1. Simplify the following Boolean logic functions to the format in the sum of products first and then create a truth table in Excel for each as the verification reference of the circuit design, finally implement by online tools at https://circuitverse.org/simulator

ANS:

a. f= (A + C' + D') (B' + C' +D) (A + B' + C")

= (A+ C' + D'B' + C' + DIA+ B' + C")

= (A+C" + D'(B' + C" + DYA+ B' + C')

= A(B' + C" + D)(A + B' + C') + C'(B' +C" + D)(A + B' + C') + D'(B' +'+ D)(A + B' + C")

Expanding further:

f = AB'A+ AB'B' + AB'CI + AC'A + AC'B' + AC'C' + AD'A+ AD'B' + AD'C" +C' B'A + C'B'B' + C'B'C + C'C'A+C'C' B' + C'C'C + C'D'A+ C' D'B' + C' D'C' +D'B'A + D'B'B' + D'B'C + D'C' A+ D'C' B'+ D'C'C" + D'D'A+ D'D'B' + D'D'C

f= AB'A+ AB'C'+ AC'A + ACB'+ AD'A + AD'B' + AD'CI + C'B'A+CB'C"+CCA+C'CB'+C'D'A+CD'B'+C'D'C+D'B'A+D'BC+D'CA+DCB'+D'D'A+D'D'B'

For example: AB'A + AB'C' = A(B' + C')

Similarly: AC' A + AC" B'

= AC' + A

b. f-(z+x)(z'+y')(y'+x) = {(z+ x) (x+y')) (y'z')

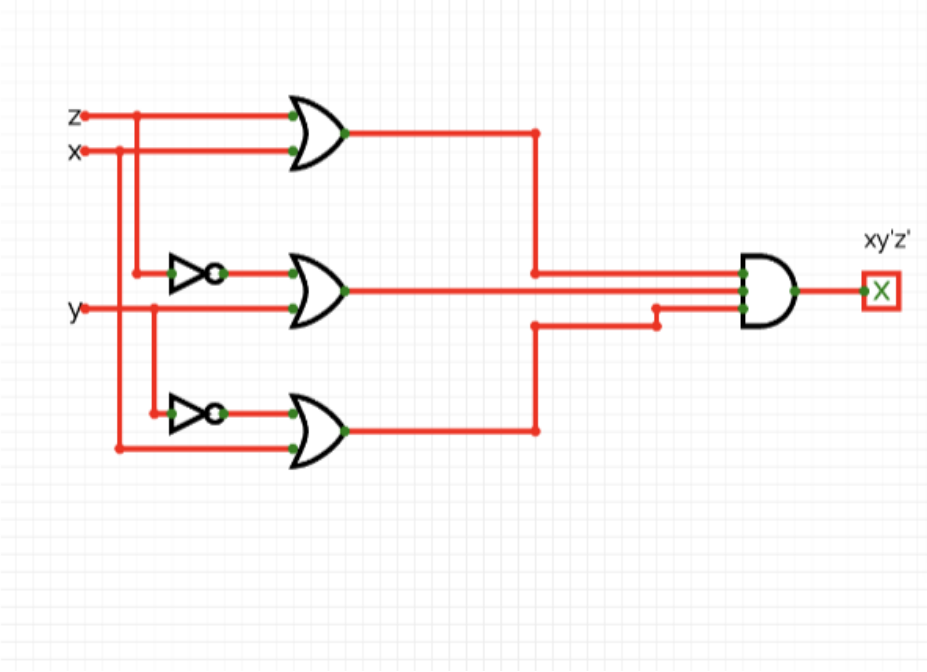
= (x + y'z) (y'z') [Distributive law: (p+q) (ptr)-ptqr]

=xy'z' + y'y'zi

=xyZ+y.O [y'.y'=y', z.z=0]

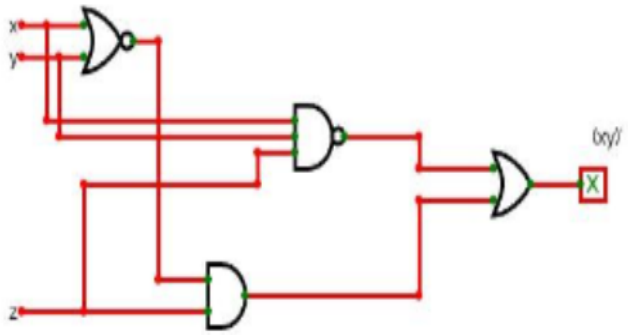
=xyZ+0 =Xy'z

x	y	z	y’	z’	xy’z’
0	0	0	1	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	0	0	0	1	0
1	1	1	0	0	0



