- Error :

  - If data sent is not the same as data received, this means an error has occurred.
  - There are 2 types of errors :

    1. Single bit error :
       - Only 1 bit of a given data unit has changed from 0 to 1 or 1 to 0
       - Eg. Sent : 1011000
         Recv. : 1010000

    2. Burst error :
       - 2 or more bits in the given data unit has changed from 0 to 1 or 1 to 0.
       - Eg. Sent : 1100100
         Recv. : 1010101

  - Burst errors have a higher chance of occurring.

  No. of corrupted or affected bits = Data rate × Noise duration

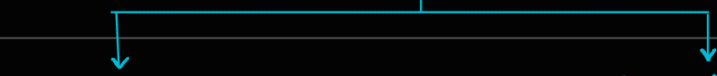
  Eg. ① Data rate = 1 Kbps
  Noise duration = 0.1 s
  No. of corrupted bits = $10^3 \times 10^{-1}$ bits
  = 100 bits

  ② Data rate = 1 Mbps
  Noise duration = 0.1 s
  No. of corrupted bits = $10^6 \times 10^{-1}$ bits
  = 10^5 bits

  (Clearly, burst error is more frequent)

- Error control :

Error Control



Error Detection

1. Simple parity
2. 2D parity
3. Checksum (only 16 extra bits required) : Used in NL and TL
4. Cyclic Redundancy Check (CRC) (only 32 extra bits) : Used in DLL

- once an error is noticed , given data is DISCARDED
- Asks for RETRANSMISSION

Error Correction

1. Hamming code
   - Capability of correcting error
   - Hamming code can correct SINGLE BIT ERRORS
   - Does not require retransmission.

Parity Checks

- **Single Parity Check :**

  - Extra bit called the PARITY BIT is sent ALONG with the original data bit
  - There are 2 types of parity : (n = no. of 1's in the original data)

1. Even parity :
   - We want even no. of 1's in the MODIFIED data
   - If n is odd, add a 0, else add a 1
2. Odd parity :
   - We want odd no. of 1's in the MODIFIED data
   - If n is odd, add a 1, else add a 0

- For eg.

Data	n	Even parity	Odd parity
0101	2	01010	01011
1101	3	11011	11010
01110110	5	011101101	011101100

- **2D parity check :**

  - Parity check bit is calculated for each row and column and is sent along with the original data
  - The receiver also calculates the parity along row and column and checks with the original parity bits
  - Eg.

Sent data stream :

10011001	11100010	00100100	10000100

In 2D form : (even parity scheme)

									Parity bits
	1	0	0	1	1	0	0	1	→ 0
	1	1	1	0	0	0	1	0	→ 0
	0	0	1	0	0	1	0	0	→ 0
	1	0	0	0	0	1	0	0	→ 0
	↓	↓	↓	↓	↓	↓	↓	↓	↓
Parity bits	1	1	0	1	1	0	1	1	→ 0

Rcvd. data stream :

100110010	111000100	001001000	100001000	11011011

- Few points to note :

Type of error	Detection	Correction
1-bit error	✓	✓
2-bit error	✓	X
3-bit error	✓	X
4-bit error	Maybe	X
> 4-bit error	X	X

4-bit error of the form: Switch 2 bits in a row and switch 2 bits in the corresponding columns CAN NOT be detected. Similar case applies for > 4-bit errors.

Checksum

- **Checksum**:
  - First, the data is divided by equal size frame (pad if necessary)
  - Checksum detects ALL ERRORS in odd no. of error bits and MOST ERRORS in even no. of error bits
  - Process:
    - Sender: Take sum of data (using 1's complement addition) and take its 1's complement.
      Send this ACROSS with the original data
    - Receiver: Take sum of sent data (using 1's complement addition) and take its 1's complement.
      If it is 0, then no error, else error

Sum using 1's complement is basically wrapping around the LAST CARRY and ADDING it to the result. We repeat this process till the last carry bit = 0.

