



**slington college**  
(इस्लिङ्टन कलेज)

**Module Code & Module Title**

**CS4051NI Fundamentals of Computing**

**Assessment Weightage & Type**

**100% Individual Coursework**

**Year and Semester**

**2019-20 Autumn**

**Student Name: Aashman Uprety**

**Group: L1N1**

**London Met ID: 19031231**

**College ID: NP01NT4A190066**

**Assignment Due Date: 2020-06-08**

**Assignment Submission Date: 2020-06-08**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.*

## Table of Contents

<b>Introduction .....</b>	<b>1</b>
<b>Logic Gates.....</b>	<b>2</b>
❖ AND gate.....	2
❖ OR gate.....	2
❖ NOT gate.....	3
❖ NAND gate .....	3
❖ NOR gate .....	4
❖ XOR gate .....	4
❖ XNOR gate.....	5
<b>Adder.....</b>	<b>5</b>
❖ Half Adder .....	5
❖ Full Adder.....	6
<b>Model.....</b>	<b>8</b>
❖ Full Adder Model .....	8
❖ Working Mechanism.....	9
<b>Algorithm .....</b>	<b>11</b>
❖ Algorithm for the coursework.....	11
❖ General algorithm for 8-bit adder (Not based on the model of coursework) .....	14
<b>FLOWCHART .....</b>	<b>15</b>
<b>PSEUDOCODE.....</b>	<b>17</b>
❖ Pseudocode for user_input_module.py .....	17
❖ Pseudocode for conversion_module.py .....	19
❖ Pseudocode for addition_module.py .....	20
❖ Pseudocode for to_run_module.py .....	21
<b>DATA STRUCTURES.....</b>	<b>23</b>
❖ List .....	23
❖ String.....	25
<b>TESTING.....</b>	<b>26</b>
❖ Test 1 .....	26
❖ Test 2 .....	27
❖ Test 3 .....	28
❖ Test 4 .....	29
❖ Test 5 .....	30
<b>CONCLUSION .....</b>	<b>31</b>
<b>References.....</b>	<b>32</b>

## List of Figures

Figure 1: Diagram of AND gate .....	2
Figure 2: Figure of OR gate.....	2
Figure 3: Diagram of NOT gate .....	3
Figure 4: Diagram of NAND gate.....	3
Figure 5: Diagram of NOR gate.....	4
Figure 6: Diagram of XOR gate .....	4
Figure 7: Diagram of XNOR or ENOR gate .....	5
Figure 8: Circuit diagram of Half Adder (A.Bakhtiar, n.d.) .....	6
Figure 9: Circuit Diagram of Full Adder .....	7
Figure 10: Full adder model .....	8
Figure 11: Flowchart for coursewrok .....	16
Figure 12: Screenshot for test 1 .....	26
Figure 13: Screenshot for test 2 .....	27
Figure 14: Screenshot for test 2 .....	28
Figure 15: Screenshot for test 4 .....	29
Figure 16: Screenshot for test 5 .....	30

## List of Tables

Table 1: Truth table of AND gate.....	2
Table 2: Truth table of OR gate:.....	2
Table 3: Truth table of NOT gate.....	3
Table 4: Truth table of NAND gate .....	3
Table 5: Truth table of NOR gate .....	4
Table 6: Truth table of XOR gate.....	4
Table 7: Truth table of XNOR gate.....	5
Table 8: Truth table of Half-Adder .....	6
Table 9: Truth table of Full-Adder.....	7
Table 10: Truth table for Full Adder of our example .....	10
Table 11: Test number 1 (To give two inputs and check the behaviour of the main module) .....	26
Table 12: Test 2 (To test the module when numbers were inputted out of range) .....	27
Table 13: Test 3 (To test what happens if the sum of the numbers is not in between 0 and 255).....	28
Table 14: Test 4 (To test what happens if data types other than integer is provided) ...	29
Table 15: Test 5 (To test whether the user was able to do another addition or not).....	30

## Introduction

The report is of the coursework that was provided to us and the coursework that we were assigned was to develop a python program that executes the sum of two integer numbers. Moreover, the operations had to be binary operations and we were even asked to explain the process of addition with the help of byte adder module. All in all, it was even asked to develop algorithm, pseudocode and flowchart for the designed program. This report comprises of everything that we were asked in the coursework. The major objective of this coursework has been listed in the points below: -

- To produce a byte adder module based on bit adder that would execute the sum in the simplest way possible,
- To write algorithm and pseudocode to the designed program and even construct a flowchart that reflects the working mechanism of the whole program,
- To write a program in python using the above designed algorithm, pseudocode and flowchart,
- To carry out different test operations so that we could be able to obtain an accurate program that we want,
- To handle the exceptions using exception handling in python.

To complete the task, the first thing that was needed was to understand about adders. As adder is a digital circuit consisting of different logical gates, understanding of these logical gates were also required to accomplish this coursework. Moreover, the task that was assigned was very hard as we had to write a program that would illustrate how a computer performs addition of integer numbers using 0s and 1s. Not only this, but we even had to present the process diagrammatically with the help of byte adder model. As the toughness was at a peak level, this coursework was started with lots of dedication and determination. With no face to face interaction with any teachers and being completely online, the task was completed. As the task itself was harder but due to the chaos created in our environment because of COVID-19 outbreak made it even tough.

## Logic Gates

Logic gates can be called as the fundamentals of digital circuits which is an electronic circuit consisting of one or more than one inputs. A logic is set or determined to specify the relationship between these inputs and the outputs.

There are seven logic gates i.e. AND, OR, NOT, XOR, NAND, NOR and XNOR. Brief explanation of each gates has been presented below: -

### ❖ AND gate

This gate gives a high or true output if all the inputs are true. Dot is the symbol to represent the AND gate.

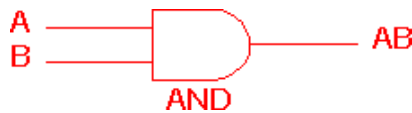


Figure 1: Diagram of AND gate

Input		Output
A	B	A.B (AND)
0	0	0
1	0	0
0	1	0
1	1	1

Table 1: Truth table of AND gate

### ❖ OR gate

This gate gives output as true or high if one or more inputs provided to it is true or high. The symbol to determine that it is a OR gate is plus.



Figure 2: Figure of OR gate

Input		Output
A	B	A+B (OR)
0	0	0
1	0	1
0	1	1
1	1	1

Table 2: Truth table of OR gate:

## ❖ NOT gate

It is also called a compliment gate because it provides the opposite output to that of input. It just reverses the input. It is denoted by either  $A'$  or a bar on top of  $A$ .



