1. Briefly discuss about Verilog HDL. Describe its history, uses and features.

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system

like a network switch or a microprocessor or a memory or a flip−flop. It means, by using a HDL we can

describe any digital hardware at any level. Designs, which are described in HDL are independent of technology,

very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

Verilog supports a design at many levels of abstraction. The major three are −

- Behavioral level
- Register-transfer level
- Gate level

Verilog HDL was invented by Phil Moorby and Prabhu Goel around 1984. It served as a proprietary hardware

modeling language owned by Gateway Design Automation Inc. At that time, the language was not standardized.

It modified itself in almost all the revisions that came out between 1984 to 1990.

Uses:

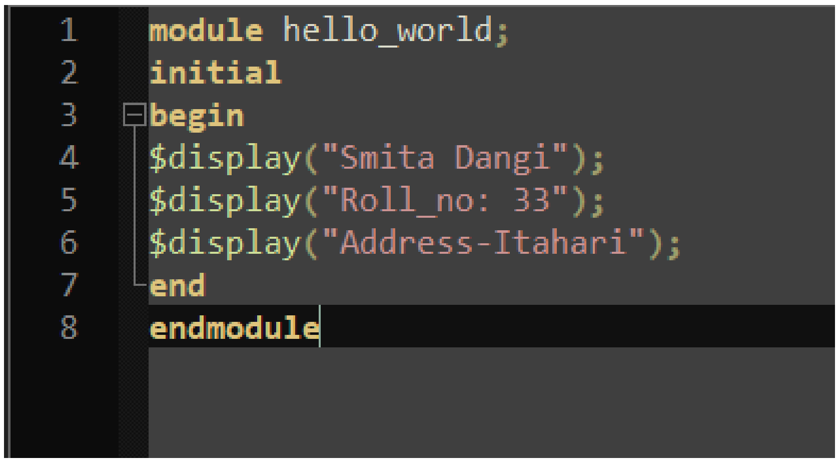
We can use Verilog HDL for designing hardware and for creating test entities to verify the behavior

of a piece of hardware. Verilog HDL is used as an entry format by a variety of EDA tools

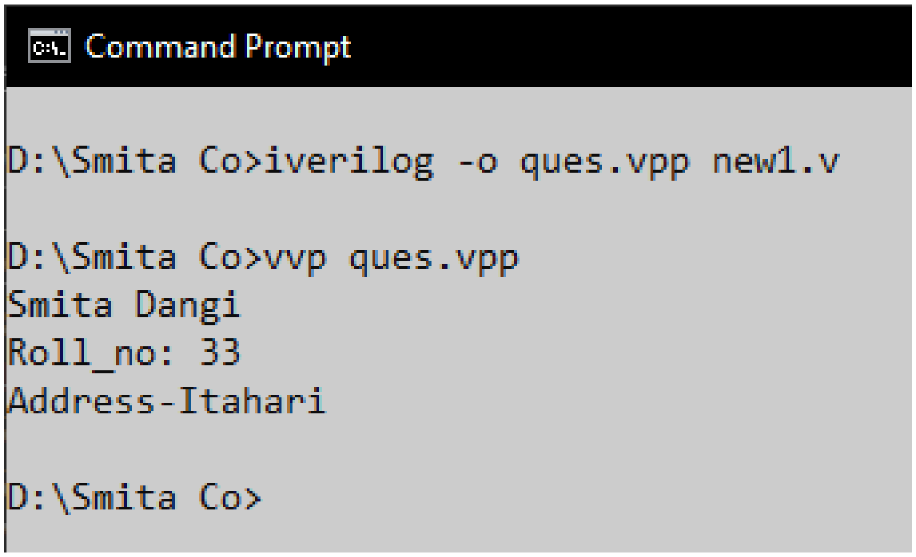
Features:

- Verilog is case sensitive.
- In verilog, Keywords are defined in lower case.
- In Verilog, Most of the syntax is adopted from "C" language.
- Verilog can be used to model a digital circuit at Algorithm, RTL, Gate and Switch level.
- There is no concept of package in Verilog.
- It also supports advanced simulation features like TEXTIO, PLI, and UDPs.

2. Write a program to display Name, Roll no and Address.

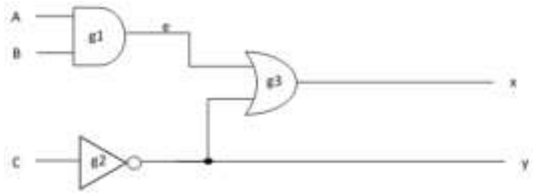


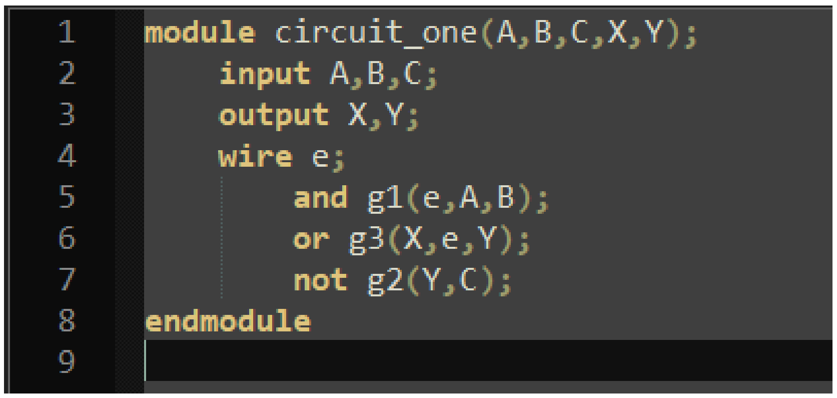
```verilog
module hello_world;
initial
begin
$display("Smita Dangi");
$display("Roll_no: 33");
$display("Address-Itahari");
end
endmodule
```



```
D:\Smita Co>iverilog -o ques.vpp new1.v

D:\Smita Co>vvp ques.vpp
Smita Dangi
Roll_no: 33
Address-Itahari

D:\Smita Co>
```

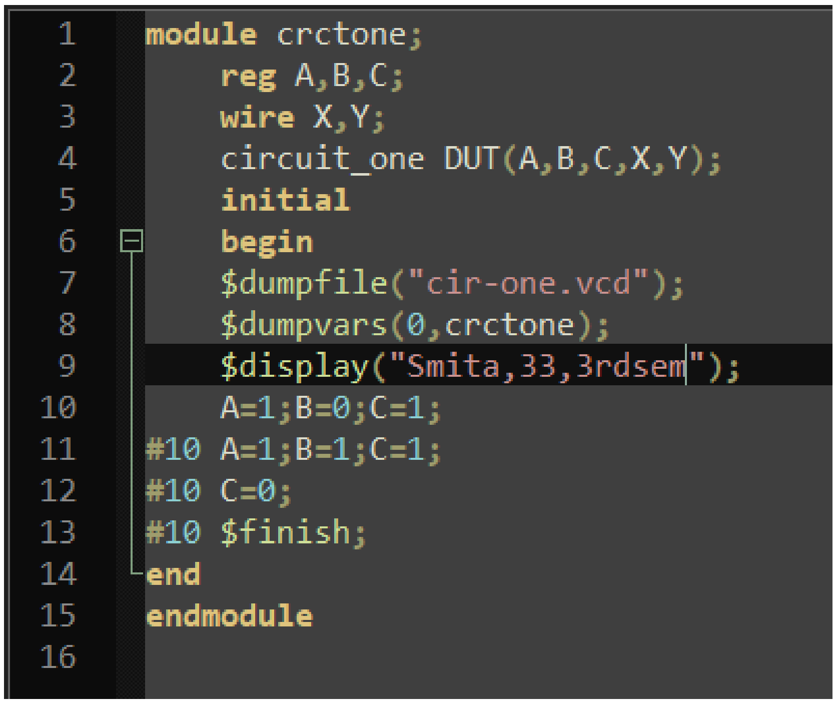
3. Write a program to implement Circuit.





