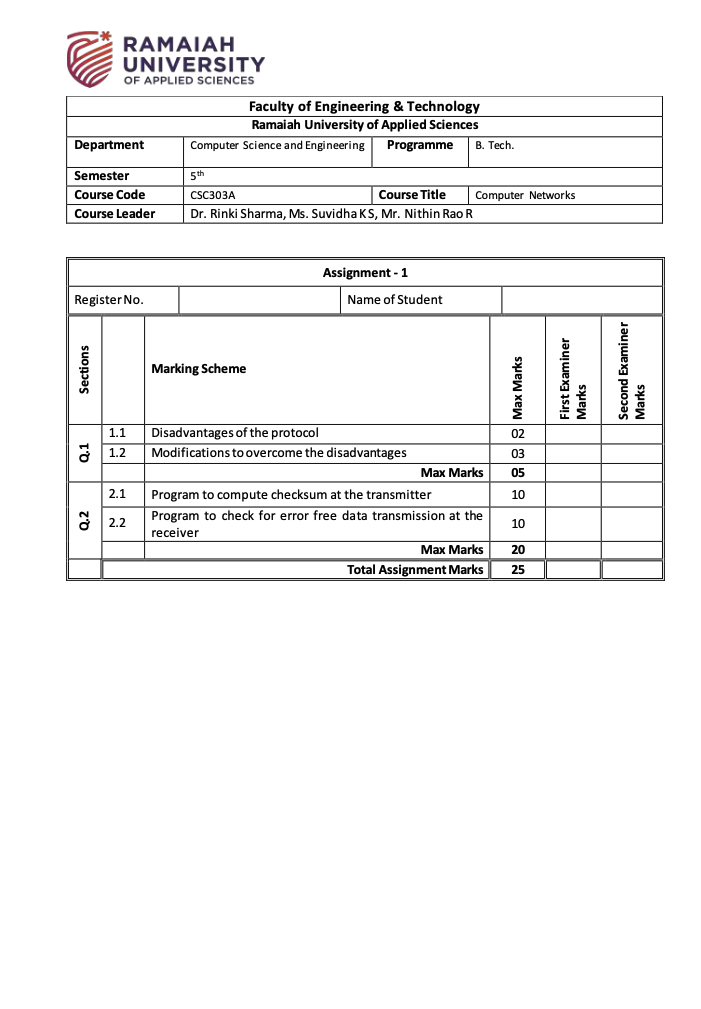


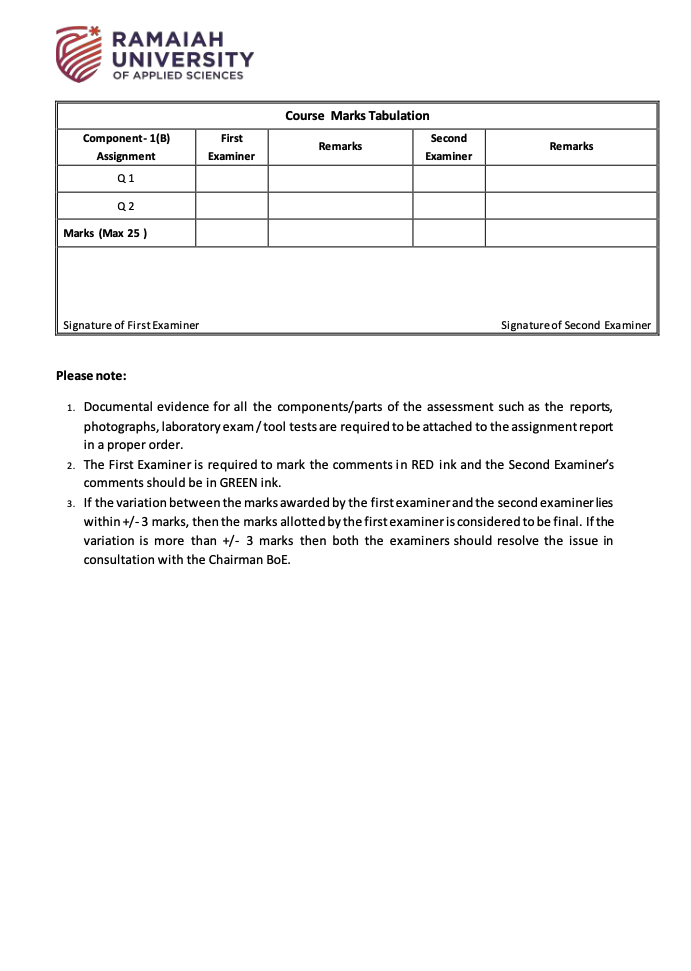
|  |  |
| --- | --- |
| **ASSIGNMENT - 1** | |
| **Course Code** | CSC303A |
| **Course Name** | Computer Networks |
| **Programme** | B Tech |
| **Department** | CSE |
| **Faculty** | FET |

