Telecommunication Networks
15B11EC611

Data Link Layer

- This PPT is containing the discussion of Error Detection and Correction and Framing of Data Link Layer.

- Kindly refer page numbers: 267 to 271, 284 to 293, and 307 to 310 of the Book_1_Data-Communications-and-Networking - By Forouzan for detailed discussion.
- Kindly note: For topics, follow this PPT, and for detailed discussion of those topics, follow the book.

Data Link Layer

Specific responsibilities of the data link layer include *framing, addressing, flow control, error control, and media access control.*

Error Detection and Correction

- **Data can be corrupted during transmission**

Types of Errors

1. ***Single-Bit Error***
   - **only 1 bit of a given data unit is changed from 1 to 0 or from 0 to 1**
2. ***Burst Error***
   - **2 or more bits in the data unit have changed.**

Error Detection and Correction

Redundancy

- **The central concept in detecting or correcting errors is redundancy.**

❖ **To detect or correct errors, we need to send extra (redundant) bits with data.**

Coding

- **Redundancy is achieved through various coding schemes.**

- **We can divide coding schemes into two broad categories:**
  1. **block coding and**
  2. **convolution coding.**

Error Detection and Correction

Modular Arithmetic

- **In modular arithmetic, we use only a limited range of integers.**

modulo-N arithmetic

- **use only the integers 0 to *N - I, inclusive***
- **Addition and subtraction in modulo arithmetic are simple.**
- **There is no carry when you add two digits in a column.**
- **There is no carry when you subtract one digit from another in a column.**

Adding:	**0+0=0**	**0+1=1**	**1+0=1**	**1+1=0**
Subtracting:	**0-0=0**	**0-1=1**	**1-0=1**	**1-1=0**

- **In this arithmetic we use the XOR (exclusive OR) operation for both addition and subtraction.**

BLOCK CODING

- **In block coding, we divide our message into blocks, each of *k bits, called datawords.***
- ***We* add *r redundant bits to each block to make the length n = k + r.***
- ***The resulting n-bit blocks* are called codewords.**

- **Dataword size = k**
- **Codeword size = n**

- **A coding scheme C is written as C (n, k)**

LINEAR BLOCK CODES

- **a linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.**

CYCLIC CODES

- Cyclic codes are special linear block codes
- In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.
- For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.

Cyclic Redundancy Check

- This is a category of cyclic codes called the cyclic redundancy check (CRC)

