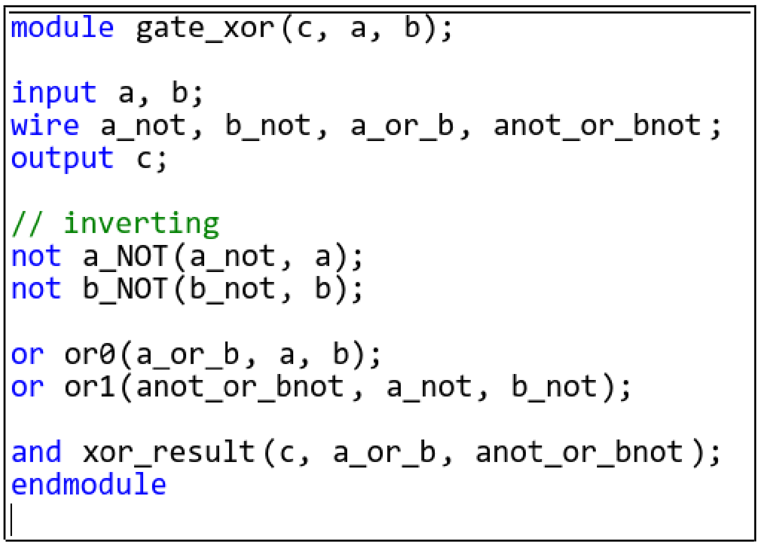
CSE331 Assignment Report

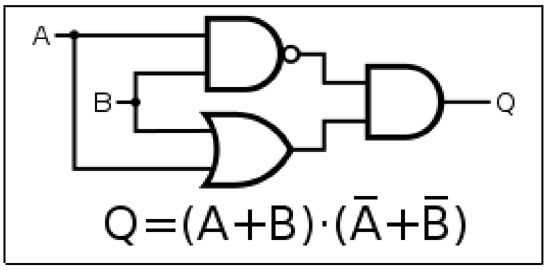
Ahmed Semih Özmekik

XOR

Simple circuit shown here implemented.

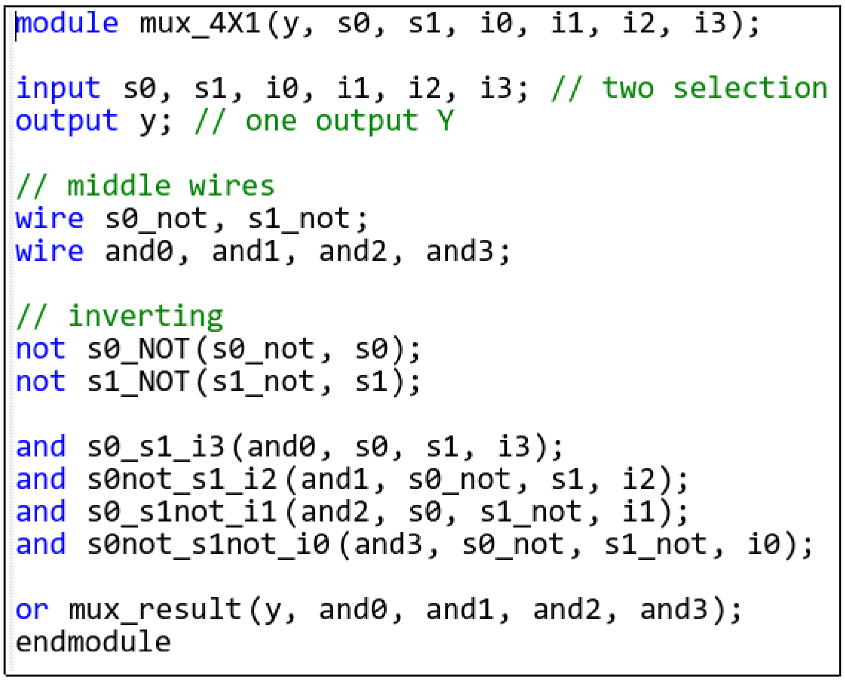


```
module gate_xor(c, a, b);

input a, b;
wire a_not, b_not, a_or_b, anot_or_bnot;
output c;

// inverting
not a_NOT(a_not, a);
not b_NOT(b_not, b);

or or0(a_or_b, a, b);
or or1(anot_or_bnot, a_not, b_not);

and xor_result(c, a_or_b, anot_or_bnot);
endmodule
```

