# INFO III

## Représentation des nombres

### introduction à l'architectures des ordinateurs

1 - Représentations des entiers & applications
2 - Représentation des réels
3 - Conséquences & limitations

# Résolution de problème

01010100011011110111010001101111

Qui/que suis-je ???

C'est la question de la représentation :

Que représente une séquence binaire donnée ?

# 1 - Principe de la représentation des nombres entiers en mémoire

Integer

Un nombre entier représente une quantité donnée de manière abstraite :

Cinq pommes,  trois pierres, etc...

La question de l'écriture d'un nombre correspond au choix d'une représentation symbolique de ces quantités.

Le système décimal ne s'impose pas en soit mais vient du fait que nous avons dix doigts ce qui était (et reste) très pratique.

Nous allons voir que les opérations sont en fait bien plus simples et rapides en binaire pourvu que nous ayons de la mémoire : c'est le cas des ordinateurs.

# Notion de base de représentation :

L'écriture d'un nombre en **base décimale** repose sur une décomposition de la quantité en une somme de facteur de puissance de 10. Concrètement, si N = 12835, la séquence de chiffres traduit la décomposition suivante : $\boxed{N = 1.10^4 + 2.10^3 + 8.10^2 + 3.10^1 + 5.10^0}$

Ce choix nous semble très naturel car c'est notre usage, depuis l'enfance.

Cette écriture décimale nécessite dix chiffres : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Il y a donc très vite un problème : il n'existe pas de chiffre pour écrire dix notons le A.
de même pour écrire onze (B) puis douze (C).... vingt et un (L)

$$A = 1.A + 0 \longrightarrow A = 10_{10}$$
$$B = 1.A + 1 \longrightarrow B = 11_{10}$$
...
$$L = 2.A + 1 \longrightarrow L = 21_{10}$$

Cette écriture nous paraît très naturelle mais n'a rien d'universelle. (ex : chiffres romains)
[ On compte aussi en base 60 : 1° = 60' et 1'= 60" pour les angles ]

Prenons un exemple plus simple : N = 45

$$N = 4.10^1 + 5.10^0 \xrightarrow[\boxed{b = 10}]{\text{écriture décimale}} N = 45_{10}$$

Mais je peux aussi n'utiliser que 8 chiffres [0, 1, 2, 3, 4, 5, 6, 7] :

$$N = 5.8^1 + 5.8^0 \xrightarrow[\boxed{b = 8}]{\text{écriture base 8}} N = 55_8$$

$$(N = 40 + 5)$$

je peux aussi n'utiliser que 5 chiffres [0, 1, 2, 3, 4] :

$$N = 1.5^2 + 4.5^1 + 0.5^0 \xrightarrow[\boxed{b = 5}]{\text{écriture base 5}} N = 140_5$$

$$(N = 25 + 20 + 0)$$

**<u>Exercice :</u>** **écrire N = 45 en base 3 et 4**

Soit b la base :

$$N = a_n b^n + \cdots + a_2 b^2 + a_1 b^1 + a_0 b^0 \qquad a_n \neq 0 \xrightarrow{\text{écriture base b}} N = \left[ a_n \cdots a_2 a_1 a_0 \right]_b$$

Soit : $$N = a_0 b^0 + a_1 b^1 + a_2 b^2 + \cdots + a_n b^n \qquad a_n \neq 0$$

On peut donner une écriture factorisée à la manière de Horner :

$$N = a_0 + b \times \left[ a_1 + b \times \left[ a_2 + b \times \left[ \cdots + b \times \left[ a_n \right] \cdots \right] \right] \right]$$

**Exercice :**

En reprennant la démarche du TP d'informatique sur la conversion binaire, écrire un programme général pour obtenir l'écriture dans une base b quelconque à partir de la valeur décimale d'un nombre.

# a - Ecriture en binaire $b = 2$

Cette base nous intéresse particulièrement parce qu'elle n'utilise que deux chiffres : [0,1]

ce qui peut être réalisé | mécaniquement par des systèmes simples : [ouvert / fermé]
| électriquement

$$N = 1.2^5 + 0.2^4 + 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0 \qquad N = 101101_2$$

$$(N = 32 + 8 + 4 + 1)$$

Il faut donc connaître par coeur les puissances de 2 successives en écriture décimale :