c.f=(x+y)'z+x'y'z' (x'y')z+ x'y'z'
=X'y' (z+ z')
=x'y'.1
= x°y=(xy)”

x	y	x’	y’	x’y’
0	0	0	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0



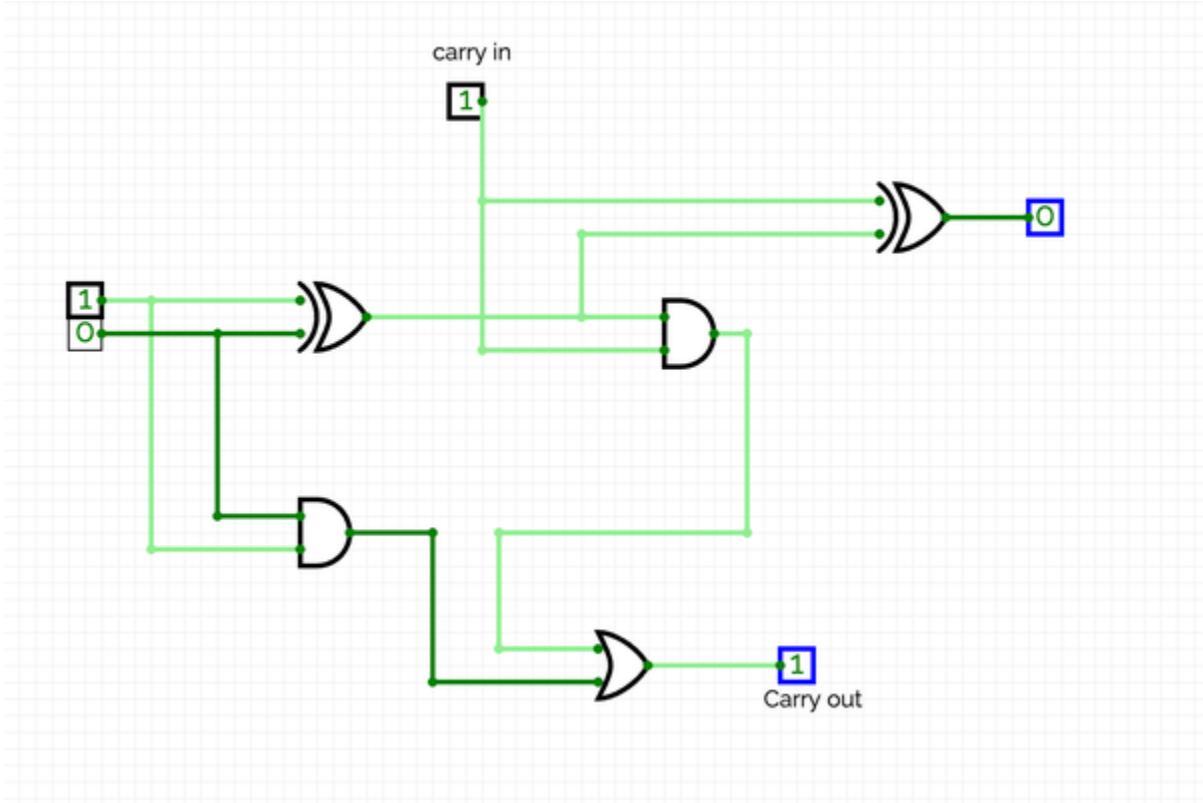
2. Design signal bit full adder based on the truth table and circuit on chapter 3 lecture handouts first and then create a 4-bit "Adder-Subtractor circuit" as follows to implement arithmetic addition and subtraction operations. - If S = 0 in the circuit, then $X3$ = $B3$, $X2$ = $B2$, $X1$ = $B1$ and $X0$ = $B0$, so the adder circuit simply adds A and B when $C0$ = S = 0 (carry in = 0). - If S = 1, then $X3$ = $B3$, $X2$ = $B2$, $X2$ =

$B2$ and $X0$ = $B0$. Since $C0$ = S = 1, the circuit is equivalent to adding the 2's complement of B to A, that is, implementing the subtraction operation "A-B".