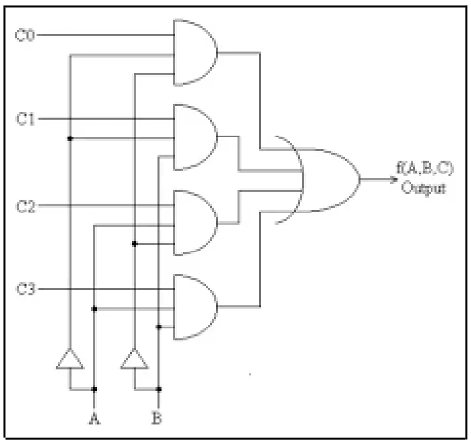


4x1 MUX

Simple circuit shown here implemented.[1]

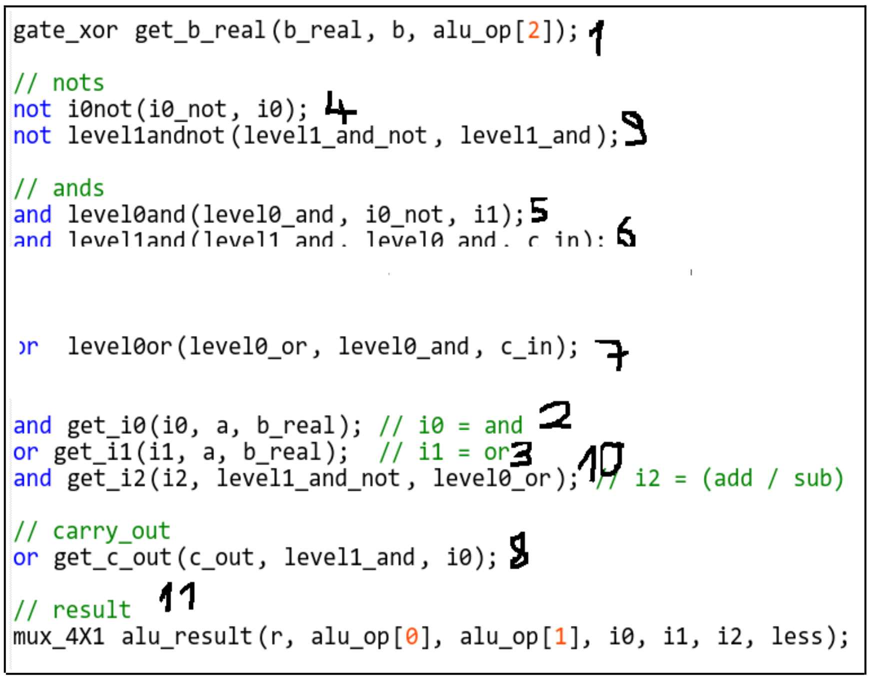


```
module mux_4X1(y, s0, s1, i0, i1, i2, i3);

input s0, s1, i0, i1, i2, i3; // two selection
output y; // one output Y

// middle wires
wire s0_not, s1_not;
wire and0, and1, and2, and3;

// inverting
not s0_NOT(s0_not, s0);
not s1_NOT(s1_not, s1);

and s0_s1_i3(and0, s0, s1, i3);
and s0not_s1_i2(and1, s0_not, s1, i2);
and s0_s1not_i1(and2, s0, s1_not, i1);
and s0not_s1not_i0(and3, s0_not, s1_not, i0);

or mux_result(y, and0, and1, and2, and3);
endmodule
```

