LECON 5 : LES SYSTEMES DE NUMERATION ET DE CODAGE

« *Les ordinateurs sont comme les dieux de l'Ancien Testament : avec beaucoup de règles, et sans pitié.* » - **Joseph Campbell**

« *Compter en octal, c'est comme compter en décimal, si on n'utilise pas ses pouces* » - **TomLehrer**

« *Il y a 10 sortes de gens au monde : ceux qui connaissent le binaire et les autres* » - Anonyme

I. OBJECTIFS

À la fin de cette leçon, vous serez capable de :

- Connaitre les différents systèmes de numération
- Coder un nombre dans une base donnée
- Convertir des nombres d'une base en une autre
- Faire des opérations arithmétiques dans une base quelconque
- Connaître les différents systèmes de codage
- Connaître le principe de codage du son et des images

II. INTRODUCTION

L'ensemble des outils informatiques sont basés sur les mêmes principes de calcul (loi de tout ou rien). Les calculs habituels sont effectués dans le système de numération décimal, par contre le calculateur électronique ne peut pas utiliser ce système car le circuit électronique ne permet pas de distinguer 10 états. Le système de numération binaire ne comportera que 2 états 0 et 1.

Le système de numération utilise habituellement est le système décimal. Un ordinateur étant basé sur le système binaire, il est utile de connaitre les systèmes binaire (base 2), hexadécimal (base 16) et octal (base 8), ainsi que les techniques de conversion entre ces différents systèmes.

De nombreux systèmes ont été utilisés par des peuples et à des époques variés.

- Un système **binaire** (base 2) utilisé dans des langues d'Amérique du Sud et d'Océanie et est à la base du système de numération utilisé en informatique (1 le courant passe, 0 le courant ne passe pas)
- Un système **quinaire** (base 5) était utilisé parmi les premières civilisations, et jusqu'au XX^{e} siècle par des peuples africains, mais aussi, partiellement, dans les notations romaine et maya.
- Un système **sénaire** (base 6)
- Un système **octal** (base 8) est utilisé en langue pame du Nord, au Mexique, et en langue yuki, en Californie, ainsi qu'en informatique.
- Un système **décimal** (base 10) a été utilisé par de nombreuses civilisations, comme les Chinois dès les premiers temps, et, probablement, les Proto-indo-européens. Aujourd'hui, il est de loin le plus répandu.

- Un système **duodécimal** (base 12) est utilisé au Népal par le peuple chepang. On le retrouve, à cause de ses avantages en matière de divisibilité (par 2, 3, 4, 6), pour un certain nombre de monnaies et d'unités de compte courantes en Europe au Moyen Âge, partiellement dans les pays anglo-saxons dans le système impérial d'unités, et dans le commerce. Il sert aussi pour compter les mois, les heures, les huitres et les œufs.
- Un système **hexadécimal** (base 16), très couramment utilisé en électronique ainsi qu'en informatique.
- Un système **vigésimal** (ou vicésimal, base 20) existe au Bhoutan en langue dzongkha, et était en usage chez les Aztèques et, quoiqu'irrégulier, pour la numération maya. Certains pensent qu'il a aussi été utilisé par les Gaulois ou par les Basques dans les premiers temps, mais on ignore en réalité si leur numération avait un caractère décimal ou vigésimal.
- Un système **sexagésimal** (base 60) était utilisé pour la numération babylonienne, ainsi que par les Indiens et les Arabes en trigonométrie. Il sert actuellement dans la mesure du temps et des angles.

III. LA NUMERATION

1. Définition

Un système de numération ou système de nombre est un ensemble de symboles différents, appelés chiffres utilisés pour représenter un nombre. (Numération=manière d'écrire ou d'énoncer les nombre en chiffres).

La numération permet de représenter un mot (ou nombre) par la juxtaposition ordonnée de variable (ou symboles) pris parmi un ensemble. Connaitre la numération revient à connaitre le mécanisme qui permet de passer d'un mot à un autre (comptage, opération).