Figure 3: Diagram of NOT gate

Input	Output
A	A' (NOT)
1	0
0	1

Table 3: Truth table of NOT gate

## ❖ NAND gate

It is a combination of two gates i.e. AND and NOT gate. The output of NAND gate is true if any one of the inputs provided is high or true. It becomes low if both the inputs are high.



Figure 4: Diagram of NAND gate

Input		Output
A	B	$\overline{A \cdot B}$
0	0	1
1	0	0
0	1	0
1	1	0

Table 4: Truth table of NAND gate

❖ **NOR gate**

This gate is also the combination of two gates i.e. NOT and OR gate. The output of this gate is false if either one of the input is true. This gate provides true or high output if both the inputs are low or false.

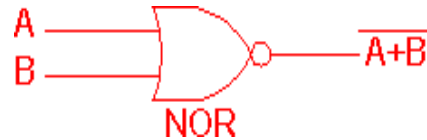


Figure 5: Diagram of NOR gate

Input		Output
A	B	$\overline{A + B}$
0	0	1
1	0	0
0	1	0
1	1	0

Table 5: Truth table of NOR gate

❖ **XOR gate**

It is also named as exclusive OR gate. It provides high output if one or more inputs are high excluding the cases where all inputs are same. It provides opposite output if all the inputs are same.



Figure 6: Diagram of XOR gate

Input		Output
A	B	$A \oplus B$
0	0	0
1	0	1
0	1	1
1	1	0

Table 6: Truth table of XOR gate



❖ **XNOR gate**

It is also called exclusive NOR gate because it does the exact of what NOR gate does. It will give false output if either of the inputs are not same but provides the opposite output if both the inputs are same.

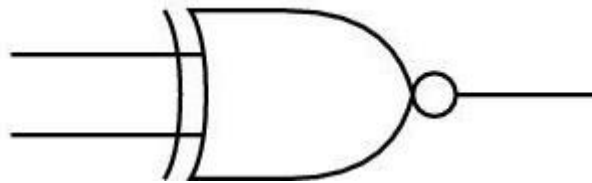


Figure 7: Diagram of XNOR or ENOR gate

Input		Output
A	B	A B
0	0	1
1	0	0
0	1	0
1	1	1

Table 7: Truth table of XNOR gate

## Adder

Adder means addition of numbers which in face is a digital logical circuit in electronics. There are two types of adder namely half adder and full adder

❖ **Half Adder**

Half adder has only two inputs i.e. A and B. It adds them and produces a carry and sum. Sum is calculated by applying XOR gate in the provided inputs whereas AND gate is applied to produce the carry over. Let us assume A and B be the input and S and C be the sum and carry respectively, then the result can be symbolized as below:

$$S = A \oplus B$$

$$C = A.B$$

This adder is used only if the user wants to add two one binary digit quantities.

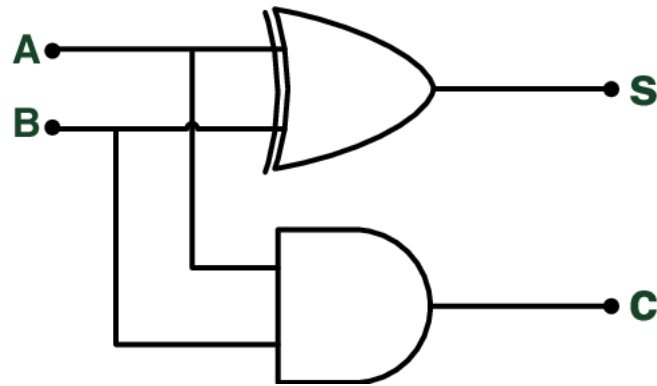


Figure 8: Circuit diagram of Half Adder (A.Bakhtiar, n.d.)

INPUTS		OUTPUTS	
A	B	SUM(S)	CARRY(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 8: Truth table of Half-Adder

### ❖ Full Adder

It has three inputs; A, B and a carry in ( $C_{in}$ ) of lower stages. Full adder adds them and produces a carry out ( $C_{out}$ ) and sum (S). The implementation of full adder is harder than that of half adder. Full adder adds three binary digits and gives two outputs as sum and carry.

There is a co-relation between these two adders. In fact, full adders are designed with the help of half adders. One full adder consists of two half adders and one OR gate but one Half-Adder comprised of one AND and one XOR gate. The thing to bear in mind is that half adder produces certain results and full adder uses the very result provided by half adder to produce some other results.

Addition in the computers are done with the help of full adder module. For adding two one-bit number, one full adder is required. Similarly, to add two eight-bit numbers, eight-bit adder is required. From this, a byte adder is formed. A byte-adder is an adder which produces eight-bit sum. A computer makes the use of byte adders and produces 8, 16 or 32 bit sum depending on the requirements.

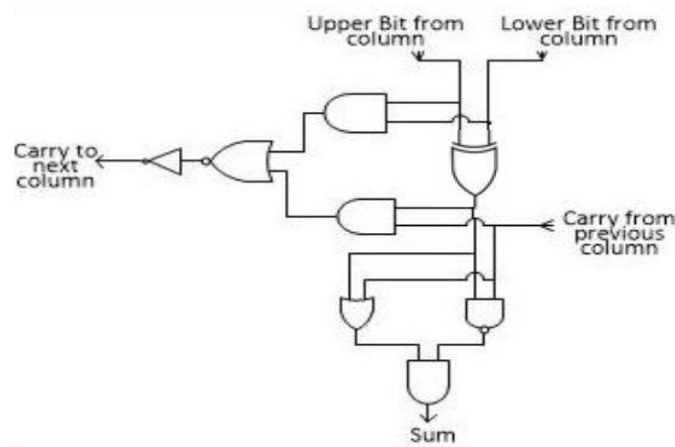


Figure 9: Circuit Diagram of Full Adder

Input			Output	
A	B	Carry from previous Column ( $C_{in}$ )	SUM	Carry to next Column ( $C_{out}$ )
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Table 9: Truth table of Full-Adder

## Model

### ❖ Full Adder Model

The full adder model to be implemented in the coursework was provided to us. With the help of the very model, we had to design a program in python that would do addition of numbers. Byte adder was also needed to be formed by the combination of those given eight and one bit full adders. Before the full adder model was utilized, it was first studied and complete understanding on its design, working mechanism and output was done.

