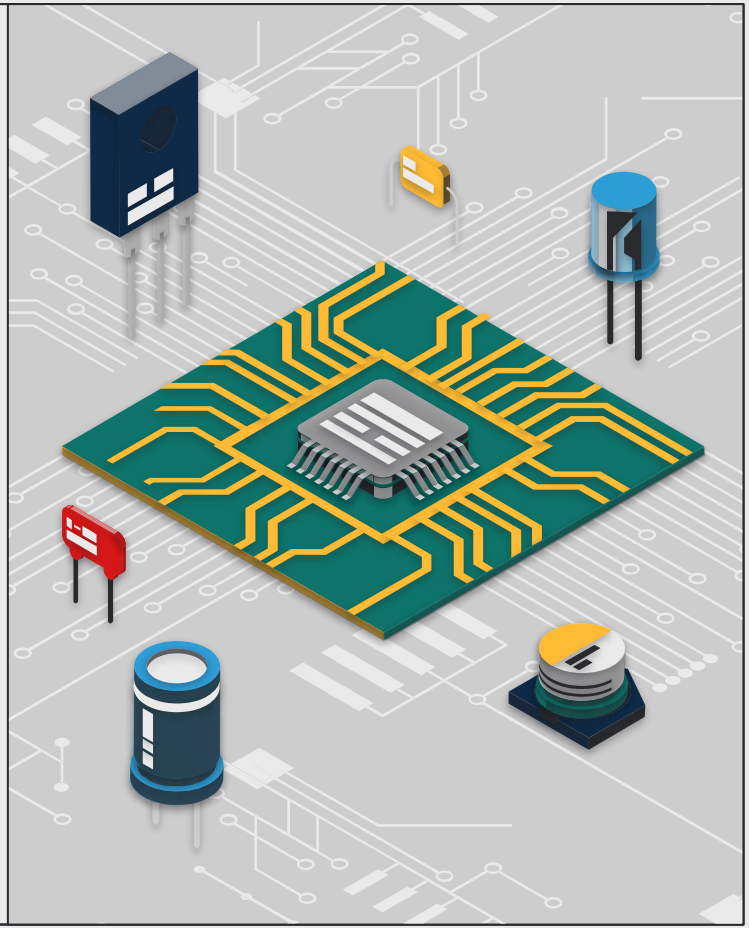
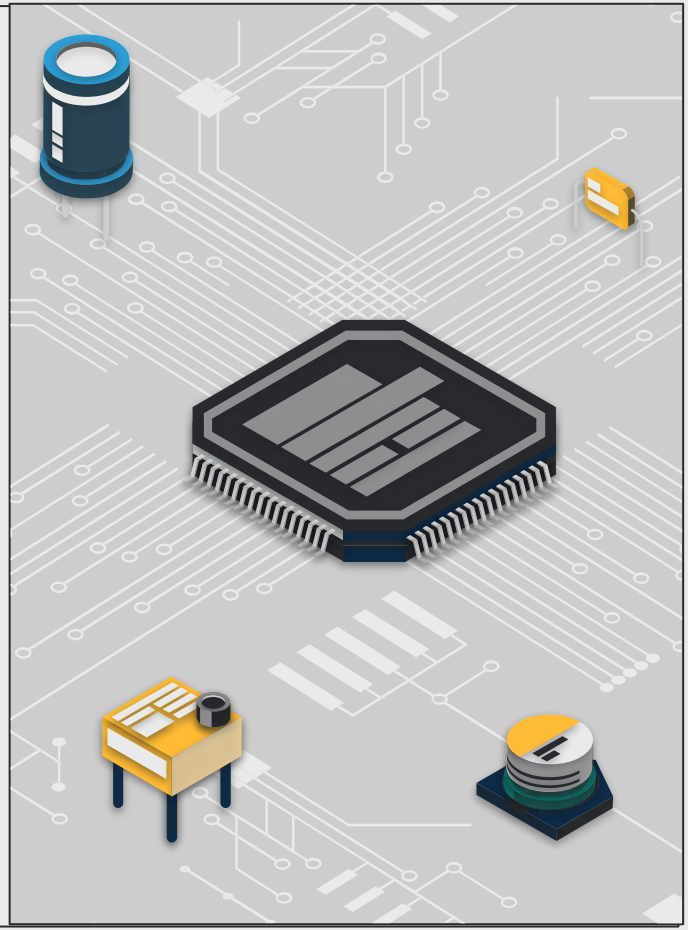