2. Les systèmes de numération

a. Représentation d'un nombre

- Dans un système de numérotation en base B, un nombre noté $N_{(B)}$ égal à :

$$N_{(B)} = \sum_{k=0}^{n-1} a_k . B^k = a_{n-1}a_{n-2} \dots a_2 a_1 a_{0_{(B)}}$$

Avec :

B : base ou nombre de chiffres différents qu'utilise le système de numérotation.

a_k: Chiffre de rang k

B^k: Pondération associée à a_k

Dans un mot binaire, le bit situé le plus à gauche est le bit le plus significatif, **MSB** (**M**ost **S**ignificant **B**it), celui situé le plus à droite est le bit le moins significatif, **LSB** (**L**ess **S**ignificant **B**it).

- De façon générale dans un système à base quelconque, tout nombre décimal N peut se décomposer de la façon suivante :

$N = a_n b^n + a_{n-1}b^{n-1} + ... + a_2b^2 + a_1b^1 + a_0b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} + ... + a_{-p}b^{-p}$

avec $0 \leq a_i \leq b-1$, **b** est la base du système.

Cette décomposition est unique, notée généralement :

$N = a_n\, a_{n-1}.....a_2\, a_1\, a_0\, ,\, a_{-1}\, a_{-2}..... a_{-p}$ **b**

b. Système décimale : (Base10)

Ce système de numération, usuel dans la vie quotidienne, dispose de dix symboles

(en l'occurrence des chiffres) qui sont:{0, 1, 2, 3, 4, 5, 6, 7, 8,9}

On parle que l'on travaille en base 10.

Exemple :

$7239 = (7.10^3 + 2.10^2 + 3.10^1 + 9.10^0)_{10}$

c. Système binaire : (Base2)

La numération binaire (ou base 2) utilise deux symboles appelés ***BIT*** (***B***inary dig***IT***) : 0 et 1 Cette base est très commode pour distinguer les 2 états logiques fondamentaux.

On écrit:

$(a_{n-1},a_{n-2},...,a_1,a_0)_2 = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_12^1 + a_02^0$

Example:

$(4)_{10} = 1.2^2 + 0.2^1 + 0.2^0 = (100)_2$

$11110010_{(2)} = 1.2^7 + 1.2^6 + 1.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 1.2^1 + 0.2^0 = 242_{(10)}$

Comment trouver le nombre de bits nécessaires à la représentation d'un nombre N donné?

Soit k ce nombre de bits. On a : $2^{k-1} \leq N \leq 2^k$

D'une manière générale un codage sur n bits pourra permettre de représenter des nombres entiers naturels compris entre 0 et 2^{n-1}

Un code à *n* chiffres en base 2 distingue 2^n états ou combinaisons.

Les puissances successives de 2 (1, 2, 4, 8, 16, 32, …) sont appelées poids binaires.

d. Système octal : Base(8)

Ce système de numération est très peu utilisé de nos jours. Anciennement, il servait au codage des nombres dans les ordinateurs de première génération. Il utilise 8 symboles : 0, 1, 2, 3, 4, 5, 6, 7.

$(N)_8 = a_{n-1}8^{n-1} + a_{n-2}8^{n-2} + \ldots + a_1 8^1 + a_0 8^0$

Exemple:

$(572)_8 = (5.8^2 + 7.8^1 + 2.8^0)_{10} = (378)_{10}$

e. Système hexadécimal : Base(16)

Ce système de numération est très utilisé dans les systèmes ordinateurs et microordinateurs ainsi que dans le domaine des transmissions de données. Il comporte 16 symboles les chiffres

de 0 à 9 et les lettres $\{A, B, C, D, E, F\}$

$(N)_{16} = a_{n-1}16^{n-1} + a_{n-2}16^{n-2} + \ldots + a_1 16^1 + a_0 16^0$

Exemple:

$(D62C)_{16} = (13.16^3 + 6.16^2 + 2.16^1 + 12.16^0)_{10} = (54828)_{10}$

$F7_{(16)} = F.16_1 + 7.16_0 = 247_{(10)}$

Note bien:

Table de correspondance entre nombre décimaux, binaires et hexadécimaux :

$N_{(10)}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$N_{(16)}$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$N_{(2)}$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

3. Conversions

a. Conversion du système Décimal vers une base quelconque

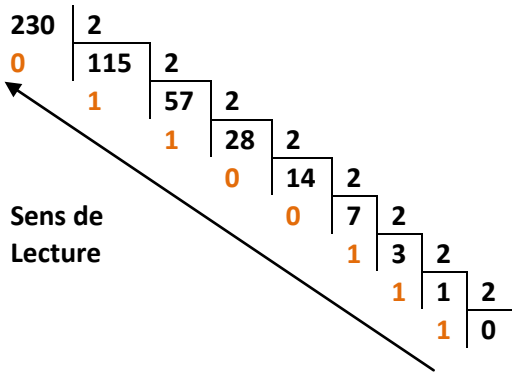
Pour convertir un nombre de la base **10** vers une base **B** quelconques, il faut faire des divisions successives par B et retenir à chaque fois le reste jusqu'à l'obtention à un quotient inférieur à la base B, dans ce cas le nombre s'écrit de la gauche vers la droite en commençant par le dernier quotient allant jusqu'au premier reste.

b. Conversion du système Décimal vers le Binaire par division successive

Pour transférer de la base décimale vers une base$B$, on applique la méthode de division successive. On divise le nombre $_{(B)}$ que l'on désire convertir par 2, puis on réitère l'opération avec le dividende obtenu jusqu'à son annulation. Le nombre cherché s'écrit en plaçant les restes des divisions successives dans l'ordre inverse de leur obtention (sens de lecture de bas vers le haut).

Exemple :

$(230)_{10}$ à convertir en base 2



Le résultat est donc : $(230)_{10}$ **=> $(11100110)_2$**

c. Conversion du système Décimal vers le Binaire par soustraction

Cette méthode consiste à retrancher du nombre la plus grande puissance de 2 possibles, et ainsi de suite dans l'ordre décroissant des puissances. Si on peut retirer la puissance de 2 concernée, on note (1) sinon on note (0) et on contenue de la même manière jusqu'à la plus petite puissance de 2 possible (20 pour les entiers).

Exemple :

$(230)_{10}$ à convertir en base 2

De	230	On peut retirer	128	reste	102	**1**
De	102	On peut retirer	64	reste	38	**1**
De	38	On peut retirer	32	reste	6	**1**
De	6	On ne peut pas retirer	16	reste	6	**0**
De	6	On ne peut pas retirer	8	reste	6	**0**
De	6	On peut retirer	4	reste	2	**1**
De	2	On peut retirer	2	reste	0	**1**
De	0	On ne peut pas retirer	1	reste	0	**0**

Sens ↓

Le résultat est donc : $(230)_{10}$ => $(11100110)_2$

d. Conversion du système Binaire vers l'hexadécimal

Pour convertir du binaire vers l'hexadécimal, on divise le nombre binaire en tranches de 4, en partant de la droite pour la partie entière et en partant de la gauche pour la partie fractionnaire.

Chacun des paquets est ensuite converti en hexadécimal.

Exemple :

$(110101110001)_2 = (1101\ \ 0111\ \ 0001)_2 = (D71)_{16}$

e. Conversion du système Hexadécimal vers le Binaire

C'est le processus directement inverse, on écrit chaque quartet sur 4 bits en complétant éventuellement avec des zéros.

Exemple :

$(FA3)_{16} = (1111\ 1010\ 0111)_2$

f. Conversion du système Binaire vers l'Octal et inversement

On reprend les mêmes principes de la conversion Binaire-Hexadécimal et Hexadécimal-Binaire mais cette fois ci en groupant les données en tranches de 3.