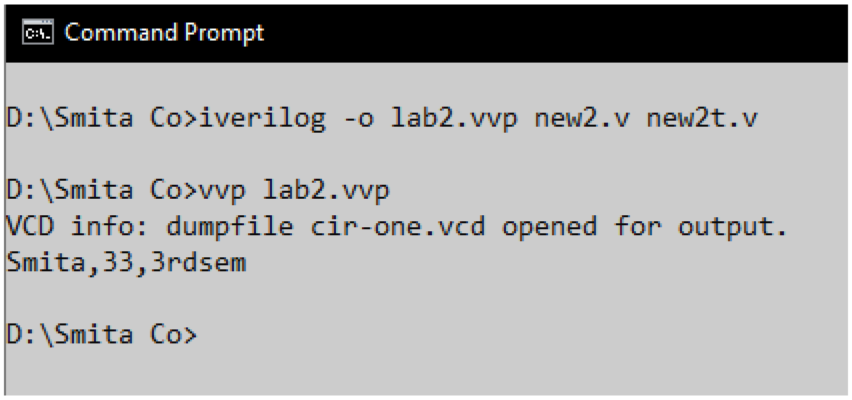
```
module circuit_one(A,B,C,X,Y);
    input A,B,C;
    output X,Y;
    wire e;
        and g1(e,A,B);
        or g3(X,e,Y);
        not g2(Y,C);
endmodule

```

Test.v



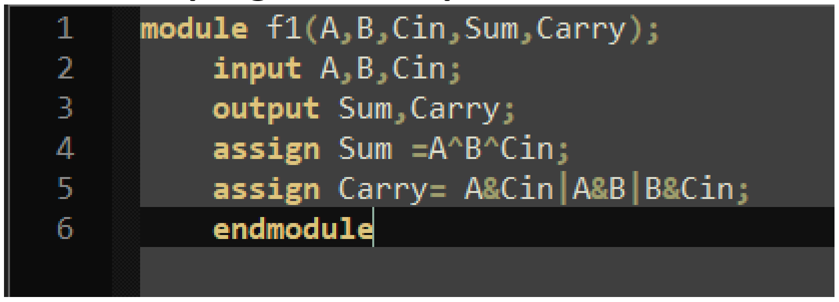
```
module crctone;
    reg A,B,C;
    wire X,Y;
    circuit_one DUT(A,B,C,X,Y);
    initial
    begin
    $dumpfile("cir-one.vcd");
    $dumpvars(0,crctone);
    $display("Smita,33,3rdsem");
    A=1;B=0;C=1;
#10 A=1;B=1;C=1;
#10 C=0;
#10 $finish;
end
endmodule

```

Output:

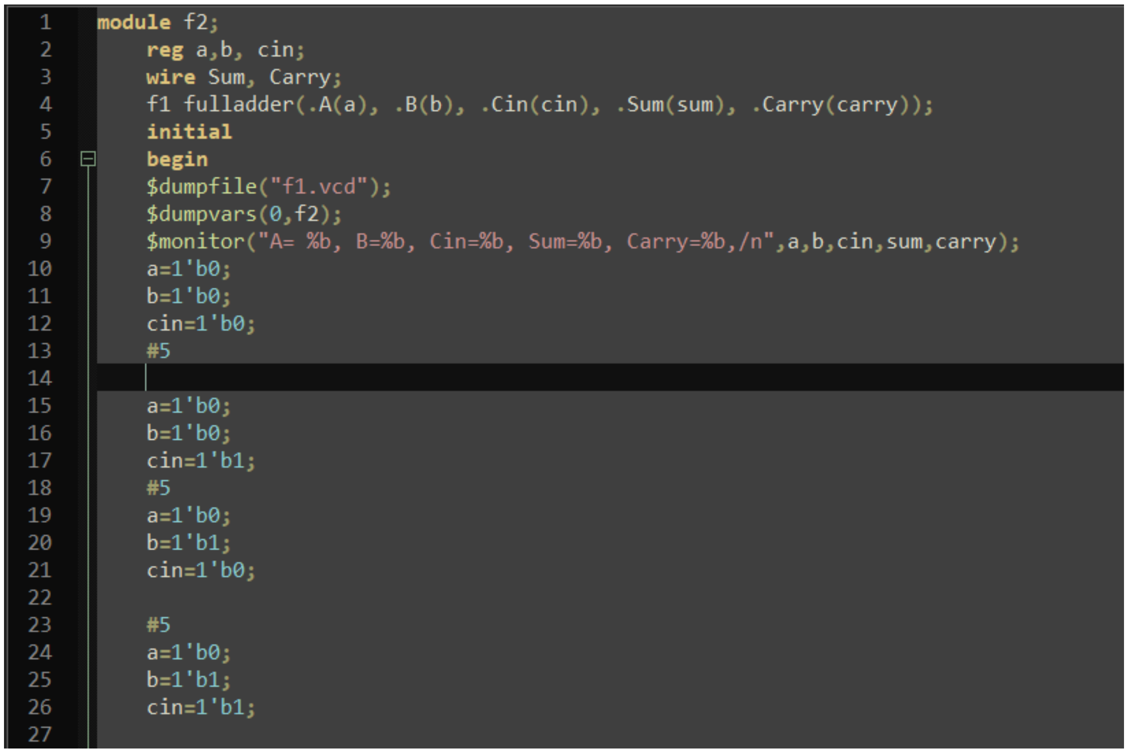


```
D:\Smita Co>iverilog -o lab2.vvp new2.v new2t.v

D:\Smita Co>vvp lab2.vvp
VCD info: dumpfile cir-one.vcd opened for output.
Smita,33,3rdsem

D:\Smita Co>
```

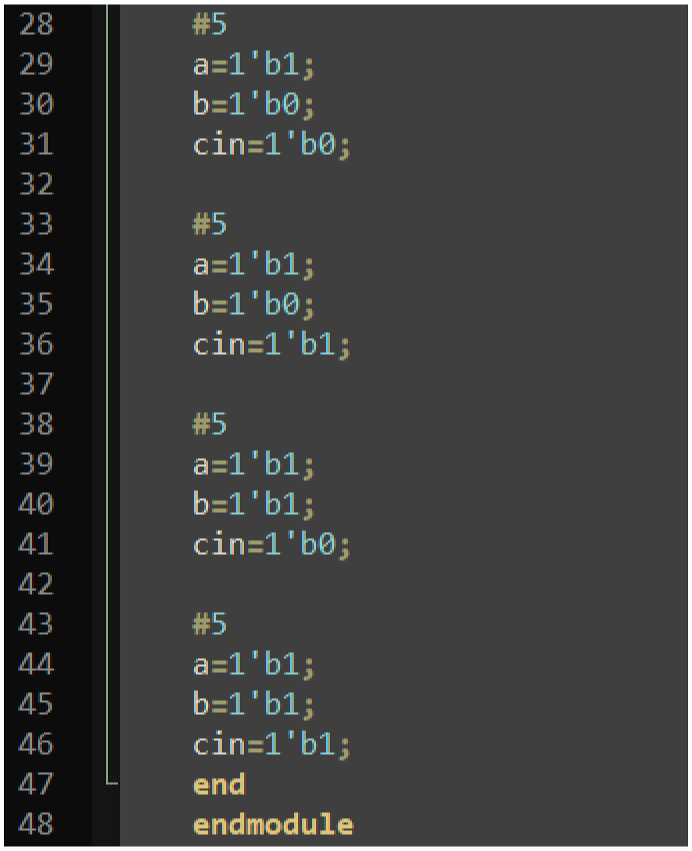
4. Write a program to implement full adder.



```
module f1(A,B,Cin,Sum,Carry);
    input A,B,Cin;
    output Sum,Carry;
    assign Sum =A^B^Cin;
    assign Carry= A&Cin|A&B|B&Cin;
    endmodule
```

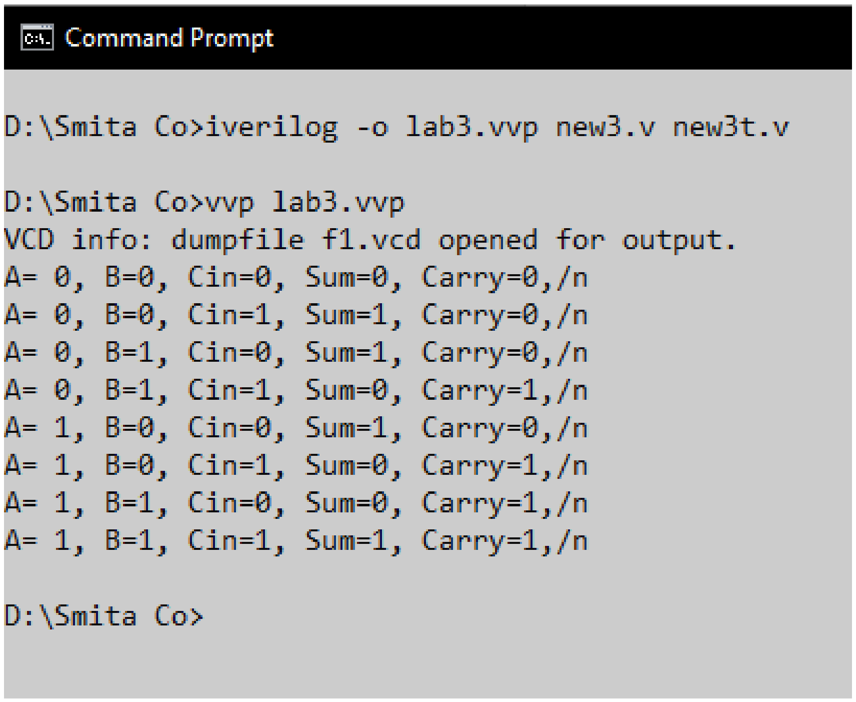


```
module f2;
    reg a,b, cin;
    wire Sum, Carry;
    f1 fulladder(.A(a), .B(b), .Cin(cin), .Sum(sum), .Carry(carry));
    initial
    begin
    $dumpfile("f1.vcd");
    $dumpvars(0,f2);
    $monitor("A= %b, B=%b, Cin=%b, Sum=%b, Carry=%b,/n",a,b,cin,sum,carry);
    a=1'b0;
    b=1'b0;
    cin=1'b0;
    #5

    a=1'b0;
    b=1'b0;
    cin=1'b1;
    #5
    a=1'b0;
    b=1'b1;
    cin=1'b0;

    #5
    a=1'b0;
    b=1'b1;
    cin=1'b1;