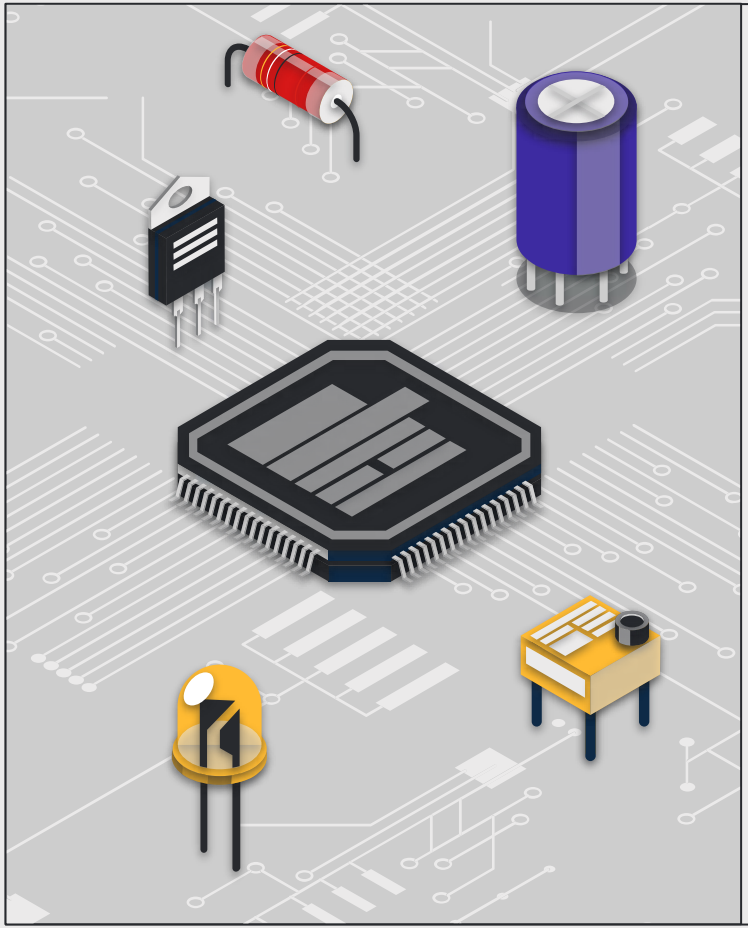
Table of contents

01

Versions

Versions of MIPS architecture





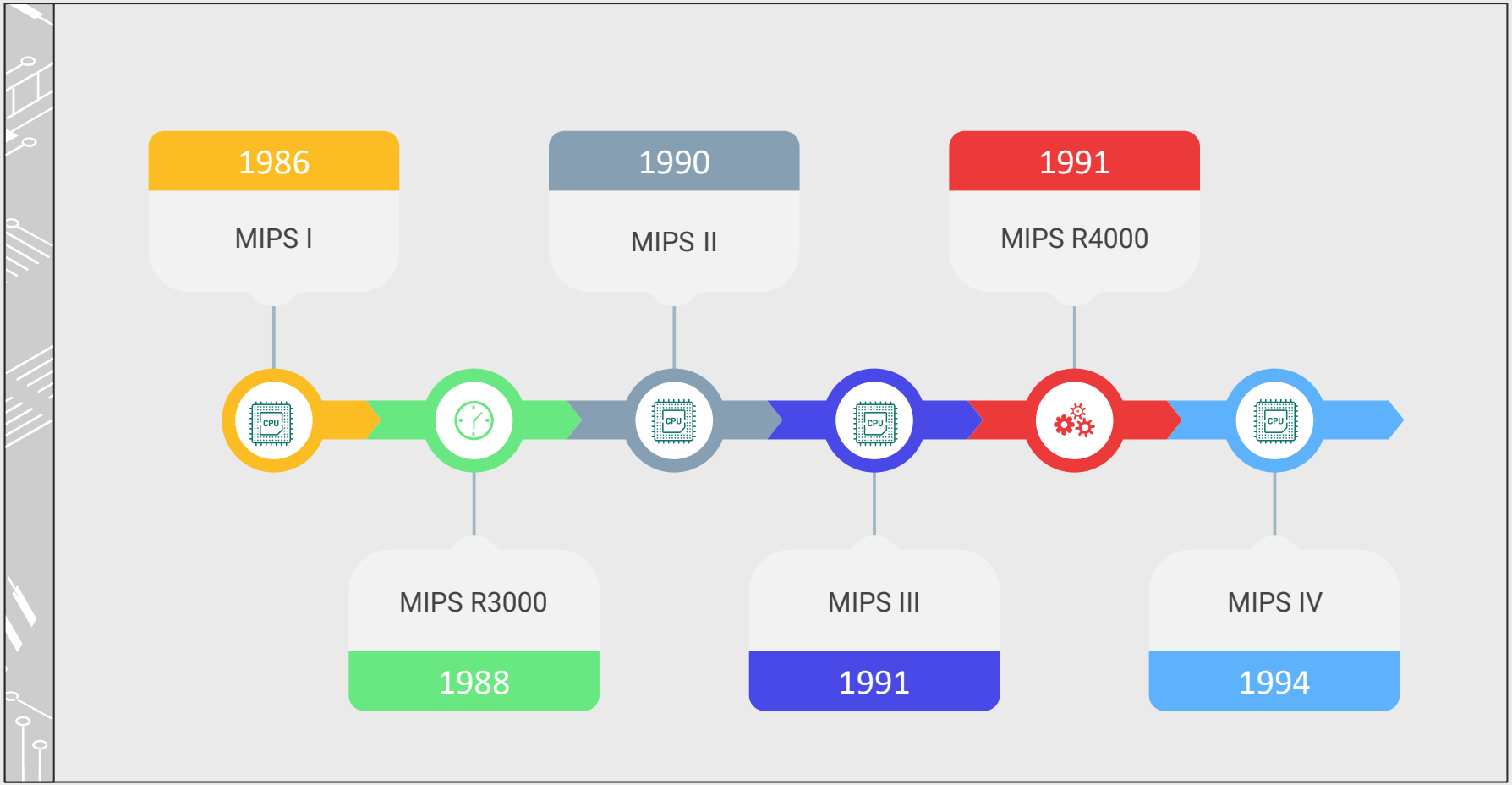
Versions!

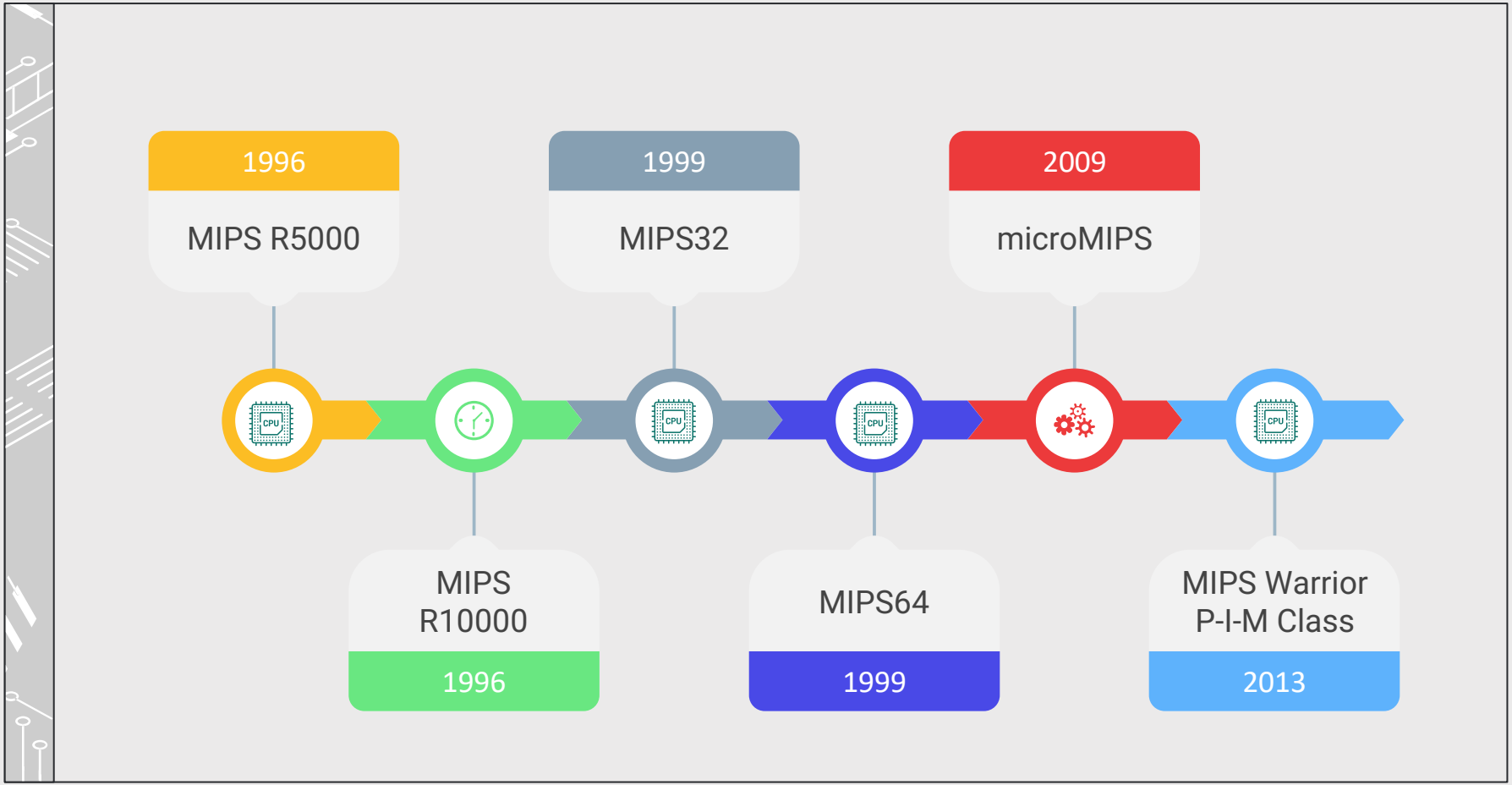
- MIPS I
- MIPS R3000
- MIPS II
- MIPS III
- MIPS R4000
- MIPS IV
- MIPS R5000
- MIPS R10000
- MIPS32
- MIPS64
- microMIPS
- MIPS Warrior P-Class
- MIPS Warrior I-Class
- MIPS Warrior M-Class