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |

un bit — un octet

| 11 | 12 | 13 | 14 | 15 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|
| 2048 | 4096 | 8192 | 16384 | 32768 | 65536 | 4294967296 | 18446744073709551615 |

deux octets — quatre octets — huit octets

**<u>Exercice :</u> écrire en binaire les nombres suivants : 3, 5, 7, 13, 15, 27, 31, 47, 99**

# Méthode générale de décomposition d'un décimal en binaire

**Exercice :** écrire en binaire les nombres suivants : 728, 1937, 22222, 314159 !

**La réciproque est immédiate :**

**Exercice :** écrire en décimal les nombres suivants : 10101, 01001001, 110110110

# Tables d'opérations

Boole a montré que dans une algèbre binaire toutes les opérations peuvent se décomposer en combinaisons de deux opérations élémentaires : addition et multiplication ( + négation)

Addition en binaire

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 10 |

Propagation de retenue

Multiplication binaire

| x | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**Exercices :** **Poser les calculs suivant :**

```
  00101101      101       111        1011        111
+ 01011011    x 11    x 101     x 1011     x 1111
```

```
  11011101
+ 11010100
```

# Conclusion :

There are only 10 types of people in the world:

those who understand binary ...

.... and those who don't !

## b - Représentation octale

$b = 8$

**Exercices :** **convertir les nombres suivants d'une base vers l'autre :**

$101111001_2 =$ $101011001_2 =$ $632_8 =$

$100101011_2 =$ $421_8 =$ $753_8 =$

# c - Représentation hexadécimal $b = 16$

Cette base «condensée» est utilisée entre autres pour écrire les adresses mémoires.
Le point délicat est qu'il faut nécessairement16 chiffres => on ajoute les 6 lettres ABCDEF aux 10 chiffres de l'écriture décimale soit [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F]

Ex : F = 4A = FF =

**Exercices :** CAFE =

**addition :**

2A
+ BC

**Ecrire en hexadecimal :**

$157_{10}$ =

$212_{10}$ =

$321_{10}$ =

**Retrouver les adresses IP :**

C0:A8:1:1

92:A7:4B:C2

**Convertir les nombres suivants d'une base vers l'autre :**

$11011001_2$ = $11100101_2$ = $5B_{16}$ =

$01101111_2$ = $11_{16}$ = $DA_{16}$ =

# Application : Table de caractères ASCII

**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

```
>>> for i in range(128):
...     if (i%32 != 0):
...         print(chr(i), end="")
...     else:
...         print(chr(i))
```

**Caractères de contrôle**

0 à 31 et 127 :
[echap, del, return, tab, beep etc ....]

! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?

syntaxe Python :

ord('A')
chr(65)

@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _

‘ a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

- On peut donc écrire tous les caractères sur *7* bits [0-127]

- Les caractères sont en fait stockés sur un octet : 8 bits

0 - 127 : caractères de base [ci-dessus]
128 - 256 : ascii étendu, charactères avec les accents ISO-Latin-1

# Ascii en hexadécimal

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | | | | | | | | | | | | | | | | |
| 0010 | | | | | | | | | | | | | | | | |
| 0020 | | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 0030 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 0040 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 0050 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 0060 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 0070 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |

**<u>Exercice :</u>**

**Donner le numéro des charactères @ et & en binaire, octale, décimale et en hexadécimale**

**Quel est le caractère décimal : 101 octal : 132 binaire : 1010101**

# Image pixelisée :

**Couleur RGB : (r, g, b)** 256 niveaux de Rouge, Vert et Bleu

$0 < r < 255$

$0 < g < 255$ D'où $256^3 = 16.7$ millions de couleurs

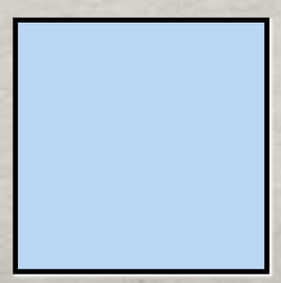
$0 < b < 255$

**On peut aussi l'écrire sous forme décimale :** $Q = 256^2 r + 256v + b$

**Ou sous forme hexadécimale : Q = AB:CD:EF**

Trouver la couleur : Q = AB:CD:EF

- en RGB
- en décimal



# d - Représentation des nombres entiers en machine

Dans l'ordinateur les entiers sont donc écrits comme une suite de 0 et de 1, correspondant à des états électriques de la mémoire [sous tension ou pas], ou encore des états magnétiques [aimantés ou pas] des disques durs, ou des bandes magnétiques.

Mais quelle que soit la nature du support la place disponible pour écrire ces données est limitée, et la taille des nombres est donc elle aussi limitée.

En informatique le type «integer» désigne les entiers relatifs. Ceux-ci peuvent être représentés de différentes manières. On distinguera, selon la nomenclature du langage C :

| | | |
|---|---|---|
| Entiers «short» | : 2 octets soit 16 bits | 65.536 valeurs successives |
| Entiers «long» | : 4 octets soit 32 bits | 4.294.967.296 valeurs successives |

Attention : si le résultat d'une opération arithmétique dépasse la taille disponible,
=> l'unité de calcul « lève » une erreur d'OVERFLOW et le programme s'arrête.

