Tristate buffer code:

```
`timescale 1ns / 1ps

module tristate(enable, data_in, data_out);
parameter n = 8;
input enable;
input [n-1:0] data_in;
output [n-1:0] data_out;

assign data_out = enable ? data_in : {n{1'bz}};
endmodule
```

Tristate buffer testbench:

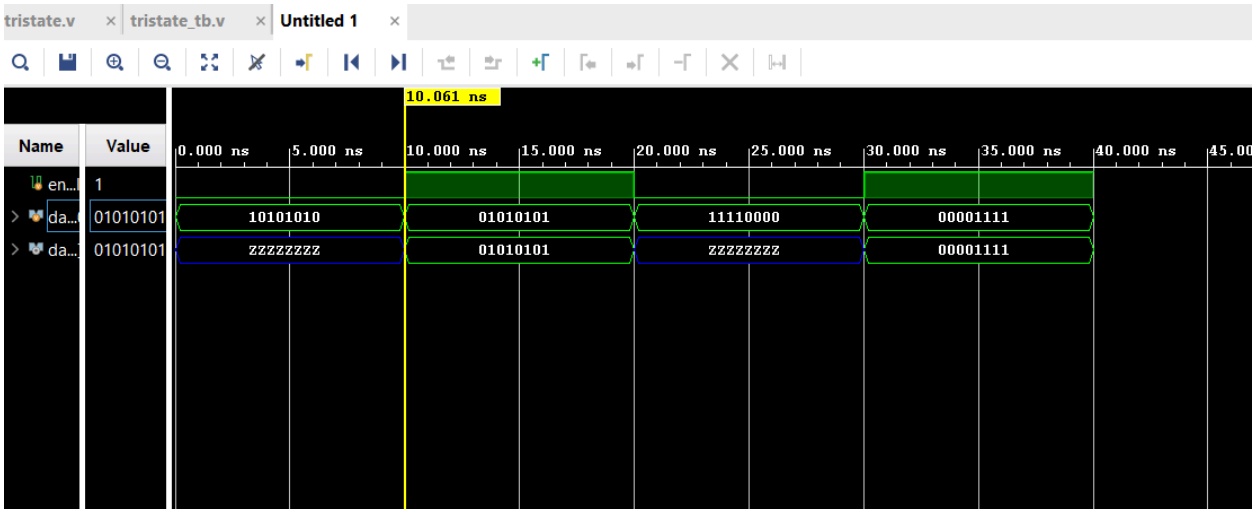
```
`timescale 1ns / 1ps

module tristate_tb();
reg enable;
reg [tristate.n-1:0] data_in;
wire [tristate.n-1:0] data_out;

tristate uut (
    .enable(enable),
    .data_in(data_in),
    .data_out(data_out)
  );

 initial
 begin
 enable = 1'b0;
 data_in = 8'b10101010;
 #10;
 enable = 1'b1;
 data_in = 8'b01010101;
 #10;
 enable = 1'b0;
 data_in = 8'b11110000;
 #10;
 enable = 1'b1;
 data_in = 8'b00001111;
 #10;
 $finish();
 end
endmodule
```

Simulation:



Processor code:

```
`timescale 1ns / 1ps

module processor(
input clock,
input enable,
output wire C_B,
//output wire clk1,
output wire [7:0] accum_wire);

parameter n = 8;
parameter m = 16;
integer i;

//wire clk;
//clock_div cd(clock, clk);

//assign clk1 = clk;

reg [7:0] q, r, accumulator, extended_reg;
reg [3:0] program_counter;
reg c_b;
wire [3:0] opcode, reg_add;
wire [7:0] instruction_code;

reg [n-1:0] registers [0:m-1];

always @(posedge clock, negedge enable)
begin
```

```
if(!enable)
begin
   registers[0] = 8'b00000000;
   registers[1] = 8'b00011111;
   registers[2] = 8'b00011110;
   registers[3] = 8'b01001101;
   registers[4] = 8'b10101100;
   registers[5] = 8'b00111110;
   registers[6] = 8'b00000011;
   registers[7] = 8'b11111111;
   for(i = 8; i<13; i = i+1)
   begin
      registers[i] = 8'b00000000;
   end
   registers[13] = 8'd42;
   registers[14] = 8'd84;
   registers[15] = 8'b00010101;
   program_counter = 4'b0000;
   extended_reg = 8'b00000000;
   accumulator = 8'b00000000;
   c_b = 1'b0;
end
else
begin
   case(opcode)
      4'b0001:
      begin
         {c_b, accumulator} = accumulator + registers[reg_add]; //ADD Ri
         program_counter = program_counter+1;
      end
      4'b0010:
      begin
         {c_b, accumulator} = accumulator - registers[reg_add]; //SUB Ri
         program_counter = program_counter + 1;
      end
      4'b0011:
      begin
         {extended_reg, accumulator} <= accumulator * registers[reg_add]; //MUL Ri
         program_counter = program_counter + 1;
      end
      4'b0100:
      begin
         r = 0;
         q = 0;
```

```
    for(i = n-1; i>=0; i=i-1)
    begin
        r = (r<<1);
        r = r + accumulator[i];
        if(r>registers[reg_add])
        begin
            r = r-registers[reg_add];
            q = (q<<1);
            q = q+1;
        end
        else
        begin
            q = (q<<1);
        end
    end
    accumulator = q;
    extended_reg = r;
    program_counter = program_counter+1;
end
4'b0101:
begin
    accumulator = accumulator & registers[reg_add]; //AND Ri
    program_counter = program_counter + 1;
end
4'b0110:
begin
    accumulator = accumulator ^ registers[reg_add]; //XRA Ri
    program_counter = program_counter + 1;
end
4'b0111:
begin
    c_b = (accumulator < registers[reg_add]); //CMP Ri
    program_counter = program_counter + 1;
end
4'b1001:
begin
    accumulator = registers[reg_add]; //MOV ACC, Ri
    program_counter = program_counter + 1;
end
4'b1010:
begin
    registers[reg_add] = accumulator; //MOV Ri, ACC
    program_counter = program_counter + 1;
end
```

