LYCÉE de l'ÉLORN
le choix de votre réussite

Représentation des données
Numération et codage
Opération sur les nombres binaires

I - Numération

1 Les systèmes de base n

1.1 Rappel sur le système décimal

La base 10 utilise 10 symboles différents : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. Un nombre N (entier positif) exprimé dans ce système de numération est défini par le polynôme :

$$N = a_n \times 10^n + a_{n-1} \times 10^{n-1} + \ldots + a_1 \times 10^1 + a_0 \times 10^0$$ (où a_n est un chiffre de rang n)

Exemple : $N = 2023$
$N = 2 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

Les puissances de 10 sont appelées les **poids** ou les **valeurs de position**. Le poids est égal à la base élevée à la puissance de son **rang**.

	Unités	***Dizaines***	***Centaines***	***Milliers***	***10×Milliers***	***100×Milliers***
Chiffre	a_0	a_1	a_2	a_3	a_4	a_5
Rang	0	1	2	3	4	5
Poids	10^0	10^1	10^2	10^3	10^4	10^5

1.2 Système binaire (binaire pur)

Le système binaire est le système de base 2, il utilise deux symboles différents, le 0 et le 1. Chacun des éléments est appelé bit (contraction de binary digit). Nous préciserons la notation d'un nombre binaire avec le préfixe 0b ou en le notant entre parenthèse avec en indice la valeur 2.

Dans ce système, le poids est une puissance de 2. Pour N = 0b110110 ou $(110110)_2$ le polynôme s'écrit :

$$N = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$
$N = 54$

Rappel des puissances de 2 :

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2^n	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536

☞ Convertir en décimal les nombres :

N = 0b1011 = ( )$_{10}$
N = 0b00001011 = ( )$_{10}$
N = 0b11110000 = ( )$_{10}$
N = 0b11111110 = ( )$_{10}$
N = 0b11111111 = ( )$_{10}$
N = 0b1111111111 = ( )$_{10}$

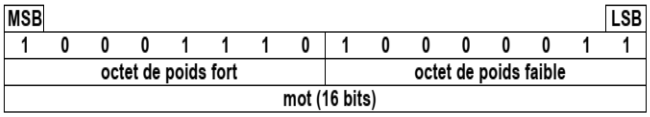
1.3 Format des nombres binaire, définitions :

Octet (byte en anglais) : Nombre binaire formé de 8 bits. *ex : 0b00000010, 0b10101111*

Mot (word) : En dehors de l'unité de transfert usuelle (octet), des regroupements plus importants sont couramment utilisés : le mot de 16 bits = 2 octets (**word)**, le mot de 32 bits = 4 octets (**double word**), et le mot de 64 bits = 8 octets (**quad word**).

MSB, LSB : Dans un mot binaire, le bit situé le plus à gauche est le bit le plus significatif. Il sera désigné comme étant le MSB (Most Significant Bit).Celui situé le plus à droite est le bit le moins significatif, LSB (Least Significant Bit).

Remarque : Pour faciliter les manipulations des mots ont peut le diviser en plusieurs octets. Le mot ci-dessous est constitué de 2 octets, celui situé à gauche sera désigné comme étant l'octet de poids fort, et celui situé à droite, l'octet de poids faible.



Quartet : nombre binaire formé de 4 éléments binaires. *Ex : 0001, 1001 ,1111*

Capacités : La capacité en octets des différents constituants tels que les circuits mémoires ou les disques durs est souvent importante. Il devient indispensable d'utiliser **des unités multiples de l'octet.** Historiquement les préfixes « kilo », « méga », « giga » ... représentaient cette capacité naturellement exprimée par une puissance de 2 mais de nouveaux noms ont été créés pour noter ces valeurs. On parle désormais de «kibi», « mébi », « gibi » …

ko (kB) = kilo-octet (kiloByte) = 10^3 octets = 1000 octets
Mo (MB) = Méga-octet (MegaByte) = 10^6 octets = 1000 ko
Go (GB) = Giga-octet (GigaByte) = 10^9 octets = 1000 Mo
To (TB) = Téra-octet (TeraByte) = 10^{12} octets = 1000 Go

