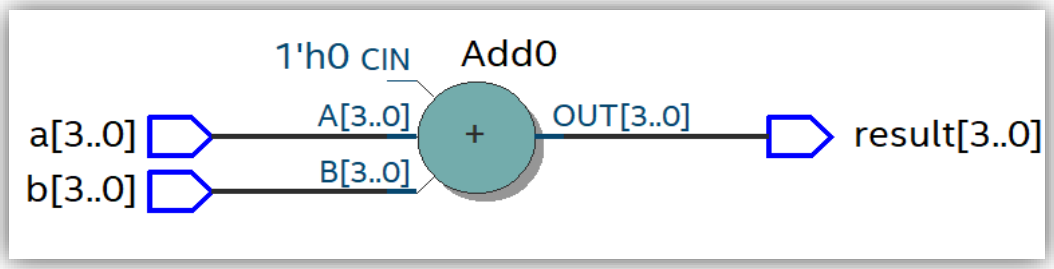
Maven Lab 2

1) Operators

CODE:SUM

```
module sum(input [3:0]a,input [3:0]b,output [3:0]result);
assign result=a+b;
endmodule
```

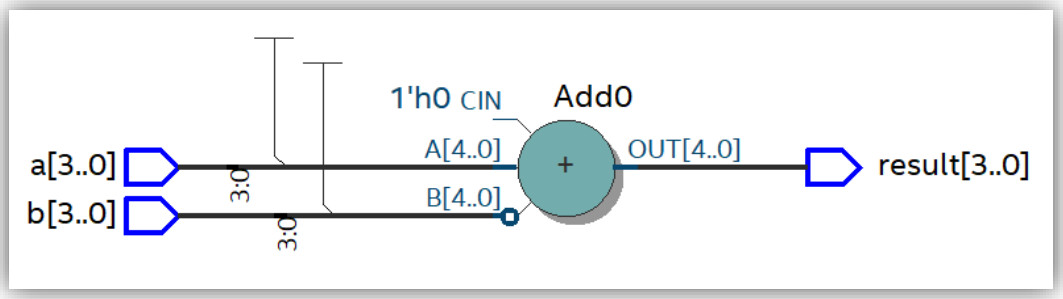
NETLIST:



CODE: SUBTRACTOR

```
module sub(input [3:0]a,input [3:0]b,output [3:0]result);
assign result=a-b;
endmodule
```

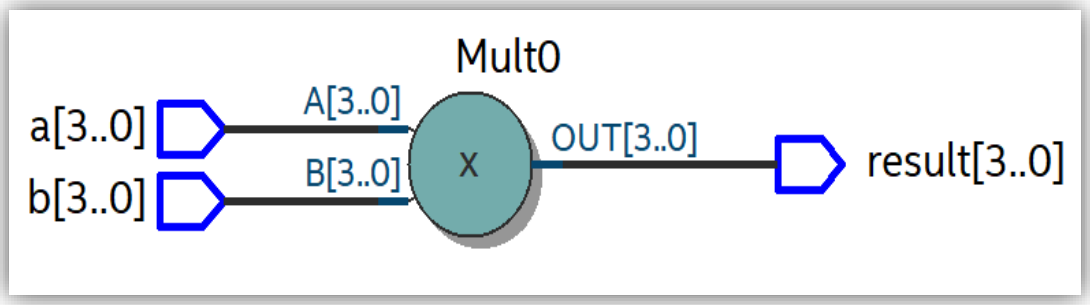
NETLIST:



CODE: MULTIPLICATION

```
module mul(input [3:0]a,input [3:0]b,output [3:0]result);
assign result=a*b;
endmodule
```

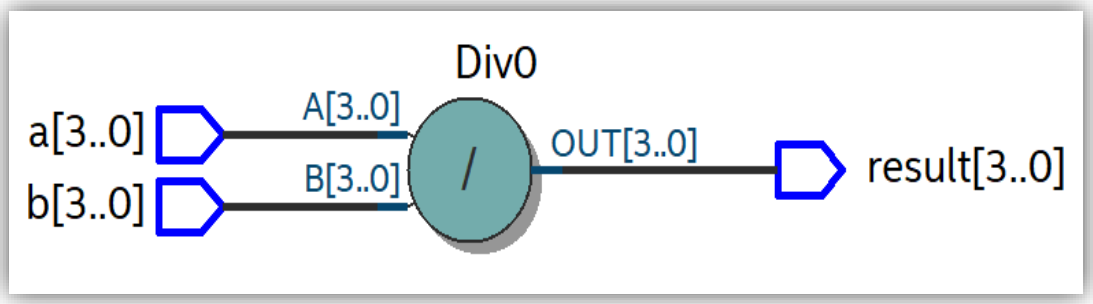
NETLIST:



CODE: DIVISION

```
module div(input [3:0]a,input [3:0]b,output [3:0]result);
assign result=a/b;
endmodule
```

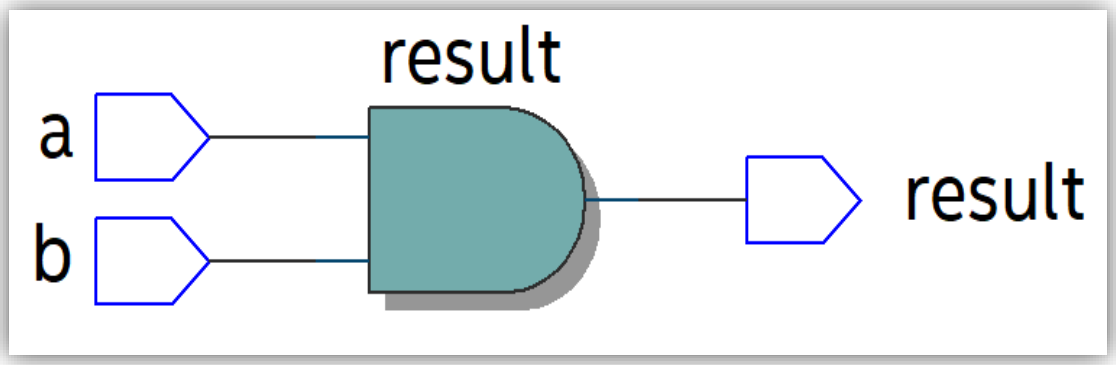
NETLIST:



CODE: AND

```
module and_gate(input a,input b,output result);
assign result=a&b;
endmodule
```

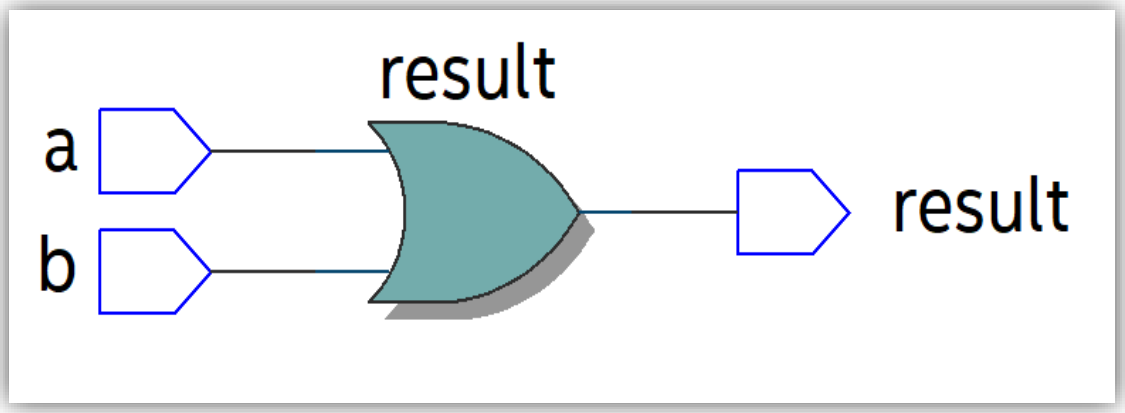
NETLIST:



CODE: OR

```
module or_gate(input a,input b,output result);
assign result=a|b;
endmodule
```

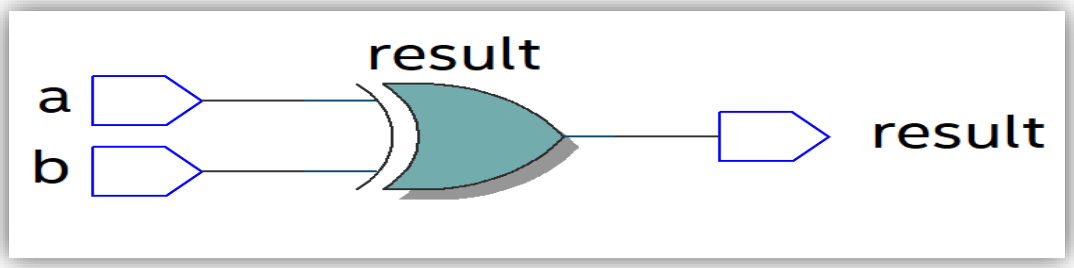
NETLIST:



CODE: XOR

```
module xor_gate(input a,input b,output result);
assign result=a^b;
endmodule
```

