January 21, 2026

Binary to decimal

10.101

$= 1\times2^1 + 0\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

$= 2.625_{10}$

Decimal to Binary

2.625

10

$$\begin{array}{r} 0.625 \\ 2\times \\ \hline 1.250 \end{array}$$

1 ~ 1.250

$$\begin{array}{r} 0.25 \\ 2\times \\ \hline 0.5 \end{array}$$

0 ~ 0.5

$$\begin{array}{r} 0.5 \\ 2\times \\ \hline 1.0 \end{array}$$

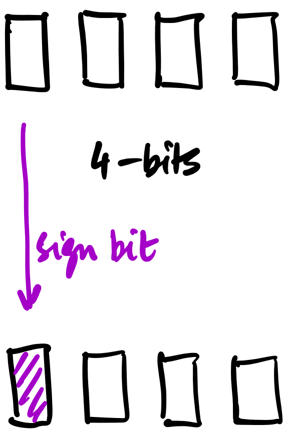
1 ~ 1.0

0 → Nothing more to do.

10.101_2

Q. How do we store negative numbers?

1



0 → +ve

1 → -ve

- unique values $2^4 = 16$
- The range of +ve numbers that you can represent $0 - 2^4 - 1 = 0 - 15$ $[0, 15]$

- The range of +ve and negative numbers that I can represent is $-7 - +7$, $[-7, 7]$

We only represent 15 unique values. ∵ 1000 & 0000 "0"

(2) Excess 7 representation.

stored value = actual value + 7

0 0 0 0	0	→ −7
0 0 0 1	1	
0 0 1 0	2	
0 0 1 1	3	→ actual value = stored value − 7 = 3 − 7 = −4
0 1 0 0	4	
0 1 0 1	5	
0 1 1 0	6	
0 1 1 1	7	
1 0 0 0	8	
1 0 0 1	9	
1 0 1 0	10	
1 0 1 1	11	
1 1 0 0	12	
→ 1 1 0 1	13	
1 1 1 0	14	
1 1 1 1	15	→ 8

This scheme can represent numbers [−7, 8].

* 32-bit Floating Point (IEEE 754)

Sign bit

Excess 127

[−127, 128]

Does not encode the leading 1. i.e., the 1 on the left side of decimal place.

Implicit 1.

Aside:

$23.56_{10} \rightarrow 2.356 \times 10^1$

$101.11_2 \rightarrow 1.0111 \times 2^2$

$0.1101 \rightarrow 1.101 \times 2^{-1}$

normal representation

standard scientific notation.

Example: store -13.625_{10}

(i) sign bit 1

(ii) $13.625_{10} = 1101.101_2$

(iii) 1.101101×2^3 normalized form.

(iv) Fraction : 101101

(v) Exponent : 3
(actual value)

Excess 127

$\boxed{\text{stored value} = \text{actual value} + 127}$

$= 3 + 127 = 130_{10} \rightarrow 1000\ 0010$

1	1000 0010 ~~EXPONENT 8 bits~~	101101 ~~MANTISSA/FRACTION 23 bits~~

31 30 23 22 0

Exercise : Convert the following Floating point numbers that is stored using 32-bits into decimal.

0x C148 0000

Hexadecimal

Error Detection

detect

correction

Parity bit : a single redundant bits that can be used to detect if a bit was changed.

① Even parity: total # 1's in a message has to be even.

$\boxed{10111100}$ $\boxed{0}$

10111000 ✓

11111000 ✗

11111010 ✓

② Odd parity: total # 1's in a message has to be odd.

$\boxed{1011100}$ $\boxed{1}$ 10111001 ✓

Hamming Distance

Given two strings of equal length, it counts the positions where the corresponding symbols are different.

a: 1 $\boxed{0}$ 0 1 1 $\boxed{1}$ 0 1
b: 1 $\boxed{1}$ 0 1 1 $\boxed{0}$ 0 1

hamming distance = 2

Parity bit can only detect errors where Hamming distance is 1.

→ One way to improve resistance to errors is to chose encoding where the Hamming distance between adjacent (successive) values is greater.

e.g. 000 ⟶ 0
111 ⟶ 1
3

010 ⟶ ϕ

Hamming (7,4)

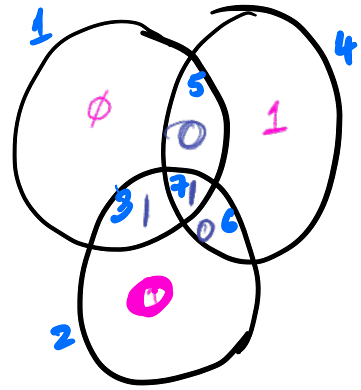
A linear correcting code that encodes 4-bits of data into 7-bits by adding 3 parity bits.

(i) It can correct upto 1 bit errors
(ii) It can detect upto 2 bit errors

Layout: p_1 p_2 d_1 p_3 d_2 d_3 d_4

positions ⟶ 1 2 3 4 5 6 7

$p_1 \longrightarrow 1, 3, 5, 7$
$p_2 \longrightarrow 2, 3, 6, 7$
$p_3 \longrightarrow 4, 5, 6, 7$



(3) (5) (6) (7)

d_1 d_2 d_3 d_4

1 0 0 1

0 0 1 1 0 0 1

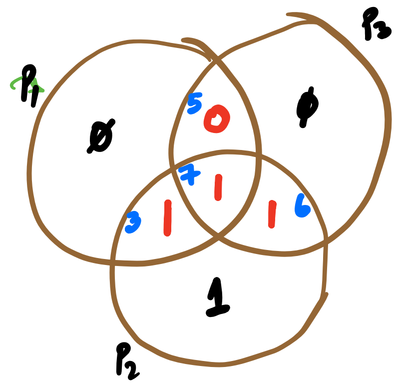
Example: Say you want to transmit

(3) (5) (6) (7)

1 0 1 1

0 1 1 0 0 1 1 data

TRANSMITTED



0 1 1 0 0 1 1

1 2 3 4 5 6 7

0 1 1 0 0 1 0

P_1 : X

P_2 : X

P_3 : X

7

P_3 P_2 P_1

$1 \; 1 \; 1_2 \longrightarrow 7_{10}$

Exercise: 0 1 1 0 1 1 1