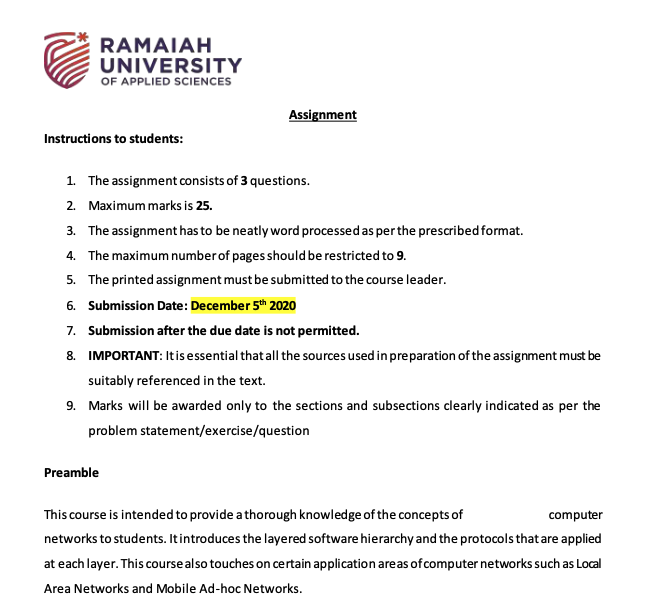
#### 

|  |  |
| --- | --- |
| **Name of the Student** | K Srikanth |
| **Reg. No** | 17ETCS002124 |
| **Semester/Year** | 5th / 3rd Year |
| **Course Leader/s** | Dr. Rinki Sharma |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaration Sheet** | | | | | | | | |
| Student Name | K Srikanth | | | | | | | |
| Reg. No | 17ETCS002124 | | | | | | | |
| Programme | B.Tech | | | | | Semester/Year | 5th / 3rd Year | |
| Course Code | CSC303A | | | | | | | |
| Course Title | Computer Networks | | | | | | | |
| Course Date | 14/09/2020 | | to | | 16/02/2021 | | | |
| Course Leader | Dr. Rinki Sharma | | | | | | | |
| **Declaration**  The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly. | | | | | | | | |
| Signature of the Student | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | Signature of the Reviewer and date | | | | |
|  | | | |  | | | | |







**Question 1**

**1.1)**

**Introduction**

Stop and wait protocol is the simplest flow control method used when a sender wants to sends the data to receiver and after sending data sender stops sending the data and waits for an acknowledgment from the receiver and this process repeats until the data flow is completed from both the parties.

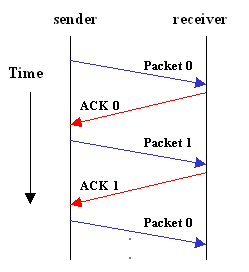


Figure 1 Stop and Wait Protocol

Looking at the image 1 we can see that sender is transmitting packet 0 and is waiting for acknowledgment from the receiver when he receives an acknowledgment that can be accepted then sender will send packet 1 and waits for the next acknowledgment and this process happens until the data is received completely, we can clearly see that it takes lot of time to send and receive data.

**Performance Issue and Disadvantages with stop and wait protocol**

* “Time” is the major performance and disadvantage issue with this protocol as the complete data takes lot of time to reach the receiver
* If there is no acknowledgment from the receiver it stops sending the data and it doesn’t retry and the whole data transfer fails
* If the data is lost from the sender side then the receiver doesn’t retry for the data and the whole data transfer fails
* If there is a delay of acknowledgment then it is considered as an incorrect acknowledgment

**1.2)**

The Modification of the above protocol can be done and improved by adding an **Automatic Repeat Request** so that the if there is a delay or error in transmitting or receiving the data then the system would send the request to server for sending or receiving data depending on the error.