# Comment représenter le signe des nombres ?

**Il existe plusieurs nomenclatures :**

-> **Utiliser le premier bit pour le signe ; les suivants pour le nombre en valeur absolue**
- Pb il y a deux zéros (ex : en 3 bits 000 et 100)
- Poser le calcul : 1 + -(1) = -2 => il faut une architecture spécifique pour la soustraction

**-> Méthode du complément à 2** [la plus utilisée]

1 - Nombres positifs : les nombres positifs ont leur 1er bit à 0
et sont indexés dans l'ordre binaire naturel (sur n-1 bits).

2 - Nombres négatifs : On remplace pour tous les bits : 0 par 1 et 1 par 0
puis on ajoute 1. [ n -> -n ]

**Rq :** Il reste un nombre qui n'est pas engendré par cette méthode -(Nmax+1)
En effet il est sont propre complément à 2 (tout comme 000)

ex : 3 bits

| | |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | -4 |
| 101 | -3 |
| 110 | -2 |
| 111 | -1 |

**Avantage :** la soustraction devient une addition de l'opposé.
=> on ne fait que des additions

**<u>Exercice :</u>** **Compléter la table des nombres entiers, par la méthode du complément à 2**

**ex : codage 4 bits**

| 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |

**<u>Exercices :</u>**

Poser en binaire les opérations suivantes avec les nombres du tableau :

2 + 5 = ?  3 + 7 = ?  1 - 3 = ?  3 - 5 = ?

2 x 3 = ?  (-2) x 4 = ?  (-3)x(-3) = ?  (-2)x(-1) = ?

Attention à ne pas sortir du domaine de définition des nombres

<u>Conclusion :</u>

Dans cette représentation, l'arithmétique binaire est élémentaire.

**-> Méthode de la représentation biaisée** [donnera l'exposant entier pour la représentation des réels]

On retranche à l'écriture binaire sur n bits, $2^{n-1}-1$

Ex : 8 bits  0 < binaire < 255  on retranche 127  soit  -127 < n < +128

**Exercice : Compléter la table des nombres entiers, par la représentation biaisée**

**ex : codage 4 bits**

| 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |

**Avantage pour comparer les valeurs :**

- Le premier chiffre donne le signe [0 : **-** et 1 : **+** => contraire de la méthode du complément à 2]
- Les valeurs sont rangées dans l'ordre croissant des écritures binaires, ce qui n'était pas le cas de la méthode du complément à 2

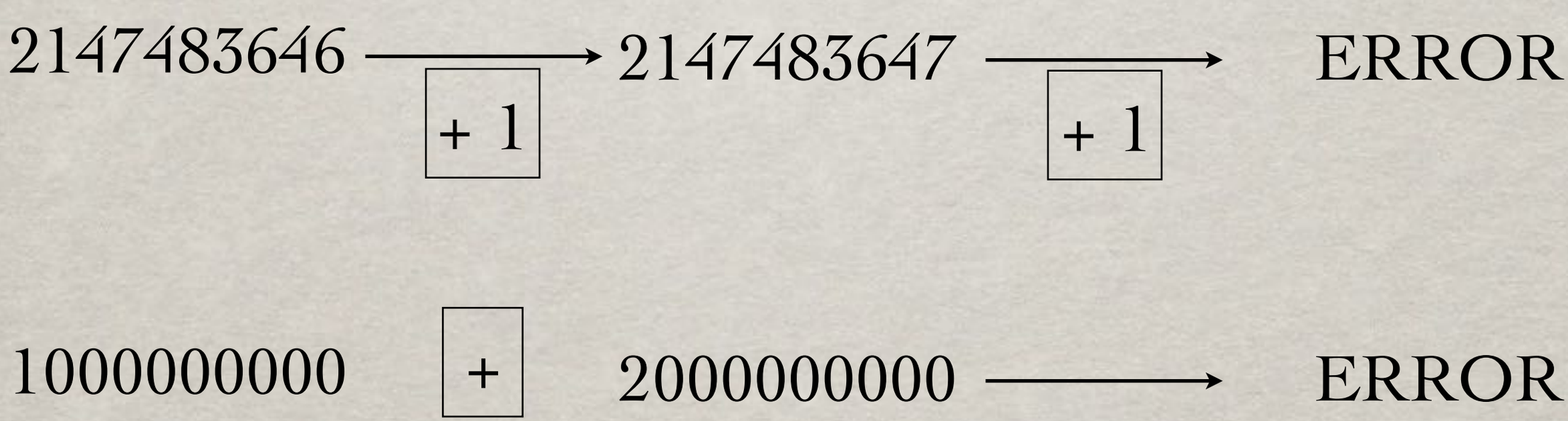
# e - Les limites des machines

A noter sur la copie double :
Représentation des nombres

Nous avons vu que les entiers sont représentés sur un nombre déterminé de bits (ou d'octets).
Toute opération donnant une valeur supérieure au Max (ou inférieure au Min) se traduira au niveau de l'unité arithmétique et logique par une levée d'erreur OVERFLOW.