The full adder model that was provided to us is given below: -

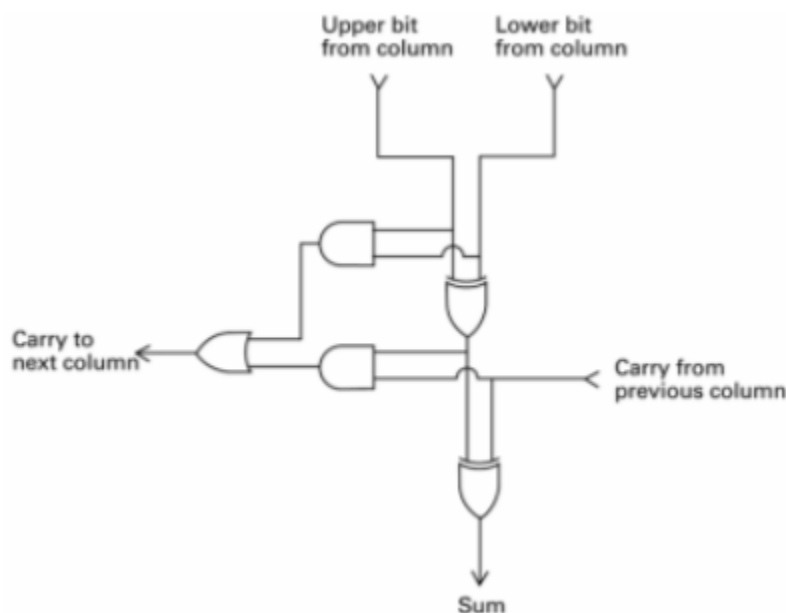


Figure 10: Full adder model

The model presented above is formed by the interconnection of various logical gates that we discussed earlier like AND gate, OR gate, NOT gate, XOR gate, NAND gate and NOR gate. As these gates got interconnected, a logical circuit has formed, and this logical circuit performs the addition of three one-bit binary numbers and results two one-bit outputs i.e. sum and carry. In the addition process using the above model, all the gates used have their own vital purpose for completion.

### ❖ Working Mechanism

The full adder model shown above takes three one-bit inputs, performs the addition and gives out two one-bit outputs.

Let us consider, A and B be the first two inputs and a carry in ( $C_{in}$ ) be the third input in the lower stage. When we pass these three inputs into the full adder, it carries out the addition by passing it into different interconnected logic gates and generates two outputs i.e. sum (S) and a carry-out ( $C_{out}$ ). The carry-out ( $C_{out}$ ) produced from the adder becomes carry in ( $C_{in}$ ) for the next adder. In this way, a single full adder works.

To understand how full adder works, let us take two integer numbers i.e. 50, 61 and pass them through the adder. As we all know, computer works in binary form, let us first convert these integer numbers into 0s and 1s.

$$50 = (00110010)_2$$

$$61 = (00111101)_2$$

Then, from the last position of both the binary numbers, one binary number is extracted and passed into full adders as two inputs i.e. A and B and the third input carry-in is taken as 0 initially. It means 0 is taken from 50 and 1 from 61 and carry-in is taken as zero so the variables become as;  $A=0$ ,  $B=1$  and  $C_{in}=0$ . Then, these three inputs are passed into different logic gates and the outputs are shown in the truth table below. After the first computation,  $Sum=1$  and  $C_{out}=0$  is obtained. Then, carry-out becomes carry-in for the next adder. Then, one binary number is taken again from both the numbers but this time the number is extracted from second last position instead of the last position. So, now the variables become as;  $A=1$ ,  $B=0$  and  $C_{in}=C_{out}$  which is equal to zero and  $Sum=1$  and  $C_{out}=0$  is obtained. In the same way, carry-out becomes carry-in in the next adder and this process keeps continuing until and unless the first binary numbers are taken from both 50 and 61 and passed into the eighth adder. As the process completes an eight-bit sum is formed, which is converted into decimal so that the user can understand it easily.

In our case eight-bit sum will be  $(01101111)_2$  and its corresponding decimal value will be (111).

The whole process that took place inside our adder can be presented as;

A	B	C <sub>in</sub>	A XOR B (XOR-1)	A AND B (AND-1)	XOR-1 AND C <sub>in</sub> (AND-2)	AND-1 NOR AND-2 (NOR-1)	NOT NOR-1 (Cout)	XOR-1 NAND C <sub>in</sub> (NAND1)	XOR-1 OR C <sub>in</sub> (OR-1)	NAND-1 AND OR-1 (Sum)
0	1	0	1	0	0	1	0	1	1	1
1	0	0	1	0	0	1	0	1	1	1
0	1	0	1	0	0	1	0	1	1	1
0	1	0	1	0	0	1	0	1	1	1
1	1	0	0	1	0	0	1	1	0	0
1	1	1	0	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	1	1	1
0	0	0	0	0	0	1	0	1	0	0

Table 10: Truth table for Full Adder of our example

The above table clearly shows how addition of two numbers i.e. 50 and 61 is performed inside adders when the numbers are passed one after another. It also displays how the inputs are passed and how one output from one adder becomes the input in the next adder. Every row in table 10 is representing a full adder.

Before actual addition takes place, computer converts the inputted numbers into either 8-bit, 16-bit or 32-bit binary and so on values depending upon what numbers were supplied. After the process of conversion gets completed, 8-bit, 16-bit or 32-bit and so on sums are calculated depending upon the converted binary values supplied for the addition. As we took 50 and 61 as inputs, which in fact were 8-bit numbers, computer had to use eight full adders or byte-adder in order to calculate the sum. The whole process is portrayed in the truth table above.

## Algorithm

An algorithm is a procedural representation of a program expressed sequentially. Before writing a program, first and foremost thing to do is write an algorithm for the program so that it will be easier for the programmer to write the program.

A good algorithm always has inputs and outputs and every step are clarified clearly and in a precise way. Algorithm should not contain the codes that we use while writing the program. Algorithm should be written in such a way that it can be used to write codes in any programming languages.

An algorithm makes it lot easier to write a program, so an algorithm was developed first for the coursework.

### ❖ Algorithm for the coursework

**Note:** The algorithm for coursework is designed in such a way that it takes two integers as inputs ranging from 0 to 255 and converts them to 8-bit binary numbers and calculates its sum.

### **Procedure:**

Step 1: Start

Step 2: Declare a variable as finput and sinput to receive the inputs and go to step 3.

Step 3: Verify if finput and sinput can be converted to integer. If can be then set fnumber=finput as integer and snumber=sinput as integer and go to step 4 else return to step 2.

Step 4: Verify if fnumber and snumber is in between the range 0-255. If yes go to step 5 else return to step 2.

Step 5: Calculate the sum of fnumber and snumber.

Step 6: If the sum is in between 0 and 255 make a list as 'lst' storing fnumber, snumber and go to step 7 else return to step 2.

Step 7: Assign a = lst[0] and make a list b = [0,0,0,0,0,0,0,0]. Proceed to step 8.

