#Solution-1:

a. f = $(A + C' + D')(B' + C' + D)(A + B' + C')$
= =(A+ C' + D')(B' + C' + D)(A+ B' + C')
= (A+C" + D')(B' + C" + D)(A+ B' + C')
= A(B' + C" + D)(A + B' + C') + C'(B' +C" + D)(A + B' + C') + D'(B' + ' + D)(A + B' + C')
f= AB'A+ AB'B' + AB'C' + 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 + AC' B' + AD'A + AD'B' + AD'C' + C' B'A+ C'B'C" + C'C'A + C'C'B' + C'D'A + C'D'B' + C'D'C' + D'B'A + D'B'C' + D'C' A + D'C' B' + D'D'A + D'D' B'

Here, AB'A + AB'C' => A(B' + C')
Likewise: AC' A + AC" B'
= AC' + A

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

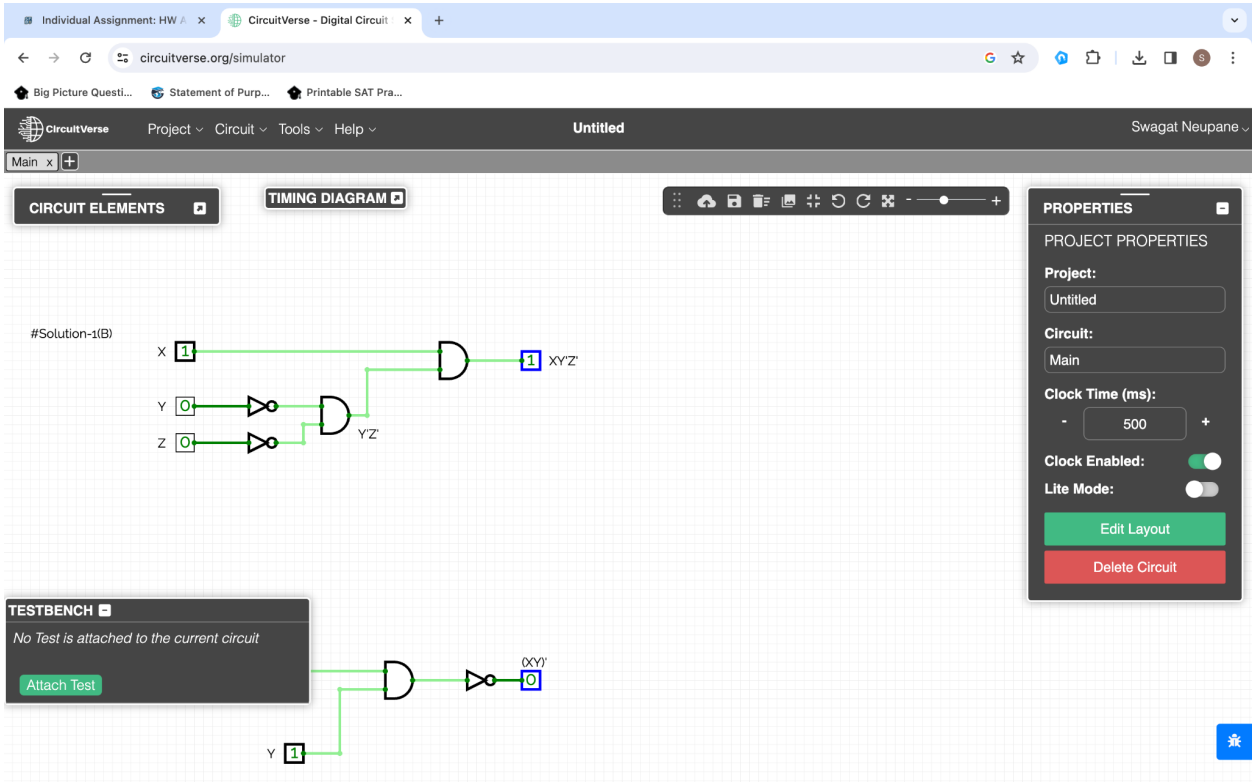
= {(z + x)(x+y')}(y'z') (Using Demorgan's law)
= (x + y'z)(y'z') (Using Distributive law)
= xy'z' + y'y'zz'
= xy'z' + y'.0 (y'.y' = y' , z.z'=0)
= xy'z' + 0
= xy'z'

