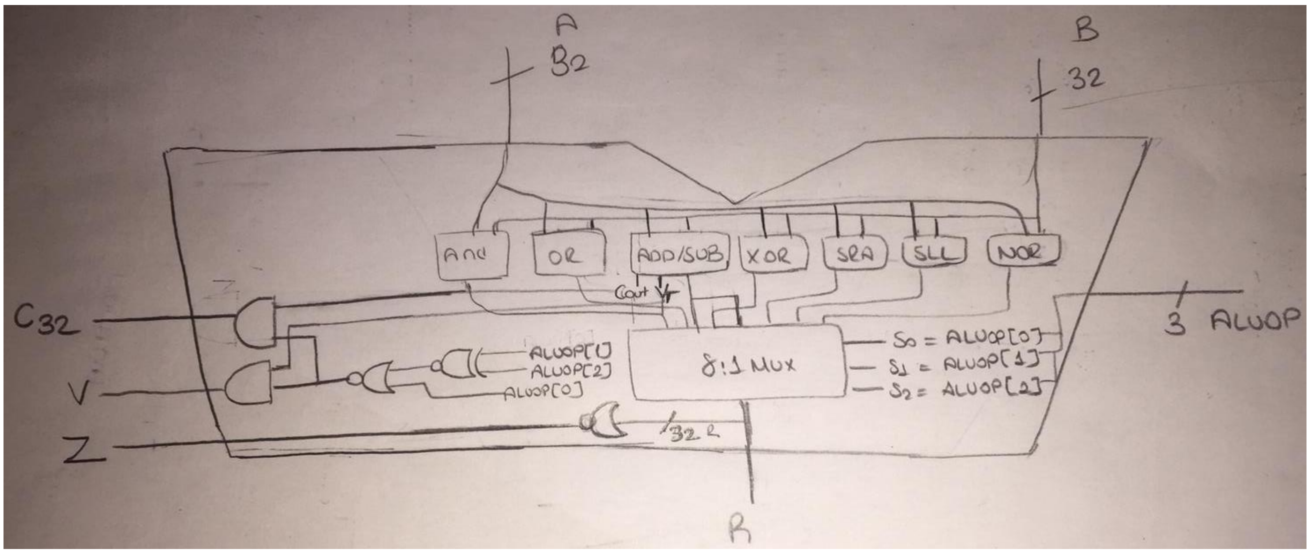
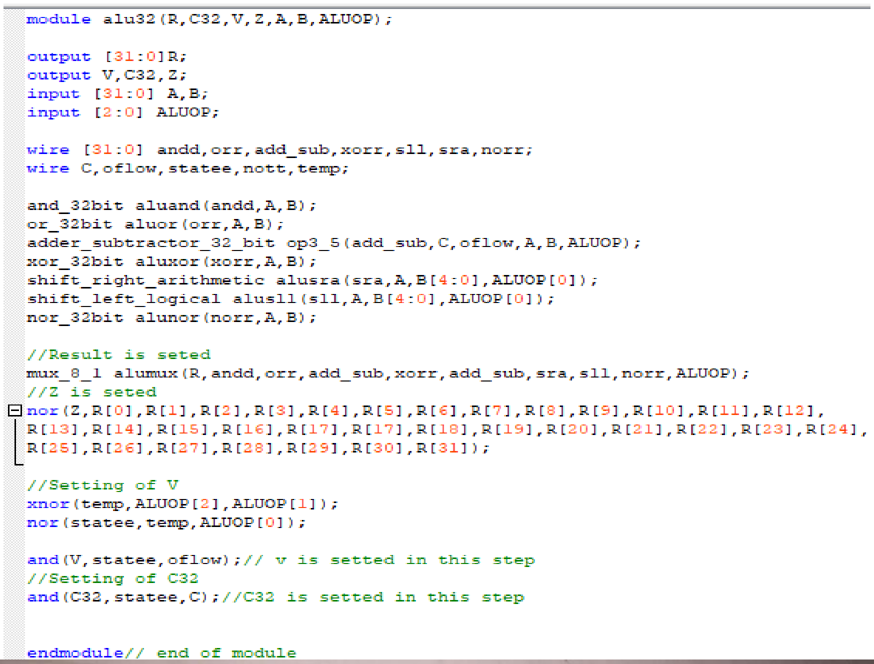
CSE 331 Computer Organization
Project 2 – ALU with Structural Verilog

SEVGİ BAYANSALDUZ – 151044076

The first thing I did was to design the module alu32. Schematic design for the module **alu32:**



The module **alu32** and its description:



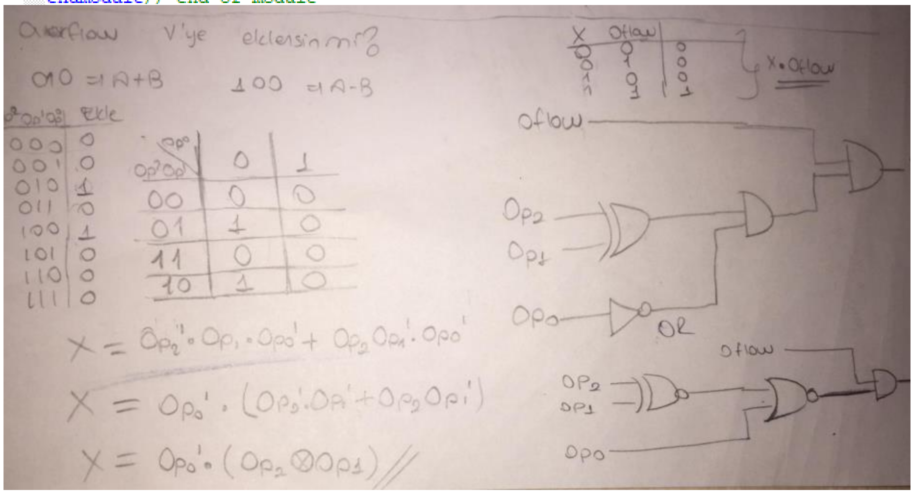
```
module alu32(R,C32,V,Z,A,B,ALUOP);

output [31:0]R;
output V,C32,Z;
input [31:0] A,B;
input [2:0] ALUOP;

wire [31:0] andd,orr,add_sub,xorr,sll,sra,norr;
wire C,oflow,statee,nott,temp;

and_32bit aluand(andd,A,B);
or_32bit aluor(orr,A,B);
adder_subtractor_32_bit op3_5(add_sub,C,oflow,A,B,ALUOP);
xor_32bit aluxor(xorr,A,B);
shift_right_arithmetic alusra(sra,A,B[4:0],ALUOP[0]);
shift_left_logical alusll(sll,A,B[4:0],ALUOP[0]);
nor_32bit alunor(norr,A,B);

//Result is seted
mux_8_1 alumux(R,andd,orr,add_sub,xorr,add_sub,sra,sll,norr,ALUOP);
//Z is seted
nor(Z,R[0],R[1],R[2],R[3],R[4],R[5],R[6],R[7],R[8],R[9],R[10],R[11],R[12],
R[13],R[14],R[15],R[16],R[17],R[17],R[18],R[19],R[20],R[21],R[22],R[23],R[24],
R[25],R[26],R[27],R[28],R[29],R[30],R[31]);

//Setting of V
xnor(temp,ALUOP[2],ALUOP[1]);
nor(statee,temp,ALUOP[0]);

and(V,statee,oflow);// v is setted in this step
//Setting of C32
and(C32,statee,C);//C32 is setted in this step


endmodule// end of module
```