Step 8: Check if a is greater than zero. If true go to step 9 otherwise jump to step 25.

Step 9: Check if a > 127. If true go to step 10 else jump to step 11.

Step 10: Assign  $b[0] = 1$  and  $a = a - 128$  and go to step 8.

Step 11: Check if  $a > 63$  and  $a < 128$  and if true go to step 12 else go to step 13

Step 12: Assign  $b[1] = 1$  and  $a = a - 64$  and go to step 8.

Step 13: Check if  $a > 31$  and  $a < 64$ . If condition is true go to step 14 else jump to step 15

Step 14: Assign  $b[2] = 1$  and  $a = a - 32$  and go to step 8.

Step 15: Check if  $a > 15$  and  $a < 32$ . If true, go to step 16 else go to step 17

Step 16: Assign  $b[3] = 1$  and  $a = a - 16$  and go to step 8.

Step 17: Check if  $a > 7$  and  $a < 16$  and go to step 18 else go to step 19.

Step 18: Assign  $b[4] = 1$  and  $a = a - 8$  and go to step 8.

Step 19: Check if  $a > 3$  and  $a < 8$ . If true go to step 20 else go to step 21.

Step 20: Assign  $b[5] = 1$  and  $a = a - 4$  and go to step 8.

Step 21: Check if  $a > 1$  and  $a < 4$ . If true go to step 22 else go to step 23.

Step 22: Assign  $b[6] = 1$  and  $a = a - 2$  and go to step 8.

Step 23: Check if  $x$  is equal to 1. If yes go to step 24 else go to step 25.

Step 24: Assign  $b[7] = 1$  and  $a = 0$  and go to step 8.

Step 25: Assign  $\text{fbinary} = b$  and go to step 26.

Step 26: Assign  $c = \text{lst}[1]$  and create another list  $\text{lst1} = [0,0,0,0,0,0,0,0]$  and go to step 27.

Step 27: Check if  $c > 0$ . If yes go to step 28 else go to step 44.

Step 28: Check if  $c > 127$  and if yes go to step 29 else go to step 30.

Step 29: Assign  $\text{lst1}[0] = 1$  and  $c = c - 128$  and go to step 27.

Step 30: Check if  $c > 63$  and  $c < 128$  go to step 31 else go to step 32.

Step 31: Assign  $\text{lst1}[1] = 1$  and  $c = c - 64$  and go to step 27.

Step 32: Check if  $c > 31$  and  $c < 64$ . If yes go to step 33 else go to step 34.

Step 33: Assign  $\text{lst1}[2] = 1$  and  $c = c - 32$  and go to step 27.

Step 34: Check if  $c > 15$  and  $c < 32$ . If true go to step 35 else go to step 36.

Step 35: Assign  $\text{lst1}[3] = 1$  and  $c = c - 16$  and go to step 27.

Step 36: Check if  $c > 7$  and  $c < 16$ , if yes go to step 37 else go to step 38.

Step 37: Assign  $\text{lst1}[4] = 1$  and  $c = c - 8$  and go to step 27.

Step 38: Check if  $c > 3$  and  $c < 8$ , if yes go to step 39 else go to step 40.



Step 39: Assign  $lst1[5] = 1$  and  $c = c - 4$  and go to step 27.

Step 40: Check if  $c > 1$  and  $c < 4$ , if yes go to step 41 else go to step 42.

Step 41: Assign  $lst1[6] = 1$  and  $c = c - 2$  and go to step 27.

Step 42: Check if  $c == 1$  and go to step 43 else go to step 44.

Step 43: Assign  $lst1[7] = 1$  and  $c = 0$  and go to step 27.

Step 44: Assign  $sbinary = lst1$  and go to step 45.

Step 45: Make a list;  $add\_list = [0,0,0,0,0,0,0,0]$  and  $carry = 0$  and go to step 46.

Assign  $d = fbinary$

Assign  $e = sbinary$

Step 46: Assign  $i = 7$  and go to step 47.

Step 47: Check if  $i > -1$ , if it is true go to step 48 else set  $bit\_add = add\_list$  and go to step 49.

Step 48: Assign  $add\_list[i] = (d[i] \wedge e[i]) \wedge carry$   
           set  $carry = (carry \& d[i]) \mid (carry \& e[i]) \mid (d[i] \& e[i])$   
           assign  $i = i - 1$   
           Go to next step.= i.e. step 49.

Step 49: Set  $carry\_bitwise = lst[0] \& lst[1]$  and  $added = lst[0] \wedge lst[1]$ . Go to step 50.

Step 50: Check if  $carry$  is greater than 0. If true go to step 51 else go to step 52.

Step 51: Assign  $shift\_left = carry \ll 1$ ,  $carry = added \& (carry \ll 1)$  and  $added = added \wedge shift\_left$ . Go to step 52.

Step 52: Assign  $bitwise\_add = added$

Step 53: Convert all i.e.  $fbinary$ ,  $sbinary$  and  $bit\_add$  into strings and print them. Proceed to next step which is step 54.

Step 54: Display  $bitwise\_add$  as the sum in decimal form and go to step 55.

Step 55: Take a input from user as 'yes' or 'no' for the question 'do you want to do another addition or not?'. Assign the inputted terms into a variable called 'check'. Proceed to step 56.

Step 56: If  $check$  is equal to 'yes', return to step 2 and if  $check$  is equal to 'no' proceed to step 57. If  $check$  is not equal to both, 'yes' or 'no', display the question again.

Step 57: Stop

**❖ General algorithm for 8-bit adder (Not based on the model of coursework)**

As I have presented the detailed algorithm for the model provided in the coursework above, it might become harder to understand. Because of which I have generalized the algorithm again and presented in a simplest way possible.

STEP 1 : Start

STEP 2 : Declare variables A, B and Sum.

STEP 3 : Read values of A and B from user.

STEP 4 : Convert A and B to binary values.

STEP 5 : Store the converted binary values into two separate lists.

STEP 6 : Take out one value at a time from each list starting from the end position.

STEP 7 : Pass those values to first AND gate and also to XOR gate.

STEP 8 : Supply the output from XOR gate and carry-in to second AND gate.

STEP 9 : Supply the output from first AND gate and second AND gate to NOR gate.

STEP 10 : Supply the output from NOR gate to NOT gate and store it in carry-out.

STEP 11 : Supply the output from XOR gate and carry-in to NAND gate and OR gate.

STEP 12 : Pass the output from NAND gate and OR gate to third NAND gate.

STEP 13 : Store the result from third AND gate and store it in the list of sum.

STEP 14 : Assign the value of carry-out to carry-in.