```



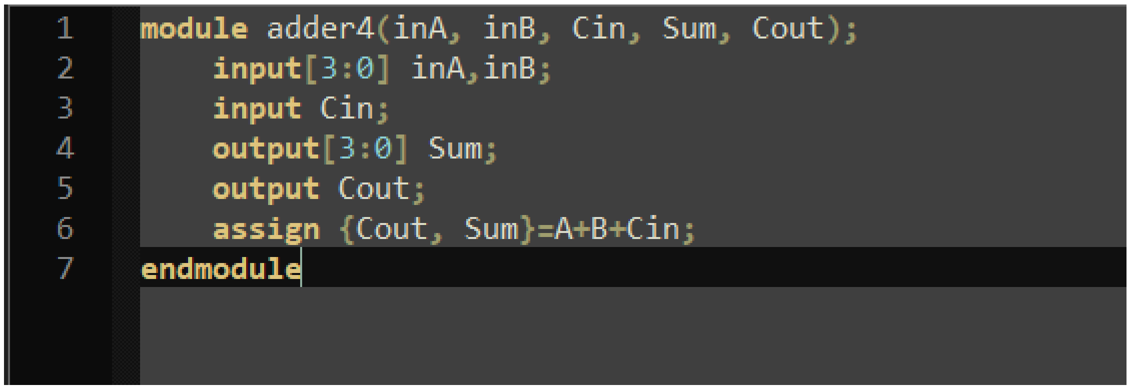
```verilog
    #5
    a=1'b1;
    b=1'b0;
    cin=1'b0;

    #5
    a=1'b1;
    b=1'b0;
    cin=1'b1;

    #5
    a=1'b1;
    b=1'b1;
    cin=1'b0;

    #5
    a=1'b1;
    b=1'b1;
    cin=1'b1;
    end
    endmodule
```

Output:

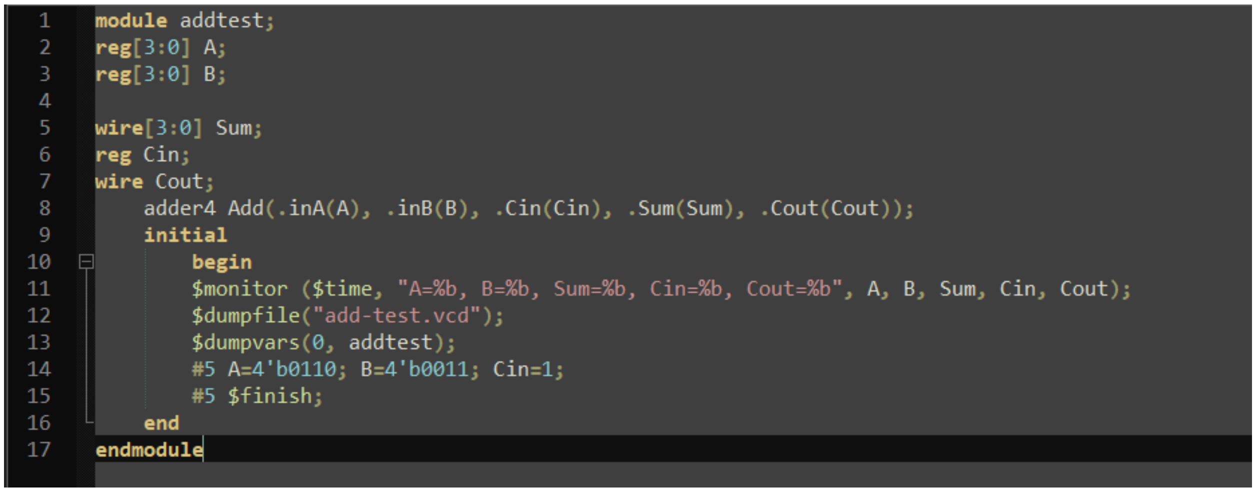


```
Command Prompt

D:\Smita Co>iverilog -o lab3.vvp new3.v new3t.v

D:\Smita Co>vvp lab3.vvp
VCD info: dumpfile f1.vcd opened for output.
A= 0, B=0, Cin=0, Sum=0, Carry=0,/n
A= 0, B=0, Cin=1, Sum=1, Carry=0,/n
A= 0, B=1, Cin=0, Sum=1, Carry=0,/n
A= 0, B=1, Cin=1, Sum=0, Carry=1,/n
A= 1, B=0, Cin=0, Sum=1, Carry=0,/n
A= 1, B=0, Cin=1, Sum=0, Carry=1,/n
A= 1, B=1, Cin=0, Sum=0, Carry=1,/n
A= 1, B=1, Cin=1, Sum=1, Carry=1,/n

D:\Smita Co>
```

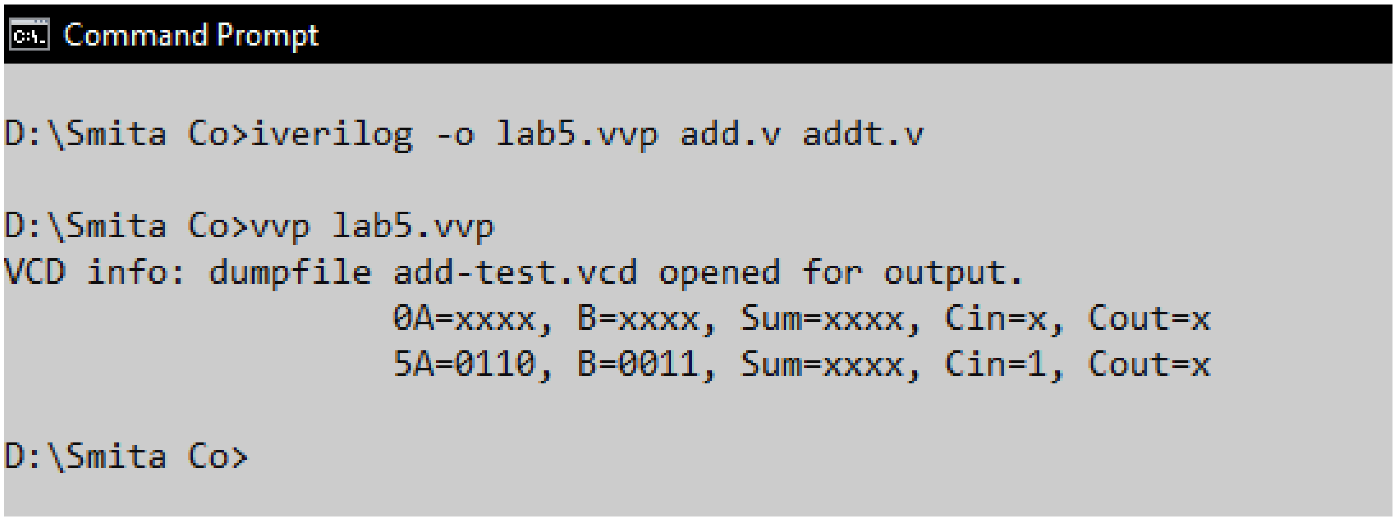
5. Write a program to add two 4-bit numbers and display the overflow if present.



```verilog
module adder4(inA, inB, Cin, Sum, Cout);
    input[3:0] inA,inB;
    input Cin;
    output[3:0] Sum;
    output Cout;
    assign {Cout, Sum}=A+B+Cin;
endmodule
```

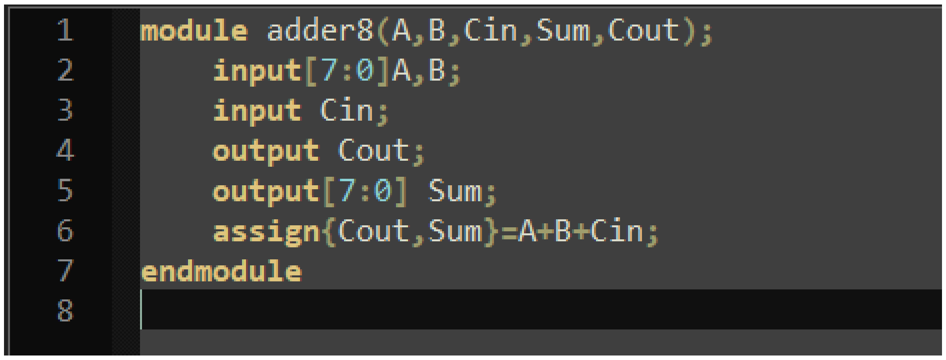


```verilog
module addtest;
reg[3:0] A;
reg[3:0] B;

wire[3:0] Sum;
reg Cin;
wire Cout;
    adder4 Add(.inA(A), .inB(B), .Cin(Cin), .Sum(Sum), .Cout(Cout));
    initial
        begin
        $monitor ($time, "A=%b, B=%b, Sum=%b, Cin=%b, Cout=%b", A, B, Sum, Cin, Cout);
        $dumpfile("add-test.vcd");
        $dumpvars(0, addtest);
        #5 A=4'b0110; B=4'b0011; Cin=1;
        #5 $finish;
    end
endmodule
```