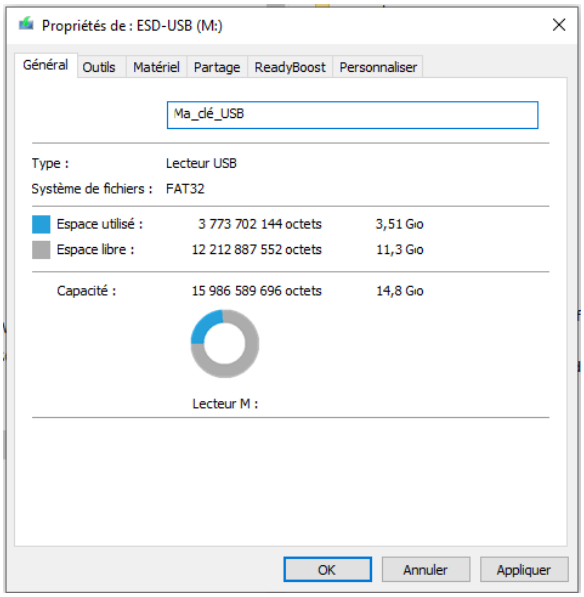
kio (kiB) = kibi-octet (kibiByte) = 2^{10} octets = 1024 octets
Mio (MiB) = Mébi-octet (MebiByte) = 2^{20} octets = 1024 kibi-octet = 1 048 576 octets
Gio (GiB) = Gibi-octet (GibiByte) = 2^{30} octets = 1024 Mébi-octet = 1 073 741 824 octets
Tio (TiB) = Tébi-octet (TebiByte) = 2^{40} octets = 1024 Gibi-octet

k, M, G, T, … = multiple du système international
b=bit, B=Byte, bi=binary

☞ *Dimensionnement d'une clé USB.*

Un clic droit sur le lecteur de ma clé USB me donne les propriétés ci-contre.

Donner la capacité mémoire totale en Gigaoctet (Go), MégaOctet (Mo), en Mébioctet (Mio) puis en gigabit.



1.4 Etendue des valeurs en binaire

En utilisant **n** bits, on peut former $\mathbf{2^n}$ nombres différents et le plus grand d'entre eux est égal à $\mathbf{(2^n-1)}$.

Par exemple si n = 8, $N_{max} = (2^8 - 1) = 255$
On peut former 256 nombres évoluant de 0 0b00000000 à 255 0b11111111.

☞ Quelle est l'étendue d'un nombre défini sur 11 bits ?

☞ On souhaite dénombrer 2 millions de valeurs, quelle doit être la grandeur du mot binaire utilisé ?

1.5 Système hexadécimal

Le système hexadécimal est de **base 16** et utilise donc 16 symboles différents. On y trouvera les dix premiers chiffres décimaux : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 et les 6 premières lettres de l'alphabet : A, B, C, D, E, F qui correspondent respectivement aux nombres décimaux de 10 à 15.

Base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Base 16	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**	**10**	**11**	**12**	**13**
Base 2	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111				

Nous préciserons la notation d'un nombre hexadécimal avec le préfixe 0x ou en le notant entre parenthèse avec en indice la valeur 16.

Exemple : $N = 0xAC53 = (AC53)_{16}$
$N = A \times 16^3 + C \times 16^2 + 5 \times 16^1 + 3 \times 16^0$ *Le poids est une puissance de 16*
$N = 10 \times 16^3 + 12 \times 16^2 + 5 \times 16^1 + 3 \times 16^0$
$N = 44115$

Etendue des valeurs en hexadécimale : En utilisant **n** symboles on peut former $\mathbf{16^n}$ nombres différents et le plus grand d'entre eux est égal à $\mathbf{(16^n-1)}$.

☞ *Donnez la valeur en décimale des nombres suivants :*

N = 0x1234 = N = 0xABCD = N = 0xFFFF =

☞ *Un code secret est utilisé pour ouvrir un fichier informatique. Celui-ci est composé de 8 symboles utilisant l'hexadécimale (par exemple : 0x6A0FD125, 0xBEFFFF58, 0x12345678 …).*
- Quelle est la dimension (nb de bits) du mot binaire correspondant à ce code ?
- Quel est le nombre total de combinaisons envisageables avec ce codage ?
- Imaginons qu'un programme informatique puisse tester un code toutes les millisecondes, quelle serait la durée maximale de découverte du code secret ?

