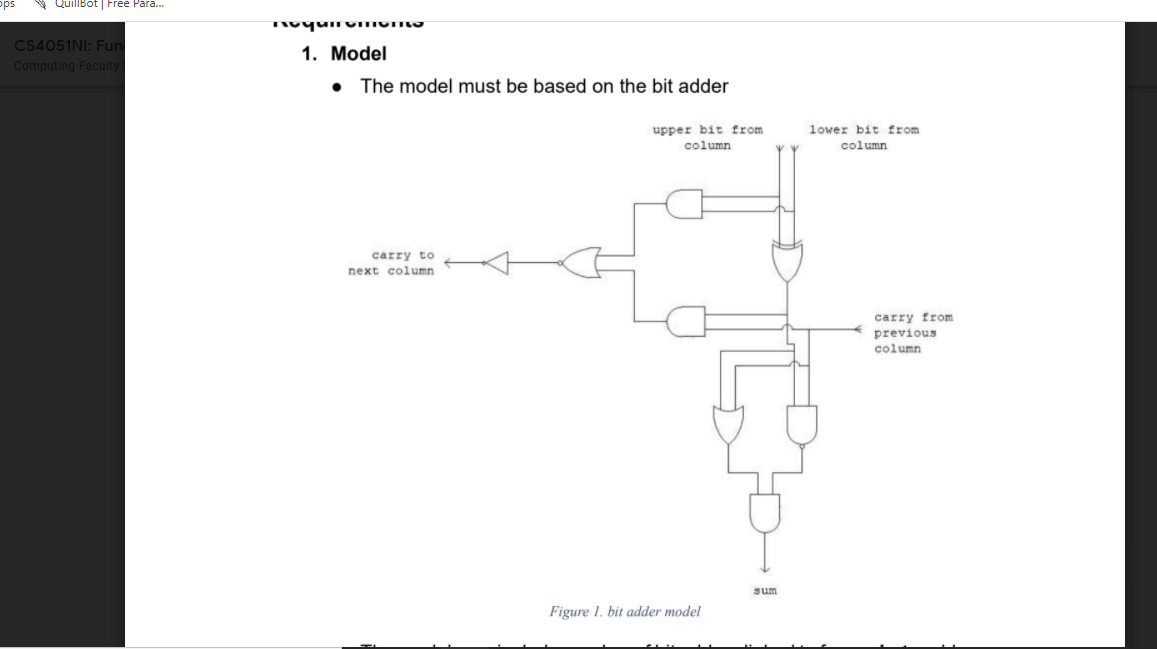
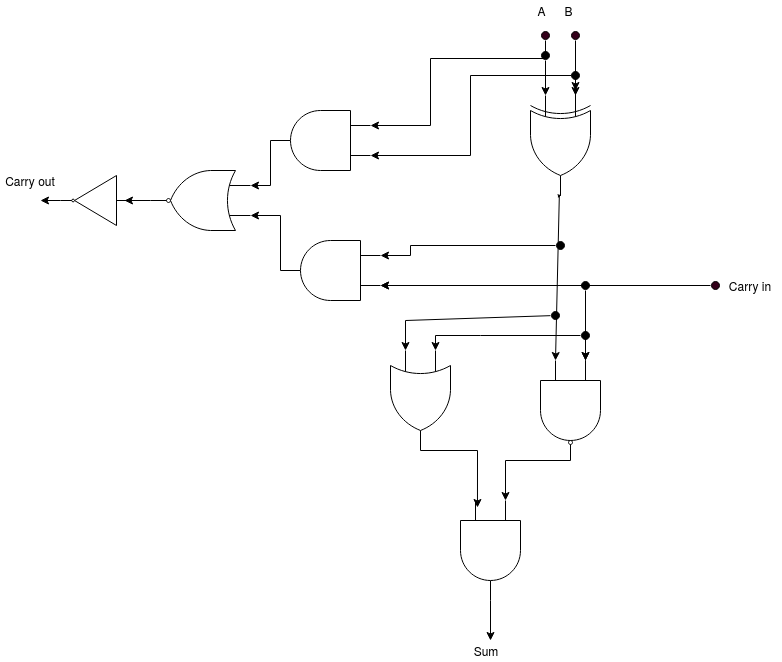
**Models:**

