Student Name and ID

[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you’re ready to add your content, just click here and start typing.]

[Document title]

[Document subtitle]

Contents

[1. Introduction 3](#_Toc98880647)

[2. Model 3](#_Toc98880648)

[3. Algorithm 3](#_Toc98880649)

[4. Data Structures 3](#_Toc98880650)

[5. Program 3](#_Toc98880651)

[6. Description 3](#_Toc98880652)

[7. Testing 3](#_Toc98880653)

[8. Conclusion 3](#_Toc98880654)

[References 3](#_Toc98880655)

[Deliverables 4](#_Toc98880656)

[Notes on the Assessment 4](#_Toc98880657)

**Task-1**

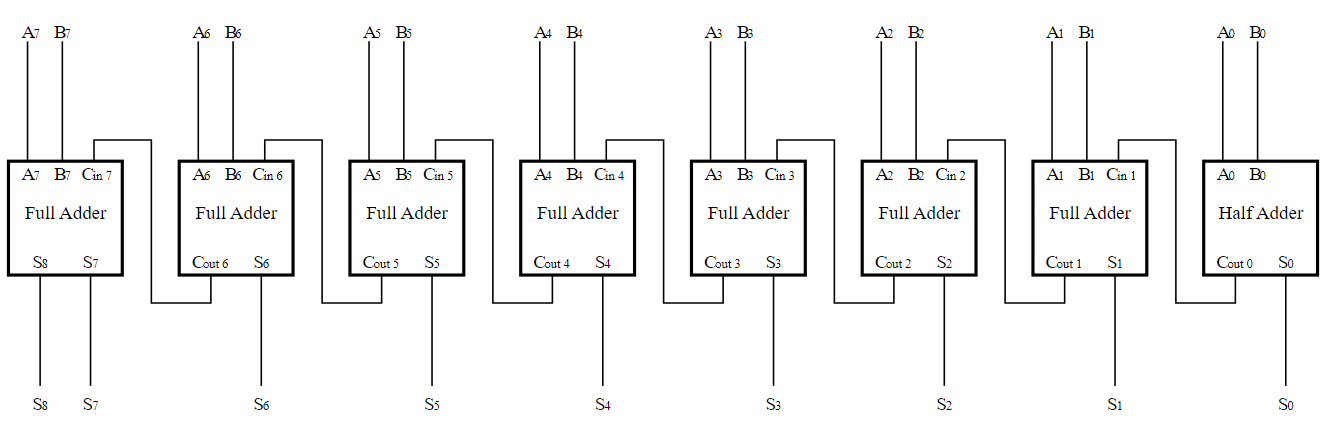
## Introduction

Bit byte adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. A bit byte adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit bit byte adder adds three operands and generates 2-bit results. Truth table of bit byte adder is given below.

