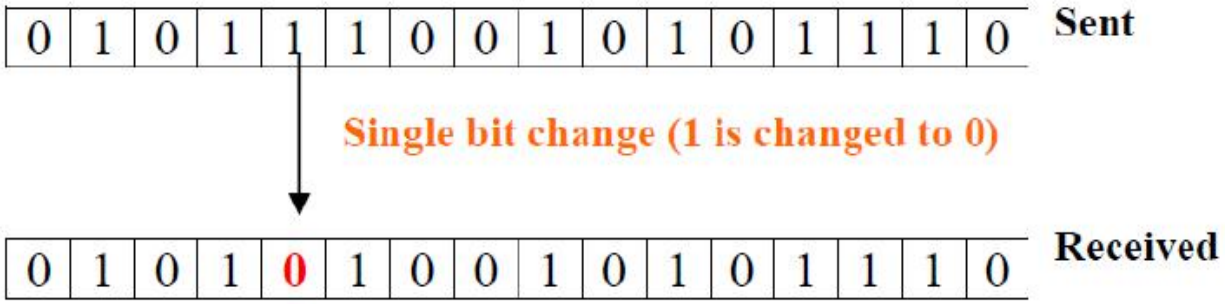
Error Control

- In data communication, there are **two types of errors**:
  1) *Single-Bit Error*
  2) *Burst Error*
- Error control has two phases:
  1) Error detection &
  2) Error correction

Types of Error: Single-Bit Error

- In a single-bit error, only 1 bit in the data unit has changed.

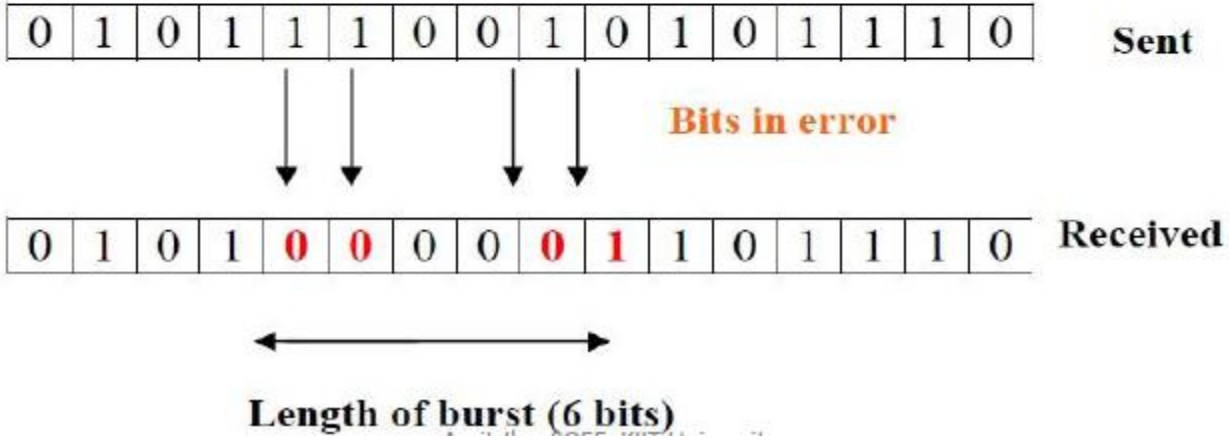


- Single-bit errors are the least likely type of error in serial data transmission.
- To understand why, imagine data sent at 1 Mbps. This means that each bit lasts only 1/1,000,000 s, or 1 microsec.
- For a single-bit error to occur, the noise must have a duration of only 1 microsec, which is very rare; noise normally lasts much longer than this.

- However, a single-bit error can happen if we are having a parallel data transmission.
- For example, if 16 wires are used to send all 16 bits of a word at the same time and one of the wires is noisy, one bit is corrupted in each word.

