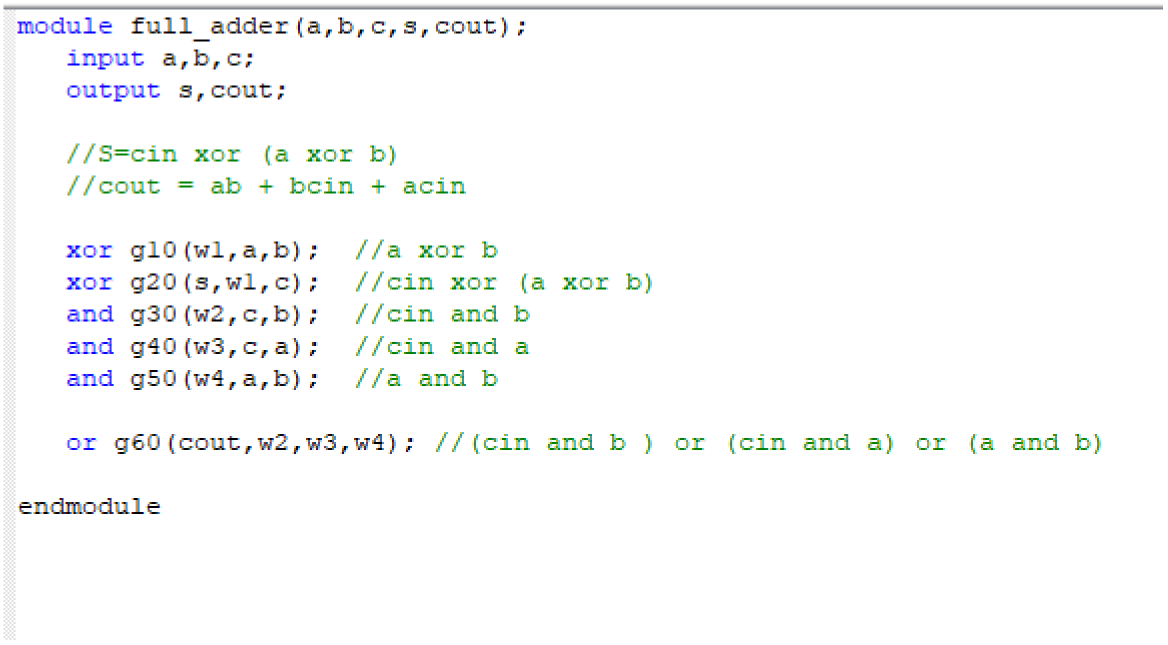
CSE 331 Computer Organization HW3 Report

Yakup Talha Yolcu

1801042609

1. 1 bit full adder

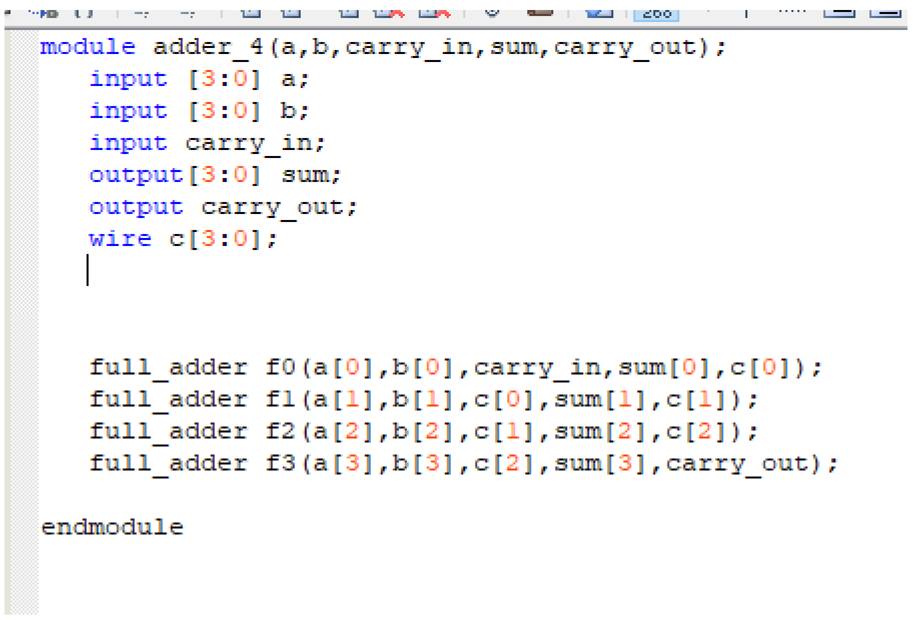
   SUM= C-IN XOR (A XOR B)
   COUT=A B + B C-IN + A C-IN



```
module full_adder(a,b,c,s,cout);
   input a,b,c;
   output s,cout;

   //S=cin xor (a xor b)
   //cout = ab + bcin + acin

   xor g10(w1,a,b);  //a xor b
   xor g20(s,w1,c);  //cin xor (a xor b)
   and g30(w2,c,b);  //cin and b
   and g40(w3,c,a);  //cin and a
   and g50(w4,a,b);  //a and b

   or g60(cout,w2,w3,w4); //(cin and b ) or (cin and a) or (a and b)

endmodule
```

2. 4 bit full adder

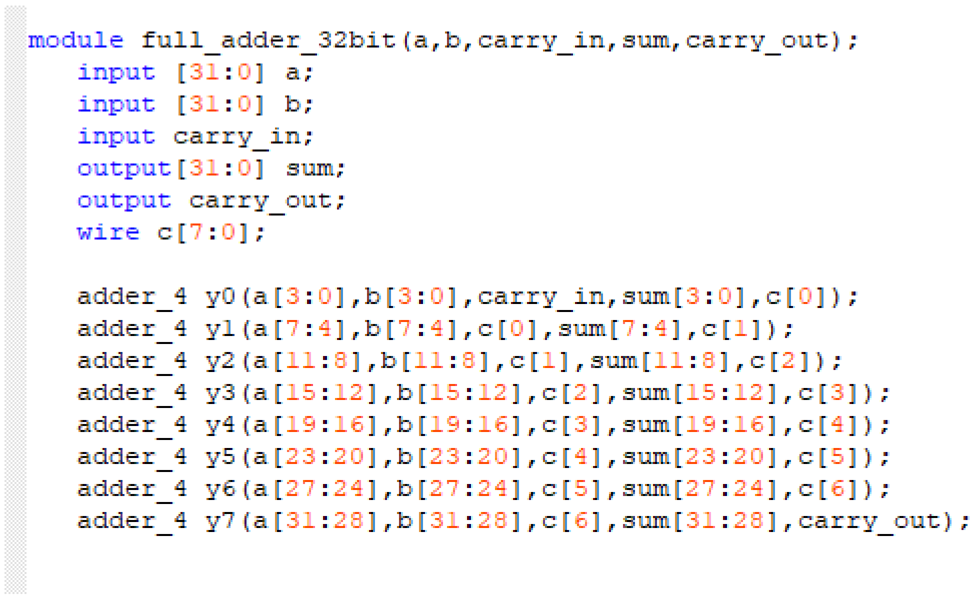
I just called 1 bit full adder 4 times and I gave carry in first, and first carry out of the 1 bit adder will be carry in of the next 1 bit adder.



```
module adder_4(a,b,carry_in,sum,carry_out);
   input [3:0] a;
   input [3:0] b;
   input carry_in;
   output[3:0] sum;
   output carry_out;
   wire c[3:0];



   full_adder f0(a[0],b[0],carry_in,sum[0],c[0]);
   full_adder f1(a[1],b[1],c[0],sum[1],c[1]);
   full_adder f2(a[2],b[2],c[1],sum[2],c[2]);
   full_adder f3(a[3],b[3],c[2],sum[3],carry_out);

endmodule
```