Eg. ① Sender: 1010100110111001

Data: | 10101001 | 10111001 |

Sum using 1's complement addition:

```
  1 1 1  1
  1 0 1 0 1 0 0 1
+ 1 0 1 1 1 0 0 1
-----------------
1 0 1 1 0 0 0 1 0
+               1
-----------------
  0 1 1 0 0 0 1 1   → 8 bit checksum
```

Ones' complement of checksum: 10011100

Sent data: | 10101001 | 10111001 | 10011100 |

Sum using 1's complement addition: (Receiver end)

```
    1 1 1
  1 0 1 0 1 0 0 1
+ 1 0 1 1 1 0 0 1
-----------------
1 1 0 0 1 1 1 0 0
+               1
-----------------
  1 1 1 1 1 1 1 1   → 8 bit Sum
```

→ Take its ones' complement: 00000000 = 0

⇒ NO ERROR

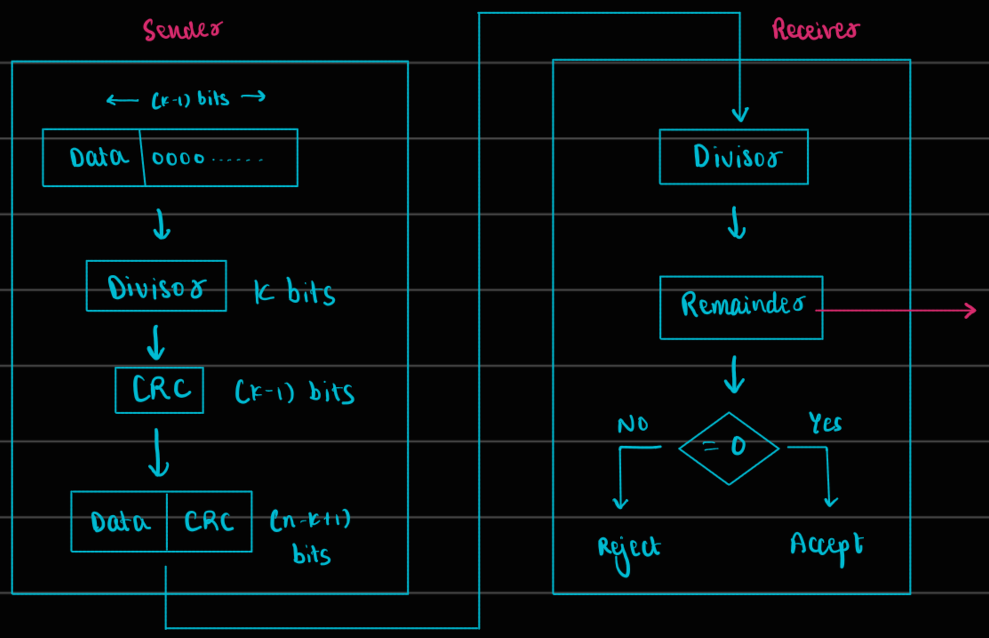
Cyclic Redundancy Check (CRC)

- **Cyclic Redundancy Check (CRC)**

  - length of dataword (original data) = n bits
  - length of divisor = k bits
  - Append (k-1) zeroes to the RIGHT of original message
  - Perform modulo 2 division
  - Remainder of this division = CRC. (Obviously, CRC must be of (k-1) bits)
  - Codeword = original data with appended 0's + CRC (+ is ADDITION NOT APPEND)
  - length of codeword = (n-k+1) bits

PTO

- **Process**:



Remainder at receiver end is also called SYNDROME

Eg. Data = 1001001 and divisor (or) CRC generator = 1101

k = 4 ⇒ Append 3 0's to data : Data = 1001001000

Sender Side:

• → leading 1

```
1101 ) 1001001000
  XOR  1101↓↓↓↓↓↓
       0100001000
  XOR   1101↓↓↓↓↓
        010101000
  XOR    1101↓↓↓↓
          01111000
  XOR      1101↓↓↓
           0010000
  XOR        1101↓
              01010
  XOR          1101
               0111
```

PROCEDURE:

- •1 Start from LEADING 1 and perform XOR with divisor.
- •2 Copy all remaining bits from the dividend.
- •3 Repeat •1 and •2 till the remainder is not divisible by the divisor

⇒ CRC = 111

length of CRC = (k-1)

= (4-1) = 3 bits ✓

Dataword = 1001001000

\+ CRC = 111

Codeword = 1001001111

[Or you could just replace the added (k-1) 0's with the CRC directly]

Receiver Side : Received data = 1001001111

Case ① : If correct data is transmitted :

```
1101 ) 1 0 0 1 0 0 1 1 1 1
XOR    1 1 0 1 ↓ ↓ ↓ ↓ ↓ ↓
       ---------------------
       0 1 0 0 0 0 1 1 1 1
XOR      1 1 0 1 ↓ ↓ ↓ ↓ ↓
         -----------------
         0 1 0 1 0 1 1 1
XOR        1 1 0 1 ↓ ↓ ↓ ↓
           -------------
           0 1 1 1 1 1 1
XOR          1 1 0 1 ↓ ↓ ↓
             -----------
             0 0 1 0 1 1 1
XOR            1 1 0 1 ↓
               ---------
               0 1 1 0 1
XOR              1 1 0 1
                 -------
                 0 0 0 0
```

