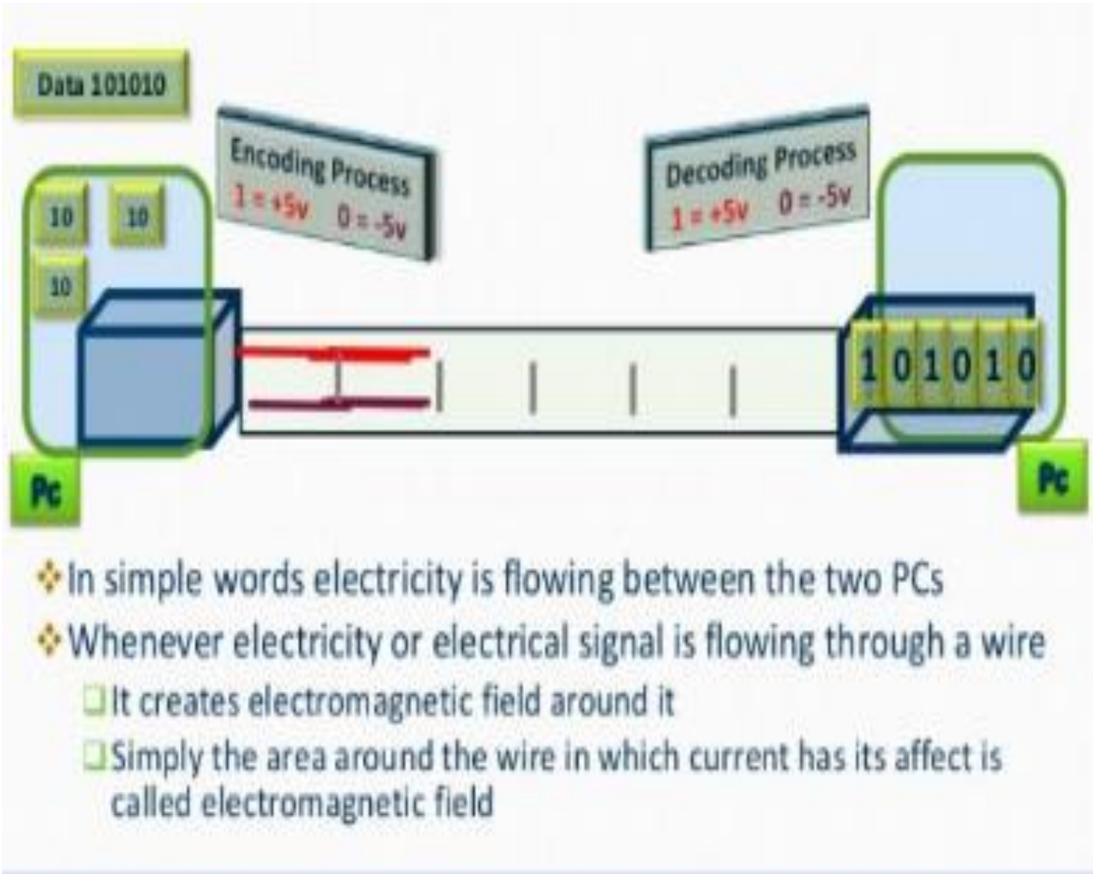
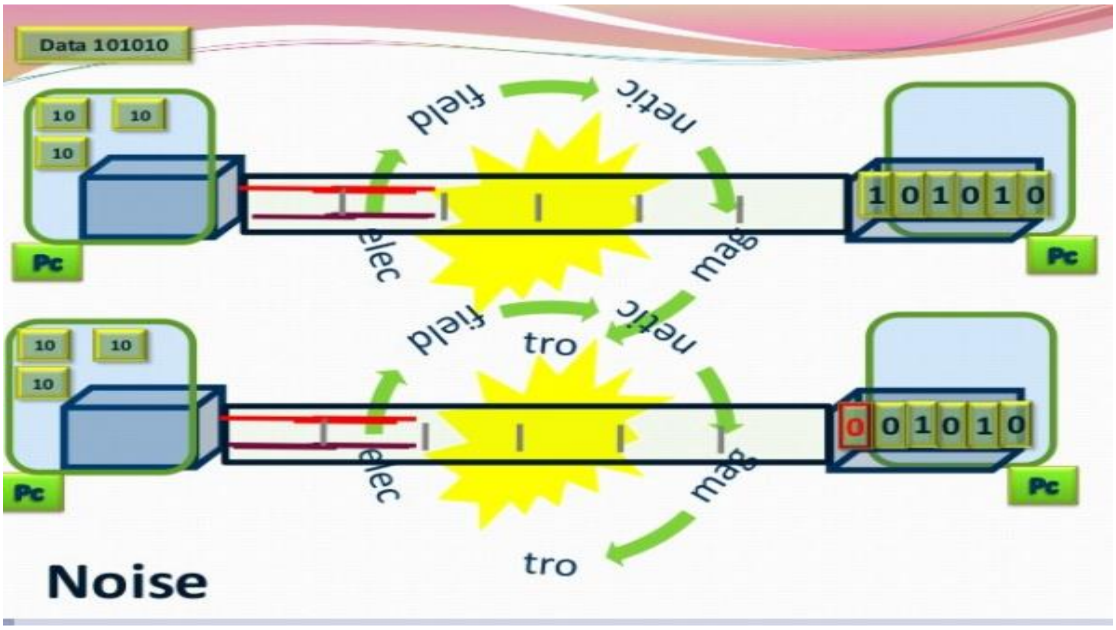
Errors Checking and Correction

Data transmissions are suffering from electromagnetic interference (called noise), and the longer the cable, the noisier the signal. This can cause a transmission error: when a bit is corrupted, a logic 1 is read as a logic 0, or vice versa.