ANS: Test the Design:
For the given test cases:
(a) A+B=7+ (-3) =4
A = 7 => 0111 (in binary)
B = -3, which is the 2's complement of 3
3 = 0011 in binary 2's complement of 3 = invert all bits and add 1 = 1101

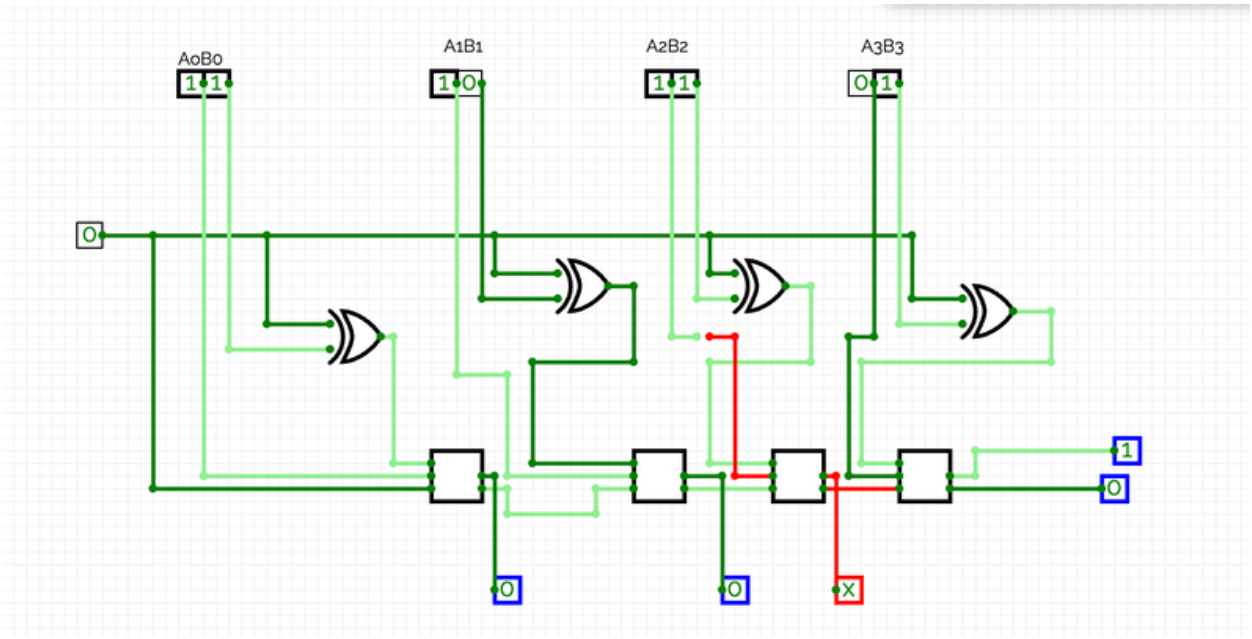
When S=0 (Addition), use:
A = 0111
B = 1101

Predicted outcome = 1000, or 8 in binary. The result is negative, or -8, as indicated by the sign bit. The absolute value can be determined by taking the 2's complement of 1000, which is 8 in decimal notation.



(b) A -B =-6 - (-1) =-5
A = -6 => 1010 (2's Complement of 6)
B = -1 => 1111 in binary (2's complement of 1)

While S=1 (Subtraction), use:
A = 1010
B = 1111
In this case, the predicted result is 1001, or -7 in decimal notation. The sign bit, which is found at the leftmost place of the integer, determines how results in 2's complement arithmetic should be interpreted. If the sign bit is 1, which denotes a negative number, the result must be taken as the 2's complement to be converted to its absolute value.



3. The following circuit is to simulate the MARIE assembly instruction "load 003", which means that it moves this instruction saved in the certain location of the memory to the instruction register (IR) using 16 D-Flip Flops within CPU. Based on the above circuit, please implement the following MARIE assembly instructions by translating assembly code to machine code depending on the given lookup table in chapter 4 lecture handouts Load 104 Add 105 Store 106

Halt Notes: - The design should be exported as ".cv" file by "Export as file" in "Project" tab on the toolbar and submitted with the truth table.

ANS:
"Load X" has the instruction 0001, where X is the location to load from.
The address to add from is represented by X in the "Add X" instruction, which is 0011.
"Store X" has the instruction 0010, where X is the address to be stored.
Given the guidelines for assembly:
In light of the assembly guidelines, load 104, add 105, and store 106.
These can be converted to machine code as follows:
What load 104 means is:
Binary code: 0001 0000 1000
Translating Hex: 1 08 Add 105 to
Binary: 1001-0001-0000
Translating Hex: 3 09 Store 106 to
0010 0000 1010 in binary
Hex: 0A-2
Thus, 1 08 3 09 2 0A is the machine code for the assembly instructions provided.