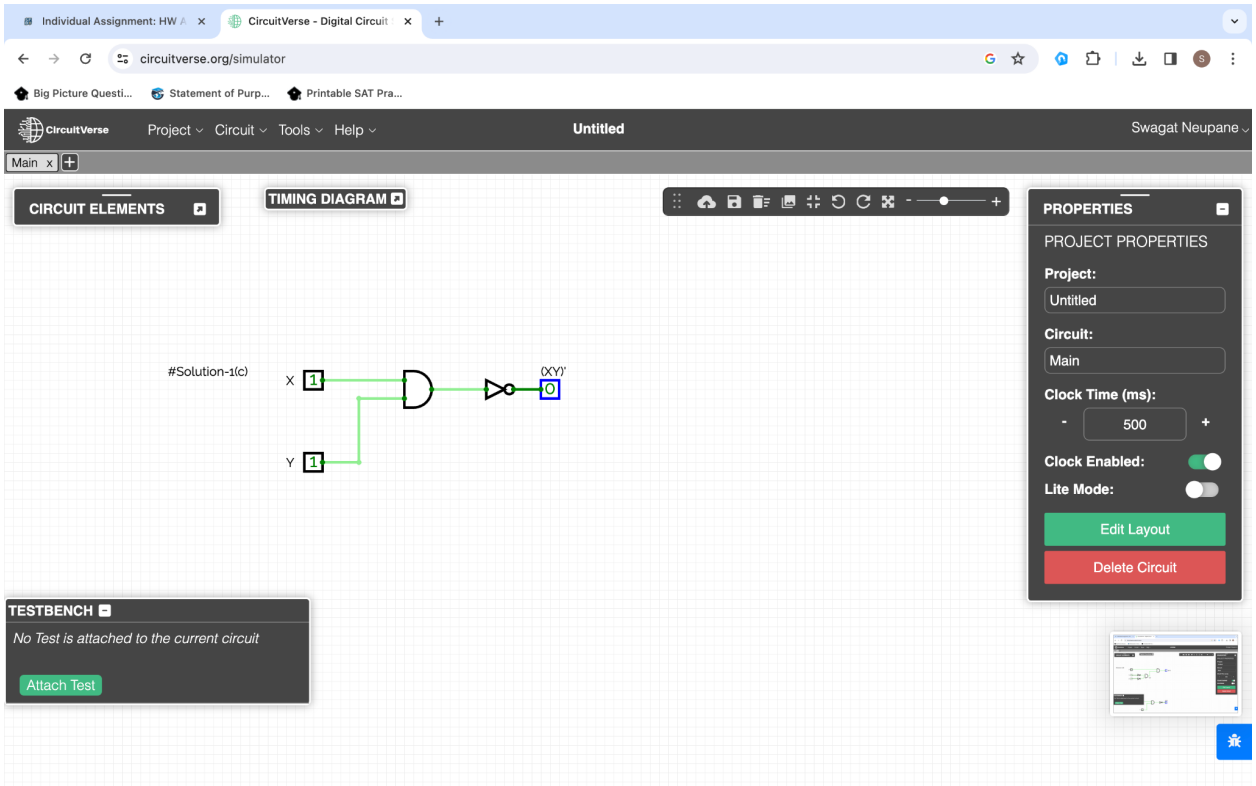
x	y	z	y'	z'	xy'z'
0	0	0	1	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	0	1	0
1	1	1	0	0	0

c. f = (x + y)'z + x'y'z'

=(x'y')z + x'y'z' (Using Demorgan's law)
= x'y'(z + z')
= x'y'.1 (Using Identity law)
= x'y'
= (xy)'

x	y	x'	y'	x'y'
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0





#Solution-2:

The 4-bit adder-subtractor design is based on the utilization of four full adders connected in a cascading fashion. This design can perform both addition and subtraction operations, taking advantage of the fact that subtraction can be achieved by adding the 2's complement of a number.

For addition (S=0): In this mode, the bits from the B number are fed directly into the adder. The carry-in of the least significant bit (C_0) is set to 0. This configuration allows the adder to simply add A and B together, providing the sum.

For subtraction (S=1): In this mode, the B input is inverted, obtaining the 1's complement of B. Additionally, a carry-in of 1 is introduced into the least significant bit adder to transform B into its 2's complement. The adder then proceeds to add A to the 2's complement of B. This operation is equivalent to subtracting B from A, as per the properties of 2's complement arithmetic.

Test the Design:

(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

Now, let's invert all bits and add 1 to 2's complement which is 1101

While S=0 (Addition), we can use:

A = 0111

B = 1101

In this case, the expected outcome is 1000 in binary representation, equivalent to 8 in decimal. The presence of the sign bit signifies a negative result, indicating -8. To obtain the absolute value, the 2's complement of 1000 is calculated, resulting in 1000, which translates back to 8 in the decimal system.

(b) A - B = -6 - (-1) = -5

A = -6 => 1010 (2's Complement of 6)

B = -1 => 1111 in binary (2's complement of 1)

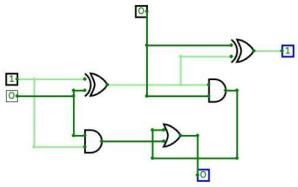
While S=1 (Subtraction), use:

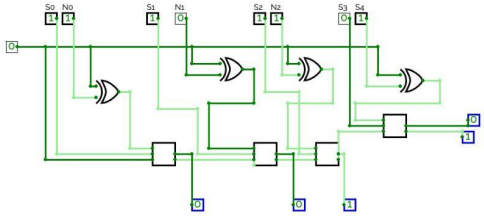
A = 1010

B = 1111

Here, the expected result = 1001, which is -7 in decimal.

In 2's complement arithmetic, the interpretation of results depends on the sign bit, located at the leftmost position of the number. If the sign bit is 1, indicating a negative number, the result needs to be converted to its absolute value by taking the 2's complement.





#Solution-3)

- The instruction for "Load X" is represented by the binary code 0001, where the value of X specifies the address from which the data should be loaded.

- The instruction for "Add X" is denoted by the binary code 0011, where the value of X specifies the address from which data should be added to the current operation.

- The instruction for "Store X" is indicated by the binary code 0010, with X representing the address where data should be stored.

- Load 104
- Add 105
- Store 106

Load 104 translates to:

- Binary: 0001 0000 1000
- Hex: 1 08
- Add 105 translates to:
- Binary: 0011 0000 1001
- Hex: 3 09
- Store 106 translates to:
- Binary: 0010 0000 1010
- Hex: 2 0A

- So, the machine code for the given assembly instructions are: 1 08 3 09 2 0A

The operation to be executed is determined by the binary representation of the instruction code. In this scenario, we will represent these instructions using a 4-bit format, following the guidance outlined in the provided table, and reserve the remaining 12 bits for specifying the memory 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

We will now create a basic circuit to illustrate the "Load" instruction. When performing a "Load" operation, the output should be active (set to 1) if the 4-bit code is 0001. To clarify, our inputs will be labeled as A, B, C, and D, representing the 4-bit code in reverse order, where D represents the most significant bit. The output, designated as Y, will be active when the input matches the code 0001.