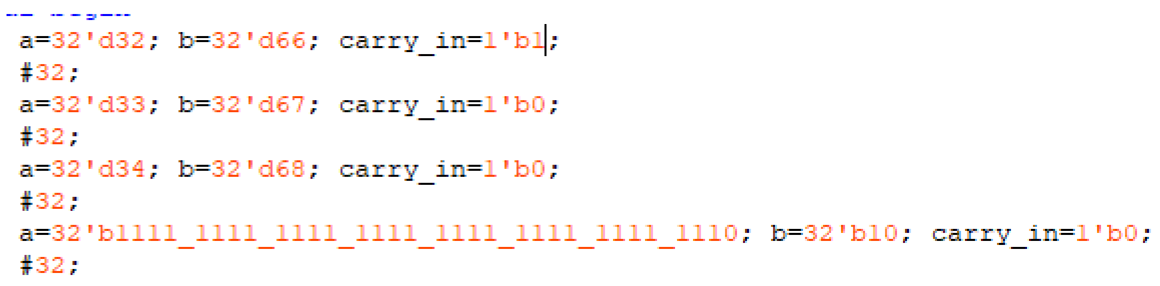
3. 32 bit full adder

I called 4 bit adder 8 times, I did the same thing here, I send carry ins as carry out of the next 4 bit adder module



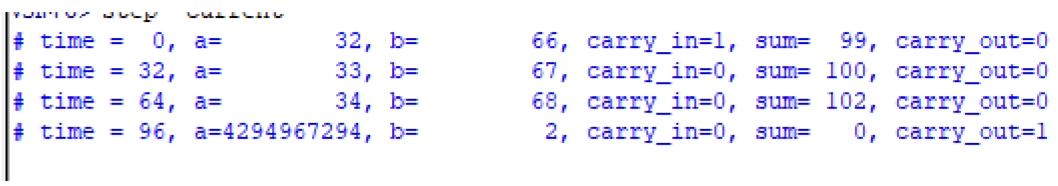
```
module full_adder_32bit(a,b,carry_in,sum,carry_out);
   input [31:0] a;
   input [31:0] b;
   input carry_in;
   output[31:0] sum;
   output carry_out;
   wire c[7:0];

   adder_4 y0(a[3:0],b[3:0],carry_in,sum[3:0],c[0]);
   adder_4 y1(a[7:4],b[7:4],c[0],sum[7:4],c[1]);
   adder_4 y2(a[11:8],b[11:8],c[1],sum[11:8],c[2]);
   adder_4 y3(a[15:12],b[15:12],c[2],sum[15:12],c[3]);
   adder_4 y4(a[19:16],b[19:16],c[3],sum[19:16],c[4]);
   adder_4 y5(a[23:20],b[23:20],c[4],sum[23:20],c[5]);
   adder_4 y6(a[27:24],b[27:24],c[5],sum[27:24],c[6]);
   adder_4 y7(a[31:28],b[31:28],c[6],sum[31:28],carry_out);
```

4. 32 bit full adder test



```
 a=32'd32; b=32'd66; carry_in=1'b1;
 #32;
 a=32'd33; b=32'd67; carry_in=1'b0;
 #32;
 a=32'd34; b=32'd68; carry_in=1'b0;
 #32;
 a=32'b1111_1111_1111_1111_1111_1111_1111_1110; b=32'b10; carry_in=1'b0;
 #32;
```

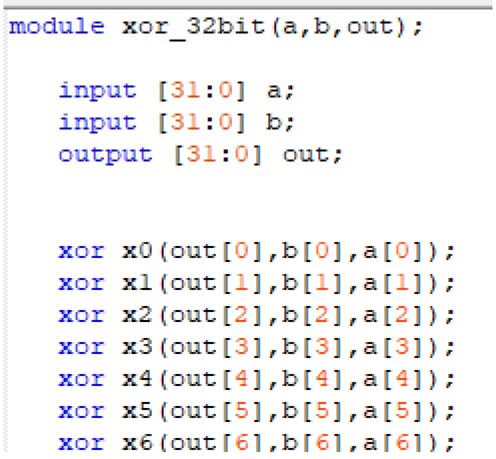
First test, 32 + 66 + 1 = 99 => a=32,b=66,carry in = 1
Second test, 33 + 67 = 100 there is no carry in
Third test 34 + 68 = 102, there is no carry in
Fourth test a=2^32-2 b= 2, I sum them up to get carry out and I get carry out.



```
# time =  0, a=         32, b=          66, carry_in=1, sum=  99, carry_out=0
# time = 32, a=         33, b=          67, carry_in=0, sum= 100, carry_out=0
# time = 64, a=         34, b=          68, carry_in=0, sum= 102, carry_out=0
# time = 96, a=4294967294, b=           2, carry_in=0, sum=   0, carry_out=1
```

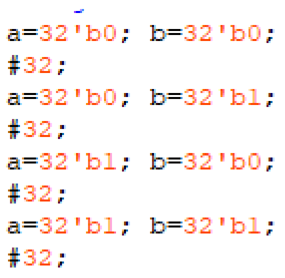
5. 32 bit xor

I just send all bits to the 1 bit xor gate



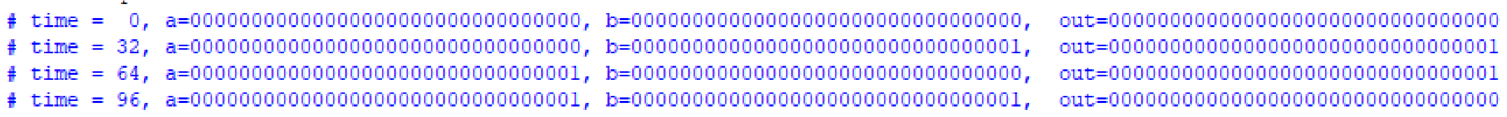
```
module xor_32bit(a,b,out);

   input [31:0] a;
   input [31:0] b;
   output [31:0] out;


   xor x0(out[0],b[0],a[0]);
   xor x1(out[1],b[1],a[1]);
   xor x2(out[2],b[2],a[2]);
   xor x3(out[3],b[3],a[3]);
   xor x4(out[4],b[4],a[4]);
   xor x5(out[5],b[5],a[5]);
   xor x6(out[6],b[6],a[6]);
```

6. 32 bit xor test



```
a=32'b0; b=32'b0;
#32;
a=32'b0; b=32'b1;
#32;
a=32'b1; b=32'b0;
#32;
a=32'b1; b=32'b1;
#32;
```

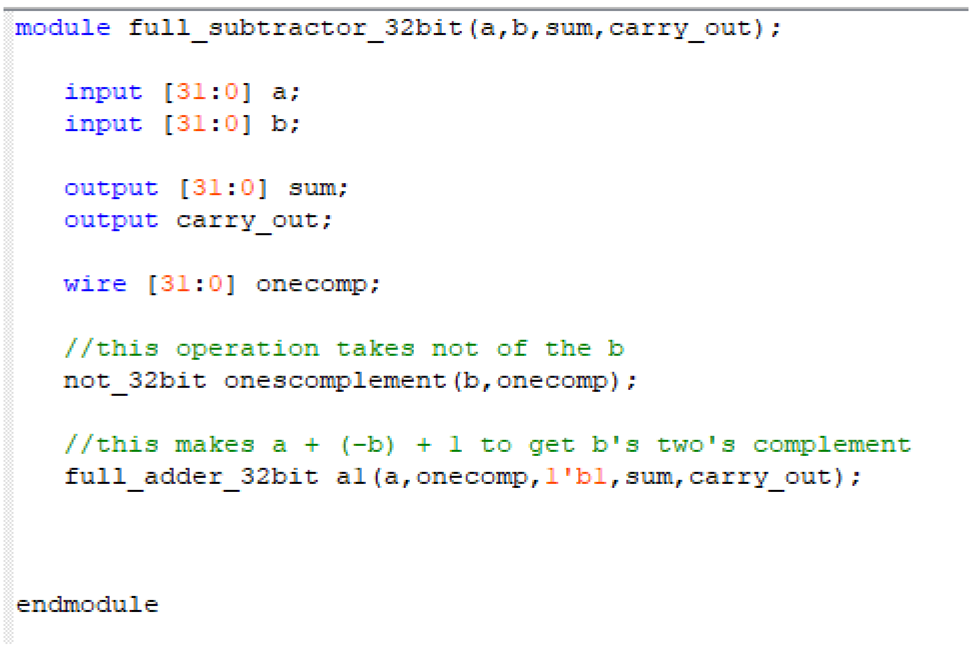
I just xor the 0 and 1 as 1 bits



```
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=00000000000000000000000000000001
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=00000000000000000000000000000000
```

