Design code:

1. ALU:

```
module ADD(a,b,s);
  input [7:0]a,b;
  output [8:0]s;
  assign s=a+b;
endmodule

module SUB(a,b,s);
  input [7:0]a,b;
  output [8:0]s;
  assign s=a-b;
endmodule

module AND(a,b,s);
  input [7:0]a,b;
  output [7:0]s;
  assign s=a&b;
endmodule

module MUL(a,b,s);
  input [7:0]a,b;
  output [15:0]s;
  assign s=a*b;
endmodule

module XRA(a,b,s);
  input [7:0]a,b;
  output [7:0]s;
  assign s=a^b;
endmodule

module CMP(a,b,s);
  input [7:0]a,b;
  output s;
  assign s=(a<b);
endmodule

module ALU(a,b,add,sub,mul,xra,cmp,andd);
  input [7:0]a,b;
  output [8:0]add,sub;
  output [15:0]mul;
  output [7:0]xra,andd;
  output cmp;
```

```
  ADD b0(a,b,add);
  SUB b1(a,b,sub);
  MUL b2(a,b,mul);
  XRA b3(a,b,xra);
  CMP b4(a,b,cmp);
  AND b5(a,b,andd);
endmodule
```

2. Instruction memory:

```
module instruction_memory(pc,instruction);

  input [3:0]pc;
  output [7:0]instruction;
  reg [7:0]memory[0:15];

  initial begin
      memory[0]=8'b00000000;  //acc=0
      memory[1]=8'b00010101; //add 73=>01001001
      memory[2]=8'b00100110;  //sub 49=>00011000
      memory[3]=8'b00111011;  // mul 98=>00110000
      memory[4]=8'b00000001;  //left shift =>01100000
      memory[5]=8'b00000010;  //right shift =>00110000
      memory[6]=8'b00000011;  //circular right shift

=>00011000
      memory[7]=8'b00000100;  //circular left shift
=>00110000
      memory[8]=8'b00000101;  //arithmatic right shift
=>00011000
      memory[9]=8'b01011100;  //and 108 =>00001000
      memory[10]=8'b01100110; //xor 49=>00111001
      memory[11]=8'b01110100; //acc same
      memory[12]=8'b00000110; //acc+1=>00111010
      memory[13]=8'b00000111; //acc-1=>00111001
      memory[14]=8'b10011001; //acc 85=>01010101
      memory[15]=8'b11111111;
  end

  assign instruction = memory[pc];

endmodule
```

3. Top module

```
module top_module(clk,rst,out);
  input clk,rst;
  output reg [7:0]out;
  reg [7:0]register[0:15];

  initial begin
      register[0]=8'd8;
      register[1]=8'd16;
      register[2]=8'd22;
      register[3]=8'd56;
      register[4]=8'd34;
      register[5]=8'd73;
      register[6]=8'd49;
      register[7]=8'd68;
      register[8]=8'd27;
      register[9]=8'd85;
      register[10]=8'd91;
      register[11]=8'd98;
      register[12]=8'd108;
      register[13]=8'd117;
      register[14]=8'd123;
      register[15]=8'd24;
  end

  reg [7:0]acc,ext;
  reg c_b;
  reg [3:0]pc;
  reg [7:0]instruction[0:15];
  wire [7:0]inst=instruction[pc];
  wire [3:0]op=inst[7:4];
  wire [3:0]address=inst[3:0];

  instruction_memory imem (.pc(pc),.instruction(inst));

  wire [8:0]add,sub;
  wire [15:0]mul;
  wire [7:0]xra,andd;
  wire cmp;

  ALU f0(acc,register[address],add,sub,mul,xra,cmp,andd);
```

```
always @(posedge clk or negedge rst) begin
    if(!rst) begin
        pc<=0;
        acc<=0;
        ext<=0;
        c_b<=0;
        register[0]<=8'd8;
        register[1]<=8'd16;
        register[2]<=8'd22;
        register[3]<=8'd56;
        register[4]<=8'd34;
        register[5]<=8'd73;
        register[6]<=8'd49;
        register[7]<=8'd68;
        register[8]<=8'd27;
        register[9]<=8'd85;
        register[10]<=8'd91;
        register[11]<=8'd98;
        register[12]<=8'd108;
        register[13]<=8'd117;
        register[14]<=8'd123;
        register[15]<=8'd148;
    end
    else
      begin
        pc<=pc + 1;
        case(op)
            4'b0000:
                if(address==4'b0000)
                    acc<=acc;
                else if(address==4'b0001)
                  begin
                    acc[7:1]<=acc[6:0];
                    acc[0]<=0;
                  end
                else if(address==4'b0010)
                  begin
                    acc[6:0]<=acc[7:1];
                    acc[7]<=0;
                  end
                else if(address==4'b0011)
                  begin
                    acc[6:0]<=acc[7:1];
                    acc[7]<=acc[0];
                  end
```

```
    else if(address==4'b0100)
      begin
        acc[7:1]<=acc[6:0];
        acc[0]<=acc[7];
      end
    else if(address==4'b0101)
      begin
        acc[6:0]<=acc[7:1];
        acc[7]<=acc[7];
      end
    else if(address==4'b0110)
      begin
        {c_b,acc}<=acc+1;
      end
    else if(address==4'b0111)
      begin
        {c_b,acc}<=acc-1;
      end

4'b0001:
  begin
    c_b<=add[8];
    acc<=add[7:0];
  end
4'b0010:
  begin
    c_b<=sub[8];
    acc<=sub[7:0];
  end
4'b0011:
  begin
    ext<=mul[15:8];
    acc<=mul[7:0];
  end
4'b0101:
  begin
    acc<=andd;
  end
4'b0110:
  begin
    acc<=xra;
  end
4'b0111:
  begin
    c_b<=cmp;
```

```
                    end
                  4'b1000:
                    if(c_b==1)
                      pc<=address;
                  4'b1001:
                    begin
                      acc<=register[address];
                    end
                  4'b1010:
                    begin
                      register[address]<=acc;
                    end
                  4'b1011:
                    begin
                      pc<=address;
                    end
                  4'b1111:
                    begin
                      $finish;
                    end
              endcase

          end
          out<=acc;
      end
    endmodule
```