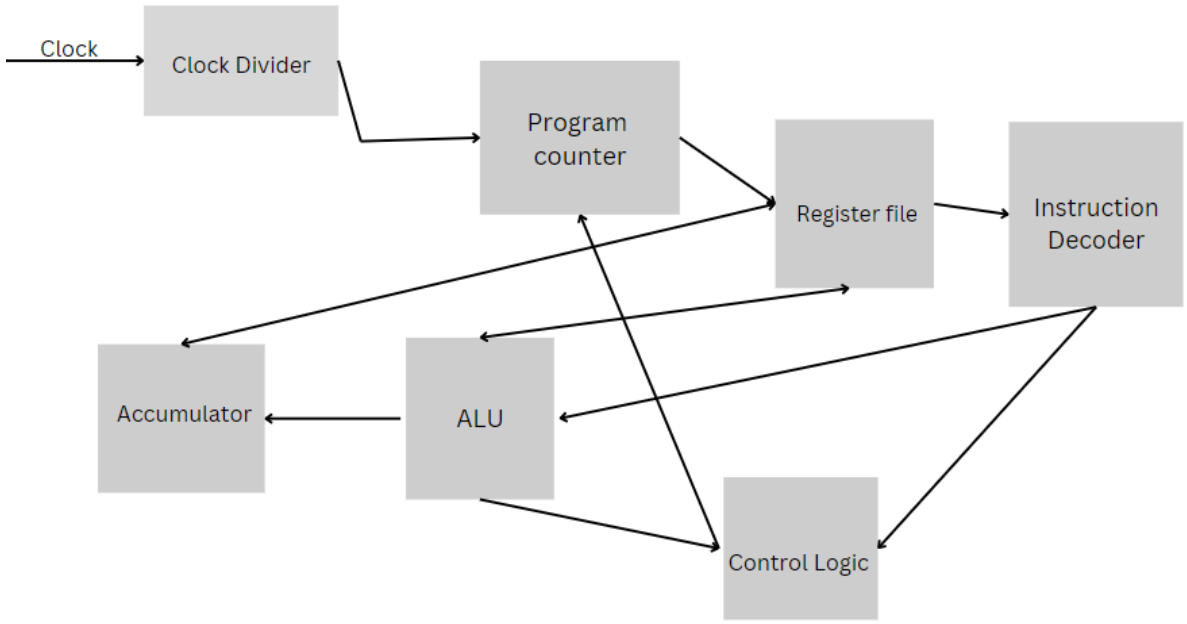
```
4'b1000:
begin
   if (c_b)
   begin
       program_counter = reg_add;
   end //Br <4-bit address>
   else
   begin
      program_counter = program_counter + 1;
   end;
end
4'b1011:
begin
   program_counter = reg_add+1; //Ret <4-bit address>
end
4'b0000:
begin
   case(reg_add)
      4'b0000:
         accumulator = accumulator; //NOP
      4'b0001:
         accumulator = accumulator << 1; //LSL ACC
      4'b0010:
         accumulator = accumulator >> 1; //LSR ACC
      4'b0011:
         accumulator = {accumulator[0], accumulator[7:1]}; //CIR ACC
      4'b0100:
         accumulator = {accumulator[6:0], accumulator[7]}; //CIL ACC
      4'b0101:
         accumulator = accumulator >>> 1; //ASR ACC
      4'b0110:
         {c_b, accumulator} = accumulator + 1; //INC ACC
      4'b0111:
         {c_b, accumulator} = accumulator - 1; //DEC ACC
      default:
         accumulator = accumulator;
   endcase
   program_counter = program_counter+1;
end
4'b1111:
   case(reg_add)
      4'b1111: program_counter = program_counter;
      default: accumulator = accumulator;
   endcase
```

```
            default:
                accumulator = accumulator;
        endcase
    end
end

assign instruction_code = registers[program_counter];
assign opcode = instruction_code [7:4];
assign reg_add = instruction_code [3:0];
assign accum_wire = accumulator;
assign C_B = c_b;

endmodule
```

Block Diagram:



Testbench code:

```
`timescale 1ns / 1ps

module processor_tb();
reg clock, enable;
wire cb, clk;
wire [7:0] accum;
```

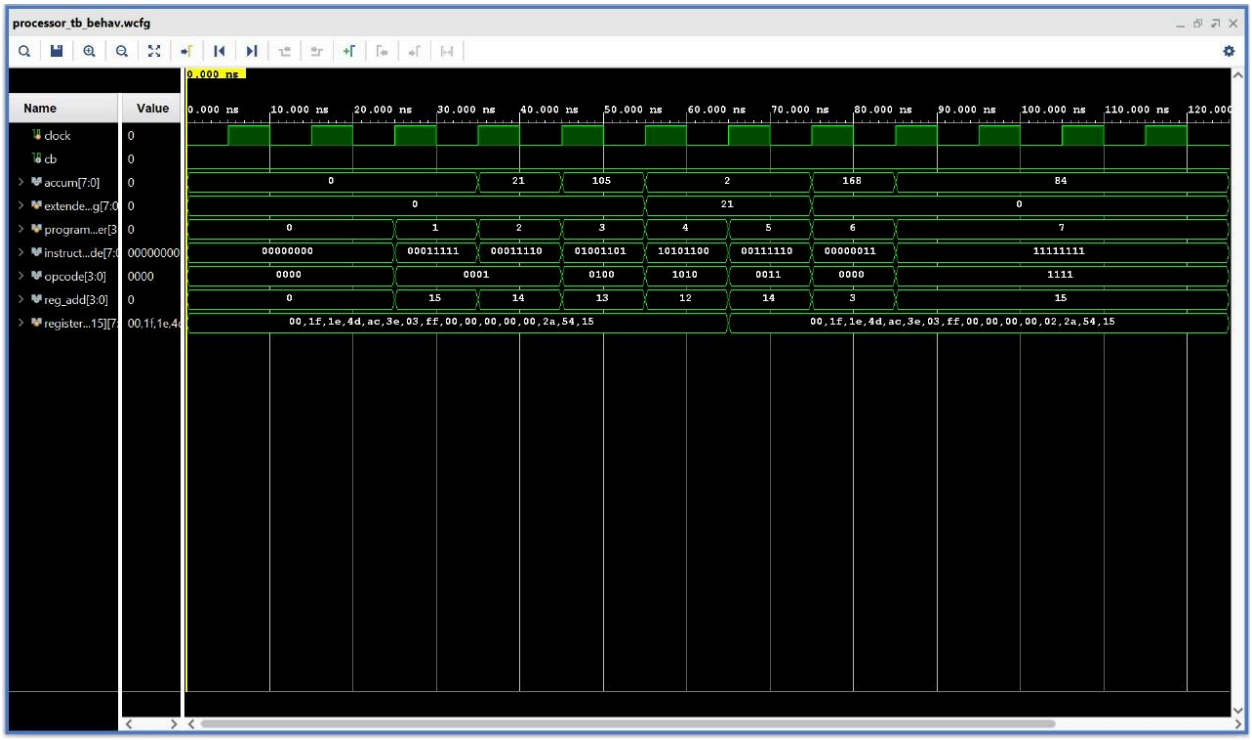
```
processor sm_processor(clock, enable, cb, accum);

initial
begin
    clock = 0;
    forever #5 clock = ~clock;
end

initial
begin
    enable = 0;
    #19;
    enable = 1;
    #106;
    $finish();
end

endmodule
```

Simulation:



Clock divider:

```
`timescale 1ns / 1ps

module clock_div(
input clock,
output slow_clock);

reg [31:0] counter = 0;

always @(posedge clock)
begin
   counter = counter + 1;
end

assign slow_clock = counter[27];
endmodule
```

XDC file:

```
set_property PACKAGE_PIN L1 [get_ports C_B]
set_property IOSTANDARD LVCMOS33 [get_ports C_B]
set_property PACKAGE_PIN P1 [get_ports {accum_wire[7]}]
set_property PACKAGE_PIN N3 [get_ports {accum_wire[6]}]
set_property PACKAGE_PIN P3 [get_ports {accum_wire[5]}]
set_property PACKAGE_PIN U3 [get_ports {accum_wire[4]}]
set_property PACKAGE_PIN W3 [get_ports {accum_wire[3]}]
set_property PACKAGE_PIN V3 [get_ports {accum_wire[2]}]
set_property PACKAGE_PIN V13 [get_ports {accum_wire[1]}]
set_property PACKAGE_PIN V14 [get_ports {accum_wire[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {accum_wire[7]}]
#set_property PACKAGE_PIN U16 [get_ports clk]
#set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN W5 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports clock]
```

```
#set_property PACKAGE_PIN R2 [get_ports reset]
#set_property IOSTANDARD LVCMOS33 [get_ports reset]

set_property PACKAGE_PIN U16 [get_ports clk1]
set_property IOSTANDARD LVCMOS33 [get_ports clk1]

set_property PACKAGE_PIN R2 [get_ports enable]
set_property IOSTANDARD LVCMOS33 [get_ports enable]
```