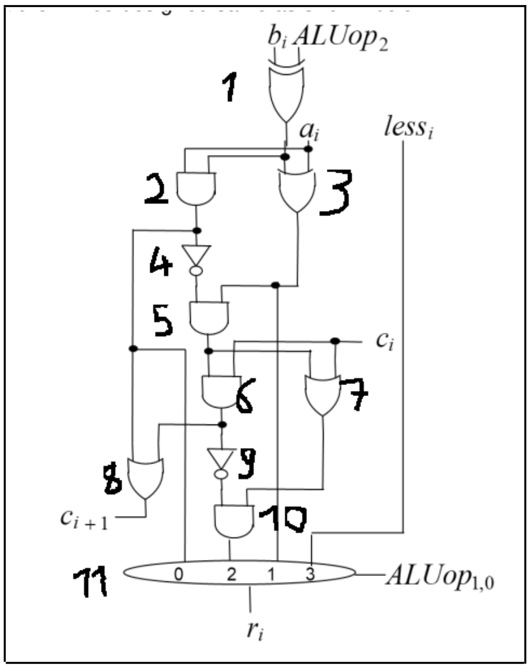


1 C's are being S's in our case, i.e. S0<-C0, S1<-C1, so on.

1-bit ALU

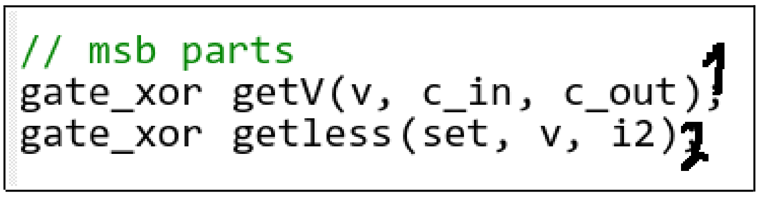


```
gate_xor get_b_real(b_real, b, alu_op[2]); 1

// nots
not i0not(i0_not, i0); 4
not level1andnot(level1_and_not, level1_and);9

// ands
and level0and(level0_and, i0_not, i1);5
and level1and(level1_and, level0_and, c_in); 6


r  level0or(level0_or, level0_and, c_in); 7


and get_i0(i0, a, b_real); // i0 = and 2
or get_i1(i1, a, b_real);  // i1 = or3
and get_i2(i2, level1_and_not, level0_or); 10// i2 = (add / sub)

// carry_out
or get_c_out(c_out, level1_and, i0); 8

// result 11
mux_4X1 alu_result(r, alu_op[0], alu_op[1], i0, i1, i2, less);
```

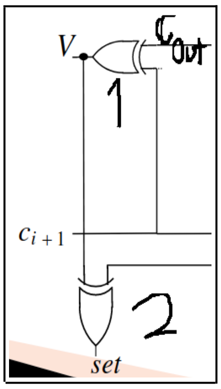


1-bit ALU for MSB

Only the different extra parts shown.

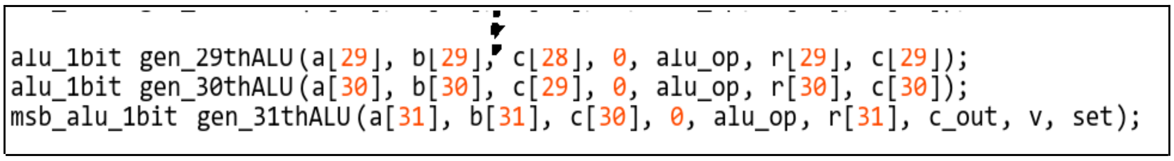


```
// msb parts
gate_xor getV(v, c_in, c_out);1
gate_xor getless(set, v, i2)2
```



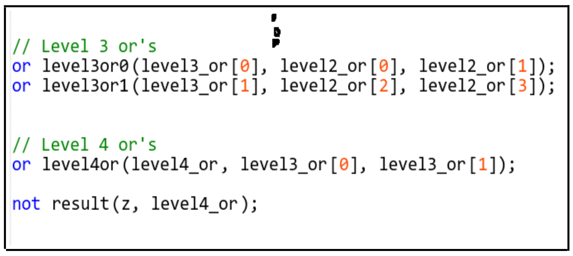
32-bit ALU

As shown with 3 dots, it comes from LSB and contiunes 'til MSB.



```
alu_1bit gen_29thALU(a[29], b[29], c[28], 0, alu_op, r[29], c[29]);
alu_1bit gen_30thALU(a[30], b[30], c[29], 0, alu_op, r[30], c[30]);
msb_alu_1bit gen_31thALU(a[31], b[31], c[30], 0, alu_op, r[31], c_out, v, set);
```