TESTBENCH:

1. **ALU TESTBENCH:**

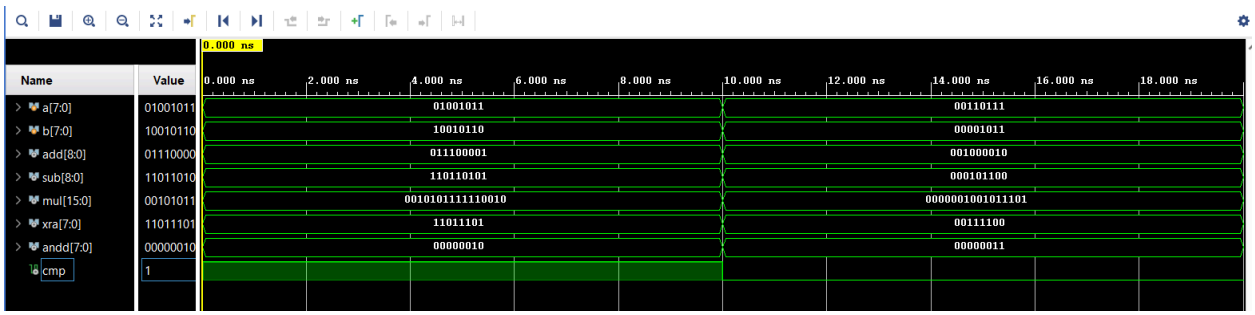
```
`timescale 1ns / 1ps

module ALUtb();

reg [7:0]a,b;
wire [8:0]add,sub;
wire [15:0]mul;
wire [7:0]xra,andd;
wire cmp;

ALU uut (a,b,add,sub,mul,xra,cmp,andd);
```

```
initial begin
$monitor("Time=%0d|a=%d,b=%d|ADD=%d,SUB=%d,MUL=%d,AND=%b,XOR=%b,CM
P=%b",
          $time,a,b,add,sub,mul,andd,xra,cmp);
a=8'b01001011; b=8'b10010110; #10;
a=8'b00110111; b=8'b00001011; #10;
$finish;
end
endmodule
```

2. **TOPMODULE TESTBENCH**

```
`timescale 1ns / 1ps
module top_module_tb;
 reg clk,rst;
 wire [7:0]out;

 top_module uut(clk,rst,out);

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

 initial begin
  $monitor("Time=%0t reset=%b => out=%d",$time,rst,out);
  rst=0; #20;
  rst=1; #200;
  $finish;
 end

endmodule
```

SIMULATION OUTPUTS:



```
`timescale 1ns / 1ps

module instruction_memory(pc,instruction);

  input [3:0]pc;
  output [7:0]instruction;
  reg [7:0]memory[0:15];

    initial begin
      memory[0]=8'b00000000;  //acc=0
      memory[1]=8'b00011111;  //acc=255
      memory[2]=8'b01111000;  //c_b=0 acc as it is
      memory[3]=8'b10001010;  //pc+1, acc as it is
      memory[4]=8'b00000001;  //skipped
      memory[5]=8'b00000001;  //logical left shift==146
      memory[6]=8'b10100111;  //7th register got value of acc 73,acc as it is
      memory[7]=8'b00010111;  //acc=146+73=219=11011011
      memory[8]=8'b10010111; //acc<=73
      memory[7]=8'b10111111; //acc as it is, pc at 15th
      memory[9]=8'b00000000;
      memory[10]=8'b00010101; //add=
      memory[11]=8'b00000000;
      memory[12]=8'b00000000;
      memory[13]=8'b00000000;
      memory[14]=8'b00000000;
      memory[15]=8'b00011101; //acc+98=10101011
  end


  assign instruction = memory[pc];

endmodule
```