7. 32 bit full subtractor

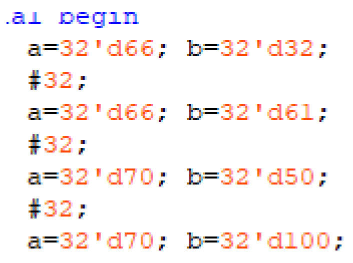
S=a+b'+1



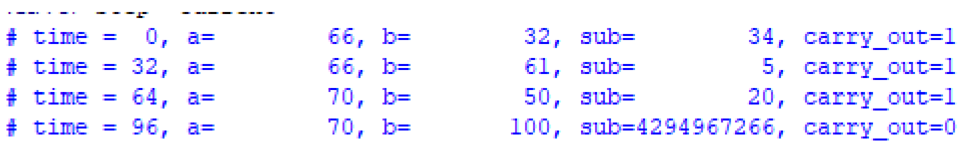
```
module full_subtractor_32bit(a,b,sum,carry_out);

   input [31:0] a;
   input [31:0] b;

   output [31:0] sum;
   output carry_out;

   wire [31:0] onecomp;

   //this operation takes not of the b
   not_32bit onescomplement(b,onecomp);

   //this makes a + (-b) + 1 to get b's two's complement
   full_adder_32bit a1(a,onecomp,1'b1,sum,carry_out);


endmodule
```

8. 32 bit full subtractor test

66-32=34 66-61=5 70-50=20 70-100=-30 as decimal

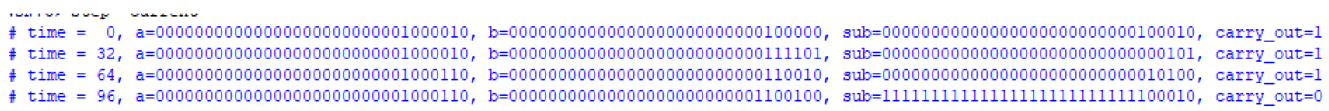


```
.al begin
  a=32'd66; b=32'd32;
  #32;
  a=32'd66; b=32'd61;
  #32;
  a=32'd70; b=32'd50;
  #32;
  a=32'd70; b=32'd100;
```



```
# time =  0, a=        66, b=        32, sub=        34, carry_out=1
# time = 32, a=        66, b=        61, sub=         5, carry_out=1
# time = 64, a=        70, b=        50, sub=        20, carry_out=1
# time = 96, a=        70, b=       100, sub=4294967266, carry_out=0
```

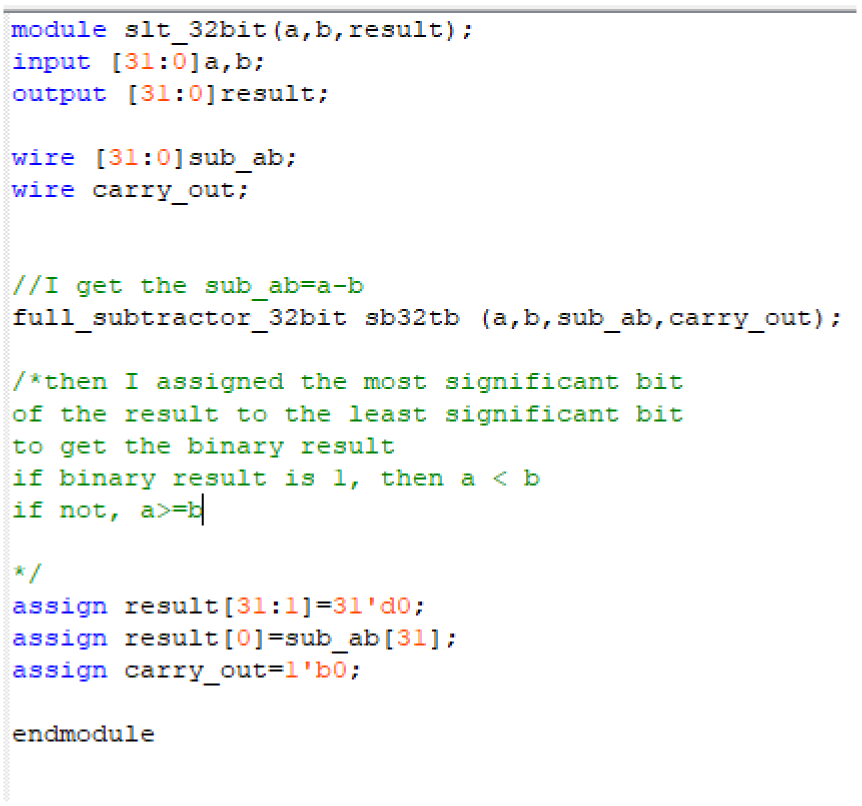
Same result as binary format, 11111111111111111111111111100010 is binary of -30 in signed 2’s complement (70-100 case)



```
# time =  0, a=00000000000000000000000001000010, b=00000000000000000000000000100000, sub=00000000000000000000000000100010, carry_out=1
# time = 32, a=00000000000000000000000001000010, b=00000000000000000000000000111101, sub=00000000000000000000000000000101, carry_out=1
# time = 64, a=00000000000000000000000001000110, b=00000000000000000000000000110010, sub=00000000000000000000000000010100, carry_out=1
# time = 96, a=00000000000000000000000001000110, b=00000000000000000000000001100100, sub=11111111111111111111111111100010, carry_out=0
```

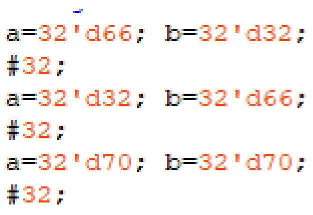
9. Multiplier => I don’t have multiplier it just makes addition
10. Because of I don’t have multiplier, I don’t have testbench of the multipler as well

11. Set less than



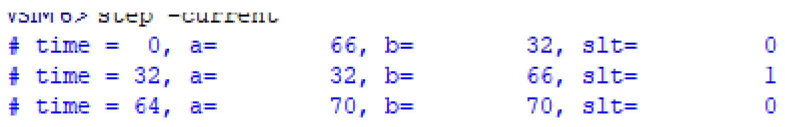
```
module slt_32bit(a,b,result);
input [31:0]a,b;
output [31:0]result;

wire [31:0]sub_ab;
wire carry_out;


//I get the sub_ab=a-b
full_subtractor_32bit sb32tb (a,b,sub_ab,carry_out);

/*then I assigned the most significant bit
of the result to the least significant bit
to get the binary result
if binary result is 1, then a < b
if not, a>=b

*/
assign result[31:1]=31'd0;
assign result[0]=sub_ab[31];
assign carry_out=1'b0;

endmodule
```

12. Set less than test

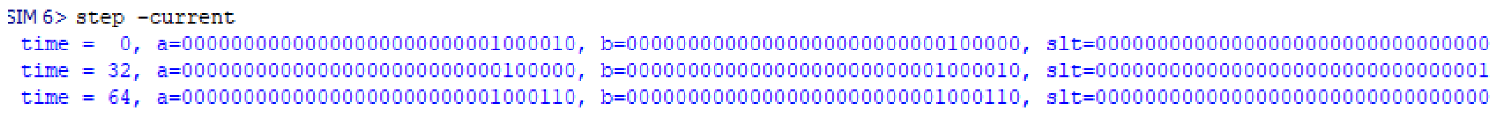


```
a=32'd66; b=32'd32;
#32;
a=32'd32; b=32'd66;
#32;
a=32'd70; b=32'd70;
#32;
```

Outputs as binary and decimal format



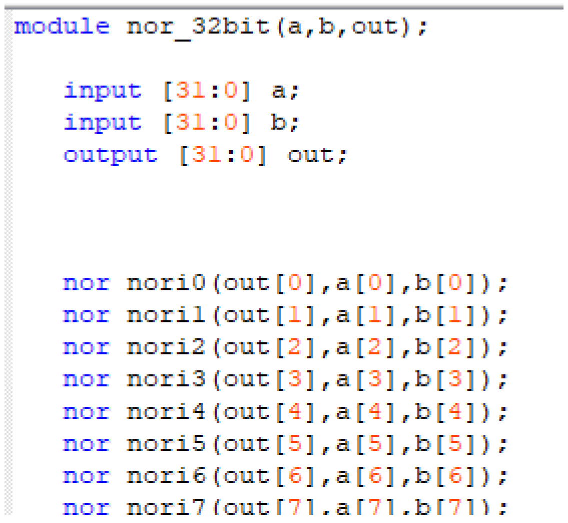
```
VSIM 6> step -current
# time =  0, a=        66, b=        32, slt=         0
# time = 32, a=        32, b=        66, slt=         1
# time = 64, a=        70, b=        70, slt=         0
```



```
SIM 6> step -current
 time =  0, a=00000000000000000000000001000010, b=00000000000000000000000000100000, slt=00000000000000000000000000000000
 time = 32, a=00000000000000000000000000100000, b=00000000000000000000000001000010, slt=00000000000000000000000000000001
 time = 64, a=00000000000000000000000001000110, b=00000000000000000000000001000110, slt=00000000000000000000000000000000
```

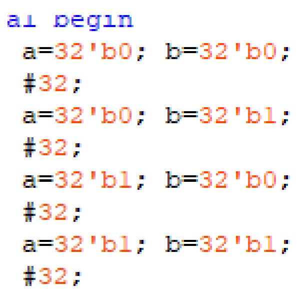
13. Nor 32 bit

I just send the bits to the 1 bit nor



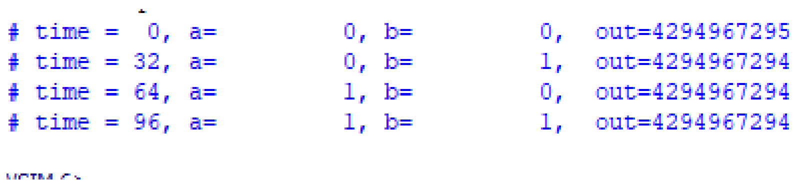
```
module nor_32bit(a,b,out);

   input [31:0] a;
   input [31:0] b;
   output [31:0] out;



   nor nori0(out[0],a[0],b[0]);
   nor nori1(out[1],a[1],b[1]);
   nor nori2(out[2],a[2],b[2]);
   nor nori3(out[3],a[3],b[3]);
   nor nori4(out[4],a[4],b[4]);
   nor nori5(out[5],a[5],b[5]);
   nor nori6(out[6],a[6],b[6]);
   nor nori7(out[7],a[7],b[7]);
```

14. Nor 32 bit test

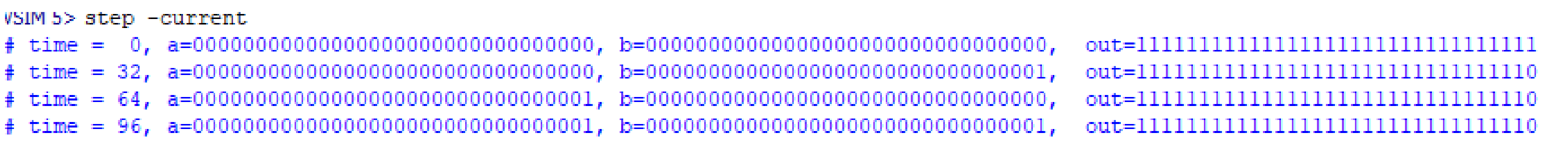


```
al begin
 a=32'b0; b=32'b0;
 #32;
 a=32'b0; b=32'b1;
 #32;
 a=32'b1; b=32'b0;
 #32;
 a=32'b1; b=32'b1;
 #32;
```

Results as binary and decimal format



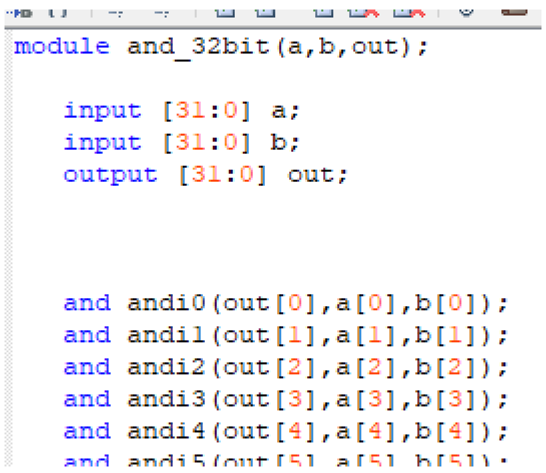
```
# time =  0, a=        0, b=        0,  out=4294967295
# time = 32, a=        0, b=        1,  out=4294967294
# time = 64, a=        1, b=        0,  out=4294967294
# time = 96, a=        1, b=        1,  out=4294967294
```



```
VSIM 5> step -current
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=11111111111111111111111111111111
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=11111111111111111111111111111110
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=11111111111111111111111111111110
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=11111111111111111111111111111110
```

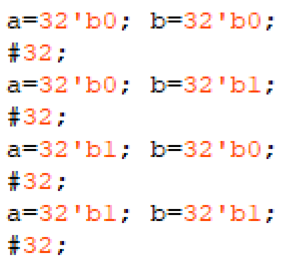
15. 32 bit and

I just send the bits to the 1 bit nor

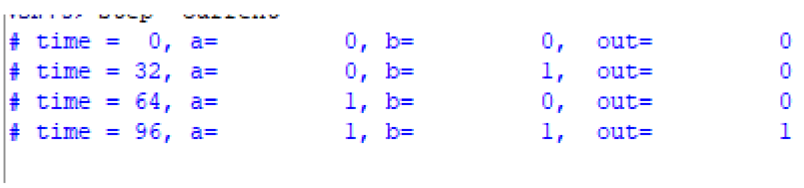


```
module and_32bit(a,b,out);

   input  [31:0]  a;
   input  [31:0]  b;
   output  [31:0]  out;



   and andi0(out[0],a[0],b[0]);
   and andi1(out[1],a[1],b[1]);
   and andi2(out[2],a[2],b[2]);
   and andi3(out[3],a[3],b[3]);
   and andi4(out[4],a[4],b[4]);
   and andi5(out[5],a[5],b[5]);
```

16. 32 bit and test

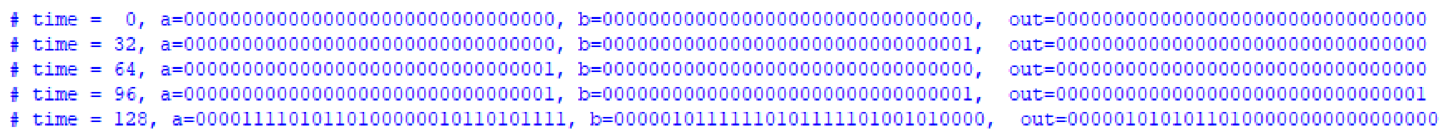


```
a=32'b0; b=32'b0;
#32;
a=32'b0; b=32'b1;
#32;
a=32'b1; b=32'b0;
#32;
a=32'b1; b=32'b1;
#32;
```



```
# time =  0, a=          0, b=          0,  out=          0
# time = 32, a=          0, b=          1,  out=          0
# time = 64, a=          1, b=          0,  out=          0
# time = 96, a=          1, b=          1,  out=          1
```

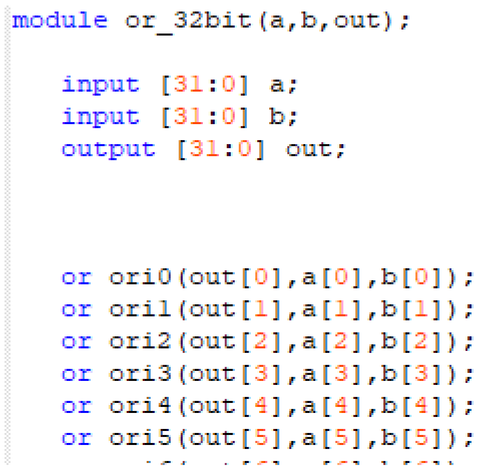
Outputs as binary format



```
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=00000000000000000000000000000000
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
# time = 128, a=00001111010110100000010110101111, b=00000101111110101111101001010000,  out=00000101010110100000000000000000
```

17. 32 bit or

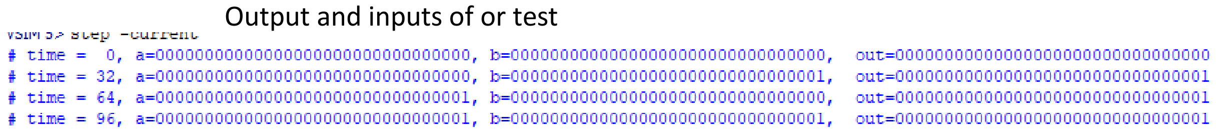
I just send the bits to the 1 bit or



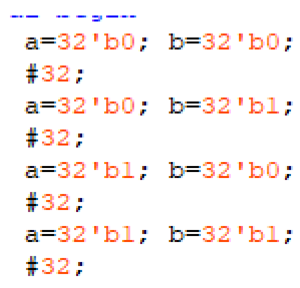
```
module or_32bit(a,b,out);

   input  [31:0]  a;
   input  [31:0]  b;
   output  [31:0]  out;



   or ori0(out[0],a[0],b[0]);
   or ori1(out[1],a[1],b[1]);
   or ori2(out[2],a[2],b[2]);
   or ori3(out[3],a[3],b[3]);
   or ori4(out[4],a[4],b[4]);
   or ori5(out[5],a[5],b[5]);
```

18. 32 bit or test

Output and inputs of or test



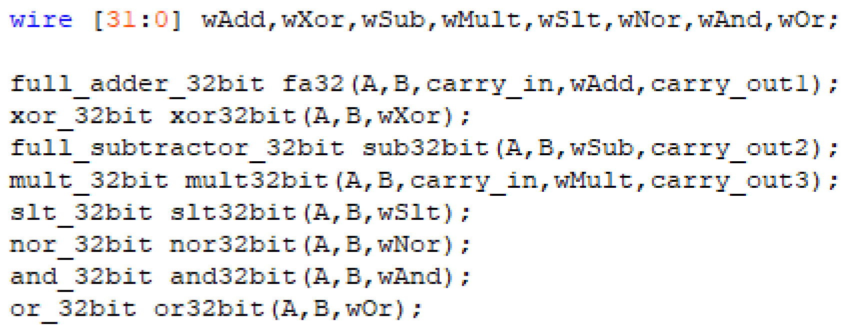
```
VSIM 5> step -current
# time =  0, a=00000000000000000000000000000000, b=00000000000000000000000000000000,  out=00000000000000000000000000000000
# time = 32, a=00000000000000000000000000000000, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
# time = 64, a=00000000000000000000000000000001, b=00000000000000000000000000000000,  out=00000000000000000000000000000001
# time = 96, a=00000000000000000000000000000001, b=00000000000000000000000000000001,  out=00000000000000000000000000000001
```



```
  a=32'b0; b=32'b0;
  #32;
  a=32'b0; b=32'b1;
  #32;
  a=32'b1; b=32'b0;
  #32;
  a=32'b1; b=32'b1;
  #32;
```

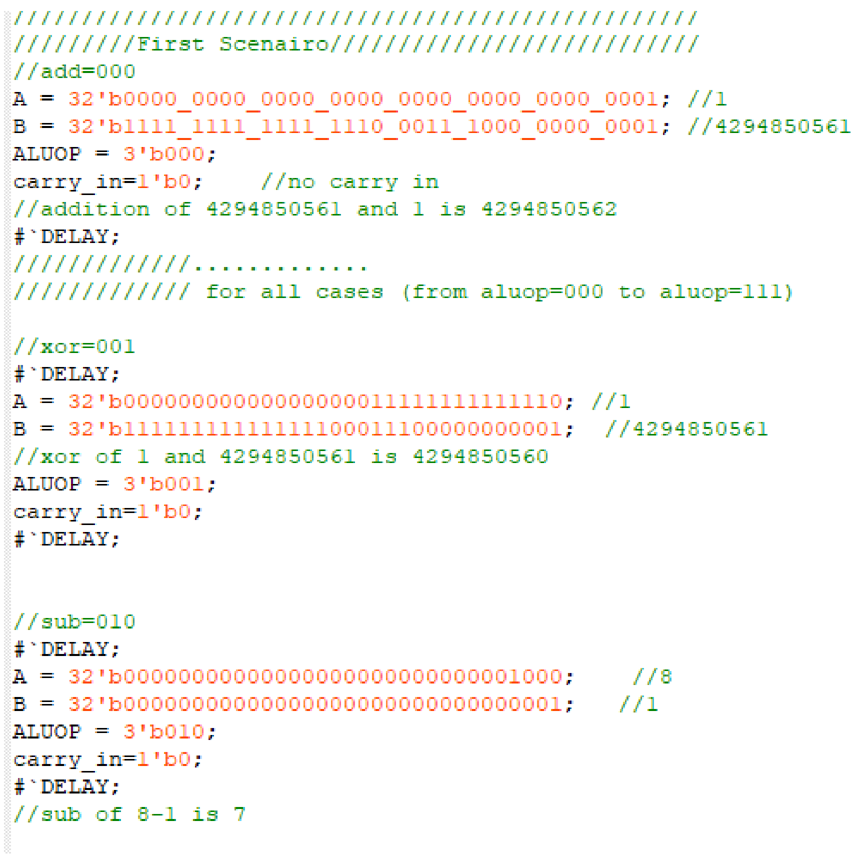
19. 32 bit ALU

First I calculated the all results, and I selected them

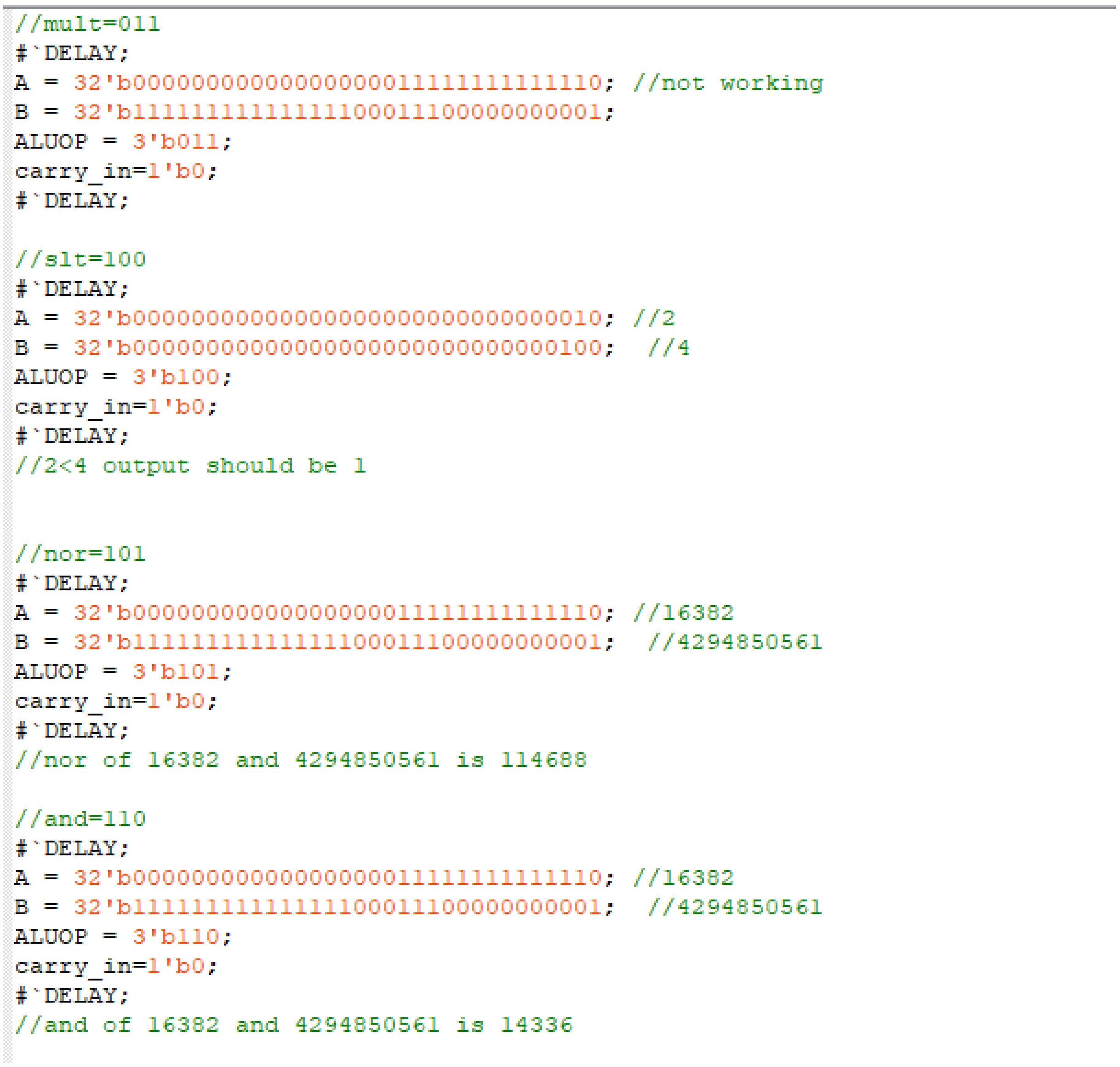


```
wire [31:0] wAdd,wXor,wSub,wMult,wSlt,wNor,wAnd,wOr;

full_adder_32bit fa32(A,B,carry_in,wAdd,carry_out1);
xor_32bit xor32bit(A,B,wXor);
full_subtractor_32bit sub32bit(A,B,wSub,carry_out2);
mult_32bit mult32bit(A,B,carry_in,wMult,carry_out3);
slt_32bit slt32bit(A,B,wSlt);
nor_32bit nor32bit(A,B,wNor);
and_32bit and32bit(A,B,wAnd);
or_32bit or32bit(A,B,wOr);
```