Remainder = 0 ⇒ ACCEPT

Case ② : If incorrect data is transmitted :

```
1101 ) 1 0 0 1 0 1 1 1 1 1
XOR    1 1 0 1 ↓ ↓ ↓ ↓ ↓ ↓
       ---------------------
       0 1 0 0 0 1 1 1 1 1
XOR      1 1 0 1 ↓ ↓ ↓ ↓ ↓
         -----------------
         0 1 0 1 1 1 1 1
XOR        1 1 0 1 ↓ ↓ ↓ ↓
           -------------
           0 1 1 0 1 1 1 1
XOR          1 1 0 1 ↓ ↓ ↓
             -----------
             0 0 0 0 1 1 1
```

Remainder ≠ 0 ⇒ REJECT

- Polynomial notation in CRC :

Data word = d(x) ; Codeword = c(x) ; Generator = g(x) ;

Syndrome = s(x) ; Error = e(x)

PTO

Steps:

- 1 Determine degree (highest power) of $g(x)$. Let this be r.
- 2 Determine $x^r d(x)$
- 3 Determine the remainder by dividing $x^r d(x)$ by $g(x)$
- 4 Codeword = $x^r d(x)$ + remainder

Eg. ① Data word = 1001001 = $1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 1 \cdot 1 = x^6 + x^3 + 1$ (same eg. as before)

Generator = 1101 = $1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1 \cdot$ $= x^3 + x^2 + 1$

Degree of generator = 3 . $x^r d(x) = x^3 (x^6 + x^3 + x^3) = x^9 + x^6 + x^3$

$$x^3 + x^2 + 1 \;\big)\; x^9 + x^6 + x^3 \quad (x^6 + x^5 + x^4 + x^3 + x^2 + 1)$$

$$x^9 + x^8 + x^6$$

$$x^8 + x^3$$

$$x^8 + x^7 + x^5$$

$$x^7 + x^3 + x^5$$

$$x^7 + x^6 + x^4$$

$$x^6 + x^5 + x^4 + x^3$$

$$x^6 + x^5 + x^3$$

$$x^4$$

$$x^4 + x^3 + x$$

$$x^3 + x$$

$$x^3 + x^2 + 1$$

$$x^2 + x + 1$$

⇒ Codeword = $x^r d(x)$ + Remainder

= $x^9 + x^6 + x^3 + x^2 + x + 1$

= 1001001111

(same as before)

⇒ Remainder or CRC = $x^2 + x + 1$ = 111 (same as before)

PTO

Eg. ① CRC. Generator polynomial = x^3+x^2+1. message $m_4m_3m_2m_1m_0$ = 11000

check bits $c_2c_1c_0$. Transmitted message = ?

```
1011 ) 11000 000
XOR    1011
       01110000
XOR     1011
        010 1000
XOR      1011
         00 0100
```

⇒ CRC = 100

⇒ Transmitted message = 11000100

② Generating function : $g(x) = x^4+x+1$

message function : $m(x) = x^7+x^6+x^4+x^2+x$

Transmission function : $t(x)$ = ?

$$x^4+x+1 \;\overline{)}\; x^{11}+x^{10}+x^8+x^6+x^5$$

x^7+x^6+x

$x^{11}+x^8+x^7$

$x^{10}+x^7+x^6+x^5$

$x^{10}+x^7+x^6$

x^5

x^5+x^2+x

x^2+x

⇒ CRC = x^2+x

⇒ $t(x) = x^r\,m(x) + CRC$

$= x^{11}+x^{10}+x^8+x^6+x^5+x^2+x$

③ Message = 1010001101 ; Divisor polynomial = $x^5+x^4+x^2+1$; CRC = ?