**1 bit adder:**

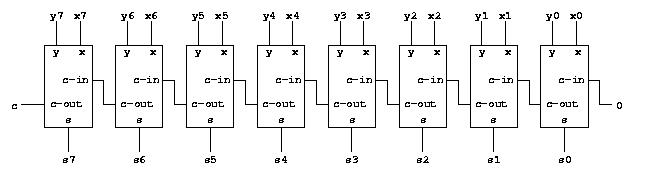
****

****

Figure(1)

Here, different logical gates (AND, OR, XOR, NOR, NAND and NOT gates) are arranged in a circuit so as it works as a 1 bit binary full adder.

**8 bit adder:**

****

Figure(2)

Here, each box represent a fully working 1 bit adder represented in the figure(1).

**Data Structure:**

Here, the data structures used are Dictionary and List.

* Dictionary: It is used to assign the number of zeros in the most significant bits (MSB) whenever the input or output binary value is less than 8 bits.
* List: It is used to store the individual bits of binary number value in it respective order so as to represent the binary value of input or output number.

**Algorithm (Pseudocode)**:

function calculation():

while True:

carry\_out = 0

carry\_in = 0

pos = -1

#1 bit adder implementation

If ( 1 bit adder )

a, b = ask for two numbers either 1 or 0

if( (a>=0 and a<=1) and (b>=0 and b<=1)):

num1 = converToBinary(a)

num2 = converToBinary(b)

bit1 = int(num1[pos])

bit2 = int(num2[pos])

#calculate XOR and NAND

xor\_1 = xorGate(bit1, bit2)

nand\_1 = nandGate(xor\_1, carry\_in)

#calculate OR and store and value from NAND in sum\_val

or\_1 = orGate(xor\_1, carry\_in)

sum\_val = andGate(or\_1, nand\_1)

#calculate AND

and\_1 = andGate(bit1, bit2)

and\_2 = andGate(xor\_1, carry\_in)

#calculate XNOR and NOT

xnor\_1 = xnorGate(and\_1, and\_2)

carry\_out = notGate(xnor\_1)

#convert the bits into string and store after concatenating

result = str(sum\_val) + result

#reinitializing the carry\_in with the current value of carry\_out

carry\_in = carry\_out

#incrementing the value of pos

pos -= 1

else(throw exception error)

#8 bit adder implementation

Else if ( 8 bit implementation)

a, b = ask for two numbers between 0 and 255

num1 = converToBinary(a)

num2 = converToBinary(b)

if( (a>=0 and a<=255) and (b>=0 and b<=255)):

loop until 8 times:

bit1 = int(num1[pos])

bit2 = int(num2[pos])

#calculate XOR and NAND

xor\_1 = xorGate(bit1, bit2)

nand\_1 = nandGate(xor\_1, carry\_in)

#calculate OR and store and value from NAND in sum\_val

or\_1 = orGate(xor\_1, carry\_in)

sum\_val = andGate(or\_1, nand\_1)

#calculate AND

and\_1 = andGate(bit1, bit2)

and\_2 = andGate(xor\_1, carry\_in)

#calculate XNOR and NOT

xnor\_1 = xnorGate(and\_1, and\_2)

carry\_out = notGate(xnor\_1)

#convert the bits into string and store after concatenating

result = str(sum\_val) + result

#reinitializing the carry\_in with the current value of carry\_out

carry\_in = carry\_out

#incrementing the value of pos

pos -= 1

else(throw exception error)

#final value of result is converted into binary after concatenating

#and typecasting into integer

result = str(c\_out) + result

answer = convertToDecimal(result)

dispay answer

choice = "Do you want to continue, y/n?"

