Anooj Pai
Homework 4

1. Simplify the following expressions using Boolean algebraic laws. Give each step of your simplification and denote which laws you're using for each step. Do not skip or combine steps!

(a) A ∗ (~A + B ∗ B) + ~(B + A) ∗ (~A + B)
=(A(~A + (B*B))) + (~B+~A * (~A+B)) (Idempotent Law)
=(A(~A+B)) + (~B+~A * (~A+B)) (Redundancy Law)
=(A*B) + (~B+~A * (~A+B)) (De Morgan's Theorem)
=(A*B) + (~B * ~A) (Absorption Law)
So A ∗ (~A + B ∗ B) + ~(B + A) ∗ (~A + B) simplifies to (A*B) + (~B * ~A)

(b) ~(C ∗ B) + (A ∗ B ∗ C) + ~A + ~C + ~B
= ~C + ~B + ABC + ~(A + C + ~B) (De Morgans Theorem)
= ~C + ~B + ABC + ~A*~C*~(~B) (De Morgans Theorem)
= ~C + ~B + ABC + ~A * ~C * B (Involution Law)
= ~C + ~B + ABC (Absorption Law)
using the Absorption Law by reducing a complicated expression to a simpler one by absorbing the like terms (AB + $\bar{A}$ = B + $\bar{A}$)
= ~C + ~B + AB (Absorption Law)
= ~C + ~B + A (Absorption Law)
So ~(C ∗ B) + (A ∗ B ∗ C) + ~A + ~C + ~B simplifies to ~C + ~B + A

(c) (A + B) ∗ (~A + C) ∗ (~C + B) =
(C*B*B)+(B*B*~A)+(A*B*C)+(A*B*~A)+(A*C*~C)+(A*~A*~C)+(B*C*~C)+(B*~A*~C)) (Rewrite)
= (C*B)+(B*B*~A)+(A*B*C)+(A*B*~A)+(A*C*~C)+(A*~A*~C)+(B*C*~C)+(B*~A*~C))
(Idempotent Law)
= (C*B)+(B*~A)+(A*B*C)+(A*B*~A)+(A*C*~C)+(A*~A*~C)+(B*C*~C)+(B*~A*~C))
(Idempotent Law)
= (C*B)+(B*~A)+(A*B*C)+(A*~A*B)+(A*C*~C)+(A*~A*~C)+(B*C*~C)+(B*~A*~C))
(Commutative Law)
= (C*B)+(B*~A)+(A*B*C)+(0 * B)+(A*C*~C)+(A*~A*~C)+(B*C*~C)+(B*~A*~C))
(complement law)
= (C*B)+(B*~A)+(A*B*C)+(0)+(A*C*~C)+(A*~A*~C)+(B*C*~C)+(B*~A*~C)) (Annulment Law)
= (C*B)+(B*~A)+(A*B*C)+(A*0)+(A*~A*~C)+(B*C*~C)+(B*~A*~C)) (complement law)
= (C*B)+(B*~A)+(A*B*C)+(0)+(A*~A*~C)+(B*C*~C)+(B*~A*~C)) (Annulment Law)
=(C*B)+(B*~A)+(A*B*C)+(0 * ~C)+(B*C*~C)+(B*~A*~C)) (complement law)
= (C*B)+(B*~A)+(A*B*C)+(0)+(B*C*~C)+(B*~A*~C)) (Annulment Law)
= (C*B)+(B*~A)+(A*B*C)+(B*0)+(B*~A*~C)) (complement law)
= (C*B)+(B*~A)+(A*B*C)+(0)+(B*~A*~C)) (Annulment Law)
= (C*B)+(A*B*C)+(B*~A)+(B*~A*~C)) (Commutative Law)

= (C*B)+(C*B*A)+(B*~A)+(B*~A*~C)) (Commutative Law)
= (C*B)+(B*~A)+(B*~A*~C)) (Absorption Law)
= (C*B)+(B*~A) (Absorption Law)
= B * (C + ~A) (Rewrite)
So (A + B) ∗ (~A + C) ∗ (~C + B) simplifies to B * (C + ~A)

2. Find all solutions of the following Boolean equations without using the truth tables:
(a) (~A + C) ∗ (~B + D + A) ∗ (D + A ∗ ~C) ∗ (~D + A) = 1
= (~A + C) ∗ (~B + D + A) ∗ (D + ~(~A ∗ ~(~C)) ∗ (~D + A) (De Morgans Theorem)
= (~A + C) ∗ (~B + D + A) ∗ (D + ~(~A ∗ C)) ∗ (~D + A) (Double Negation)
= (~A + C) ∗ (D + ~(~A ∗ C)) ∗ (~B + D + A) ∗ (~D + A) (Commutative Law)
= (~A + C) ∗ (~(~A ∗ C) + D) ∗ (~B + D + A) ∗ (~D + A) (Commutative Law)
= (~A + C) ∗ D ∗ (~B + D + A) ∗ (~D + A) (Redundancy Law)
= (~A + C) ∗ D ∗ (D + ~B + A) ∗ (~D + A) (Commutative Law)
= (~A + C) ∗ D ∗ (~D + A) (Absorption Law)
= (~A + C) ∗ (D ∗ A) (Redundancy Law)
= A ∗ (~A + C) ∗ D (Commutative Law)
= A ∗ C ∗ D (Redundancy Law)

Now since we know it simplifies down to A ∗ C ∗ D we just need to find when that would be true. This is simple because the ∗ just means and so to be true A, C, and D all need to be true.