Exemple:

$(101010)_2 = [101]_2\ [010]_2 = (52)_8$

NB: pour la conversion Octal-Hexadécimal et Hexadécimal-Octal, la plus simple méthode et de passer par le système Binaire.

Exemple:

(34.61)8 $= (011100,110001)_2 = (1C.C4)_{16}$.

g. Base i vers base j

- si *i* et *j* sont des puissances de 2, on utilise la base 2 comme relais ;

Exemple

base 8 → base 2 → base 16

- sinon, on utilise la base 10 comme relais.

Exemple

base 5 → base 10 → base 2

4. Représentation des nombres comportant une partie fractionnaire

a. Conversion de la base 10 vers une Base quelconque

Principe de conversion

- **Partie entière :**

  Divisions entières successives par la base (condition d'arrêt : quotient nul).

  Lecture du reste

- **Partie fractionnaire (partie décimale) :**

  Multiplications successives par la base (condition d'arrêt : partie fractionnaire nulle).
  Lecture de la partie entière

Exemple:

Soit à convertir le nombre $(462,625)_{10}$ vers une base quelconque. Pour résoudre ce problème il faut procéder comme suit :

Convertir la partie entière (462)

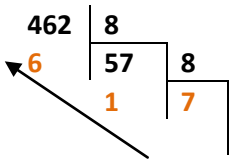
Convertir la partie fractionnaire en faisant des multiplications successives par la base et en conservant à chaque fois le chiffre devenant entier.

$(462,625)_{10}= (?)_8$

$(462,625)_{10}= (?)_2$

$(462,625)_{10}= (?)_{16}$

Partie entière



$(462)_{10}$ => $(716)_8$

Partie fractionnaire

0,625*8=5 ,00

Le résultat est donc : $(462, 625)_{10}$ =$(716,5)_8$

$(462)_{10}$ =$(716)_8$=$(111001110)_2$

0,625*2=1,25

0, 25*2= 0,5

0, 5*2= 1,0

Le résultat est donc : $(462, 625)_{10} = (111001110,101)_2$

Exemple 2 : convertissons 1234,347 en base 2.

- La partie entière se transforme comme suit : $1234_{10} = 10011010010_2$
- On transforme la partie décimale selon le schéma suivant :

0,347·2 = **0**,694	0,347 = 0,**0**...
0,694·2 = **1**,388	0,347 = 0,0**1**...
0,388·2 = **0**,766	0,347 = 0,01**0**...
0,766·2 = **1**,552	0,347 = 0,010**1**...
0,552·2 = **1**,104	0.347 = 0,0101**1**...
0,104·2 = **0**,208	0,347 = 0,01011**0**...
0,208·2 = **0**,416	0,347 = 0,010110**0**...
0,416·2 = **0**,832	0,347 = 0,0101100**0**...
0,832·2 = **1**,664	0,347 = 0,01011000**1**...
0,664·2 = **1**,328	0,347 = 0,010110001**1**...
0,328·2 = **0**,656	0,347 = 0,0101100011**0**...

On continue jusqu'à la précision désirée.

$(462)_{10} = (111001110)_2 = (1CE)_{16}$

0,625*16=10 ,00

Le résultat est donc : $(462, 625)_{10} = (1CE,A)_{16}$

Remarque :

Parfois en multipliant la partie décimal par la Base B, on n'arrive pas à convertir toute la partie entière .ceci est dû essentiellement au fait que le nombre à convertir n'a pas un équivalent exact dans la Base B et sa partie décimale est cyclique.

Exemple

0,1 5*2= 0,3
0 ,3*2=0,6
0 ,6*2=1,2
0 ,2*2=0,4
0 ,4*2=0,8
0 ,8*2=1,6
0 ,6*2=1,2

Le résultat est donc : $(0, 15)_{10} = (0, 0010011\underline{001}...)_2$

On dit $(0,15)_{10}$ est cyclique dans la Base 2 de période <u>1001</u>

b. Conversion d'une Base quelconque vers la Base 10

Pour ce type de conversion, il suffit de représenter le nombre par une combinaison linéaire des puissances successives de la Base et faire la somme, le résultat ainsi trouvé s'écrit directement dans la BASE 10.