02

History

History of MIPS architecture

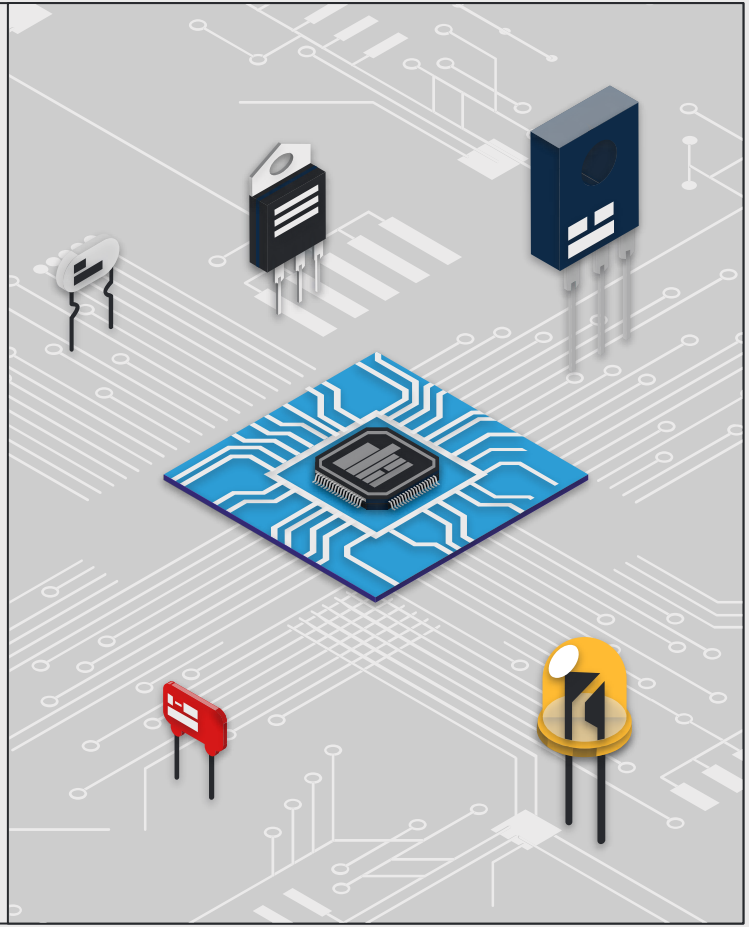

1986
MIPS I
1990
MIPS II
1991
MIPS R4000
MIPS R3000
1988
MIPS III
1991
MIPS IV
1994


1996
MIPS R5000
1999
MIPS32
2009
microMIPS
MIPS
R10000
1996
MIPS64
1999
MIPS Warrior
P-I-M Class
2013

03

MIPS architectures

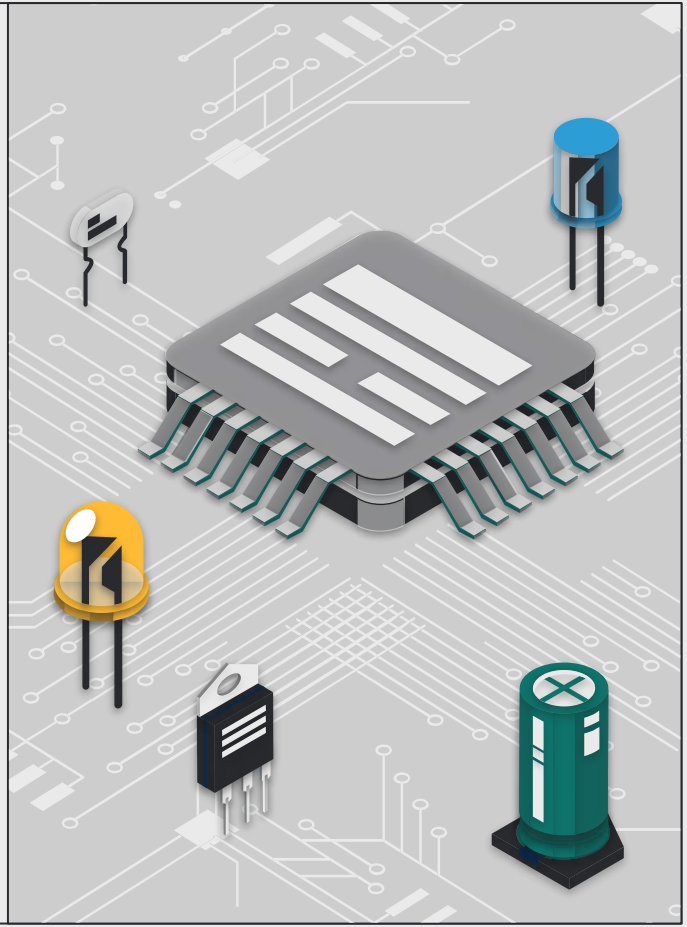
MIPS Architecture Comparison



MIPS architectures

MIPS I

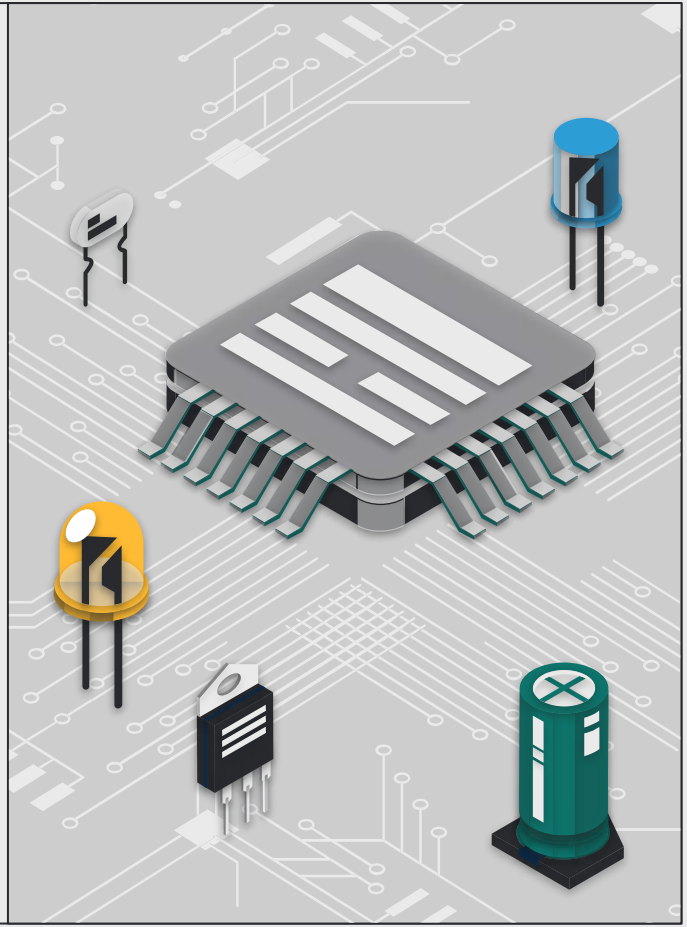
- **Pipeline Stages :** 5-stage
- **Speed (Performance) :** Low to Medium
- **Power Consumption :** Low
- **Area (Silicon Footprint) :** Small
- **Instruction Set Features :** 32-bit ISA, no FPU, basic integer ops.
- **Target Applications :** Embedded systems,
- **Key Advancements :** First commercial MIPS architecture.



MIPS architectures

MIPS II

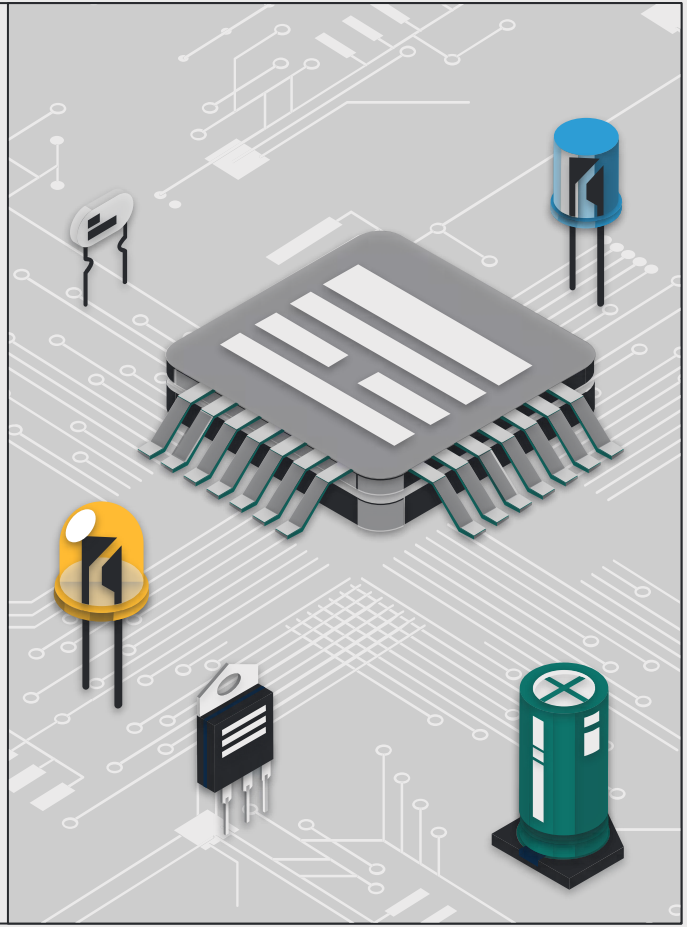
- **Pipeline Stages :** 5-stage
- **Speed (Performance) :** Medium
- **Power Consumption :** Low to Medium
- **Area (Silicon Footprint) :** Small
- **Instruction Set Features :** Adds 64-bit support, better branch prediction.
- **Target Applications :** Workstations, embedded systems.
- **Key Advancements :** Improved performance over MIPS I.



