CSE331 HOMEWORK 2 REPORT

32-BIT ALU DESIGN

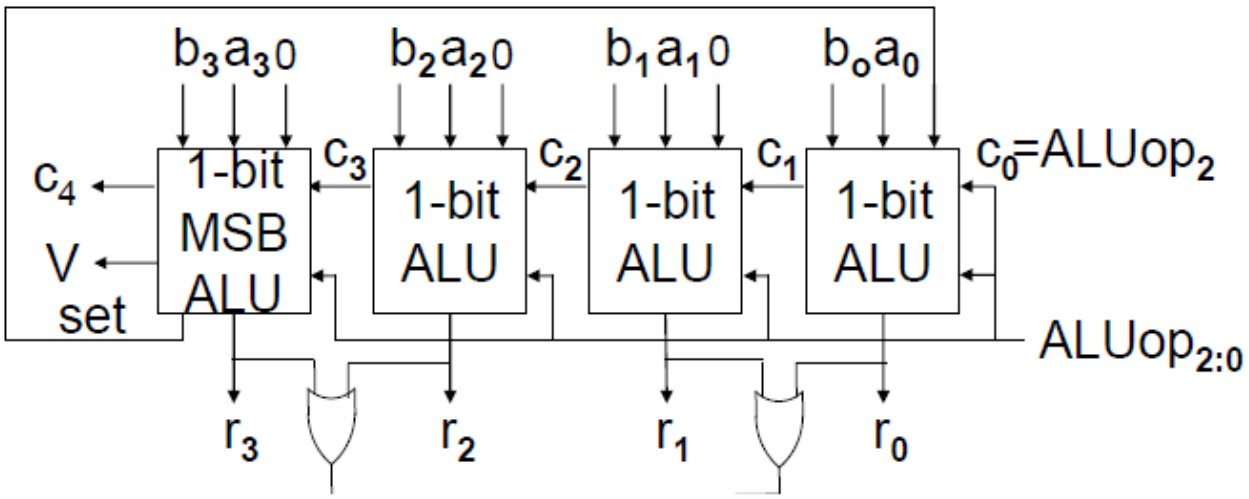
Introduction

ALU(Arithmetic Logic Unit) is a complex component that computes the specified operation given through input by user.The Arithmetic Logic Unit(ALU) that designed in this project supports five operations:

AND,OR,ADD,SUBTRACTION,SET-LESS-THAN

ALU takes given 2 32-bit number as parameter for computing and also 3 choice bits(AluOp[0:2]) to determine which operation will be performed.

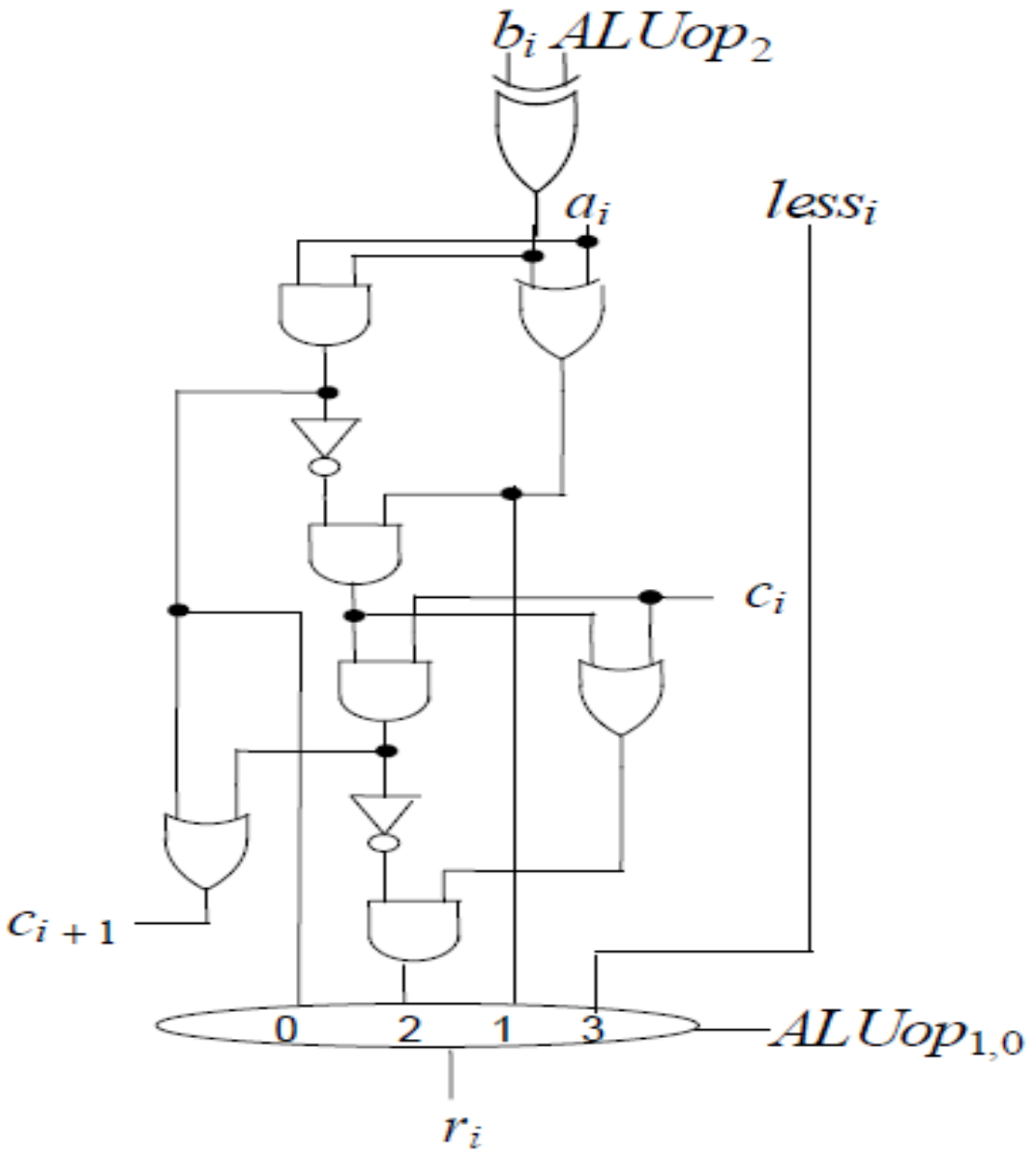
Structural Design



32-Bit Alu is compose of combining 32 1-bit alu as putting them sequential in a order.

In order to achieve this,first of all 1-bit alu needed to be designed.

The structural logic of 1-bit alu is explained below.



This structure performs the 4 operation for 1-bit input by using 4x1 multiplexer and an XOR gate.

Therefore the 4x1 multiplexer and XOR modules were needed to be created before designing 1-bit ALU.

XOR Module:

```
module myXOR (
input inputXOR1,
input inputXOR2,
output outputXOR
 );
  wire inputXOR1_not, inputXOR2_not;
  wire wireXOR1,wireXOR2;
    not notForXor1 (inputXOR1_not, inputXOR1);
    not  notForXor2 (inputXOR2_not, inputXOR2);
    and andForXor1 (wireXOR1, inputXOR1_not, inputXOR2);
    and andForXor2 (wireXOR2, inputXOR2_not, inputXOR1);
    or orForXor1 (outputXOR, wireXOR1, wireXOR2);
endmodule
```

5 gates are used to design XOR Module

4x1 Mux Module:

```
module FourToOneMux(
input i0,
input i1,
input i2,
input i3,
input s0,
input s1,
output resultMux
);
    wire not_s0,not_s1;
    wire m00,m01,m10,m11;
    wire r00,r01,r10,r11;
    wire sum1,sum2;
    not notForMux1 (not_s0,s0);
    not notForMux2 (not_s1,s1);
    and andForMux1 (m00,not_s1,not_s0);
    and andForMux2 (m01,not_s1,s0);
    and andForMux3 (m10,s1,not_s0);
    and andForMux4 (m11,s1,s0);

    and andForMux5 (r00,m00,i0);
    and andForMux6 (r01,m01,i1);
    and andForMux7 (r10,m10,i2);
    and andForMux8 (r11,m11,i3);

    or orForMux1 (sum1,r00,r01);
    or orForMux2 (sum2,r10,r11);
```

13 logic Gates are used to create 4x1 mux

1-Bit-Alu Module:

```
module OneBitAlu(
    input a1,
    input b1,
    input less1,
    input cin1,
    input aluOP0,
    input aluOP1,
    input aluOP2,
    output cout1,
    output r1
    );
    wire xorBi;
    myXOR xorFor1Bit1(b1, aluOP2, xorBi);
    wire andAB,orAB;
    and andFor1Bit1 (andAB,a1,xorBi);
    or rFor1Bit1  (orAB,a1,xorBi);
    wire notAB;
    not notFor1Bit1  (notAB,andAB);
    wire w1;
    and andFor1Bit2 (w1,notAB,orAB);
    wire orGate;
    or orFor1Bit2 (orGate,cin1,w1);
    wire andGate;
    and andFor1Bit3  (andGate,cin1,w1);
    or rFor1Bit3 (cout1,andAB,andGate);
    wire w2;
    not notFor1Bit2 (w2,andGate);
```

1-Bit-Alu uses 9 logic Gates additional to 4x1 mux and an XOR gate which makes total number logic Gates = 9 + 13 + 5 = 27

Note: The most significant bit of 32 bit alu is a msb alu which has additional 2 xor gate which makes it 27+2*5 = 37.

32-Bit-ALU Module:

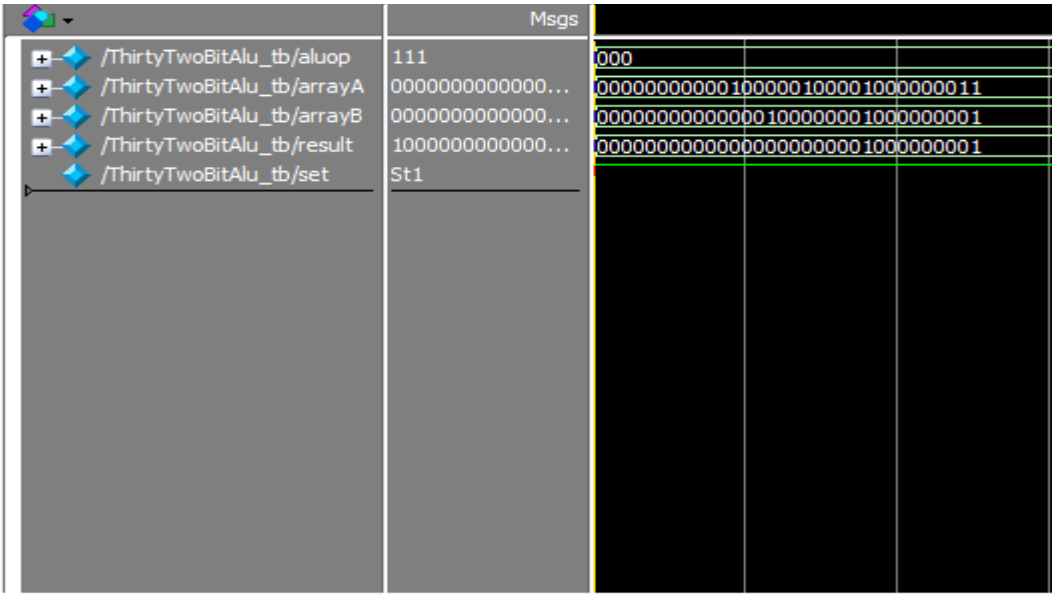
As mentioned in 1-Bit-Alu module,32-bit Alu is a combining of 31 1-bit-alus and a 1-bit-msb-alu.In order to compute Z value,31 or gates are used.Therefore the total number of logic Gates is 37+31*27+31 = 905

TESTBENCH

The program tested with 4 different input pair of A and B and also for 5 operation that 32-Bit-Alu supports.The testbench code is below

```
 initial begin
  arrayA = 32'b00000000000100000100001000000011;
  arrayB = 32'b00000000000000010000000100000001;
  aluop =  3'b000;
  #20
  arrayA = 32'b00000000000000011110000000000010;
  arrayB = 32'b00000000000000000000000001001001;
  aluop =  3'b001;
  #20
  arrayA = 32'b00000000000000000000000000000010;
  arrayB = 32'b00000000000000000000000000000001;
  aluop =  3'b010;
  #20
  arrayA = 32'b00000000000000000000000000001111;
  arrayB = 32'b00000000000000000000000000000011;
  aluop =  3'b110;
  #20
  arrayA = 32'b00000000000000000000000000000001;
  arrayB = 32'b00000000000000000000000000000111;
  aluop =  3'b111;
  #20
  $finish;
end
```

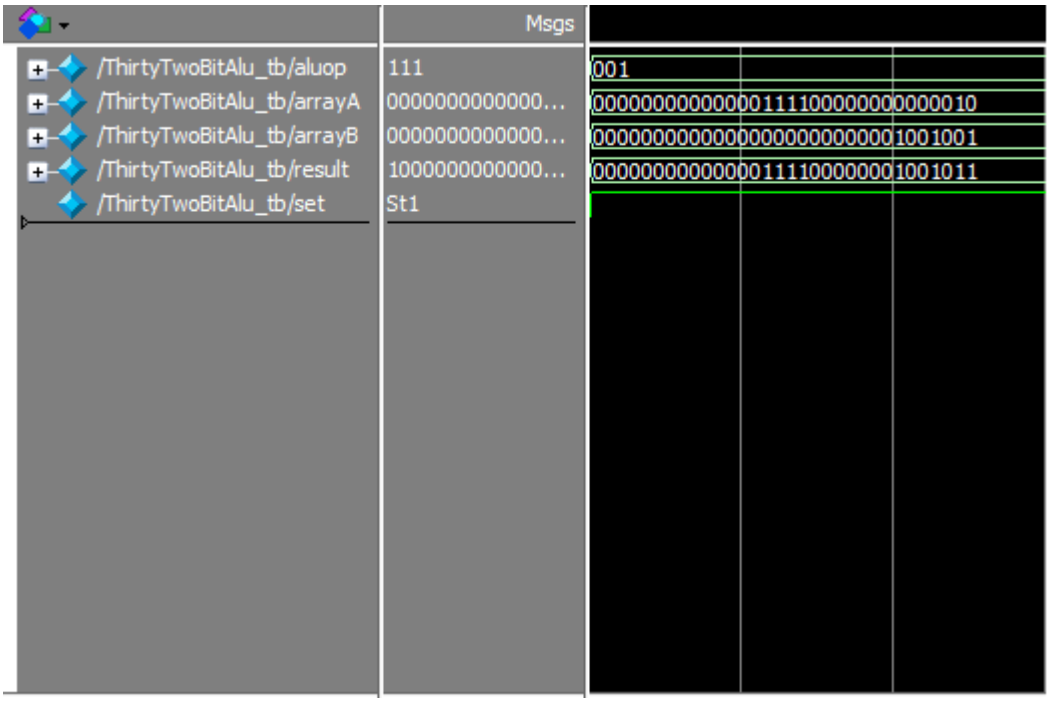
For each time shift the output is below:



In the input above,the user input for operation is 000 which means add operation.32-bit numbers(A,B) and result(R) are:

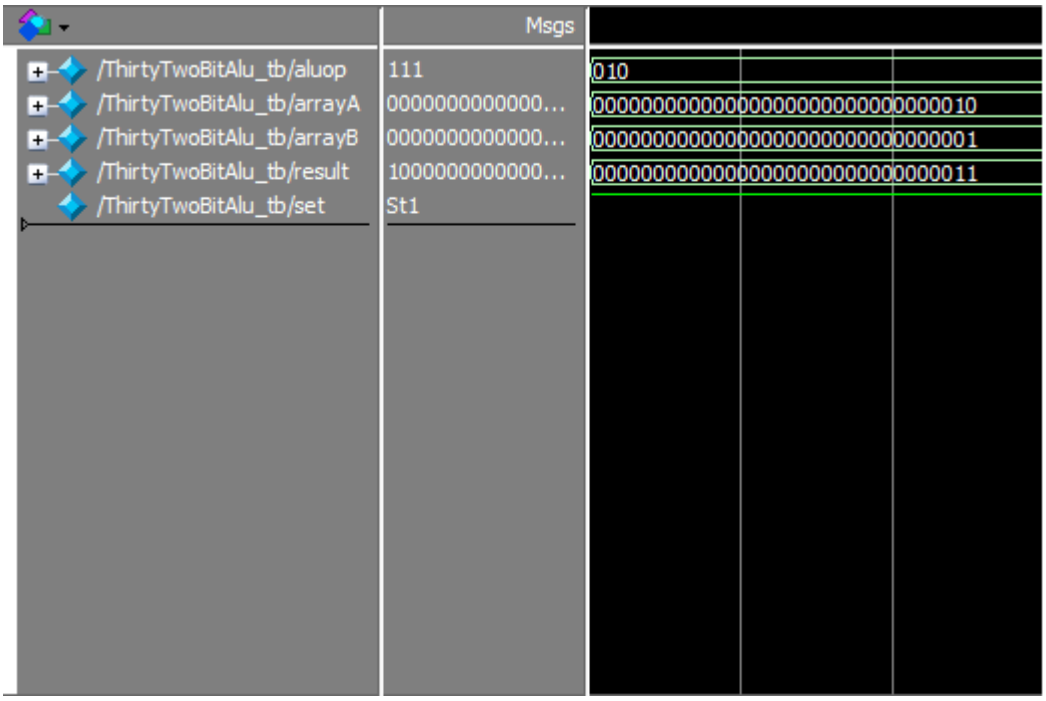
A:00000000000100000100001000000011
B:00000000000000100000001000000001
R:00000000000000000000001000000001

Which is correct result.

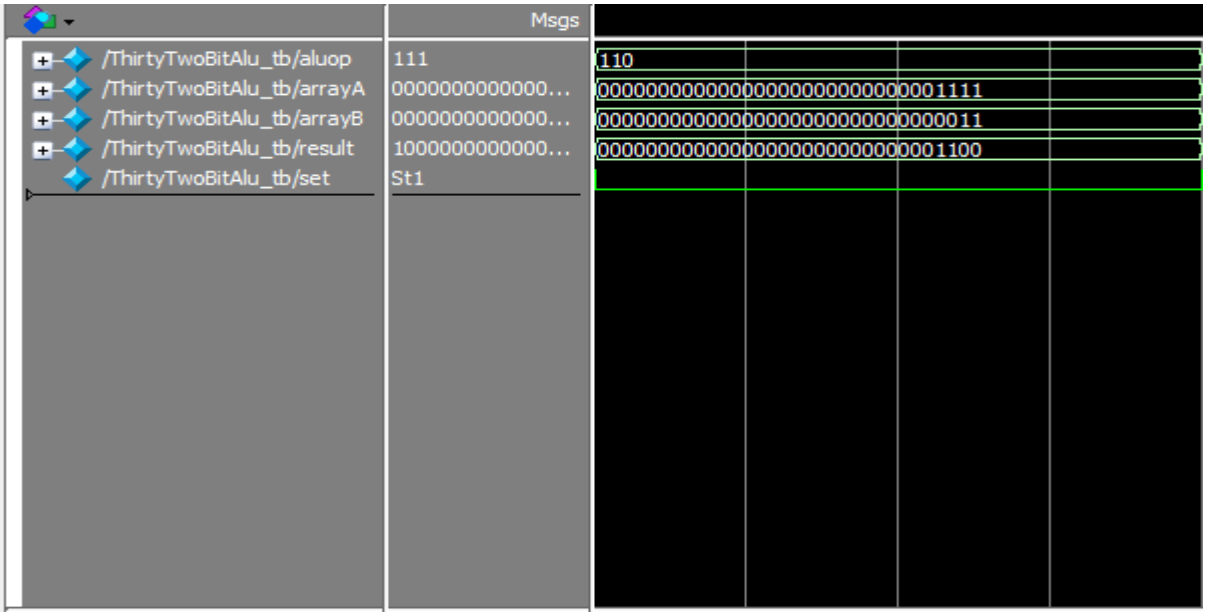


Second operator is or(001)

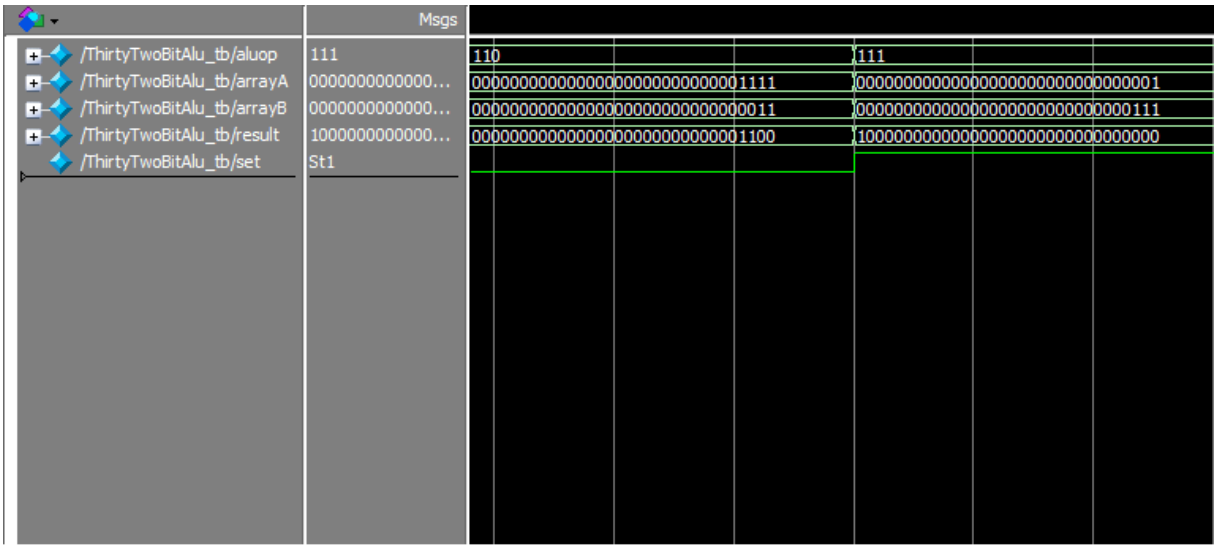
A:00000000000000111100000000000010
B:00000000000000000000000001001001
R:00000000000000111100000001001011
Correct answer.



Third operator is add(010)
A:00000000000000000000000000000010
B:00000000000000000000000000000001
R: 00000000000000000000000000000011
correct answer.



Fourth operation is subtraction(110)
A:00000000000000000000000000001111
B:00000000000000000000000000000011
R: 00000000000000000000000000001100
correct answer.



The last operation is set-less-than(111)

A:00000000000000000000000000000001
B: 00000000000000000000000000000111
In set-less-than operation,

i) İf A<B , then z = 1
ii) İf A>B, then z = 0

In this example A is less than B.Therefore the Z output has value of 1.