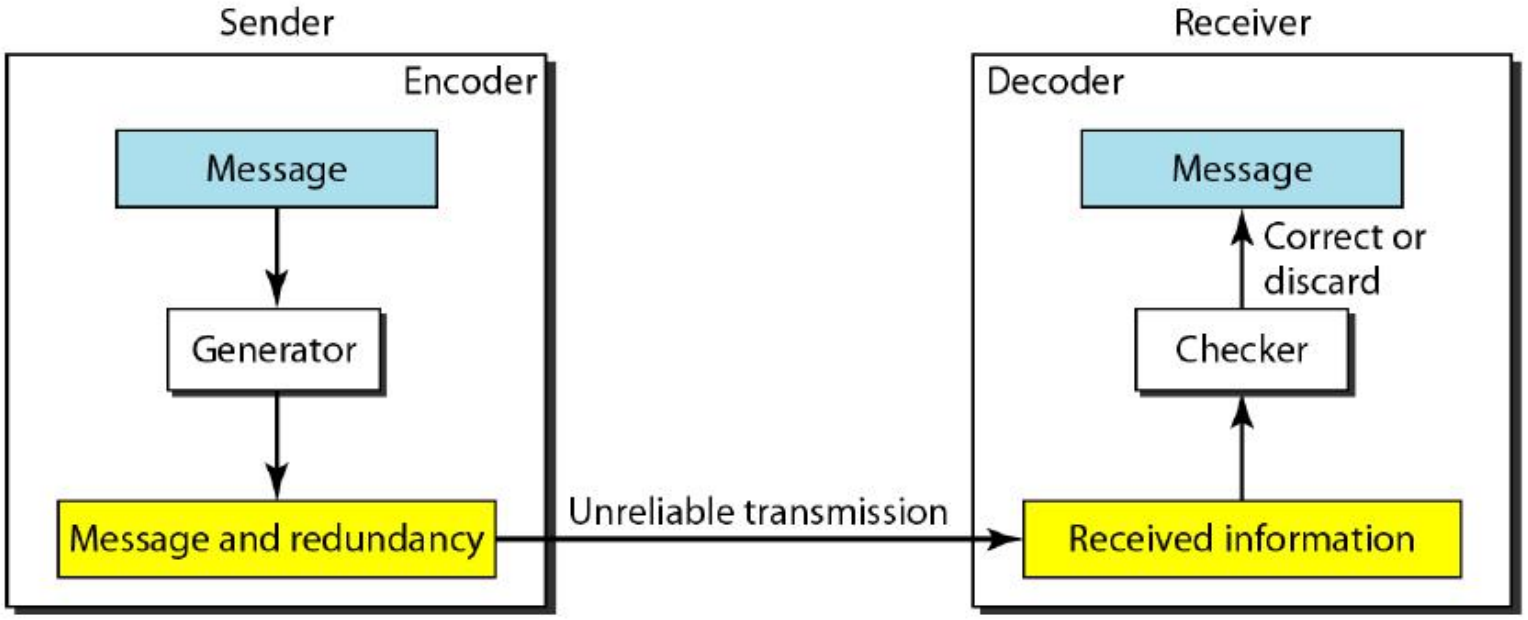
Types of Error: Burst Error

- The term ***burst error*** means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1.
- Note that burst error doesn't necessary means that error occurs in consecutive bits.
- The length of the burst error is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not be corrupted.



- A burst error is more likely to occur than a single-bit error.
- This is because, the duration of noise is generally more than the duration of 1 bit, which means that when noise affects data, it affects a set of bits.
- The number of bits affected depends on the data rate and duration of noise.
- For e.g., if we are sending data at 1kbps, a noise of 1/100 sec duration can affect 10 bits.

The structure of encoder and decoder



Introduction to Block Code

- *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.*

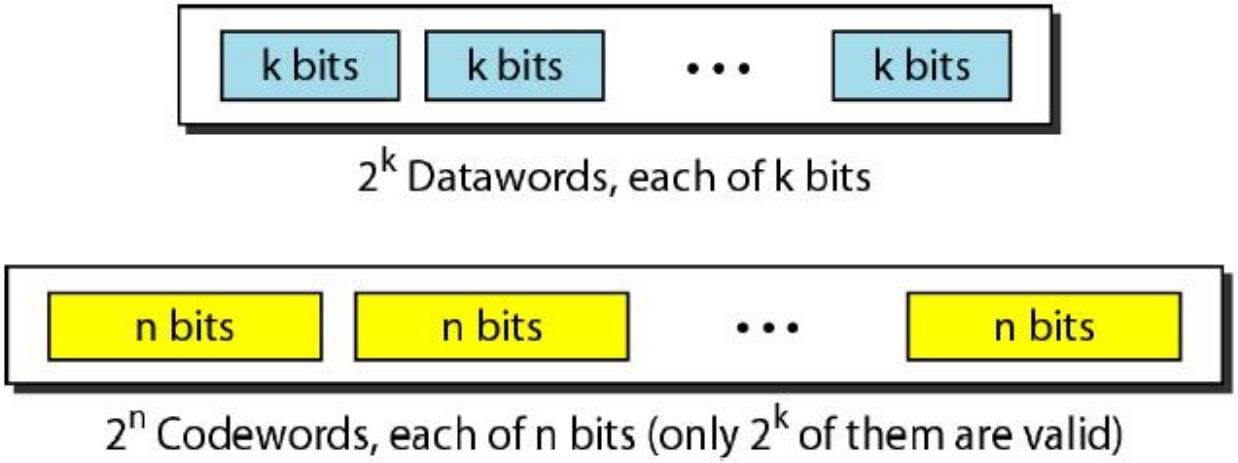


Fig: Datawords and codewords in block coding

- *In 4B/5B coding scheme, $k = 4$ and $n = 5$.*
- *As we saw, we have $2^k = 16$ datawords and $2^n = 32$ codewords. We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.*

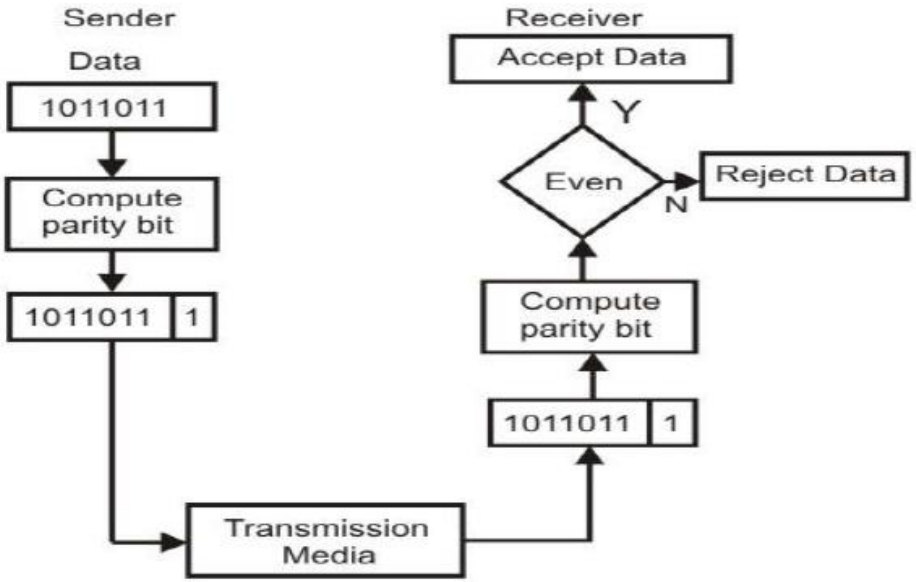
Error Detecting Codes

- The central concept in detecting or correcting errors is ***redundancy,*** which means adding some extra bits along with data.
- The sender adds some redundant bits, whereas receiver removes it.
- Redundant bits (extra bits) facilitate detection and corrections of errors.
- Popular techniques for error detection are:
  1. Simple parity check
  2. Two-dimensional parity check
  3. Checksum
  4. Cyclic redundancy check

Error Detecting Codes: Simple parity check

- It is also known as **one-dimensional parity check**.
- In this technique, a redundant bit called **parity bit,** is appended to every data unit so that number of 1's in the unit (including the parity bit) becomes even.

For Even parity check



- *At the Transmitter side*, parity bit generator adds
  - 1 if the data block contains odd number of 1's
  - 0 if the data block contains even number of 1's
- *At the receiver side*, parity bit is computed from the received data bits.
- If the receiver finds odd number of 1's in received data, then it ensures that an error has occurred.