(b) (((~K ∗ L ∗ N) ∗ (L + M)) + ((~K + L + N) ∗ (K ∗ ~L ∗ ~M))) ∗ (~K + ~N) = 1
= ((~K ∗ L ∗ (L + M) ∗ N) + ((~K + L + N) ∗ (K ∗ ~L ∗ ~M))) ∗ (~K + ~N)
(Commutative Law)
= ((~K ∗ L ∗ N) + ((~K + L + N) ∗ (K ∗ ~L ∗ ~M))) ∗ (~K + ~N) (Absorption Law)
= ((~K ∗ L ∗ N) + (K ∗ (~K + L + N)∗ ~L ∗ ~M)) ∗ (~K + ~N) (Commutative Law)
= ((~K ∗ L ∗ N) + (K ∗ (L + N)∗ ~L ∗ ~M)) ∗ (~K + ~N) (Redundancy Law)
= ((~K ∗ L ∗ N) + (K ∗ ~L ∗ (L + N) ∗ ~M)) ∗ (~K + ~N) (Commutative Law)
= ((~K ∗ L ∗ N) + (K ∗ (L + N) ∗ ~M)) ∗ (~K + ~N) (Redundancy Law)
= (N ∗ ((L + ~K) + (K ∗ ~L ∗ ~M)) ∗ (K ∗ ~N)) (Rewrite)
= (N ∗ ((~K + ~N) ∗ ((L * ~K) + (K ∗ ~L ∗ ~M))) (Commutative Law)
= (N ∗ ((~N + ~K) ∗ ((L * ~K) + (K ∗ ~L ∗ ~M))) (Commutative Law)
= (N ∗ ~K) ∗ ((L * ~K) + (K ∗ ~L ∗ ~M)) (Redundancy Law)
= ((L ∗ N ∗ ~K ∗ ~K) + (K ∗ N ∗ ~K ∗ ~L ∗ ~M)) (Rewrite)
= ((L ∗ N ∗ ~K) + (K ∗ N ∗ ~K ∗ ~L ∗ ~M)) (Idempotent Law)
= ((L ∗ N ∗ ~K) + (K ∗ ~K ∗ N ∗ ~L ∗ ~M)) (Commutative Law)
= ((L ∗ N ∗ ~K) + (0 ∗ N ∗ ~L ∗ ~M)) (Complement Law)
= ((L ∗ N ∗ ~K) + (0 ∗ ~L ∗ ~M)) (Annulment Law)
= ((L ∗ N ∗ ~K) + (0 ∗ ~M)) (Annulment Law)
= ((L ∗ N ∗ ~K) + (0) (Annulment Law)
= (L ∗ N ∗ ~K) (Identity Law)

Since the expression simplifies down to L * N * ~K we just need to find when that would be true. This is simple because the * is and, and the ~ is not, so L and N must be true while K is false.

3) Simplify the following expression by first constructing a truth table, using that truth table to construct a K-map, and then using that K-map to simplify.
Q = ~X * ~Y * Z + X * Y * ~Z + ~X * Y * ~Z + X * ~Y * ~Z

Truth Table

x	y	z	~X * ~Y * Z + X * Y * ~Z + ~X * Y * ~Z + X * ~Y * ~Z
True	True	True	False
True	True	False	True
True	False	True	False
True	False	False	True
False	True	True	False
False	True	False	True
False	False	True	True
False	False	False	False

K-map

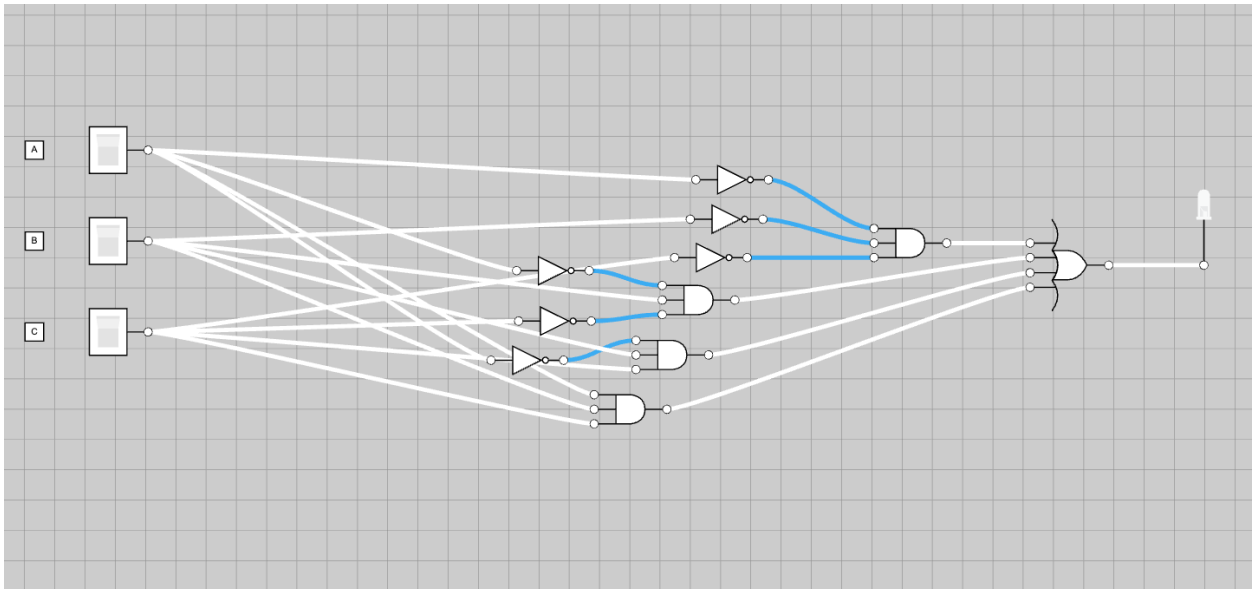
x/yz	00	01	11	10
0	0	1	0	1
1	1	1	0	1

Simplified:
y = B*C + A*C + ~(A*B*C)