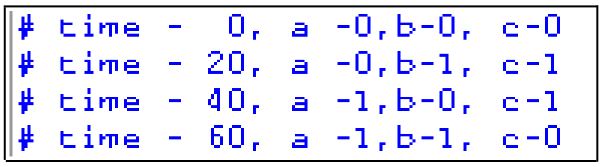
And comes from first level of OR's and continues 'til 4th level.



```
// Level 3 or's
or level3or0(level3_or[0], level2_or[0], level2_or[1]);
or level3or1(level3_or[1], level2_or[2], level2_or[3]);


// Level 4 or's
or level4or(level4_or, level3_or[0], level3_or[1]);

not result(z, level4_or);
```

XOR

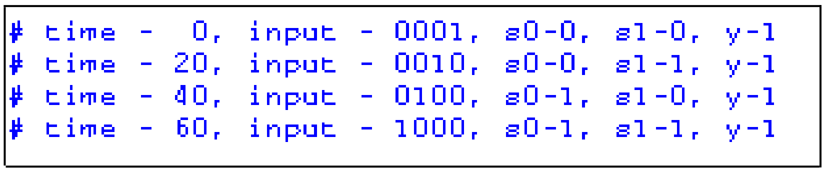
Every combinations tested.



```
# time -  0, a -0,b-0, c-0
# time - 20, a -0,b-1, c-1
# time - 40, a -1,b-0, c-1
# time - 60, a -1,b-1, c-0
```

4x1 MUX

As shown below, in each one different bit positions set and *mux* chooses the correct one.

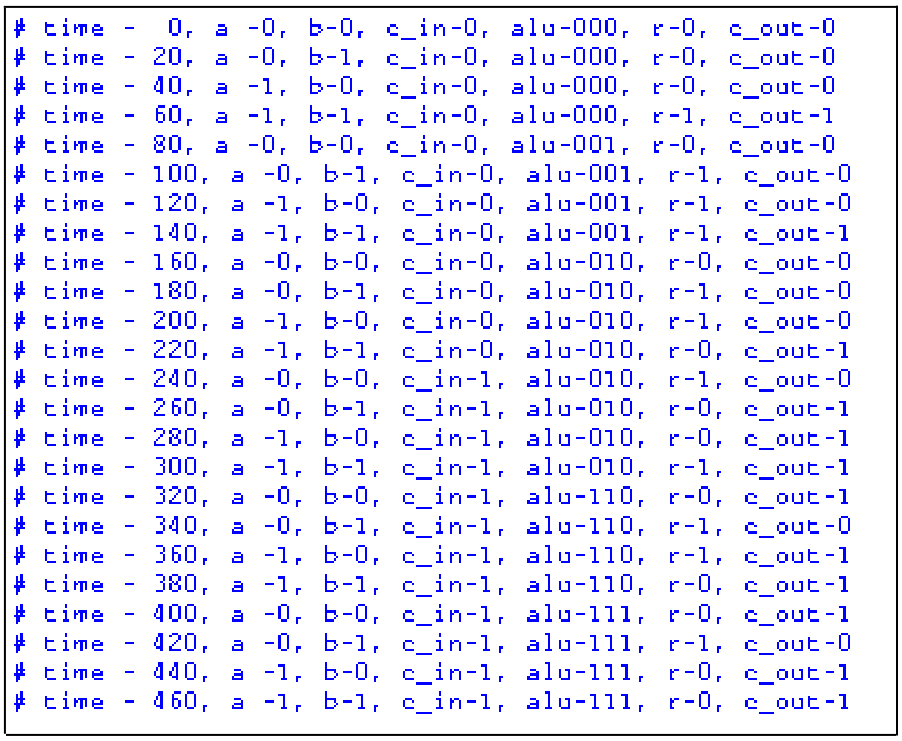


```
# time -  0, input - 0001, s0-0, s1-0, y-1
# time - 20, input - 0010, s0-0, s1-1, y-1
# time - 40, input - 0100, s0-1, s1-0, y-1
# time - 60, input - 1000, s0-1, s1-1, y-1
```

1-bit ALU

Every –valid– combinations tested.