- For an e.g., the complete list of data words (for 4-bit) and the corresponding code words are given below:

Decimal value	Data Block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101
3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

- It is also possible to use ***odd-parity*** checking, where the number of 1's should be odd.
- From the last table it can be observed that
  - If we move from one code word to another, at least 2 data bits should be changed.
  - Thus, these set of code words are said to have a minimum distance ***(hamming distance)*** of 2.
  - So, if there exists one bit error then receiver will be able to detect it, but it can't detect two bit error.
  - This is because, a code word with **two bits error** again becomes a valid member of the set.
- For a linear code, ***number of errors detected= dmin-1***

where, dmin= minimum hamming distance

- The ***Hamming distance*** between two words is the number of differences between corresponding bits.

1) The Hamming distance d(000, 011) is 2 because

$000 \oplus 011$ is 011 (two 1s)

2) The Hamming distance d(10101, 11110) is 3 because

$10101 \oplus 11110$ is **01011** (three 1s)

- The ***minimum Hamming distance*** is the smallest Hamming distance between **all possible** pairs in a set of words.

Q) Find the minimum hamming distance for the following codewords.

Datawords	*Codewords*
00	000
01	011
10	101
11	110

- **Sol:**

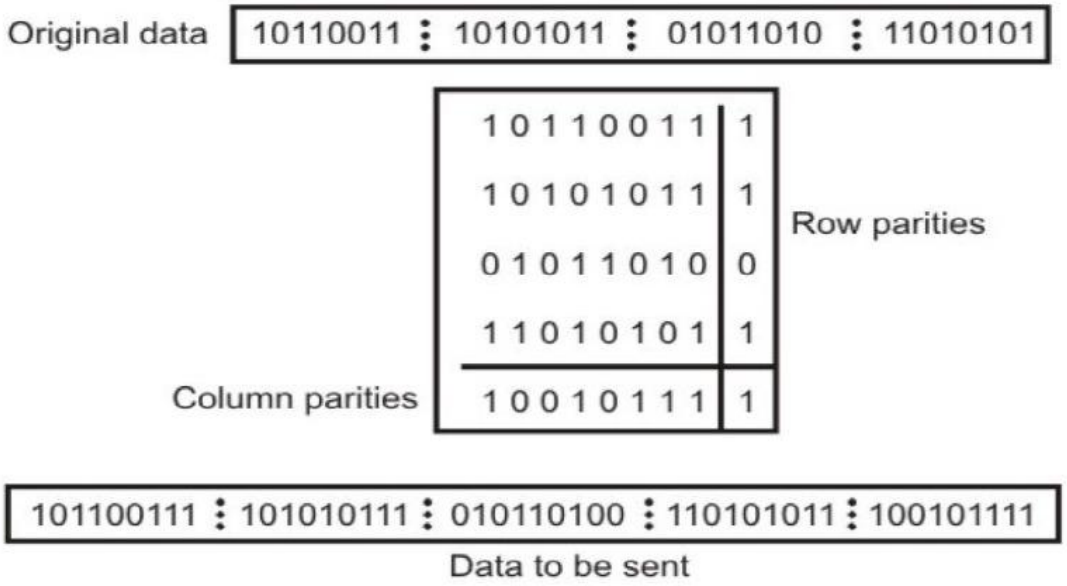
We first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$
$d(011, 110) = 2$ $d(101, 110) = 2$

The d_{min} in this case is 2.

Error Detecting Codes: Two-dimensional parity check

- In a two-dimensional parity check, the block of bits are organized in the form of a table.
- Parity check bits are calculated
  - For each row
  - as well as for each column
- Then, the complete table is sent in the form of block including data as well as both parity bits.
- At the receiver side, these are calculated with the parity bits calculated on the received data.
- The existence of odd number of 1's ensures that the data are corrupted.



if two bits in one data unit is damaged and two bits in another data unit is damaged at exactly same position, the 2-D parity checker will not detect any error.

Explanation: *Let us assume that data to be sent after parity is*
101100111 : 101010111 : 0101101100 : 110101011 : 100101111

But due to noise we receive the following data:

101100111 : 001110111 : 0101101100 : 110101011 : 000001111

Error Detecting Codes: Checksum

- In checksum error detection scheme, the data is divided into k segments each of m bits.
- Traditionally, the Internet has been using a 16-bit checksum, called ***Internet checksum***.
- Its very simple.....
- Suppose our data is a list of five 4-bit numbers that we want to send to a destination.
  - In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12,0,6,36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum.
  - If the two sums are same then receiver assumes no error and accepts the data; otherwise discard it.
- Now, we can further reduce the burden of receiver as follows:
  - Instead of sending sum, we send negative of sum, called ***checksum.*** In this case, we send (7, 11, 12, 0, 6, -36). The receiver adds all the numbers received (including the checksum). If result is 0, it assumes no error, otherwise there is an error.