4) Convert the following truth table into its sum of products representation:
A B C Output
0 0 0 1 (!A!B!C)
0 0 1 0

0 1 0 1 (!AB!C)
0 1 1 1 (!ABC)
1 0 0 0
1 0 1 0
1 1 0 0
1 1 1 1 (ABC)
y = (!A!B!C) + (!AB!C) + (!ABC) + (ABC)

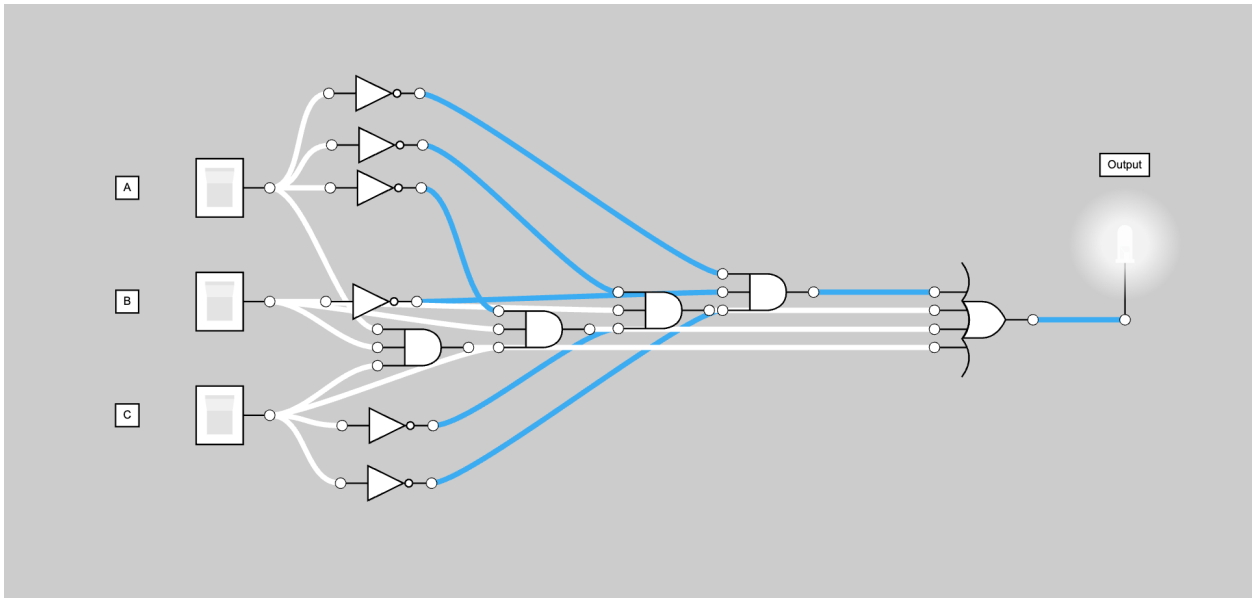
5) Submitted on submitty:



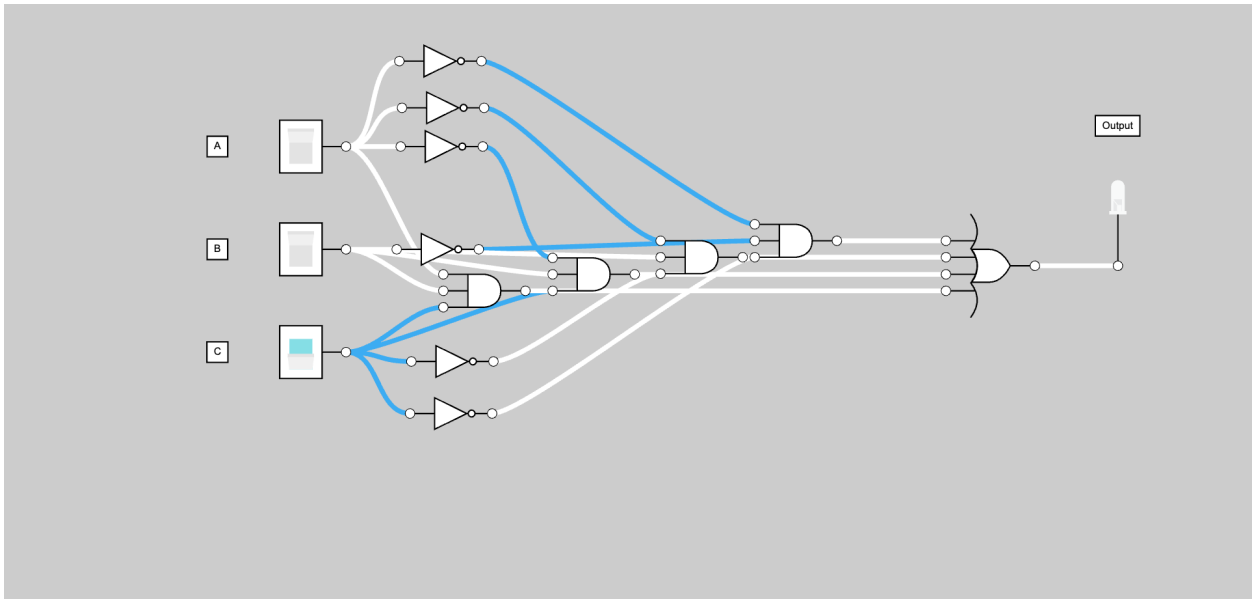
6)Test you circuit by supplying appropriate inputs and observing the expected values of the output. Explain why your set of tests is sufficient to prove that your logical circuit does in fact implement the required Boolean function. For each test, provide a picture (snapshot) of your circuit. Insert all such pictures in the hw4.pdf PDF file. You can download pictures (PNG, JPEG, or PDF) of your circuit diagram using OpenCircuits' "Export Image" feature.