STEP 15 : Repeat STEP 6 to STEP 14 until the sum reaches to eight bits.

STEP 16 : Reverse the SUM list.

STEP 17 : Convert the reversed SUM list to string and then to decimal.

STEP 18 : Store the obtained output from STEP 17 to sum variable

STEP 19 : Display SUM




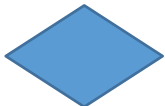

STEP 20 : Prompt the user if they want to continue. If 'yes' go to step 2 else proceed

STEP 21 : STOP

## FLOWCHART

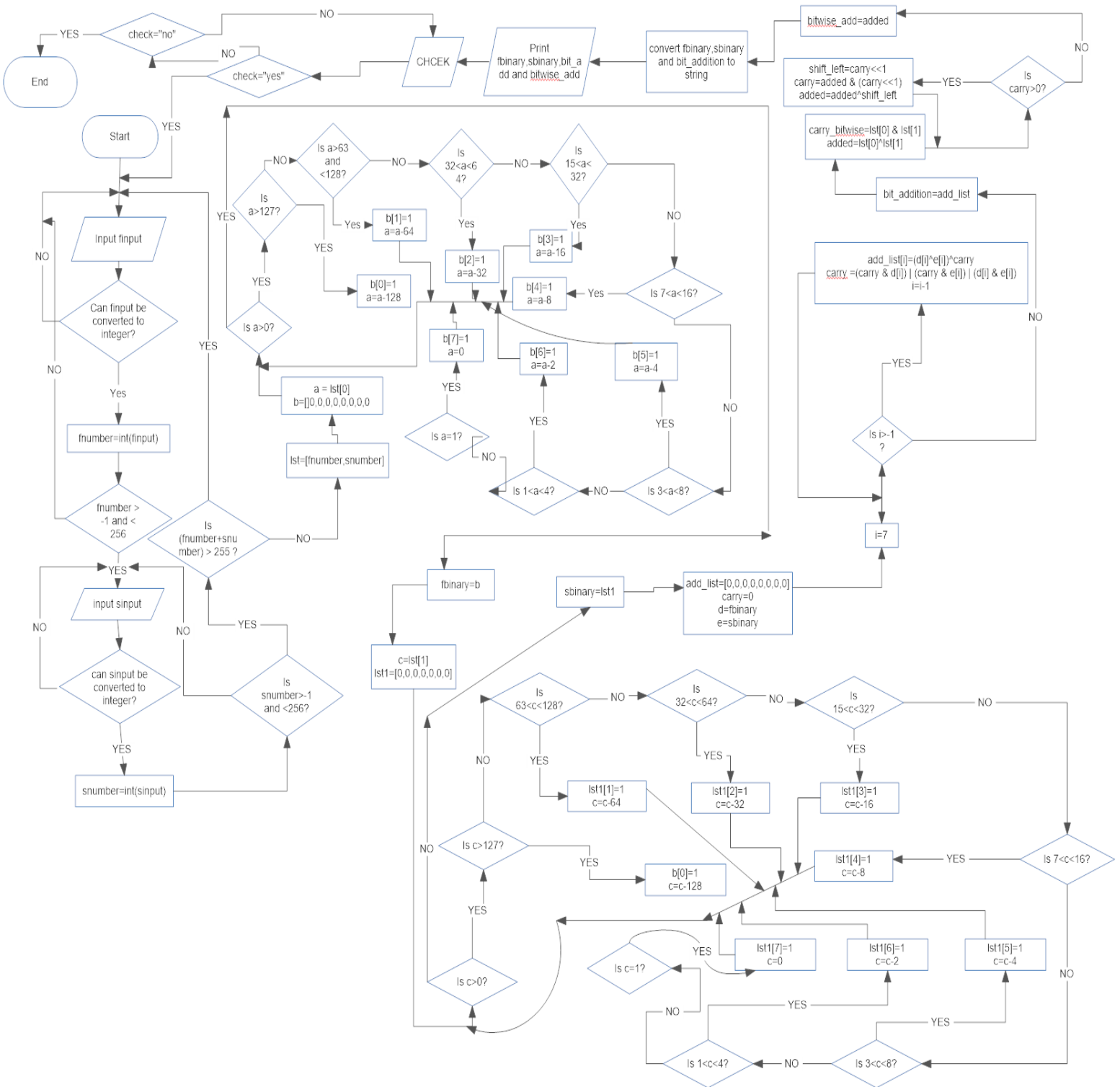
Flowchart is the diagrammatical representation of the algorithm of a program using various shapes, arrows and symbols. Algorithms and pseudocodes are the textual representation of a program whereas flowchart is a graphical method of designing a program. It shows the steps that are carried out during the execution of a program. A flowchart plays a vital role in the process of program development as well as it makes easier for the one who is trying to understand our written codes in different programming language.

Flowchart has lots of different shapes which has its own meaning. Following are the shapes and the meanings of most widely used shapes while creating a flowchart: -

-  : - This symbol is called terminal or a oval shaped which used to represent start and end of a program.
-  : - This symbol is needed in a flowchart when we have to symbolize the input and output
-  : - This symbol is used for mathematical operations and data manipulations.
-  : - This symbol is used for comparison or decision making.
-  : - This is used to indicate the flow of the program in the flowchart through various shapes.

Flowchart is a key in programming. Using flowchart, programmers can easily express their program logic in an effective way. Similarly, flowchart is also a must while documenting a program. As flowchart expresses how the code works step by step, it even makes the programmer easier in changing and debugging the codes when needed.

As flowchart is a most needed thing, following is the flowchart to the coursework: -



## PSEUDOCODE

Pseudocode is one of the simplest ways of writing programming language in English which can also be told as structured English. A pseudocode and a programming code are two completely different things as pseudocodes can't be compiled. A pseudocode uses keywords like **READ, GET, PRINT, WRITE, DISPLAY, OUTPUT, etc.** while writing codes. The main advantage of writing a pseudocode is that it can help the programmers maintain the complexity and flexibility when they are writing long programs in any programming language. Moreover, pseudocode helps in minimizing the chances of errors in the program to be written.

As pseudocode is must while developing a program, I have listed down the pseudocodes of four different modules I have created in python.