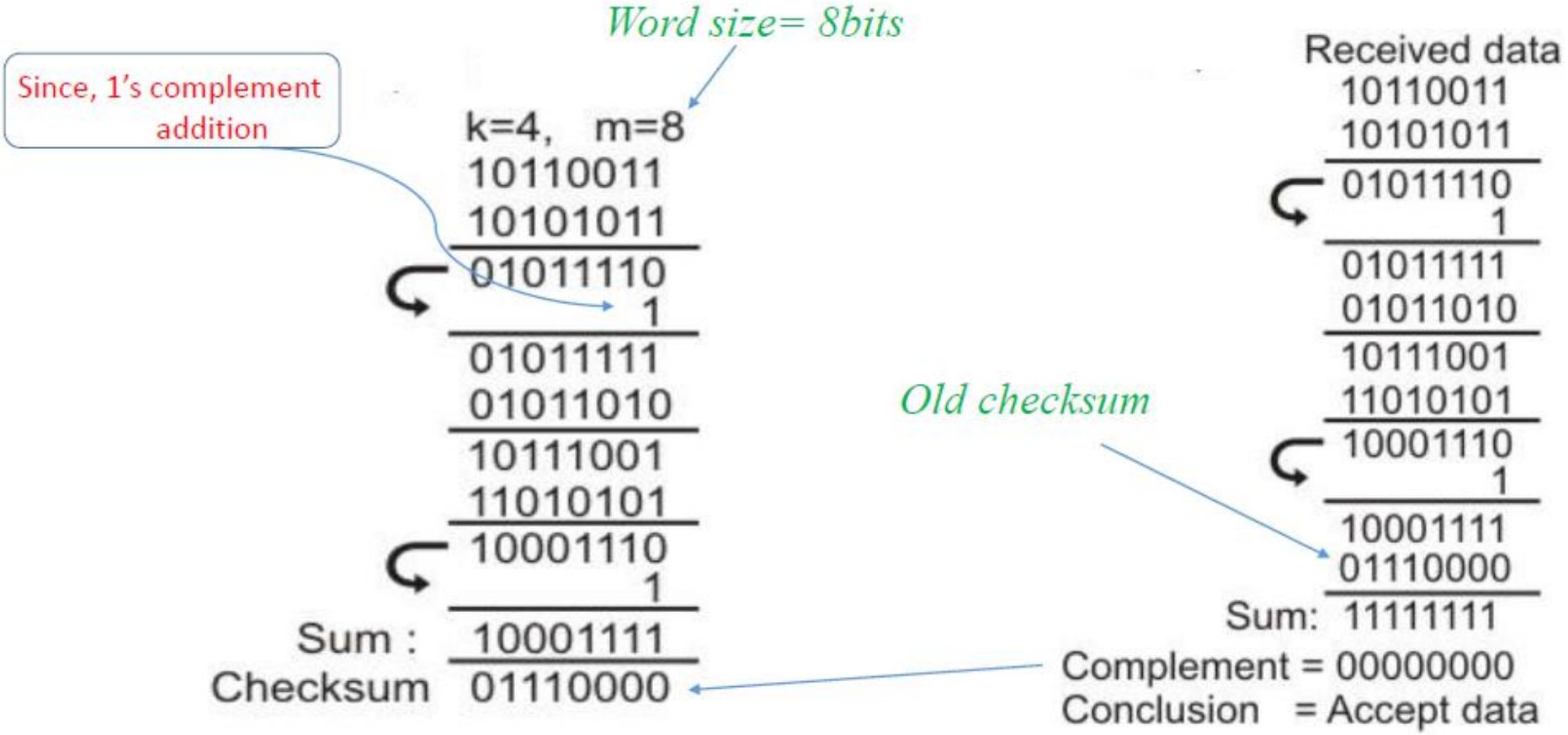
Internet Checksum

- The sender calculates the checksum as follows:
  - The message is divided into 16-bit words.
  - The value of checksum word is set to 0.
  - All words including the checksum are added using one's complement addition.
  - The sum is complemented and becomes the checksum.
- The checksum segment is sent along with the data segments.
- The receiver uses the following steps to detect error:
  - The message (including checksum) is divided into 16-bit words.
  - All words are added using **one's complement addition**.
  - The sum is complemented and becomes the new checksum.
  - If the value of checksum is 0, the message is accepted; otherwise it is rejected.

Ex.1: *For the given data,* **10110011101010110101011010101,** *calculate the checksum by dividing the data into words of 8 bits. Also, find whether the error has occurred or not, if the received data is*

10110011101010110101011010101

- **Sol:**



- **Ex 2:** *Calculate the checksum for a text of 8 characters ("Forouzan"). Assume that the checksum is of 16-bit, as used in Internet today. The ASCII code (in Hex) for the character used here is:* F=0x46, o=0x6F, r=0x72, u= 0x75, z= 0x7A, a=0x61, n=0x6E.

- **Sol:**

1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	F	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
0	0	0	0	Checksum (initial)
8	F	C	6	Sum (partial)
			1	
8	F	C	7	Sum
7	0	3	8	Checksum (to send)

a. Checksum at the sender site

1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	F	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
7	0	3	8	Checksum (received)
F	F	F	E	Sum (partial)
			1	
F	**F**	**F**	**F**	Sum
0	0	0	0	Checksum (new)

a. Checksum at the receiver site

Internet checksum: Key points

- It is used in several Internet protocols like **IP, TCP, UDP** to detect errors in IP header, or in the header and data for TCP/UDP.
- Generally, a checksum is calculated for header content and included in a special field.
- The checksum is calculated **at every router**.
- Overhead is less as compared to two-dimension parity check.

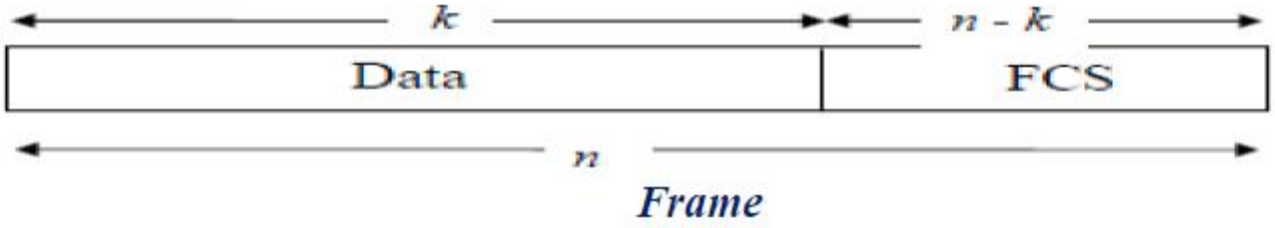
• **Disadvantage:** It will not be able to detect all errors if the total sum remains constant.