My set of tests below are sufficient because I tested every row of the truth table and only the rows that the SOP function was true for lit up in the output. This proves that my equation and circut are correct.

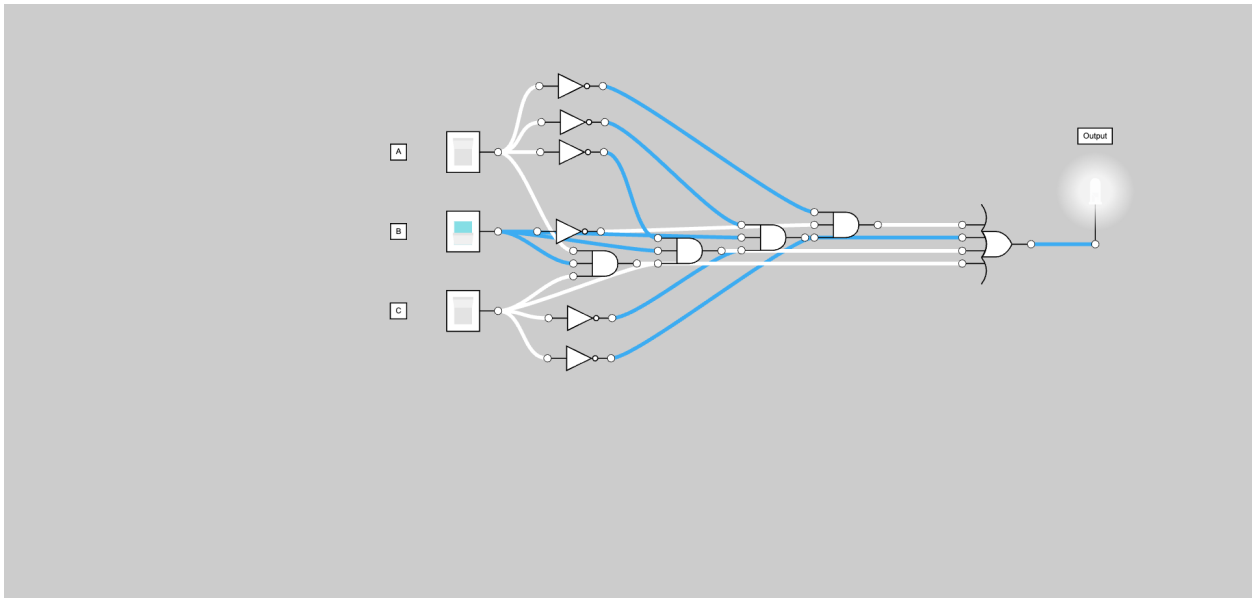
Row 1 of table, all inputs off and output is on as shown in the table above



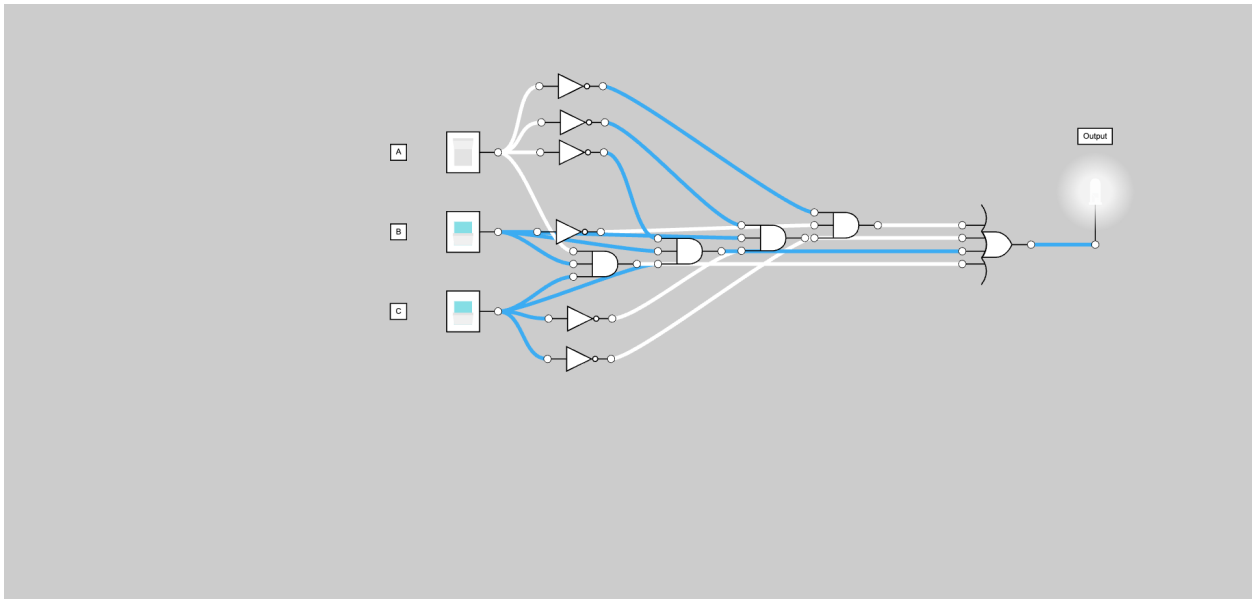
Row 2 of table, inputs A and B are off with C on and the output is off as shown in the table



