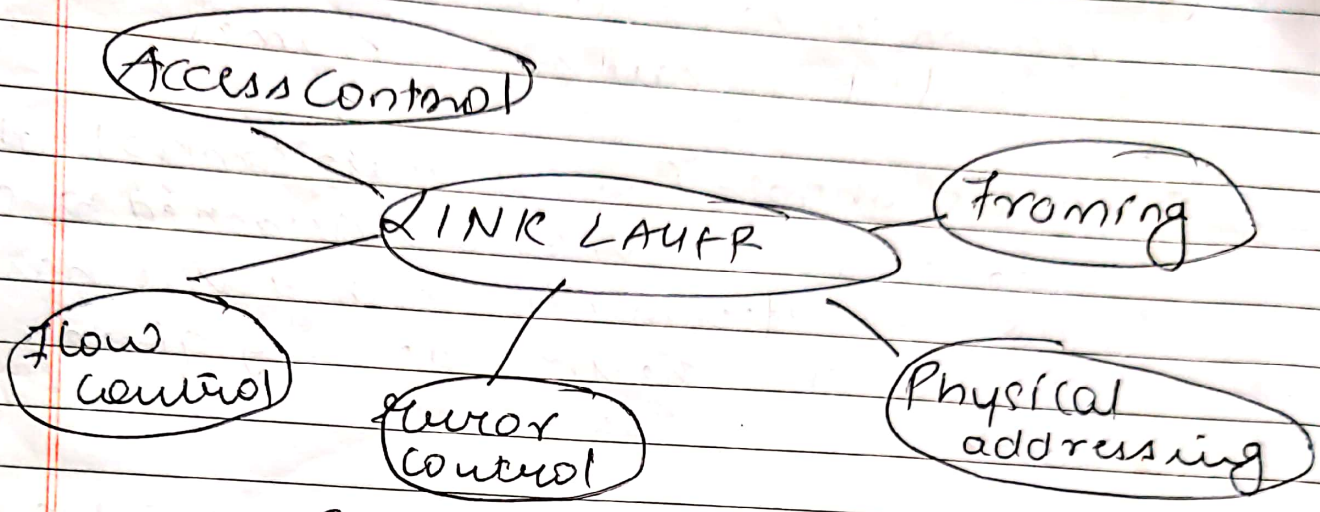
Data Link Layer.



> Second Layer of OSI reference model

> Communication at application transport and network layer is end to end whereas communication at data Link Layer is node to node

* Functionalities

① Framing - DLL receives data from network layer and divides it into manageable units called frames

Header	Packet	Trailer

Protocol data link

② Physical addressing - It provides addressing information by adding header to each frame.

Physical addresses of source and destination machine are added to each frame.

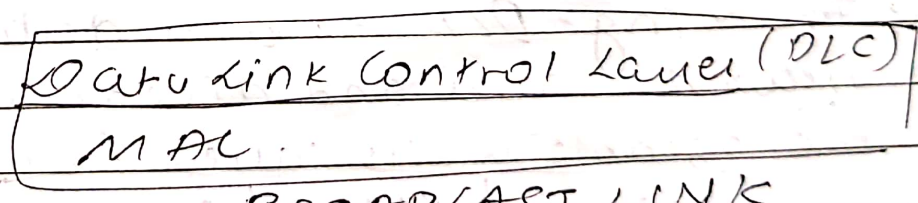
③ Flow control - Provides flow control mechanism to ensure that sender is not sending the data at speed receiver cannot process

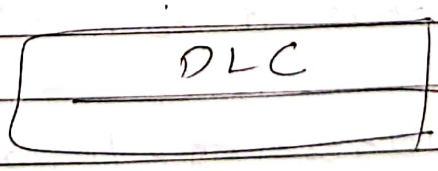
(4) Error control - It provides error control mechanism to detect and retransmit damaged duplicate or lost frames

(5) Access control - Provides access control when two or more devices are attached to same link. DLL determines which device has control over the link at any given point.

> Data link layer is divided into 2 sublayers
1) Data link control layer (DLC)
2) Media access control layer (MAC)

> DLC deals with all issues common to both point to point & broadcast links whereas MAC sublayer deals only with issues specific to broadcast links.





> DLC functions include framing, flow & error control, and error detection & error correction.

The DLL is divided into 2 sub layers
i) DL control ie DLC
ii) Media Access Control ie MAC
The DLC deals with all issues common to both point-to-point & broadcast links whereas the MAC sub layer deals only with issues specific to broadcast links

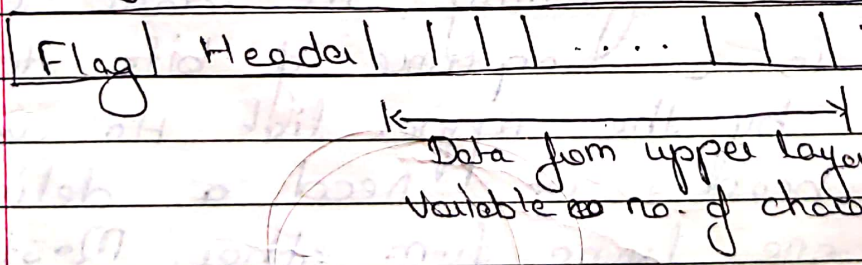
i) DLC SubLayer.
DLC functions include framing, flow & error control & error detection & correction.

A) Framing:
The DLL needs to pack bits into frames so that each frame is distinguishable from another. Framing separates a message from one source to a destination by adding a sender address & destination address. Frames can be

of fixed or variable size. In fixed size framing there is no need for defining the boundaries of frame, the size itself can be used as a delimiter.
In variable size framing we need a way to define the end of 1 frame & the beginning of next. Two approaches are used for this purpose:

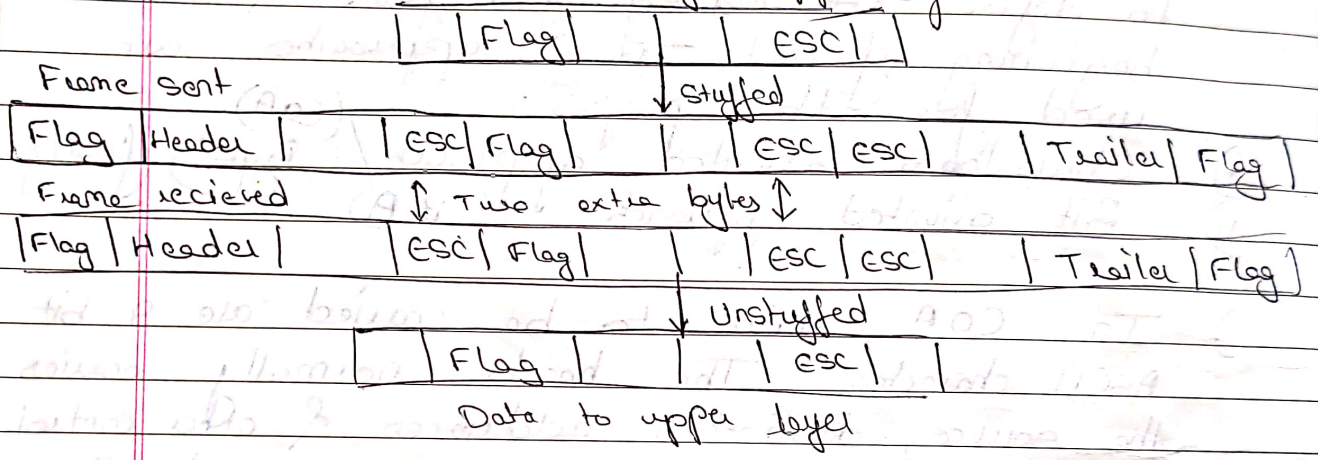
i) Character oriented approach (COA) / (Byte) framing
ii) Bit oriented approach (BOA)

i) In COA, data to be carried are 8 bit ASCII characters. The header normally carries the source, destination addresses & other control information & the trailer carries error detection bits. To separate one frame from the next 8 bit (1 byte) flag is added at the beginning & end of frame. The flag is composed of protocol dependent special characters that signals the start & end of frame.



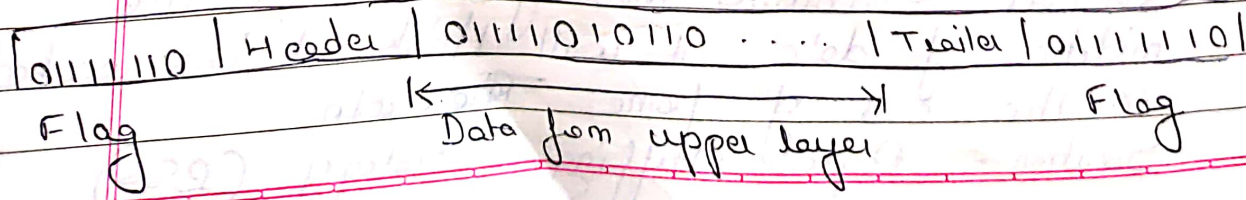
Sending info such as graphs, audio, video any pattern used for flags could also be a part of the info. In this case if the reciever encounters this pattern in the middle of data, it will think it has reached to the end of frame.
Solution - Byte stuffing Strategy (BSS)

In BSS, a special byte is added to data section of the frame when there is a char with the same pattern as the flag. This byte is usually called the ESC char & has a predefined bit pattern.



Whenever the reciever encounters the ESC char it removes it from the data section & treats the next char as data & not a delimiter flag.

ii) In Bit Oriented framing, the data section of a frame is a sequence of bits to be interpreted by the upper link. However in addition to headers, we need a delimiter to separate one frame from other. Most protocols use a special 8 bit pattern flag is 01111110 as the delimiter to define the beginning & end of frame.

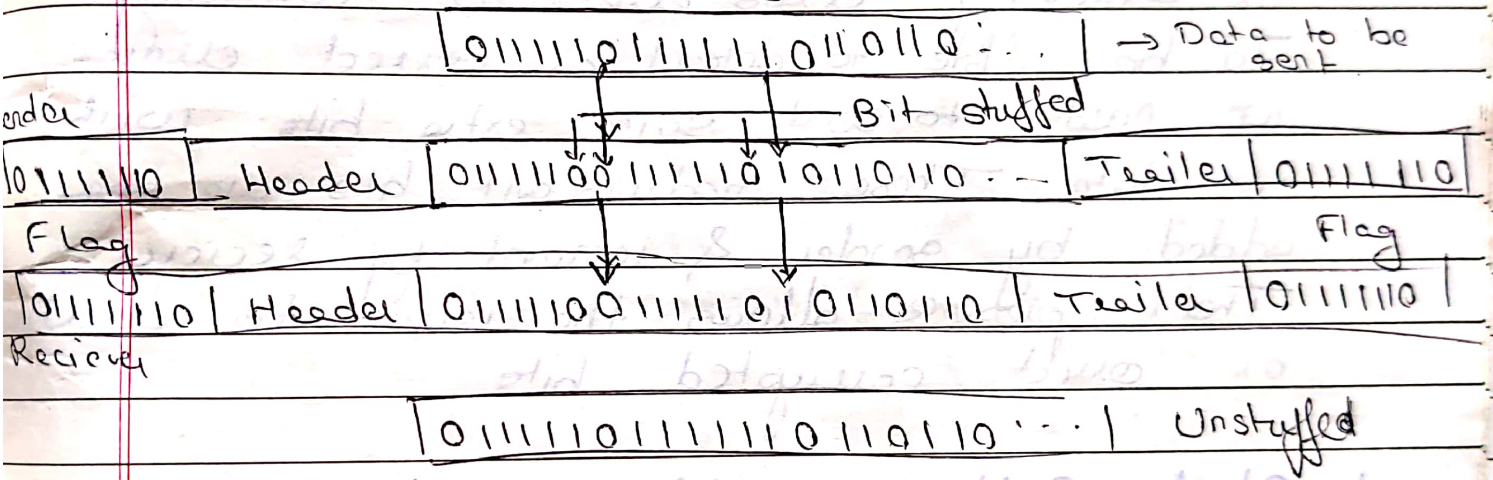


sender
01111110
Flag
01111110
Reciver

Problem: if the flag pattern appears in the data we need to inform the reciever that it is not end of frame

Solution: Bit Stuffing

To avoid the above problem, we stuff one single bit to prevent the pattern from looking like a flag. In bit stuffing, if a 0 & 5 consecutive 1's are encountered an extra 0 is added. This extra stuffed bit is eventually removed from data by reciever



B) Flow & Error Control

Flow Control: Coordinates the amount of data that can be sent before recieving an ack

The idea of flow control in DLL follows the same principle as for the transport layer. The only difference is the transport layer flow control is end to end (source-destination) whereas flow control at DLL is node to node.

Error control: refers to the method of error detection & retransmission. Any time an error is detected, corrupted frames are retransmitted. The error control at DLL

is node to node ie each time a frame passes through a link we need to make sure data is not corrupted.

- Error detection :- Types of error

i) Single bit error - means that only one bit of a given data unit is changed from $1 \to 0$ or $0 \to 1$

ii) Burst Error - Means that 2 or more bits in data unit have changed from $1 \to 0$ or $0 \to 1$

Redundancy - The central concept in detecting or correcting ~~errors~~ errors is redundancy. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are ~~set~~ added by sender & removed by reciever. Their presence allows the reciever to detect or correct corrupted bits.

* Block Coding - In block coding, we divide our message into blocks each of k bits called data words. We add r redundant bits ~~to~~ to each block to make the length $n = k + r$. The resulting n bit is called the codeword. Each codeword sent to the reciever may change during transmission

- If the recieved codeword is same as one of valid codewords, the word is accepted & corresponding data word is extracted. If the recieved codeword is not valid, it is discarded. However if codeword is corrupted during transmission but the recieved word still matches a codeword valid

the error remains undetected

eg. $n = k + 1$, $n = 3$

Dataword	Codeword
00	000
01	011
10	101
11	110

Assume the sender sends the dataword 01, it encodes it & sends 011. 3 cases are possible:

1) The reciever recieves 011, it is a valid codeword, the reciever extracts 01.
2) The codeword is corrupted during transmission & 111 is recieved. This is not a valid codeword & is ~~dicat~~ discarded
3) Codeword is corrupted during transmission & 000 is recieved, this is a valid codeword, the reciever incorrectly extracts data to 00. Two corrupted bits have made error undetectable.

* Cyclic Redundancy Check (CRC)

Q. Generate CRC code for the dataword 1001 & divisor is 1011

⇒ No of bits in divisor $n = 4$

Dividend = Dataword + $(n-1)$ 0's

= 1001000

At sender end:

```
        1010
1011 ) 1001000
     ⊕ 1011 ↓
       0100
     ⊕ 0000 ↓
       1000
     ⊕ 1011 ↓
       0110
     ⊕ 0000
       110   → Remainder
```

Code word Generation :- In CRC the required codeword is obtained by writing dataword followed by remainder. 1001110

At recievers side :

```
        1010
1011 ) 1001110
     ⊕ 1011 ↓
       0101
     ⊕ 0000 ↓
       1011
     ⊕ 1011 ↓
       0000
     ⊕ 0000
       0000
```

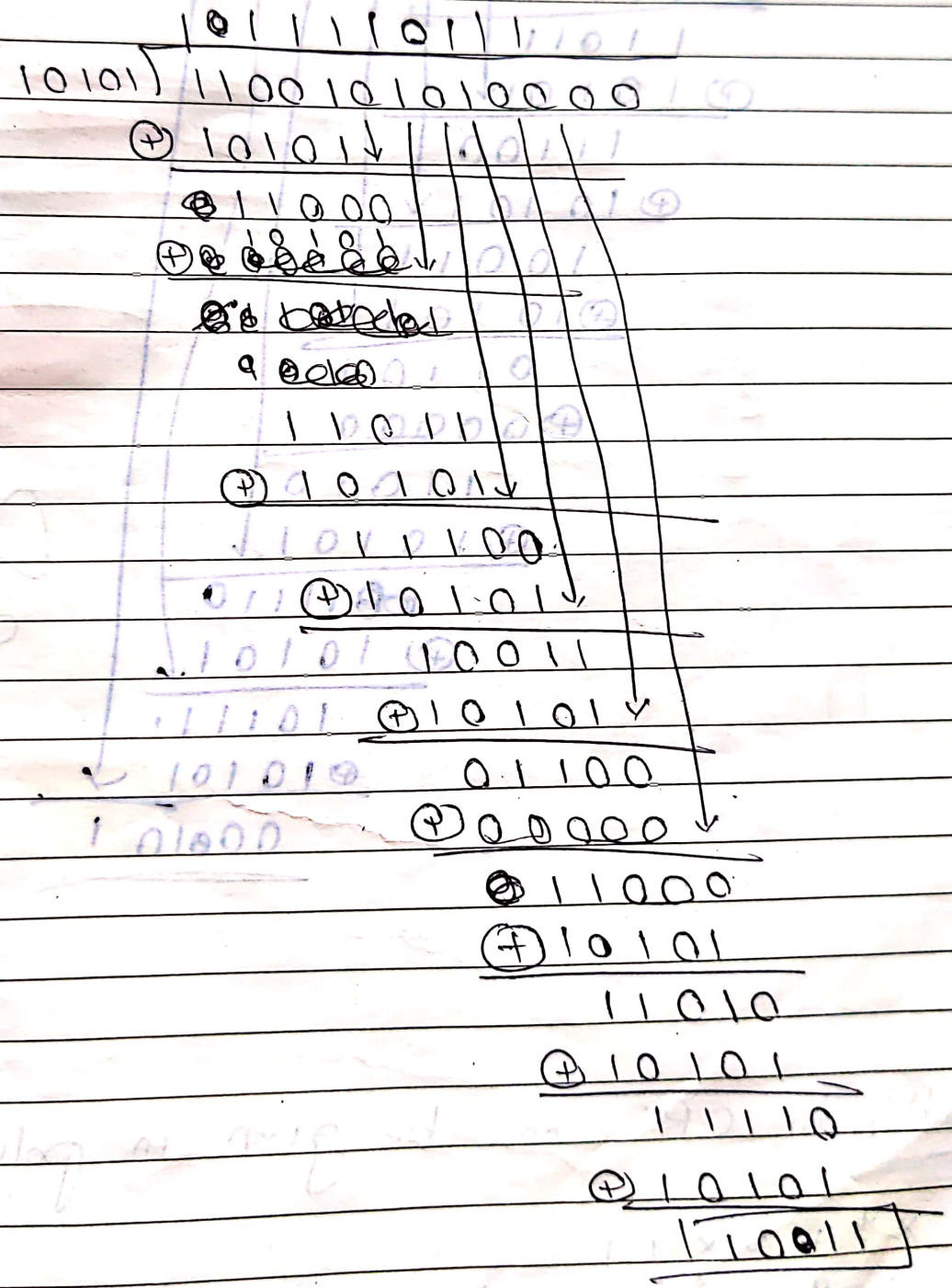
If the remainder at reciever end is 0, then the recieved code is error free & is accepted by reciever.

A non-zero remainder indicates presence of error & hence the corresponding code word

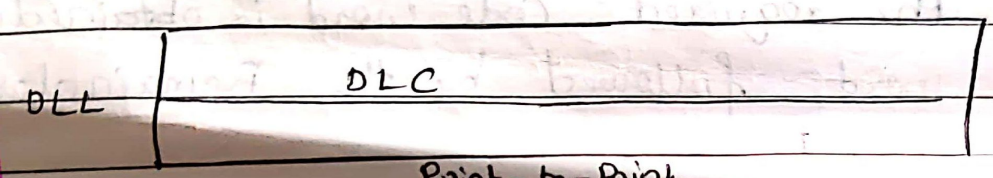
is rejected by receiver.

Q. Generate CRC code for dataword 11001010 & divisor 10101

Dividend = 1100101010000



BROADCAST LINK.



DLC: DLC functions include framing, flow & error control & error detection & correction.

CRC [Cyclic Redundancy Check]:

⇒ Flow & Error Control:

Error Detection Technique :-

i.] Cyclic Redundancy check:

Q. Generate CRC code for dataword: 1001, & divisor: 1011.

Soln: no. of bits in divisor (n) = 4

Dividend = Dataword + (n-1)"0's

= 1001000

Sender: CRC

```
        1010
1011 ) 1001000
Ex-OR → ⊕ 1011 ↓
        00100
       ⊕ 0000 ↓
         1000
        ⊕ 1011 ↓
          0110
         ⊕ 0000
          [110]
```

→ [At the first place of every division we should get a zero (no need to write it) So for that choose your quotient accordingly to get the above condition]

Remainder ← [CRC-code]

Code Word Generation:

In CRC, the required code word is obtained by adding data word, followed by the Remainder i.e. Code = 1001110.

Receiver:-

```
        1010
1011 ) 1001 [110]
       ⊕ 1011 ↓
         0101
        ⊕ 0000 ↓
          1011
         ⊕ 1011 ↓
           0000
          ⊕ 0000
           0000
```

If the Remainder at the receiver end is zero, then the receive code is Error-free & is accepted by the receiver. A non-zero remainder indicates presence of error & hence corresponding code word is rejected by the receiver.

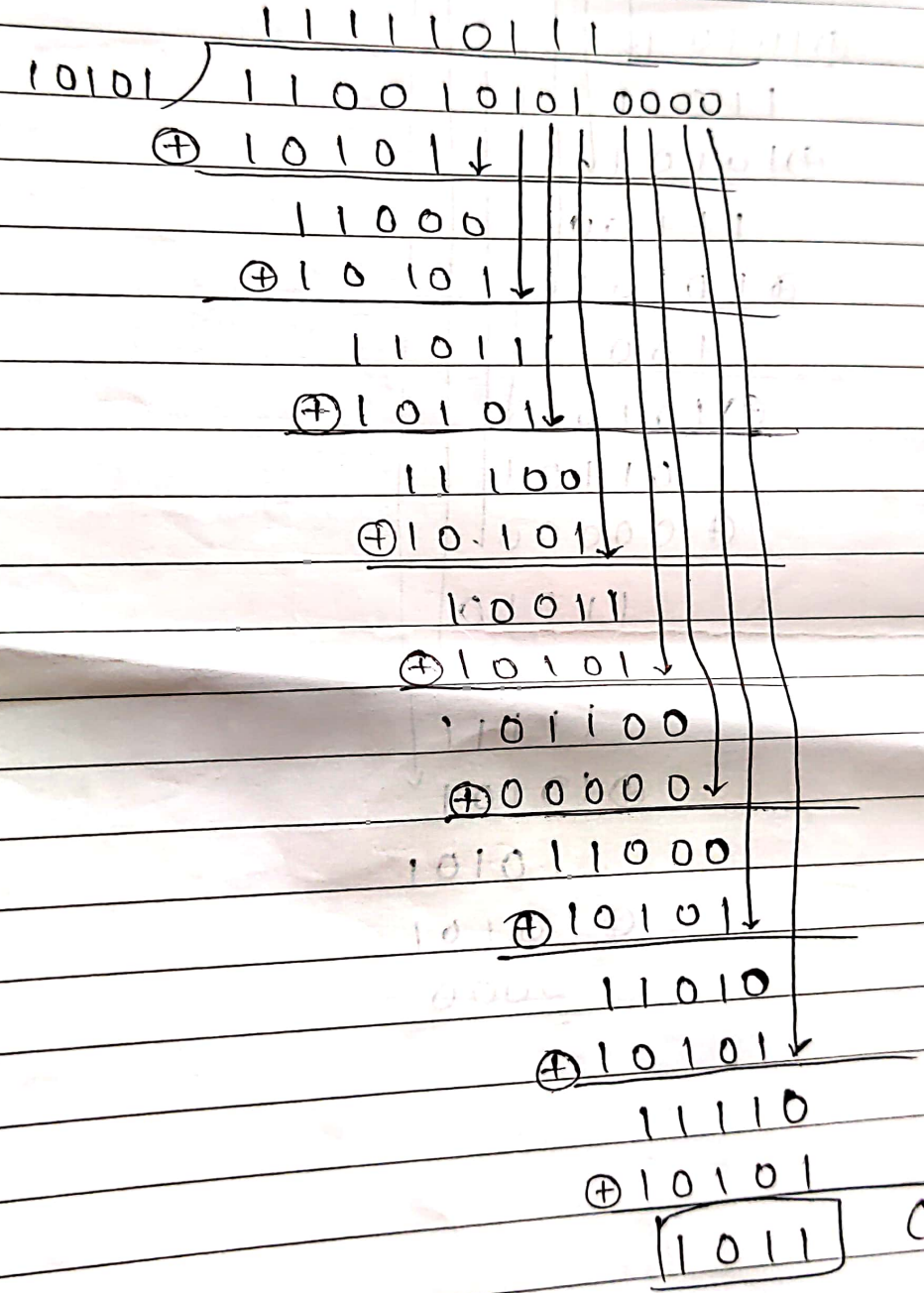
2) Dataword: 11001010 1

Divisior: 10101 n=5

Soln. Dividend = Dataword + (n-1) 0's

= 110010101 0000

Sender:



For polynomial question is given

eg: $x^4 + x + 1 = 1x^4 + 0x^3 + 0x^2 + 1x + 1$

$= \underline{10011} \leftarrow$ Dataword.

Receiver.

```
                111110111
10101 ) 1100101011011
        +10101
         11000
        ⊕10101
          11011
         ⊕10101
           11100
          ⊕10101
            10011
           ⊕10101
             01101
            ⊕00000
              11010
             ⊕10101
               11111
              ⊕10101
                10101
               ⊕10101
                 0000
```

* Parity Check:

In this technique, a k bit dataword is changed to an n-bit codeword where n = k+1. The extra bit or parity bit is selected to make total nos of 1's in codeword even.

eg k = 2

Dataword	Codeword
00	000 → Parity
01	011
10	101
11	110

At the encoder side, if we assume there is a 4 bit dataword ($A_0 A_1 A_2 A_3$) then the parity is calculated by $r_0 = a_3 + a_2 + a_1 + a_0$. If the no of 1's is even, the result is 0. If the no of 1's is odd, the result is 1. In both cases, the total no 1's in codeword is even. The sender sends the codeword which may be corrupted during transmission. The receiver recieves a 5 bit word. The checker at the reciever end does the same thing as the generator in sender with one exception. The addition is done over all 5 bits of result is just 1 bit i.e if no of 1's in recieved codeword is even the result is 0. & hence the reciever understands there is no error in data. If the result is 1, the no of 1's are odd & hence the data recieved has error & is discarded by reciever.

$s_0 = b_3 + b_2 + b_1 + b_0 + q_0 \pmod 2$

Give eg stating all 3 conditions:

1) Right codeword
2) Show error & how parity check detects it
3) Disadvantage

Parity check code can detect only odd no of errors.

* Checksum : It is an error detecting technique that can be applied to a message of any length. In the internet, is mostly used network & transport layer. At the source, the msg is first divided into m bits, the generator then creates an extra m bits called checksum which is sent with the msg. At the destination, the checker creates a new checksum & compares it with the checksum sent by sender. If it matches, the msg is accepted else discarded.

eg. Suppose the msg is a list of 5 nos that we want to send to destination. In addition to sending nos, we send the sum of nos. Suppose msg is (7, 11, 12, 0, 6) we send (7, 11, 12, 0, 6, 36) where 36 is sum of original nos. The receiver adds the 5 nos & computes the result with the sum. If the two are same, the reciever assumes no error & accepts & discards the sum otherwise there is an error & msg is discarded.

5.8 Error Correcting Code

UQ. 5.8.1 Justify Hamming code is error detection and correction code.

Error correction

- This is difficult than error detection.
- In error detection, the receiver need to know only that the received code word is invalid; but in error correction, the receiver needs to find the original code word sent. Thus we need more redundant bits for error correction.
- The following Fig 5.8.1 shows the process of error correction.
- It is similar to detection only checker correct the error instead of only detecting and discarding.

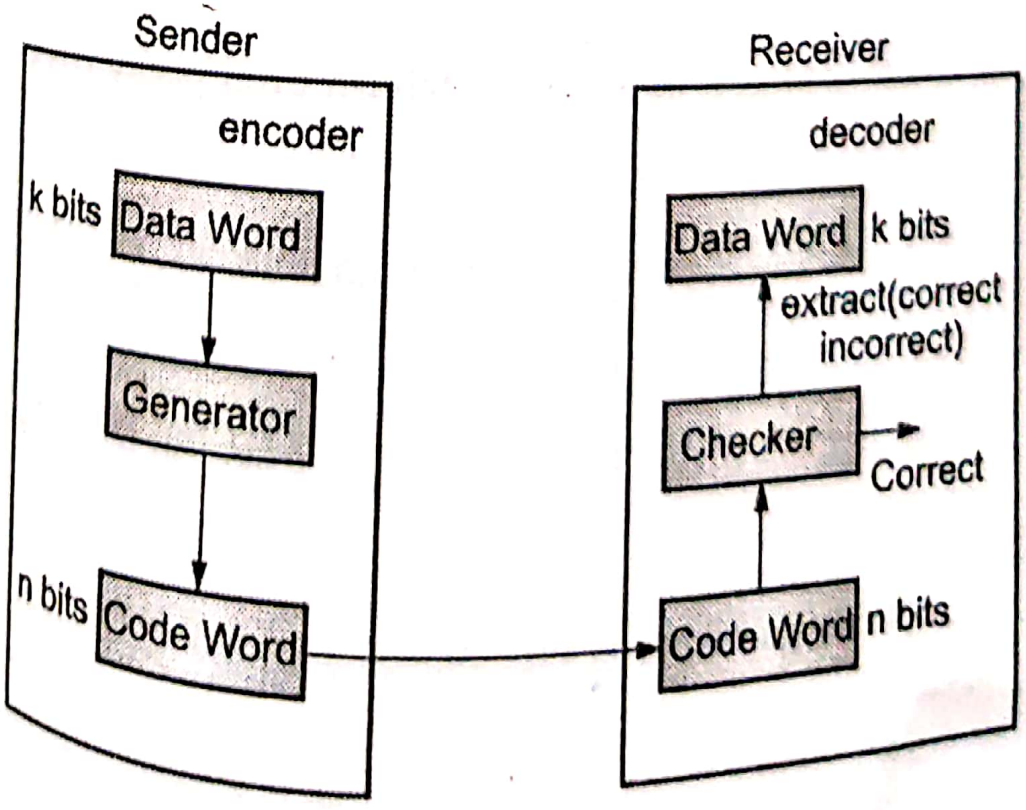


Fig. 5.8.1 : Process of Error correction

Hamming Weight of a Code Word [w(x)]

- The Hamming weight of a code word x is defined as the number of non zero elements in the code word.
- Hamming weight of a code vector (code word) is the distance between that code word and an all zero code vector. (a code having all elements equal to zero).

Example

- if x = 1 1 0 1 0 1 0 0 then the Hamming weight W (x) = 4.
- As, the number of non-zero elements in the above code word is 4,

Hamming Distance

- Used for error control.

(MU - May 18, 10 Marks) vectors (or code words) having the same number of elements.

- The "Hamming distance" between the two code words is defined as the number of differences between the corresponding bits Hamming distance between 2 words x and y is denoted as d(x,y).
- For example consider the two code words given below :

  Code word 1 = 1 0 1 0 1

  Code word 2 = 1 1 1 1 0
- Note that the bits 2, 4 and 5 are different from each other. Hence Hamming distance is 3.

Minimum Hamming distance

- It is the smallest hamming distance between all possible pairs.
- We use d_{min} to define the minimum hamming distance in a coding scheme.
- To find this value, we find the hamming distances between all words and select the smallest one.
- **Example :**

  Six Hamming distances are given below find minimum hamming distance

  d (00000,01011) = 3, d (00000,10101) = 3,

  d (00000,11110) = 4, d (01011, 10101) = 4,

d (01011,11110) = 3, d (10101,11110) = 3

- The smallest hamming distance in above example is 3

Thus $d_{min} = 3$

Role of "d_{min}" in error detection and correction

- The error detection is always possible when the number of transmission errors in a code word is less than the minimum distance d_{min}, because then the erroneous word is not a valid code word.
- But when the number of errors equals or exceeds d_{min}, the erroneous code word may correspond to another valid code word and errors cannot be detected.
- The error detection and correction capabilities of a coding technique depend on the minimum distance d_{min} as shown in the Table 5.8.1

Table 5.8.1 : Role of d_{min} for detection and correction of errors

Detect upto "s" errors per word.	$d_{min} \geq (s + 1)$
Correct upto "t" errors per word.	$d_{min} \geq (2t + 1)$
Correct upto "t" errors and detect s > t errors per word.	$d_{min} \geq (t + s + 1)$

5.8.1 Hamming Codes

- It is a single bit error correcting code. These are linear block codes.
- The family of (n, k) hamming codes for an integer m > = 3 is defined by the following equations:
- Block length or code word length : $n = 2^m - 1$
- No. of message bits or data words :

  $k = 2^m - m - 1$ or n-m
- No. of parity bits : $(n - k) = m$
- That is the minimum number of parity bits is 3
- Therefore, minimum hamming distance $d_{min} = 3$.
- The code rate = k / n
- If m >> 1 then code rate $r \approx 1$

For example, if m = 3 then n = 7 and k = 4. This is the hamming code C (7, 4) with $d_{min} = 3$.

Hamming code structure

- It is and error correcting code. The parity / redundant bits are inserted between the data bits.
- The 7-bit hamming code is used commonly but the concept can be extended to any number of bits

D7	D6	D5	P4	D3	P2	P1

Fig. 5.8.2(a) : 7-bit hamming code C (7,4)

D15	D14	D13	D12	D11	D10	D9	P8	D7	D6	D5	P4	D3	P2	P1

Fig. 5.8.2(b) : 15-bit hamming code C (15, 11)

D = Data bits

P = Parity bits.

- Parity bits are inserted at each 2^n bit where, n = 0,1,2,3.....
- Thus, P1 is at 2^0 i.e. 1^{st} bit, P2 is at $2^1 = 2$, P4 is at $2^2 = 4$ and P8 is at $2^3 = 8$

7 bit hamming code

- Assume that 4 data bits are to be transmitted.
- Therefore code word pattern is

D7	D6	D5	P4	D3	P2	P1
D7 = 1	D6 = 0	D5 = 1	P4	D3 = 1	P2 = 0	P1 = 1

Section of parity bits

- It is adjusted to 0 or 1 as to establish even parity over bits.
- For P1 Section of parity bits are 1,3,5,7 i.e. P1, D3, D5, D7
- For P2 Section of parity bits are 2,3,6,7 i.e. P2, D3, D6, D7
- For P4 Section of parity bits are 4,5,6,7 i.e. P4, D5, D6, D7

Example 5.8.1 : A data bit word 1011 is to be transmitted. Construct even parity 7- bit hamming code.

Solution :

Step 1 : The codeword format

Given data bit = 1011

D7 = 1	D6 = 0	D5 = 1	P4	D3 = 1	P2	P1

P4, P2 and P1 is to be decided.

Step 2 : Decide P1

- For P1, sections to be considered are 1, 3, 5, 7
- Here, we have to set P1 = 1 as 3, 5, 7 = 111 in order to have the even parity.

D7 = 1	D6 = 0	D5 = 1	P4	D3 = 1	P2	P1 = 1

Step 3 : Decide P2

- For P2, sections to be considered are 2, 3, 6, 7
- Here, we have to set P2 = 0 as 3, 6, 7 = 101 in order to have the even parity.

D7 = 1	D6 = 0	D5 = 1	P4	D3 = 1	P2 = 0	P1 = 1

Step 4 : Decide P4

- For P4, sections to be considered are 4, 5, 6, 7
- Here, we have to set P4 = 0 as 5, 6, 7 = 101 in order to have the even parity.

D7 = 1	D6 = 0	D5 = 1	P4 = 0	D3 = 1	P2 = 0	P1 = 1

Thus, the code word which is transmitted to the receiver = 1010101

Example 5.8.2 : A 7 bit hamming code is received as 1110101. What is the correct code?

Solution :

Hamming code = 1110101

D7 = 1	D6 = 1	D5 = 1	P4 = 0	D3 = 1	P2 = 0	P1 = 1

Step 1 : Check bits 4, 5, 6, 7 = 0111

- In this, number of ones are 3 which is considered to be odd parity. Hence there is an error.
- Thus set P4 = 1 for even parity.

▶ **Step 2 :** Check bits 2, 3, 6, 7 = 0111

- In this, number of ones are 3 which is considered to be odd parity. Hence there is an error.
- Thus set P2 = 1 for even parity.

▶ **Step 3 :** Check bits 1, 3, 5, 7 = 1111

- In this, number of ones are 4 which is considered to be even parity. Hence there is no error.
- Thus set P1 = 0 for even parity.

Therefore, Error word E = 110

P4 = 1	P2 = 1	P1 = 0

▶ **Step 4 : Decimal equivalent of E = 110 = $(6)_{10}$**

- Therefore, 6^{th} bit in the received code word is incorrect. Thus, Invert and correct it by changing 0 to 1 or 1 to 0.
- Therefore, Corrected code word is

D7 = 1	D6 = 0 (inverted bit)	D5 = 1	P4 = 0	D3 = 1	P2 = 0	P1 = 1

15 bit hamming code

Assume that 4 data bits are to be transmitted. Therefore code word pattern is

D15	D14	D13	D12	D11	D10	D9	P8	D7	D6	D5	P4	D3	P2	P1

Section of parity bits

- It is adjusted to 0 or 1 as to establish even parity over bits.
- For P1 Section of parity bits are 1, 3, 5, 7, 9, 11, 13 i.e. P1, D3, D5, D7, D9, D11, D13
- For P2 Section of parity bits are 2, 3, 6, 7, 10, 11 i.e. P2, D3, D6, D7, D10, D11
- For P4 Section of parity bits are 4,5,6,7, 12, 13 i.e. P4, D5, D6, D7, D12, D13
- For P8 Section of parity bits are 8, 9, 10, 11, 12, 13 i.e. P8, D9, D10, D11, D12, D13

Advantages of Hamming code

- Hamming code method is effective on networks where the data streams are given for the single-bit errors.
- Hamming code not only provides the detection of a bit error but also helps you to indent bit containing error so that it can be corrected.
- The ease of use of hamming codes makes it best them suitable for use in computer memory and single-error correction.

Disadvantages of Hamming code

- Single-bit error detection and correction code. However, if multiple bits are founded error, then the outcome may result in another bit which should be correct to be changed.
- This can cause the data to be further errored. Hamming code algorithm can solve only single bits issues.

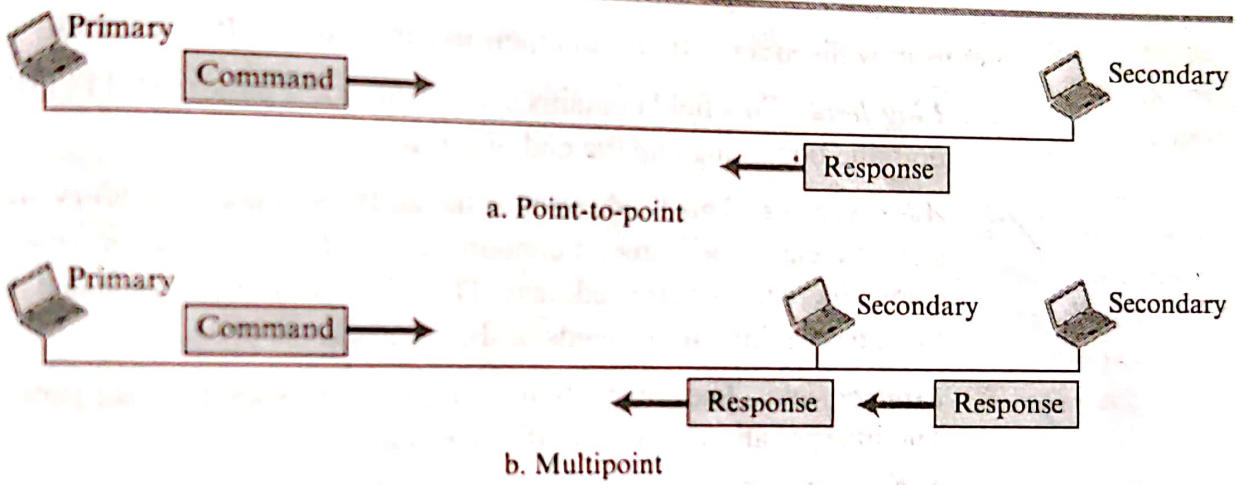
HDLC

High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the Stop-and-Wait protocol we discussed in Chapter 3.

Configurations and Transfer Modes

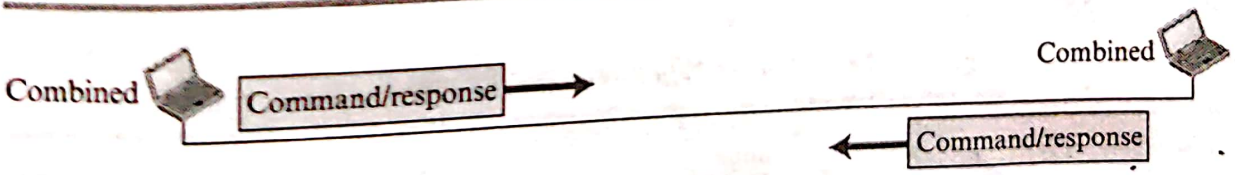
HDLC provides two common transfer modes that can be used in different configurations: *normal response mode (NRM)* and *asynchronous balanced mode (ABM)*. In *normal response mode (NRM)*, the station configuration is unbalanced. We have one primary station and multiple secondary stations. A *primary station* can send commands; a *secondary station* can only respond. The NRM is used for both point-to-point and multipoint links, as shown in Figure 5.20.

Figure 5.20 *Normal response mode*



In ABM, the configuration is balanced. The link is point-to-point, and each station can function as a primary and a secondary (acting as peers), as shown in Figure 5.21. This is the common mode today.

Figure 5.21 *Asynchronous balanced mode*

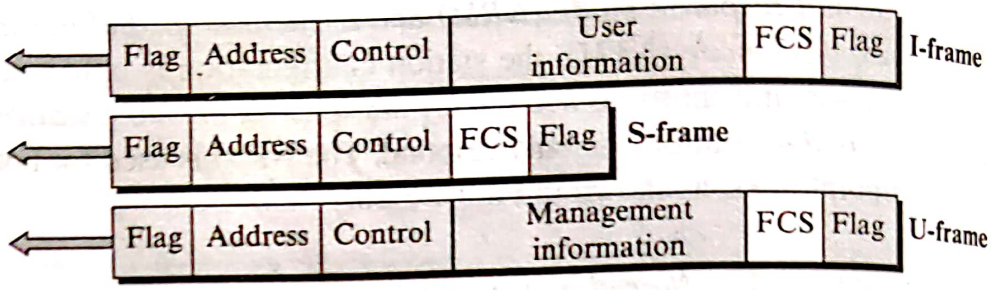


Frames

To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames: *information frames (I-frames)*, *supervisory frames (S-frames)*, and *unnumbered frames (U-frames)*. Each type of frame serves as an envelope for the transmission of a different type of message. I-frames are used to transport user data and control information relating to user data (piggybacking). S-frames are used only to transport control information. U-frames are reserved for system management. Information carried by U-frames is intended for managing the

link itself. Each frame in HDLC may contain up to six fields, as shown in Figure 5.22: a beginning flag field, an address field, a control field, an information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame.

Figure 5.22 *HDLC frames*

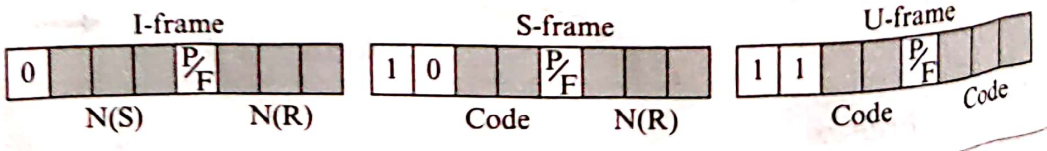


Let us now discuss the fields and their use in different frame types.

- ***Flag field.*** This field contains synchronization pattern 01111110, which identifies both the beginning and the end of a frame.
- ***Address field.*** This field contains the address of the secondary station. If a primary station created the frame, it contains a *to* address. If a secondary station creates the frame, it contains a *from* address. The address field can be one byte or several bytes long, depending on the needs of the network.
- ***Control field.*** The control field is one or two bytes used for flow and error control. The interpretation of bits are discussed later.
- ***Information field.*** The information field contains the user's data from the network layer or management information. Its length can vary from one network to another.
- ***FCS field.*** The frame check sequence (FCS) is the HDLC error detection field. It can contain either a 2- or 4-byte CRC.

The control field determines the type of frame and defines its functionality. So let us discuss the format of this field in detail. The format is specific for the type of frame, as shown in Figure 5.23.

Figure 5.23 *Control field format for the different frame types*



Control Field for I-Frames I-frames are designed to carry user data from the network layer. In addition, they can include flow- and error-control information (piggybacking). The subfields in the control field are used to define these functions. The first bit defines the type. If the first bit of the control field is 0, this means the frame is an I-frame. The next 3 bits, called $N(S)$, define the sequence number of the frame. Note that with 3 bits, we can define a sequence number between 0 and 7. The last 3

bits, called $N(R)$, correspond to the acknowledgment number when piggybacking is used. The single bit between $N(S)$ and $N(R)$ is called the P/F bit. The P/F field is a single bit with a dual purpose. It has meaning only when it is set (bit = 1) and can mean poll or final. It means *poll* when the frame is sent by a primary station to a secondary (when the address field contains the address of the receiver). It means *final* when the frame is sent by a secondary to a primary (when the address field contains the address of the sender).

Control Field for S-Frames Supervisory frames are used for flow and error control whenever piggybacking is either impossible or inappropriate. S-frames do not have information fields. If the first 2 bits of the control field are 10, this means the frame is an S-frame. The last 3 bits, called $N(R)$, corresponds to the acknowledgment number (ACK) or negative acknowledgment number (NAK) depending on the type of S-frame. The 2 bits called *code* are used to define the type of S-frame itself. With 2 bits, we can have four types of S-frames, as described below:

- ***Receive ready (RR).*** If the value of the code subfield is 00, it is an RR S-frame. This kind of frame acknowledges the receipt of a safe and sound frame or group of frames. In this case, the value of the $N(R)$ field defines the acknowledgment number.
- ***Receive not ready (RNR).*** If the value of the code subfield is 10, it is an RNR S-frame. This kind of frame is an RR frame with additional functions. It acknowledges the receipt of a frame or group of frames, and it announces that the receiver is busy and cannot receive more frames. It acts as a kind of congestion-control mechanism by asking the sender to slow down. The value of $N(R)$ is the acknowledgment number.
- ***Reject (REJ).*** If the value of the code subfield is 01, it is an REJ S-frame. This is a NAK frame, but not like the one used for Selective Repeat ARQ. It is a NAK that can be used in Go-Back-N ARQ to improve the efficiency of the process by informing the sender, before the sender timer expires, that the last frame is lost or damaged. The value of $N(R)$ is the negative acknowledgment number.
- ***Selective reject (SREJ).*** If the value of the code subfield is 11, it is an SREJ S-frame. This is a NAK frame used in Selective Repeat ARQ. Note that the HDLC Protocol uses the term *selective reject* instead of *selective repeat*. The value of $N(R)$ is the negative acknowledgment number.

Control Field for U-Frames Unnumbered frames are used to exchange session management and control information between connected devices. Unlike S-frames, U-frames contain an information field, but one used for system management information, not user data. As with S-frames, however, much of the information carried by U-frames is contained in codes included in the control field. U-frame codes are divided into two sections: a 2-bit prefix before the P/F bit and a 3-bit suffix after the P/F bit. Together, these two segments (5 bits) can be used to create up to 32 different types of U-frames.

PAGE No.
DATE / /

* HDLC

High level data link control is a bit oriented protocol for a communication over point to point & multipoint link.