MIPS architectures

MIPS III

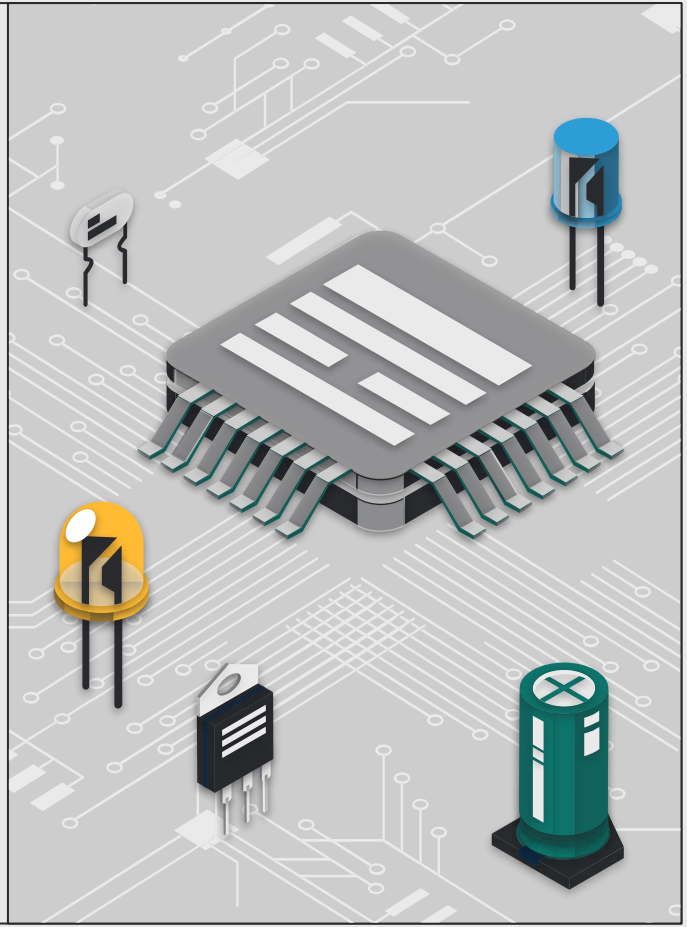
- **Pipeline Stages :** 5-stage
- **Speed (Performance) :** Medium to High
- **Power Consumption :** Medium
- **Area (Silicon Footprint) :** Medium
- **Instruction Set Features :** 64-bit ISA, FPU, superscalar execution.
- **Target Applications :** High-performance computing, servers.
- **Key Advancements :** Introduced 64-bit addressing.



MIPS architectures

MIPS IV

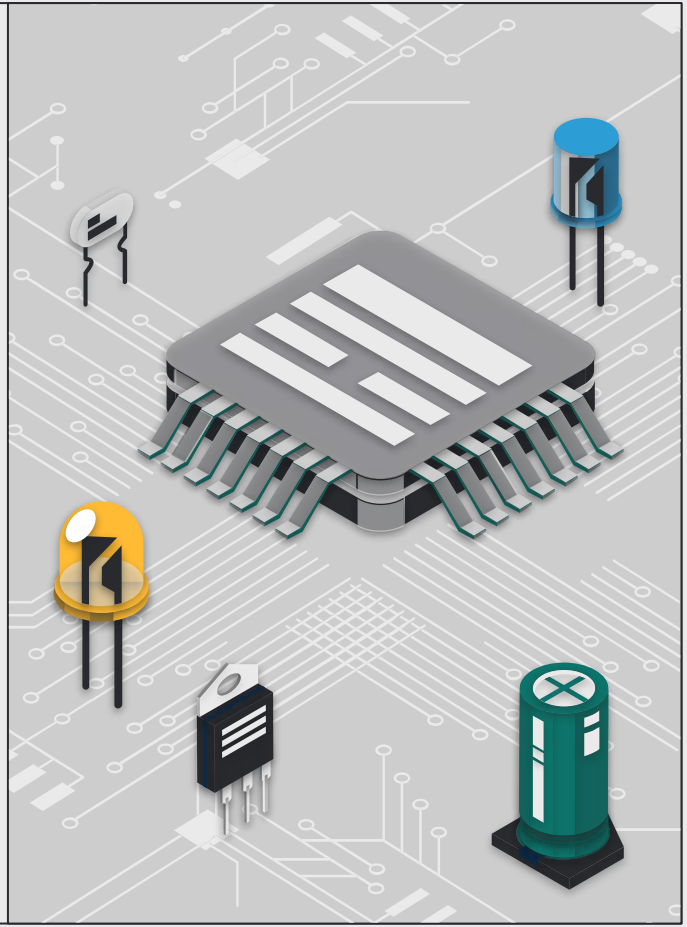
- **Pipeline Stages :** 6-stage
- **Speed (Performance) :** High
- **Power Consumption :** Medium to High
- **Area (Silicon Footprint) :** Medium to Large
- **Instruction Set Features :** Enhanced FPU, multimedia instructions.
- **Target Applications :** Graphics, multimedia, servers.
- **Key Advancements :** Improved FPU and SIMD(Single instruction, multiple data) support.



MIPS architectures

MIPS 32

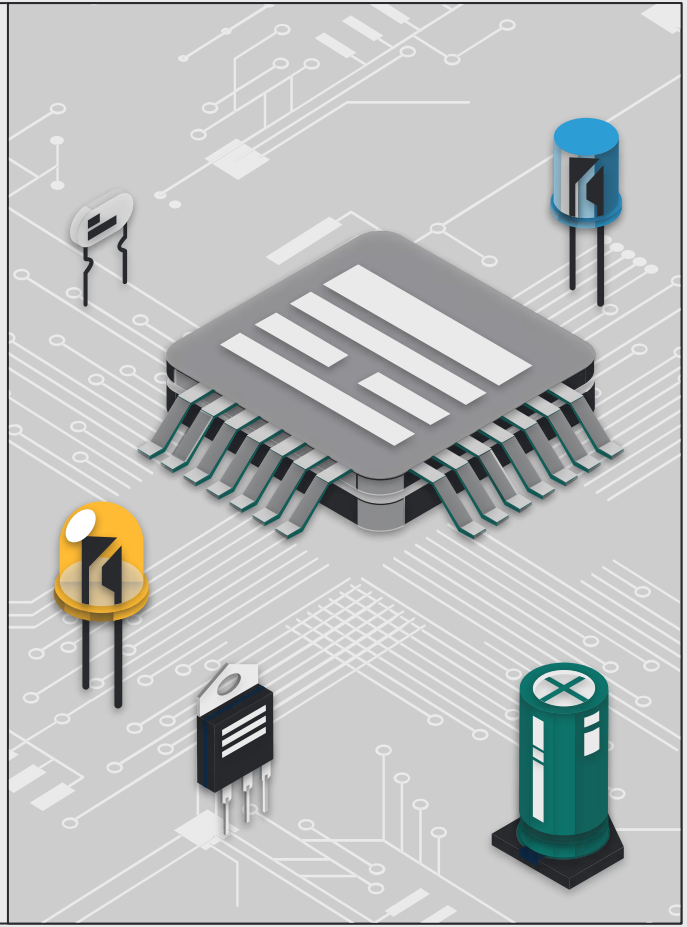
- **Pipeline Stages :** 5-7-stage
- **Speed (Performance) :** High
- **Power Consumption :** Medium
- **Area (Silicon Footprint) :** Medium
- **Instruction Set Features :** 32-bit ISA, DSP(Digital Signal Processing) extensions,FPU.
- **Target Applications :** Embedded systems, networking.
- **Key Advancements :** Optimized for embedded applications.



MIPS architectures

MIPS 64

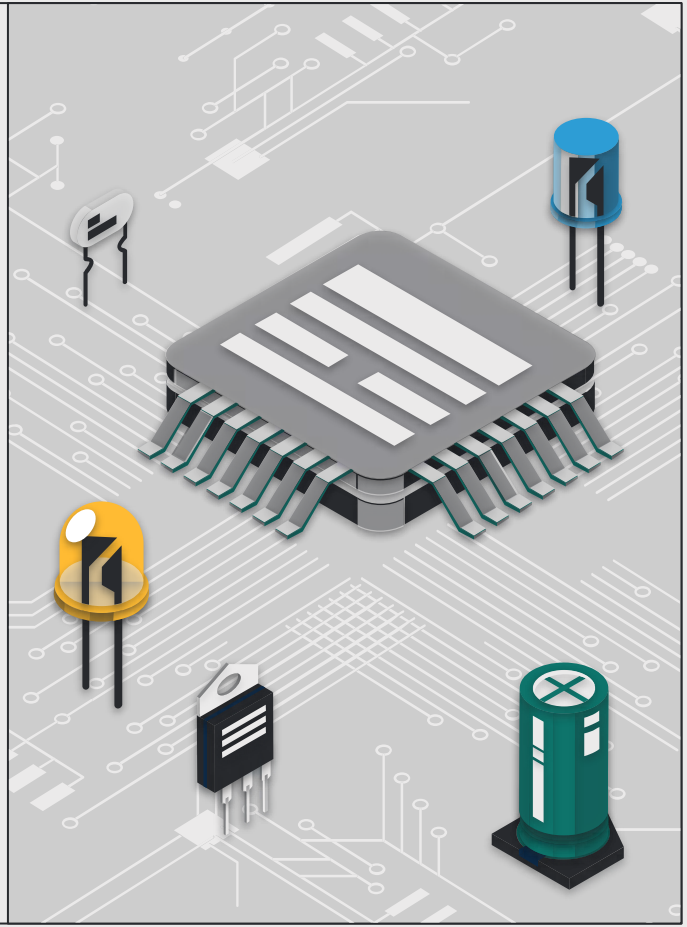
- **Pipeline Stages :** 6-8-stage
- **Speed (Performance) :** Very High
- **Power Consumption :** High
- **Area (Silicon Footprint) :** Large
- **Instruction Set Features :** 64-bit ISA, advanced FPU, SIMD.
- **Target Applications :** Servers, high-end embedded systems.
- **Key Advancements :** Scalable performance for 64-bit apps.