Error Detecting Codes: Cyclic Redundancy Check (CRC)

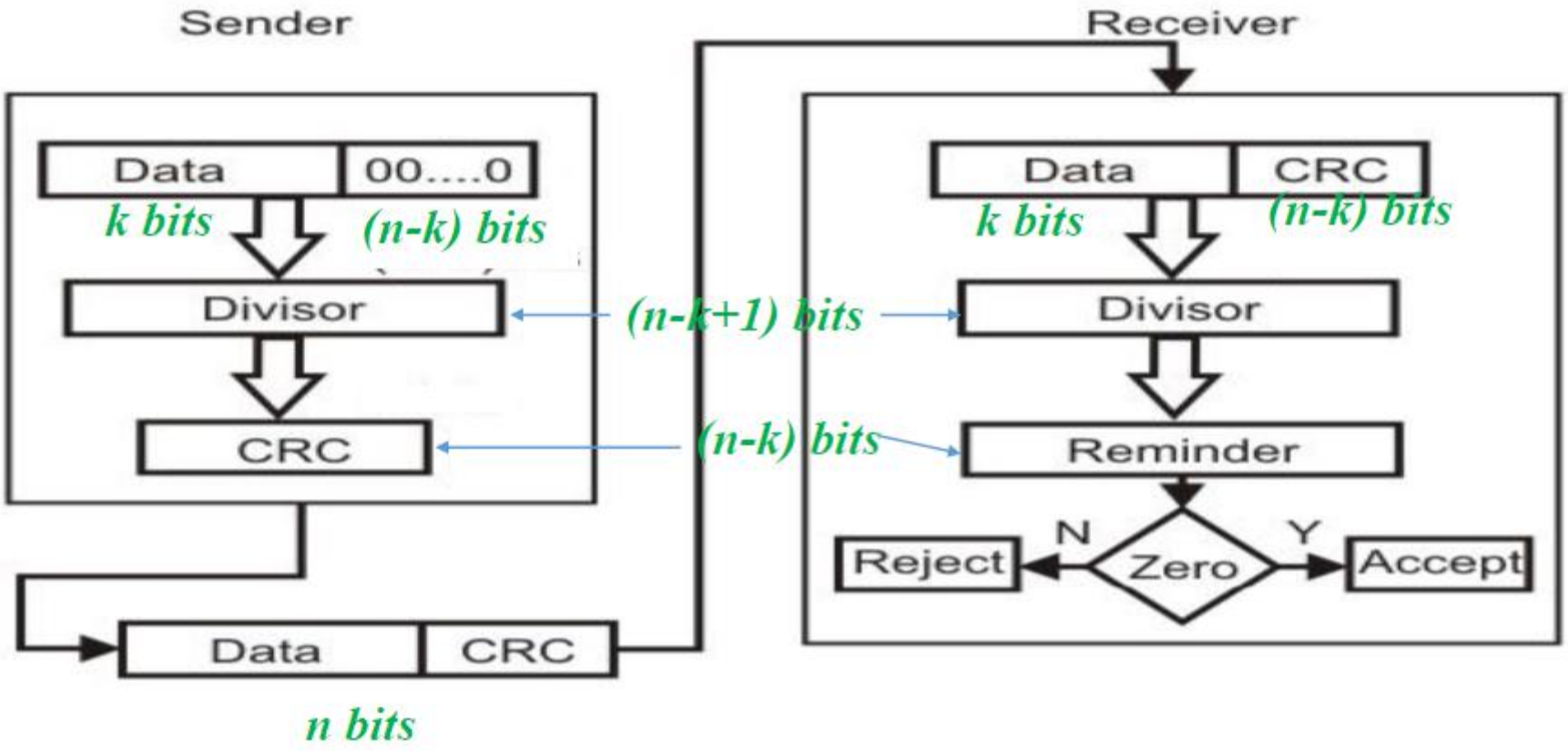
- One of the **most powerful** and commonly used error detecting codes.

- Concept behind it.....
  - Given a k-bit block of bit sequence, the sender generates an $(n\text{-}k)$ bit sequence, known as ***frame check sequence(FCS),*** so that the resulting frame, consisting of n bits, is exactly divisible by same ***predetermined number*** *(which is of $(n\text{-}k+1)$ bits).*
  - The receiver divides the incoming frame by that number and, if there is no remainder, it assumes that there was no error.

