Question 1: Let 01111110 be the FLAG. To transmit the flowing bit string 011110000111110001111110 at the data link layer, what is the string transmitted after bit stuffing?

Solution.

Bit stuffing is used in data transmission to ensure proper synchronization and to distinguish between data bits and control bits. In this context, bit stuffing is employed to prevent the occurrence of consecutive sequences that might be mistaken for control flags, such as the FLAG sequence 01111110. When transmitting the bit string 011110000111110001111110, bit stuffing ensures that the transmitted string contains the necessary additional bits to avoid consecutive sequences of 1s. In this case, after bit stuffing, the transmitted string becomes 01111000011111000011111010, where an extra 0 is inserted after every sequence of five consecutive 1s, ensuring that the FLAG sequence is not inadvertently detected within the data.

01111000011111000011111010, where reds are consecutive “1”, after 5 consecutive “1” there’s a blue marked 0.

Question 2: The following character encoding is used in a data link protocol:

A: 01111000	**B: 01111100**	**FLAG: 01111110**	**ESC: 11100000**

Show the bit sequence transmitted (in binary) for the four-character frame: “A B ESC FLAG” when FLAG bytes with byte stuffing is used.

Solution.

ESC characters used in byte stuffing are distinct from control characters and are specifically employed to ensure that some of the bit sequences and flags within the data frame are correctly interpreted as part of the transmitted data, not as control sequences or characters by the receiver. Here, the four-character frame is part of the bit sequence which includes a byte stuffing ESC. When we are transmitting the bit sequence, we are making sure that the receiver understands that the byte stuffing ESC symbol is part of the data sequence itself, but not part of the control sequences. For that, we are adding a control sequenced ESC before the byte stuffing ESC. We are also adding the control sequenced ESC before FLAG.

Here the bit sequence is A B ESC FLAG &
The transmitted sequence is A B **ESC** ESC **ESC** FLAG.

Following the character encoding used in the data-link protocol, the transmitted bit sequence which includes the data frame is:

01111000 011111100 11100000 11100000 11100000 01111110

(Concatenated) 0111100001111110011100000111000001110000001111110

Please Turn Over

Question 3: An 8-bit byte with binary value 10101100 is to be encoded using Hamming code. What is the binary value after encoding? [Hint: use 4 check bits]

Solution.

Given, the binary value is 10101100, which is an 8-bit binary value. We know 8-bit sequence can contain 4 check bits (2^0=0, 2^1=2, 2^2=4, 2^3=8). We can also check if the number of check bits (r) and data bits (m) follows the relation mentioned in the textbook as:

m+r+1 ≤ 2^r
or, 8+4+1 ≤ 2^4
or, 13 ≤ 16 *which satisfies the condition for the number of checkbits = number of parity bits.*

Bit Position	1	2	3	4	5	6	7	8	9	10	11	12
Binary Before Encoding	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

From the Binary Values before encoding, we get this table:

Seq.	Binary Table			
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Check Bit Positions	XOR
Check bit positions for 1: **3**, **5**, **7**, **9**, **11**	0
Check bit positions for 2: **3**, **6**, **7**, **10**, **11**	1
Check bit positions for 4: **5**, **6**, **7**, **12**	0
Check bit positions for 8: **9**, **10**, **11**, **12**	1

From this stipulation, we have:

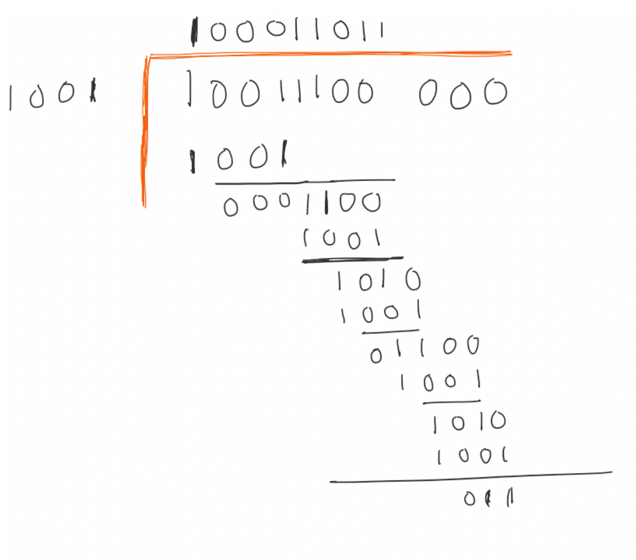
Bit Position	1	2	3	4	5	6	7	8	9	10	11	12
Binary Before Encoding	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
Data Bits			1		0	1	0		1	1	0	0
Check Bits	0	1		1				0				
Binary After Encoding	0	1	1	1	0	1	0	0	1	1	0	0