MIPS architectures

MIPS R3000

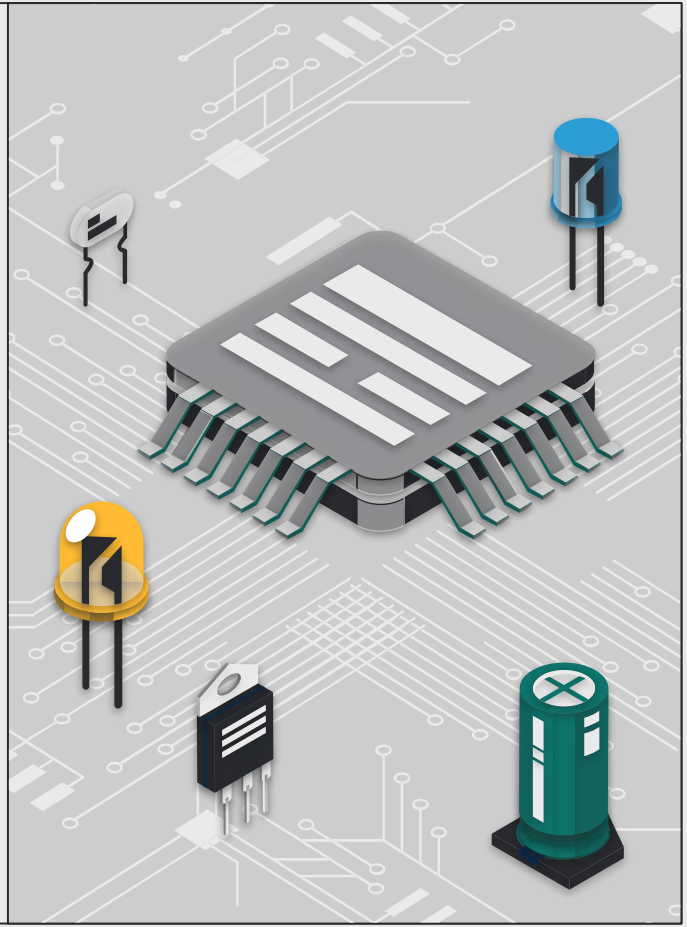
- **Pipeline Stages :** 5-stage
- **Speed (Performance) :** Low to Medium
- **Power Consumption :** Low
- **Area (Silicon Footprint) :** Small
- **Instruction Set Features :** 32-bit ISA, no FPU, basic integer ops.
- **Target Applications :** PCs, embedded systems
- **Key Advancements :** First widely successful MIPS CPU.



MIPS architectures

MIPS R4000

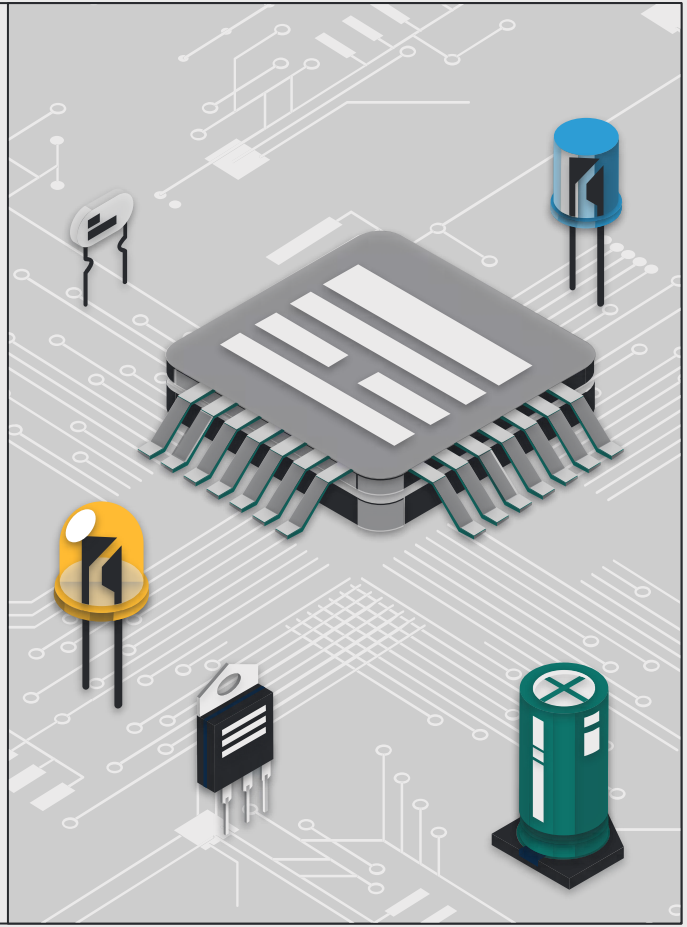
- **Pipeline Stages :** 8-stage
- **Speed (Performance) :** Medium to High
- **Power Consumption :** Medium
- **Area (Silicon Footprint) :** Medium
- **Instruction Set Features :** 64-bit ISA, FPU, superscalar.
- **Target Applications :** Workstations, servers.
- **Key Advancements :** First 64-bit MIPS CPU.



MIPS architectures

MIPS R5000

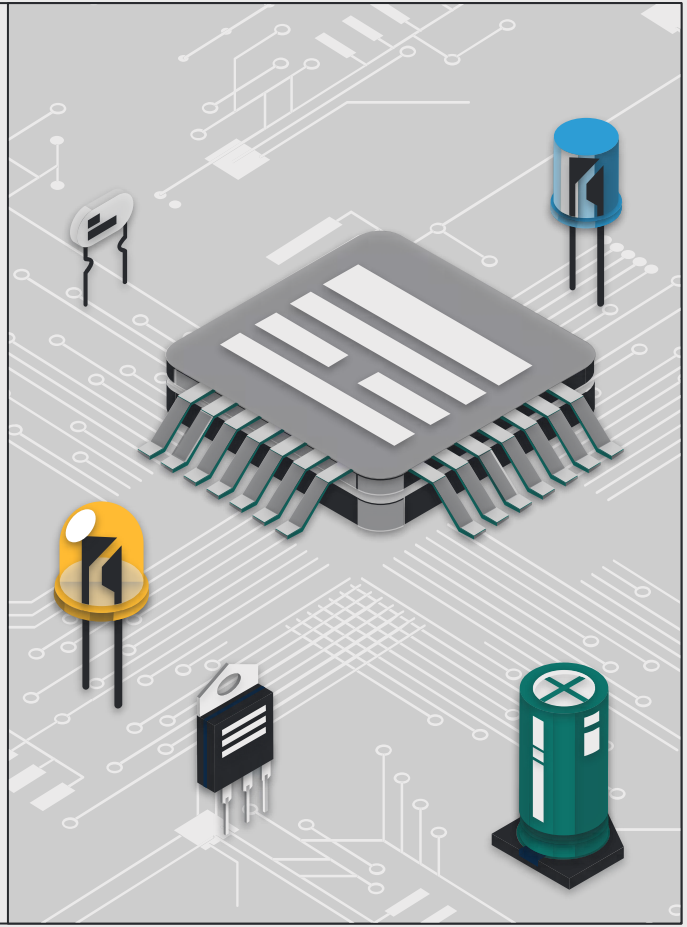
- **Pipeline Stages : 6**-stage
- **Speed (Performance) :** High
- **Power Consumption :** Medium
- **Area (Silicon Footprint) :** Medium
- **Instruction Set Features :** Enhanced FPU, better cache.
- **Target Applications :** Graphics, multimedia.
- **Key Advancements :** Improved FPU performance.