➢ Dataword size = k bits

➢ Codeword size = n bits

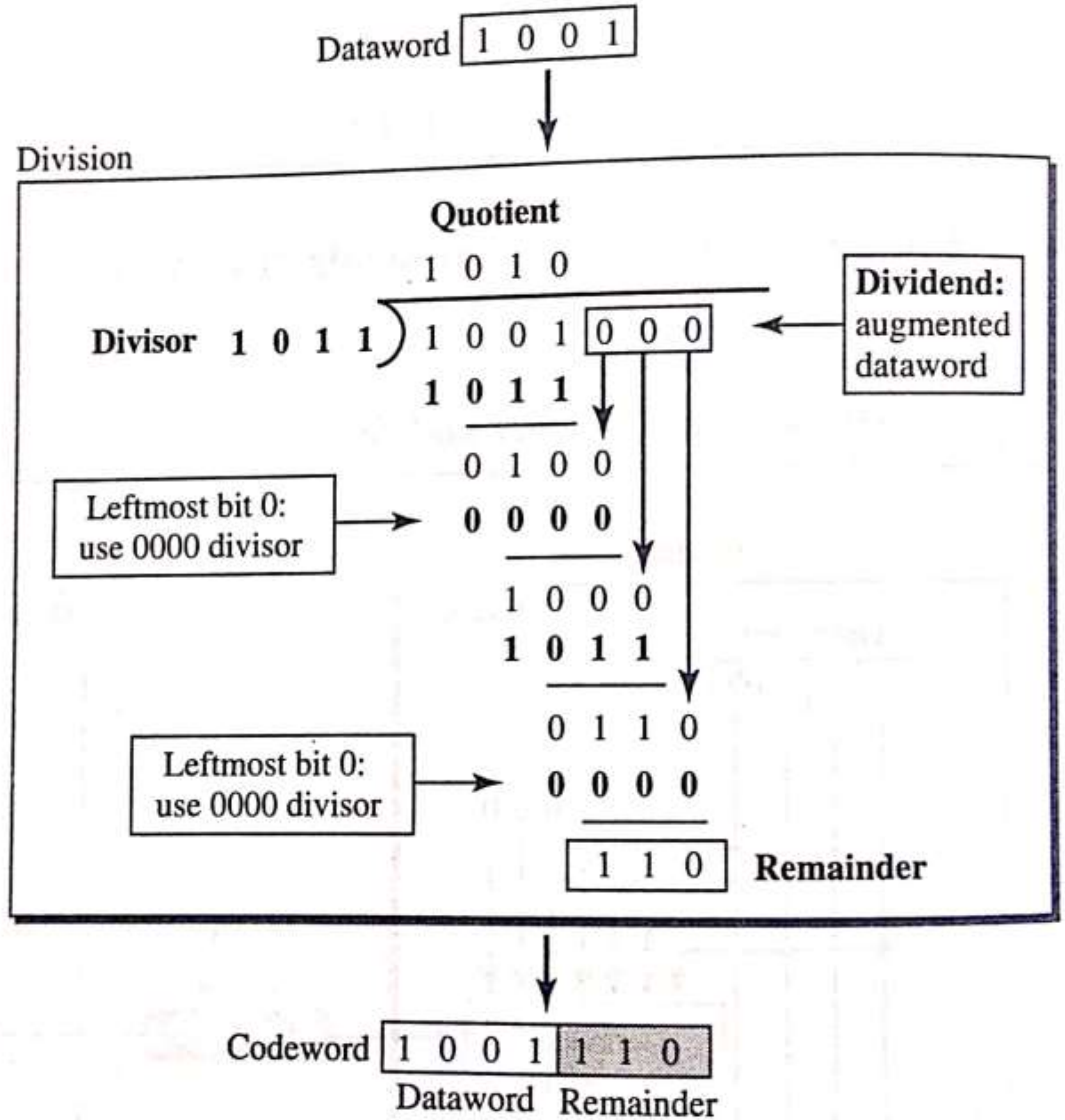
➢ The size of the dataword is augmented by adding n - k 0s to the right-hand side of the word.

➢ Divisor size = n – k + 1

Cyclic Redundancy Check

- **The generator divides the augmented dataword by the divisor (modulo-2 division).**
- **The quotient of the division is discarded; the remainder *($r_2r_1r_0$) is appended to the dataword to create the codeword.***
- **There is one important point we need to remember in this type of division. If the leftmost bit of the dividend is 0, we need to use an all-0s divisor.**
- **When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits (r_2, r_1, and r_0). They are appended to the dataword to create the codeword.**

