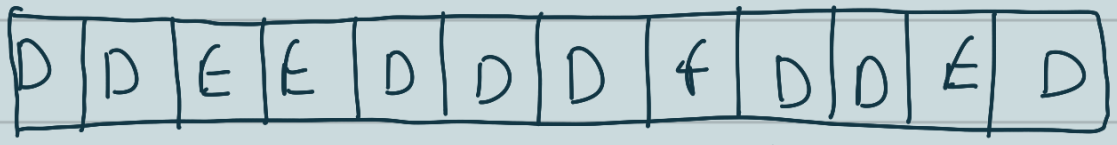
Data communication and networking

HW - II

Name: Basil Khowaja
id: Bko8432

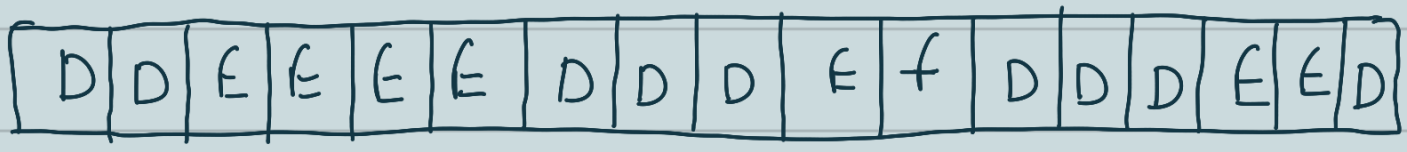
Q1) a) Given frame Payload:-



⇒ now we will see when a flag byte occurs in the data then we will add Esc byte before it, also in this way if we see that an Esc byte is present then we will add an additional Esc before it too.

↓

byte-Stuffed frame Payload:-



b) Given flag bits = 01111110

Given frame:-

000111111100111110100011111111111000111

Ans//:

⇒ now for bit stuff we will see that whenever five consecutive 1's are present in the data, then we will insert a 0 after it to avoid confusion with the flag sequence given = 01111110

So the frame will be:

00011111011001111101000111110111100001111

Q2) a) Show the generation of the CRC-32 codeword at the Sender Site (using binary division) when the dataword is "ABC" in ASCII.

Ans//:- first of all converting ABC in Binary:

A = 01000001 C = 01000011

B = 01000010

⇒ combining:-

01000001 01000010 00100001 → ①

A B C

⇒ Since this is CRC-32, → so we will have to add 32 zeros to the ①.

⇒ divisor = 100000100110000010001110110111

⇒ doing division here in textual format because of space constraint

011

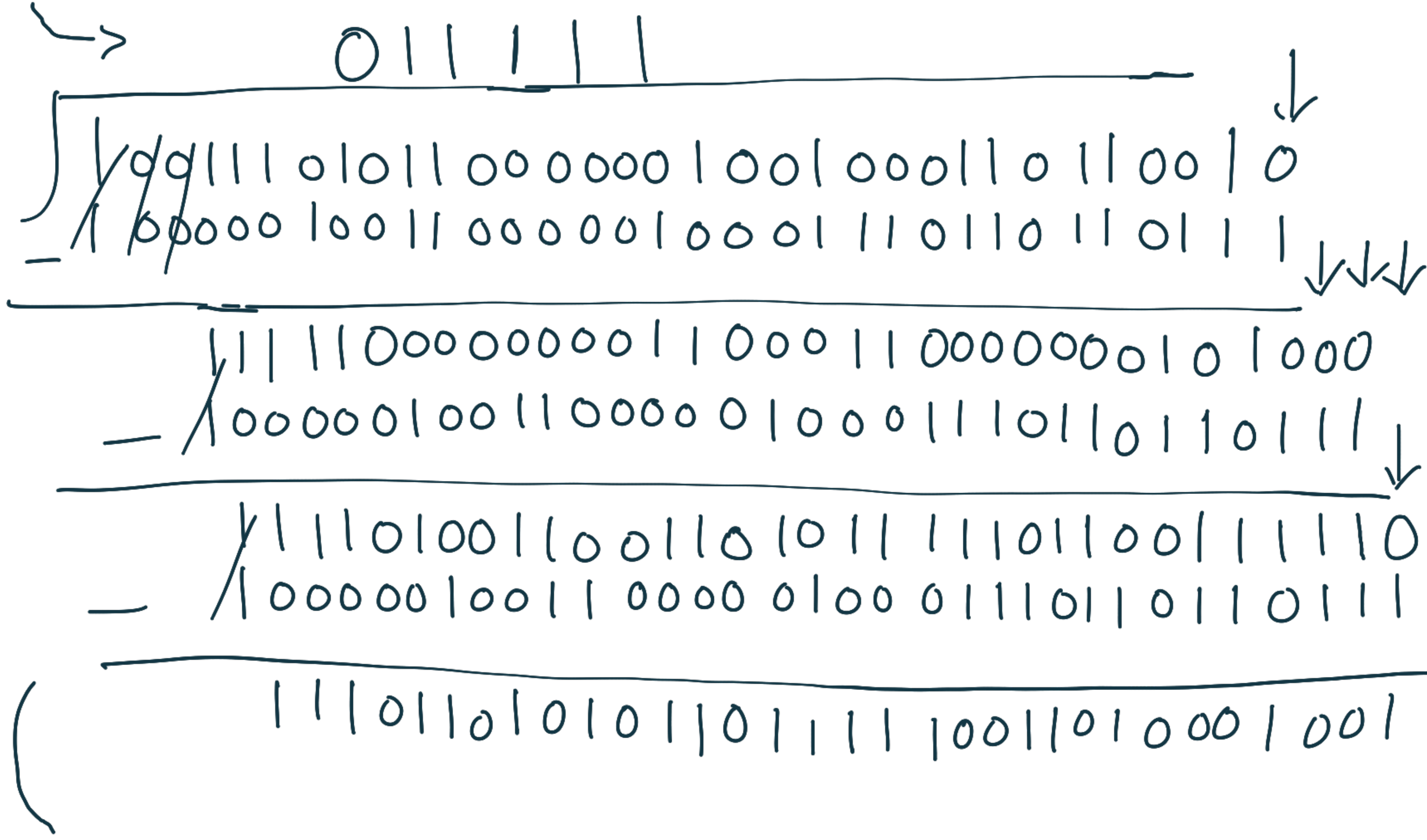
100000101000010010000110000000000
−000000000000000000000000000000000

