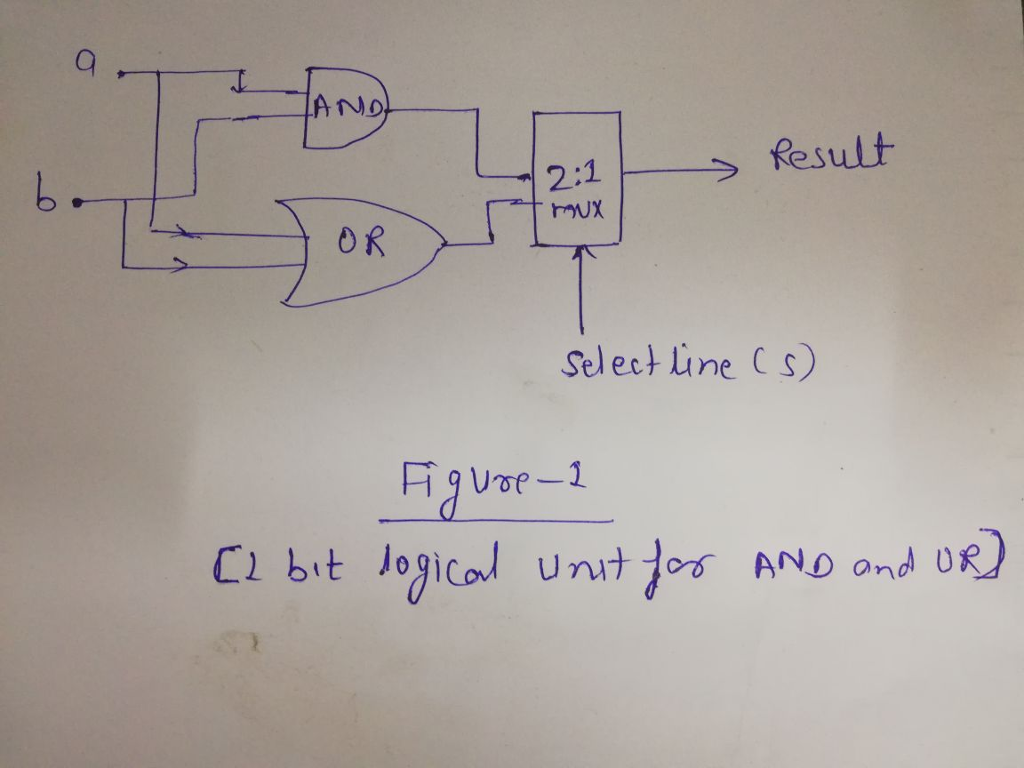
**Describe the 1-bit ALU enhanced with overflow detection. Draw the diagram by hand.**

ALU operation means performing 1 bit logical operation or 1-bit arithmetic operation.Logical operation includes performing bit wise AND, bitwise OR etc and arithmetic operation includes performing addition ,subtraction etc.

Here 1-bit ALU means all inputs for operation are 1-bit.

The 1-bit logical unit for AND and OR looks like Figure 1. The multiplexor on the right then selects a AND b or a OR b, depending on whether the value of Operation is 0 or 1.



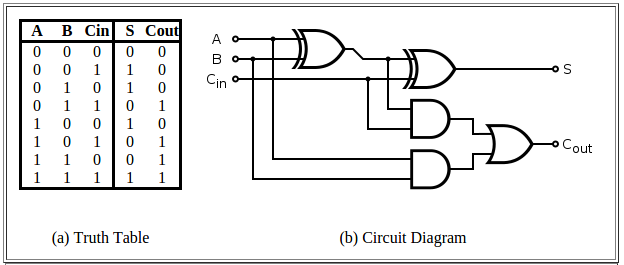
The next function to include is addition. An adder must have two inputs for the operands and a single-bit output for the sum. There must be a second output to pass on the carry, called CarryOut.Since the CarryOut from the neighbor adder must be included as an input, we need a third input. This input is called CarryIn. Figure-

Figure-2 shows the inputs and the outputs of a 1-bit adder

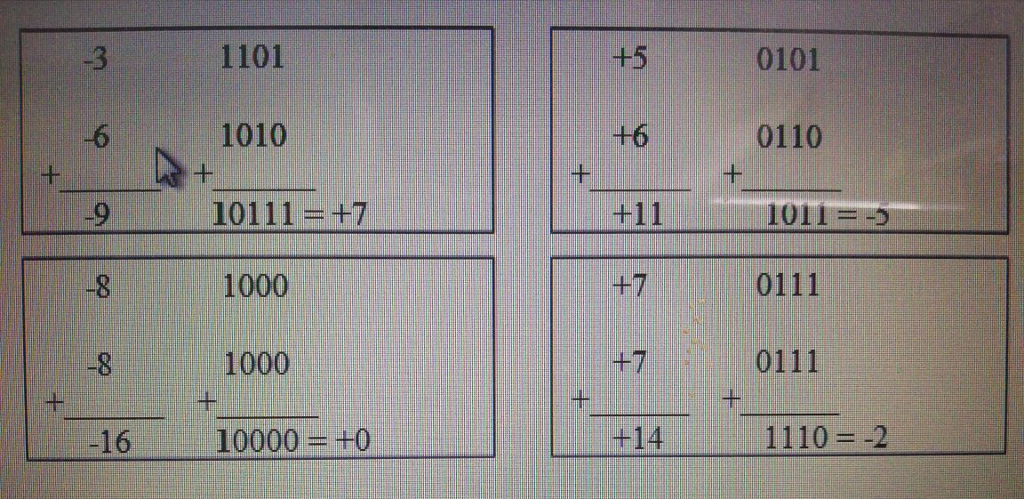
Here A,B and Cin are input to 1-bit full adder and S means and Cout are output of full adder.

**Overflow:**

Overflow occurs when there are insufficient bits in a binary number representation to portray the result of an arithmetic operation. Overflow occurs because computer arithmetic is not closed with respect to addition, subtraction, multiplication, or division. Overflow *cannot* occur in addition (subtraction), if the operands have different (resp. identical) signs.

Example:

These are some example shows that how overflow changes the sign of result.



Here are some facts about overflow in 2C.

* If **x** and **y** have opposite signs (one is negative, the other is non-negative), then the sum will never overflow. Just try it out. The result will either be **x** or **y** or somewhere in between.
* Thus, overflow can only occur when **x** and **y** have the same sign.
* One way to detect overflow is to check the sign bit of the sum. If the sign bit of the sum does not match the sign bit of **x** and **y**, then there's overflow. This only makes sense.
* Suppose **x** and **y** both have sign bits with value 1. That means, both representations represent negative numbers. If the sum has sign bit 0, then the result of adding two negative numbers has resulted in a non-negative result, which is clearly wrong. Overflow has occurred.
* Suppose **x** and **y** both have sign bits with value 0. That means, both representations represent non-negative numbers. If the sum has sign bit 1, then the result of adding two non-negative numbers has resulted in a negative result, which is clearly wrong. Overflow has occurred.

**A Simpler Formula for Overflow:**

V = ck-1 XOR ck-2

if V=1 then there is overflow else not.

A 1-bit ALU circuit of AND , OR and 1-bit additition with overflow detection is given below.