Please Turn Over

Question 4: A bit stream 10011100 is transmitted using the standard CRC method. The generator polynomial is $x^3 + 1$.

a) Show the actual bit string transmitted.

Solution.



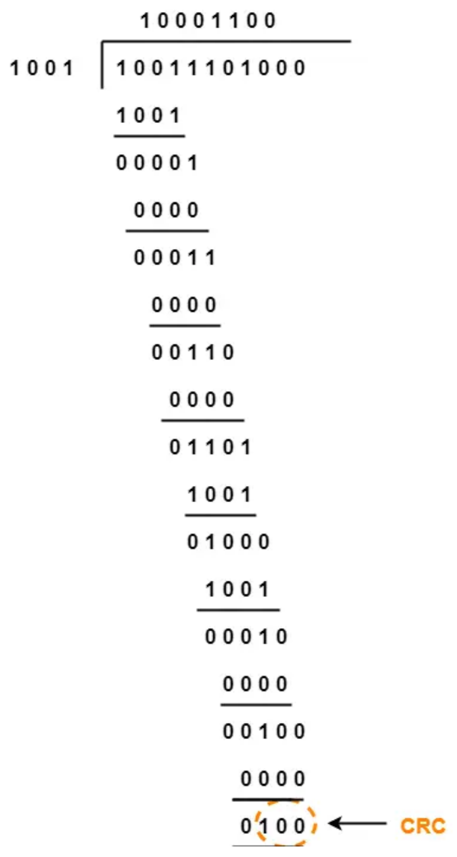
Dividing 10011100 by $x^3 + 1$ which is an equivalent of $1x^3 + 0x^2 + 0x^1 + 1$ or, 1001

We get the CRC as **011**.

Therefore, the bit string transmitted is **10011100011**.

b) Suppose one bit is inverted during transmission. Show that this error is detected at the receiver's end.

Solution.



Let's assume that in the transmitted bit sequence (10011100) the first bit from the right gets inverted (10011101). Dividing it by 1001 will give us a CRC of **0100**.

We know that for the transmission to be detected we use the CRC method where we use Cyclic Redundancy Check (CRC) to validate our transmitted bit sequence. This involves appending a checksum to the transmitted data based on polynomial division. At the receiver's end, the received data is divided by the same polynomial. If the remainder is zero, the transmission is deemed error-free.

From our worked-out example, we can easily see that our CRC is not 0. This is how the error is detected on the receiver's end.

Please Turn Over

5. With the 2-D parity method, a block of bits with n rows and k columns uses horizontal and vertical parity bits for error correction or detection. During the class, we have shown that 2-D parity method can correct single bit errors.

(a) Whether the 2-D method can detect double errors? Triple errors?

Solution.

The 2-D parity method can detect double errors as well as triple errors with some exceptions.

- o If two bits are inverted in the same row, the parity checks for that row and column will fail, indicating an error.
- o If two bits are inverted in the same column, and both bits are opposite to each other (one is a 1, other is a 0) then the chance of getting undetected arises.
- o If three bits are inverted, chances of detection are highly probable with some exceptions.

Answer. 2D parity method can detect both double and triple inversions.

(b) Find an example of a pattern of six errors that cannot be detected using 2-D parity method.

Solution.

<>	<>	<>	<>	<>
<>	1	1	0	<>
<>	1	0	1	<>
<>	0	1	1	<>
<>	<>	<>	<>	<>

In this data sequence, <> signifies negligible bits. Errors in these consecutive rows and columns will not be detectable since the resultant number of odds and even numbers will be equal to the number before the inversion of the bits.