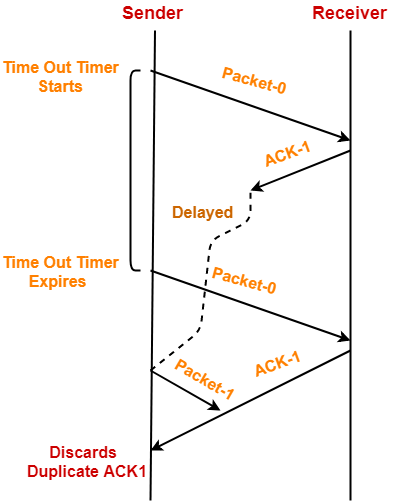


Figure 2 Stop and Wait Protocol with ARQ

The modifications are been made by adding **Automatic Repeat Request** to Stop and wait protocol

**Scenario 1**

Sender transmits the data packet 0 to receiver it is received at receiver side now the receiver has to send an acknowledgment to sender but it fails while transmitting back so what ARQ does is it waits for a certain amount of time and if there is no acknowledgment then it ARQ sends the data packet 0 to receiver and now the acknowledgment is received if by any chance that the previous acknowledgment was interfered then it will discard the previous acknowledgment. we can still see the delay of receiving the full data at that receiver side but compared to Stop and wait protocol this protocol performed better and all the disadvantages were resolved by adding an **Automatic Repeat Request to the protocol**.

