# CSE 331 Computer Organization
## Project 2 – ALU with Structural Verilog

The ALU will get two 32-bit numbers **A** and **B**, and a 3-bit **S** (select) signal as inputs and a 32-bit **Y** signal as output. The ALU perform the following operations and modules:

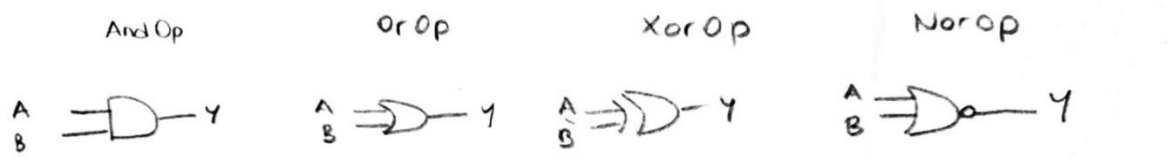
module alu32(ALUOP,A,B,C,V,Z,Y);

alu32 module is the top level module.

A and B is the 32 bit inputs. V is the overflow detect bit. Z is the zero detect bit.

Y is the 32 bit output bit.C is carry out bit.

**1.And , Or ,Xor and Nor operations module's schematic designs and their descriptions**




module andop(Y,A,B); // that module takes 32 bit A and B inputs and Y output

module orop(Y,A,B); // that module takes 32 bit A and B inputs and Y output

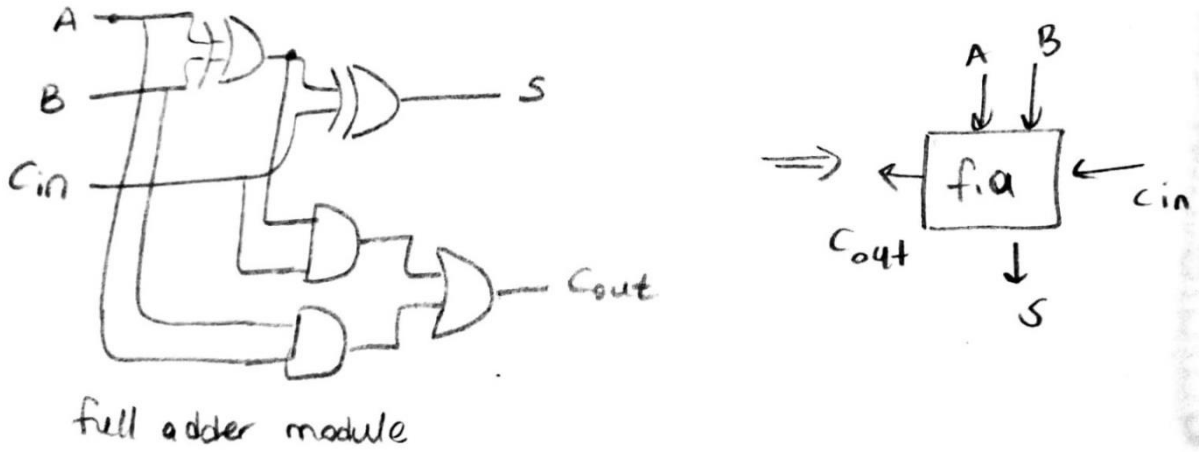
module norop(Y,A,B); // that module takes 32 bit A and B inputs and Y output

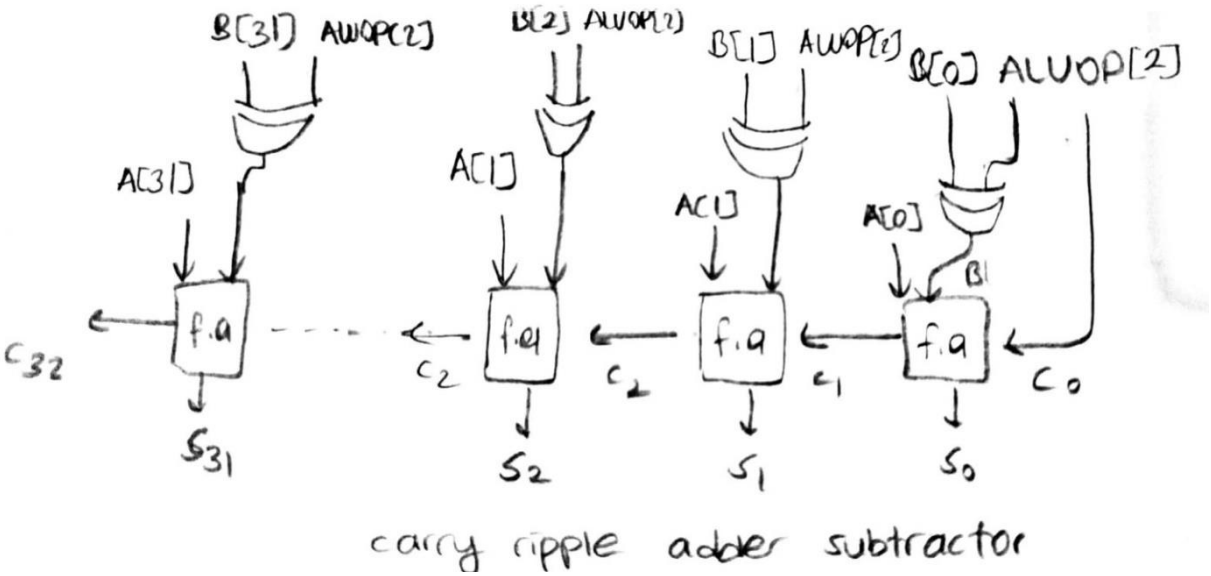
module xorop(Y,A,B); // that module takes 32 bit A and B inputs and Y output

All modules that described above process every bit one by one and assign result to Y's related bit.

That process continue for every 32 bit of A and B inputs.

**2.Full adder module's and carry ripple adder and subtractor modules' schematics and their descriptions**



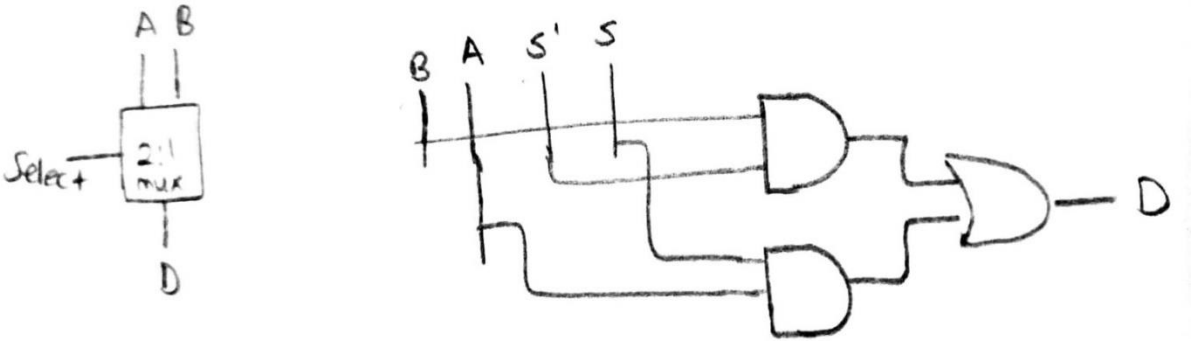



module full_adder(Result, Cout, A, B, Cin);// full adder module takes A  B and Cin(carry bit) one bit inputs and gives one bit Result and Cout (carry bit )for output.It wrote to use with carry ripple adder and subtractor.

module carry_ripple_adder_subtractor(ALUOP,A,B,Cout,S,Vtemp);  //carry ripple adder subtractor modules uses full adder.It has A and B for input 32 bits and 3 bit ALUOP for doing right process.

010 ALUOP A+B
100 ALUOP A-B
According to ALUOP bits carry ripple check's ALUOP[2] for choosing add or subtract operation.The above schematic shows full adder takes two inputs.First is A's bit and second is B's bit and ALUOP[2]'s xor output.In here xor operation's output specify the add or subtract operation.

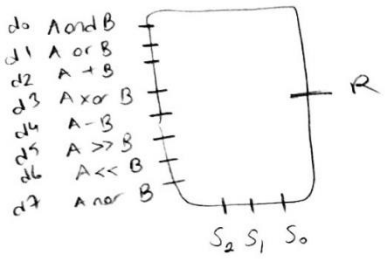
**3. 2x1 mux module design for shift operations**



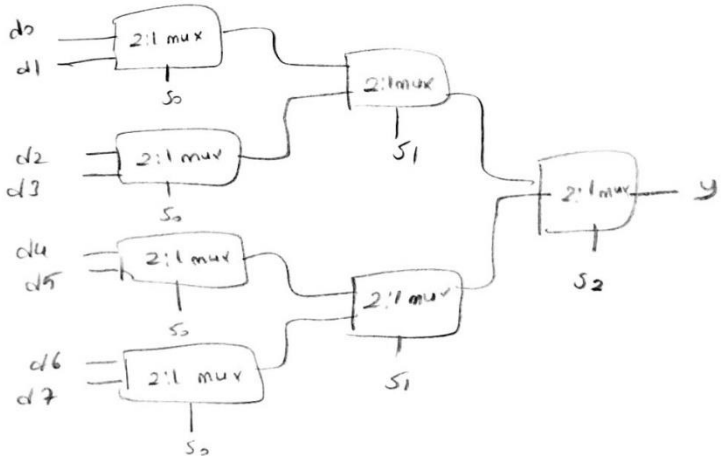

**module mux_2x1(A,B,S,D);**

As seen above described module 2x1 mux takes B and A as one bit input and S is input for one bit select. D is output one bit.

**4. 8x1 mux for choosing the true operation for output with using 2x1 muxes**




$$R = \bar{S}_2.\bar{S}_1.\bar{S}_0.I_0 + \bar{S}_2.\bar{S}_1.S_0.I_1 + \bar{S}_2.S_1.\bar{S}_0.I_2 + \bar{S}_2.S_1.S_0.I_3 + S_2.\bar{S}_1.\bar{S}_0.I_4 + S_2.\bar{S}_1.S_0.I_5 + S_2.S_1.\bar{S}_0$$




module mux_2_x_1(B,A,S,D);

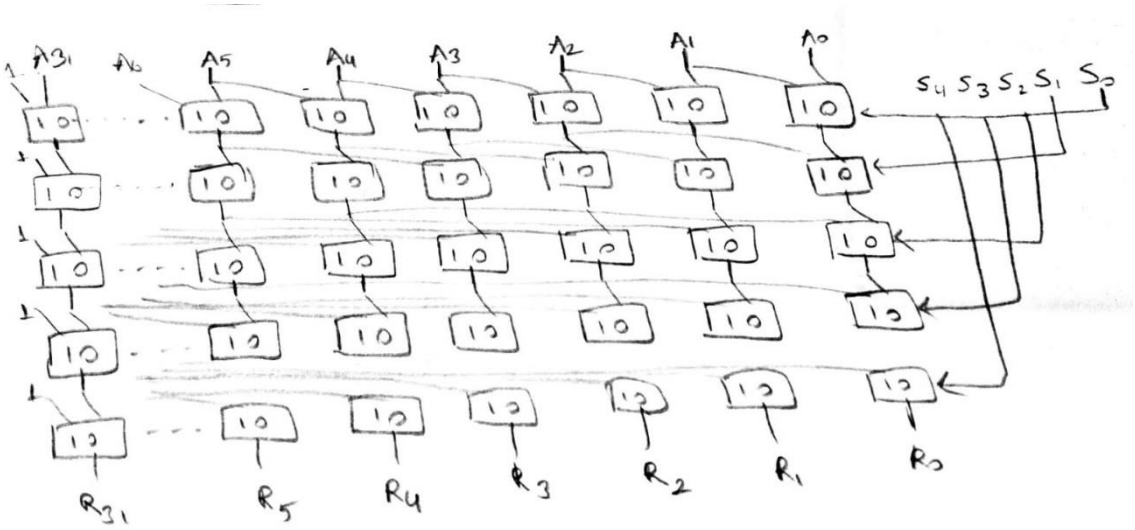
As seen above described module 2x1 mux takes B and A as 32 bits and S is the one bit select bit.D is the 32 bit output.It designed for operation choosing. Because of the A and B bits are 32 bit this mux is designed for 32 bit and module's calling.Other 2x1 mux can't work this for operation because it works for one bit inputs.

module mux_8_x_1(I0,I1,I2,I3,I4,I5,I6,I7,S,Y);

For that 8x1 mux declared above used module mux_2_x_1 and send it 32 bit inputs for every operation's result with the 1 bit select and Y is output. 2x1 mux needs 32 bit input and 1 bit select to and opeation with them.So new and operation is wrote for it;

module andop2(Y,A,B); // module that process and operation with 32 bit input and 1 bit input one by one

**5.Right and left shifter module's design**



(design is right shifter's)

module right_shifter(R,A,S);

Right shifter module takes one 32 bit input(A) , 5 bit select bits(S) and one 32 bit output(R).It uses 2x1 muxes and firstly , it sends mux to first bit of input and second bit of the input.Then it increases the bits until the A[31].If the bit run short of the input paramater of the bit 1 is the second parameter.

Second step is shifting the input bit by 2 and and the input is the previous mux's output.Other step is shifting the input bit by 4 then, 8 and last is 16 bit shifting for getting input from the input bit of A.Select bit is firstly fitst bit of the S and it continue by increasing one.

Result is the last step's muxes output's for every bit taken.

module left_shifter(L,A,S);

Left shifter is the almost the same design with right shifter.It uses same number of muxes and shift numbers but left shifter gets input opposite of the right shifter.Left shifter starts with the A[31] and get output from the mux from the last bit again.Additionally it gives the input 0 instead 1 like in the right shifter.

**6.Test benches;**

module alu32_testbench_1();

module alu32_testbench_2();

module alu32_testbench_3();

# Test Results

```
sim:/alu32_testbench_1/Z
VSIM 18> step -current
# time =  0, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=000, Result=10001000001100000011100000000100, Cout=0, Overflw=0, Zero=0
# time = 20, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=001, Result=11101111101101100011111111111111, Cout=0, Overflw=0, Zero=0
# time = 40, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=010, Result=01110111111001100111100000000011, Cout=1, Overflw=1, Zero=0
# time = 60, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=011, Result=01100111100001100000011111111011, Cout=0, Overflw=0, Zero=0
# time = 80, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=100, Result=10011000011110100000011111111001, Cout=0, Overflw=0, Zero=0
# time = 100, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=101, Result=11111100010000011000000111111111, Cout=0, Overflw=0, Zero=0
# time = 120, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=110, Result=00000110000001111111111111000000, Cout=0, Overflw=0, Zero=0
# time = 140, A=10001000001100000011111111111110, B=11101111101101100011100000000101, ALUOP=111, Result=00010000010010011100000000000000, Cout=0, Overflw=0, Zero=0
```



```
sim:/alu32_testbench_2/V \
sim:/alu32_testbench_2/Z
VSIM 21> step -current
# time =  0, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=000, Result=00000000000000000000000000000000, Cout=0, Overflw=0, Zero=1
# time = 20, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=001, Result=11111111111111100011111111111111, Cout=0, Overflw=0, Zero=0
# time = 40, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=010, Result=11111111111111100011111111111111, Cout=0, Overflw=0, Zero=0
# time = 60, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=011, Result=11111111111111100011111111111111, Cout=0, Overflw=0, Zero=0
# time = 80, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=100, Result=00000000000000100011111111111101, Cout=0, Overflw=0, Zero=0
# time = 100, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=101, Result=10000000000000000011111111111111, Cout=0, Overflw=0, Zero=0
# time = 120, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=110, Result=00000000000000000111111111111100, Cout=0, Overflw=0, Zero=0
# time = 140, A=00000000000000000011111111111110, B=11111111111111100000000000000001, ALUOP=111, Result=00000000000000011100000000000000, Cout=0, Overflw=0, Zero=0
```



```
sim:/alu32_testbench_3/Z
VSIM 15> step -current
# time =  0, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=000, Result=10100000111000000011000000000010, Cout=0, Overflw=0, Zero=0
# time = 20, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=001, Result=10100011111111110001111010111111, Cout=0, Overflw=0, Zero=0
# time = 40, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=010, Result=01000100110111100110111000000001, Cout=1, Overflw=1, Zero=0
# time = 60, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=011, Result=00000011000111100000110111111101, Cout=0, Overflw=0, Zero=0
# time = 80, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=100, Result=11111100111000011111110111111011, Cout=0, Overflw=0, Zero=0
# time = 100, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=101, Result=11110100000111000000011010111111, Cout=0, Overflw=0, Zero=0
# time = 120, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=110, Result=00000111000000011010111111110000, Cout=0, Overflw=0, Zero=0
# time = 140, A=10100000111000000011010111111110, B=10100011111111110001110000000011, ALUOP=111, Result=01011100000000011100001000000000, Cout=0, Overflw=0, Zero=0
```