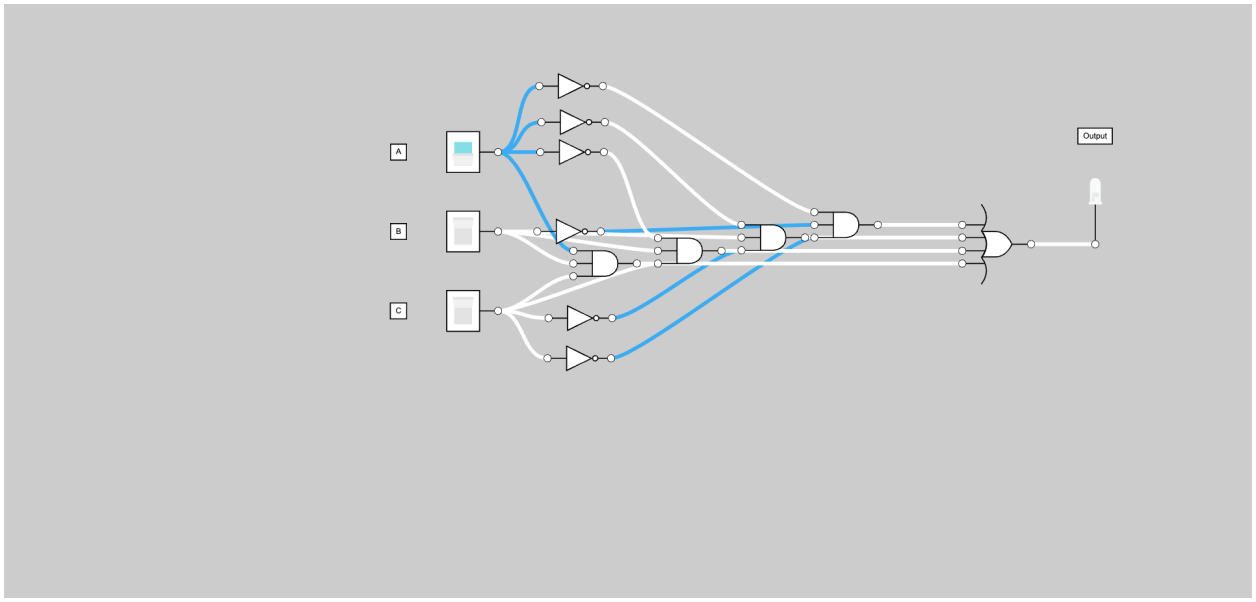
Row 3 of table, inputs A and C are off with B on and the output is on as shown in the table



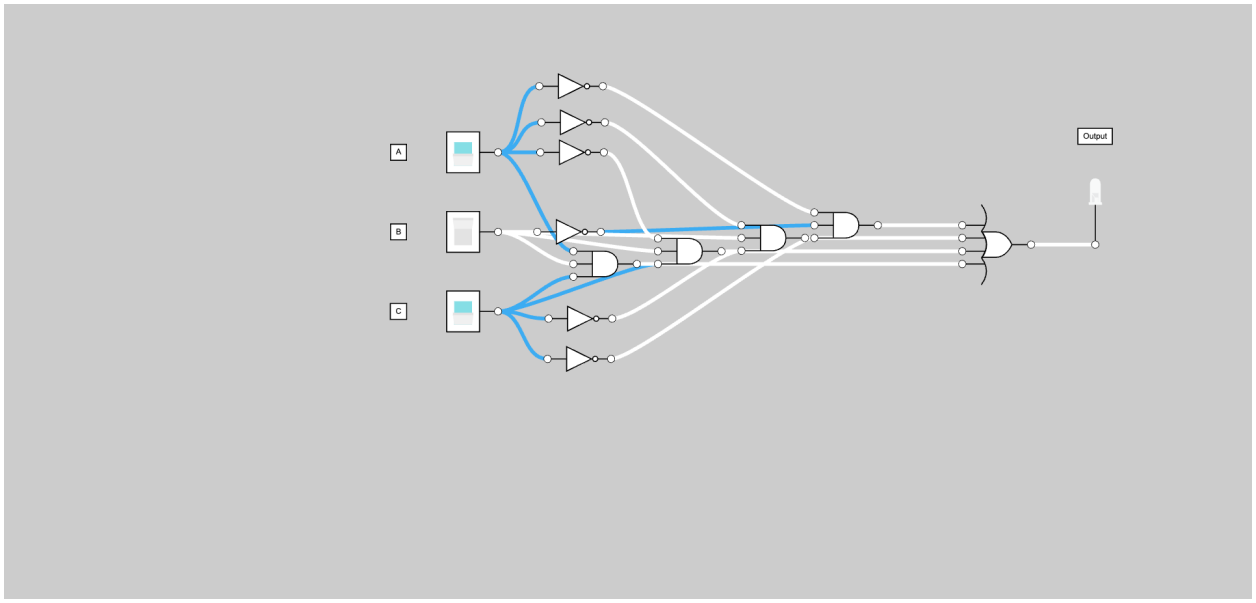
Row 4 of table, inputs B and C are on with A off and the output is on as shown in the table