2 Changement de base

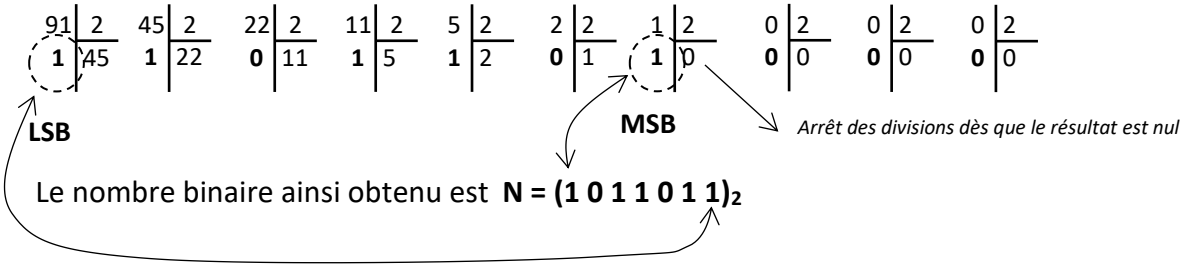
2.1 Conversion d'un nombre décimal en un nombre d'un système d'une autre base :

Un nombre **N** étant donné en base 10, cherchons à l'écrire dans un système de base **b**.

1ère méthode : la division successive

Nous divisons le nombre décimal à convertir par la base b et nous conservons le reste (division entière). Le quotient obtenu est ainsi successivement divisé tant qu'il n'est pas nul. Les restes successifs sont écrits, en commençant par le dernier, de la gauche vers la droite pour former l'expression de N dans le système de base b.

Exemple : Conversion de N = 91 en un nombre du système binaire (b=2).



☞ Avec cette méthode, convertissez en binaire puis en hexadécimale 29, 210 et 2023.

2nd méthode : Approximation successive *(méthode du « compte est bon »)*

On reconstitue le mot binaire en utilisant les poids rencontrés dans la base souhaitée.

☞ Avec cette méthode, convertissez en binaire 42 et 203

Poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
binaire								
binaire								

2.2 Passerelle entre la notation binaire et hexadécimale

Conversion d'un nombre hexadécimal en un nombre binaire :
Chaque symbole du nombre hexadécimal est remplacé par son équivalent écrit sur 4 bits dans le système binaire.

Exemple : $N = (B\ F\ 8)_{16}$
$N = (1011\ \ 1111\ \ 1000)_2$
B F 8

Conversion d'un nombre binaire en un nombre hexadécimal :
C'est l'inverse de la précédente. Il faut donc regrouper les bits du nombre par quartets en commençant par la droite, puis chaque groupe est remplacé par le symbole hexadécimal correspondant à sa valeur.

Exemple : $N = (100001101111)_2$
$N = (1000\ \ 0110\ \ 1111)_2 = (86F)_{16}$

2.3 Fonction de conversion en Python

Exemple sur la console :

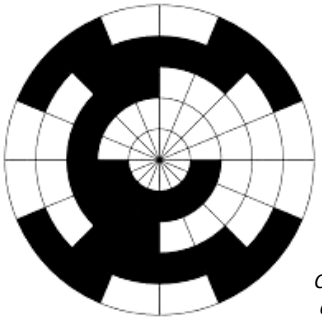
Base10 -> base 2	bin(55) donne 0b110111
Base10 -> base 16	hex(55) donne 0x37
Base2 -> base 10	int('110111',2) ou 0b110111 donne 55
Base16 -> base 10	int('37',16) ou 0x37 donne 55

3 Le code binaire réfléchi (ou code Gray)

Sa propriété réside dans le fait qu'un seul bit change d'état entre deux nombres consécutifs.