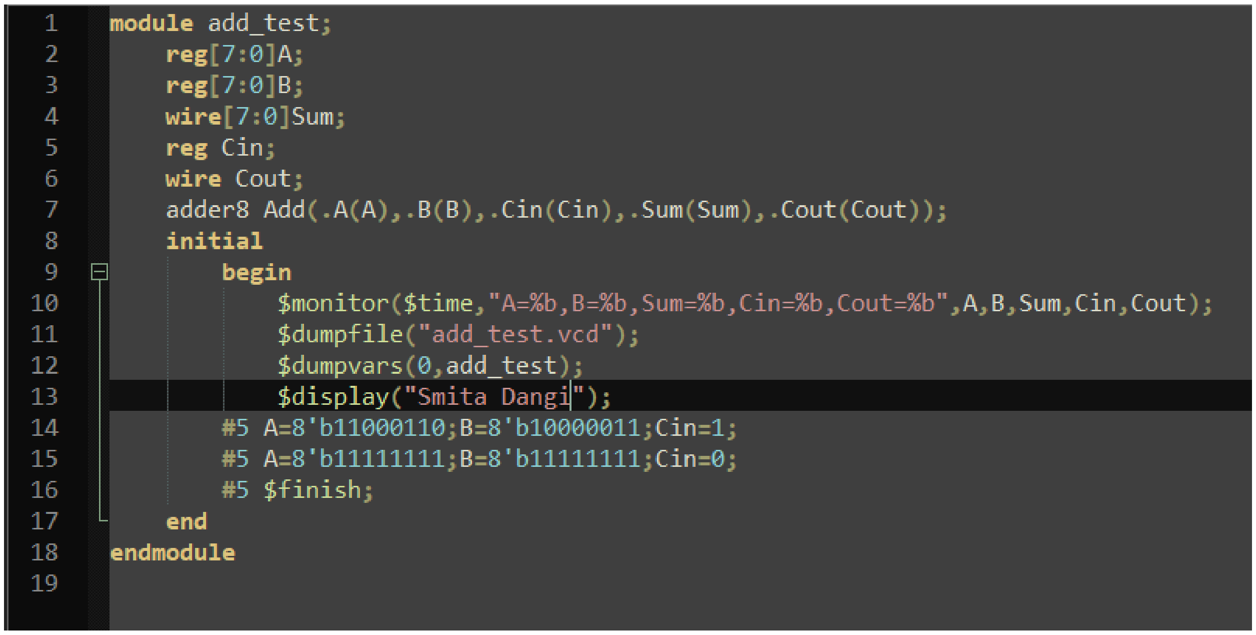


```
Command Prompt

D:\Smita Co>iverilog -o lab5.vvp add.v addt.v

D:\Smita Co>vvp lab5.vvp
VCD info: dumpfile add-test.vcd opened for output.
                   0A=xxxx, B=xxxx, Sum=xxxx, Cin=x, Cout=x
                   5A=0110, B=0011, Sum=xxxx, Cin=1, Cout=x

D:\Smita Co>
```

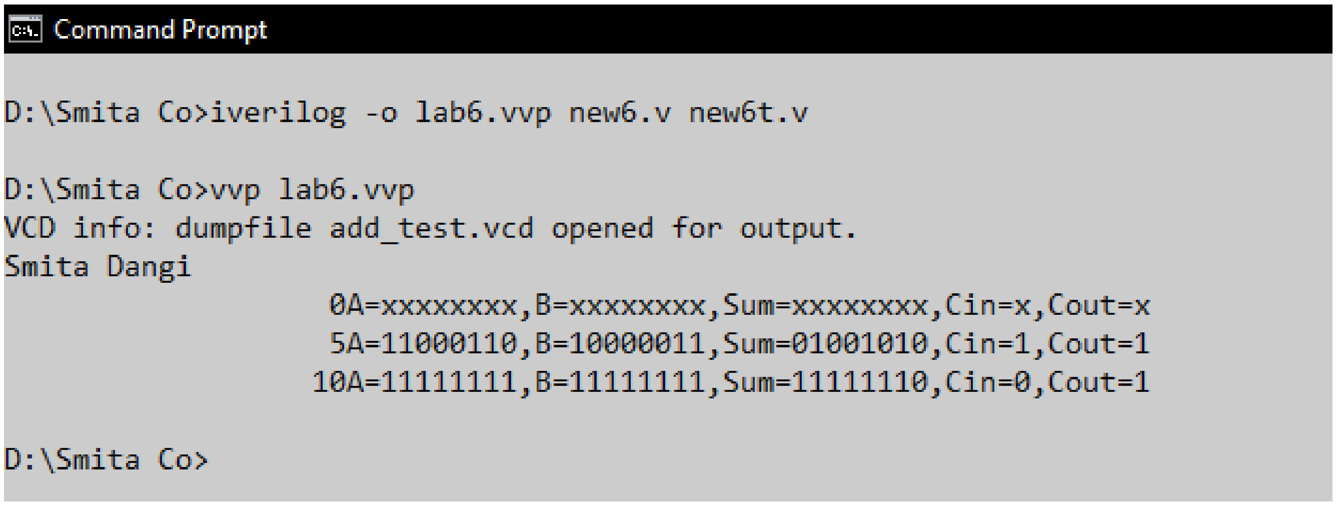
6. Write a program to add two 8-bit numbers and display the overflow if present.



```verilog
module adder8(A,B,Cin,Sum,Cout);
    input[7:0]A,B;
    input Cin;
    output Cout;
    output[7:0] Sum;
    assign{Cout,Sum}=A+B+Cin;
endmodule
```

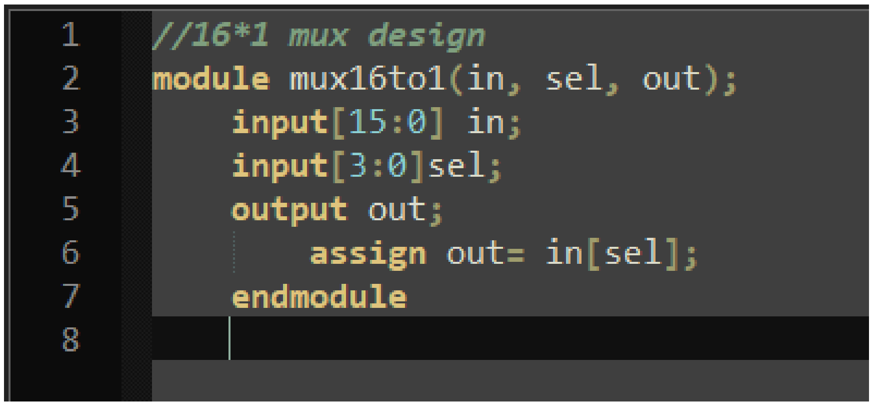


```verilog
module add_test;
    reg[7:0]A;
    reg[7:0]B;
    wire[7:0]Sum;
    reg Cin;
    wire Cout;
    adder8 Add(.A(A),.B(B),.Cin(Cin),.Sum(Sum),.Cout(Cout));
    initial
        begin
            $monitor($time,"A=%b,B=%b,Sum=%b,Cin=%b,Cout=%b",A,B,Sum,Cin,Cout);
            $dumpfile("add_test.vcd");
            $dumpvars(0,add_test);
            $display("Smita Dangi");
        #5 A=8'b11000110;B=8'b10000011;Cin=1;
        #5 A=8'b11111111;B=8'b11111111;Cin=0;
        #5 $finish;
    end
endmodule
```

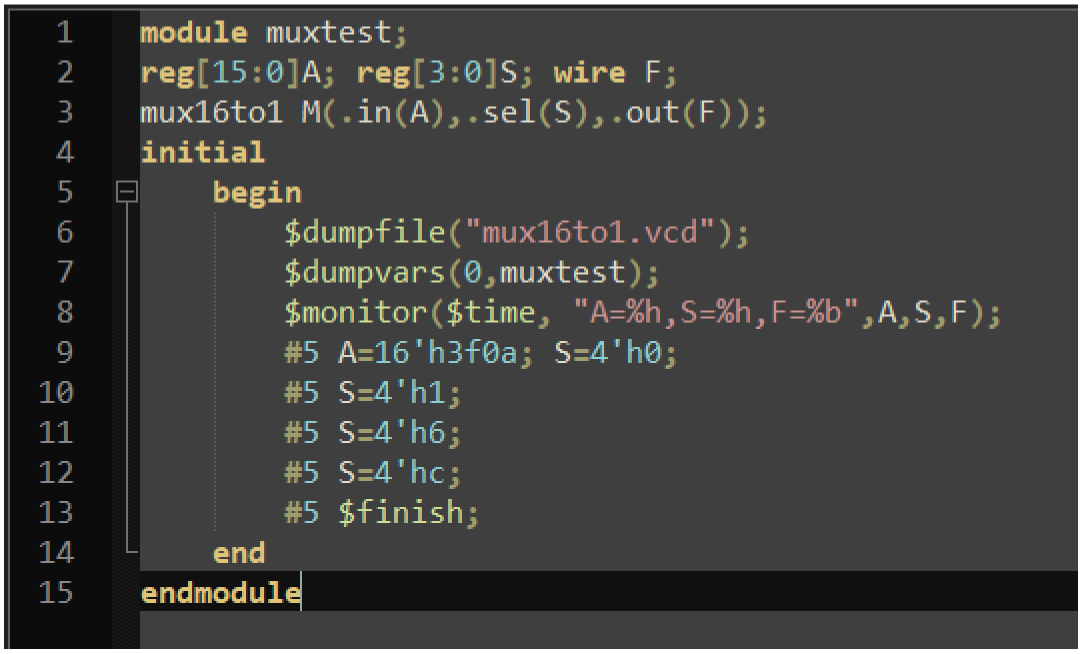


```
Command Prompt

D:\Smita Co>iverilog -o lab6.vvp new6.v new6t.v

D:\Smita Co>vvp lab6.vvp
VCD info: dumpfile add_test.vcd opened for output.
Smita Dangi
                   0A=xxxxxxxx,B=xxxxxxxx,Sum=xxxxxxxx,Cin=x,Cout=x
                   5A=11000110,B=10000011,Sum=01001010,Cin=1,Cout=1
                  10A=11111111,B=11111111,Sum=11111110,Cin=0,Cout=1

D:\Smita Co>
```