### ❖ Pseudocode for user\_input\_module.py

**module** user\_input\_module

**def** check\_for\_integer(number):

**try**:

            int(number)

**return** True

**except** ValueError:

**return** False

**end** check\_for\_integer(number)

**def** user\_input\_module():

    fcheck = "no"

    scheck = "no"

    last\_check = "no"

**while** last\_check == "no" :

**while** fcheck == "no" :

            fninput = input("Enter first number: ")

**if** check\_for\_integer(fninput) == False:

                print("In order to add, the data type must be integer. So, please re-

check and enter.")

```
        end if
    else
        fnumber = int(fninput)
        if fnumber > -1 and fnumber < 256 :
            fcheck = "yes"
        end if
    else
        print("As we are using 8 bit adder, please bear in mind
        that only numbers between 0-255 is acceptable. So, please re-
        check and enter.")
    end else
end else
end while

while scheck == "no" :
    sinput = input("Enter second number: ")
    if check_for_integer(sininput) == False then
        print("In order to add, the data type must be integer. So, please re-check
        and enter.")
    end if
    else
        snumber = int(sininput)
        if snumber > -1 and snumber < 256 then
            scheck = "yes"
        end if
    else
        print("As we are using 8 bit adder, please bear in mind that only
        numbers between 0-255 is acceptable. So, please re-check and enter.")
    end else
end else
end while
```

```

    if (fnumber + snumber) > 255 then
        print("The sum of the two numbers inputted is greater than 255
which is not possible as we are using 8-bit adder. So, please re-check
and enter")
        fcheck = "no"
        scheck = "no"
    end if
    else
        last_check = "yes"
        return[fnumber,snumber]
    end else
end user_input_module()
end user_input_module

```

#### ❖ Pseudocode for conversion\_module.py

```

module conversion_module
    def decimal_to_binary(number):
        a = number
        b = [0,0,0,0,0,0,0,0]
        while a > 0:
            if a > 127 then
                b[0] = 1
                a = a - 128
            end if
            elif a > 63 and a < 128 then
                b[1] = 1
                a = a - 64
            end elif
            elif a > 31 and a < 64 then
                b[2] = 1
                a = a - 32

```

```

    end elif
    elif a > 15 and a < 32 then
        b[3] = 1
        a = a - 16
    end elif
    elif a > 7 and a < 16 then
        b[4] = 1
        a = a - 8
    elif a > 3 and a < 8 then
        b[5] = 1
        a = a - 4
    elif a > 1 and a < 4 then
        b[6] = 1
        a = a - 2
    elif a == 1:
        b[7] = 1
        a = 0
    end elif
end while
return b
end conversion_module

```

#### ❖ Pseudocode for addition\_module.py

```

module addition_module
    def full_adder(carry_in, d, e)
        added = (d ^ e) ^ carry_in
        carry_out = (carry_in & d) | (carry_in & e) | (d & e)
        return [carry_out, added]
    end full_adder(carry_in, d, e)
def bit_addition(d, e)
    add_list = [0,0,0,0,0,0,0,0]

```



```

    carry = 0
    for i = 7 to -1 with step -1
        add_list[i] = full_adder(carry, d[i], e[i])[1]
        carry = full_adder(carry, d[i], e[i])[0]
    end for
    return(add_list)
end bit_addition(d, e)
def bitwise_addition(d, e):
    carry = d & e
    added = d ^ e
    while carry > 0:
        shift_left = carry << 1
        carry = added & (carry << 1)
        added = added ^ shift_left
    end while
    return added
end bitwise_addition(d, e)
end addition_module

```

#### ❖ Pseudocode for to\_run\_module.py

```

module to_run_module
    import addition_module
    import conversion_module
    import user_input_module
def list_to_string(lst)
    string = ""
    for i in lst
        i = str(i)
        string = string + i
    end for
    return string

```

```
end list_to_string(lst)
check = "yes"
while check != "no":
    if check == "yes":
        lst = user_input_module.user_input_module()
        fbinary = conversion_module.decimal_to_binary(lst[0])
        sbinary = conversion_module.decimal_to_binary(lst[1])
        bit_addition = addition_module.bit_addition(fbinary, sbinary)
        bitwise_add = addition_module.bitwise_addition(lst[0],lst[1])
        print("First inputted number in binary is: ", list_to_string(fbinary))
        print("Second inputted number in binary is: ", list_to_string(sbinary))
        print("Sum of the two inputted numbers in binary is: ",
list_to_string(bit_addition))
        print("Sum of the two inputted numbers in decimal is: ",bitwise_add)
    end if
    else
        print("please only enter yes or no.")
        check = input("Are you willing for another addition? If yes just type yes
than you can else just type no, the program will terminate : ")
        check = check.lower()
    end else
end while
    if check == "no":
        print("The program is terminating. BYEEEEEEEEEE.")
    end if
end to_run_module
```

## DATA STRUCTURES

As python provides varieties of data types like integer, string, Boolean, etc. and data structures like lists, tuples, dictionaries and sets, the most suitable and authentic data types and structures were implemented while writing python code to the coursework. Without the use of data type and data structures, it would have been impossible to develop a program.

The data structures used while developing the program are as follows: -

1. List
2. String

### ❖ List

Lists are also called the collection data type in python which holds the elements or items in closed square bracket. The thing to bear in mind is that it is not necessary that the items between the brackets needs to be of same data type. Each element of a list is linked with an index where the index of first item is always zero. The content as well as the size of a list can be changed as per the requirement during the execution of the program because an element can be added or removed from it.

The elements can be added to the lists with the help of methods like `append()`, `len()`, `remove()`, `clear()` and `extend()`. `Append` can be used to add a single element whereas `extend` can be used to add multiple values to the list. Likewise, `remove()` and `clear()` methods helps to remove the items from the lists whereas `len()` returns the length or the total number of items present in the list. One more advantage of using lists in python is that we can do many more operations like indexing, slicing, concatenating, multiplying and even checking the membership.

The primary or the main data type that have been used in this program is also a list. As we know that lists are mutable, and our coursework was based on binary numbers because of which using a list to hold binary data can be a perfect idea. To take as an example, 8-bit binary number can be represented in a list having length eight. And we

even know that lists work on the basis of indexing its elements so that each element of the list can represent a single bit.

Let us take two integer numbers 25 and 107 represented by two lists `li_a` and `li_b` respectively in binary form.

```
li_a = [0,0,0,1,1,0,0,1]
```

```
li_b = [0,1,1,0,1,0,1,1]
```

Now, each element of both the lists can be accessed with the help of index. For example: `li_a[7]` represents the least significant bit of the number 24 and `li_b[7]` represents the least significant bit of the number 107. Now, let us create another list to store the addition of these two 8-bit binary numbers.

```
li_addition=[0,0,0,0,0,0,0,0]
```

For least significant bit,

```
li_addition[0]=(li_a[7]^li_b[7])^0
```

```
carry_out= (0 & li_a[7]) | (0 & li_b[7]) | (li_a[7] & li_b[7])
```

In this way, a list was used in the binary addition operation for holding the binary bits. Moreover, the same process can be done for most significant bits by setting the `carry_out` obtained from previous calculation as the carry in for the current one.

