REPRESENTATION DES NOMBRES ENTIERS EN INFORMATIQUE

I. COMMENT ÉCRIT-ON LES NOMBRES ?

Historiquement, de nombreux systèmes pour écrire les nombres (on parle de ***système de numération***) ont été inventés ; il en existe de deux types principaux :

1. Les systèmes additifs :

→ le nombre écrit est égal à la *somme de la valeur de chacun des chiffres* utilisés pour écrire ce nombre.

Les chiffres romains :

I = 1
V = 5
X = 10
C = 100
D = 500
M = 1 000

Exemple : la numération romaine était additive.

→ que vaut dans notre système le nombre romain : MMXIX = **1000 + 1000 + 10 + 9 = 2019**

→ comment s'écrit en chiffres romains le nombre 1515 = **MDXV**

Inconvénient : essayez de faire l'addition *en chiffres romains* : MMXIX + CVII = **sans convertir, c'est impossible !!!**

→ *très difficile de faire des calculs avec ce type de numération !!*

2. Les systèmes positionnels : historiquement, une révolution, qui a permis de simplifier considérablement la manière de faire des calculs.

II. PRINCIPE DE LA NUMÉRATION POSITIONNELLE DE BASE *N* :

1. Principe :

- la numération s'appuie sur une certaine ***base*** de calcul → pour nous, c'est la base 10
- on dispose, pour écrire un nombre, d'autant de ***chiffres*** que la valeur de la base → en base 10 : 0,1,2,.....,8,9 (soit 10 chiffres)
- chaque chiffre du nombre correspond, *selon sa position depuis la droite dans le nombre*, à une ***puissance croissante de la base*** → en base 10, c'est donc, depuis la droite : $10^0 = 1$, puis $10^1 = 10$, puis $10^2 = 100$, etc....
- le nombre écrit est alors la ***somme des produits de chacun des chiffres par la puissance correspondante***.

Exemple : 5 238 = 5 x 1000 + 2 x 100 + 3 x 10 x 8 x 1 = 5 x 10^3 + 2 x 10^2 + 3 x 10^1 + 8 x 10^0

5	2	3	8
10^3	10^2	10^1	10^0

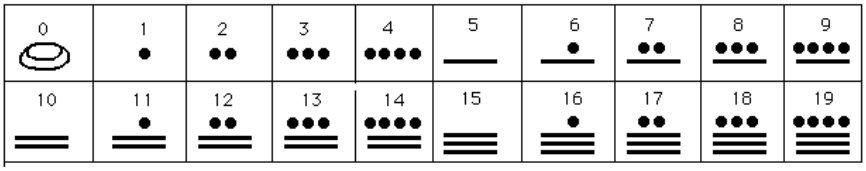
2. Exercice : décomposer les nombres suivants en base 10 :

- 65 536 = **6 x 10^4 + 5 x 10^3 + 5 x 10^2 + 3 x 10^1 + 6 x 10^0**
- 2019 = **2 x 10^3 + 0 x 10^2 + 1 x 10^1 + 9 x 10^0**

3. Choix de la base :

Actuellement, la base 10 est la plus utilisée, mais de nombreuses civilisations ont utilisé une numération positionnelle dans d'autres bases. Pouvez-vous ainsi déterminer la valeur des nombres suivants ?

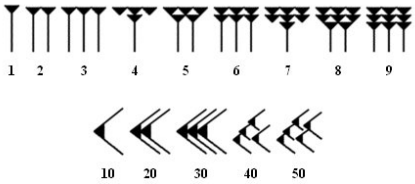
- Les Mayas *(peuple qui habitait l'Amérique centrale et le Mexique actuel)* ont utilisé un système positionnel de **base 20**. De plus, les nombres étaient écrits du haut vers le bas.
  → quelle est en base 10 la valeur du nombre écrit ci-dessous ?



5
0
12

→ **5 x 20^2 + 0 x 20^1 + 12 x 20^0 = 5 x 400 + 0 + 12 x 1 = 2 012**

- Les Babyloniens (peuple de la Mésopotamie antique) utilisaient la **base 60** (*bizarre...mais cela est cependant parvenu jusqu'à nous dans notre manière d'exprimer les minutes et les secondes ou alors les angles*). Ils n'utilisaient que deux symboles : "un clou" vertical 𒁹 représentant l'unité et un "chevron" 𒌋 associé au nombre 10 . Les « chiffres » 1 à 59 sont représentés d'une manière additive en répétant chacun de ces deux symboles : 𒌋𒐗 est égal à 19 (1 chevron + 9 clous).