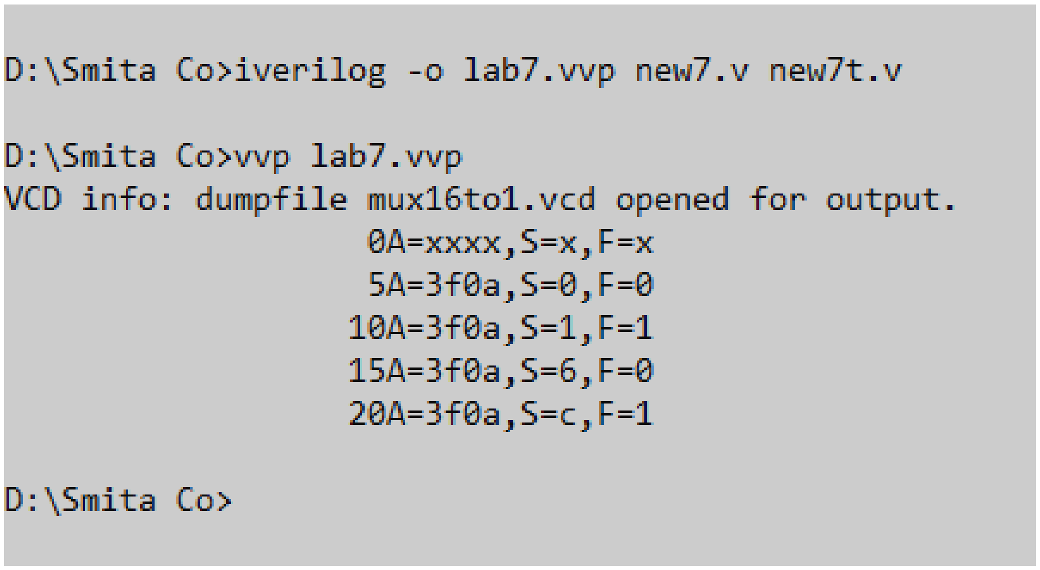
7. Write a program to implement 16X1 multiplexer.4



```verilog
//16*1 mux design
module mux16to1(in, sel, out);
    input[15:0] in;
    input[3:0]sel;
    output out;
        assign out= in[sel];
    endmodule

```

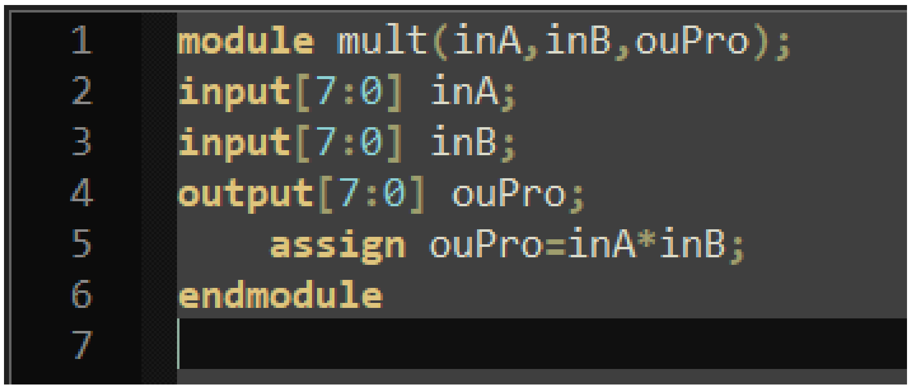


```verilog
module muxtest;
reg[15:0]A; reg[3:0]S; wire F;
mux16to1 M(.in(A),.sel(S),.out(F));
initial
    begin
        $dumpfile("mux16to1.vcd");
        $dumpvars(0,muxtest);
        $monitor($time, "A=%h,S=%h,F=%b",A,S,F);
        #5 A=16'h3f0a; S=4'h0;
        #5 S=4'h1;
        #5 S=4'h6;
        #5 S=4'hc;
        #5 $finish;
    end
endmodule
```

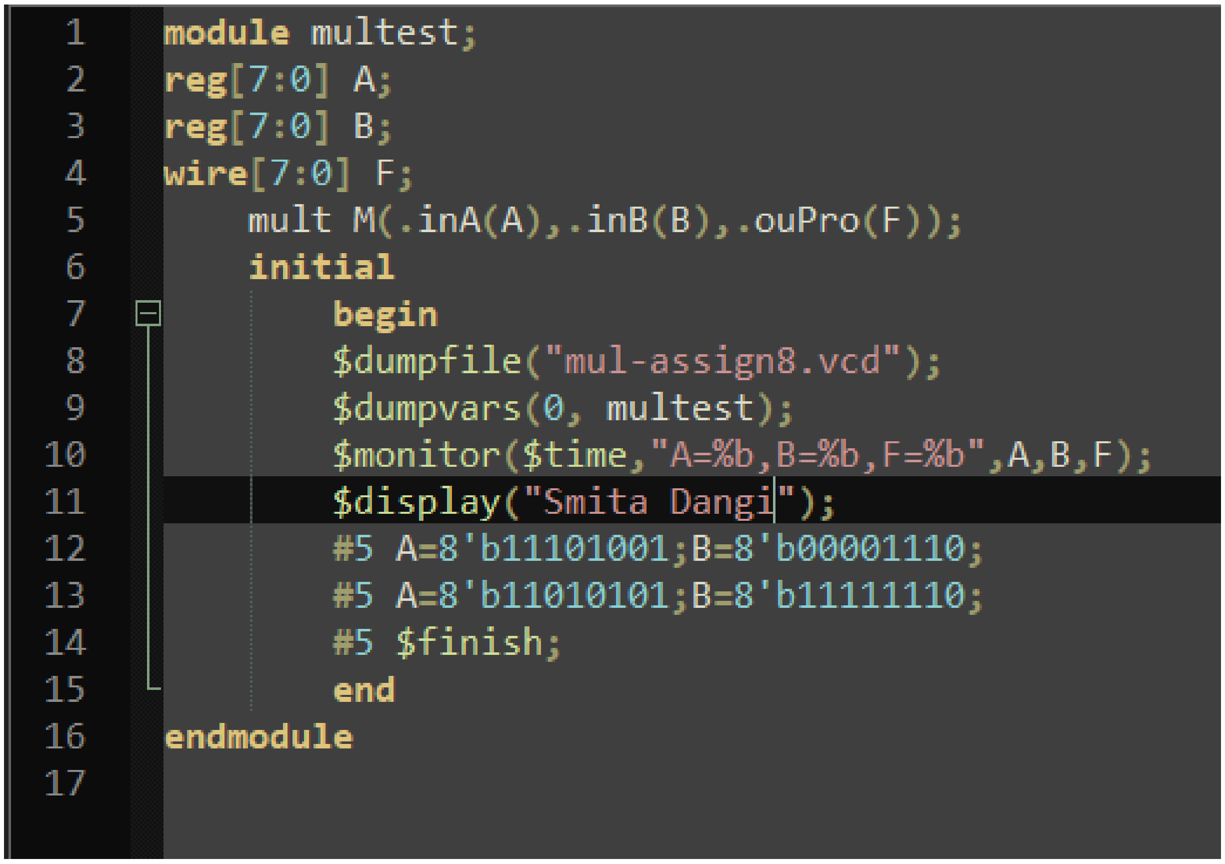


```
D:\Smita Co>iverilog -o lab7.vvp new7.v new7t.v

D:\Smita Co>vvp lab7.vvp
VCD info: dumpfile mux16to1.vcd opened for output.
                   0A=xxxx,S=x,F=x
                   5A=3f0a,S=0,F=0
                  10A=3f0a,S=1,F=1
                  15A=3f0a,S=6,F=0
                  20A=3f0a,S=c,F=1

D:\Smita Co>
```