if choice == 'n':

break

#final value of result is converted into binary after concatenating

#and typecasting into integer

result = str(c\_out) + result

answer = convertToDecimal(result)

dispay answer

choice = "Do you want to continue, y/n?"

if choice == 'n':

break

convertToBinary(val):

#converts decimal into binary

val = bin(num).replace('0b', "")

#depending on the length of val concatenating the number of zeros to make

#8 bits using switcher dictionary

switcher = {

1:"0000000",

2:"000000",

3:"00000",

4:"0000",

5:"000",

6:"00",

7:"0",

8:""

}

# returns either number of zeros as per length or the value itself. If the

# length is greater than 8 simply return the value otherwise concatenate

# required number of zeros and return

if len(val) > 8:

final\_value = val

else:

final\_value = switcher.get(len(val), val)+val

return final\_value

# XOR gate returns 0 for eqaul bits and 1 for unequal bits

function xorGate(a, b):

if a != b:

return 1

else:

return 0

# XNOR gate returns negation of value returned by XOR gate

function xnorGate(a, b):

val = xorGate(a, b)

return notGate(val)

# NAND gate returns the negation of value returned by AND gate

function nandGate(a, b):

val = andGate(a, b)

return notGate(val)

# AND gate returns when both bits are 1 otherwise 0

function andGate(a, b):

if a == 1 and b == 1:

return 1

else:

return 0

# NOT gate returns 0 for input bit 1 and vice versa

function notGate(n):

if n == 0:

return 1

else:

return 0

# OR gate returns 1 for one of the bit 1, otherwise returns 0

function orGate(a, b):

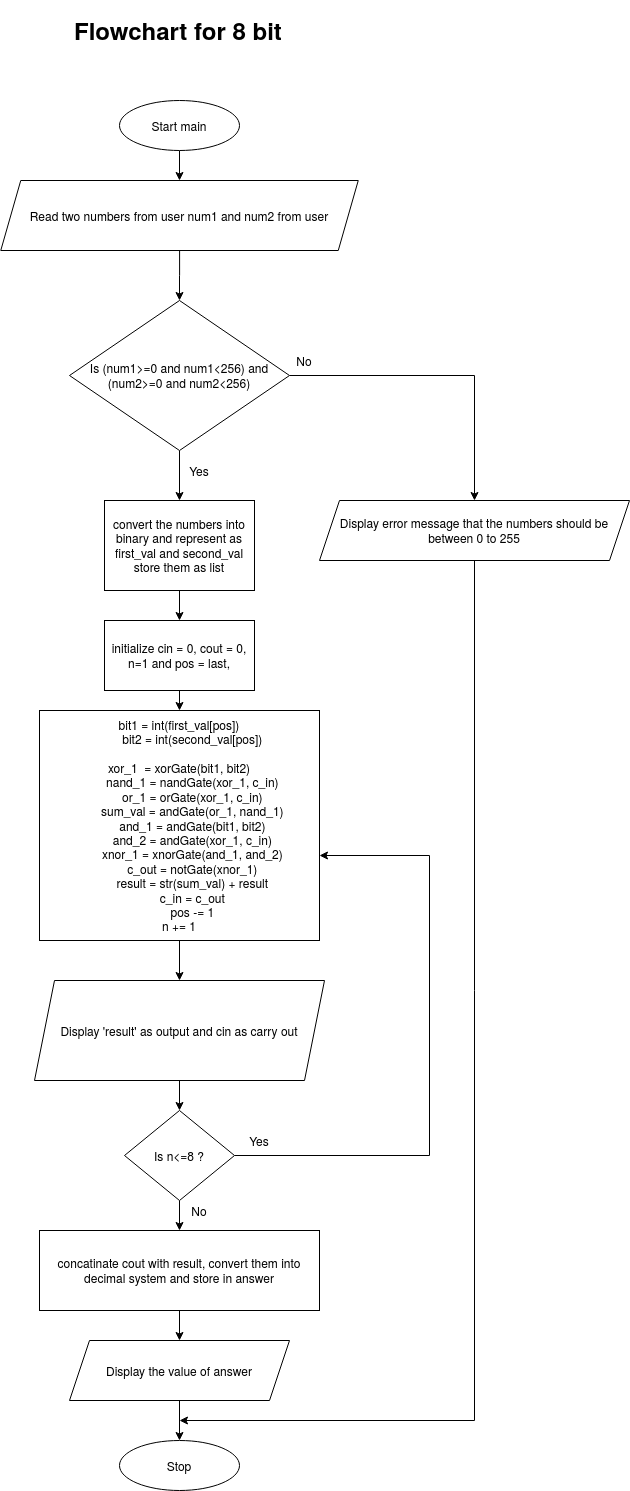
if a == 1 or b == 1:

return 1

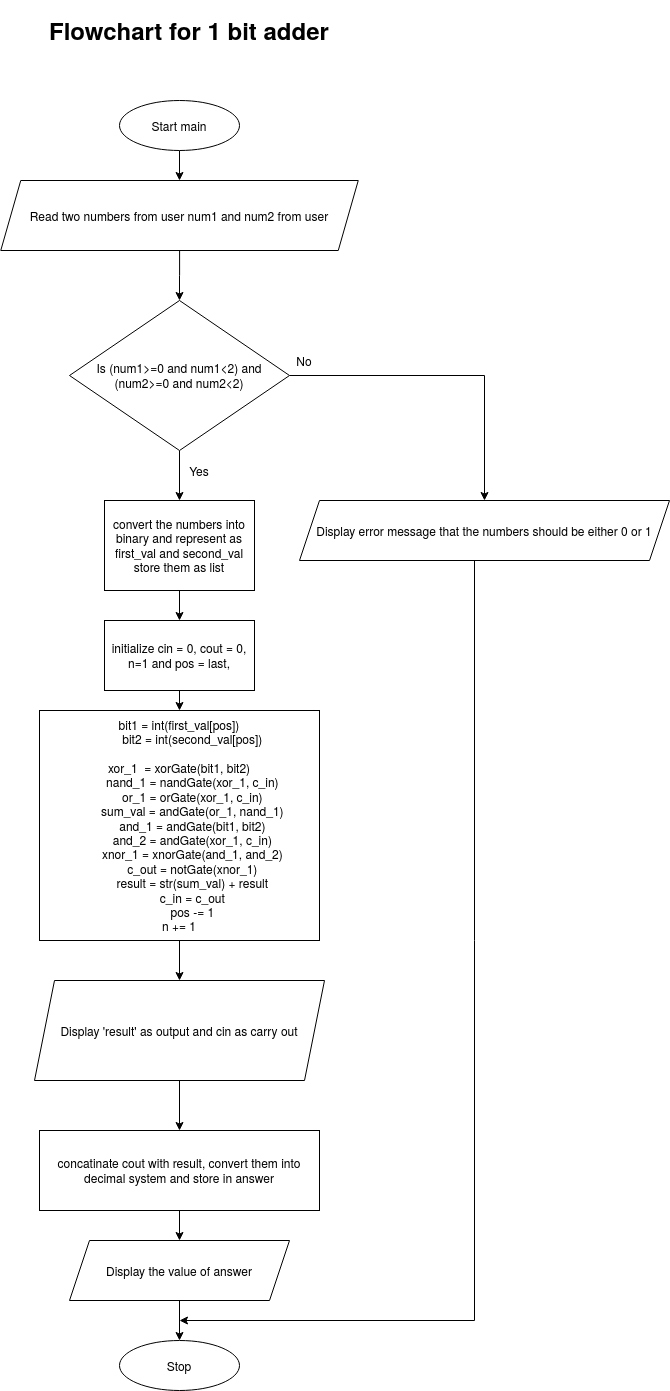
else:

return 0

**Flowchart :**



Figure(3) : Flowchart for 8 bit adder



Figure(4) : Flowchart for 1 bit adder