Comparaison entre le binaire et le binaire réfléchi

Décimal	Binaire pur	Code Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101



La roue codeuse :
Capteur de position angulaire utilisant le code Gray

II Opérations sur les nombres binaires

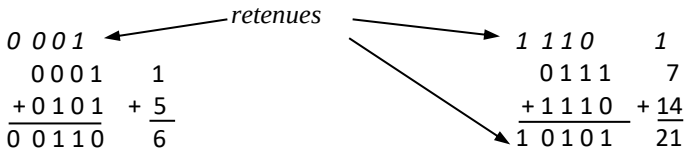
1 Opérations arithmétiques

1.1 Instructions d'additions

Les opérations arithmétiques se font de la même façon en binaire qu'en base décimale. Lorsque l'on dépasse la valeur maximale au niveau du rang *n* (résultat > 1) on génère une retenue au bit de rang *n+1*. De fait, une addition de 2 mots de n bits donne un résultat sur n+1 bits.

Ainsi : 0+0=00 ; 0+1 = 01 ; 1+0=01 ; 1+1=10

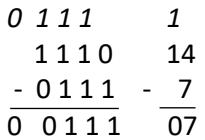
Voyons cela sur une addition de deux mots de 4 bits :



1.2 Instructions de soustractions

Même principe que précédemment en admettant les bases suivantes : 0–0=00 ; 1-0=01 ; 1-1=00 ; 0-1=11 ; 0-1-1=10

Ce qui donne sur 4 bits :



☞ *Faites l'opération 20-14 en binaire.*

2 Complément à 1 et à 2

2 .1 Le complément à 1 (ou complément restreint)

Le complément à un d'un nombre binaire se forme en soustrayant de 1 chaque bit de ce nombre. Cela revient à inverser chacun des bits.

Exemple : Si A = 00010101 alors $\overline{A^1}$ = 11101010

2.2 Le complément à 2 (ou complément vrai)

Soit A un nombre binaire, le complément à deux de A est $\overline{A^2} = \overline{A^1} + 1$

Exemple : Si A = 00001001 → $\overline{A^1}$ = 11110110 → $\overline{A^2}$ = 11110111

On présente ainsi une notion de **nombre négatif** en binaire puisque quel que soit A écrit sur n bits :

$$A + \overline{A^2} = 0 \quad \text{d'où} \quad \mathbf{\overline{A^2} = -A}$$

Intérêt : On peut remplacer les soustractions par les additions

2.3 Les nombres négatifs en base 2

Par convention, le bit de poids fort (MSB) d'un nombre défini son signe (+ ou -). Un nombre négatif s'exprime en complément à 2 en ajoutant un bit de poids fort pour intégrer le signe.

Soit un nombre de 8 bits : **A = a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0**

Bit de signe
A > 0 a_7 = 0
A < 0 a_7 = 1

Signification
A > 0 : binaire pur
A < 0 : Complément à 2

Exemple A = - 3 → |A| = 3 ≡ **0**0000011 d'où $\overline{A^1}$ = 11111100 puis $\overline{A^2}$ = **1**1111101

0 : Signe **+**

Valeur absolue

1 : Signe -

Complément à 2 de +3 ≡ -3

Tableau des nombres signés sur un format de 4 bits :

0	0 000	*-1*	1 111
1	0 001	*-2*	1 110
2	0 010	*-3*	1 101
3	0 011	*-4*	1 100
4	0 100	*-5*	1 011
5	0 101	*-6*	1 010
6	0 110	*-7*	1 001
7	0 111		

A consulter : Connaissez-vous le Bug de l'an 2038 ? → https://fr.wikipedia.org/wiki/Bug_de_l'an_2038

Application :

Donner sur 8 bits la réprésentation binaire des nombres suivants : 1, -1 , +125, -125, +110, -110

……

……

3 Instructions logiques

Les instructions logiques permettent de faire des opérations bit-à-bit sur des nombres binaires. C'est-à-dire en considérant chacun des bits de même poids indépendamment des autres, sans se soucier de la retenue.