## 

**Task-2**

## Model



**Task-3**

## 

## First\_and function

## Works in the same principles of an AND gate

## Comparison of inputs A and B (00,10,01,11)

## The output here will enter our OR gate

## Returns to give us the opportunity to solve XOR gate

## Just in case negative integers or higher than 1 pass the program

## First\_XOR function

## operate on the principles of XOR gate

## Again compares inputs A and B (00,10,01,11)

## Output will enter the second XOR gate as well the second AND gate

## We return so that we can move on (depends on which gate se solve first)

## Similar process as with the AND gate (acts as a safety measure)

## And\_Mk2 Function

## Operates like an AND gate but takes different inputs

## Result of our first XOR gate and previous value

## Previous value has no input so its automatically 0 hence only 2 possible combinations

## Returns so that we have a chance to solve the AND gate or move on

## Again it is just incase something goes wrong

## XOR\_MK2 Function

## Similar operation to the first XOR gate and exact scenario with the AND MK2

## Only two possible inputs are possible

## Output for one part of the Byte Adder

## if none of the above are processed

## first\_OR Function

## Program completes and restarts

## Outputs of our 2 AND gates the only possible combinations are 00, 10

## stores the output in the variable NEXT

## Boot Function

## Program starts from here onwards

## Take input from user

## Return the input

## Define our input with nothing first of all

## loop the program each time we havent entered either 1 or 0

## print the number we have typed

## function gate\_path\_A

## Loads when its called from the Bootup function

## Type specified number from options above

## Calls the function according to the number typed

## If value 1 call and, if value 2 call XOR,if value 3 call gate\_path

## Execute while Loop until previous value become none

## Assign value according to truth table like if value remain same XOR is 0 otherwise Operates the same way as the first gate path but new set of gates to solve

## If value is 1 call AND\_MK2, if value is 2 call XOR\_MK2, if value is 3 call gate\_path\_c function respectively

## Check value for AND operation according to truth table if all bits is one then result is one 1 otherwise zero

## loads the final gate OR

## loops until 1 of 2 numbers are entered.

## Call gate\_path\_A

## Call gate\_path\_B

## Call gate\_path\_C

## Call first\_OR

## Call AND\_MK2

## Call XOR\_MK2

## Call first\_AND

## Call first\_XOR

## Main Execution (program starts off from here)

## Actions Function

## Type 1 to convert integer -> decimal / 2 for subtract / 3 for addition / 4 for multiplier / 5 for floating point

## prompts the user to enter a number according to the options given above

## Call respective function according to input

## Number corresponds to a function

## Conversion Function(Takes a integer number positive or negative and convert to binary)

## 256; total of 8 binary digits (11111111)

## remove the recent input and restart the function

## Addition Function

## Enter two numbers which will add together and then convert into a binary

## Check 1st number range between -255 to 255:

## Check 2nd number range between -255 to 255:

## We add our two successful inputs if they have met the criteria above and then take the SUM and convert to Binary

## Subtract Function

## Similar process to the addition except it will subtract our two inputs instead

## Multiplier Function

## Similar process to the addition except it will multiplies our two inputs instead

## Option=1 for load or start a program and option 2 for terminating the program

## 

**Task-4**

## 4.Data Structures

## Python's input() method is used to receive input from the user. The input function turns any input you provide into a string. The input() function still converts an integer value you enter into a string.

## Grammar: input (prompt)

## Parameter:

## • Prompt: (optional) The string that is printed without a newline to standard output, which is often the screen.

## Object of String returned

Python's print() function prints the message to the screen or to any other standard output device as the output function.

print(value(s), sep=", end=", file=file, flush=flush) is a syntax expression.

Parameters:

• metric(s): Any quantity, as many as you want. Before printing, it will be transformed to a string.

If there are many objects, specify how to divide them using the sep='separator' option.

• end='end': (Optional) Indicate what to print at the end. Default:

Standard: 'n'

• file: A write-method-equipped object that is optional. Default :sys.stdout

• flush: (Optional) A Boolean indicating whether the output is buffered (False) or flushed (False). Standard: False

Returns: It returns output to the screen.

## String Literals

## The main purpose of string literals in the print() function of the Python language is to format or design the appearance of a particular string.

## • n: When printing a statement, this string literal is used to insert a new blank line.

## • "": To print an empty line, use an empty quote ("").

## Character:

#### chr(n)

Returns a character value for the given integer.

chr() does the reverse of ord(). Given a numeric value n, chr(n) returns a string representing the character that corresponds to n:

Bitwise Operator:

|  |  |  |
| --- | --- | --- |
| & Binary AND | Operator copies a bit to the result if it exists in both operands | (a & b) (means 0000 1100) |
| | Binary OR | It copies a bit if it exists in either operand. | (a | b) = 61 (means 0011 1101) |
| ^ Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49 (means 0011 0001) |
| ~ Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61 (means 1100 0011 in 2's complement form due to a signed binary number. |
| << Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240 (means 1111 0000) |
| >> Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15 (means 0000 1111) |

Logical Operator:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is False. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is True. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is True. |

## Python Lists

The list is the most versatile datatype available in Python, which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that the items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"];

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

**Python Dictionary:**

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

## Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print ("dict['Name']: ", dict['Name'])

print ("dict['Age']: ", dict['Age'])

* We used Input/ output , string, logical operator in our program.

**Task-5**

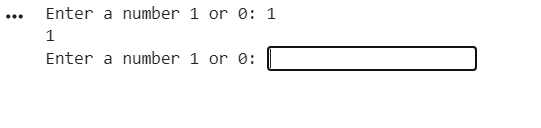
**Program:**

|  |  |
| --- | --- |
| **Code** | **Description** |
| def start(): #program starts off from here  print("Welcome to advanced calculations")  A = int(input("Enter 1 to begin or press 2 to exit: ")) #User enters 1 of the provided options  return A | This function will take input from user to check whether to start or terminate the program |
| def Actions(): #User has 5 options to select from, once they have select one option they will return here to select another  print("Type 1 to convert integer -> decimal / 2 for subtract / 3 for addition / 4 for multiplier / 5 for floating point")  B = int(input("Option: ")) #prompts the user to enter a number according to the options given above  if B == 1: #Number corresponds to a function  print("Loading the converter...")  Conversion() #calls this function to convert decimal to binary  elif B == 2:  print("Loading...")  Subtractor() #calls this function to subtract two decimal numbers and then convert to binary  elif B == 3:  print("Loading...")  Addition() #calls this function  elif B == 4:  print("Loading...")  Multiplier() #Incompleted for other reasons  elif B == 5:  print("Loading...")  Floating\_Point() #Incompleted for other reasons  else:  print("Type the numbers specified")  Actions() | In this function we will select the number for executing out code for add, multiply and subtract and conversion |
| def Conversion():  Den = (int(input("Enter a number between -255 to 255: ")))  print (Den)  if (Den > 256):  print ("Number between -255 to 255 please")  Den = None #remove the recent input and restart the function  Conversion()  elif (Den < -256 ):  print ("Number between -255 to 255 please")  Den = None  Conversion()  else:  print ("Processing...")  print ("In Decimal:" , Den)  print ("In Binary:" , "{0:0{1}b}".format(Den,8))  print("Returning to option menu")  Actions() | Takes a integer number positive or negative and convert to binary  Den short for Denary or Decimal  256; total of 8 binary digits (11111111)  Presents our Denary number and converts into a Binary  Denary but both Denary/Decimal have the same meaning  Return to option tab to select something else |
| def Addition():  X = int(input("Enter a number between -255 to 255: "))  if X < -256:  print(X ,"is in range")  elif X < 256:  print(X ,"is in range")  else:  print("Follow the provided instructions")  X = None  Addition()  Y = int(input("Enter a number between -255 to 255: "))  if Y < -256:  print(Y ,"is in range")  elif Y < 256:  print(Y ,"is in range")  else:  print("Follow the provided instructions")  Y = None  Addition()  SUM = X + Y  print(SUM)    print ("Processing...")  print ("In Decimal:" ,SUM)  print ("In Binary:" , "{0:0{1}b}".format(SUM,9))  print("Returning to option menu")  Actions() | Enter two numbers which will add together and then convert into a binary  we define X as nothing and restart the function, same applies with Y  We add our two successful inputs if they have met the criteria above and then take the SUM and convert to Binary  9 because the user might double number as high as 255  Transfers back to the option tab to try something else out |
| def Subtractor():  X = int(input("Enter a number between -255 to 255: "))  if X < -256:  print(X ,"is in range")  elif X < 256:  print(X ,"is in range")  else:  print("Follow the provided instructions")  X = None  Addition()  Y = int(input("Enter a number between -255 to 255: "))  if Y < -256:  print(Y ,"is in range")  elif Y < 256:  print(Y ,"is in range")  else:  print("Follow the provided instructions")  Y = None  Addition()  SUM = X - Y  print(SUM)    print ("Processing...")  print ("In Decimal:" ,SUM)  print ("In Binary:" , "{0:0{1}b}".format(SUM,9))  print("Returning to option menu")  Actions() | Similar process to the addition except it will subtract our two inputs instead  provide a number between the given range  Subtract our two successful inputs then convert its decimal format into binary  9 because the user might double number as high as 255 |
| option = start()    if option == 1:  print("Loading...")  Actions()  elif option == 2:  print("Terminating program...")  exit() #  else:  print("What did you type?")  start() | Based on what option we chose it follows accordingly  closes down the module (the execution phase)  Just in case the user didnt type either 2 numbers |
| def first\_AND (A,B):  print("Once this gate has been completed, return to solve the XOR gate")  if A == 0 and B == 0:  AND = 0  print(A , " & " , B , " = " , AND)  gate\_path\_A()  elif A == 1 and B == 0:  AND = 0  print(A , " & " , B , " = " , AND)  gate\_path\_A()  elif A == 0 and B == 1:  AND = 0  print(A , " & " , B , " = " , AND)  gate\_path\_A()  elif A == 1 and B == 1:  AND = 1  print(A , " & " , B , " = " , AND)  gate\_path\_A()  else: | Works in the same principles of an AND gate  Comparison of inputs A and B (00,10,01,11)  The output here will enter our OR gate  Returns to give us the opportunity to solve XOR gate  Just incase negative integers or higher than 1 pass the program  print("AND gate malfunctioned") |
| def first\_XOR (A,B):  print("Once this gate has been completed, return to solve the AND gate")  if A == 0 and B == 0  XOR = 0  print(A , " ^ " , B , " = " , XOR)  gate\_path\_A()  elif A == 1 and B == 0:  XOR = 1  print(A , " ^ " , B , " = " , XOR)  gate\_path\_A()  elif A == 0 and B == 1:  XOR = 1  print(A , " ^ " , B , " = " , XOR)  gate\_path\_A()  elif A == 1 and B == 1:  XOR = 0  print(A , " ^ " , B , " = " , XOR)  gate\_path\_A()  else: print("XOR gate malfunctioned") | operate on the principles of XOR gate  :Again compares inputs A and B (00,10,01,11)  Output will enter the second XOR gate as well the second AND gate  We return so that we can move on (depends on which gate se solve first)  Similar process as with the AND gate (acts as a safety measure) |
| def AND\_MK2(XOR,PREV):  print("Once this gate has been completed, return to solve the XOR\_MK2 gate")  if XOR == 0 and PREV == 0:  AND\_II = 0  print(XOR , " & " , PREV , " = " , AND\_II)  gate\_path\_B()  elif XOR == 1 and PREV == 0:  AND\_II = 0  print(XOR , " & " , PREV , " = " , AND\_II)  gate\_path\_B()  else: print("AND\_MK2 gate malfunctioned") | Operates like an AND gate but takes different inputs  Result of our first XOR gate and previous value  Previous value has no input so its automatically 0 hence only 2 possible combinations  Returns so that we have a chance to solve the AND gate or move on  Again it is just incase something goes wrong |
| def XOR\_MK2(XOR,PREV):  print("Once this gate has been completed, return to solve the AND\_MK2 gate")  if XOR == 0 and PREV == 0:  SUM = 0  print(XOR , " ^ " , PREV , " = " , SUM)  gate\_path\_B ()  elif XOR == 1 and PREV == 0:  print(XOR , " XOR" , PREV , " = " , SUM)  SUM = 1  gate\_path\_B ()  else:  print("XOR\_MK2 gate malfunctioned") | Similar operation to the first XOR gate and exact scenario with the AND MK2  Only two possible inputs are possible  Output for one part of the Byte Adder  If none of the above are processed |
| def first\_OR(AND,AND\_II):  print("Byte Adder Complete and will return to the start")  if AND == 0 and AND\_II == 0:  NEXT = 0  print(AND , " | " , AND\_II , " = " , NEXT)  Bootup()  elif AND == 1 and AND\_II == 0:  NEXT = 1  print(AND , " | " , AND\_II , " = " , NEXT)  Bootup()  else:  print("first\_OR gate malfunctioned") | Works on the principles of an OR gate, it akes the outputs of our 2 AND gates  Program completes and restarts  Outputs of our 2 AND gates the only possible combinations are 00, 10  stores the output in the variable NEXT |
| B = None  while B is None:  try:  B = int(input("Enter a number 1 or 0: "))  except ValueError:  print("Did you read what the program stated?")  print(str(B)); | Traverse until B is none |
| def gate\_path\_A ():  print("Press 1 to solve AND gate or Press 2 to solve XOR gate or Press 3 to solve new set of gates")  x = int(input("write 1/2/3: "))  print("Loading...")  if x == 1:  first\_AND(A,B)  elif x == 2:  first\_XOR(A,B)  elif x == 3:  gate\_path\_B()  else:  print("Type a number 1,2 or 3")  gate\_path\_A() | Loads when its called from the Bootup function  Type specified number from options above  Calls the function according to the number typed |
| PREV = None  while PREV is None:  try:  PREV = 0  except ValueError:  print("Input is 0; we have nothing stored in the first place")  print("The only input for previous value is: " , PREV); | Execute until previous is none |
| XOR = None  if A == 0 and B == 0: #Depends on what inputs A and B are  XOR = 0  print("The output from the first XOR gate is:" ,XOR)  elif A == 1 and B == 0:  XOR = 1  print("The output from the first XOR gate is:" ,XOR)  elif A == 0 and B == 1:  XOR = 1  print("The output from the first XOR gate is:" ,XOR)  elif A == 1 and B == 1:  XOR = 0    print("The output from the first XOR gate is:" ,XOR)  else:  print("Are you sure you stored input of 1 or 0?") | Assigning value to XOR on the basis of truth table of XOR , if input is different than value is one otherwise zero |
| AND = None  if A == 0 and B == 0: #Depends on what inputs A and B are  AND = 0  print("The output from the first AND gate is:" ,AND)  elif A == 1 and B == 0:  AND = 0  print("The output from the first AND gate is:" ,AND)  elif A == 0 and B == 1:  AND = 0  print("The output from the first AND gate is:" ,AND)  elif A == 1 and B == 1:  AND = 1  print("The output from the first AND gate is:" ,AND)  else:  print("Are you sure you stored inputs of 1 or 0?") | Assigning value to AND on the basis of truth table of AND , if input all input are one then result one otherwise zero on the basis of condition |
| Resume = 1  while Resume == 1:  ALT = Bootup()  if ALT == 1:  print("Simulating Byte Adder...")  gate\_path\_A()  elif ALT == 2:  print("Exiting Byte Adder")  exit()  else:  print("Type a specified number")  Bootup() | loops until 1 of 2 numbers are entered. |
| gate\_path\_A()  gate\_path\_B()  gate\_path\_C()  first\_OR(AND,AND\_II)  AND\_MK2(XOR,PREV);  XOR\_MK2(XOR,PREV);  first\_AND(A,B)  first\_XOR(A,B) | Calling functions one by one |

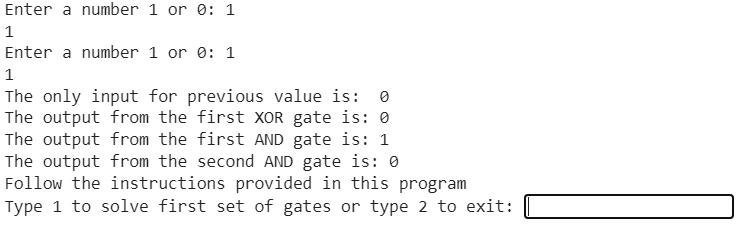
**Task-6**

**Testing:**

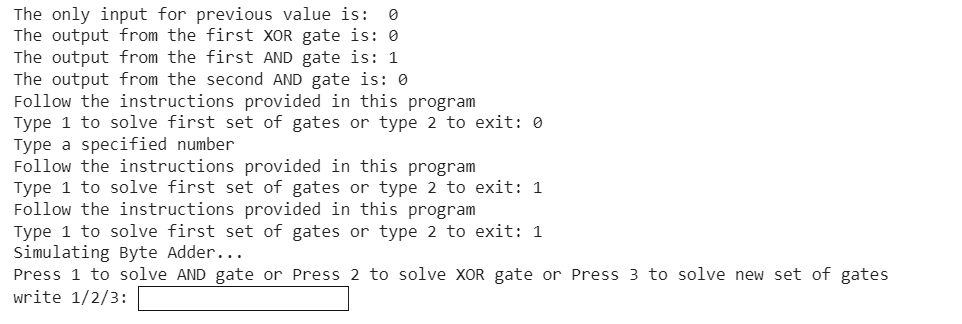
**Test Case=1**

****

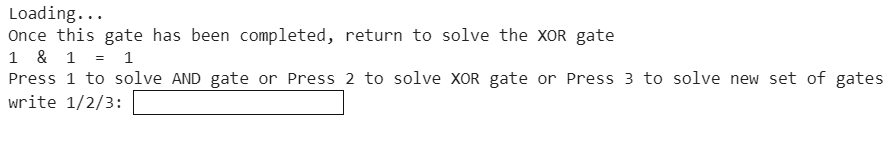
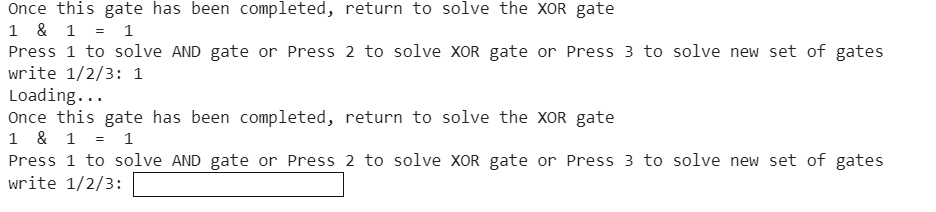
**Test Case=2**

**Test Case=3**

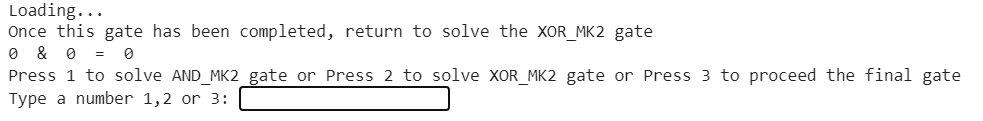
**Test Case=4**

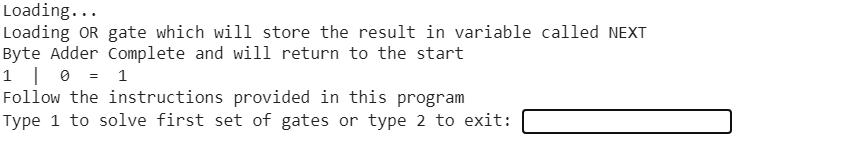
****

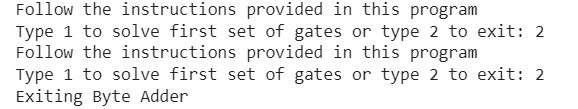
**Test Case=5**

****

**Test Case=6**

**Test Case=7**

**Test Case=8**

****

**Test Case=9**

**Task-7**

**References**

**www.pynative .com**