Divisor = 110101 . Degree = 5

PTO

```
                                                    ⇒ CRC = 01110
110101 ) 10100011010000000
XOR      110101
         01110111010000000
XOR       110101
          00111010100000000
XOR         110101
            00111111000000
XOR           110101
              00101110000
XOR             110101
                01100100
                110101
                 0001110
```

- Error detecting capability of CRC:

  - 1. If the generator has more than one term and the coefficients 1, ALL SINGLE BIT ERRORS can be detected
  - 2. If the generator contains a factor of $(x+1)$, it DETECT ALL ODD NUMBERED ERRORS.
  - 3. If the generator can not divide x^t+1 $(0 \le t \le n-1)$ ALL DOUBLE ERROR can be detected

- A good polynomial generator:

  - 1. Should have atleast 2 terms
  - 2. Should have the coefficient of x^0 be 1.
  - 3. Should have a factor $(x+1)$
  - 4. Should not divide x^t+1 $(0 \le t \le n-1)$

Hamming Distance

Hamming Distance :

- Hamming dist. (HD) b/w 2 binary strings of equal length is the no. of differences b/w corresponding bits i.e. number of 1's in the XOR of the 2 binary strings
- Minimum Hamming Distance = min (HD of all pairs of given codewords)

For eg. Codewords : a: 010, b: 101, c: 110, d: 001. Min. HD = ?

min. HD = min(d(a,b), d(a,c), d(a,d), d(b,c), d(b,d), d(c,d))

= min (3, 1, 2, 2, 1, 3)

= 1

Minimum Hamming Distance for Error Detection :

MHD to DETECT 'd' bit error	= d+1
MHD to CORRECT 'd' bit error	= 2d+1 → If you get decimal here, take FLOOR (round down)

Eg. ① Codewords : a: 00000, b: 01011, c: 10101, d: 11110 MHD = P and maximum no. of erroneous bits that can be corrected = q. P and q = ?

MHD = min (3, 3, 4, 4, 3, 3) = 3

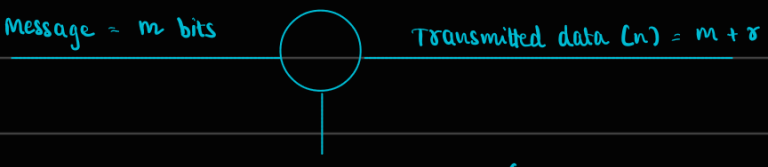
MHD to correct 'd' bit error = 2d+1 ⇒ 3 = 2d+1 ⇒ d = 1 = q

P = 3 and q = 1

Hamming Code

- **Hamming Code :**

- Can CORRECT 1 bit errors and can DETECT 2 bit errors.
- Used for error correction



According to Hamming Code, no. of redundant bits must satisfy : $m + r + 1 \leq 2^r$

where r = lower limit

- Positions of redundant bits : 2^i where $i \geq 0$
- Calculating the value of redundant bits : (say P_i where $i = 2^j$; $j \geq 0$)
  - 1 Pick the values from all the positions where i^{th} bit is 1
  - 2 Calculate parity of the chosen values (including P_i obviously)
  - 3 P_i = Bit after applying even or odd parity scheme to chosen values

> If parity scheme is not given ALWAYS assume ODD PARITY, it is always preferred more than even parity.

Eg. message = 1010111 $\Rightarrow$ m = 7

So, $7 + r + 1 \leq 2^r \Rightarrow 8 + r \leq 2^r$. By trial and error, we get the minimum value of r = 4

So, no. of bits in transmitted data = m + r

Assuming EVEN PARITY

P_1	P_2	3	P_4	5	6	7	P_8	9	10	11
		1		0	1	0		1	1	1

16 8 4 2 1		
0 0 0 0 0	0	
0 0 0 0 1	1	P_1
0 0 0 1 0	2	P_2
0 0 0 1 1	3	P_1 P_2
0 0 1 0 0	4	P_4
0 0 1 0 1	5	P_1 P_4
0 0 1 1 0	6	P_2 P_4
0 0 1 1 1	7	P_1 P_2 P_4
0 1 0 0 0	8	P_8
0 1 0 1 0	9	P_1
0 1 0 1 1	10	P_2
0 1 1 0 0	11	P_1 P_2

P_1: P 1 3 5 7 9 11

1 1 0 0 1 1

P_2: P 2 3 6 7 10 11

0 1 1 0 1 1

P_3: P 4 5 6 7

1 0 1 0

P_4: 8 9 10 11

1 1 1 1

Parity Bits : $P_8 P_4 P_2 P_1$

: 1101

Transmitted Data :

10110101111

① If receiver received uncorrupted data :

1 2 3 4 5 6 7 8 9 10 11

Received data : 1 0 1 1 0 1 0 1 1 1 1

Even parity that is why Even : $P_i = 0$ if no. of 1's are even. It would be 1 if we follow Odd parity scheme. Obviously.