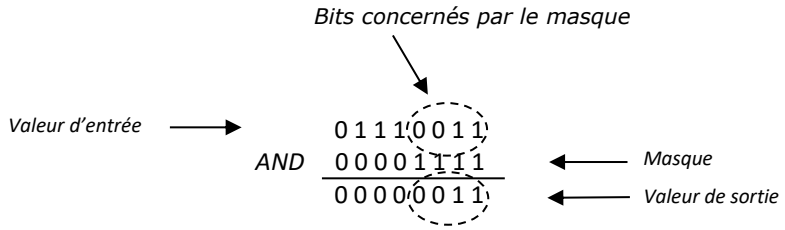
- L'instruction **AND, ET** (& en python) : Cette instruction met le bit du résultat à 1 si les deux bits de même poids des opérandes sont à 1, sinon il le met à zéro.
- L'instruction **OR, OU** (| en python) : Met le bit du résultat à 0 si les deux bits de même poids des opérandes sont à 0, sinon il le met à un.
- La fonction **XOR**, **OU exclusif**, (^ en python) : Met le bit du résultat à 1 si les deux bits de même poids sont différents.
- La fonction **NOT, NON** : Inverse chacun des bits d'un nombre binaire.

Exemple :

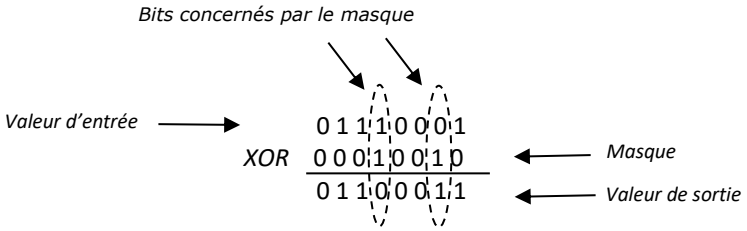
```
    0 1 1 1                0 1 1 1
AND 1 1 1 0            OR  0 1 1 0        NOT (1 1 0 0) = 0 0 1 1
  ---------              ---------
    0 1 1 0                0 1 1 1
```

Application du Masquage :

Il est possible de masquer, c'est-à-dire mettre à zéro tous les bits qui ne nous intéressent pas dans un nombre. Dans l'exemple ci-dessous, seuls les 4 derniers bits de l'octet 01110011 seront récupérés dans le résultat de l'opération 01110011 AND 00001111



Ici l'on procède à l'inversion des bits 2 et 5 de l'octet 01110011 avec l'opération 01110011 XOR 00010010



Exemple d'application sur l'adressage IP :

On utilise le principe du masquage avec l'adressage IP. A partir d'une adresse IP et d'un masque de sous-réseau on peut générer l'adresse commune dite « adresse réseau » des systèmes intégrés à un même réseau.

☞ Compléter ce tableau :

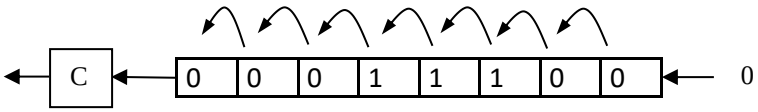
IP décimale	192	168	155	24
IP binaire				
Masque décimal	255	255	240	0
Masque binaire				
Adresse réseau (netid) **= IP and Masque**				

4 Décalage et rotation

Ces instructions permettent de décaler d'un côté ou de l'autre les bits contenus dans une cellule mémoire (plus précisément un des registres d'un microprocesseur). Cette opération a plusieurs applications très intéressantes :

- permettre de lire un à un les bits d'un registre ou d'une cellule mémoire.
- permettre une multiplication par 2^n (en effet le fait de décaler un nombre binaire d'un chiffre à gauche le multiplie par 2, ainsi en effectuant cette opération n fois on obtient une multiplication par 2^n)
- permettre une division par 2^n (comme précédemment mais en effectuant une rotation sur la droite)

Une opération de décalage déplace chacun des bits d'un nombre binaire sur la gauche (ou la droite), mais ceux-ci sortent vers le bit de retenue C lorsqu'ils arrivent au bit de poids fort (ou de poids faible) puis sont définitivement perdus.
exemple:



00011100
00111000 (on décale d'un bit à gauche)
01110000 (on décale d'un bit à gauche)
11100000 (on décale d'un bit à gauche)
11000000 (on décale d'un bit à gauche)

En python :

```
bin(0b0011 << 2)
'0b1100'

3 << 2
12

8 >> 2
2
```

Une opération de rotation agit comme une opération de décalage à la différence près que les bits qui sortent d'un côté rentrent de l'autre.
exemple: