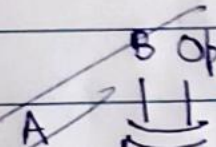
2. For the implementation of 8-bit adder/subtractor we have used 8-1 bit adder/subtractor for each bit of the no.. We have used a input called as opcode which is 0 and 1 for addition and subtraction respectively.

Now it is not always true whether we can represent addition/subtraction of 2 number as a 8 bit number and this condition is called overflow in which our answer will come out to be incorrect.
We have given an output variable which is 1 when there is overflow condition and in rest case it is 0.

1-bit adder/subtractor.



1 bit adder.

a	b	c_{in}	sum	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Sum:

C_i \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

→ $Sum = C_i \oplus A \oplus B$

Cout:

C_i \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

→ $C_o = C_i \cdot A + C_i \cdot B + A \cdot B$.

→ Description:

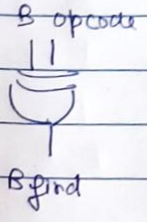
① We have designed first a normal 1 bit adder then, ~~A-B~~

A-B == A + (2'comp B) == A + (1'comp B) + 1 = A + (inversion gate

Now for the inversion we have used a XOR gate

⇒ whenever op code = 1 ⇒ invert else don't invert

B	opcode	Bfinal
0	0	0
0	1	1
1	0	1
1	1	0

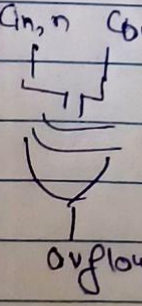


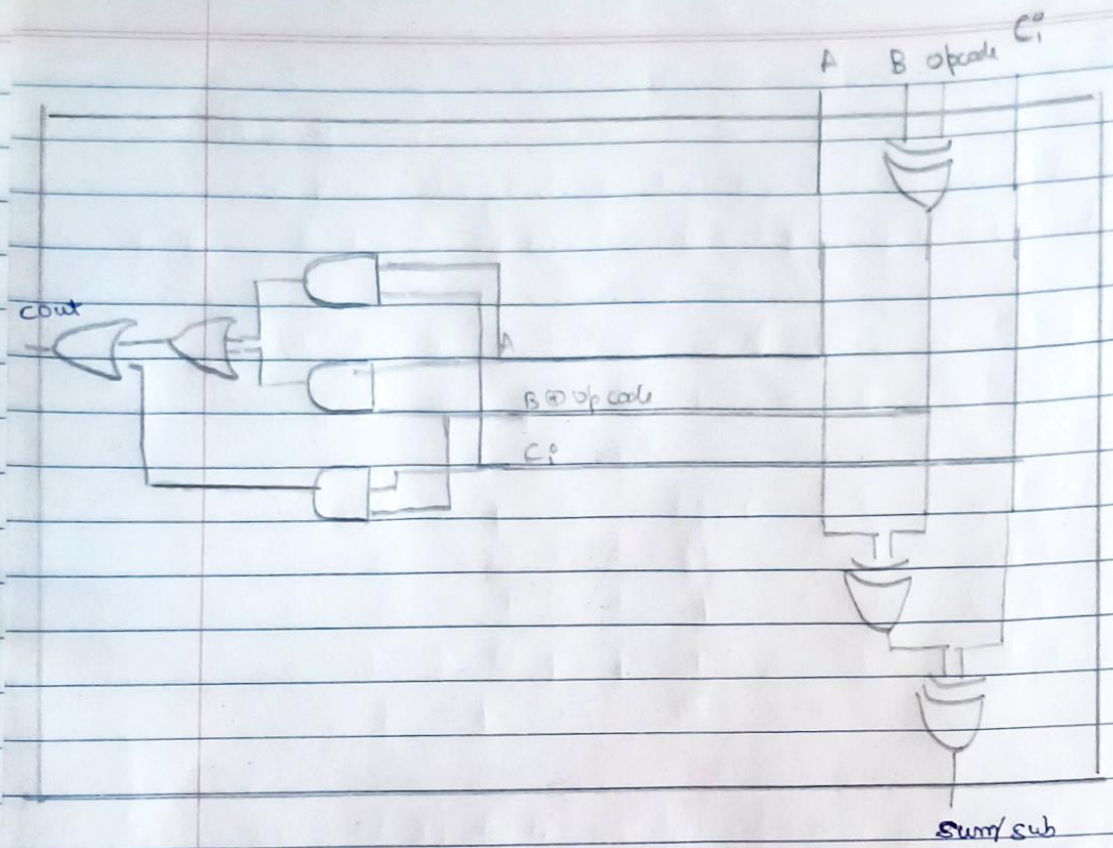
② To add 1 do the whole answer we have given the Cin of first 1 bit adder as opcode so that whenever opcode = 1 ~~use~~, to find 2's complement we can add 1 to inversion

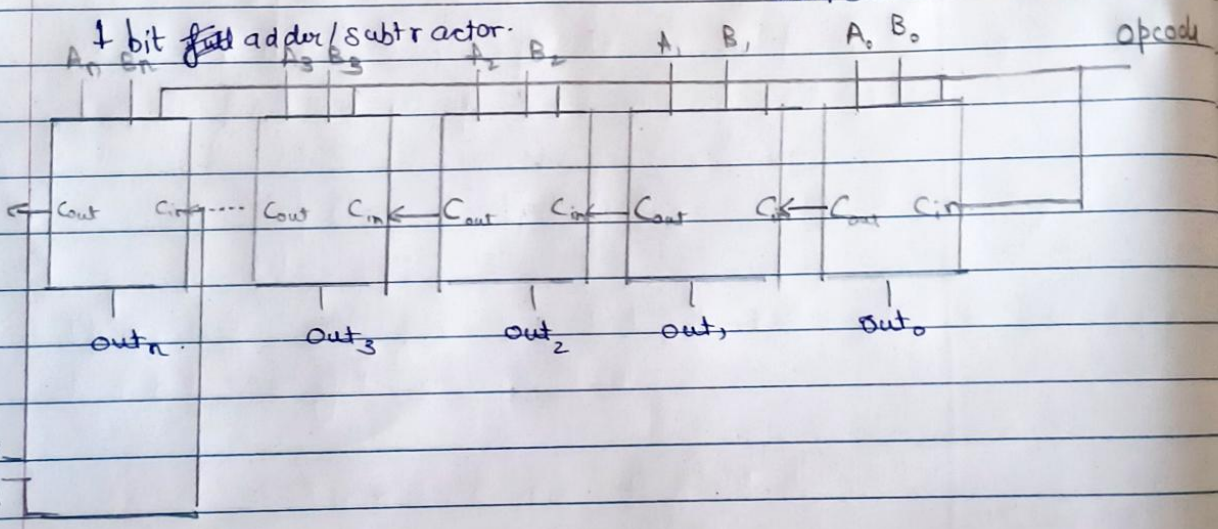
$$C_{in,0} = opcode.$$

③ whenever C_{in} for last bit is not equal to C_{out} for the last bit we get an overflow

$C_{in,n}$	$C_{out,n}$	overflow
0	0	0
1	1	0
0	1	1
1	0	1




A
B
opcode
C_i^0
cout
A
B ⊕ opcode
C_i^0
Sum/sub


1 bit adder/substractor.
A_n B_n
A_3 B_3
A_2 B_2
A_1 B_1
A_0 B_0
opcode
Cout
C_{in}
out_n
out_3
out_2
out_1
out_0
overflow