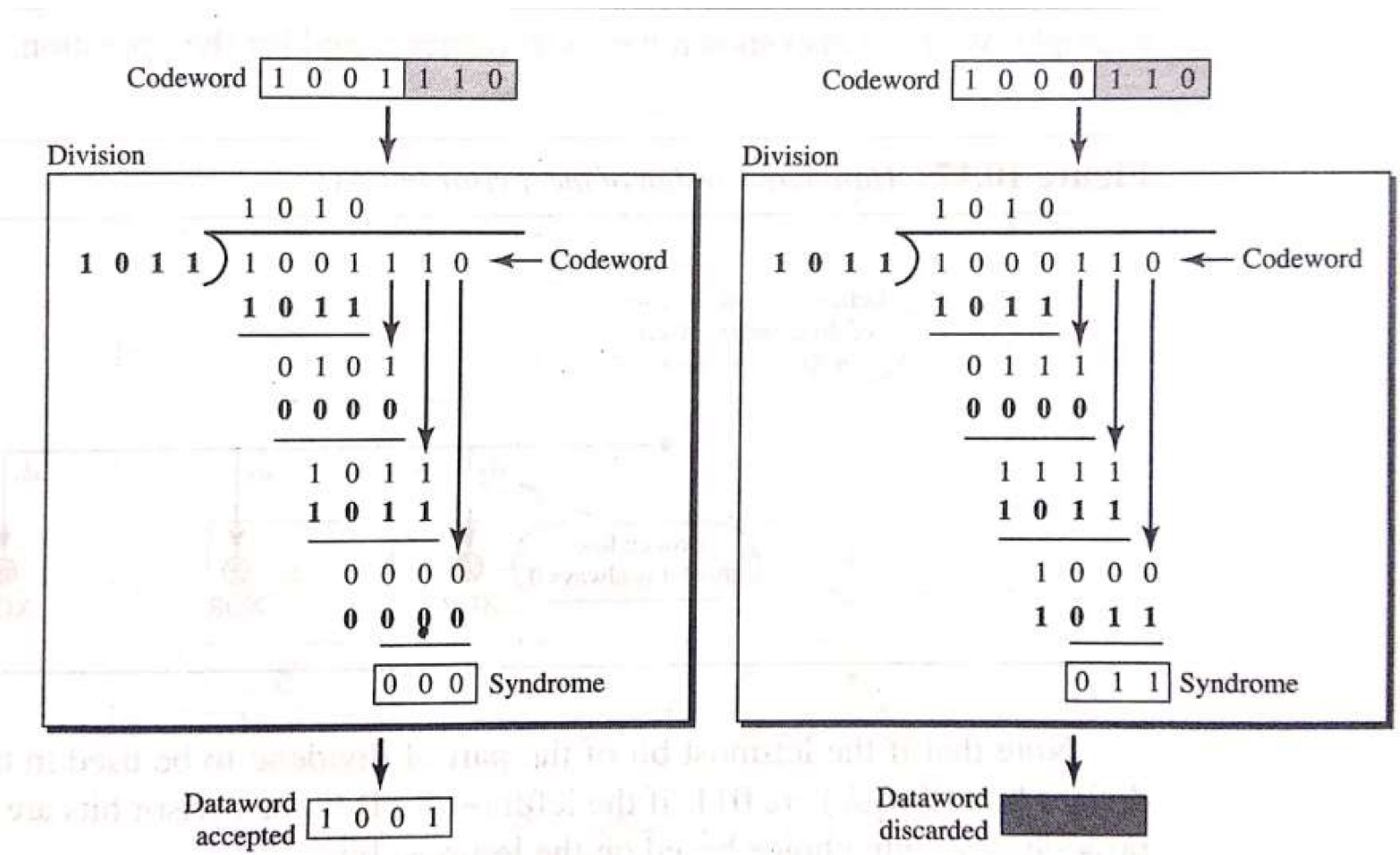
Encoder



Dataword 1 0 0 1
Division
Quotient
1 0 1 0
Divisor 1 0 1 1
1 0 0 1 0 0 0
1 0 1 1
0 1 0 0
0 0 0 0
1 0 0 0
1 0 1 1
0 1 1 0
0 0 0 0
1 1 0
Remainder
Dividend: augmented dataword
Leftmost bit 0: use 0000 divisor
Leftmost bit 0: use 0000 divisor
Codeword 1 0 0 1 1 1 0
Dataword Remainder