Toutefois celle-ci n'est pas toujours signalée par le programme/ calculatrice / autre.
Le programme garde alors une valeur erronée du fait de l'arrondie.

**Ex calculatrice 10 chiffres SY-10 :**




**Exercice :**

Quels sont les nombres (Min-Max) encodés en 16, 32 puis 64 bits ?

Réponses :

Rq : On considère ici des entiers signés

# 2 - Principe de la représentation des nombres réels en mémoire

Float

**Tous les nombres réels ne peuvent pas s'écrire simplement, avec une écriture exacte comme :**

$$3.0000000 \text{ etc} = 3 \qquad \frac{1}{2} = 0.5000000 = 0.5$$

- Certains nombres rationnels dits périodiques nécessitent une infinité prévisible de chiffres :

$$\frac{1}{3} = 0.33333333 \text{ etc} \qquad \frac{1}{7} = 0.\underline{142857}142857 \text{ etc}$$

- On ne peut pas prévoir la succession de chiffres d'un nombre irrationnnel

$$\pi = 3.14159265358979... \text{ etc}$$

=> Dans tous les cas, limiter le nombre de décimales à 12, 16, ou plus induit une erreur d'arrondi inévitable ! **La représentation des réels en machine n'est pas exacte.**

# Notion de virgule flottante :

$x = 4.321$ peut aussi s'écrire $x = 43{,}21.10^{-1} = 000{,}4321.10^{1}$

La compatibilité entre différents systèmes impose une norme pour l'écriture décimale :

$$x = 4{,}321 \cdot 10^{0}$$

1,000 ≤ mantisse décimale < 10,000

Avantage virgule fixe :
- L'addition est la même que pour des entiers => plus rapide

Avantage en virgule flottante :
- pas besoin de savoir à l'avance, le nombre de chiffres à gauche ou à droite de la virgule
  => permet de représenter beaucoup plus de valeurs différentes.
  => permet de travailler avec des quantités d'ODG très différents

Pb : on ne peut ajouter que des nombres ayant le même exposant
=> Il est nécessaire de se ramener au même exposant, pour pouvoir ajouter deux réels

# Ecriture binaire des fractions :

Il suffit de multiplier chaque chiffre après la virgule par des puissances négatives de la base :

$$x = [a_0, a_{-1}a_{-2}a_{-3}]_2 = a_0 + a_{-1}2^{-1} + a_{-2}2^{-2} + a_{-3}2^{-3}$$

$$x = 0.101_2$$

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

$2^{-4} = 0.0625$

*etc…*

L'opération inverse est plus délicate, car l'écriture n'est pas nécessairement finie !!!

$$x = 0.1_{10}$$

**Exercices :** Convertir d'une base à l'autre (2 ou 10) les nombres suivants

$x = 0.011_2$ $\quad$ $x = 0.1111111111_2$ $\quad$ $x = 0.625_{10}$

$x = 1.01_2$ $\quad$ $x = 0.5_{10}$ $\quad$ $x = 0.3333_{10}$

Taper « ieee754 converter » sous gogole

IEEE 754 Converter (JavaScript), v0.13

Note: This JavaScript-based version is still under development, please report errors here.

| | Sign | Exponent | Mantissa |
|---|---|---|---|
| Value: | +1 | $2^{-4}$ | 1.600000023841858 |
| Encoded as: | 0 | 123 | 5033165 |
| Binary: | ☐ | ☐☑☑☑☑☐☑☑ | ☑☐☐☑☑☐☐☑☑☐☐☑☑☐☐☑☑☐☐☑☑☐☑ |

| | |
|---|---|
| Decimal Representation | 0.1 |
| Binary Representation | 00111101110011001100110011001101 |
| Hexadecimal Representation | 0x3dcccccd |
| After casting to double precision | 0.10000000149011612 |

# Réponses :

IEEE 754 Converter (JavaScript), V0.12

Note: This JavaScript-based version is still under development, please report errors here.