P_1: 1 3 5 7 9 11

1 1 0 0 1 1 → Even : $P_1 = 0$

P_2: 2 3 6 7 10 11

0 1 1 0 1 1 → Even : $P_2 = 0$

P_4: 4 5 6 7

1 0 1 0 → Even : $P_4 = 0$

P_8 : 8 9 10 11

1 1 1 1 → Even : $P_8 = 0$

Parity bits = $P_8 P_4 P_2 P_1$ = 0000 ⇒ NO ERRORS FOUND

② If receiver receives corrupted data (1-bit error) :

1 2 3 4 5 6 7 8 9 10 11

Received data : 1 0 1 1 0 1 0 1 0 1 1

PTO

P_1 : 1 3 5 7 9 11

1 1 0 0 0 1 → Odd : $P_1 = 1$

P_4 : 4 5 6 7

1 0 1 0 → Even : $P_4 = 0$

P_2 : 2 3 6 7 10 11

0 1 1 0 1 1 → Even : $P_2 = 0$

P_8 : 8 9 10 11

1 0 1 1 → Odd : $P_8 = 1$

Parity bits = $P_8P_4P_2P_1$ = 1001 ⇒ 1-BIT ERROR FOUND AT POSITION : $(1001)_{10} = 9$

⇒ 9^{th} bit was corrupted.

Corrected data by receiver : 1 0 1 1 0 1 0 1 1 1

② If receiver receives corrupted data (2-bit error) :

1 2 3 4 5 6 7 8 9 10 11

Received data : 1 0 1 1 1 1 0 1 0 1 1

P_1 : 1 3 5 7 9 11

1 1 1 0 0 1 → Even : $P_1 = 0$

P_4 : 4 5 6 7

1 1 1 0 → Odd : $P_4 = 1$

P_2 : 2 3 6 7 10 11

0 1 1 0 1 1 → Even : $P_2 = 0$

P_8 : 8 9 10 11

1 0 1 1 → Odd : $P_8 = 1$

Parity bits = $P_8P_4P_2P_1$ = 1100 ⇒ 2-BIT ERROR FOUND. Position can NOT be found :

⇒ $(1100)_{10} = 12$. There are only 11-bits in the received data

Shortcut to find the positions for parity bit P_i :

START FROM i, PICK 2^i POSITIONS then SKIP THE NEXT 2^i POSITIONS, PICK THE NEXT 2^i POSITIONS then SKIP THE NEXT 2^i POSITIONS .... till all the positions are exhausted. It is OKAY if no. of positions $< 2^i$ in the last iteration.

for eg. P_1 : Pick positions : (START AT) 1 , (SKIP 2) , 3 , (SKIP 4) , 5 , (SKIP 6) , 7 , ...

P_2 : Pick positions : (START AT) 2 , 3 , (SKIP 4,5) , 6 , 7 , (SKIP 8,9) , 10 , 11 , ...

P_4 : Pick positions : (START AT) 4 , 5 , 6 , 7 (SKIP 8,9,10,11) , 12 , 13 , 14 , 15 ....

P_8 : Pick positions : (START AT) 8,9,10,11,12,13,14,15 (SKIP 16,17,18,19,20,21,22,23) ,

24,25,26,27,28,29,30,31, ...

and so on.

Eg. ① Assume 12-bit Hamming code. Data bits : $\overset{d_8\ d_7\ d_6\ d_5\ d_4\ d_3\ d_2\ d_1}{1\ 1\ 0\ x\ 0\ 1\ 0\ 1}$ and Check bits : $\overset{c_8\ c_4\ c_2\ c_1}{y\ 0\ 1\ 0}$

x and y = ?

message :	P_1	P_2	3	P_4	5	6	7	P_8	9	10	11	12
	0	1	1	0	0	1	0	y	x	0	1	1

C_1 : 1 3 5 7 9 11 → 0 1 0 0 x 1

C_4 : 4 5 6 7 12 → 0 0 1 0 1 → Even

C_2 : 2 3 6 7 10 11 → 1 1 1 0 0 1 → Even

C_8 : 8 9 10 11 12 → y x 0 1 1

(Deducing from C_2 and C_4 : EVEN PARITY SCHEME is used)

$\Rightarrow x = 0$ and $y = 0$

② For a single error detecting Hamming code, the code length for 12-bits of data is ?

m = 12. WKT, $m + r + 1 \leq 2^r \Rightarrow 13 + r \leq 2^r$. By inspection, $r = 5$

$\Rightarrow$ length of code word = 12 + 5 = 17

③ Hamming code. Minimum parity bits required for 60 data bits = ?

$m = 60$. WKT, $m + r + 1 \leq 2^r \Rightarrow 61 + r \leq 2^r$. By inspection, $r = 7$

$\Rightarrow$ No. of parity bits required = 7