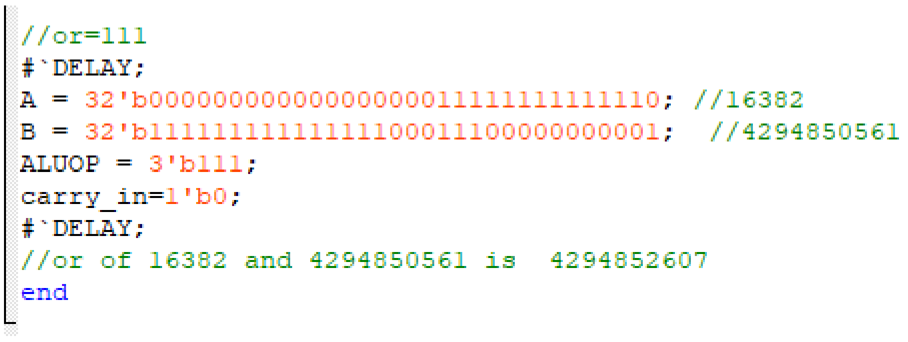
20. 32 bit alu test



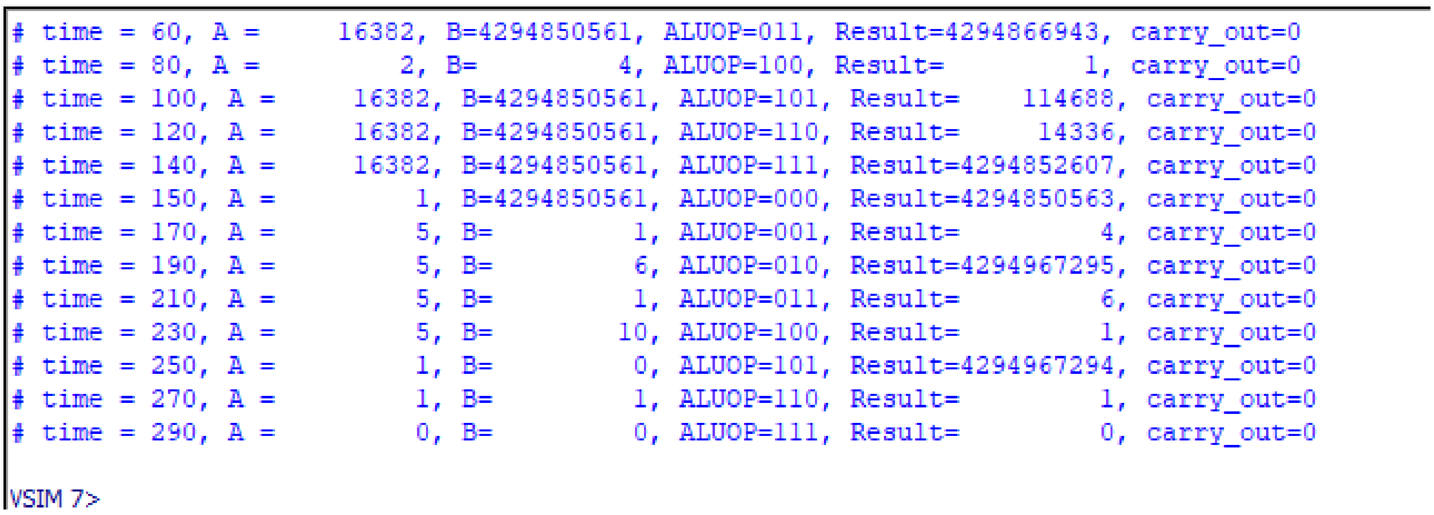
```
////////////////////////////////////////////////////
/////////First Scenairo///////////////////////////
//add=000
A = 32'b0000_0000_0000_0000_0000_0000_0000_0001; //1
B = 32'b1111_1111_1111_1110_0011_1000_0000_0001; //4294850561
ALUOP = 3'b000;
carry_in=1'b0;    //no carry in
//addition of 4294850561 and 1 is 4294850562
#`DELAY;
/////////////.............
///////////// for all cases (from aluop=000 to aluop=111)

//xor=001
#`DELAY;
A = 32'b00000000000000000001111111111110; //1
B = 32'b11111111111111100011100000000001;  //4294850561
//xor of 1 and 4294850561 is 4294850560
ALUOP = 3'b001;
carry_in=1'b0;
#`DELAY;


//sub=010
#`DELAY;
A = 32'b00000000000000000000000000001000;    //8
B = 32'b00000000000000000000000000000001;   //1
ALUOP = 3'b010;
carry_in=1'b0;
#`DELAY;
//sub of 8-1 is 7
```



```
//mult=011
#`DELAY;
A = 32'b00000000000000000011111111111110; //not working
B = 32'b11111111111111100011100000000001;
ALUOP = 3'b011;
carry_in=1'b0;
#`DELAY;

//slt=100
#`DELAY;
A = 32'b00000000000000000000000000000010; //2
B = 32'b00000000000000000000000000000100;  //4
ALUOP = 3'b100;
carry_in=1'b0;
#`DELAY;
//2<4 output should be 1


//nor=101
#`DELAY;
A = 32'b00000000000000000011111111111110; //16382
B = 32'b11111111111111100011100000000001;  //4294850561
ALUOP = 3'b101;
carry_in=1'b0;
#`DELAY;
//nor of 16382 and 4294850561 is 114688

//and=110
#`DELAY;
A = 32'b00000000000000000011111111111110; //16382
B = 32'b11111111111111100011100000000001;  //4294850561
ALUOP = 3'b110;
carry_in=1'b0;
#`DELAY;
//and of 16382 and 4294850561 is 14336
```

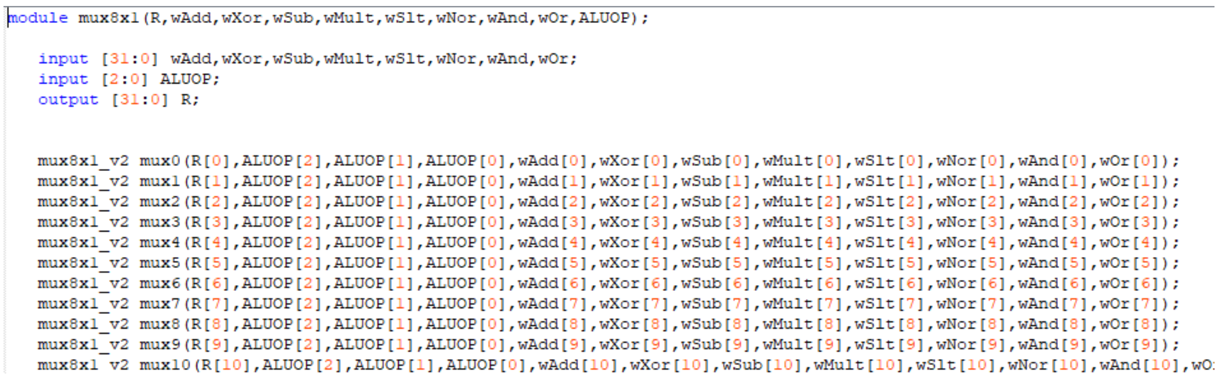


```
//or=111
#`DELAY;
A = 32'b00000000000000000011111111111110; //16382
B = 32'b11111111111111100011100000000001;  //4294850561
ALUOP = 3'b111;
carry_in=1'b0;
#`DELAY;
//or of 16382 and 4294850561 is  4294852607
end
```