| | Sign | Exponent | Mantissa |
|---|---|---|---|
| Value: | +1 | $2^{-2}$ | 1.3331999778747559 |
| Encoded as: | 0 | 125 | 2795084 |
| Binary: | ☐ | ☐☑☑☑☑☑☐☑ | ☐☑☐☑☐☑☐☑☐☑☐☐☑☑☐☐☑☐☐☑☑☐☐ |

| | |
|---|---|
| Decimal Representation | 0.3333 |
| Binary Representation | 00111110101010101010011001001100 |
| Hexadecimal Representation | 0x3eaaa64c |
| After casting to double precision | 0.333299994468688 96 |

non représentable en 32 bits

IEEE 754 Converter (JavaScript), V0.13

Note: This JavaScript-based version is still under development, please report errors here.

| | Sign | Exponent | Mantissa |
|---|---|---|---|
| Value: | +1 | $2^{3}$ | 1.953125 |
| Encoded as: | 0 | 130 | 7995392 |
| Binary: | ☐ | ☑☐☐☐☐☐☑☐ | ☑☑☑☑☐☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ |

Decimal Representation: 15.625

Binary Representation: 01000001011110100000000000000000

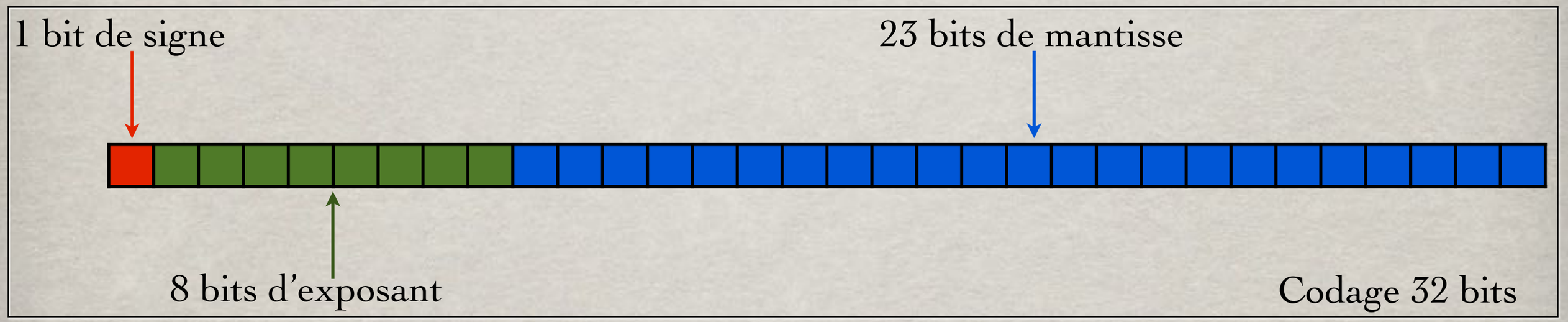
Hexadecimal Representation: 0x417a0000

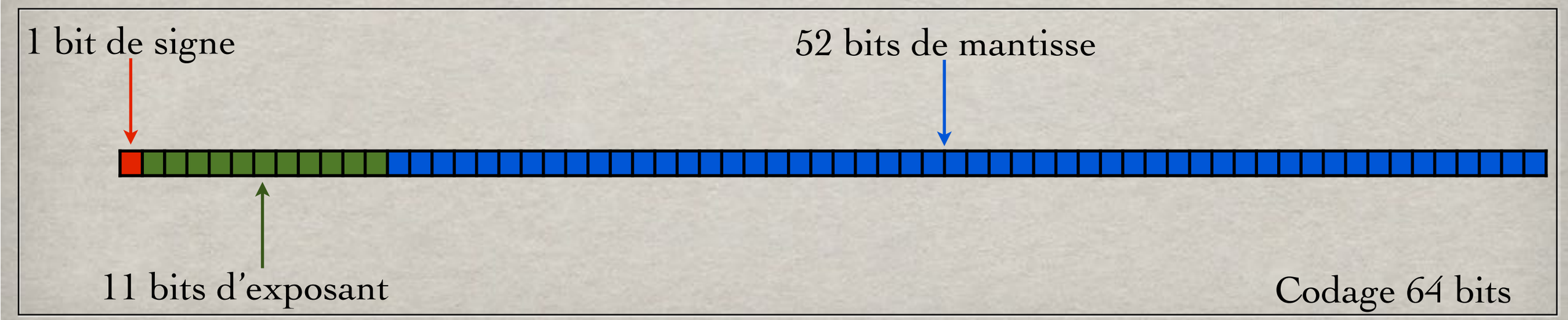
After casting to double precision: 15.625

# Représentation numérique des réels :

La norme IEEE 754 !!!

Les nombres flottants sont écrits sous la forme : $x = s \cdot m \cdot 2^{e}$



**Signe :** 1 bit 0 : + et 1 : -

**Exposant :** C'est un entier !

Celui-ci est stocké par la méthode biaisée.
[et non en complément à 2 car les comparaisons sont alors plus complexes]

| | | |
|---|---|---|
| 8 bits | $-126 < e < +127$ | $e_{Max} = 2^{n-1} - 1$ |
| 11 bits | $-1022 < e < +1023$ | $e_{Min} = 1 - e_{Max}$ |

Valeurs d'exceptions : 32-bits : -127 et +128 64-bits : -1023 et +1024 sont écartées

**Mantisse :** **Celle-ci est écrite en virgule flottante binaire normalisée :**

On peut toujours écrire le nombre flottant 0,00101... sous la forme 1,01... x 2^n
Autrement dit le premier chiffre **avant** la virgule est un 1. [Car il n'y a que deux chiffres en binaire !]
et on modifie la valeur de l'exposant.

**=> l'écriture est alors dite normalisée**

Erratum

23 bits $m-1 = 0.000\ 000\ 000\ 000\ 000\ 000\ 000\ 01_2 = 2^{-23} = 1.19...10^{-7}$

52 bits $m-1 = 0.00$ ..47 zéros... $001_2 = 2^{-52} = 2.2...10^{-16}$

$$1.0000000_{10} < m <\sim 1.999999999_{10}$$

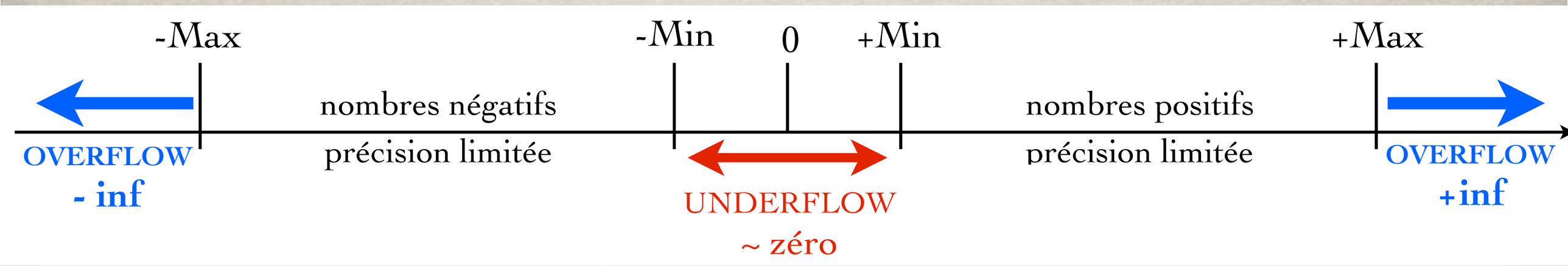
**Valeur du FLOAT :**

$$Décalage = 2^{nb\_bits-1} - 1$$

$$v = Signe \cdot Mantisse \cdot 2^{Exposant - Décalage}$$

# L'échelle des nombres flottants :

0 | 00...00 | 00..00




| | Min | Max |
|---|---|---|
| 32 bits<br>[simple précision] | $+2^{-22}2^{-127} = 1.40E-45$ | $+(1.999...)2^{+127} = 3.40E+38$ |
| 64 bits<br>[double précision] | $+2^{-51}2^{-1023} = 4.94e-324$ | $+(1.999...)2^{+1023} = 1.798E+308$ |

(dénormalisée)

**Exercices :** Calculer l'espace entre les barreaux aux deux extrémités positives de l'échelle

# Les «exceptions» :

La valeur supérieure de l'exposant +128 [ou +1024] est mise de coté pour les infinis :

$+\infty \rightarrow$ 0 11111111 00000000000000000000000

$-\infty \rightarrow$ 1 11111111 00000000000000000000000

*NaN* $\rightarrow$ 1 11111111 0000000...1...00000000

La valeur inférieure de l'exposant -127 [ou -1023] est mise de coté pour les zéros :

$0^{+} \rightarrow$ 0 00000000 00000000000000000000000

$0^{-} \rightarrow$ 1 00000000 00000000000000000000000

Si l'exposant vaut -127 [ou -1023] **la mantisse n'est pas normalisée** c-à-d qu'on l'écrit sous la forme **0.0010**... et non 1 .00101... Cela permet de descendre vers des nombres plus petits.