AND
OR
ADD
SUBSTRACT
LESS

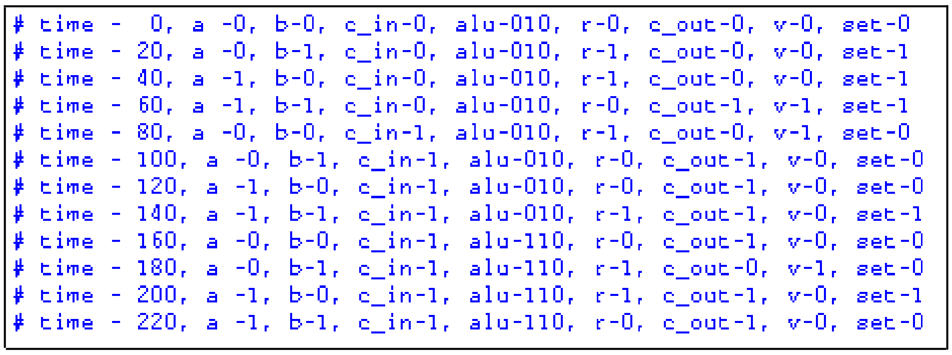


```
# time -   0, a -0, b-0, c_in-0, alu-000, r-0, c_out-0
# time -  20, a -0, b-1, c_in-0, alu-000, r-0, c_out-0
# time -  40, a -1, b-0, c_in-0, alu-000, r-0, c_out-0
# time -  60, a -1, b-1, c_in-0, alu-000, r-1, c_out-1
# time -  80, a -0, b-0, c_in-0, alu-001, r-0, c_out-0
# time - 100, a -0, b-1, c_in-0, alu-001, r-1, c_out-0
# time - 120, a -1, b-0, c_in-0, alu-001, r-1, c_out-0
# time - 140, a -1, b-1, c_in-0, alu-001, r-1, c_out-1
# time - 160, a -0, b-0, c_in-0, alu-010, r-0, c_out-0
# time - 180, a -0, b-1, c_in-0, alu-010, r-1, c_out-0
# time - 200, a -1, b-0, c_in-0, alu-010, r-1, c_out-0
# time - 220, a -1, b-1, c_in-0, alu-010, r-0, c_out-1
# time - 240, a -0, b-0, c_in-1, alu-010, r-1, c_out-0
# time - 260, a -0, b-1, c_in-1, alu-010, r-0, c_out-1
# time - 280, a -1, b-0, c_in-1, alu-010, r-0, c_out-1
# time - 300, a -1, b-1, c_in-1, alu-010, r-1, c_out-1
# time - 320, a -0, b-0, c_in-1, alu-110, r-0, c_out-1
# time - 340, a -0, b-1, c_in-1, alu-110, r-1, c_out-0
# time - 360, a -1, b-0, c_in-1, alu-110, r-1, c_out-1
# time - 380, a -1, b-1, c_in-1, alu-110, r-0, c_out-1
# time - 400, a -0, b-0, c_in-1, alu-111, r-0, c_out-1
# time - 420, a -0, b-1, c_in-1, alu-111, r-1, c_out-0
# time - 440, a -1, b-0, c_in-1, alu-111, r-0, c_out-1
# time - 460, a -1, b-1, c_in-1, alu-111, r-0, c_out-1
```

1-bit ALU for MSB

Parts which are not shown is same as with 1-bit ALU, because circuit is copied and pasted, overflow and set bits added to end of the circuit in which tested below.