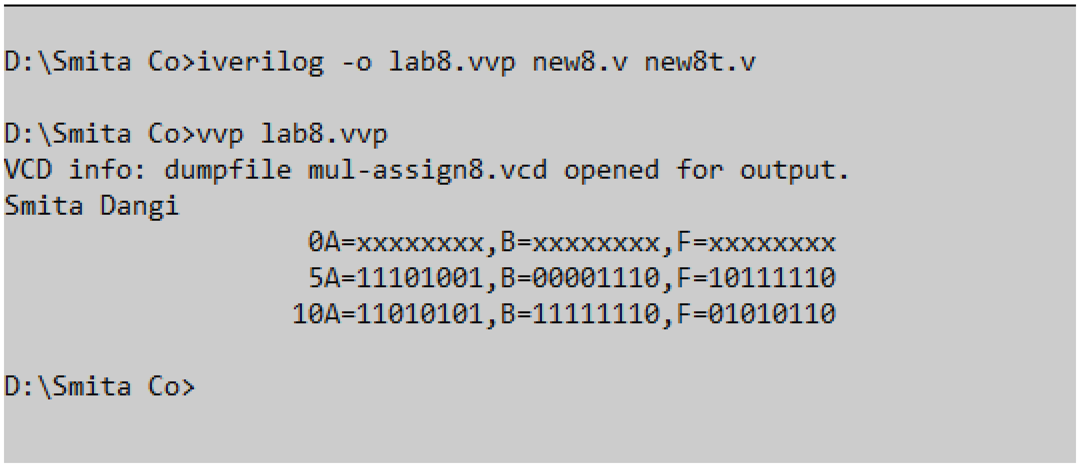
8. Write a program to implement 8X1 multiplexer



```
module mult(inA,inB,ouPro);
input[7:0] inA;
input[7:0] inB;
output[7:0] ouPro;
    assign ouPro=inA*inB;
endmodule
```

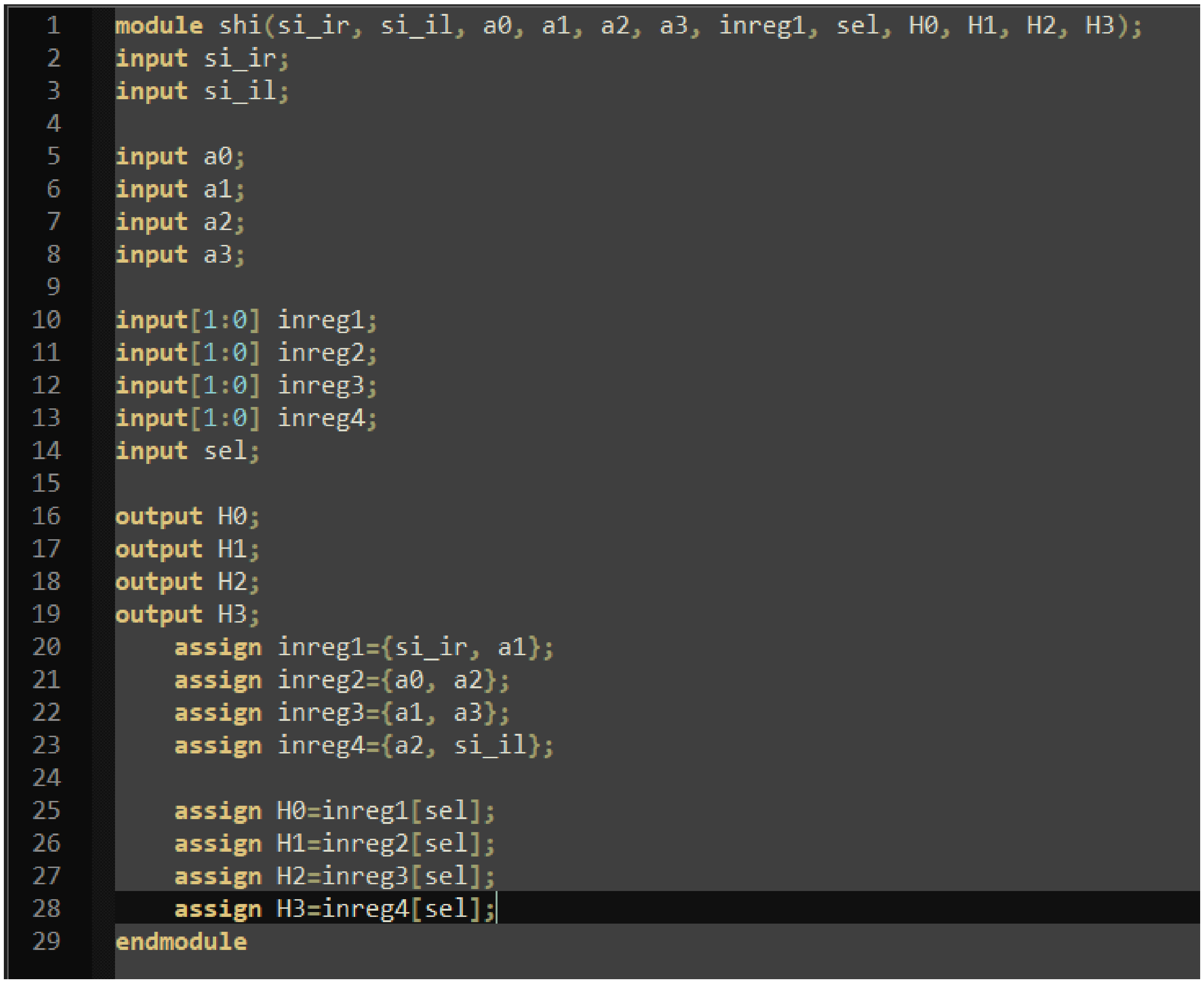


```
module multest;
reg[7:0] A;
reg[7:0] B;
wire[7:0] F;
    mult M(.inA(A),.inB(B),.ouPro(F));
    initial
        begin
        $dumpfile("mul-assign8.vcd");
        $dumpvars(0, multest);
        $monitor($time,"A=%b,B=%b,F=%b",A,B,F);
        $display("Smita Dangi");
        #5 A=8'b11101001;B=8'b00001110;
        #5 A=8'b11010101;B=8'b11111110;
        #5 $finish;
        end
endmodule
```

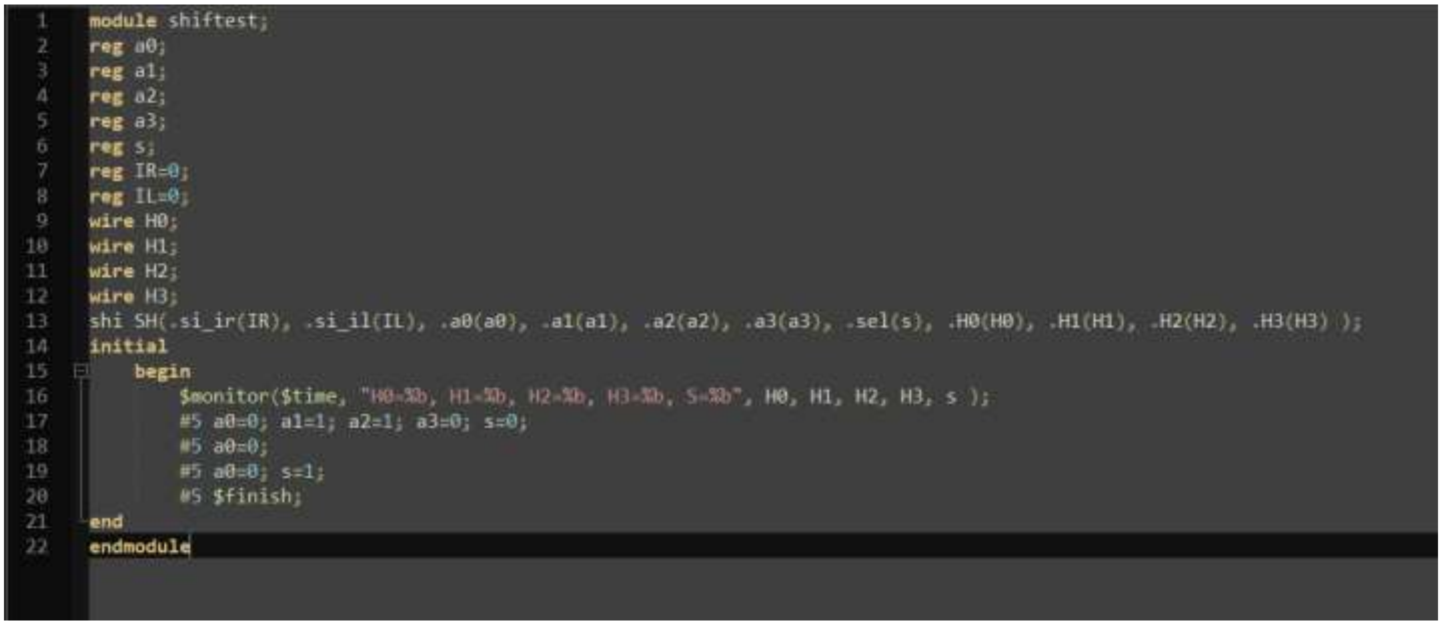


```
D:\Smita Co>iverilog -o lab8.vvp new8.v new8t.v

D:\Smita Co>vvp lab8.vvp
VCD info: dumpfile mul-assign8.vcd opened for output.
Smita Dangi
                   0A=xxxxxxxx,B=xxxxxxxx,F=xxxxxxxx
                   5A=11101001,B=00001110,F=10111110
                  10A=11010101,B=11111110,F=01010110

D:\Smita Co>
```