**1.0110….**

**0 < m < 2 codée sur 23 bits (resp. 52)**

# 3 - Conséquences de la limitation de la représentations des nombres

# Exemples tristement célèbres d'erreurs d'arrondis

## 1982 : Bourse de Vancouver

On créé une cotation tronquée à 3 chiffres au lieu de l'arrondir
=> 2 ans plus tard : indice = 524.881 [estimé en réalité à 1098.811 sans l'arrondi....]

## 1991 : Erreur de trajectoire d'un missile SCUD

pas de temps de 0.1 décimal --> rationnel périodique en binaire
=> tronquée à chaque pas de temps : horloge défectueuse !
=> trajectoire incorrecte.

Conclusion : 28 morts et des centaines de blessés

## 1996 : fusée Ariane V

Utilisation logiciel Ariane IV avec Ariane V : incompatibilité logiciel induit la conversion d'un FLOAT 64 bit en entier de 16bits ===> OVERFLOW dû à une accélération beaucoup plus forte qu'Ariane IV.

Conclusion : fusée et satellites détruits --> 500 millions $

# Plus simplement :

Passage deux chiffres à 3 chiffres

Une patiente de 106 ans enregistre son dossier à l'accueil de l'hôpital,
on l'envoie en pédiatrie car son âge est codé sur une chaîne de 2 caractères ....

Trouver les limites de base de votre calculatrice :

$$1.000000003 \xrightarrow{\div\ 2} 0.500000001 \xrightarrow{\times 2} 1.000000002$$

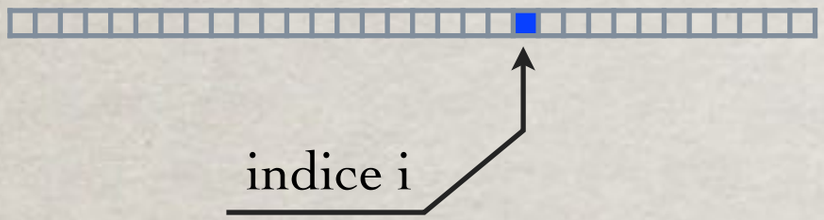
Pb. d'affichage => l'erreur n'est même pas signalée

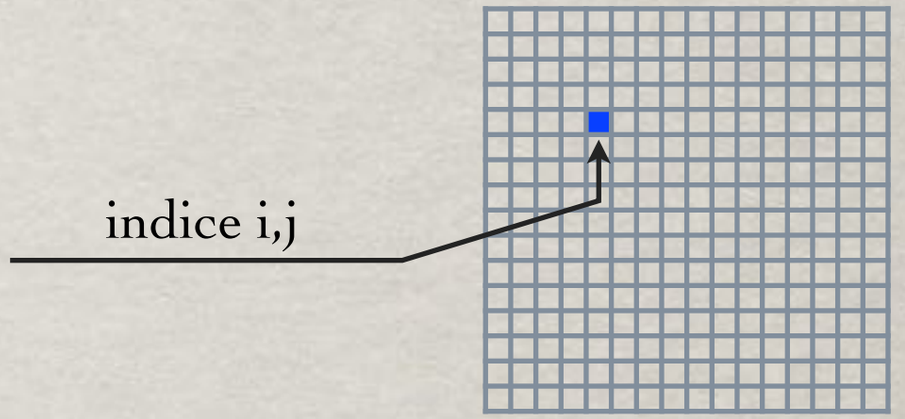
Exemples de calcul faux (cf -TP)

# Comment décrire un problème physique dans un ordinateur ?

→ il faut passer du continu au discret : grille

1D : Tableau




2D

La position x est remplacée par un indice de position i : $x_i = i\Delta x$

Soit f(x) une fonction : $f_i = f(x_i) = f(i\Delta x)$

Définir une «méthode» pour la dérivation :

→ Difficile : C'est tout un art !

Ex :

$$\left.\frac{df}{dx}\right|_i \simeq \frac{f_{i+1} - f_i}{\Delta x}$$

# Exemple 2D : propagation non dispersive de la chaleur dans une plaque.

(Source : ENS -Paris)