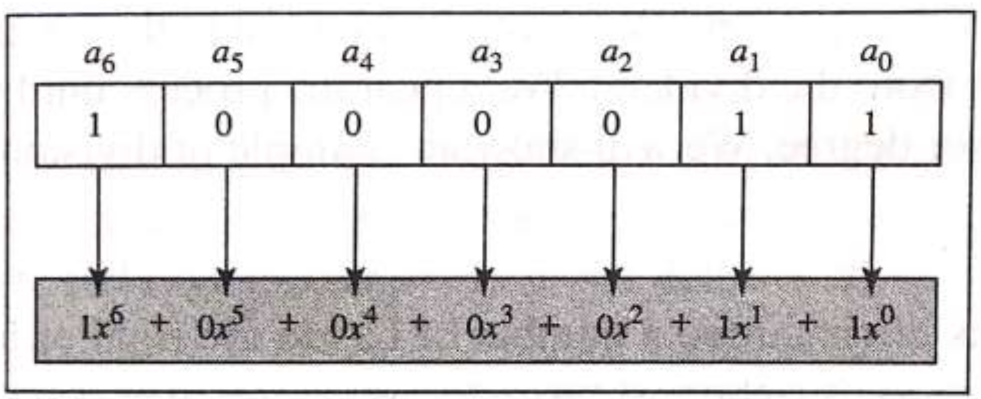
Decoder

- **The codeword can change during transmission.**

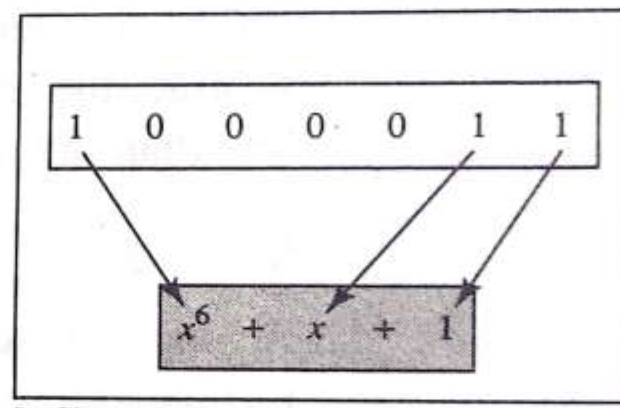


Polynomials

- A pattern of Os and 1s can be represented as a polynomial with coefficients of 0 and 1.
- The power of each term shows the position of the bit



a. Binary pattern and polynomial



b. Short form

- Figure shows one immediate benefit; a 7-bit pattern can be replaced by three terms

Degree of a Polynomial

- **The degree of a polynomial is the highest power in the polynomial.**
- **For example, the degree of the polynomial *$x^6 + x + 1$ is 6.***
- ***Note that the degree of a polynomial is 1 less* that the number of bits in the pattern. The bit pattern in this case has 7 bits.**

Adding and Subtracting Polynomials

- **addition and subtraction are the same**
- **adding or subtracting is done by combining terms and deleting pairs of identical terms.**

- **For example, adding *$x^5 + x^4 + x^2$ and $x^6 + x^4 + x^2$ gives* just *$x^6 + x^5$.***
- ***The terms x^4 and x^2 are deleted.***

➢ **note that if we add three polynomials and we get *x^2 three times, we delete a pair of them and keep the third.***

Multiplying or Dividing Terms

Multiplying Two Polynomials

- **Each term of the first polynomial must be multiplied by all terms of the second.**
- **The pairs of equal terms are deleted.**
- **The following is an example:**

$$
\begin{aligned}
&(x^5 + x^3 + x^2 + x)(x^2 + x + 1) \\
&\quad = x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^3 + x^2 + x \\
&\quad = x^7 + x^6 + x^3 + x
\end{aligned}
$$