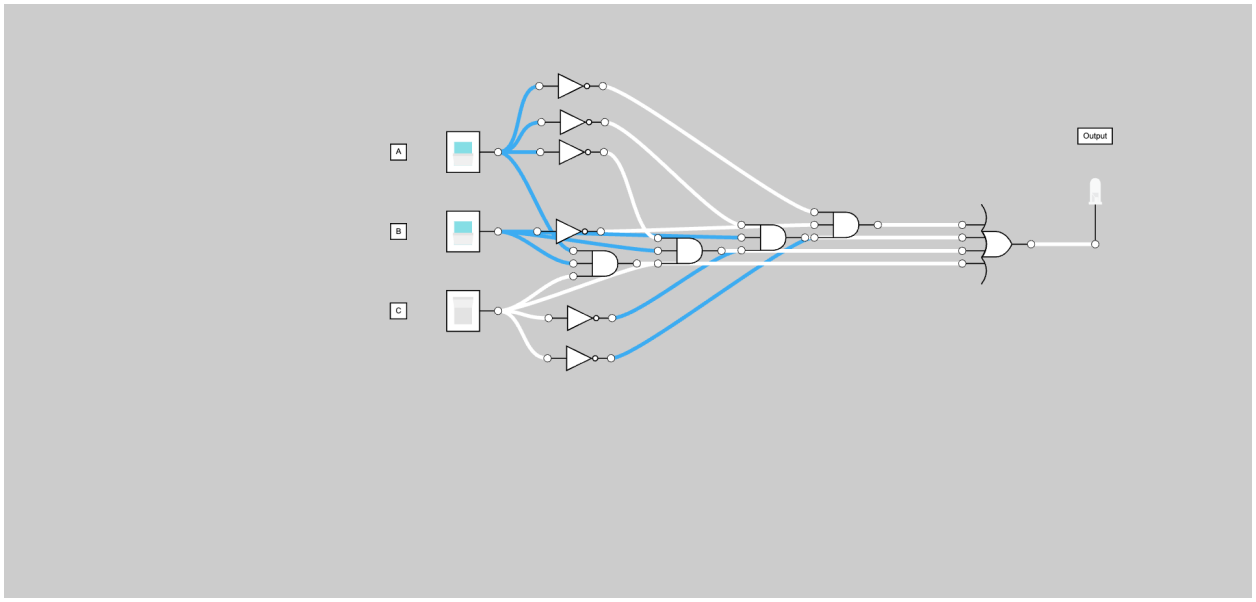
Row 5 of table, inputs B and C are off with A on and the output is off as shown in the table



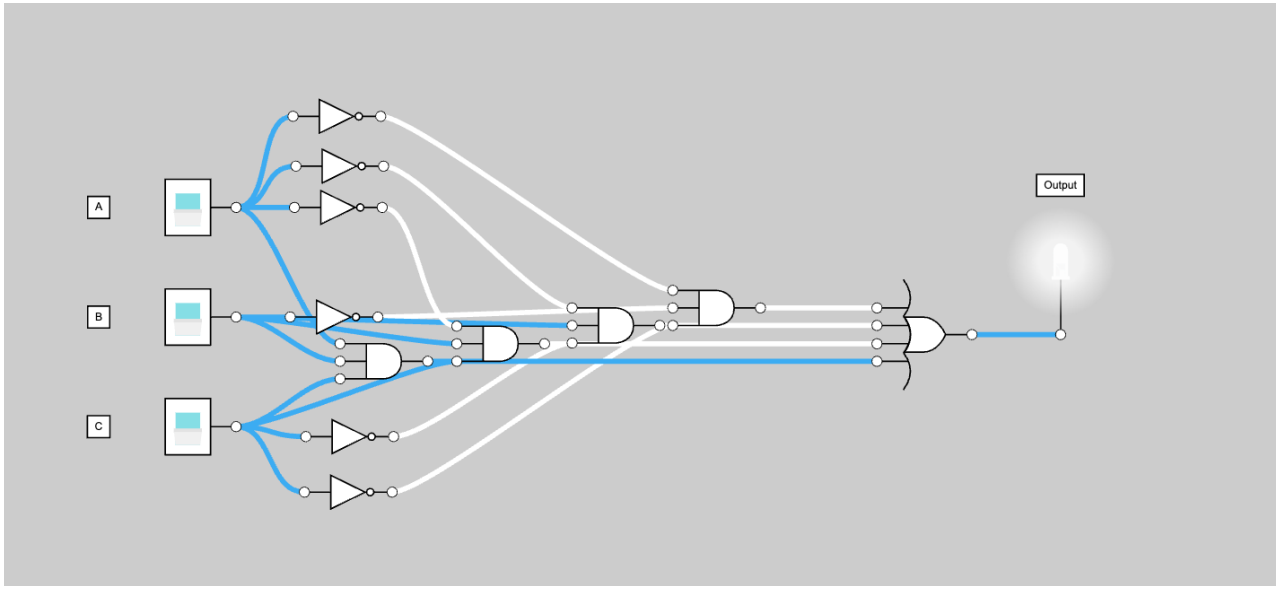
Row 6 of table, inputs A and C are on with B off and the output is off as shown in the table



Row 7 of table, inputs A and B are on with C off and the output is off as shown in the table

