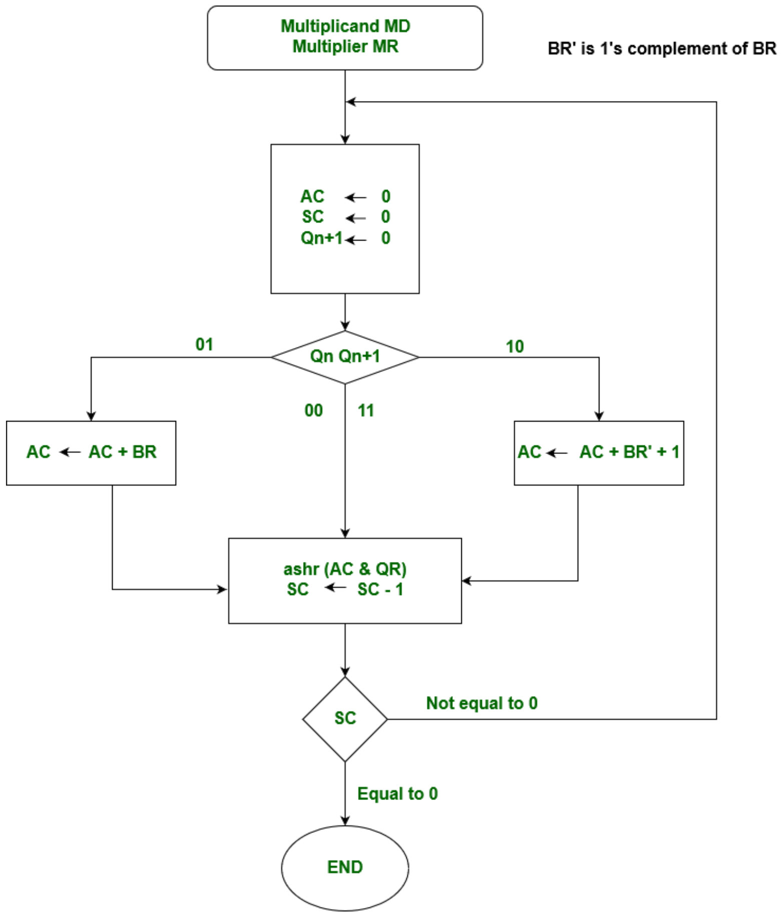
Computer Organization Booth's Implementation Project

Project Statement - Implement the Booth's algorithm for multiplying binary numbers

What is Booth's Algorithm?
Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation. It is very efficient for big Integers as less number of additions/subtractions are required.
As in all multiplication schemes, booth algorithms require examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

- The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
- The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier.
- The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

The flowchart of the above algorithm is



Reference - https://www.geeksforgeeks.org/computer-organization-booths-algorithm/

Prerequisites for running the program

- Python 3.3 must be present in the user's device

Description of Code

I have first made various functions that might be used again and again in the implementation of the algorithm

Functions in code:

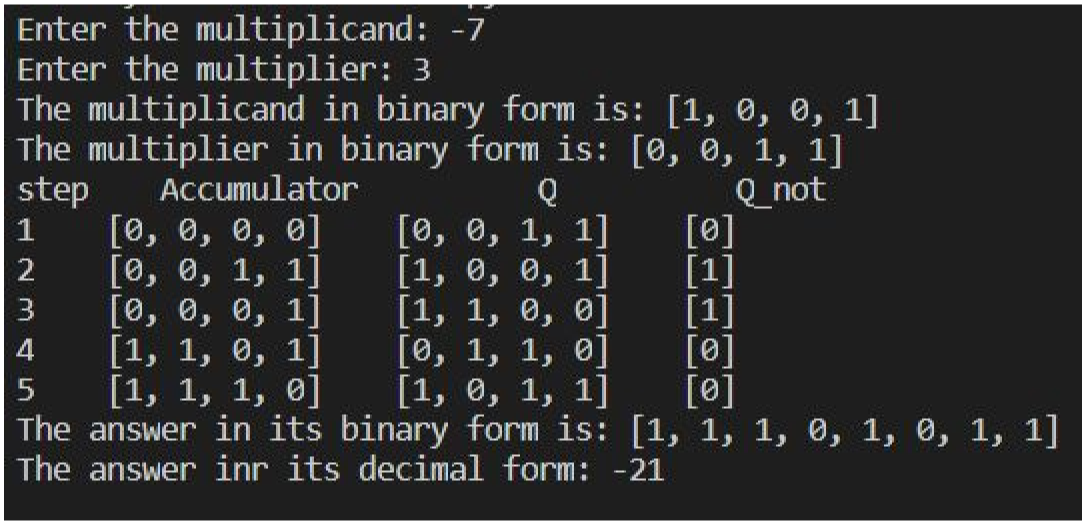
- binary(int n):In this function, I convert the integer in its binary form and store it in the form of a list. For positive numbers, I keep dividing them by two while they are greater than one and keep appending the remainder in a list. After the integer becomes less than or equal to 1, I append 1 to the list and return it in its reversed form. For negative numbers, I first find the binary form of their absolute value and then find the two's complement of the list I get.
- twos_compliment(list l):In this function, I first find the one's complement of the binary form of the number (stored in the form a list) and then add 1 to it. I return the resultant list.
- ones_compliment(list l): This function receives a list of 1's and 0's. It changes all 0's to 1 and all 1's to 0 and returns the list.
- add(list a, list b): In this function, I iterate on both the list starting with the last element and towards the first. I add them and store the result is another list taking care of the overflow. In the end, I return the resultant list.
- make_n_bits(list l, int n): This function returns the list making it of the required size (that is n) by adding zeros in front of positive numbers and ones in front of negative numbers.
- decimal(list l): This function converts the binary number (in the form of a list) into its decimal form. It returns the decimal number.
- right_shift(list a, list q, list q_not): It is required while implementing the algorithm to do a right shift in among the accumulator, q and q_not

Implementing the algorithm:

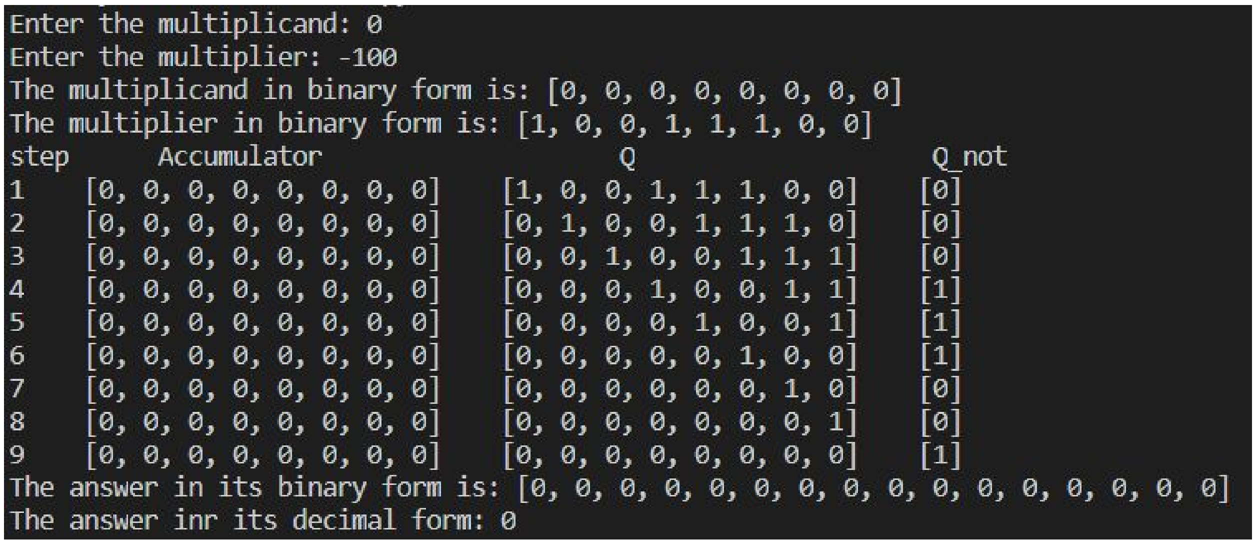
To keep the code independent of the number of registers, I calculated 'n' as the maximum of the length of multiplier and multiplicand. Then I initialize an accumulator as a list 'a' of size n with zeros. I put the multiplicand in list 'm' and multiplier in list 'q' (using function binary()), and I make sure both of these lists have size n by calling the function make_n_bits(). I initialize 'q_not' with zero.