MIPS architectures

MIPS R10000

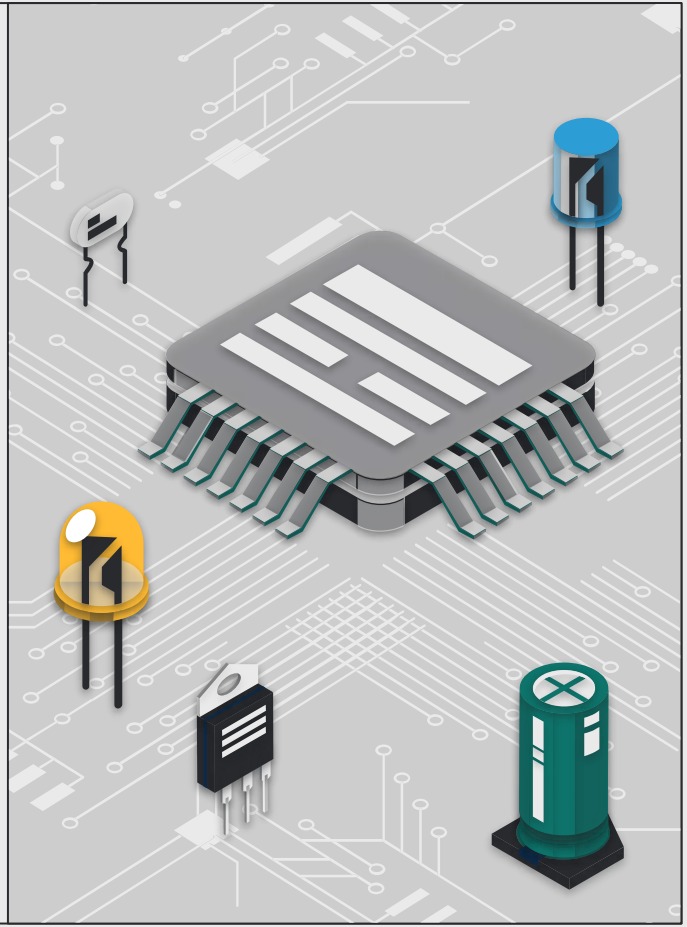
- **Pipeline Stages : 6**-stage
- **Speed (Performance) :** Very High
- **Power Consumption :** High
- **Area (Silicon Footprint) :** Large
- **Instruction Set Features :** Out-of-order execution, 4-way superscalar.
- **Target Applications :** High-performance computing.
- **Key Advancements :** Introduced out-of-order execution.



MIPS architectures

Micro MIPS

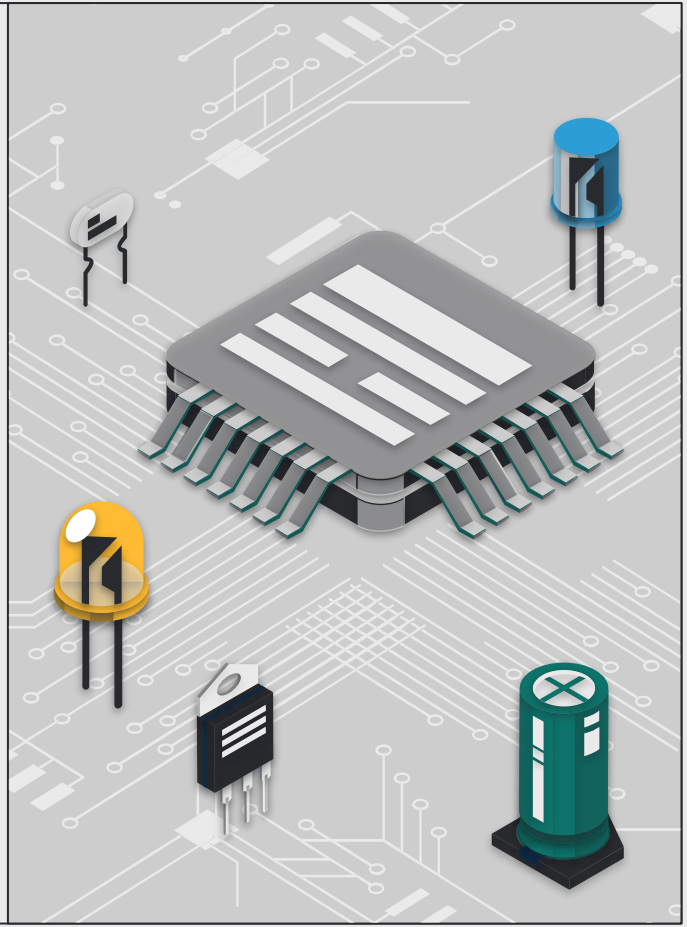
- **Pipeline Stages : 5**-stage
- **Speed (Performance) :** Medium
- **Power Consumption :** Low
- **Area (Silicon Footprint) :** Small
- **Instruction Set Features :** 16/32-bit hybrid ISA, DSP extensions.
- **Target Applications :** Ultra-low-power embedded systems.
- **Key Advancements :** Reduced code size for embedded apps.



MIPS architectures

MIPS Warrior P-Class

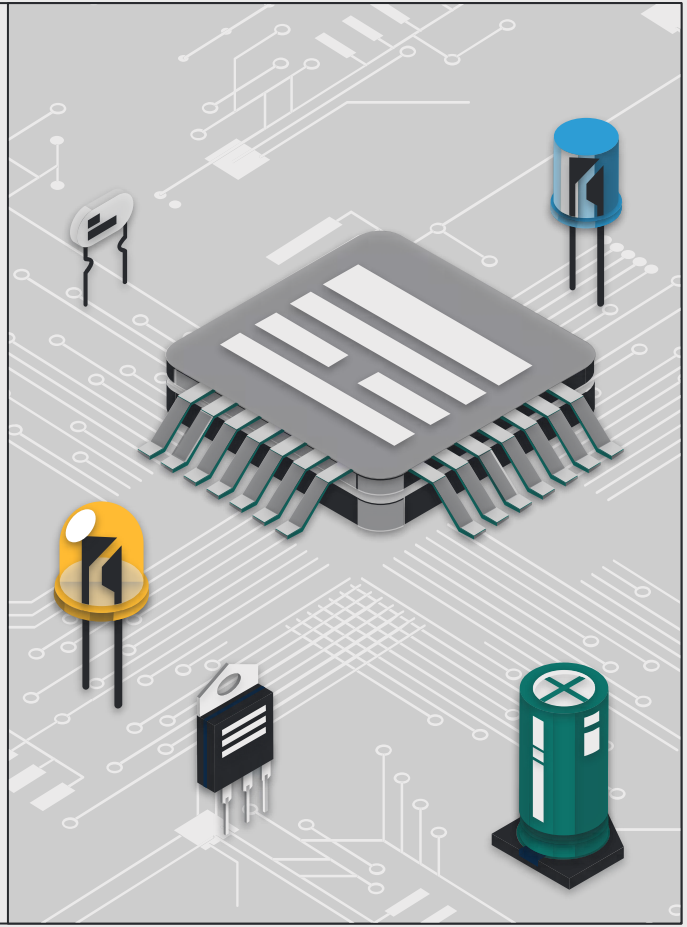
- **Pipeline Stages :** 6-8-stage
- **Speed (Performance) :** High
- **Power Consumption :** Medium
- **Area (Silicon Footprint) :** Medium
- **Instruction Set Features :** 32/64-bit ISA, FPU, SIMD.
- **Target Applications :** High-performance embedded systems.
- **Key Advancements :** Optimized for performance.