Shifting

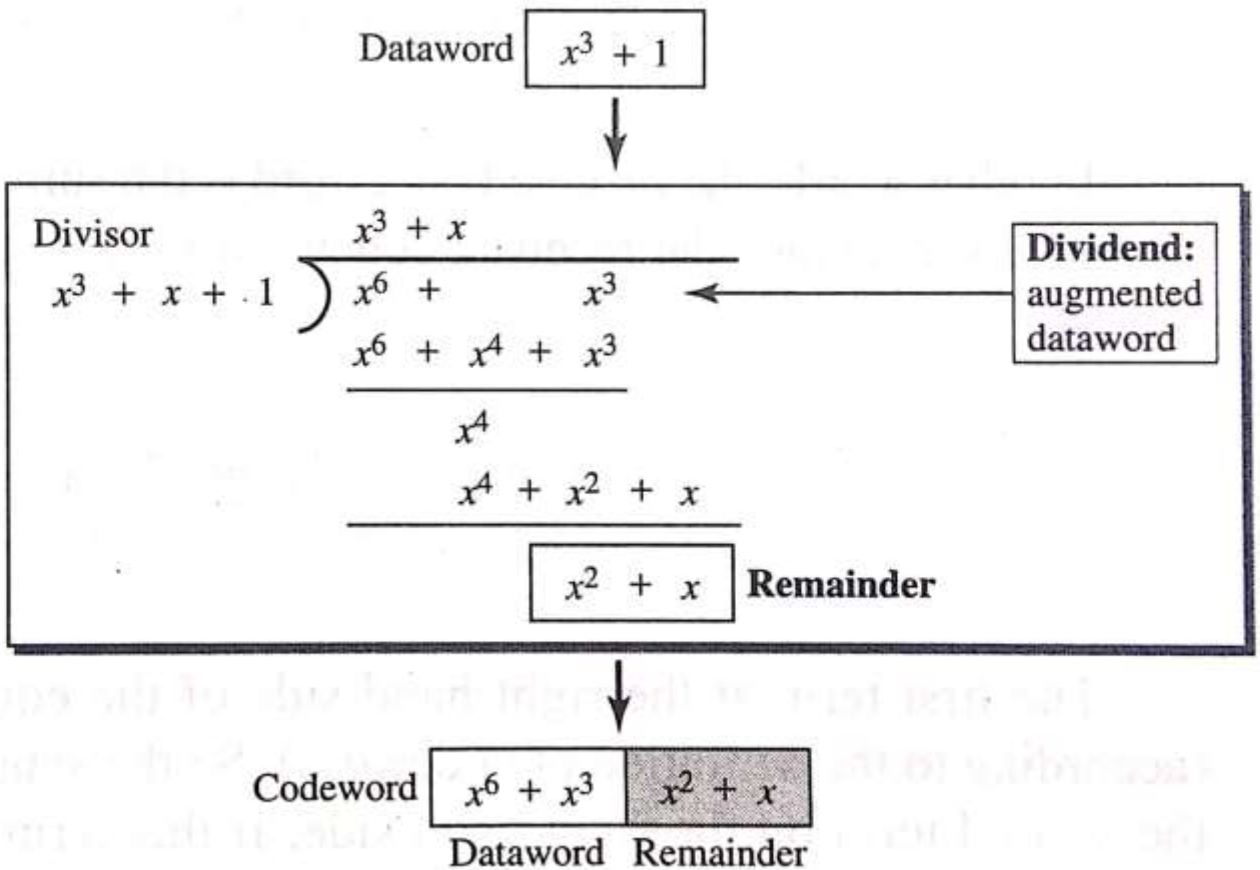
- **A binary pattern is often shifted a number of bits to the right or left.**
- **Shifting to the left means adding extra 0s as rightmost bits;**
- **shifting to the right means deleting some rightmost bits.**
- **Shifting to the left is accomplished by multiplying each term of the polynomial by** x^m ***where m is the number of shifted bits;***
- ***shifting to the right is accomplished by*** **dividing each term of the polynomial by** x^m**.**

Shifting left 3 bits: 10011 becomes 10011000 $x^4 + x + 1$ ***becomes*** $x^7 + x^4 + x^3$

Shifting right 3 bits: 10011 becomes 10 $x^4 + x + 1$ ***becomes*** x

CRC division using polynomials

- The process is shorter
- The dataword 1001 is represented as $x^3 + 1$.
- *The divisor* 1011 is represented as $x^3 + x + 1$.
- *To find the augmented dataword, we have left-shifted* the dataword 3 bits (multiplying by x^3). *The result is* $x^6 + x^3$



Note: we continue to divide until the degree of the remainder is less than the degree of the divisor.

Data Link Layer

- In broad sense, the functions of the data link layer are:
  1. data link control, and
  2. media access control.

- Data link control functions include framing, flow and error control, and software implemented protocols, that provide smooth and reliable transmission of frames between nodes.

FRAMING

- Framing in the data link layer separates a message from one source to a destination, or from other messages to other destinations, by adding a sender address and a destination address.
- Destination address defines where the packet is to go.
- Sender address helps the recipient acknowledge the receipt.

FRAMING

- Frames can be of fixed or variable size.
  - **Fixed-Size Framing: no need for defining the boundaries of the frames**
  - **Variable-Size Framing: need to define the end of the frame and the beginning of the next.**
    - **Two approaches were used for this purpose:**

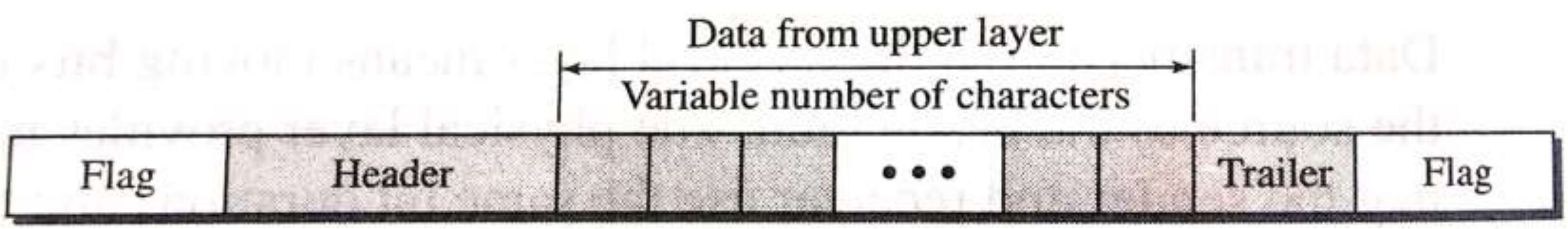
    1. **Character-oriented approach**
    2. **Bit-oriented approach.**