After doing all this, I start a while loop till n is not equal to zero. If the last element of q is equal to q_not, I call the function right_shift(a,q,q_not) to perform the right shift. If the last element of q is 1 and that of q_not is 0, I subtract m from the accumulator. I do this by first finding twos_compliment() of m and then adding it to a. After that, I again call right_shift(a,q,q_not) to perform the right shift. If the last element of q is 0 and that of q_not is 1, I add m to the accumulator. I do this by calling the function add(a,m). I then call right_shift(a,q,q_not) which performs the right shift. These steps are repeated n times after which the answer can be formed by appending the accumulator list to list q. Finally, I find the decimal form of the answer and print it.

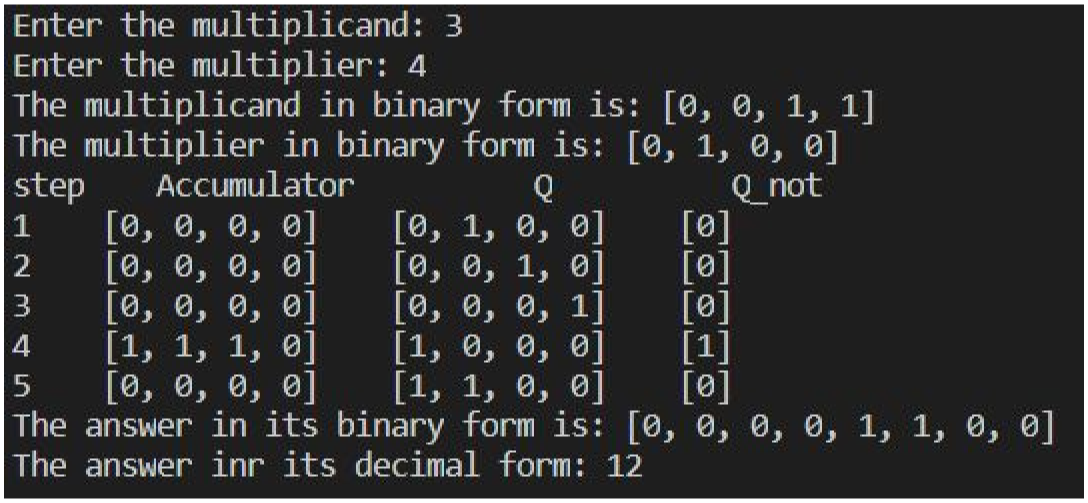
Sample Cases:



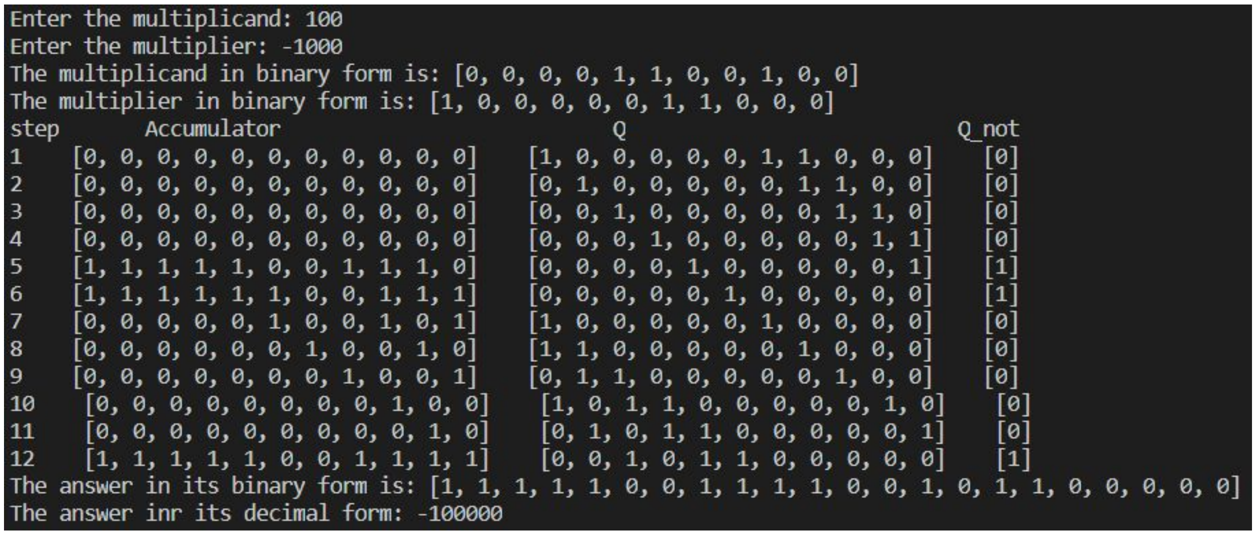
```
Enter the multiplicand: -7
Enter the multiplier: 3
The multiplicand in binary form is: [1, 0, 0, 1]
The multiplier in binary form is: [0, 0, 1, 1]
step    Accumulator         Q          Q_not
1    [0, 0, 0, 0]    [0, 0, 1, 1]    [0]
2    [0, 0, 1, 1]    [1, 0, 0, 1]    [1]
3    [0, 0, 0, 1]    [1, 1, 0, 0]    [1]
4    [1, 1, 0, 1]    [0, 1, 1, 0]    [0]
5    [1, 1, 1, 0]    [1, 0, 1, 1]    [0]
The answer in its binary form is: [1, 1, 1, 0, 1, 0, 1, 1]
The answer inr its decimal form: -21
```