MIPS architectures

MIPS Warrior I-Class

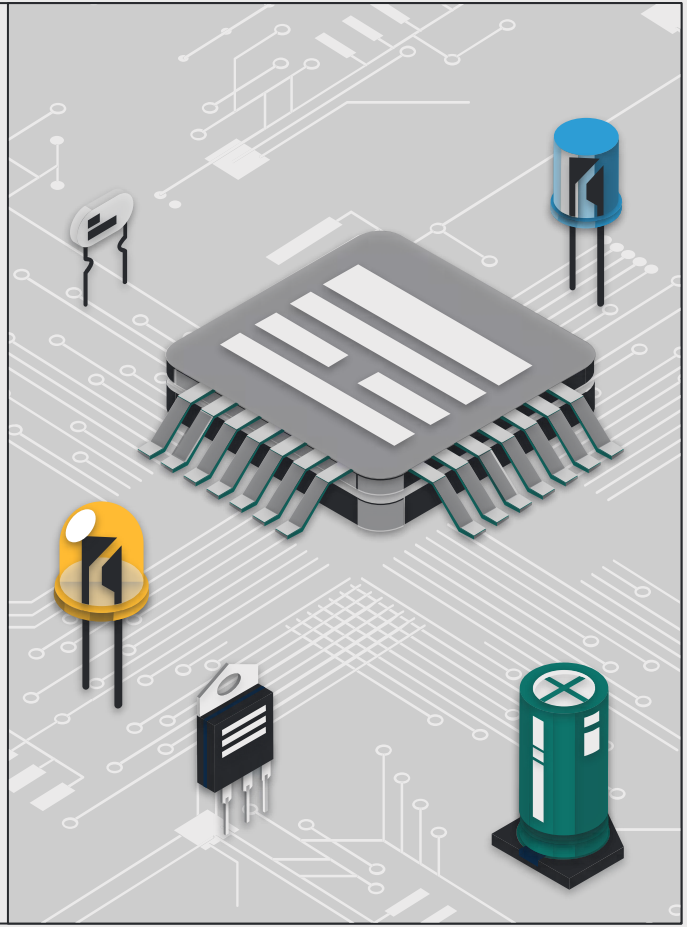
- **Pipeline Stages :** 5-7-stage
- **Speed (Performance) :** Medium to High
- **Power Consumption :** Low to Medium
- **Area (Silicon Footprint) :** Small to Medium
- **Instruction Set Features :** 32-bit ISA, DSP extensions.
- **Target Applications :** IoT, networking, automotive.
- **Key Advancements :** Balanced performance and power.



MIPS architectures

MIPS Warrior M-Class

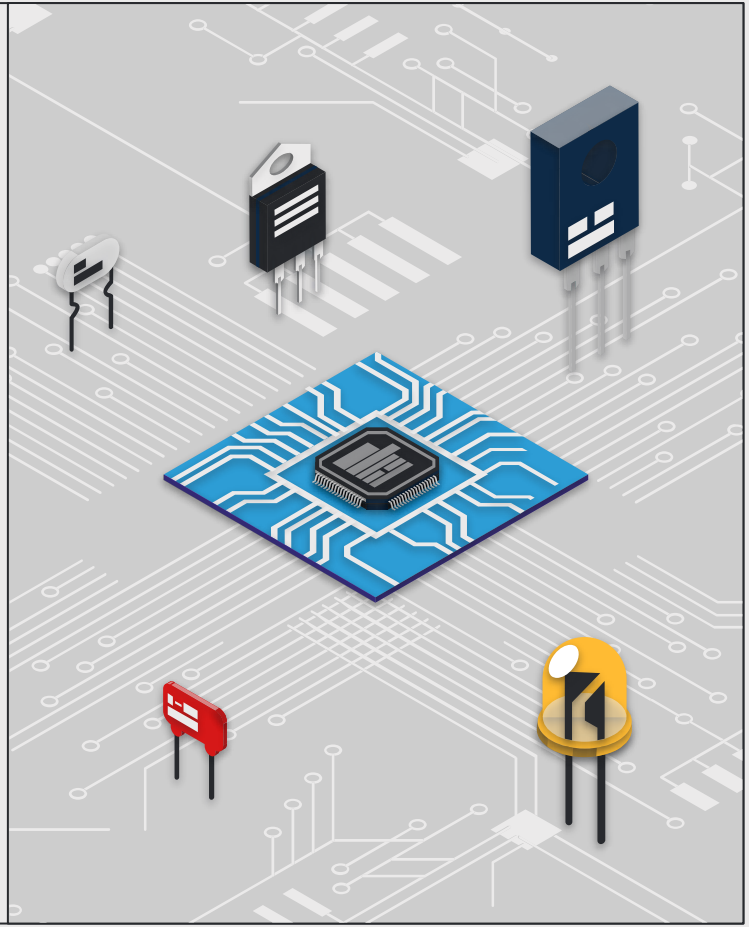
- **Pipeline Stages :** 3-5-stage
- **Speed (Performance) :** Low to Medium
- **Power Consumption :** Very Low
- **Area (Silicon Footprint) :** Small
- **Instruction Set Features :** 32-bit ISA, ultra-low power.
- **Target Applications :** IoT, microcontrollers.
- **Key Advancements :** Optimized for ultra-low power.



04

MIPS I

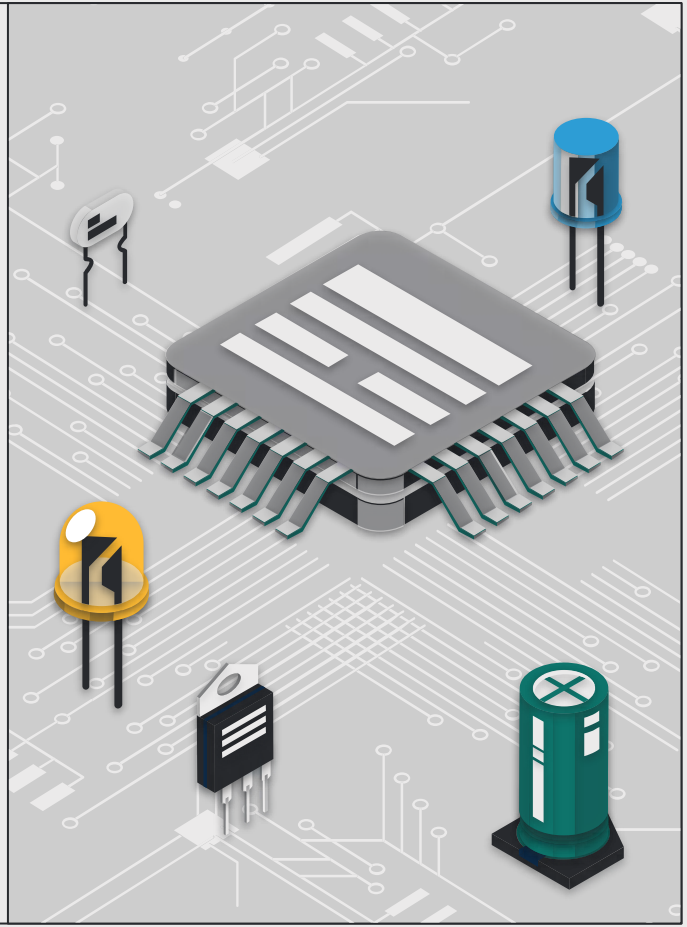
What is MIPS I?



MIPS architectures

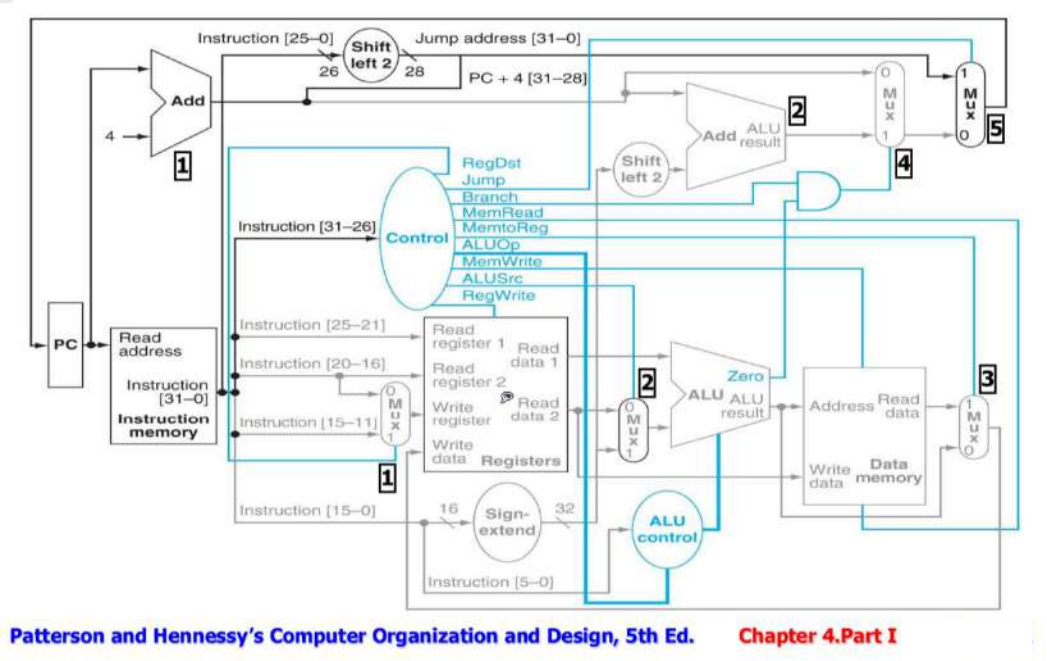
MIPS I

- **Pipeline Stages :** 5-stage
- **Speed (Performance) :** Low to Medium
- **Power Consumption :** Low
- **Area (Silicon Footprint) :** Small
- **Instruction Set Features :** 32-bit ISA, no FPU, basic integer ops.
- **Target Applications :** Embedded systems.
- **Key Advancements :** First commercial MIPS architecture.