It can be shown as the following :



So that it will be :



These types of noise are called electrical interference as we mentioned before, and it can be solved using several strategies:

- Parity checks
- Majority voting
- Check digits

Parity checks:

Parity bit examples

sequence of seven bits	with eighth even parity bit:	with eighth odd parity bit:
0100010	01000100	01000101
1000000	10000001	10000000

To allow the receiving end to detect errors in data transmission, an extra bit, called a parity bit, is introduced. The parity bit is added either at the end (as the least significant bit) or at the start (as the most significant bit) of a byte. The parity bit does not represent data; instead, it is set to make the total number of 1s in the byte odd for odd

parity, or even for even parity. The type of parity (odd or even) and the position of the parity bit are specified within the communication protocol. As long as the receiver knows whether the transmitter is using odd or even parity, it can detect whether a single bit error has occurred by summing the bits and finding out whether the total is odd or even.

Parity bits are typically used in the transmission of 7-bit standard ASCII codes, which means that only seven bits are used to send actual data. Including a parity bit allows the receiver to detect the presence of an error, but not to detect the exact position of the erroneous bit. Also, if more than one-bit changes during the transmission, the receiver might not be able to detect an error in the transmission. For example, when using even parity, if two 1s change to 0s, the total number of 1s would remain even.

So that to overcome the problem of parity bit we can use other strategy which is majority voting:

Majority voting:

Unlike parity check majority voting can repair the errors where each bit is transmitted 3 times to make it easier for the computer to detect errors.

Ruels :

1. Each bit is **transmitted 3 times**
2. If a set of 3 (1 bit) doesn't have the same three values, **majority voting** will show and fix errors.

    e.g. 010 – '1' is the error therefore the transmission should be 000 according to a majority vote

Example of using Majority Voting

Original binary 8 bit code → 11001010
Transmission (with errors)
→101,111,001,010,110,100,011,001
Correction → 111,111,000,000,111,000,111,000
Original code executed → 11001010

But also majority voting has some disadvantages:

Disadvantages
• If there is more than one error in one bit, it will not be detected and the computer will correct it incorrectly assuming that it is right.
• The transmission is 3 times longer than what you want to send
• Increased processing time

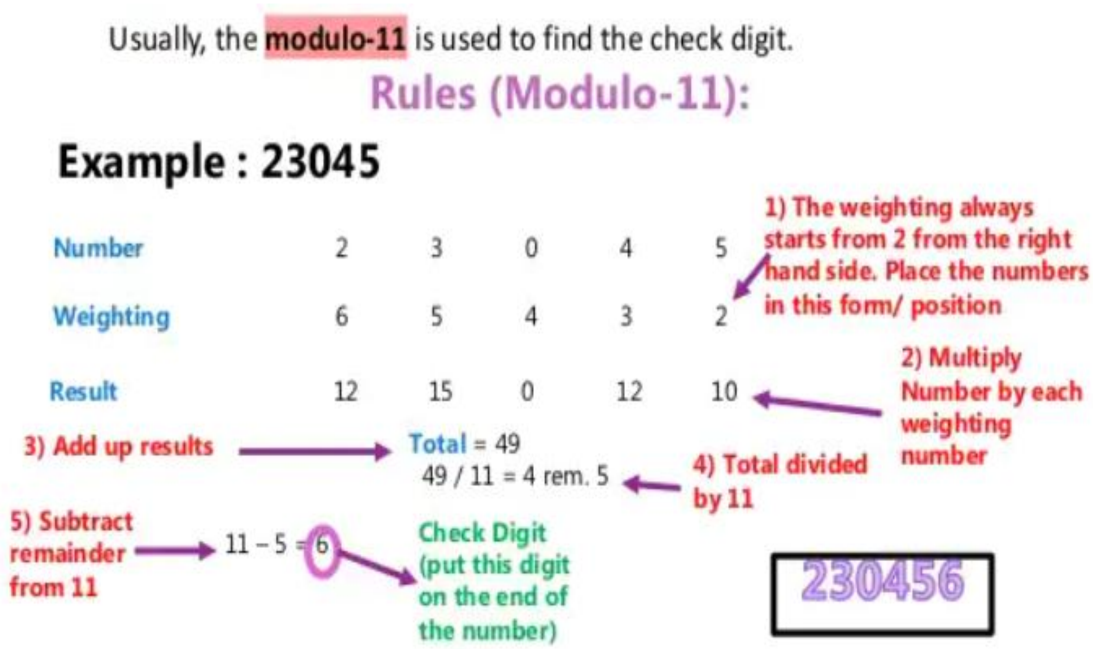
Check digits :

Check digits are similar to parity bits in that a value, calculated based on the data itself, is added to the transmitted data.

There is no fixed method for calculating a check digit and different systems will use their own methods.

An example of a method for calculating a check digit is the modulo function, this is used in barcodes to ensure that the data has been scanned correctly.A digit is added to the end of binary data to check if the data is accurate .

Module 11 method :



Another check digit example :

1) 73409

Number	7	3	4	0	9
Weighting	6	5	4	3	2
Result	42	15	16	0	18

Total = 91

91 / 11 = 8 Rem. 3

11 – 3 = 8

Check Digit

$\boxed{734098}$