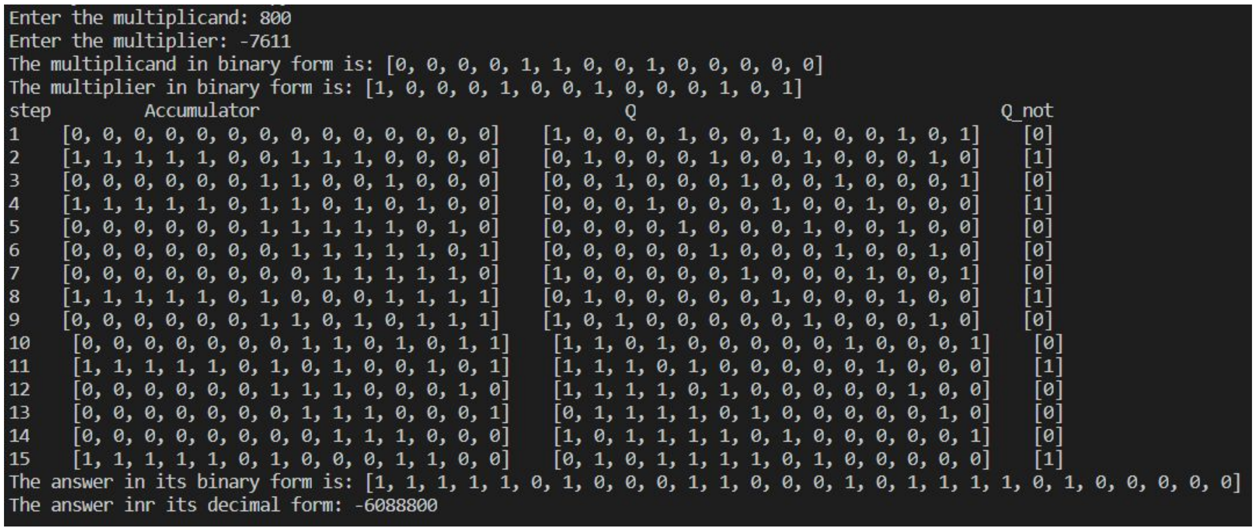
```
Enter the multiplicand: 0
Enter the multiplier: -100
The multiplicand in binary form is: [0, 0, 0, 0, 0, 0, 0, 0]
The multiplier in binary form is: [1, 0, 0, 1, 1, 1, 0, 0]
step      Accumulator                       Q                       Q_not
1    [0, 0, 0, 0, 0, 0, 0, 0]    [1, 0, 0, 1, 1, 1, 0, 0]    [0]
2    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 1, 0, 0, 1, 1, 1, 0]    [0]
3    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 1, 0, 0, 1, 1, 1]    [0]
4    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 0, 1, 0, 0, 1, 1]    [1]
5    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 0, 0, 1, 0, 0, 1]    [1]
6    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 0, 0, 0, 1, 0, 0]    [1]
7    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 0, 0, 0, 0, 1, 0]    [0]
8    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 0, 0, 0, 0, 0, 1]    [0]
9    [0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 0, 0, 0, 0, 0, 0]    [1]
The answer in its binary form is: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
The answer inr its decimal form: 0
```



```
Enter the multiplicand: 3
Enter the multiplier: 4
The multiplicand in binary form is: [0, 0, 1, 1]
The multiplier in binary form is: [0, 1, 0, 0]
step    Accumulator         Q           Q_not
1    [0, 0, 0, 0]    [0, 1, 0, 0]    [0]
2    [0, 0, 0, 0]    [0, 0, 1, 0]    [0]
3    [0, 0, 0, 0]    [0, 0, 0, 1]    [0]
4    [1, 1, 1, 0]    [1, 0, 0, 0]    [1]
5    [0, 0, 0, 0]    [1, 1, 0, 0]    [0]
The answer in its binary form is: [0, 0, 0, 0, 1, 1, 0, 0]
The answer inr its decimal form: 12
```



```
Enter the multiplicand: 100
Enter the multiplier: -1000
The multiplicand in binary form is: [0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0]
The multiplier in binary form is: [1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0]
step       Accumulator                          Q                             Q_not
1    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]    [1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0]    [0]
2    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]    [0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0]    [0]
3    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0]    [0]
4    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]    [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1]    [0]
5    [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0]    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]    [1]
6    [1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1]    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]    [1]
7    [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1]    [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]    [0]
8    [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]    [1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]    [0]
9    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]    [0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0]    [0]
10    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]    [1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0]    [0]
11    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]    [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1]    [0]
12    [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1]    [0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0]    [1]
The answer in its binary form is: [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0]
The answer inr its decimal form: -100000
```



```
Enter the multiplicand: 800
Enter the multiplier: -7611
The multiplicand in binary form is: [0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0]
The multiplier in binary form is: [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1]
step         Accumulator                                   Q                                  Q_not
1    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]    [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1]    [0]
2    [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0]    [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0]    [1]
3    [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0]    [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1]    [0]
4    [1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0]    [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]    [1]
5    [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0]    [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0]    [0]
6    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1]    [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]    [0]
7    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0]    [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1]    [0]
8    [1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1]    [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]    [1]
9    [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1]    [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0]    [0]
10    [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1]    [1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]    [0]
11    [1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1]    [1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]    [1]
12    [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0]    [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]    [0]
13    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1]    [0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0]    [0]
14    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0]    [1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1]    [0]
15    [1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0]    [0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0]    [1]
The answer in its binary form is: [1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0]
The answer inr its decimal form: -6088800
```