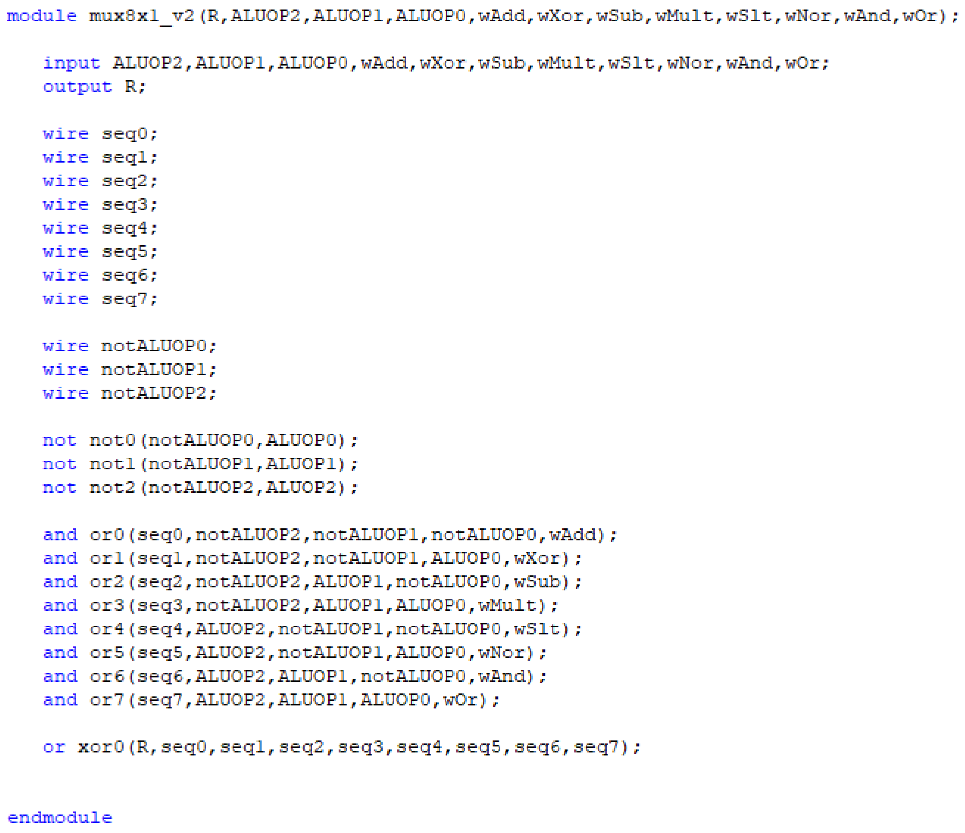
```
# time = 60, A =      16382, B=4294850561, ALUOP=011, Result=4294866943, carry_out=0
# time = 80, A =          2, B=          4, ALUOP=100, Result=         1, carry_out=0
# time = 100, A =      16382, B=4294850561, ALUOP=101, Result=    114688, carry_out=0
# time = 120, A =      16382, B=4294850561, ALUOP=110, Result=     14336, carry_out=0
# time = 140, A =      16382, B=4294850561, ALUOP=111, Result=4294852607, carry_out=0
# time = 150, A =          1, B=4294850561, ALUOP=000, Result=4294850563, carry_out=0
# time = 170, A =          5, B=          1, ALUOP=001, Result=         4, carry_out=0
# time = 190, A =          5, B=          6, ALUOP=010, Result=4294967295, carry_out=0
# time = 210, A =          5, B=          1, ALUOP=011, Result=         6, carry_out=0
# time = 230, A =          5, B=         10, ALUOP=100, Result=         1, carry_out=0
# time = 250, A =          1, B=          0, ALUOP=101, Result=4294967294, carry_out=0
# time = 270, A =          1, B=          1, ALUOP=110, Result=         1, carry_out=0
# time = 290, A =          0, B=          0, ALUOP=111, Result=         0, carry_out=0

VSIM 7>
```

21. 32 bit mux



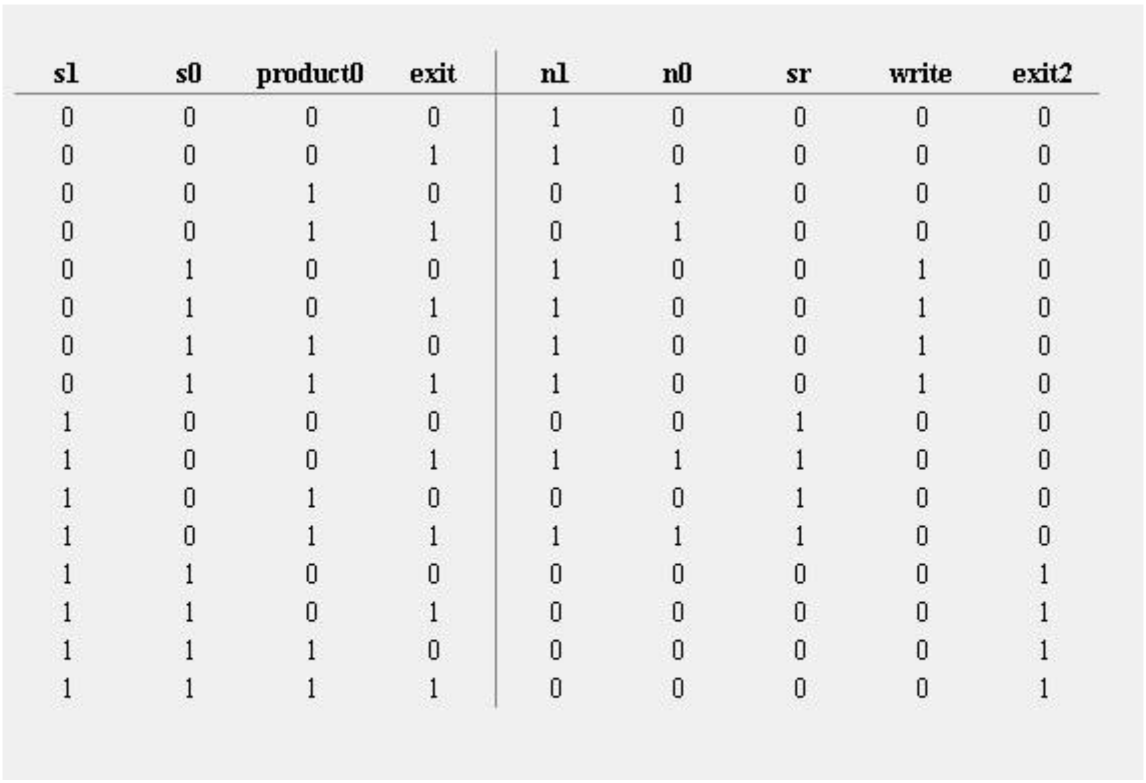
```
module mux8x1(R,wAdd,wXor,wSub,wMult,wSlt,wNor,wAnd,wOr,ALUOP);

   input [31:0] wAdd,wXor,wSub,wMult,wSlt,wNor,wAnd,wOr;
   input [2:0] ALUOP;
   output [31:0] R;


   mux8x1_v2 mux0(R[0],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[0],wXor[0],wSub[0],wMult[0],wSlt[0],wNor[0],wAnd[0],wOr[0]);
   mux8x1_v2 mux1(R[1],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[1],wXor[1],wSub[1],wMult[1],wSlt[1],wNor[1],wAnd[1],wOr[1]);
   mux8x1_v2 mux2(R[2],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[2],wXor[2],wSub[2],wMult[2],wSlt[2],wNor[2],wAnd[2],wOr[2]);
   mux8x1_v2 mux3(R[3],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[3],wXor[3],wSub[3],wMult[3],wSlt[3],wNor[3],wAnd[3],wOr[3]);
   mux8x1_v2 mux4(R[4],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[4],wXor[4],wSub[4],wMult[4],wSlt[4],wNor[4],wAnd[4],wOr[4]);
   mux8x1_v2 mux5(R[5],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[5],wXor[5],wSub[5],wMult[5],wSlt[5],wNor[5],wAnd[5],wOr[5]);
   mux8x1_v2 mux6(R[6],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[6],wXor[6],wSub[6],wMult[6],wSlt[6],wNor[6],wAnd[6],wOr[6]);
   mux8x1_v2 mux7(R[7],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[7],wXor[7],wSub[7],wMult[7],wSlt[7],wNor[7],wAnd[7],wOr[7]);
   mux8x1_v2 mux8(R[8],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[8],wXor[8],wSub[8],wMult[8],wSlt[8],wNor[8],wAnd[8],wOr[8]);
   mux8x1_v2 mux9(R[9],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[9],wXor[9],wSub[9],wMult[9],wSlt[9],wNor[9],wAnd[9],wOr[9]);
   mux8x1 v2 mux10(R[10],ALUOP[2],ALUOP[1],ALUOP[0],wAdd[10],wXor[10],wSub[10],wMult[10],wSlt[10],wNor[10],wAnd[10],wO
```