ADD
SUBSTRACT

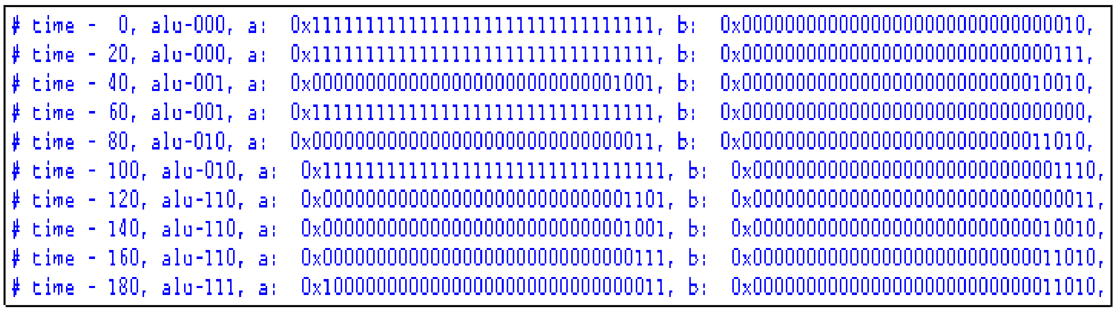


```
# time -  0, a -0, b-0, c_in-0, alu-010, r-0, c_out-0, v-0, set-0
# time - 20, a -0, b-1, c_in-0, alu-010, r-1, c_out-0, v-0, set-1
# time - 40, a -1, b-0, c_in-0, alu-010, r-1, c_out-0, v-0, set-1
# time - 60, a -1, b-1, c_in-0, alu-010, r-0, c_out-1, v-1, set-1
# time - 80, a -0, b-0, c_in-1, alu-010, r-1, c_out-0, v-1, set-0
# time - 100, a -0, b-1, c_in-1, alu-010, r-0, c_out-1, v-0, set-0
# time - 120, a -1, b-0, c_in-1, alu-010, r-0, c_out-1, v-0, set-0
# time - 140, a -1, b-1, c_in-1, alu-010, r-1, c_out-1, v-0, set-1
# time - 160, a -0, b-0, c_in-1, alu-110, r-0, c_out-1, v-0, set-0
# time - 180, a -0, b-1, c_in-1, alu-110, r-1, c_out-0, v-1, set-0
# time - 200, a -1, b-0, c_in-1, alu-110, r-1, c_out-1, v-0, set-1
# time - 220, a -1, b-1, c_in-1, alu-110, r-0, c_out-1, v-0, set-0
```

32-bit ALU

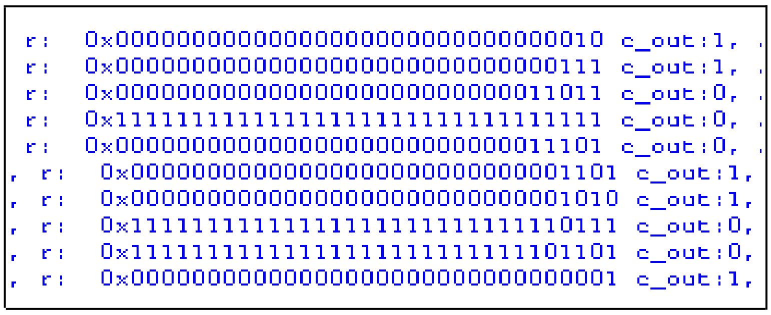
Tests are shown with seperate shots of same output. Since *ModelSim* does not print (-) sign before negative numbers (calculates and shows the number properly), I put a small line to indicate (-) sign in the last picture.