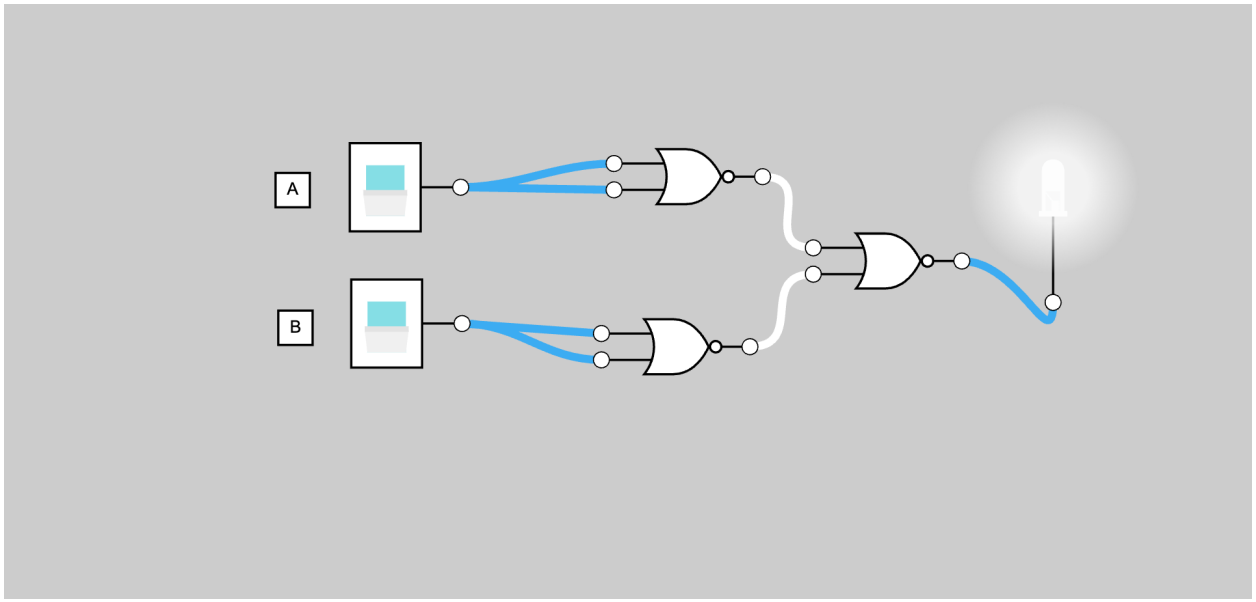


Row 8 of table, all inputs are on and the output is on as shown in the table

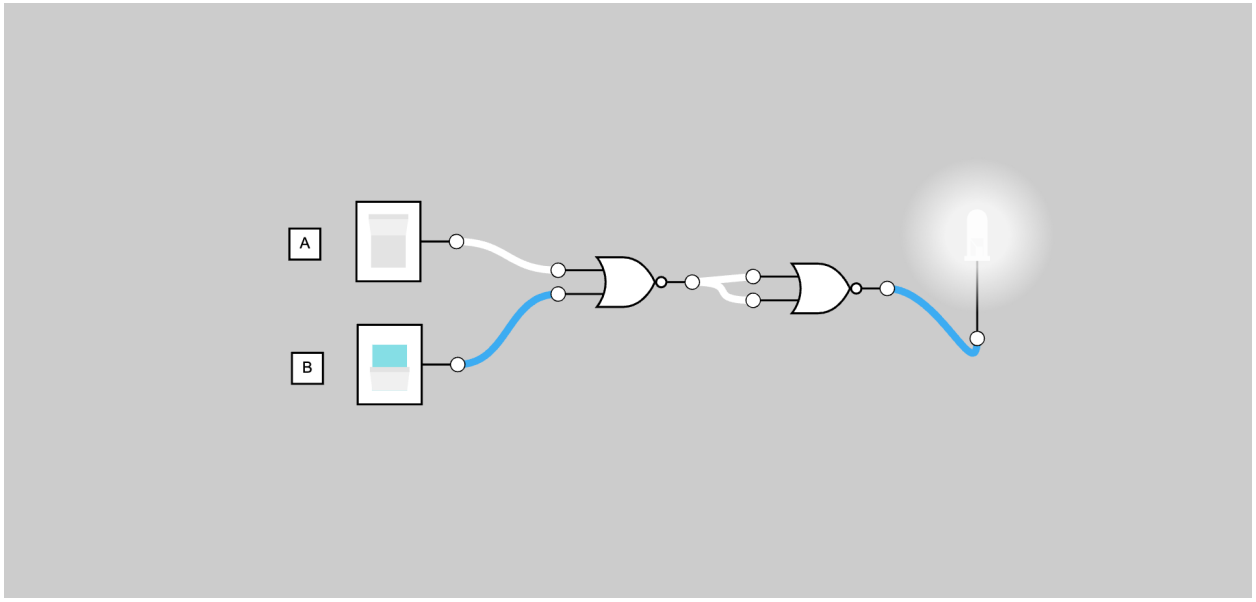


7) Given inputs A and B, show that NOR {(A + B)} is functionally complete by giving logical circuits equivalent to AND {(A ∗ B)}, OR {(A + B)}, and NOT {A} gates using only NOR gates in their construction.