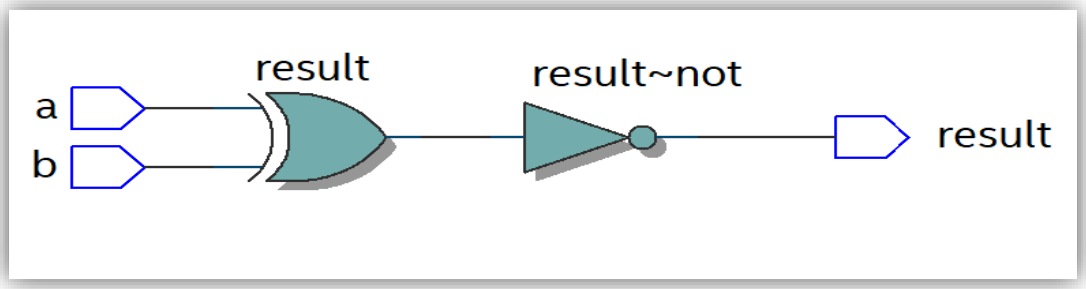
NETLIST:



CODE: EQUAL

```
module equal_op(input a,input b,output result);
assign result=(a==b);
endmodule
```

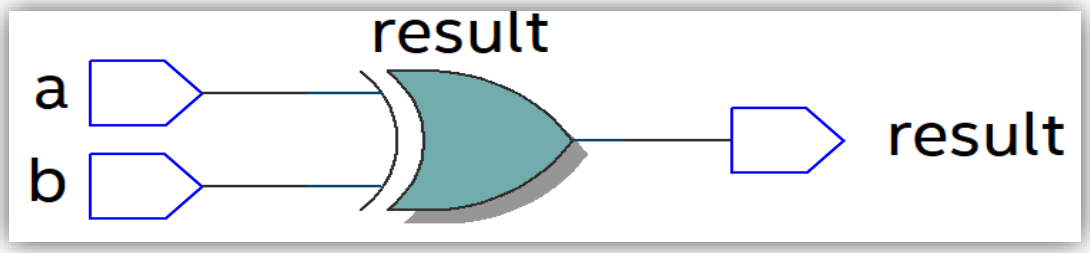
NETLIST:



CODE: EQUAL

```
module equal_op(input a,input b,output result);
assign result=(a==b);
endmodule
```

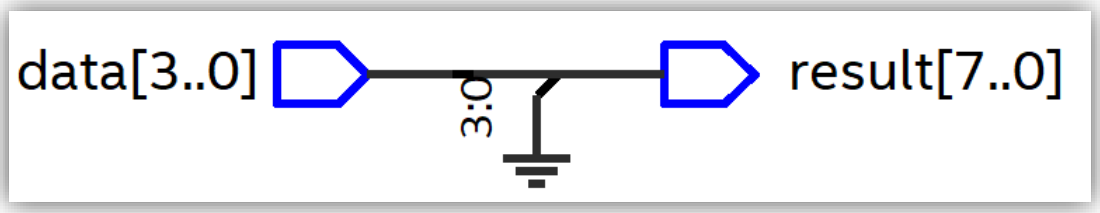
NETLIST:



CODE:LEFT SHIFT

```
module Left_Shift (  input [3:0] data,  output [7:0] result);
assign result = data << 2;
endmodule
```

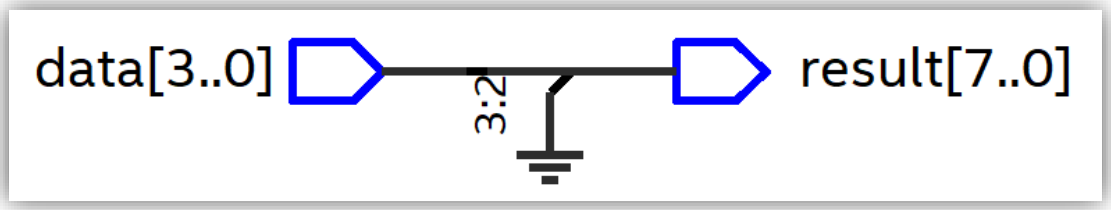
NETLIST:



CODE:RIGHT SHIFT

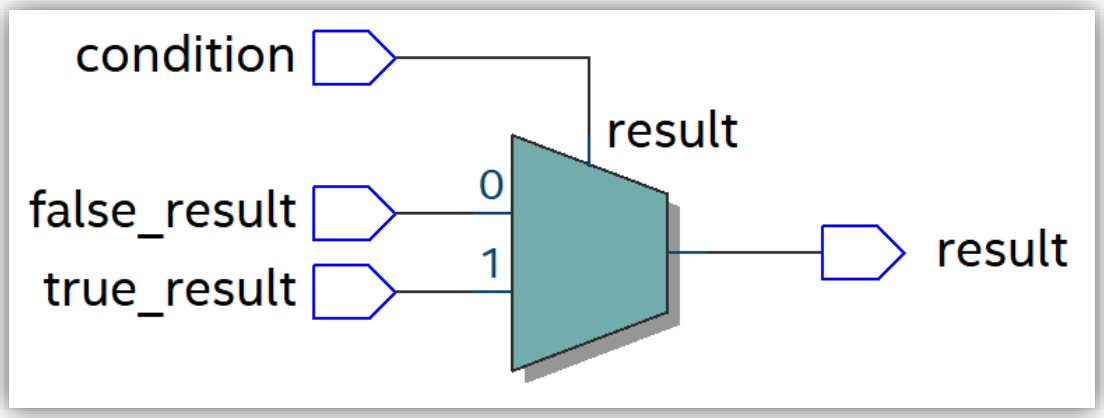
```
module Right_Shift (  input [3:0] data,  output [7:0] result);
assign result = data >> 2;
endmodule
```

NETLIST:



CODE:CONDITION OPERATOR

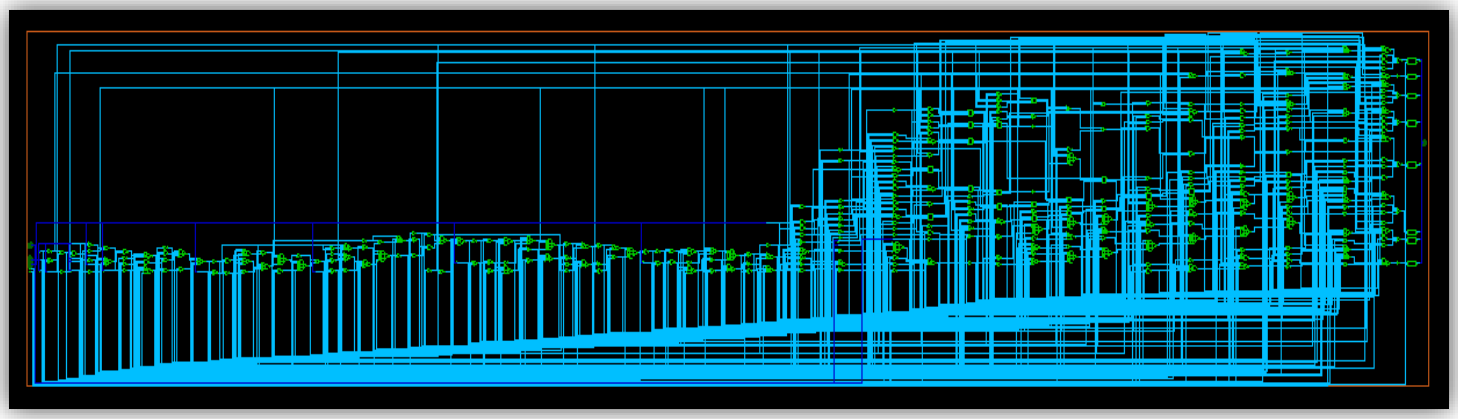
```
module Conditional_Op( input condition,input  true_result,input  false_result,output result);
assign result = (condition) ? true_result : false_result;
endmodule
```