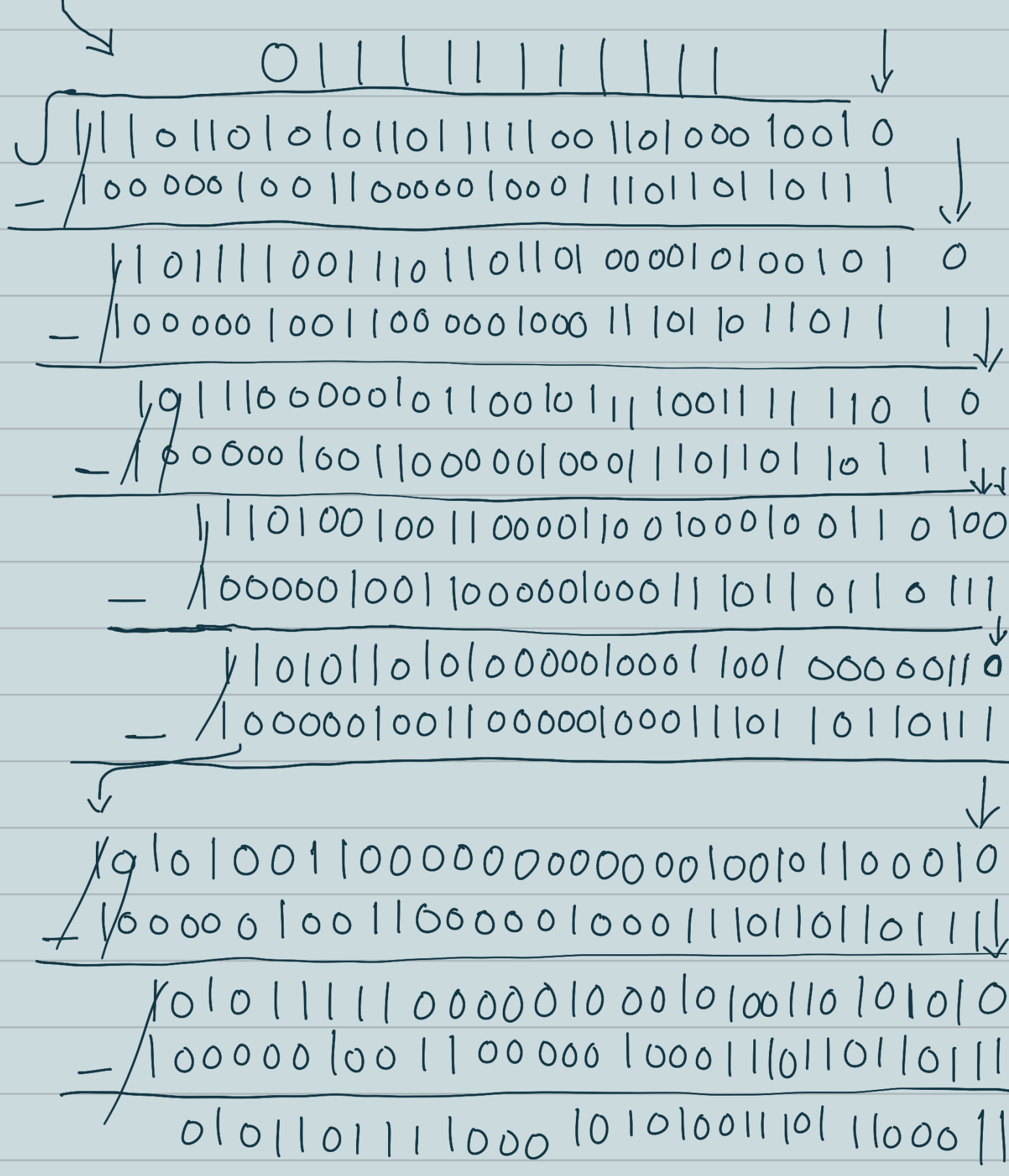
1000001010000100100001100000000 0
−100000100110000010001110110111 1