**Question 2**

**Binary Addition Function**

*Binary Addition Function Algorithm withs args x, y*

*1. Start*

*2. length = max(len(x), len(y))*

*3. x= fill a with zeros with length of length*

*4. y= fill b with zeros with length of length*

*5. initialize result*

*6. initialize carry*

*7. for loop begins*

*a. for i in range of length*

*b. reminder = carry*

*c. reminder = reminder + 1*

*i. if x[i] == “1” else 0*

*d. reminder = reminder + 1*

*i. if y[i] == “1” else 0*

*e. result = (“1” if reminder = 1 else 0) + result*

*f. if reminder<2 then carry = 0 else 1*

*g. Exit*

*8.if carry! = 0 then result = Binary Addition function (result,’1’)*

*9. return result. zfill(length)*

*10. Stop*

**One’s Compliment Function**

*One’s compliment function algorithm with args a*

*1. Start*

*2. initialize data*

*3. for loop beings*

*a. for in range of length (a)*

*b. if a[i] = 0*

*i. data = data + 1*

*c. else*

*i. data = data + 0*

*d. exit*

*4.return data*

*5.Stop*

**Check Sum Function**

*Check Sum Function algorithm with args x,k*

*1. Start*

*2. blocks = split x with k for i in range of 0, length(x), k*

*3. result = zero fill with length of k*

*4. for loop begins*

*i. result = binary\_addition(blocks[i],result)*

*ii.exit*

*5. result = onescompliment(result)*

*6. return result*

*7. Stop*

**Main Function**

*Main Function algorithm*

*1. Start*

*2. while loop begins*

*i. input choice*

*ii. if choice = 1*

*a. checksum at the transmitter (Question 2.1).*

*iii. if choice = 2*

*a. checksum at the receiver (Question 2.2).*

*iv. if choice = 3*

*a. break*

*v. exit*

*3.Stop*

**2.1)**

**Algorithm for checksum at the transmitter.**

*1. Start*

*2. Input 32-bit string into an array = “Data”*

*3. Input number of segments*

*(only with powers of 2^n where n>=0) = “Segmented”*

*4. Display Function call checksumcalculator (data, segmented)*

*5. Stop*

**2.2)**

**Algorithm for checksum at the receiver with error free data transmission.**

*1. Start*

*2. Input 32-bit string with check sum into an array = “Data”*

*3. Input number of segments*

*(only with powers of 2^n where n>=0) = “Segmented”*

*4. Display Function call checksumcalculator (data, segmented)*

*5. Stop*

**Code (Python)**

**Binary Addition Function**

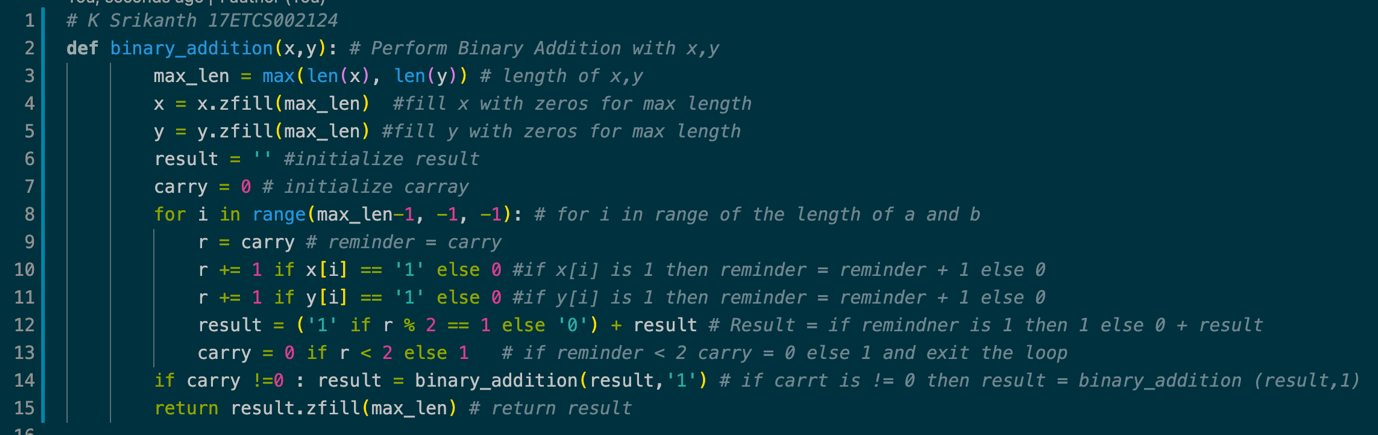
****

Figure 3 Binary Addition Function for given problem statement in python

**One’s Compliment Function**

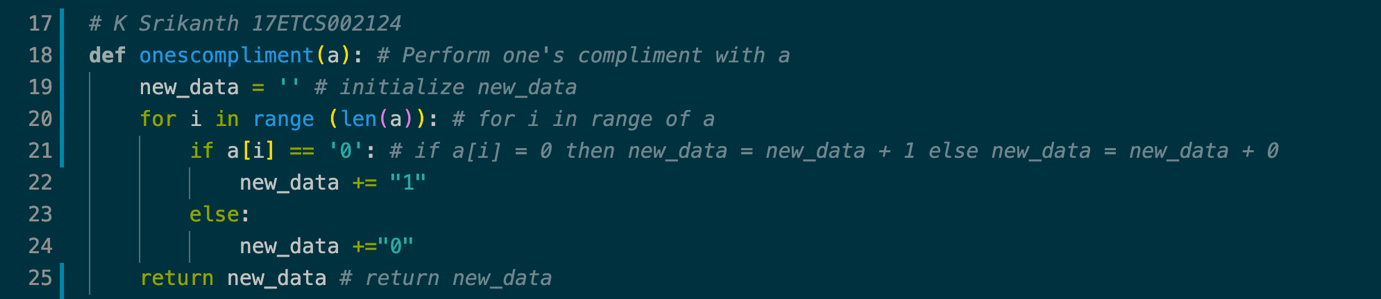
****

Figure 4 One’s Compliment Function for given problem statement in python

**Check Sum Function**

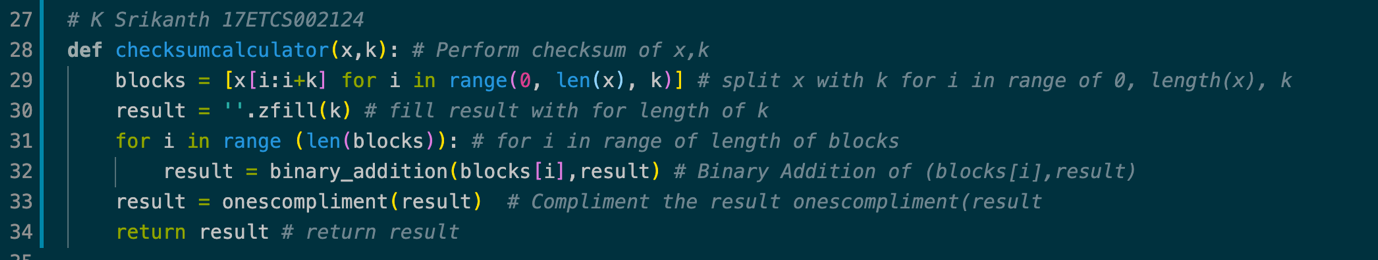
****

Figure 5 Check Sum Calculator Function for given problem statement in python

**Main Function**

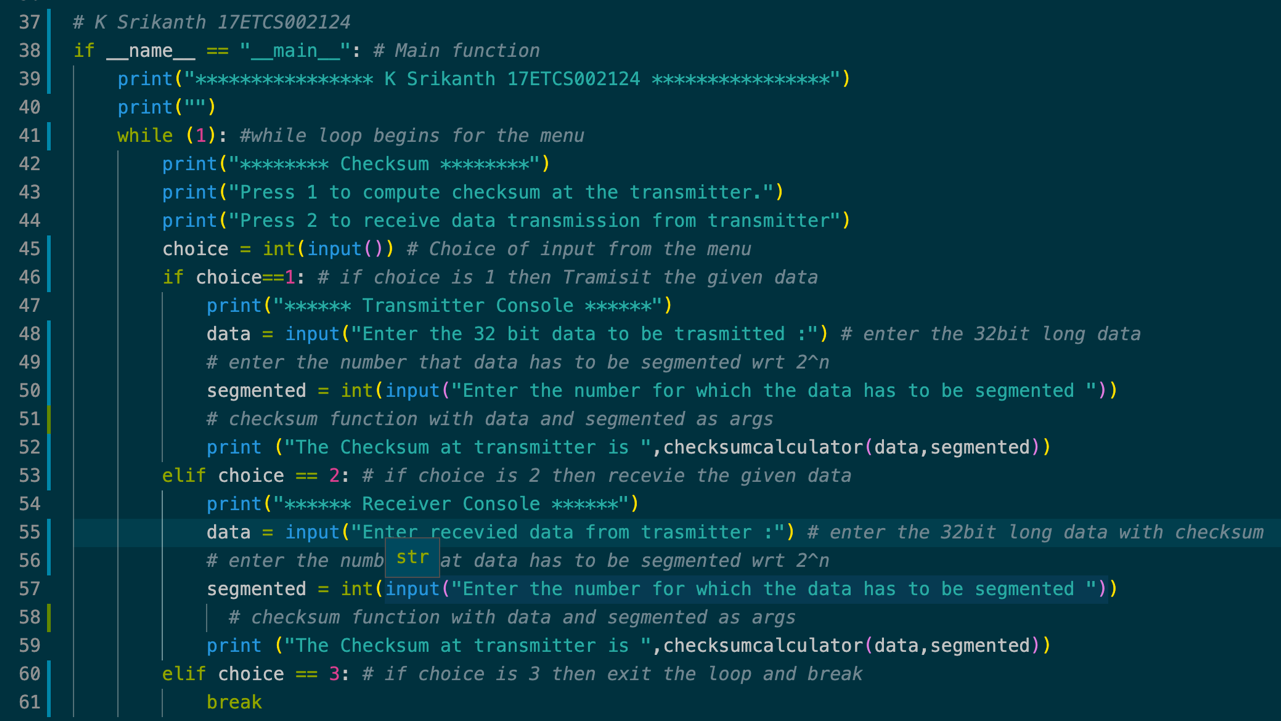
****

Figure 6 Main Function for given problem statement in python

**Link for my code**

[Check Sum Python](https://github.com/Srikanth-Kandarp/Labs-Code-5th-Sem/blob/main/Networks/Codes/Assignment_1/Assignment.py)

**To run this python code**

Python3 filename.py **<- Unix Distribution**

Filename.py **<- Windows Distribution**

**Result**

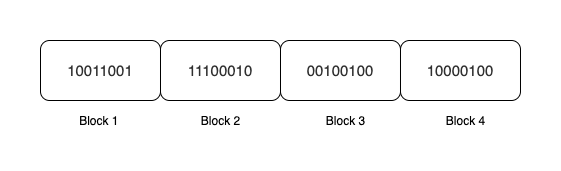
**Test Case 1**

**Sender Side**

**Let’s take input 32-bit stream that has to be transmitted**

10011001111000100010010010000100

**Let’s divide the input but stream into segments of 8 bits**

****

Now the Number of Blocks is 4 why so 32 / 8 = 4 Blocks

**Now we perform binary addition of all the four blocks**

**10011001**

**11100010**

**00100100**

**10000100**

**+10**

**--------------**

**00100101 <- Addition Result**

Since there are two extra bits, we add them at last to make it 8 bits

**Now we perform 1’s compliment on addition result 00100101 and it would be 11011010.**

**Check Sum = 11011010.**

**Now the 32-bit string along with checksum**

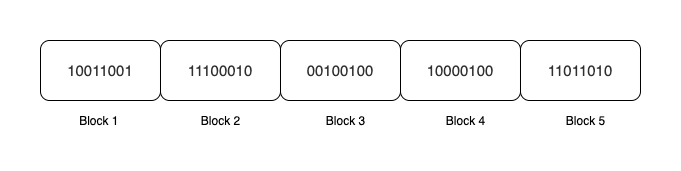
1001100111100010001001001000010011011010

**Receiver Side**

**Let’s take input 40-bit stream that has been received with check sum added**

1001100111100010001001001000010011011010

**Let’s divide the input but stream into segments of 8 bits**

****

Now the Number of Blocks is 5. why so 40 / 8 = 5 Blocks

**Now we perform binary addition of all the five blocks with check sum**

**00100101 <- Addition Result from sender side**

**11011010**

**--------------**

**11111111 <- Addition result on receiver side**

Now we perform 1’s compliment on addition result on receiver side **11111111** and it would be **0000000**. Which is an error free data

**Python Result**

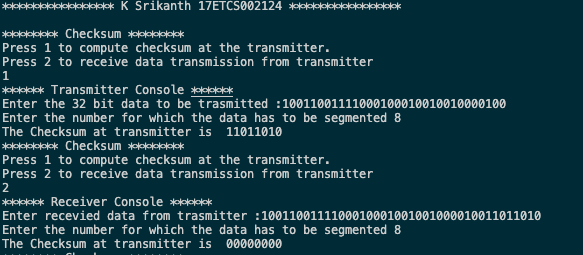
****

Figure 7 Python output console for Test case 1

**Test Case 2**

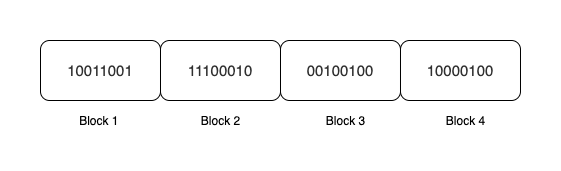
**Now let’s flip a bit and check for the error**

**Sender Side**

**Let’s take input 32-bit stream that has to be transmitted**

10011001111000100010010010000100

**Let’s divide the input but stream into segments of 8 bits**

****

Now the Number of Blocks is 4 why so 32 / 8 = 4 Blocks

**Now we perform binary addition of all the four blocks**

**10011001**

**11100010**

**00100100**

**10000100**

**+10**

**--------------**

**00100101 <- Addition Result**

Since there are two extra bits, we add them at last to make it 8 bits

**Now we perform 1’s compliment on addition result 00100101 and it would be 11011010.**

**Check Sum = 11011010.**

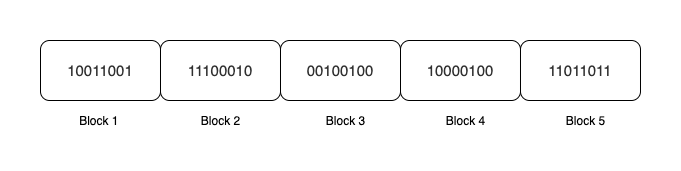
**So, I am going to change the check sum last bit to 1 and that would be 11011011.**

**Receiver Side**

**Let’s take input 40-bit stream that has been received with check sum added**

1001100111100010001001001000010011011011

**Let’s divide the input but stream into segments of 8 bits**

****

Now the Number of Blocks is 5. why so 40 / 8 = 5 Blocks

**Now we perform binary addition of all the five blocks with check sum**

**00100101 <- Addition Result from sender side**

**11011011 <- Check sum**

**--------------**

**00000001 <- Addition result on receiver side**

Now we perform 1’s compliment on addition result on receiver side **00000001** and it would be **11111110** Which is an error data for a successful error free data when you do check sum the data bits have to be 0’s

**Python Result**

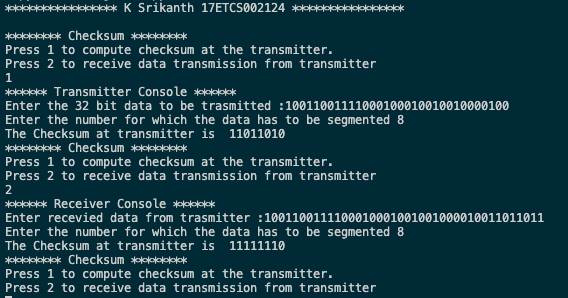
****

Figure 8Python output console for Test case 2