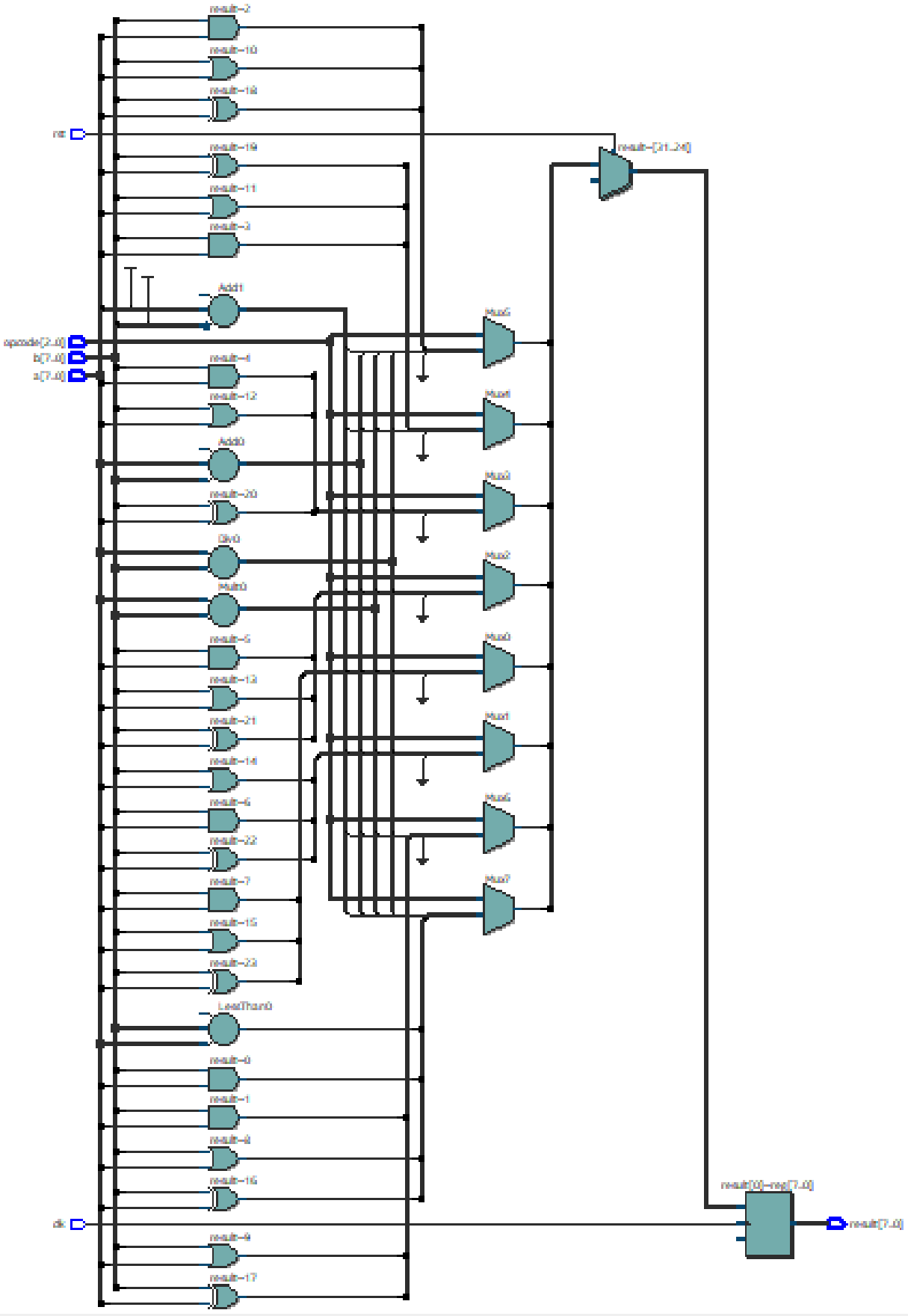


2) ALU

CODE:

```
module ALU(input [7:0]a,input [7:0]b,input [2:0]opcode,input clk,input rst,output reg [7:0]result);
always@(posedge clk)
        begin
                if(rst)
                        begin
                        result=8'b0;
                        end
                else
                        begin
                        case(opcode)
                        3'b000: result<=a+b;
                        3'b001: result<=a-b;
                        3'b010: result<=a*b;
                        3'b011: result<=a/b;
                        3'b100: result<=a&b;
                        3'b101: result<=a|b;
                        3'b110: result<=a^b;
                        3'b111: result<=(a>b)?8'b1:8'b0;
                        endcase
                        end
        end
endmodule
```




result~2
result~10
result~18
rst
result~19
result~11
result~3
Add1
opcode[2..0]
b[7..0]
a[7..0]
result~4
result~12
Add0
result~20
Div0
Mult0
result~5
result~13
result~21
result~14
result~6
result~22
result~7
result~15
result~23
LessThan0
result~0
result~1
result~8
result~16
clk
result~9
result~17
result~[31..24]
Mux5
Mux4
Mux3
Mux2
Mux0
Mux1
Mux6
Mux7
result[0]~reg[7..0]
result[7..0]

TESTBENCH:

```
module ALU_tb;
reg [7:0]a;
reg [7:0]b;
reg [2:0]opcode;
reg clk,rst;
wire [7:0]result;
integer i;
ALU dut(a,b,opcode,clk,rst,result);
initial
        begin
        a=8'b01100011;
        b=8'b00011001;
        clk=0;
        rst=0;
        opcode=8'b0;
        end
always
        begin
        #5 clk=~clk;
        end
initial
        begin
        rst=1;
        #10 rst=0;
    for(i=0;i<8;i=i+1)
        begin
                opcode = i;
                #10;
                end
        end

endmodule
```