Likewise, we know that in python a single function cannot return two values at a same time. So, to overcome this issue also lists has played a vital role. A list was created to store the values and a function was created to return those values.

To illustrate let us take an example: A full adder is designed in such a way that it returns two outputs i.e. sum and carry out. As already mentioned, functions in python cannot return two values, these two values i.e. sum and carry out were stored in a list and a function was made to return the very list.

```
def f_adder(c_in,x,y)
    added = (x^y)^c_in
    c_out = (c_in & x) | (c_in & y) | (x & y)
    return[c_out,added]
```

Now to access the sum and carry out we can use the indexes of the list as:

`f_adder(c_in, x, y)[1]` extracts the sum whereas `f_adder(c_in,x,y)[0]` extracts carry out.

## ❖ String

String is a data type in python which is used to store a textual data. Just like an integer data type, string values can also be stored in a variable. In python, string data types are denoted by either a single or a double quotation which depends on the user, but a rule is that either a single or a double quote should be used throughout the program. As in the list, string can also do lots of operations like concatenation, slicing, membership checking, repetition etc. String data type can be converted into other data types with the help of different in-built functions of python.

**Syntax:**

```
<variable_name> = string value
```

Example: name = sita

In the program, lists were used to store and represent binary numbers whereas strings were used for displaying the elements of those lists as it is more convenient.

For example:

```
def li_to_string(list):  
    s = ""  
    for i in list:  
        i = str(i)  
        s = s + i  
    RETURN string
```

The above created function can convert a list to string. As in the above example, the same thing was done in the created program by converting a list containing 8-bit binary number to a string so that when it gets displayed, it becomes easier for the user to understand.

For example:

```
print(li_to_string([0,0,0,1,0,1,1,1]))
```

This prints 00010111, which is easy to understand and is a 8-bit binary number.

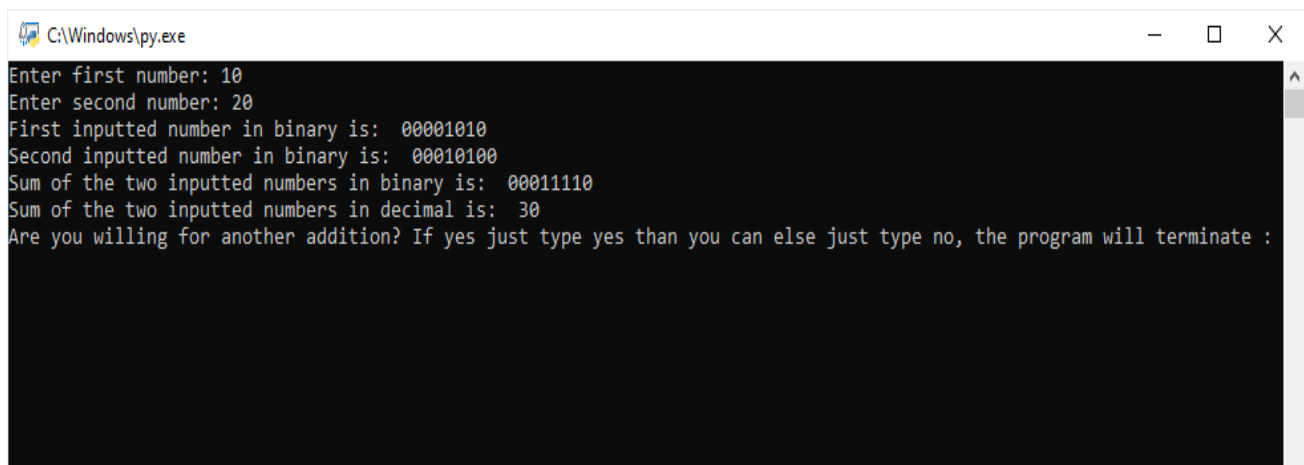
## TESTING

Testing is a very important task during the development of a program or a software as it enables to point a programmer the errors committed and ensure that the developed software is error free. Several tests were performed in the program and the corresponding test tables, results and screenshots has been presented in this report.

### ❖ Test 1

Test No	1
Action	To give two integer inputs and check the to_run_module.py
Expected Result	The module should ask the user to input two integer numbers, compute its sum and deliver the numbers in binary and the sum of the numbers. And at last it should ask if s/he wants to do another addition or not.
Actual Result	The inputted numbers were shown in binary as well as the correct sum between these two numbers was displayed. It was even prompted if the user wants to continue or not.
Test Result	The test was successful.

Table 11: Test number 1 (To give two inputs and check the behaviour of the main module)



```

C:\Windows\py.exe
Enter first number: 10
Enter second number: 20
First inputted number in binary is: 00001010
Second inputted number in binary is: 00010100
Sum of the two inputted numbers in binary is: 00011110
Sum of the two inputted numbers in decimal is: 30
Are you willing for another addition? If yes just type yes than you can else just type no, the program will terminate :
  
```

Figure 12: Screenshot for test 1

## ❖ Test 2

Test No	2
Action	To test what happens if the values less than zero or greater than 255 is entered while inputting first and second numbers.
Expected Result	A suitable message should be displayed saying that enter the numbers between zero and 255.
Actual Result	A suitable message was displayed saying that numbers should be in between 0 and 255 and the user asked to input the number again.
Test Result	The test was successful.