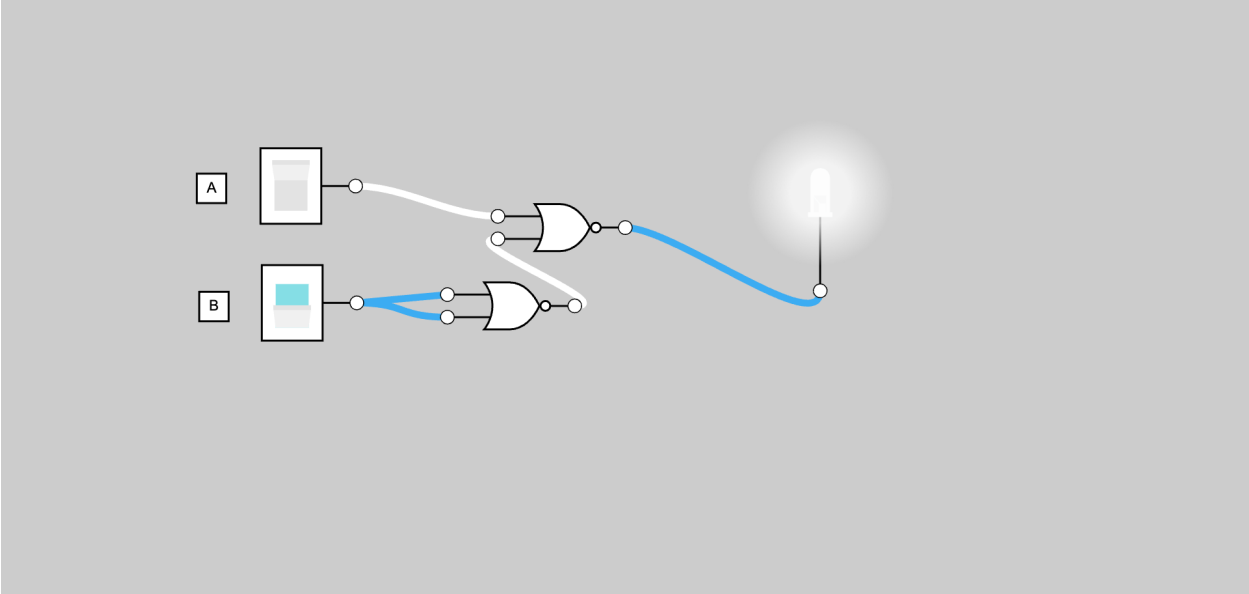
AND function using NOR:



OR function using NOR:



NOT A function using NOR:



8) What is 5ED4 - 07A4 when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

5 = 0101, E = 1110, D = 1101, 4 = 0100
0 = 0000, 7 = 0111, A = 1010, 4 = 0100
5ED4 = 0101 1110 1101 0100
07A4 = 0000 0111 1010 0100
5ED4 - 07A4 = 0101 1110 1101 0100 - 0000 0111 1010 0100 = 0101 0111 0011 0000
0101 = 5, 0111 = 7, 0011 = 3, 0000 = 0
0101 0111 0011 0000 = 5730
5ED4 - 07A4 = 5730

9) What is 5ED4 - 07A4 when these values represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.
$5ED4_{16}$ = 0101 1110 1101 0100
$07A4_{16}$ = 0000 0111 1010 0100
$5ED4_{16}$ - $07A4_{16}$ = 0101 1110 1101 0100 - 0000 0111 1010 0100
0101 = 5, 0111 = 7, 0011 = 3, 0000 = 0
0101 0111 0011 0000 = 5730_{16}

10) Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate 185 - 122. Is there overflow, underflow, or neither?
185 in unsigned 8-bit decimal integers = 10111001
122 in unsigned 8-bit decimal integers = 01111010
185-122 = 63
10111001 - 01111010 = 0111111

In range of 0 and 255 which means that there is neither underflow nor overflow

11) Assume 151 and 214 are signed 8-bit decimal integers stored in two's complement format. Calculate 151 - 214 using saturating arithmetic. The result should be written in decimal. Show your work.
151 in signed 8-bit decimal integer = 10010111
214 in signed 8-bit decimal integer = 11010110

2's compliment +1 value of 151 = 01101001
2's compliment +1 value of 214 = 00101010

01101001 - 00101010 = 00111111 (63)
151 - 214 = 63
in range of -128 127 for signed 8-bit decimal integers
63

12) Assume 151 and 214 are unsigned 8-bit integers. Calculate 151 + 214 using saturating arithmetic. The result should be written in decimal. Show your work.
151 in signed 8-bit decimal integer = 10010111
214 in signed 8-bit decimal integer = 11010110
151 + 214 = 255
10010111 + 11010110 = 11111111 (255)
255

13) What decimal number does the bit pattern 0×0C000000 represent if it is a two's complement integer? An unsigned integer?

0×0C000000 =
$0 * 16^7 + 12 * 16^6 + 0 * 16^5 + 0 * 16^4 + 0 * 16^3 + 0 * 16^3 + 0 * 16^2 + 0 * 16^1$
$= 0 + 12 * 16777216 + 0 + 0 + 0 + 0 + 0 + 0$
= 201,326,592

As an unsigned integer: 201,326,592
As a 2's complement integer: 201,326,592

14)If the bit pattern 0×0C000000 is placed into the Instruction Register, what MIPS instruction will be executed?
0×0C000000:
0 = 0000, C = 1100, 0 = 0000, 0 = 0000, 0 = 0000, 0 = 0000, 0 = 0000, 0 = 0000
In binary the bit pattern would be 00001100000000000000000000000000
We can get rid of the extra 0's at the end so it would be 000011
The opcode 000011 is for the jal function

jal 0 would be executed

15)Give a reason why we use two's complement representation for negative numbers in computer arithmetic. Give an example of its usage.

One of the reasons that we use two's compliment for negative numbers is because it is faster to subtract using it than true subtraction. This is because the two's compliment just checks the most important bit which is the first one and figures out whether it needs to add or subtract based on that. This is much faster than checking for the sign of the number then subtracting.

Example:
Lets represent -10 by using 2's complement:
10 in binary is 1010
10 after completing bits = 00001010
Then taking the complement = 0000 1010
From there you add 1, 1110 1110 + 1 = 1111 0110
The two's complement of -10 is 11110110_2