AND
OR
ADD
SUBSTRACT
LESS



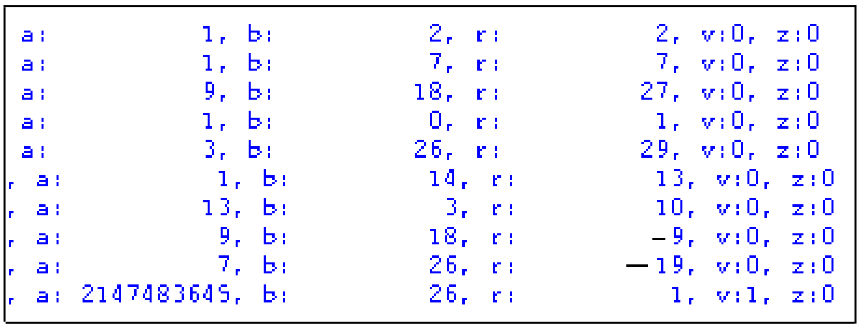
```
# time -  0, alu-000, a:  0x11111111111111111111111111111111, b:  0x00000000000000000000000000000010,
# time - 20, alu-000, a:  0x11111111111111111111111111111111, b:  0x00000000000000000000000000000111,
# time - 40, alu-001, a:  0x00000000000000000000000000001001, b:  0x00000000000000000000000000010010,
# time - 60, alu-001, a:  0x11111111111111111111111111111111, b:  0x00000000000000000000000000000000,
# time - 80, alu-010, a:  0x00000000000000000000000000000011, b:  0x00000000000000000000000000011010,
# time - 100, alu-010, a:  0x11111111111111111111111111111111, b:  0x00000000000000000000000000001110,
# time - 120, alu-110, a:  0x00000000000000000000000000001101, b:  0x00000000000000000000000000000011,
# time - 140, alu-110, a:  0x00000000000000000000000000001001, b:  0x00000000000000000000000000010010,
# time - 160, alu-110, a:  0x00000000000000000000000000000111, b:  0x00000000000000000000000000011010,
# time - 180, alu-111, a:  0x10000000000000000000000000000011, b:  0x00000000000000000000000000011010,
```

AND
OR
ADD
SUBSTRACT
LESS



```
 r:  0x00000000000000000000000000000010 c_out:1, .
 r:  0x00000000000000000000000000000111 c_out:1, .
 r:  0x00000000000000000000000000011011 c_out:0, .
 r:  0x11111111111111111111111111111111 c_out:0, .
 r:  0x00000000000000000000000000011101 c_out:0, .
,  r:  0x00000000000000000000000000001101 c_out:1,
,  r:  0x00000000000000000000000000001010 c_out:1,
,  r:  0x11111111111111111111111111110111 c_out:0,
,  r:  0x11111111111111111111111111101101 c_out:0,
,  r:  0x00000000000000000000000000000001 c_out:1,
```

AND
OR
ADD
SUBSTRACT
LESS



```
 a:           1, b:          2, r:           2, v:0, z:0
 a:           1, b:          7, r:           7, v:0, z:0
 a:           9, b:         18, r:          27, v:0, z:0
 a:           1, b:          0, r:           1, v:0, z:0
 a:           3, b:         26, r:          29, v:0, z:0
,  a:           1, b:         14, r:          13, v:0, z:0
,  a:          13, b:          3, r:          10, v:0, z:0
,  a:           9, b:         18, r:          -9, v:0, z:0
,  a:           7, b:         26, r:         —19, v:0, z:0
,  a: 2147483645, b:         26, r:           1, v:1, z:0
```

Number of Logic Gates

The number of logis gates I have used is shown below with my calculation. (only NOT, AND, OR gates used and counted.)

XOR	= 2*(NOT) + 2*(OR)
	= 4
4x1 MUX	= 2*(NOT) + 4*(AND) + 1*(OR)
	= 7
1-bit ALU	= 2*(NOT) + 4*(AND) + 3*(OR) + 1*(MUX)
	= 16
1-bit ALU for MSB	= (1-bit ALU) + 2*(XOR)
	= 24
32-bit ALU	= 31*(1-bit ALU) + 1*(1-bit ALU MSB) + (4 LEVEL OR)[2] + 1*(NOT)
	= 31*16 + 1*24 + 31 + 1
	= 552
Total Gates	= 552

2 (4 LEVEL OR) $= 2^4 + 2^3 + 2^2 + 2^1 + 2^0$