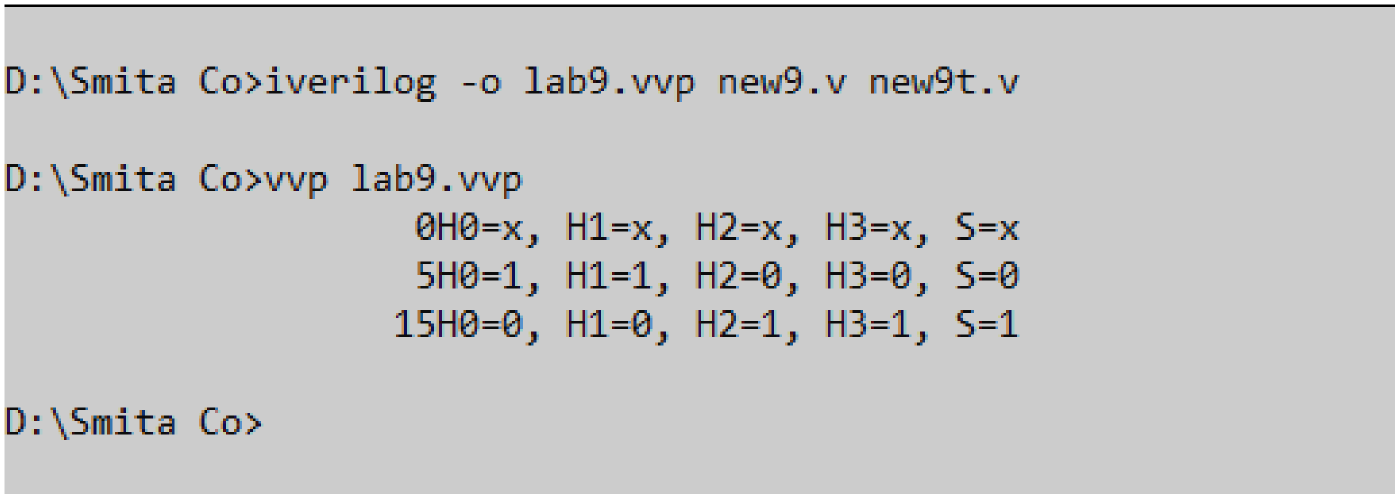
9. Write a program to perform hardware implementation of Shift microoperation.



```verilog
module shi(si_ir, si_il, a0, a1, a2, a3, inreg1, sel, H0, H1, H2, H3);
input si_ir;
input si_il;

input a0;
input a1;
input a2;
input a3;

input[1:0] inreg1;
input[1:0] inreg2;
input[1:0] inreg3;
input[1:0] inreg4;
input sel;

output H0;
output H1;
output H2;
output H3;
    assign inreg1={si_ir, a1};
    assign inreg2={a0, a2};
    assign inreg3={a1, a3};
    assign inreg4={a2, si_il};

    assign H0=inreg1[sel];
    assign H1=inreg2[sel];
    assign H2=inreg3[sel];
    assign H3=inreg4[sel];
endmodule
```

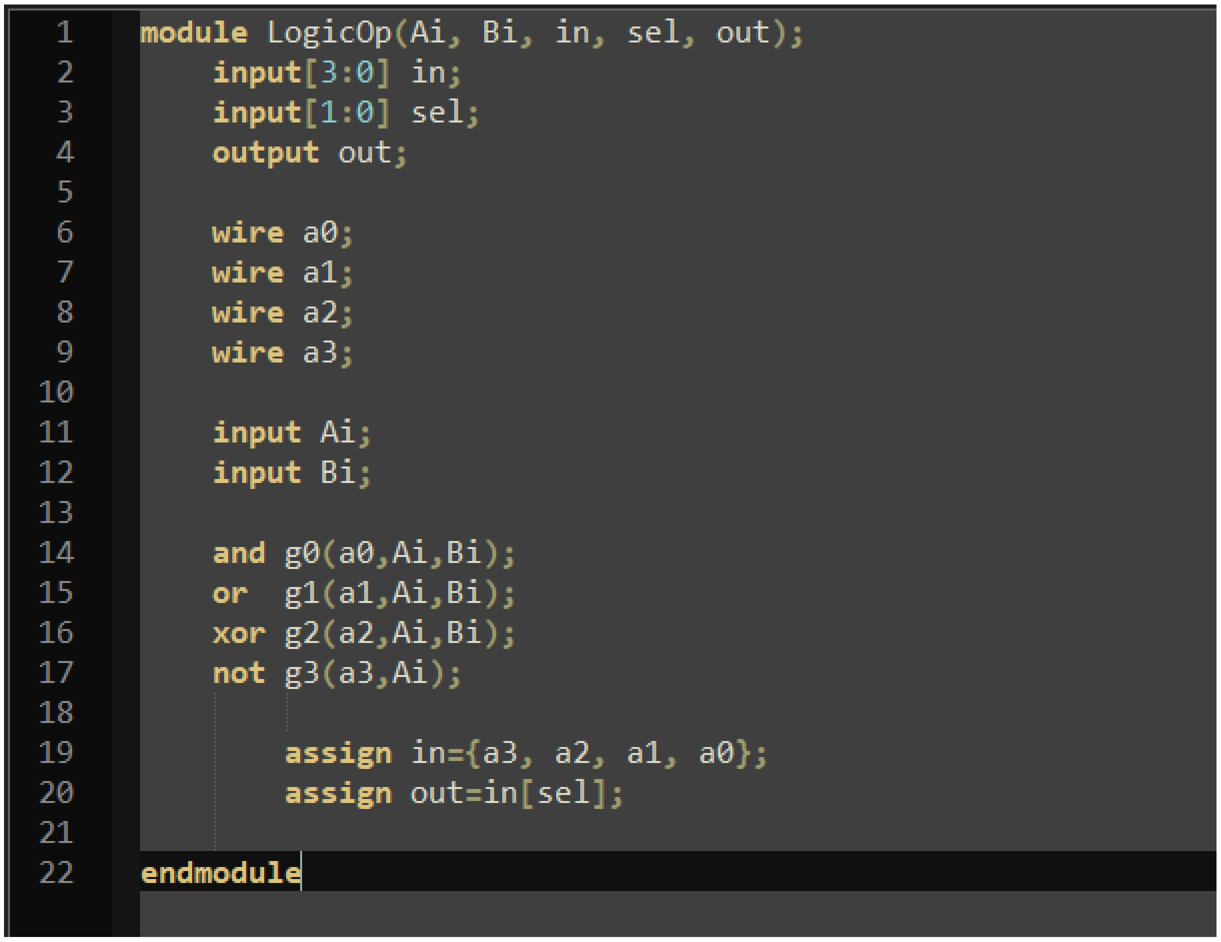


```
module shiftest;
reg a0;
reg a1;
reg a2;
reg a3;
reg s;
reg IR=0;
reg IL=0;
wire H0;
wire H1;
wire H2;
wire H3;
shi SH(.si_ir(IR), .si_il(IL), .a0(a0), .a1(a1), .a2(a2), .a3(a3), .sel(s), .H0(H0), .H1(H1), .H2(H2), .H3(H3) );
initial
    begin
        $monitor($time, "H0=%b, H1=%b, H2=%b, H3=%b, S=%b", H0, H1, H2, H3, s );
        #5 a0=0; a1=1; a2=1; a3=0; s=0;
        #5 a0=0;
        #5 a0=0; s=1;
        #5 $finish;
end
endmodule
```