1100100000010001101101110000000
−100000100110000010001110110111

1001100110100001010101011011 0
−100000100110000010001110110111

100111010110000001000101011001





⇒ this approach was time consuming and I got errors in between this so wrote a basic Python code for calculating the remainder

```python
def remainder(dataword, divisor):
    data=list(map(int, dataword))
    divisor=list(map(int, divisor))
    #adding zeroes in last
    data.extend([0]*(len(divisor)-1)) #this will end zeroes (in our case will append 32 0's)
    for i in range(len(dataword)):    #running the loop for xor of every bit with divisor
        #since we only need to peform xor if the current bit in dataword is 1:
        if data[i]==1:
            for j in range(len(divisor)):
                data[i+j]= data[i+j] ^ divisor[j]

    #remainder will be the last bits after the division
    remainder=data[-(len(divisor)-1):]
    return ''.join(map(str, remainder))

dataword = '01000001010000101000011' #we got this dataword for ABC in asci-II
divisor = '100000100110000010001110110110111' #divisor which was given
remainder=remainder(dataword, divisor)
print("remainder is:", remainder)

```

↳ here I used simple level code for the purpose of calculating remainder.

Output remainder :-

```
PS C:\Users\Dell> & C:/Python311/python.exe "e:/fall 24/DCN/remainder.py"
remainder is: 10101111100000100010100110101010
PS C:\Users\Dell>
```

Remainder :-

10101111100000100010100110101010

⇒ Since final codeword:

Original dataword + remain

01000001010000100100001110101111100001000101001
10101010

b) for checking the Syndrome we will check that if we divide the codeword recieved by divisor then we get all 0's or not in remainder

if all 0's → codeword accepted

if not → codeword not accepted by reciever

```
> fall 24 > DCN > remainder.py > ...
1  def check_codeword(codeword, divisor):
2      data=list(map(int, codeword))
3      divisor=list(map(int, divisor))
4      for i in range(len(codeword)-len(divisor)+1):
5          if data[i]==1:
6              for j in range(len(divisor)):
7                  data[i + j]= data[i + j]^divisor[j]
8      syndrome=data[-(len(divisor)-1):]
9      return ''.join(map(str, syndrome))
10
11 #codeword which we got after combining the original dataword and the remainder
12 codeword = '01000001010000100100001110101111100001000101001101010101'
13 divisor = '10000010011000001000111011011011 1'
14 syndrome=check_codeword(codeword, divisor)
15 print("syndrome is:",syndrome)
16 #checking whether we are getting all 0's as remainder or not
17 if syndrome == '0' * (len(divisor) - 1):
18     print("codeword is accepted")
19 else:
20     print("codeword is rejected")
```

↳ I used the code which I already wrote in the Part (a), just removed the 0's padding part from the function definition since now we aren't dealing with the dataword which needed 32 0's to be added.

⇒ when I ran the code I got the remainder as all 0's.

calculated Syndrome:

```
PS C:\Users\Dell> & C:/Python311/python.exe "e:/fall 24/DCN/remainder.py"
syndrome is: 00000000000000000000000000000000
codeword is accepted
```

⇒ the recieved codeword is accepted by the reciever because the calculated Syndrome is all zeros which shows us that no errors were detected during the transmission. Also, this zero Syndrome implies that the transmitted dataword when appended with the remainder has not changed the Pattern and the data hence the codeword is accepted.