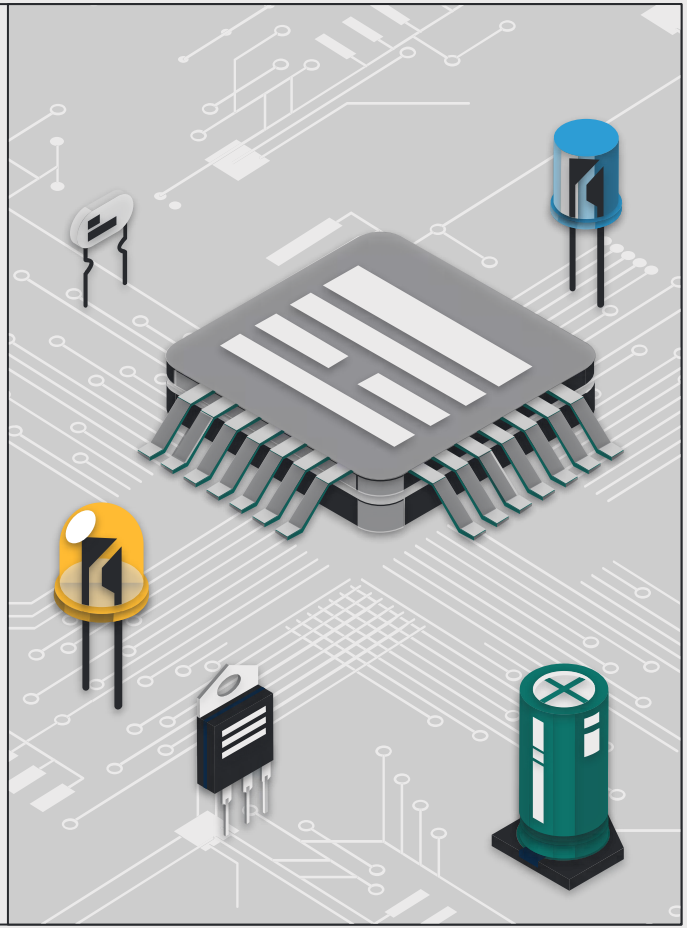


MIPS architectures

MIPS I



Patterson and Hennessy's Computer Organization and Design, 5th Ed. Chapter 4.Part I



References

1.MIPS Architecture Overview:

"MIPS32 Architecture for Programmers" by MIPS Technologies.

"Computer Organization and Design: The Hardware/Software Interface" by David A. Patterson and John L. Hennessy.

2.MIPS R-Series Processors:

"See MIPS Run" by Dominic Sweetman.

MIPS Technologies White Papers on R3000, R4000, and R10000.

3.MIPS Warrior Series:

MIPS Warrior Processor Family Datasheets (Imagination Technologies).

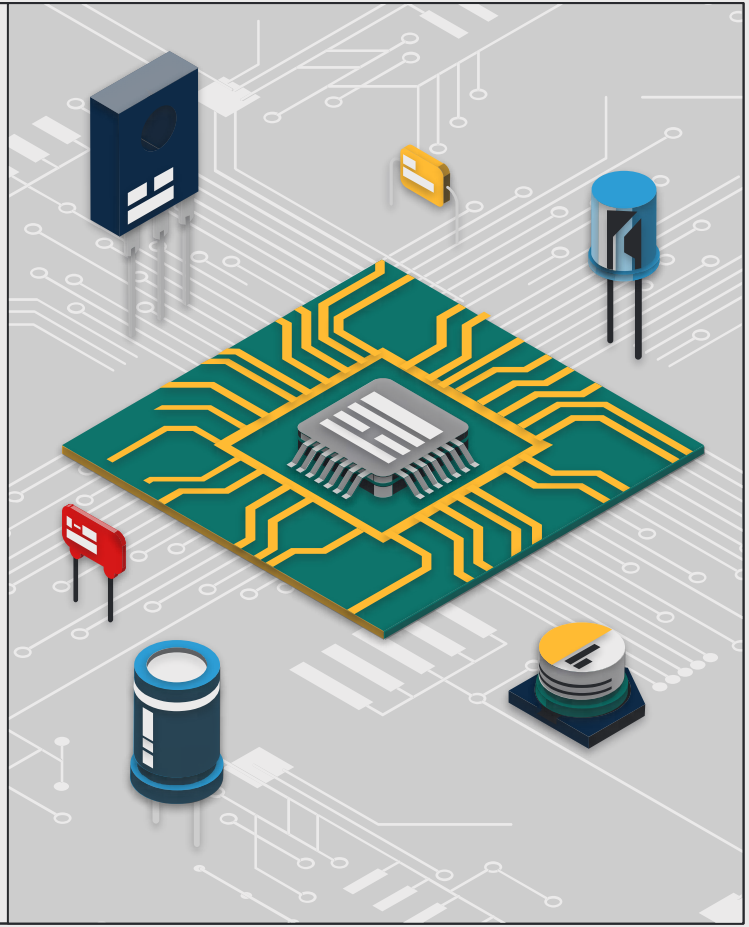
4.microMIPS:

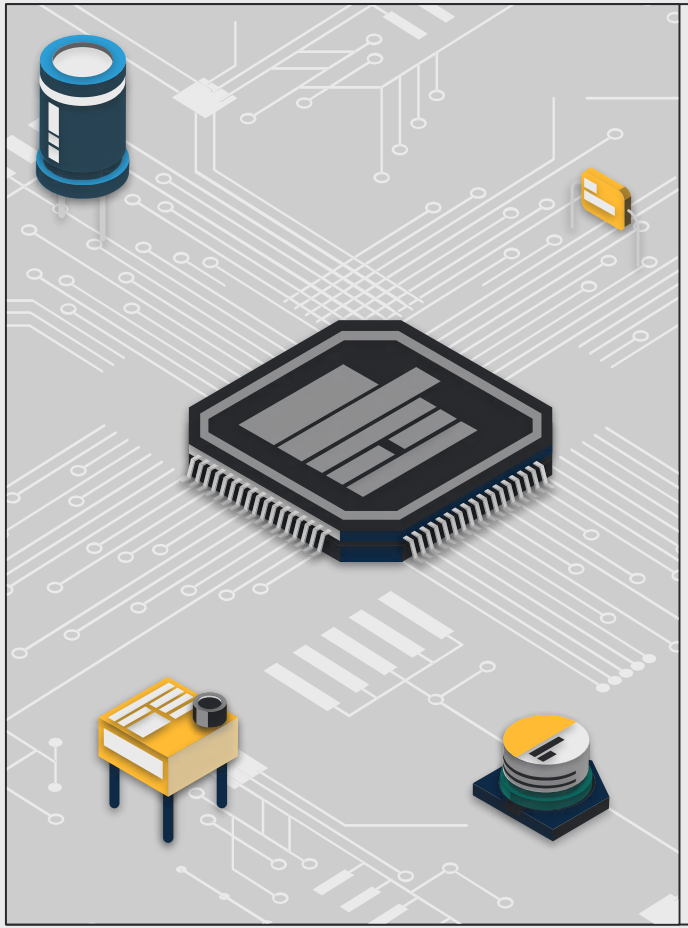
"MIPS32 microMIPS Architecture" by MIPS Technologies.

5.General MIPS Comparisons:

Academic papers on MIPS architecture performance and power analysis.

6. patterson and hennessy computer organization and design





Thanks