Which operation is performed depends on the binary representation of the instruction code. Based on the supplied table, we will now represent these instructions in a 4-bit format, using the remaining 12 bits for the address.

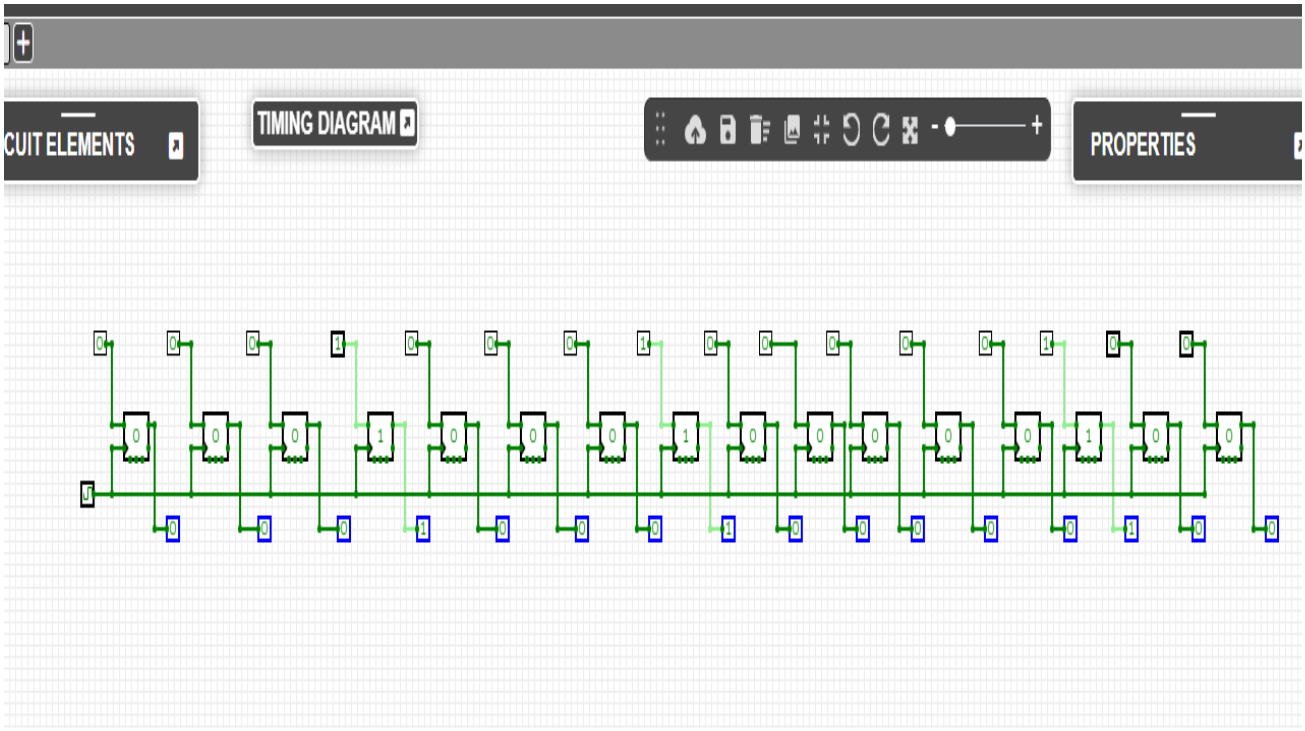
Instruction	4-bit code	Address	Machine code
Load	0001	0000 1000	0001 0000 1000
Add	0011	0000 1001	0011 0000 1001
Share	0010	0000 1010	0010 0000 1010

As an example, let's now create a basic circuit for the Load instruction. When the 4-bit code is 0001, the output will be active (1) for a "Load" action. This is a basic illustration: A, B, C, and D are the inputs (which represent the 4-bit code in reverse, with D being the most important bit). When the input is 0001, the output is Y.

D	C	B	A	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0

Regarding the loading process: Y = D' × C' × B' ×A

Based on the unique 4-bit codes for the "Add" and "Store" operations, construct analogous circuits. Let's now construct this circuit: Apply NOT gates to the D, C, and B inputs. Combine the outputs of these NOT gates with input A using an AND gate. The output of this circuit will be 1 only if the 4-bit code is the same as the load operation. Add and Store would have similar designs, but depending on each input's unique 4-bit code, you could change which inputs have NOT gates and how they are combined.


CUIT ELEMENTS
TIMING DIAGRAM
PROPERTIES