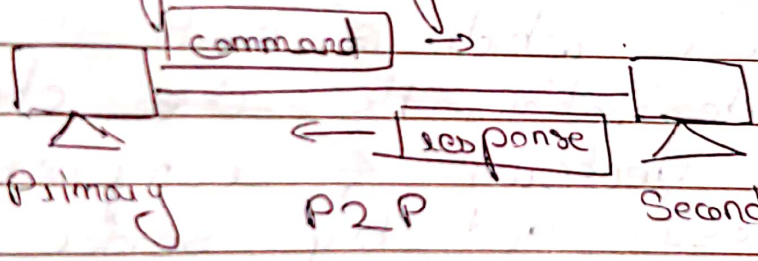
- Configuration and transfer mode.

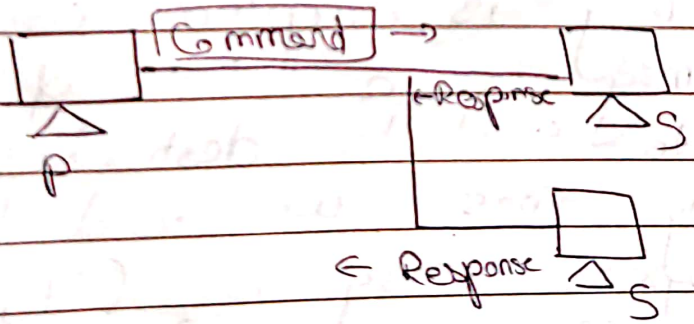
HDLC provides two common transfer mode

i) Normal response mode (NRM)

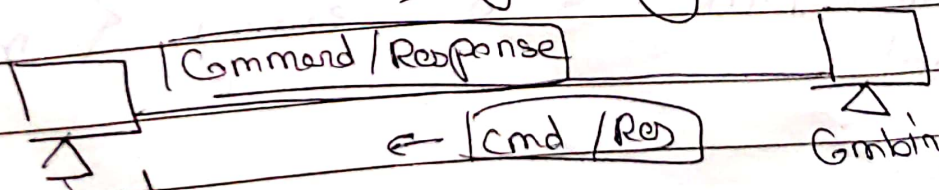
ii) Asynchronous Balance Mode (ABM)

In NRM, the station configuration is unbalanced. We have 1 primary station & multiple secondary stations. Primary station can send commands & secondary station can only respond to commands. NRM is used for both point to point & multipoint





In ABM, the configuration is balanced. The link is point to point & each station can function as a primary & secondary station



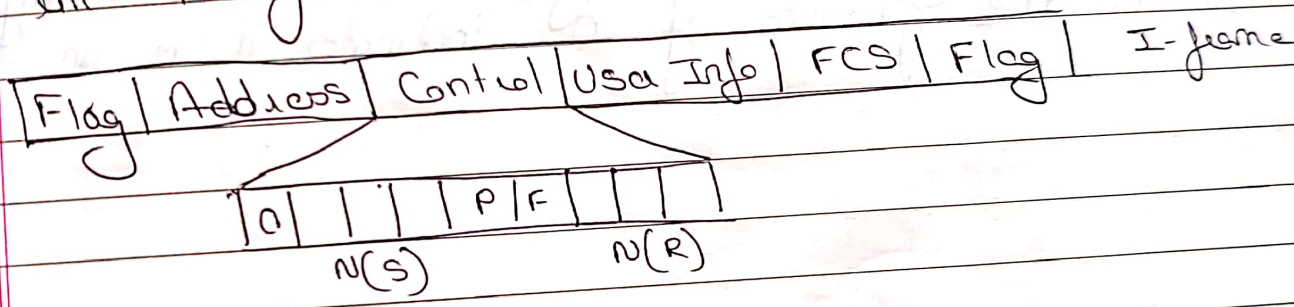
- Frames

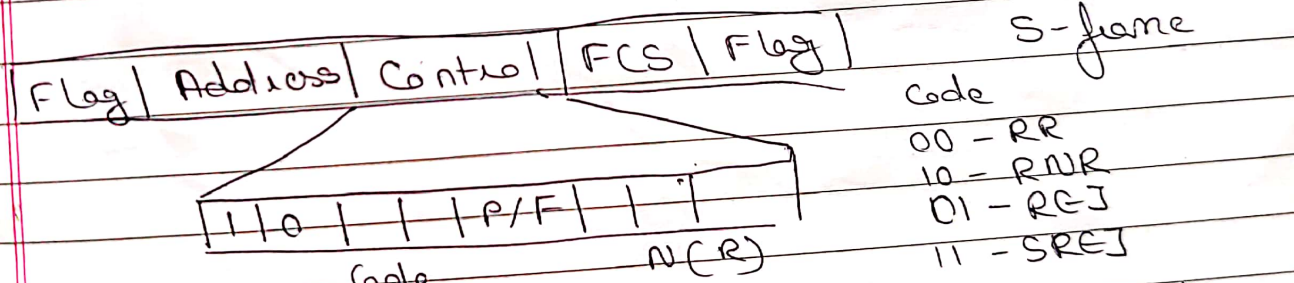
HDLC defines 3 types of frames

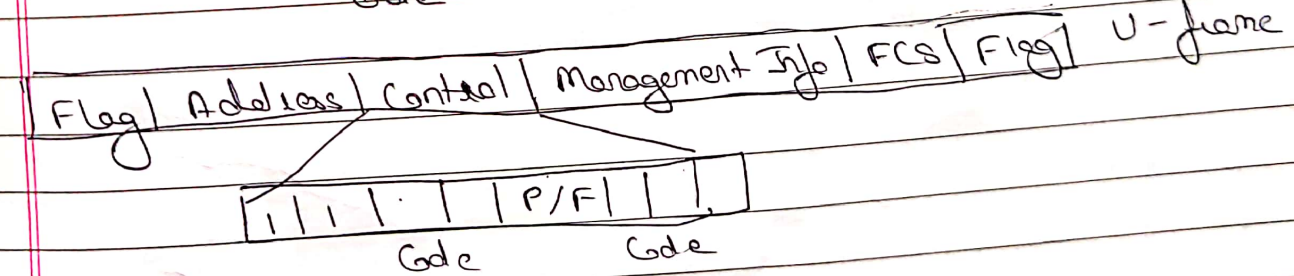
i) Information frame (I-frame): Used to transport user data & control information related to user data (Piggybacking)

ii) Supervisory frame (S-frame): Used only to transport control information

iii) Unnumbered frame (U-frame): Reserved for system management. The info carried by this frame is used for managing the link itself







Flag Field - This field contains the pattern 01111110 which identifies both beginning and end of frame

Address field - Contains the address of the secondary station if the primary station created a

frame & vice versa.
Control field : Used for flow & error control.
Information field : Contains user data from the n/w layer or management info.
FCS field : The frame check sequence is the HDLC error detection ~~field~~ field. It contains a two byte or 4 byte CRC code.

- Control field determines the type of frame & defines its functionality
For I-frame - The first bit is always zero ie the presence of 0 indicates it is an I-fram