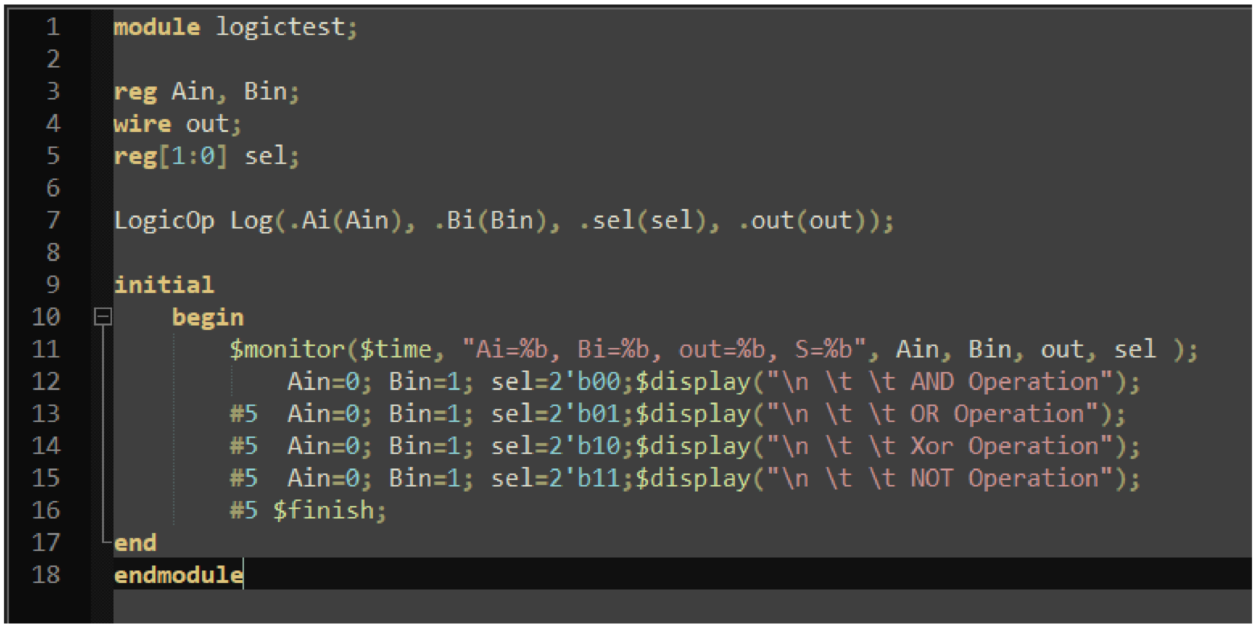


```
D:\Smita Co>iverilog -o lab9.vvp new9.v new9t.v

D:\Smita Co>vvp lab9.vvp
                   0H0=x, H1=x, H2=x, H3=x, S=x
                   5H0=1, H1=1, H2=0, H3=0, S=0
                  15H0=0, H1=0, H2=1, H3=1, S=1

D:\Smita Co>
```

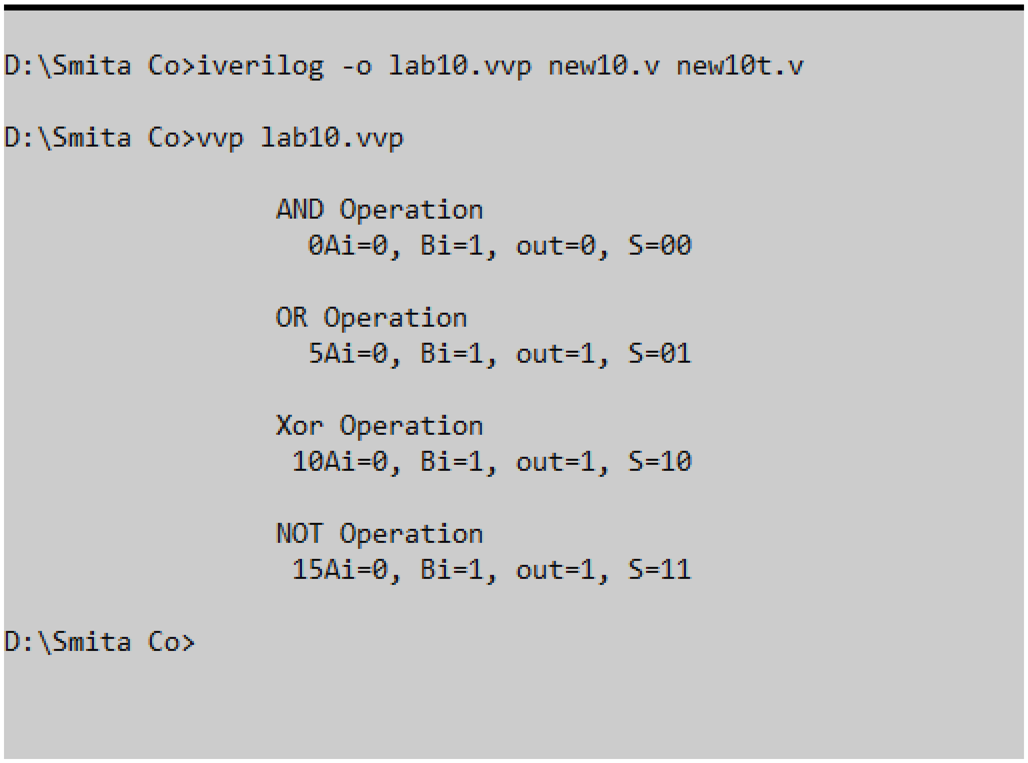
10. Write a program to perform hardware implementation of logical microoperation.



```verilog
module LogicOp(Ai, Bi, in, sel, out);
    input[3:0] in;
    input[1:0] sel;
    output out;

    wire a0;
    wire a1;
    wire a2;
    wire a3;

    input Ai;
    input Bi;

    and g0(a0,Ai,Bi);
    or  g1(a1,Ai,Bi);
    xor g2(a2,Ai,Bi);
    not g3(a3,Ai);

        assign in={a3, a2, a1, a0};
        assign out=in[sel];

endmodule
```



```verilog
module logictest;

reg Ain, Bin;
wire out;
reg[1:0] sel;

LogicOp Log(.Ai(Ain), .Bi(Bin), .sel(sel), .out(out));

initial
    begin
        $monitor($time, "Ai=%b, Bi=%b, out=%b, S=%b", Ain, Bin, out, sel );
            Ain=0; Bin=1; sel=2'b00;$display("\n \t \t AND Operation");
        #5  Ain=0; Bin=1; sel=2'b01;$display("\n \t \t OR Operation");
        #5  Ain=0; Bin=1; sel=2'b10;$display("\n \t \t Xor Operation");
        #5  Ain=0; Bin=1; sel=2'b11;$display("\n \t \t NOT Operation");
        #5 $finish;
end
endmodule
```



```
D:\Smita Co>iverilog -o lab10.vvp new10.v new10t.v

D:\Smita Co>vvp lab10.vvp

                 AND Operation
                   0Ai=0, Bi=1, out=0, S=00

                 OR Operation
                   5Ai=0, Bi=1, out=1, S=01

                 Xor Operation
                  10Ai=0, Bi=1, out=1, S=10

                 NOT Operation
                  15Ai=0, Bi=1, out=1, S=11

D:\Smita Co>
```

Q. Write a program in C/C++/java to implement Booth Multiplication algorithm.

```
#include<iostream>
using namespace std;
void add(int a[], int x[], int q);
void complement(int a[], int n) {
  int i;
  int x[8] = { NULL };
  x[0] = 1;
  for (i = 0; i < n; i++) {
    a[i] = (a[i] + 1) % 2;
  }
  add(a, x, n);
}
void add(int ac[], int x[], int q) {
  int i, c = 0;
```

```
for (i = 0; i < q; i++) {
  ac[i] = ac[i] + x[i] + c;
  if (ac[i] > 1) {
    ac[i] = ac[i] % 2;
    c = 1;
  }else
    c = 0;
  }
}
void ashr(int ac[], int qr[], int &qn, int q) {
  int temp, i;
  temp = ac[0];
  qn = qr[0];
  cout << "\t\tashr\t\t";
  for (i = 0; i < q - 1; i++) {
    ac[i] = ac[i + 1];
    qr[i] = qr[i + 1];
  }
  qr[q - 1] = temp;
}
void display(int ac[], int qr[], int qrn) {
  int i;
  for (i = qrn - 1; i >= 0; i--)
    cout << ac[i];
  cout << " ";
  for (i = qrn - 1; i >= 0; i--)
    cout << qr[i];
}
int main(int argc, char **argv) {
  int mt[10], br[10], qr[10], sc, ac[10] = { 0 };
  int brn, qrn, i, qn, temp;
  cout << "\n--Enter the multiplicand and multipier in signed 2's      complement
form if negative--";
  cout << "\n Number of multiplicand bit=";
  cin >> brn;
```

```
cout << "\nmultiplicand=";
for (i = brn - 1; i >= 0; i--)
  cin >> br[i]; //multiplicand
for (i = brn - 1; i >= 0; i--)
  mt[i] = br[i];
complement(mt, brn);
cout << "\nNo. of multiplier bit=";
cin >> qrn;
sc = qrn;
cout << "Multiplier=";
for (i = qrn - 1; i >= 0; i--)
  cin >> qr[i];
  qn = 0;
  temp = 0;
  cout << "qn\tq[n+1]\t\tBR\t\tAC\tQR\t\tsc\n";
  cout << "\t\t\tinitial\t\t";
  display(ac, qr, qrn);
  cout << "\t\t" << sc << "\n";
  while (sc != 0) {
    cout << qr[0] << "\t" << qn;
    if ((qn + qr[0]) == 1) {
      if (temp == 0) {
        add(ac, mt, qrn);
        cout << "\t\tsubtracting BR\t";
        for (i = qrn - 1; i >= 0; i--)
          cout << ac[i];
        temp = 1;
      }
    else if (temp == 1) {
      add(ac, br, qrn);
      cout << "\t\tadding BR\t";
      for (i = qrn - 1; i >= 0; i--)
        cout << ac[i];
        temp = 0;
    }
```

```
            cout << "\n\t";
            ashr(ac, qr, qn, qrn);
          }
          else if (qn - qr[0] == 0)
            ashr(ac, qr, qn, qrn);
            display(ac, qr, qrn);
            cout << "\t";
            sc--;
            cout << "\t" << sc << "\n";
    }
    cout << "Result=";
    display(ac, qr, qrn);
  }
```