22. 8x1 mux

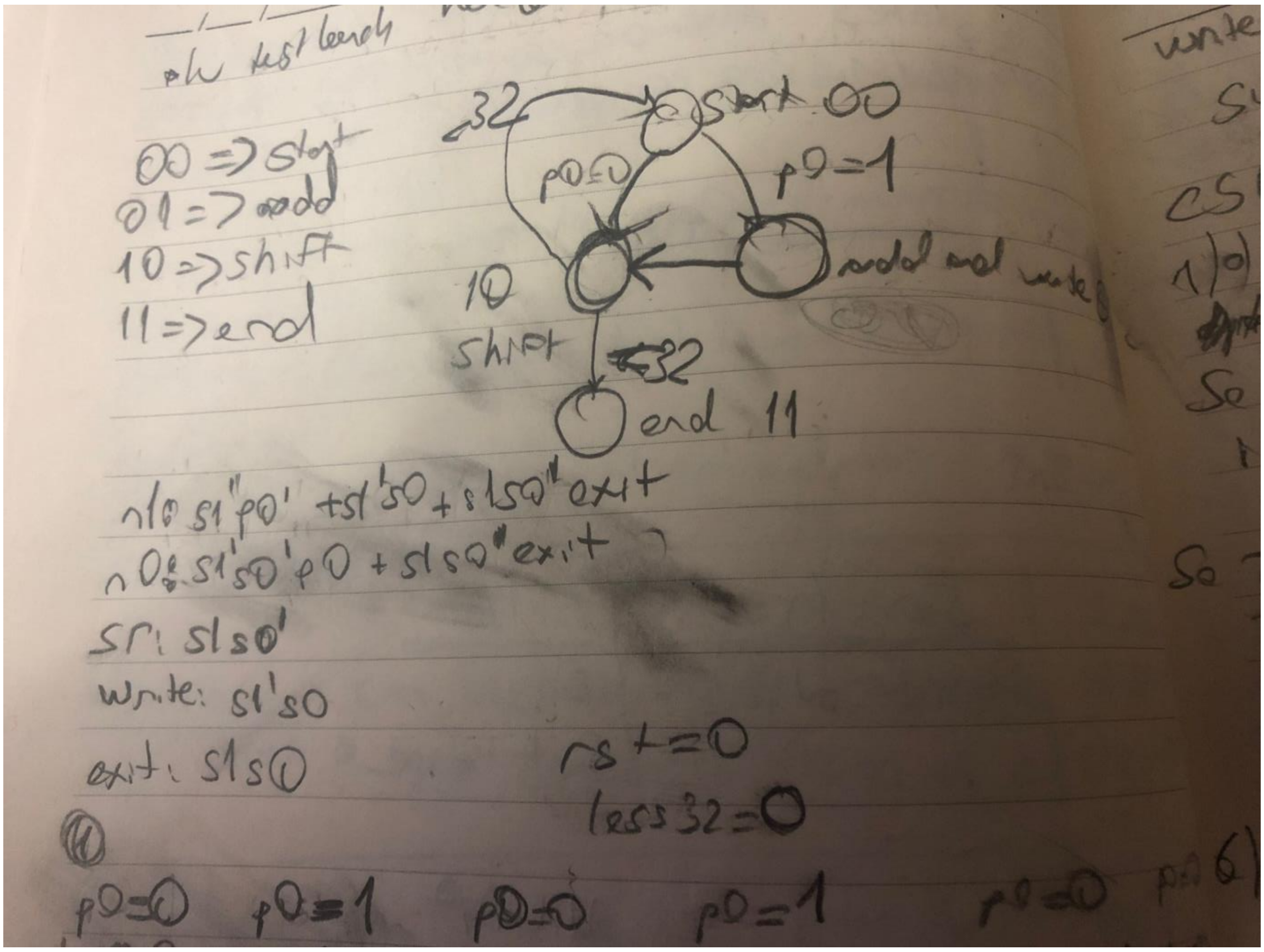


```
module mux8x1_v2(R,ALUOP2,ALUOP1,ALUOP0,wAdd,wXor,wSub,wMult,wSlt,wNor,wAnd,wOr);

   input ALUOP2,ALUOP1,ALUOP0,wAdd,wXor,wSub,wMult,wSlt,wNor,wAnd,wOr;
   output R;

   wire seq0;
   wire seq1;
   wire seq2;
   wire seq3;
   wire seq4;
   wire seq5;
   wire seq6;
   wire seq7;

   wire notALUOP0;
   wire notALUOP1;
   wire notALUOP2;

   not not0(notALUOP0,ALUOP0);
   not not1(notALUOP1,ALUOP1);
   not not2(notALUOP2,ALUOP2);

   and or0(seq0,notALUOP2,notALUOP1,notALUOP0,wAdd);
   and or1(seq1,notALUOP2,notALUOP1,ALUOP0,wXor);
   and or2(seq2,notALUOP2,ALUOP1,notALUOP0,wSub);
   and or3(seq3,notALUOP2,ALUOP1,ALUOP0,wMult);
   and or4(seq4,ALUOP2,notALUOP1,notALUOP0,wSlt);
   and or5(seq5,ALUOP2,notALUOP1,ALUOP0,wNor);
   and or6(seq6,ALUOP2,ALUOP1,notALUOP0,wAnd);
   and or7(seq7,ALUOP2,ALUOP1,ALUOP0,wOr);

   or xor0(R,seq0,seq1,seq2,seq3,seq4,seq5,seq6,seq7);


endmodule
```

23. Expression table for part1



s1	s0	product0	exit	n1	n0	sr	write	exit2
0	0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	1	0	0	1	0	0	0
0	0	1	1	0	1	0	0	0
0	1	0	0	1	0	0	1	0
0	1	0	1	1	0	0	1	0
0	1	1	0	1	0	0	1	0
0	1	1	1	1	0	0	1	0
1	0	0	0	0	0	1	0	0
1	0	0	1	1	1	1	0	0
1	0	1	0	0	0	1	0	0
1	0	1	1	1	1	1	0	0
1	1	0	0	0	0	0	0	1
1	1	0	1	0	0	0	0	1
1	1	1	0	0	0	0	0	1
1	1	1	1	0	0	0	0	1



<32
start 00
p0=0
p0=1
add and write
10
shift
≥32
end 11

ɑh test bench

00 => start
01 => add
10 => shift
11 => end

n1: s1'p0' + s1's0 + s1s0'exit
n0: s1's0'p0 + s1s0'exit
sr: s1s0'
write: s1's0
exit: s1s0

rst=0
less32=0

p0=0 p0=1 p0=0 p0=1 p0=0