Ils plaçaient ensuite un espace entre chaque « chiffre » correspondant aux puissances croissantes de la base.
→ quelle est en base 10 la valeur du nombre écrit ci-dessous ?

→ **$9 \times 60^2 + 5 \times 60^1 + 28 \times 60^0 = 9 \times 3\,600 + 5 \times 60 + 28 \times 1 = 32\,728$**

III. ET LE RAPPORT AVEC L'INFORMATIQUE ??.....

On y vient...En informatique, c'est beaucoup plus sommaire ; les ordinateurs ne « comprennent » en effet que *deux* informations :

- le courant passe
- le courant ne passe pas

→ *tout* dans un ordinateur doit donc être codé avec *deux* symboles correspondant à chacune de ces informations ; la base utilisée pour représenter les nombres dans un ordinateur est donc la **base 2**. On parle alors de **numération binaire**.

1. Principe : c'est une numération de position, donc qui suit exactement ce qui a été présenté au paragraphe II.1. :

- la numération s'appuie sur la base **2**
- on dispose, pour écrire un nombre, de **2** chiffres (notés par convention **0** et **1** par ordre croissant)
- chaque chiffre du nombre correspond, *selon sa position depuis la droite dans le nombre*, à une *puissance croissante de la base* , soit ici :
  $2^0 = 1$, puis $2^1 = 2$, puis $2^2 = 4$, etc .....
- le nombre écrit est alors la ***somme des produits de chacun des chiffres par la puissance de 2 correspondante***.

(Remarque : pour éviter toute ambiguïté, on écrira dorénavant en indice à la fin du nombre, la base dans laquelle il est exprimé.)

Exemple : $\mathbf{101101}_2 = \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 = 32 + 0 + 8 + 4 + 0 + 1 = 45_{10}$

2. Quelques définitions à connaître :

- un **bit** (contraction de *binary digit* = chiffre binaire) : un chiffre de la numération binaire (donc un 0 ou un 1...)
- un **octet** (byte en anglais) : un ensemble de 8 bits; c'est donc un "nombre" binaire formé de 8 "chiffres".
  Ce regroupement des bits par 8 est une habitude héritée de l'architecture des premiers microprocesseurs grand public, qui traitaient l'information par groupes de 8 bits (les processeur actuels le font par groupes de 64 bits...)
- le **LSB** (= Least Significant Bit : bit de **poids le plus faible**) : bit situé le plus *à droite* dans un nombre binaire.
- le **MSB** (= Most Significant Bit : bit de **poids le plus fort**) : bit situé le plus *à gauche* dans un nombre binaire

Remarque importante : dans toute base, rajouter des 0 devant le nombre ne change pas ce nombre.

- *En base 10, par exemple, $001256_{10} = 1256_{10}$ → les 0 de devant ne sont pas significatifs ; ceux de derrière par contre le sont !!*

C'est valable également en base 2 : $001101_2 = 1101_2$

3. Exercices :

1. Convertir les nombres binaires suivants en base 10 :

- 10111010_2 = **$1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 186_{10}$**
- 1111111_2 = **$1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 127_{10}$**

2. Quel est le nombre binaire le plus grand que l'on peut coder avec 2 bits ? 4 bits ? 8 bits ? Quel est sa valeur en base 10 ?
Combien de valeurs différentes peut-on coder avec 2 bits ? 4 bits ? 8 bits ?
Généraliser les deux résultats précédents à un nombre codé sur un nombre quelconque *N* de bits.

	2 bits	**4 bits**	**8 bits**	**N bits**
Plus grand nombre	**$11_2 = 3_{10}$**	**$1111_2 = 15_{10}$**	**$11111111_2 = 255_{10}$**	**$2^N - 1$**
N^{bre} de valeurs différentes	**$4 = 2^2$**	**$16 = 2^4$**	**$256 = 2^8$**	**2^N**

3. Comment savoir simplement si un nombre binaire donné correspond à un nombre en base 10 *pair* ? *impair* ?

Selon la valeur du LSB (= bit le plus à droite) : LSB = 0 → pair LSB = 1 → impair

4. Pour multiplier par dix un entier naturel exprimé en base dix, il suffit d'ajouter un 0 à sa droite ; par exemple : 12 × 10 = 12**0**.
Réalisée en base deux, quel est logiquement le résultat de la même opération (on dit qu'on *décale tous les bits vers la gauche*) ?
Montrer que « ça marche » en convertissant le nombre 24_{10} puis en décalant ses bits...

Logiquement, on devrait *multiplier par 2* le nombre.
$24_{10} = 11000_2$ → si on rajoute un 0 à droite : $110000_2 = 48_{10}$ OK !