FRAMING

❑ *Character-Oriented Protocols*

- **Data to be carried are 8-bit characters**
- **The header (carries the source and destination addresses and other control information) and the trailer (carries error detection or error correction redundant bits) are also multiples of 8 bits.**
- **To separate one frame from the next, an 8-bit (1-byte) flag (0111 1110) is added at the beginning and the end of a frame.**

Figure shows the format of a frame in a character-oriented protocol



Note: Flag could be selected to be any character

FRAMING

- ***Character-Oriented Protocols***

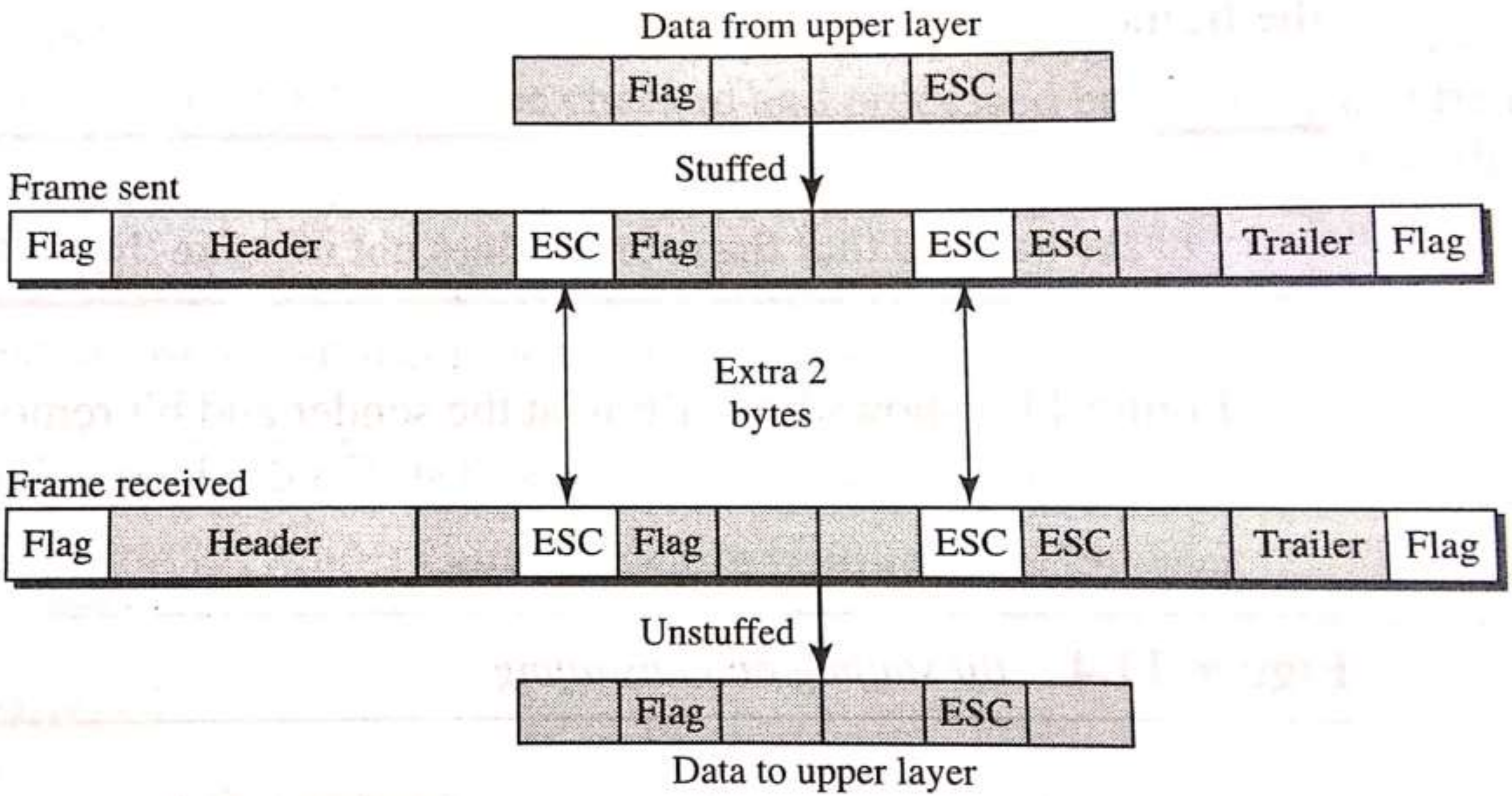
  - **Flag could be selected to be any character and any pattern used for the flag could also be part of the information.**
  - **If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame.**
  - **To fix this problem, a byte-stuffing strategy was added to character-oriented framing**

Byte stuffing (or Character stuffing)

- **Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.**
- **A special byte called the escape character (ESC) is added to the data section of the frame when there is a character with the same pattern as the flag.**
- **Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data**

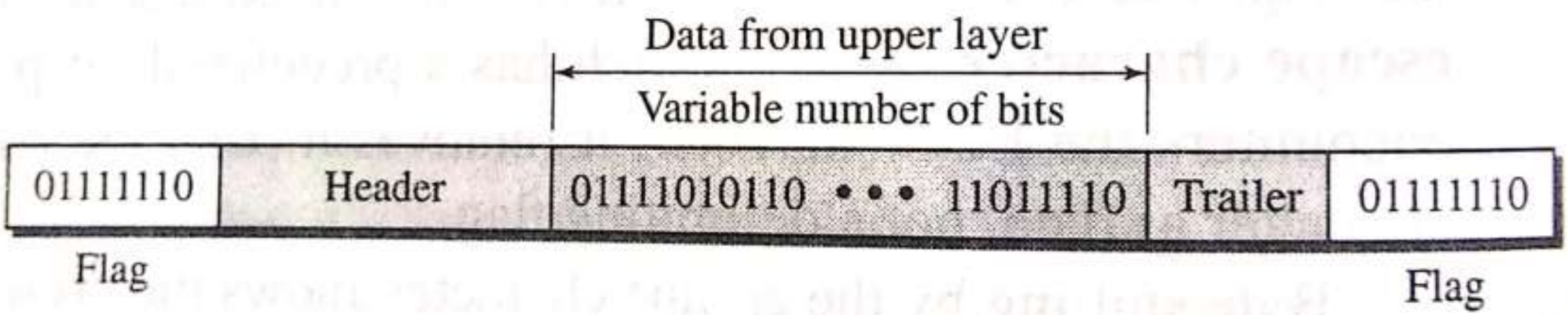
Byte stuffing and unstuffing

- The escape characters that are part of the text must also be marked by another escape character.