The module requests three inputs: two 32 bits number, and a 3bits alu operation`s code. And returns a 32bits number, a zero bit, a overflow bit and a carry out bit.

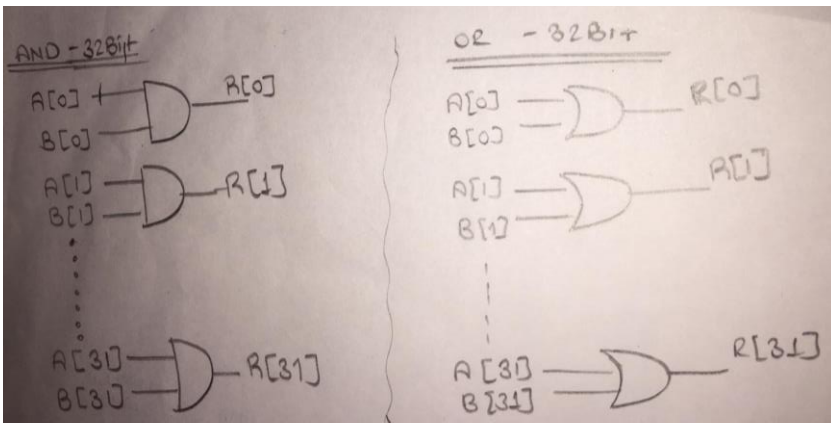
This module calls the modules shown in the above picture one by one. And then saves their results in the wires.
Then sends these wires to the 8:1 multiplexer to get the R(Result) according to the alu operation`s code.
After R is set, this module calculates Zero bit .If one or more bits in the Result number is equal to 1,This bit will be 0, otherwise will be 1.

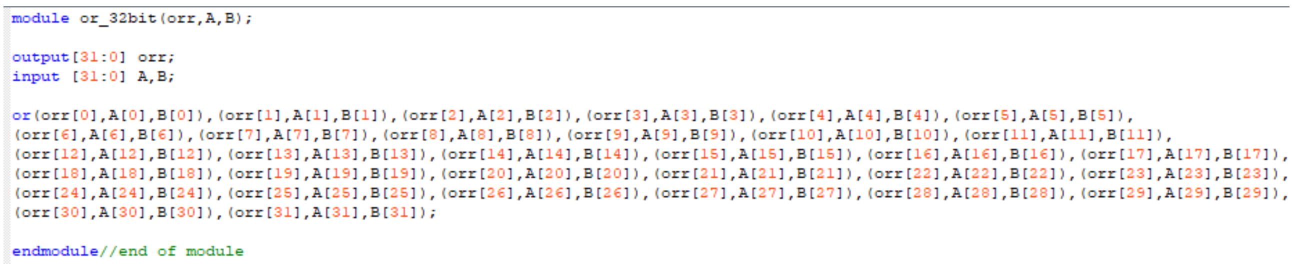
If alu operation`s code is 100 or 010(sub or add),statee will be 1,otherwise will be 0.
After state is set, C32 and V will be set; if state is 1 and the wire C/oflow is 1,C/oflow will be 1,otherwise will be 0.

The schematic design of the statee isin the side picture.



After the design of the module alu32,I designed the modules **32 bit and/or/xor/nor.** Schematic design for some modules.

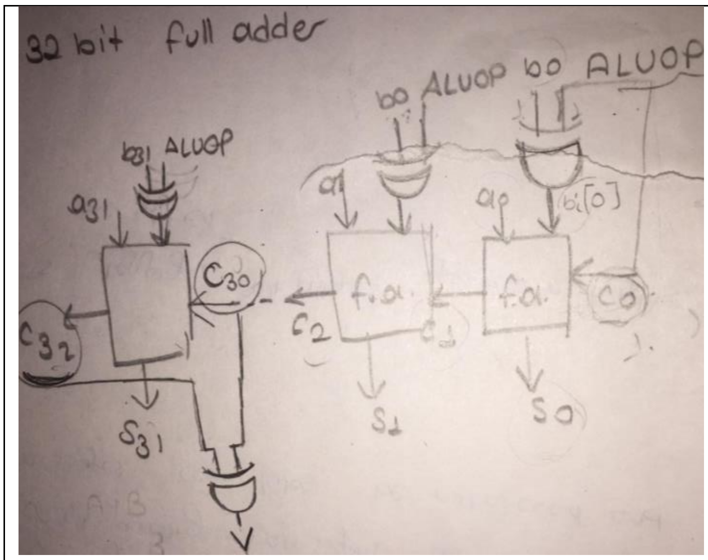
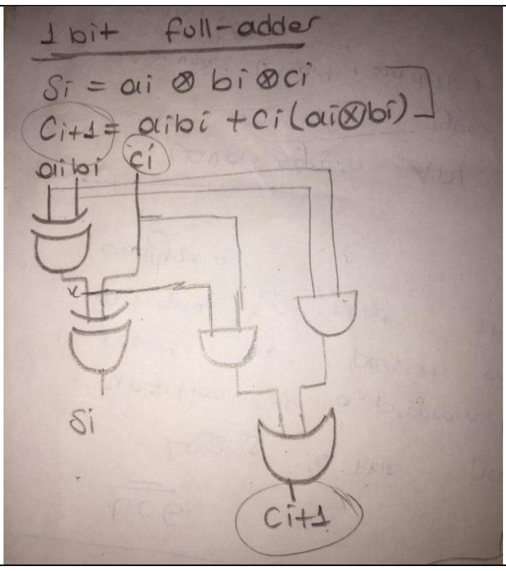