$$\frac{\partial^2 u}{\partial t^2} = c^2 \, \Delta u \, .$$

de la même manière, on peut écrire un schéma explicite en temps

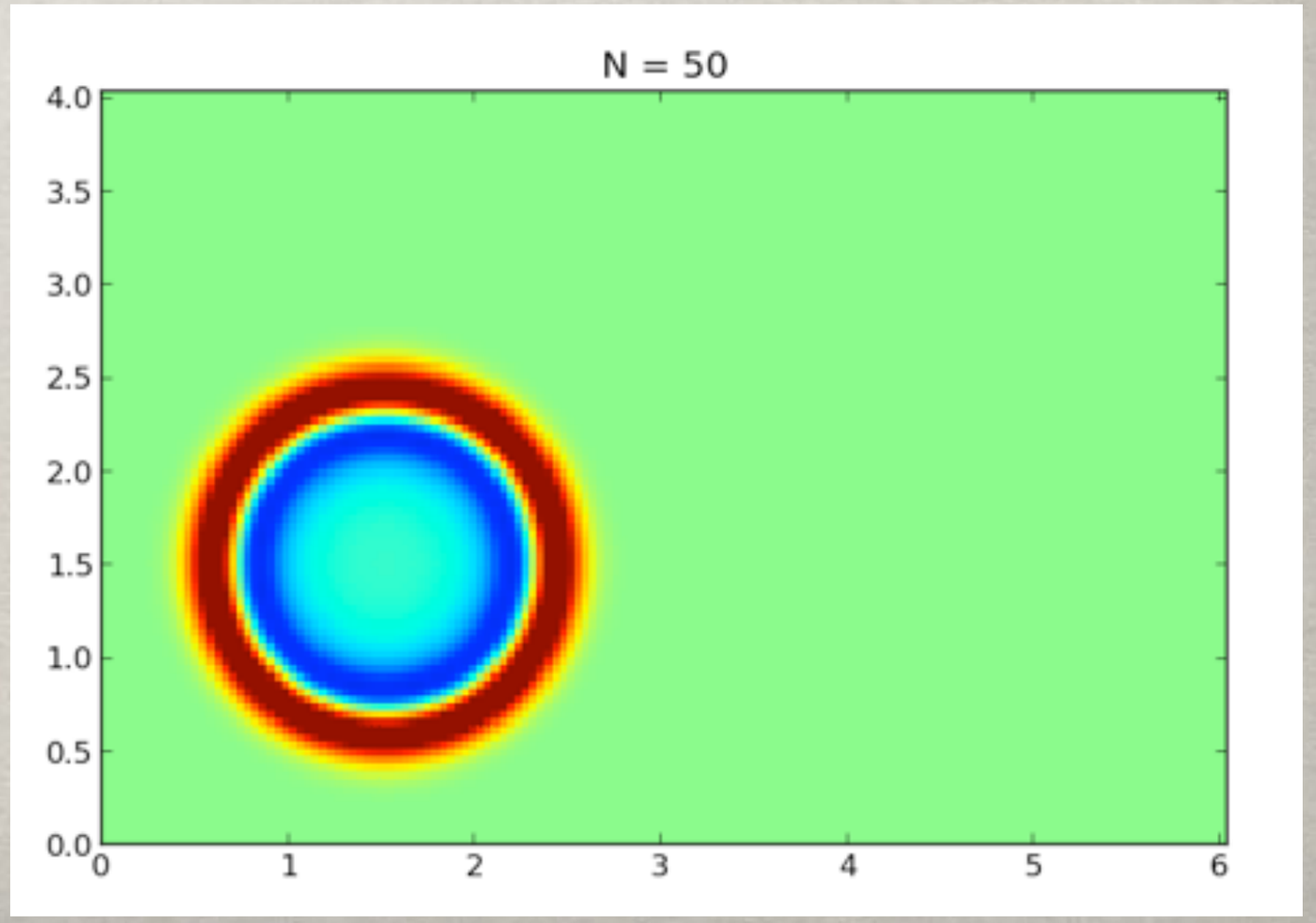
$$u_{i,j}^{n+1} = 2\,u_{i,j}^{n} - u_{i,j}^{n-1} + \Delta t^2\, c^2 \, \left[ (u_{i-1,j}^{n} - 2\,u_{i,j}^{n} + u_{i+1,j}^{n})/\Delta x^2 + (u_{i,j-1}^{n} - 2\,u_{i,j}^{n} + u_{i,j+1}^{n})/\Delta y^2 \right] \, .$$

On pourra considérer une condition initiale $u|_0$ sous la forme d'une gaussienne et $\partial_t u|_0 = 0$.

temps $n$ $\quad t_n = n\Delta t$

position $i, j$ $\quad x_i = i\Delta x$, $\; y_j = j\Delta x$

$$u(t,x,y) \rightarrow u_{i,j}^{n}$$



Rq :
---

Les premiers super-calculateurs ont été développés par des gens comme Von Neumann, pour décrire la propagation/diffusion de la chaleur dans un gaz
--> Bombe A

# Résolution de problème

01010100011011111011101000110111

Qui/que suis-je ???

C'est la question de la représentation :

Que représente une séquence binaire donnée ?

CONCLUSION :

Sans une norme, une nomenclature ou un protocole, une séquence binaire peut signifier absolument n’importe quoi !!!

# Résolution de problème

01010100.01101111.01110100.01101111