Bit-Oriented Protocols

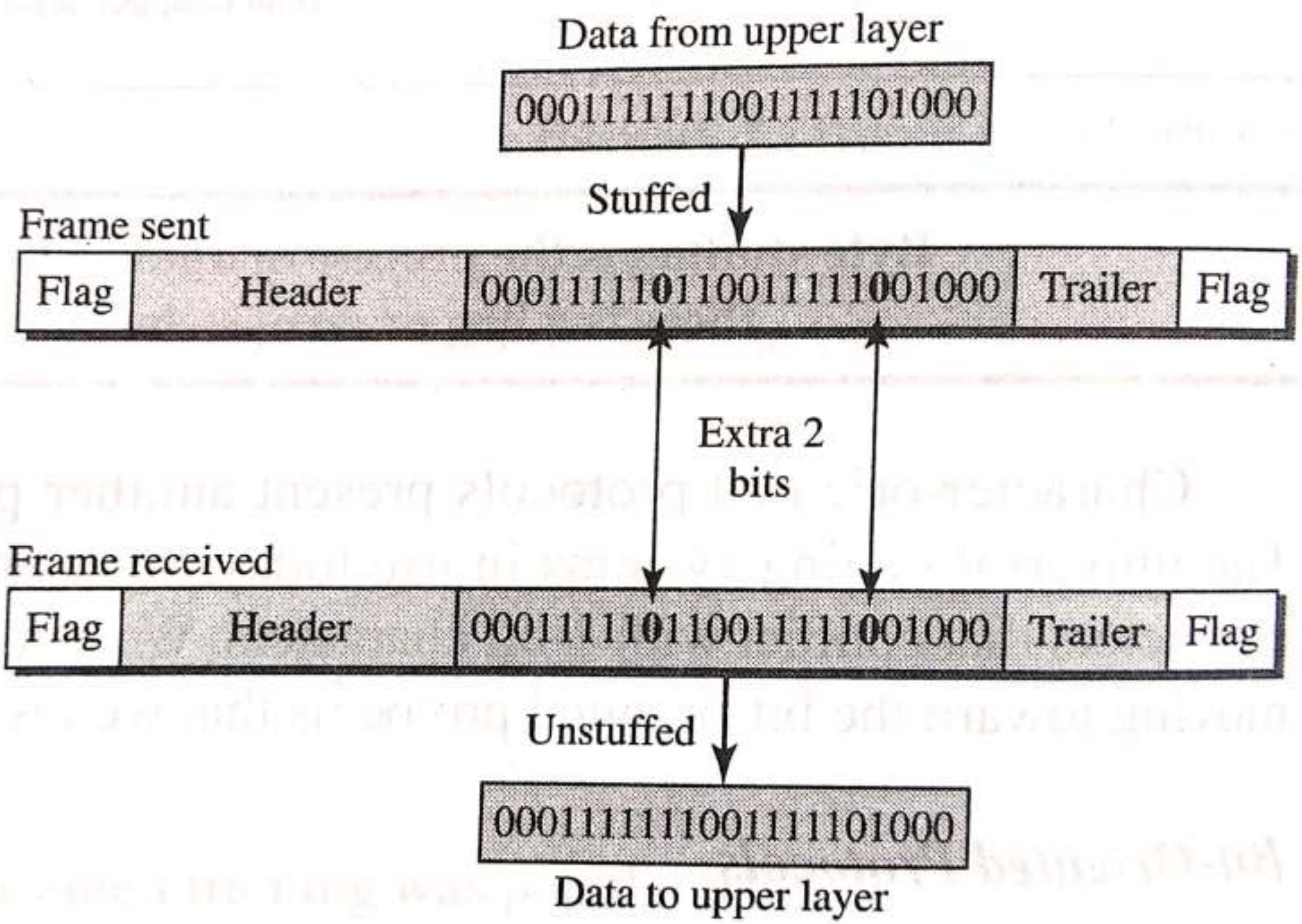
A frame in a bit-oriented protocol



- **This flag can create the same type of problem we saw in the byte-oriented protocols.**
- **That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame.**
- **We do this by stuffing 1 single bit to prevent the pattern from looking like a flag. The strategy is called bit stuffing.**

Bit Stuffing

❖ Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.



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

A: 01000111: B: 11100011: FLAG: 01111110: ESC: 11100000

Show the bit sequence transmitted (in binary) after bit stuffing for the four-character frame: A B FLAG ESC.

Example: A bit string, **0111101111101111110**, needs to be transmitted at the data link layer. What is the string actually transmitted after bit stuffing?

Example: **The following character encoding is used in a data link protocol:**

A: 01000111: B: 11100011: ESC: 11100000: FLAG: 01111110

Show the bit sequence transmitted (in binary) after bit stuffing for the four-character frame: A B FLAG ESC.

Solution:

Dataword: **01000111111000110111111011100000**

Codeword after bit stuffing: **01000111110100011011111010111100000**

Transmitted bit sequence after bit stuffing:

$$\underbrace{01111110}_{\text{FLAG}}\ 01000111110100011011111010111100000\ \underbrace{01111110}_{\text{FLAG}}$$

Example: A bit string, 0111101111101111110, needs to be transmitted at the data link layer. What is the string actually transmitted after bit stuffing?

Solution:

Transmitted bit sequence after bit stuffing

$\overbrace{01111110}^{\text{FLAG}}$ 011110 1111100 11111010 $\overbrace{01111110}^{\text{FLAG}}$

THANK YOU