- Frame check sequence is also known as ***cyclic redundancy check bits.***

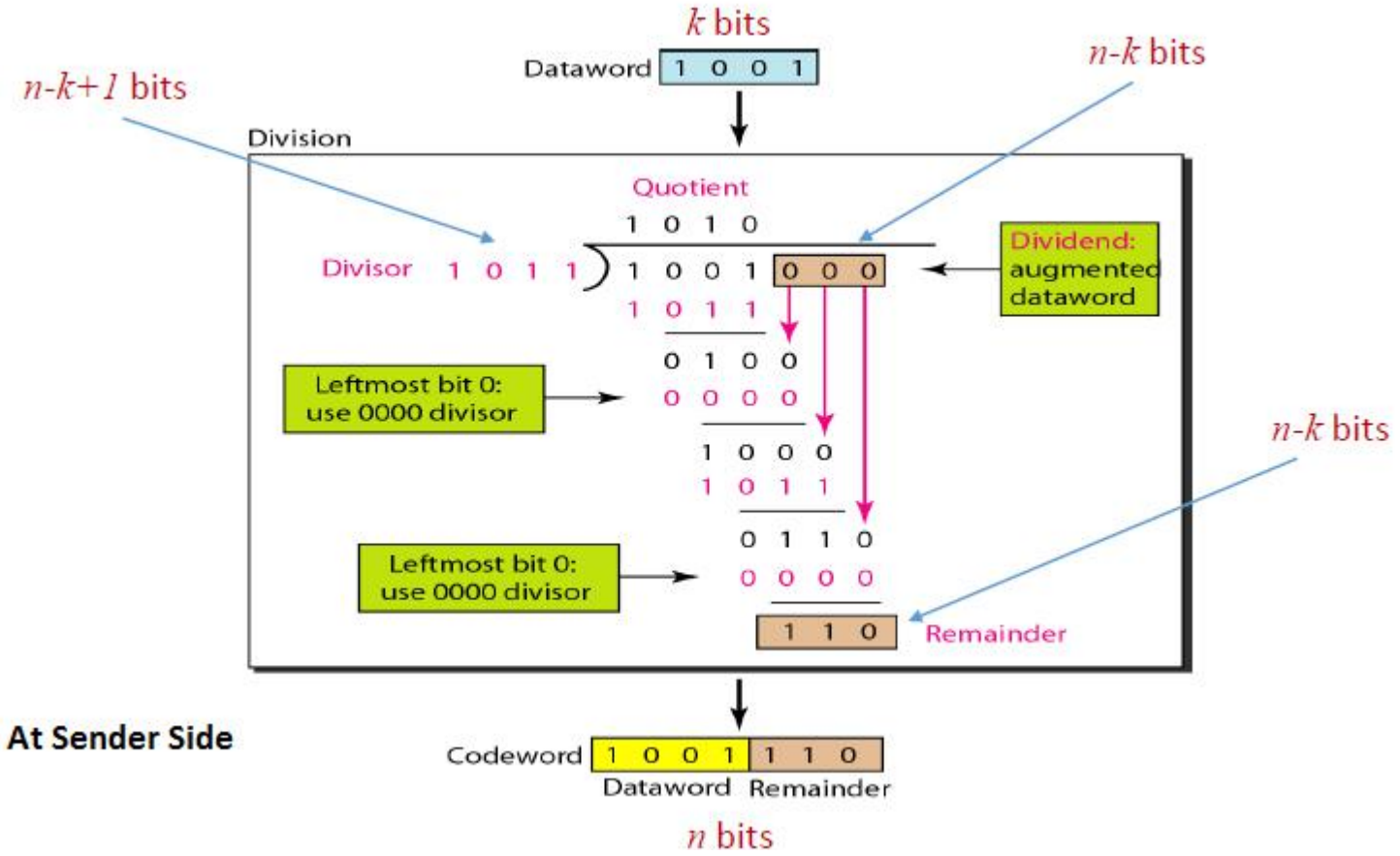


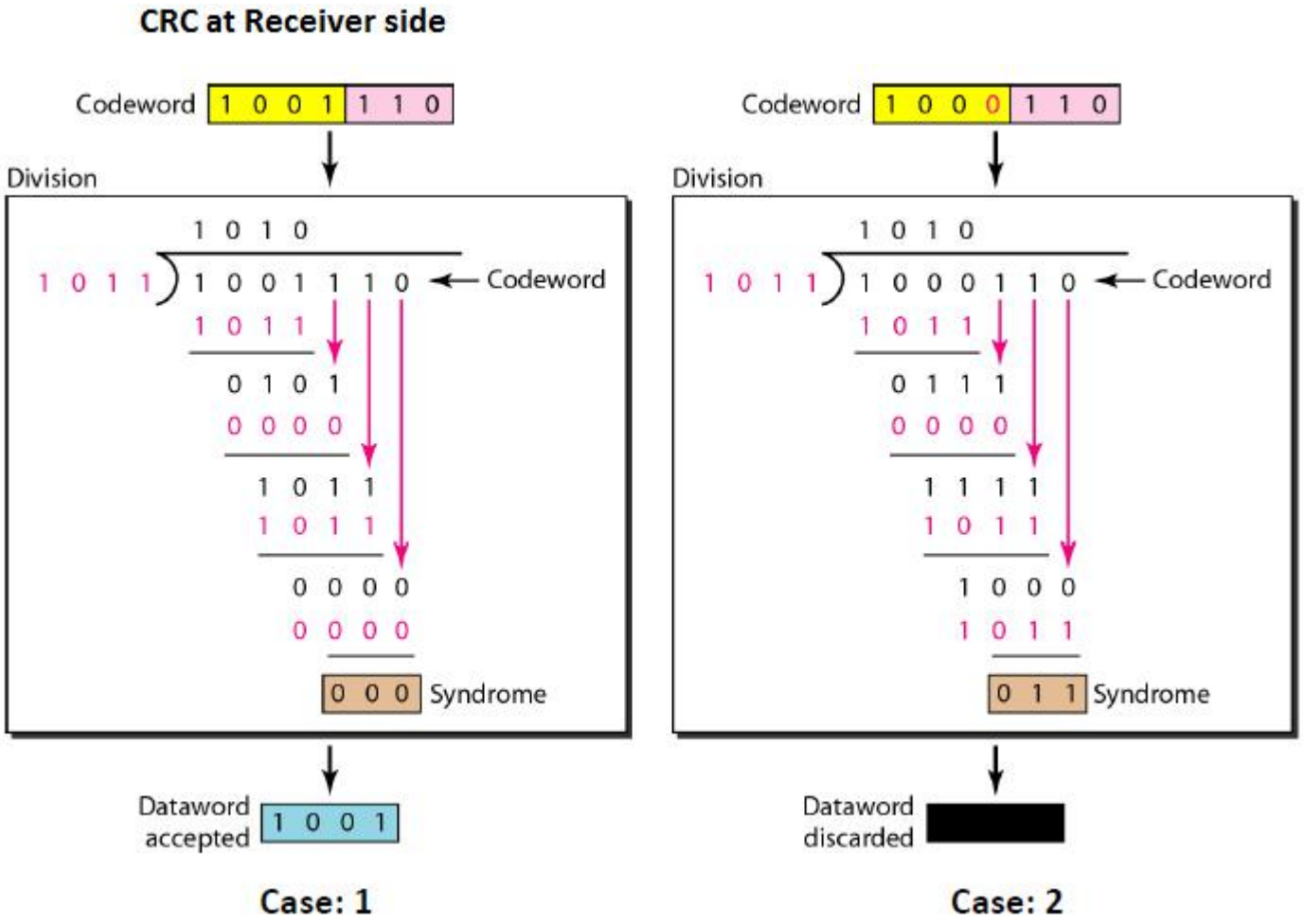
Basic scheme for Cyclic Redundancy Checking



- **Ex:** Consider a case where data word is 1001 (i.e., m=4 bits) and predetermined number is 1011. Determine FCS.
- **Sol:**

- Initially, we choose CRC as all zero bits. Since, predetermined number is of 4 bits, we choose 3 zero bits, i.e., 000 as CRC.
- So, we append 3 zeros to dataword and then it will be divided by 1011.

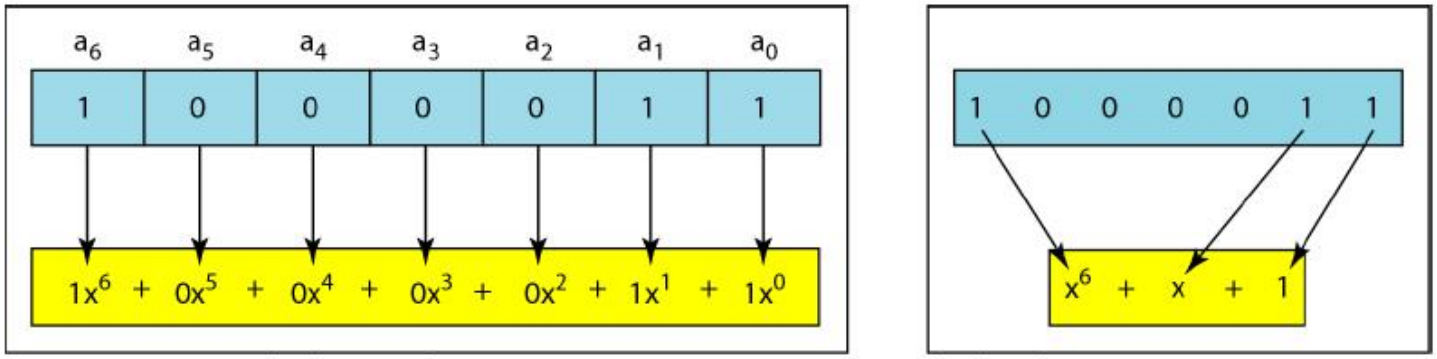



CRC at Receiver side
Codeword 1 0 0 1 1 1 0
Division
1 0 1 0
1 0 1 1) 1 0 0 1 1 1 0 ← Codeword
1 0 1 1
0 1 0 1
0 0 0 0
1 0 1 1
1 0 1 1
0 0 0 0
0 0 0 0
0 0 0 Syndrome
Dataword accepted 1 0 0 1
Case: 1
Codeword 1 0 0 0 1 1 0
Division
1 0 1 0
1 0 1 1) 1 0 0 0 1 1 0 ← Codeword
1 0 1 1
0 1 1 1
0 0 0 0
1 1 1 1
1 0 1 1
1 0 0 0
1 0 1 1
0 1 1 Syndrome
Dataword discarded
Case: 2

CRC code using Polynomial

- It is a better way to understand cyclic code.
- Using polynomial, analysis of cyclic code is easier as compared to binary representation.
- We can use polynomial to represent a binary word.
- Each bit from right to left is mapped onto a power term.
- The rightmost bit represents the “0” power term. The bit next to it the “1” power term, so on.
- If the bit is of value zero, the power term is deleted from the expression.

A polynomial to represent a binary word



a. Binary pattern and polynomial

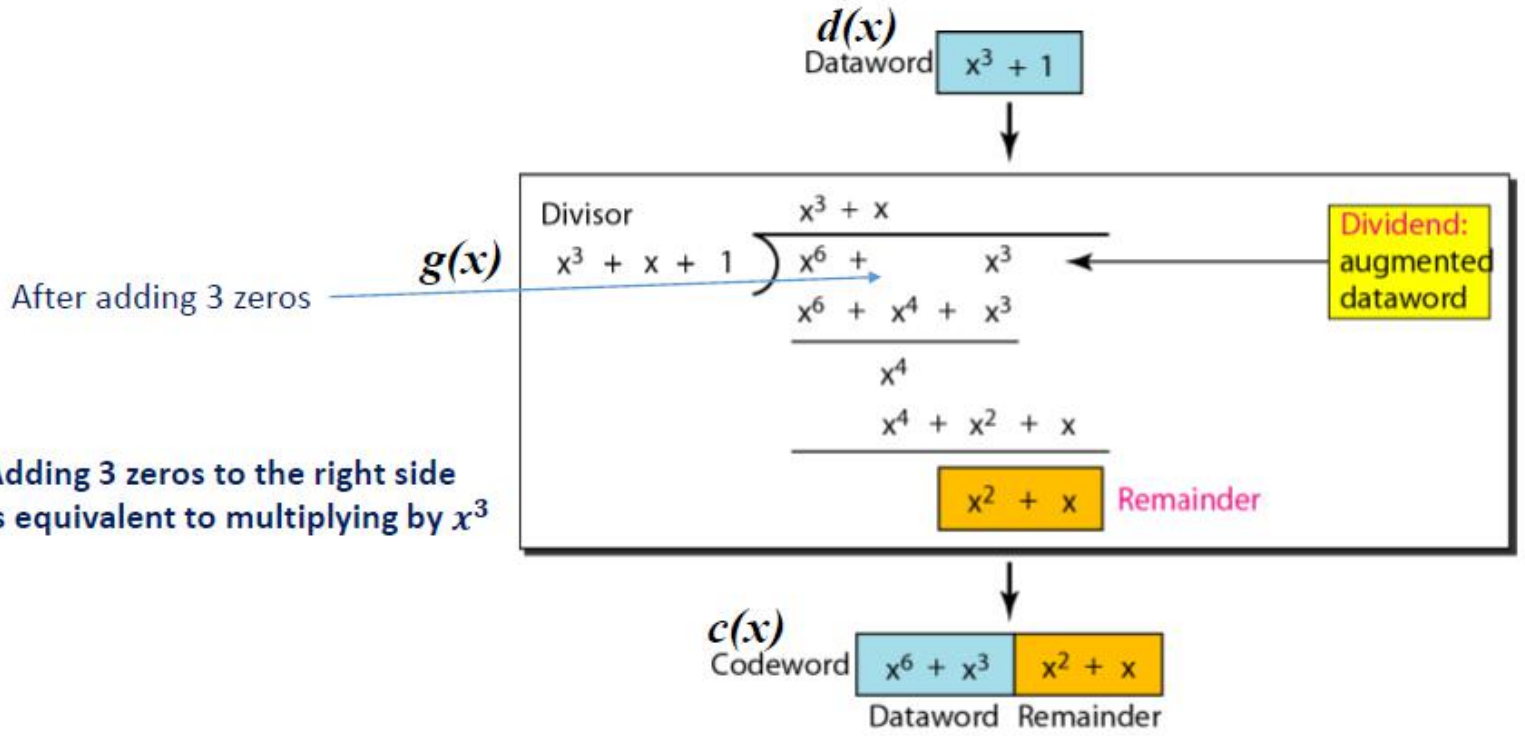
b. Short form

Note: for n bits, polynomial can have a maximum degree of n-1.

- We use the following notations in polynomial representation.
  - *Dataword: d(x)*
  - *Codeword: c(x)*
  - *Error: e(x)*
  - *Generator: g(x)*
  - *Syndrome: s(x)*

- ***Syndrome:*** *It is the remainder produced by the checker (at Rx. side). It is also treated as set of patterns accepted. For e.g., generally in CRC, we say that if syndrome is all 0's then no error; otherwise there is some error.*

Ex: Consider 1001 be the dataword and 1011 be the pattern (generator). Find CRC using polynomial approach.



CRC code: Key points

- In a cyclic code,
  - Divisor is called generator polynomial or simply generator, denoted by $g(x)$.
  - Remainder at the receiver side is called syndrome, denoted by $s(x)$.
  - Dataword and remainder together are called codeword, denoted by $c(x)$.
  - Error polynomial is represented as $e(x)$.
  - Received codeword is $c'(x)=c(x)+e(x)$.
- In cyclic code, those e(x) errors that are divisible by g(x) are not caught.

$$\textit{Received codeword } (c(x) + e(x))/g(x) = c(x)/g(x) + e(x)/g(x)$$

- Properties of the CRC code are determined by the choice of the generator polynomial. Choose g(x) such that as many error patterns e(x) as possible may be detected.

How to choose generator polynomial...

- Two main properties which must be satisfied by a generator polynomial.
  1) It should not be divisible by X.
  2) It should not be divisible by (X+1).

Performance:

- CRC can detect
  - All single-bit errors
  - All double-bit errors
  - All burst errors of less than the degree of the polynomial.
  - Most of the larger burst errors with a high probability.
  - For e.g., CRC-12 detects 99.97% of errors with a length 12 or more.

Standard Generator Polynomials

CRC = cyclic redundancy check

CCITT = Consultative Committee for International Telephony and Telegraphy

CRC-8:

ATM

$= x^8 + x^2 + x + 1$

CRC-16:

Bisync

$= x^{16} + x^{15} + x^2 + 1$

$= (x + 1)(x^{15} + x + 1)$

CCITT-16:

HDLC, XMODEM

$= x^{16} + x^{12} + x^5 + 1$

IEEE 802

CCITT-32:

$= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Error Correcting Codes:

Two basic approach for error correction

- **Forward error correction:** In this method, receiver can use an error-correcting code, which automatically corrects certain errors.
  - E.g., Hamming Code, BCH Code, etc.
- **Backward error correction:** In this method, when an error is discovered; the receiver can have the sender retransmit the entire data unit.
  - E.g., Go-Back-N ARQ, Selective Repeat ARQ, etc.

Hamming code, for 1-bit error

- **Requirements for error detection:** A code is an error detecting codes if and only if the minimum distance (Hamming distance) between any two code word is two.
- **Requirement for error correction:** For a code to be error-correcting, the minimum distance between any two code words must be more than two.

The # errors can be detected = dmin-1

and

The # errors can be corrected = (dmin-1)/2

Hamming Code: It is a binary linear block code denoted as *(n,k,d)*, *where,* $n \leq 2^r - 1$, *and* $n=k+r$.

k= # message bits, *r= # parity bits*

Q.1) How to decide number of parity bits for a given data/message?

It is decided by, $k + r \leq 2^r - 1$

NOTE:

- *The number of parity bits required may be same for two different data words with different number of bits.*
- Position of the parity bits is given by relation 2^r, where $r = 0,1,2,3....$
- For first parity bit, it is obtained by doing XOR operation of all those data bits whose position in binary has 1st bit 1 from the right side.

  Simillarly, for i^{th} parity bit, it is obtained by doing XOR operation of all those data bits whose position in binary has i^{th} bit 1 from the right side.

- **Q.4) How an error is corrected?**

Ans: For received data, again calculate value of each parity bit, now there can be two cases:

Case 1: If there is error in any one of the parity bits.

The parity bit which will be in error will differ from the old parity bit.

Case 2: If there is error in any one of the data bits.

Any two or more parity bits will differ from the old parity bits. In this case, the position of data bit in error is calculated by doing the sum of positions of respective parity bits.

- **Example:** Let us assume that data to be transmitted be **1011** (here k=4).

- Since k=4, r=3 satisfies equation 1. So, we have to add 3 parity bits.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1		1		
111	110	101	100	011	010	001

$$\because 2^0 = 1, 2^1 = 2, 2^2 = 4$$

- Parity bits will be at positions 1,2 and 4.

- **Values of parity bits:**

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1		1		
111	110	101	100	011	010	001

$$P_1 = D_3 \oplus D_5 \oplus D_7 = 1 \oplus 1 \oplus 1 = 1$$

$$P_2 = D_3 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

$$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	1	0	1
111	110	101	100	011	010	001

Data to be transmitted

- **Case 1: At the Rx side, if received data is**

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	1	1	0	1
111	110	101	100	011	010	001

$P_1 = D_3 \oplus D_5 \oplus D_7 = 1 \oplus 1 \oplus 1 = 1$ ..... *matched*

$P_2 = D_3 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$ .....*matched*

$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$ .....*not matched*

So, the error is in the 4th position i.e. P4

- **Case 2: At the Rx side, if received data is**

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	0	0	1
111	110	101	100	011	010	001

$P_1 = D_3 \oplus D_5 \oplus D_7 = 0 \oplus 1 \oplus 1 = 0$ .....*not matched*

$P_2 = D_3 \oplus D_6 \oplus D_7 = 0 \oplus 0 \oplus 1 = 1$ .....*not matched*

$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$ ..... *matched*

So, the error is at position 3(=1+2) i.e. D3