5. **Addition binaire** : vous vous souvenez comment vous faisiez des additions à l'école ? Et bien ça marche aussi dans n'importe quelle base bien sûr...Par exemple, quel est le résultat des additions suivantes (*attention à la retenue !*) :

```
  1 1
  1 0 1 0 1 1 2
+   1 1 1 0 0 2
  ---------------
  1 0 0 0 1 1 1
```

```
  1 1 1 1 1 1 1
  1 1 1 0 1 0 1 1 2
+ 1 1 1 1 1 1 0 1 2
  -------------------
  1 1 1 1 0 1 0 0 0
```

6. **De la base 10 à la base 2** : dans l'autre sens c'est facile, mais là c'est un peu plus délicat.

Une des possibilités : on compare le nombre à convertir aux puissances *décroissantes* de 2 (128, 64, 32,...,1 sur 8 bits).

- si le nombre est supérieur ou égal à cette puissance : on note un bit à '1', on retranche la puissance de 2 au nombre et on recommence avec le résultat
- si le nombre est inférieur : on note un bit à '0', et on recommence avec la puissance de 2 inférieure.

On s'arrête quand le nombre est devenu égal à 0; la succession des '0' et '1' donne, dans l'ordre, le nombre converti en base 2.
Exemple : convertir 156_{10} en base 2 :

156	156 – 128 = 28	28	28	28 – 16 = 12	12 – 8 = 4	4 – 4 = 0	0
$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	0	0	1	1	1	0	0

→ $156_{10} = 10011100_2$

A l'aide de cette méthode (*votre premier **algorithme***), convertir en base 2 les nombres suivants :

239_{10} = **$1\,1\,1\,0\,1\,1\,1\,1_2$**

94_{10} = **$1\,0\,1\,1\,1\,1\,0_2$**

4. La numération hexadécimale

On constate que tout cela devient un peu fastidieux dès que le nombre de bits devient un peu élevé !! C'est en effet la rançon de l'utilisation de seulement deux "chiffres" pour écrire les nombres...
Pour améliorer la compréhension humaine, on a introduit un codage, dit *hexadécimal*, qui permet de « condenser » la représentation des nombres binaires ; la base utilisée est cette fois-ci la **base 16**.

Principe :

Les 16 « chiffres » de la notation hexadécimale sont les suivants :

valeur en base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
valeur en base 2	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111
hexadécimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Pour convertir un nombre binaire vers l'hexadécimal, on regroupe ses bits en groupes de 4 bits (ce que l'on appelle un *nibble* = quartet), et on fait correspondre chaque groupe avec le symbole hexadécimal correspondant.

Exemple : conversion du nombre $124_{10} = 0111\ 1100_2$

- 1er nibble : $0111_2 = 7_{10} = 7_{16}$
- 2ème nibble : $1100_2 = 12_{10} = C_{16}$
  $\Rightarrow 124_{10} = 0111\ 1100_2 = 7C_{16}$

Exercices :

Convertir en hexadécimal les valeurs binaires suivantes :

11111111_2 = **$1111\ \ 1111_2 = FF_{16}$**

10101011001_2 = **$0101\ \ 0101\ \ 1001_2 = 559_{16}$**

2. **De l'hexadécimal au décimal :** en vous inspirant du principe de la décomposition d'un nombre en puissances de la base de numération utilisée, convertir en base 10 les valeurs hexadécimales suivantes sans passer par le binaire :

¤ BAC_{16} = **$B \times 16^2 + A \times 16^1 + C \times 16^0 = 11 \times 16^2 + 10 \times 16^1 + 12 \times 16^0 = 2988_{10}$**

¤ $CABA_{16}$ = **51898_{10}**

3. **Des calculs en hexadécimal** : comme dans toute base, on peut très simplement faire des opérations (*sans conversion !*):

```
          1 1 1                          
          D A D A16                      F A D A16
+         B E B E16              -       B A B A16
-----------------------        -----------------------
          1 9 9 8 16                     4 0 2 0 16
```

(E + A = $24_{10} = 1 \times 16^1 + 8 \times 16^0 = 18_{16}$ .... 1 + D + B = 1 + A + E = $25_{10} = 1 \times 16^1 + 9 \times 16^0 = 19_{16}$)

4. On trouve sur le web des pages « pour geek » qui affichent l'heure de différentes façons, et notamment sous forme hexadécimale.
Quelle était l'heure lorsque la page suivante a été consultée ?

16:2f:33

→ en base 2 : 0001 0110 : 0010 1111 : 0011 0011
soit en base 10 : 22 : 47 : 51