c) I inserted the given codeword Pn my Python code which will do the division Process and tell whether we have 0 Syndrome or not:

```
fall 24 > DCN > remainder.py > ...
1  def check_codeword(codeword, divisor):
2      data=list(map(int, codeword))
3      divisor=list(map(int, divisor))
4      for i in range(len(codeword)-len(divisor)+1):
5          if data[i]==1:
6              for j in range(len(divisor)):
7                  data[i + j]= data[i + j]^divisor[j]
8      syndrome=data[-(len(divisor)-1):]
9      return ''.join(map(str, syndrome))
10
11 #codeword which we got after combining the original dataword and the remainder
12 codeword = '1000010010001010110000100101101001011101101000011001111011'
13 divisor = '10000010011000001000111010110111'
14 syndrome=check_codeword(codeword, divisor)
15 print("syndrome is:",syndrome)
16 #checking whether we are getting all 0's as remainder or not
17 if syndrome == '0' * (len(divisor) - 1):
18     print("codeword is accepted")
19 else:
20     print("codeword is rejected")
```

```
PS C:\Users\Dell> & C:/Python311/python.exe "e:/fall 24/DCN/remainder.py"
syndrome is: 00110001100111101110010101110100
codeword is rejected
```

=> Since we can see that the Syndrome is not zero hence the codeword will be rejected by the reciever

Q3) 1) Pure aloha :-

$$S = G \cdot e^{-2G}$$

↓

avg number of transmission attempts per frame time

$\Rightarrow$ differentiating S with respect to G and making equal to zero

$$\frac{dS}{dG} = \frac{d}{dG}\left(G \cdot e^{-2G}\right)$$

using Product rule of derivative.

$u = G \quad v = e^{-2G}$

$$\frac{dS}{dG} = u' \cdot v + v' \cdot u$$

$$u' = \frac{d}{dG}(G) = 1$$

$$v' = \frac{d}{dG} e^{-2G} = -2 \cdot e^{-2G}$$

$$\frac{ds}{dG} = 1 \cdot e^{-2G} + G \cdot (-2) \cdot e^{-2G}$$

$$\frac{ds}{dG} = e^{-2G} \cdot (1 - 2G)$$

$$e^{-2G} \cdot (1 - 2G) = 0$$

$$1 - 2G = 0$$

$$\boxed{G = 1/2} \rightarrow \text{condition for max throughput}$$

2) Slotted aloha :-

$$S = G \cdot e^{-G}$$

$$\frac{ds}{dG} = \frac{d}{dG}(G \cdot e^{-G})$$

$$u = G \quad , \quad v = e^{-G}$$

$$u' = \frac{d}{dG}(G) = 1$$

$$v' = \frac{d}{dG}(e^{-G}) = -e^{-G}$$

$$\frac{ds}{dG} = 1 \cdot e^{-G} + G \cdot (-1) \cdot e^{-G}$$

$$\frac{ds}{dG} = e^{-G} \cdot (1 - G) = 0$$

$$1 - G = 0$$

$$\boxed{G = 1}$$

$\Downarrow$

condition for max throughput

$\Rightarrow$ Pure aloha is random access Protocol where the device transmits data whenever it has information to Send and it doesn't waits for a specific time Slot, if for example lets Say two devices transmit at the same time a collision will occur and both messages will be lost and then each device will retransmit after the random delay. while the Slotted aloha improves on the Pure aloha by dividing time into

slots, the devices are only allowed to transmit the data at the beginning of each time slot.

Pure aloha → max throughput → 18.4%

Slotted aloha → " → 36.8%

Q7) a) Sum of all words

$= 4500 + 0073 + 0000 + 4000$

$$\begin{array}{r} 4500 \\ +\ 0073 \\ \hline 4573 \end{array}$$

↓

$$\begin{array}{r} 4573 \\ +\ 0000 \\ \hline 4573 \end{array}$$

↓

$$\begin{array}{r} 4573 \\ +\ 4000 \\ \hline 8573 \end{array}$$

→ one's complement:-

$(8573)_{16}$

↓

1000 0101 0111 0011

↓

$(7A8C)_{16}$ ← 0111 1010 1000 1100

Checksum = $(7A8C)_{16}$

b) message that the reciever gets :-

words + checksum

4500 0073 0000 4000 7A8C

↓

Sum = 8573 (done earlier)

$$\begin{array}{r} 8573 \\ +\ 7A8C \\ \hline FFFF \end{array}$$

↓

1111 1111 1111 1111

↓

0000 0000 0000 0000

↓

$(0000)_{16}$

Hence the recieved Packet will be error free

c) the recalculated Syndrome at reciever's end is $(8573)_{16}$

adding it to the checksum sent by the Sender $\rightarrow (7A8C)_{16}$

we get,

$$(FFFF)_{16}$$

$\Rightarrow$ the Syndrome value has all bits equal to 1 which means there is no error and the message is Perfect match.