Table 12: Test 2 (To test the module when numbers were inputted out of range)

```

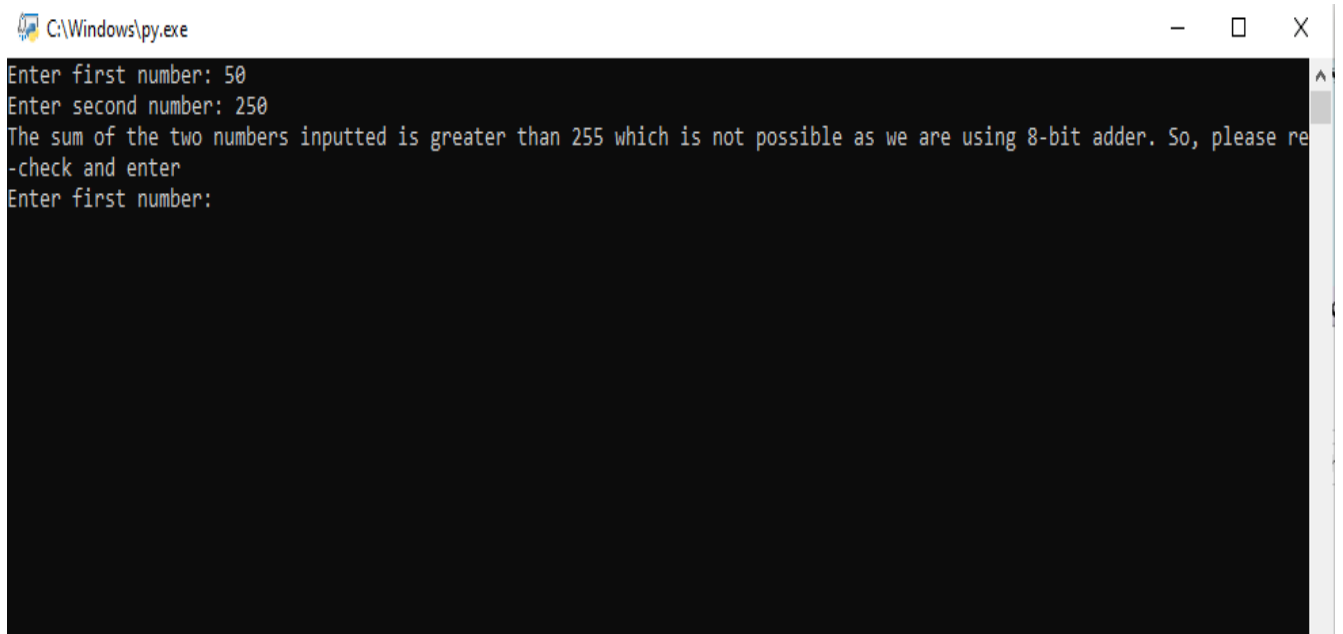
C:\Windows\py.exe
Enter first number: -2
As we are using 8 bit adder, please bear in mind that only numbers between 0-255 is acceptable. So, please re-check and enter.
Enter first number: 256
As we are using 8 bit adder, please bear in mind that only numbers between 0-255 is acceptable. So, please re-check and enter.
Enter first number:
  
```

Figure 13: Screenshot for test 2

## ❖ Test 3

Test No	3
Action	To test what happens if the inputted numbers is in between 0 and 255 but the sum of these numbers is less than 0 or exceeds 255.
Expected Result	A suitable message should be displayed saying that the sum of the two inputted numbers should not be less than 0 or more than 255 and the user was made to input the numbers again.
Actual Result	A suitable message was displayed saying that the sum should also be in between the numbers 0 and 255 and the user was made to input the numbers again.
Test Result	The test was successful.

Table 13: Test 3 (To test what happens if the sum of the numbers is not in between 0 and 255)



```

C:\Windows\py.exe
Enter first number: 50
Enter second number: 250
The sum of the two numbers inputted is greater than 255 which is not possible as we are using 8-bit adder. So, please re-check and enter
Enter first number:
  
```

Figure 14: Screenshot for test 2

**Analysis:** This happened as we know that we are working with 8-bit binary and the maximum value it can return is 255, so even the sum could not exceed 255.



## ❖ Test 4

Test No	4
Action	To test what happens if data types other than integer is inputted. For this test an integer and a floating data type was made to enter.
Expected Result	The program should not accept those data types and a suitable message should be displayed to the user.
Actual Result	The program did not accept the string and the floating data type and the user was displayed a suitable a suitable message and was made to enter the numbers again.
Test Result	The test was successful.

*Table 14: Test 4 (To test what happens if data types other than integer is provided)*

```

C:\Windows\py.exe
Enter first number: abcd
In order to add, the data type must be integer. So, please re-check and enter.
Enter first number: 20.2
In order to add, the data type must be integer. So, please re-check and enter.
Enter first number:
  
```

*Figure 15: Screenshot for test 4*

## ❖ Test 5

Test No	5
Action	To test the reusability of program so that it can give the sum to the user again and again until the user wishes not to do so. The user was made to input 'yes' or 'no' to the question 'if s/he wants to another addition or not. The program should keep running until 'yes' is given and should terminate when 'no' is given.
Expected Result	The user should be able to control the flow of program and do another addition until they want and should terminate when they want.
Actual Result	On giving the input 'yes', the user was able to do another addition whereas on giving 'no', the program was terminated.
Test Result	The test was successful.

Table 15: Test 5 (To test whether the user was able to do another addition or not)

```

Enter first number: 10
Enter second number: 20
First inputted number in binary is: 00001010
Second inputted number in binary is: 00010100
Sum of the two inputted numbers in binary is: 00011110
Sum of the two inputted numbers in decimal is: 30
Are you willing for another addition? If yes just type yes than you can else just type no, the program will terminate : yes
Enter first number: 20
Enter second number: 30
First inputted number in binary is: 00010100
Second inputted number in binary is: 00011110
Sum of the two inputted numbers in binary is: 00110010
Sum of the two inputted numbers in decimal is: 50
Are you willing for another addition? If yes just type yes than you can else just type no, the program will terminate : no
The program is terminating. BYEEEEEEEEEE.
>>> |

```

Figure 16: Screenshot for test 5

## CONCLUSION

All the tasks assigned in the coursework was completed with lots of research and hard works as it was not easy enough to do this one when the world is suffering from such a pandemic (COVID19) and the scenario is full of chaos. For the successful completion of all the tasks, everything was done step by step and the program was finally made. At first, lots of study was done on the topics like byte adder, algorithm, flowchart, pseudocodes, etc. Then the byte adder circuit was created for writing the program using the suitable data types. And at last, a report was designed to demonstrate and document the development, working and functionality of the designed python program.

From various research, it was found that python is a dynamical interpreted programming language which is being widely used in the world as it is straight forward. We can even conclude that the logical operators in python becomes a key to make a digital logic circuit. This report has even shown how algorithm, pseudocode and flowchart are the backbone while developing a program.

Talking about the difficulties, it was very hard to choose the data structure at first that could hold an 8-bit binary number and even do all the tasks mentioned in the coursework. But after thorough research and thinking, lists and strings were selected because in my opinion they were the most suitable and easiest to use for doing the tasks assigned. Moreover, making flowchart and writing pseudocode were too time consuming and lots of hard work was required to do so too.

Last but not the least, the thing that I want to say in the report is that python plays a vital role in the development of program in this modern era. The things that makes python programming language an excellent language is because it is simple, portable and extensible and it can even be used in web development, artificial intelligence, big data and data science. Moreover, python is so popular, and the python developers have very high salary as compared to the developers of other programming language.

**References**

A.Bakhtiar, L. (n.d.). *Researchgate*. Retrieved from Researchgate:  
[https://www.researchgate.net/figure/Half-adder-circuit-diagram\\_fig1\\_272015389](https://www.researchgate.net/figure/Half-adder-circuit-diagram_fig1_272015389)