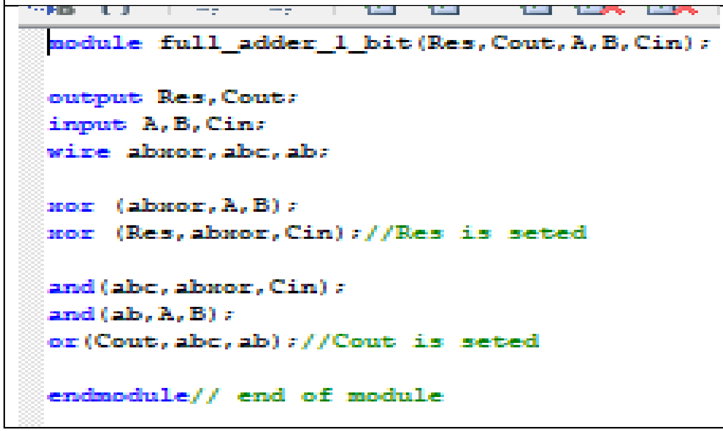


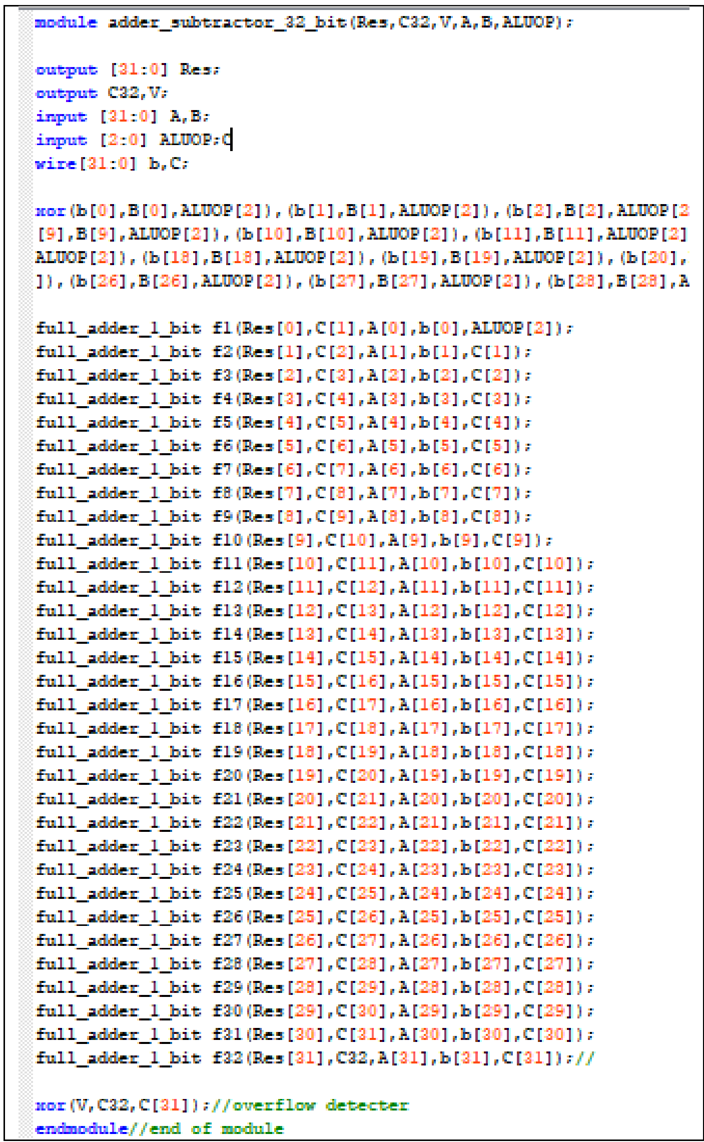
```
module or_32bit(orr,A,B);

output[31:0] orr;
input [31:0] A,B;

or(orr[0],A[0],B[0]),(orr[1],A[1],B[1]),(orr[2],A[2],B[2]),(orr[3],A[3],B[3]),(orr[4],A[4],B[4]),(orr[5],A[5],B[5]),
(orr[6],A[6],B[6]),(orr[7],A[7],B[7]),(orr[8],A[8],B[8]),(orr[9],A[9],B[9]),(orr[10],A[10],B[10]),(orr[11],A[11],B[11]),
(orr[12],A[12],B[12]),(orr[13],A[13],B[13]),(orr[14],A[14],B[14]),(orr[15],A[15],B[15]),(orr[16],A[16],B[16]),(orr[17],A[17],B[17]),
(orr[18],A[18],B[18]),(orr[19],A[19],B[19]),(orr[20],A[20],B[20]),(orr[21],A[21],B[21]),(orr[22],A[22],B[22]),(orr[23],A[23],B[23]),
(orr[24],A[24],B[24]),(orr[25],A[25],B[25]),(orr[26],A[26],B[26]),(orr[27],A[27],B[27]),(orr[28],A[28],B[28]),(orr[29],A[29],B[29]),
(orr[30],A[30],B[30]),(orr[31],A[31],B[31]);

endmodule//end of module
```

The modules 32 bit and,or,xor are same like the module 32 bit or.

Then I designed the module **adder_subtractor_32_bit**. Schematic design of this module is the below:

 32 bit full adder b31 ALUOP a31 C30 C32 S31 b0 ALUOP b0 ALUOP a1 a0 bi[0] f.a f.a. C2 C1 C0 S1 S0	 1 bit full-adder Si = ai ⊗ bi ⊗ ci Ci+1 = aibi + ci(ai⊗bi) aibi ci Si Ci+1

 `module full_adder_1_bit(Res,Cout,A,B,Cin);`
 	
`output Res,Cout;`
`input A,B,Cin;`
`wire abxor,abc,ab;`
 	
`xor (abxor,A,B);`
`xor (Res,abxor,Cin);//Res is seted`
 	
`and(abc,abxor,Cin);`
`and(ab,A,B);`
`or(Cout,abc,ab);//Cout is seted`
 	
`endmodule// end of module`
	The mdule full adder, I wirte accoording to above picture.



```
module adder_subtractor_32_bit(Res,C32,V,A,B,ALUOP);

output [31:0] Res;
output C32,V;
input  [31:0] A,B;
input  [2:0] ALUOP;C
wire[31:0] b,C;

xor(b[0],B[0],ALUOP[2]),(b[1],B[1],ALUOP[2]),(b[2],B[2],ALUOP[2
[9],B[9],ALUOP[2]),(b[10],B[10],ALUOP[2]),(b[11],B[11],ALUOP[2]
ALUOP[2]),(b[18],B[18],ALUOP[2]),(b[19],B[19],ALUOP[2]),(b[20],
]),(b[26],B[26],ALUOP[2]),(b[27],B[27],ALUOP[2]),(b[28],B[28],A

full_adder_1_bit f1(Res[0],C[1],A[0],b[0],ALUOP[2]);
full_adder_1_bit f2(Res[1],C[2],A[1],b[1],C[1]);
full_adder_1_bit f3(Res[2],C[3],A[2],b[2],C[2]);
full_adder_1_bit f4(Res[3],C[4],A[3],b[3],C[3]);
full_adder_1_bit f5(Res[4],C[5],A[4],b[4],C[4]);
full_adder_1_bit f6(Res[5],C[6],A[5],b[5],C[5]);
full_adder_1_bit f7(Res[6],C[7],A[6],b[6],C[6]);
full_adder_1_bit f8(Res[7],C[8],A[7],b[7],C[7]);
full_adder_1_bit f9(Res[8],C[9],A[8],b[8],C[8]);
full_adder_1_bit f10(Res[9],C[10],A[9],b[9],C[9]);
full_adder_1_bit f11(Res[10],C[11],A[10],b[10],C[10]);
full_adder_1_bit f12(Res[11],C[12],A[11],b[11],C[11]);
full_adder_1_bit f13(Res[12],C[13],A[12],b[12],C[12]);
full_adder_1_bit f14(Res[13],C[14],A[13],b[13],C[13]);
full_adder_1_bit f15(Res[14],C[15],A[14],b[14],C[14]);
full_adder_1_bit f16(Res[15],C[16],A[15],b[15],C[15]);
full_adder_1_bit f17(Res[16],C[17],A[16],b[16],C[16]);
full_adder_1_bit f18(Res[17],C[18],A[17],b[17],C[17]);
full_adder_1_bit f19(Res[18],C[19],A[18],b[18],C[18]);
full_adder_1_bit f20(Res[19],C[20],A[19],b[19],C[19]);
full_adder_1_bit f21(Res[20],C[21],A[20],b[20],C[20]);
full_adder_1_bit f22(Res[21],C[22],A[21],b[21],C[21]);
full_adder_1_bit f23(Res[22],C[23],A[22],b[22],C[22]);
full_adder_1_bit f24(Res[23],C[24],A[23],b[23],C[23]);
full_adder_1_bit f25(Res[24],C[25],A[24],b[24],C[24]);
full_adder_1_bit f26(Res[25],C[26],A[25],b[25],C[25]);
full_adder_1_bit f27(Res[26],C[27],A[26],b[26],C[26]);
full_adder_1_bit f28(Res[27],C[28],A[27],b[27],C[27]);
full_adder_1_bit f29(Res[28],C[29],A[28],b[28],C[28]);
full_adder_1_bit f30(Res[29],C[30],A[29],b[29],C[29]);
full_adder_1_bit f31(Res[30],C[31],A[30],b[30],C[30]);
full_adder_1_bit f32(Res[31],C32,A[31],b[31],C[31]);//

xor(V,C32,C[31]);//overflow detecter
endmodule//end of module
```

This module requests three inputs: two 32 bits number, and a 3bits alu operation`s code. And returns a 32bits number, a overflow bit and a carry out bit.

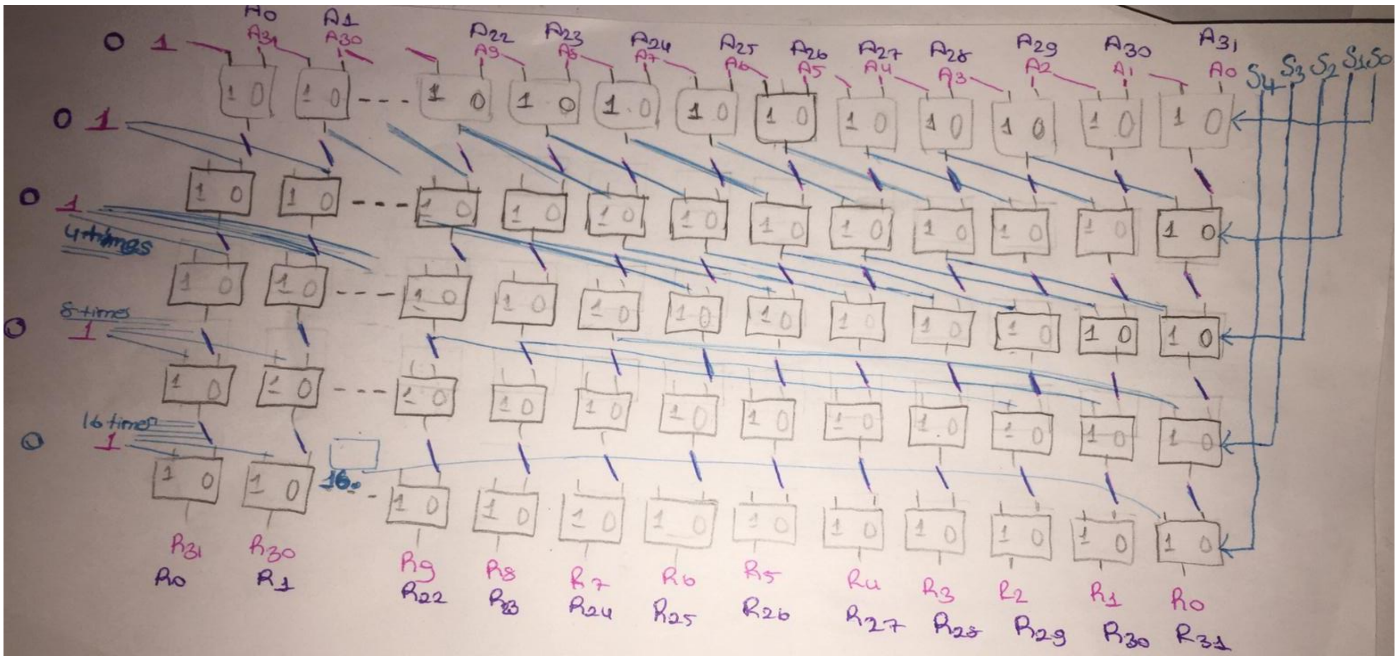
First thing, number B and aluOp[2] (if this bit is 1,this module will make subtraction ,otherwise will make addition) is sent to the xor, and the return value is set to array b.

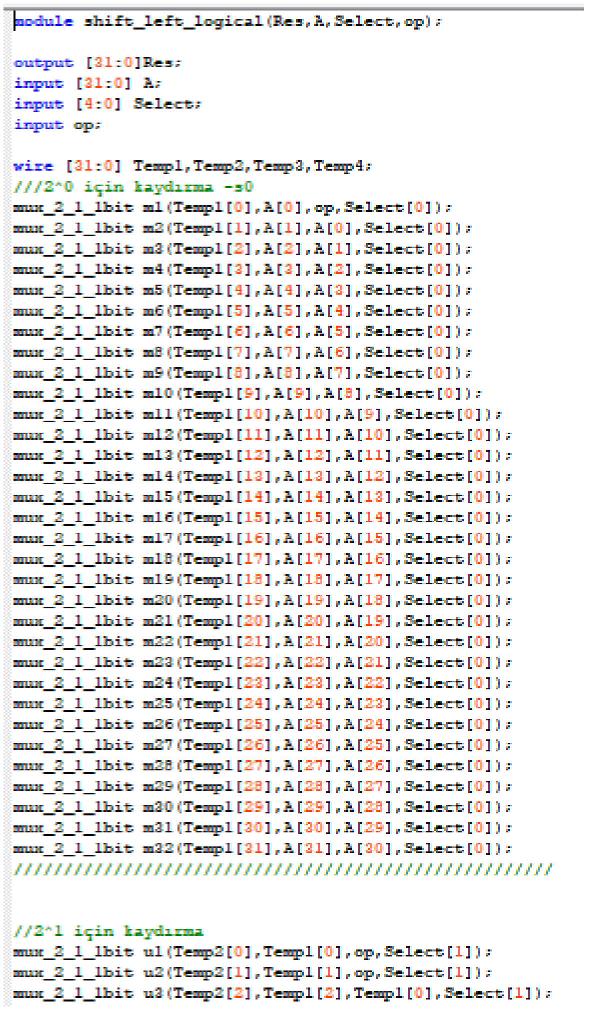
Then the bits of these two number will be sent to the module full_adder_1_bit, according to the above schematic design.

V will set according to the schematic design.

The schematic design of the module shift_left_logical (sll)and shift_right_arithmetic(sra):

The purple writings represent sll and the pink writings represent sra.



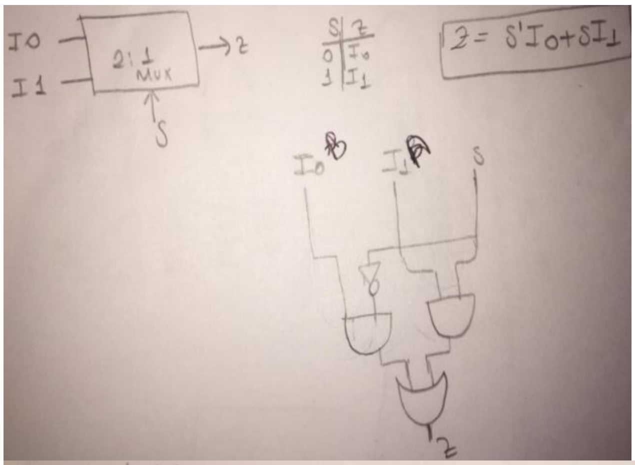


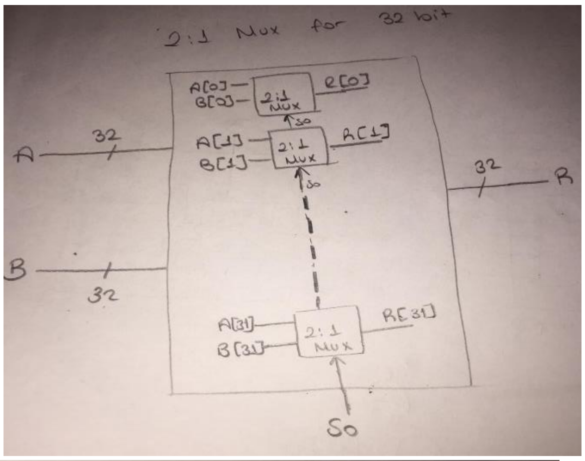
```
module shift_left_logical(Res,A,Select,op);

output [31:0]Res;
input [31:0] A;
input [4:0] Select;
input op;

wire [31:0] Temp1,Temp2,Temp3,Temp4;
///2^0 için kaydırma -s0
mux_2_1_1bit m1(Temp1[0],A[0],op,Select[0]);
mux_2_1_1bit m2(Temp1[1],A[1],A[0],Select[0]);
mux_2_1_1bit m3(Temp1[2],A[2],A[1],Select[0]);
mux_2_1_1bit m4(Temp1[3],A[3],A[2],Select[0]);
mux_2_1_1bit m5(Temp1[4],A[4],A[3],Select[0]);
mux_2_1_1bit m6(Temp1[5],A[5],A[4],Select[0]);
mux_2_1_1bit m7(Temp1[6],A[6],A[5],Select[0]);
mux_2_1_1bit m8(Temp1[7],A[7],A[6],Select[0]);
mux_2_1_1bit m9(Temp1[8],A[8],A[7],Select[0]);
mux_2_1_1bit m10(Temp1[9],A[9],A[8],Select[0]);
mux_2_1_1bit m11(Temp1[10],A[10],A[9],Select[0]);
mux_2_1_1bit m12(Temp1[11],A[11],A[10],Select[0]);
mux_2_1_1bit m13(Temp1[12],A[12],A[11],Select[0]);
mux_2_1_1bit m14(Temp1[13],A[13],A[12],Select[0]);
mux_2_1_1bit m15(Temp1[14],A[14],A[13],Select[0]);
mux_2_1_1bit m16(Temp1[15],A[15],A[14],Select[0]);
mux_2_1_1bit m17(Temp1[16],A[16],A[15],Select[0]);
mux_2_1_1bit m18(Temp1[17],A[17],A[16],Select[0]);
mux_2_1_1bit m19(Temp1[18],A[18],A[17],Select[0]);
mux_2_1_1bit m20(Temp1[19],A[19],A[18],Select[0]);
mux_2_1_1bit m21(Temp1[20],A[20],A[19],Select[0]);
mux_2_1_1bit m22(Temp1[21],A[21],A[20],Select[0]);
mux_2_1_1bit m23(Temp1[22],A[22],A[21],Select[0]);
mux_2_1_1bit m24(Temp1[23],A[23],A[22],Select[0]);
mux_2_1_1bit m25(Temp1[24],A[24],A[23],Select[0]);
mux_2_1_1bit m26(Temp1[25],A[25],A[24],Select[0]);
mux_2_1_1bit m27(Temp1[26],A[26],A[25],Select[0]);
mux_2_1_1bit m28(Temp1[27],A[27],A[26],Select[0]);
mux_2_1_1bit m29(Temp1[28],A[28],A[27],Select[0]);
mux_2_1_1bit m30(Temp1[29],A[29],A[28],Select[0]);
mux_2_1_1bit m31(Temp1[30],A[30],A[29],Select[0]);
mux_2_1_1bit m32(Temp1[31],A[31],A[30],Select[0]);
/////////////////////////////////////////////////////////


//2^1 için kaydırma
mux_2_1_1bit u1(Temp2[0],Temp1[0],op,Select[1]);
mux_2_1_1bit u2(Temp2[1],Temp1[1],op,Select[1]);
mux_2_1_1bit u3(Temp2[2],Temp1[2],Temp1[0],Select[1]);
```

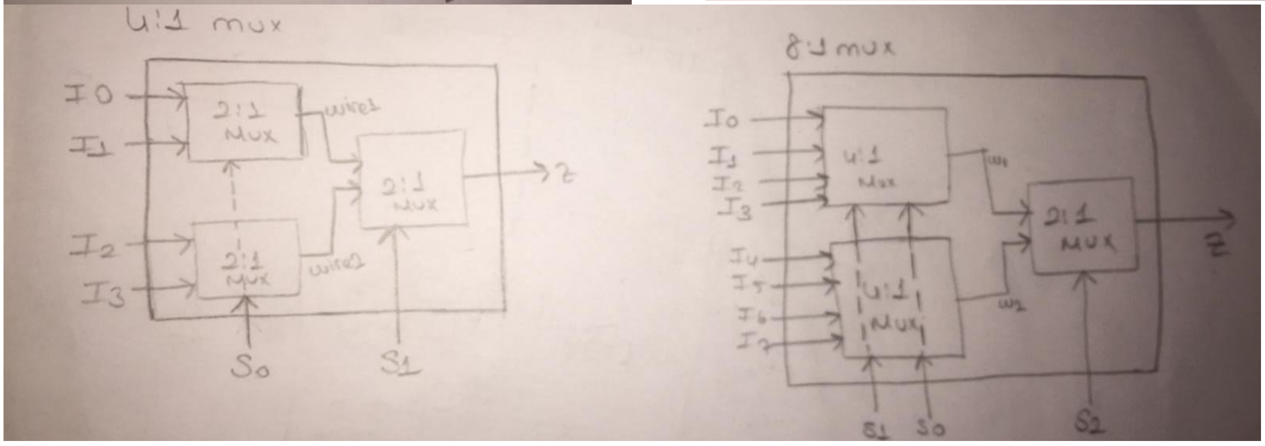
I wrote the the sra and sll modules according to their schematic design. This module sends the bit A[i] to mux with select bit and op/A[i-1] (according to the schematic design.)

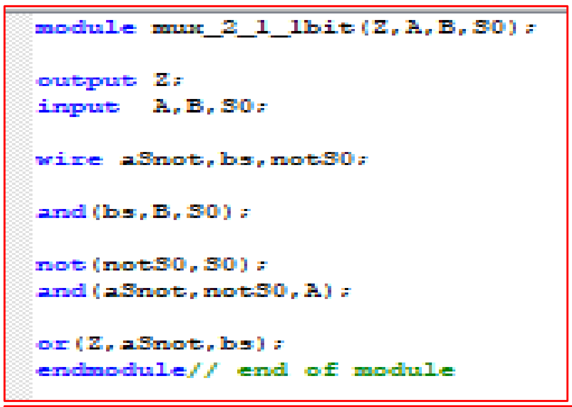
There is 5 select bit because this module takes a 32bits number.

After designing of the operation modules, I designed and wrote modules for the multiplexer.Schematic designs:



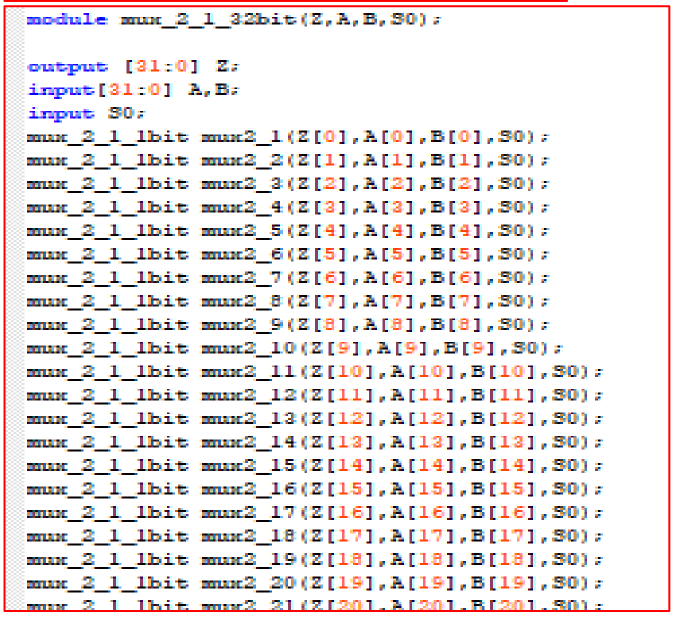






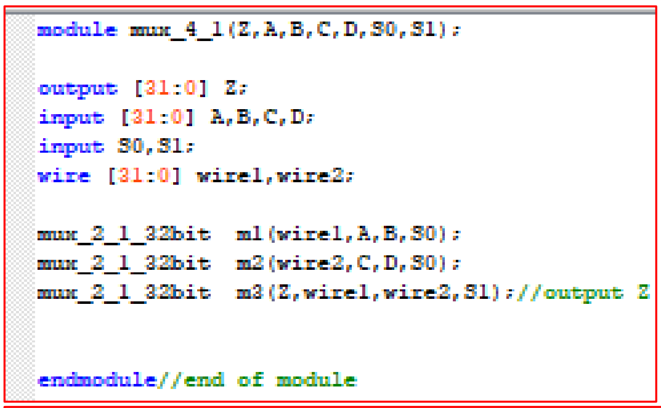
```
module mux_2_1_1bit(Z,A,B,S0);

output Z;
input  A,B,S0;

wire aSnot,bs,notS0;

and(bs,B,S0);

not(notS0,S0);
and(aSnot,notS0,A);

or(Z,aSnot,bs);
endmodule// end of module
```

First I created 2:1 mux for one bit according to the schematic design.



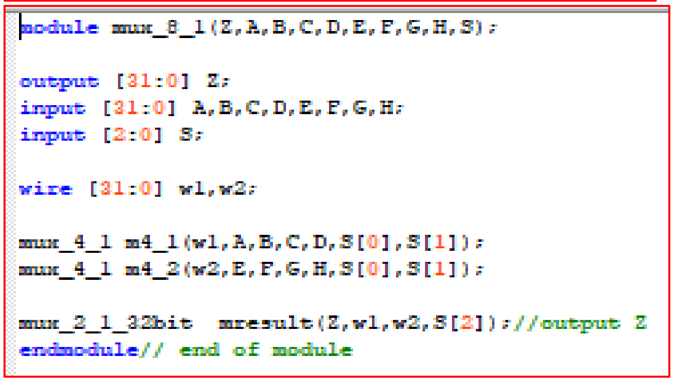
```
module mux_2_1_32bit(Z,A,B,S0);

output [31:0] Z;
input[31:0] A,B;
input S0;
mux_2_1_1bit mux2_1(Z[0],A[0],B[0],S0);
mux_2_1_1bit mux2_2(Z[1],A[1],B[1],S0);
mux_2_1_1bit mux2_3(Z[2],A[2],B[2],S0);
mux_2_1_1bit mux2_4(Z[3],A[3],B[3],S0);
mux_2_1_1bit mux2_5(Z[4],A[4],B[4],S0);
mux_2_1_1bit mux2_6(Z[5],A[5],B[5],S0);
mux_2_1_1bit mux2_7(Z[6],A[6],B[6],S0);
mux_2_1_1bit mux2_8(Z[7],A[7],B[7],S0);
mux_2_1_1bit mux2_9(Z[8],A[8],B[8],S0);
mux_2_1_1bit mux2_10(Z[9],A[9],B[9],S0);
mux_2_1_1bit mux2_11(Z[10],A[10],B[10],S0);
mux_2_1_1bit mux2_12(Z[11],A[11],B[11],S0);
mux_2_1_1bit mux2_13(Z[12],A[12],B[12],S0);
mux_2_1_1bit mux2_14(Z[13],A[13],B[13],S0);
mux_2_1_1bit mux2_15(Z[14],A[14],B[14],S0);
mux_2_1_1bit mux2_16(Z[15],A[15],B[15],S0);
mux_2_1_1bit mux2_17(Z[16],A[16],B[16],S0);
mux_2_1_1bit mux2_18(Z[17],A[17],B[17],S0);
mux_2_1_1bit mux2_19(Z[18],A[18],B[18],S0);
mux_2_1_1bit mux2_20(Z[19],A[19],B[19],S0);
mux_2_1_1bit mux2_21(Z[20],A[20],B[20],S0);
```

Then I created 2:1 mux for 32 bits number.This module call the module 2: 1 mux for one bit 32 times.



```
module mux_4_1(Z,A,B,C,D,S0,S1);

output [31:0] Z;
input [31:0] A,B,C,D;
input S0,S1;
wire [31:0] wire1,wire2;

mux_2_1_32bit  m1(wire1,A,B,S0);
mux_2_1_32bit  m2(wire2,C,D,S0);
mux_2_1_32bit  m3(Z,wire1,wire2,S1);//output Z


endmodule//end of module
```

This module use three 2:1 mulriplexers.

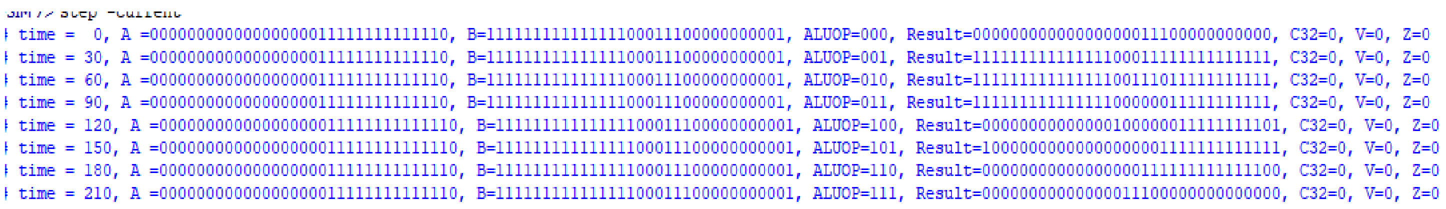


```
module mux_8_1(Z,A,B,C,D,E,F,G,H,S);

output [31:0] Z;
input [31:0] A,B,C,D,E,F,G,H;
input [2:0] S;

wire [31:0] w1,w2;

mux_4_1 m4_1(w1,A,B,C,D,S[0],S[1]);
mux_4_1 m4_2(w2,E,F,G,H,S[0],S[1]);

mux_2_1_32bit  mresult(Z,w1,w2,S[2]);//output Z
endmodule// end of module
```

This module use two 4:1 multiplxers and 1 2:1 multiplexer.

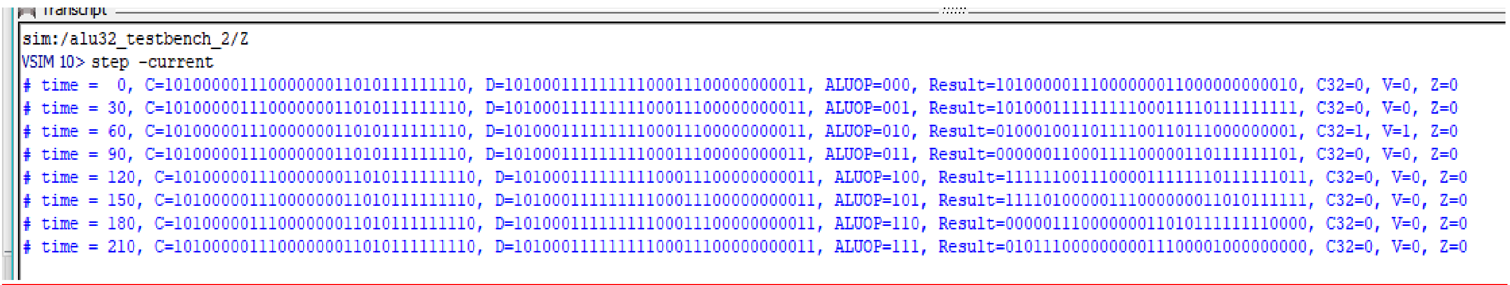
Modelsim Simulation results:

1.



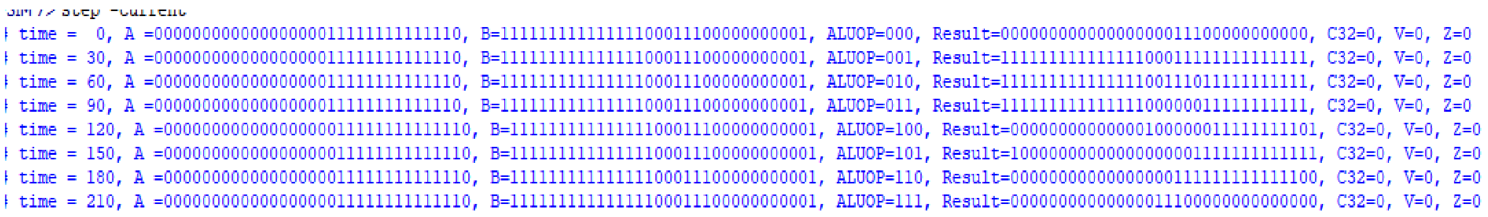
```
SIM 7> step -current
# time =  0, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=000, Result=00000000000000000011000000000000, C32=0, V=0, Z=0
# time = 30, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=001, Result=11111111111111000111111111111111, C32=0, V=0, Z=0
# time = 60, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=010, Result=11111111111111001110111111111111, C32=0, V=0, Z=0
# time = 90, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=011, Result=11111111111111100000011111111111, C32=0, V=0, Z=0
# time = 120, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=100, Result=00000000000000010000001111111101, C32=0, V=0, Z=0
# time = 150, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=101, Result=10000000000000000001111111111111, C32=0, V=0, Z=0
# time = 180, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=110, Result=00000000000000001111111111111100, C32=0, V=0, Z=0
# time = 210, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=111, Result=00000000000000011100000000000000, C32=0, V=0, Z=0
```

2.



```
Transcript
sim:/alu32_testbench_2/Z
VSIM 10> step -current
# time =  0, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=000, Result=10100000111000000011000000000010, C32=0, V=0, Z=0
# time = 30, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=001, Result=10100011111111100011110111111111, C32=0, V=0, Z=0
# time = 60, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=010, Result=01000100110111100110111000000001, C32=1, V=1, Z=0
# time = 90, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=011, Result=00000011000111100000110111111101, C32=0, V=0, Z=0
# time = 120, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=100, Result=11111100111000011111110111111011, C32=0, V=0, Z=0
# time = 150, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=101, Result=11110100000111000000001101011111, C32=0, V=0, Z=0
# time = 180, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=110, Result=00000111000000011010111111110000, C32=0, V=0, Z=0
# time = 210, C=10100000111000000011010111111110, D=10100011111111100011100000000011, ALUOP=111, Result=01011100000000011100001000000000, C32=0, V=0, Z=0
```

3.



```
SIM 7> step -current
# time =  0, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=000, Result=00000000000000000011000000000000, C32=0, V=0, Z=0
# time = 30, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=001, Result=11111111111111000111111111111111, C32=0, V=0, Z=0
# time = 60, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=010, Result=11111111111111001110111111111111, C32=0, V=0, Z=0
# time = 90, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=011, Result=11111111111111100000011111111111, C32=0, V=0, Z=0
# time = 120, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=100, Result=00000000000000010000001111111101, C32=0, V=0, Z=0
# time = 150, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=101, Result=10000000000000000001111111111111, C32=0, V=0, Z=0
# time = 180, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=110, Result=00000000000000001111111111111100, C32=0, V=0, Z=0
# time = 210, A =00000000000000000011111111111110, B=11111111111111000111000000000001, ALUOP=111, Result=00000000000000011100000000000000, C32=0, V=0, Z=0
```