Exemple

$(0,001011)_2 = 0*2^{-1}+0*2^{-2}+1*2^{-3}+0*2^{-4}+1*2^{-5}+1*2^{-6} = (0,171875)_{10}$

$(0,32)_8 = 3*8^{-1}+2*8^{-2} = (0,40625)_{10}$

$(AF, 19)_{16} = 10 * 16^1 + 15 * 16^0 + 1 * 16^{-1} + 9 * 16^{-2} = 175,09765625$

5. Représentation des entiers relatif ou nombres signés

a. Introduction

Un entier relatif est un entier pouvant être négatif. Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif, et il faut de plus que les règles d'addition soient conservées. Le signe (+) ou (-) est identifié par un bit, dit le bit de signe et le nombre ainsi formé est dit signé. L'astuce consiste à utiliser un codage que l'on appelle complément à deux. Cette représentation permet d'effectuer les opérations arithmétiques usuelles naturellement.

- **Un entier relatif positif** ou nul sera représenté en binaire (base 2) comme un entier naturel, à la seule différence que le bit de poids fort (le bit situé à l'extrême gauche) représente le signe. Il faut donc s'assurer pour un entier positif ou nul qu'il est à zéro (0 correspond à un signe positif, 1 à un signe négatif). Ainsi, si on code un entier naturel sur 4 bits, le nombre le plus grand sera 0111 (c'est-à-dire 7 en base décimale).
- Sur 8 bits (1 octet), l'intervalle de codage est $[-128, 127]$.
- Sur 16 bits (2 octets), l'intervalle de codage est $[-32768, 32767]$.
- Sur 32 bits (4 octets), l'intervalle de codage est $[-2147483648, 2147483647]$. D'une manière générale le plus grand entier relatif positif codé sur n bits sera $2^{n-1}-1$.
- **Un entier relatif négatif** sera représenté grâce au codage en complément à deux.

b. Principe du complément à deux

1. Écrire la valeur absolue du nombre en base 2. Le bit de poids fort doit être égal à 0. C'est **la représentation en valeur absolue et signe (VAS)**.

2. Inverser les bits : les 0 deviennent des 1 et vice versa. On fait ce qu'on appelle **le complément à un ou complément restreint**.

3. On ajoute 1 au résultat (les dépassements sont ignorés). C'est la représentation par **le complément vrai appelé complément à 2**

Cette opération correspond au calcul de $2^n - |x|$, où n est la longueur de la représentation et $|x|$ la valeur absolue du nombre à coder.

Ainsi -1 s'écrit comme $256 - 1 = 255 = 11111111_2$, pour les nombres sur 8 bits.

Exemple

On désire coder la valeur -19 sur 8 bits. Il suffit :

1. d'écrire 19 en binaire : 00010011
2. d'écrire son complément à 1 : 11101100
3. et d'ajouter 1 : 11101101

La représentation binaire de -19 sur 8 bits est donc 11101101.

On remarquera qu'en additionnant un nombre et son complément à deux on obtient 0. En effet, 00010011 + 11101101 = 00000000 (avec une retenue de 1 qui est éliminée).

La représentation en valeur absolue et signe

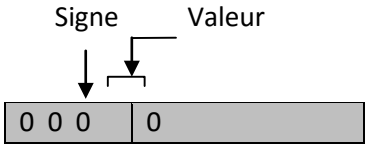
Il s'agit ici d'utiliser un bit pour représenter le signe de la valeur à représenter. Selon que le nombre est positif ou négatif, le bit d'extrême gauche prendra par convention la valeur 0 ou la valeur 1 (0 : positif, 1 : négatif). Par exemple, sur 4 bits, 1 bit sera réservé au signe et trois bits seront utilisés pour représenter les nombres en valeur absolue:

Sur n bits:

- Signe : bit de poids fort (0 : positif, 1 : négatif)
- Valeur absolue : n − 1 bits
- Intervalle de valeurs représentées : $[-2^{n-1} + 1, 2^{n-1} - 1]$

Exemple :

Sur 3 bits, l'intervalle de valeurs représentées : [−3, + 3]



0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	-0
1 0 1	-1
1 1 0	-2
1 1 1	-3

Inconvénients: Cette méthode impose que le signe soit traité indépendamment de la valeur. Il faut donc des circuits différents pour l'addition et la soustraction. De plus, on obtient deux représentations différentes pour 0, soit +0 et -0.

La notation en complément à 1 ou complément restreint

On pourrait définir le complément à 1 comme ce qu'il faut ajouter à une valeur pour obtenir la valeur maximale représentable sur le nombre de bits disponibles. On appel complément à un d'un nombre N un autre nombre N' tel que :

$$N+N'=2^n-1$$

n : est le nombre de bits de la représentation du nombre N.

Exemple :

Soit N=1010 sur 4 bits donc son complément à un de N :

$N'= (2^4 - 1)-N$

$N'= (16-1)-(1010)_2= (15) - (1010)_2 = (1111)_2 – (1010)_2 = 0101$

On constate que le complément à 1 d'un nombre binaire se trouve simplement en remplaçant les 0 par des 1 et les 1 par des 0.

Notons que l'utilisation du complément à 1 pour représenter les nombres négatifs nous donne encore une double représentation pour le 0.

Exemple :

Valeur	Complément à 1
000	111
001	110

010	101
011	100

Exemple :

On va déterminer la valeur décimale représentée par la valeur 101011 en complément à 1 sur 6 bits :

Le bit poids fort indique qu'il s'agit d'un nombre négatif. Le complément à 1 de la valeur (101011)

-CA1 (101011) = - $(010100)_2$= - $(24)_{10}$

- **La notation en complément à 2**

La représentation en complément à deux (complément à vrai) est la représentation la plus utilisée pour la représentation des nombres négatifs dans la machine.

Le complément à 2 d'une valeur binaire est ce qu'il faut ajouter à cette valeur pour qu'elle atteigne une unité de plus que la valeur maximale qu'on peut représenter sur n bits. C'est donc le (complément à 1) + 1.

Cette technique élimine le problème de la double représentation du 0 (+0 et -0) comme c'est le cas dans la représentation "signe et valeur absolue" ou celle du complément à 1.Cela s'explique parce que le complément à 2 permet d'éliminer la double représentation de 0 tout en gardant la facilité de reconnaître le signe par le bit d'extrême gauche. Notons que le complément à 2 du complément à 2 d'un nombre redonne le nombre.

Ainsi, sur 4 bits, avec le signe représenté sur le bit le plus significatif et 3 bits qui permettent de représenter les valeurs, on peut représenter les entiers de -8 à 7, soit un entier négatif de plus qu'un complément à 1.

Sur n bits:

- Complément à 1 : + 1. $|x| + (- |x|) = 2^n$

- Intervalle de valeurs représentées : $[-2^{n-1},2^{n-1} - 1]$

Exemple :

Valeur	Complément à 2
001	111
010	110
011	101

- **Truc**

Pour transformer de tête un nombre binaire en son complément à deux, on parcourt le nombre de droite à gauche en laissant inchangés les bits jusqu'au premier 1 (compris), puis on inverse tous les bits suivants. Prenons comme exemple le nombre 20 : 00010100.

1. On garde la partie à droite telle quelle : 00010**100**
2. On inverse la partie de gauche après le premier un : **11101**100
3. Et voici −20 : 11101100

> Le 4 juin 1996, une fusée Ariane 5 a explosé 40 secondes après l'allumage. La fusée et son chargement avaient coûté 500 millions de dollars. La commission d'enquête a rendu son rapport au bout de deux semaines. Il s'agissait d'une erreur de programmation dans le système inertiel de référence. À un moment donné, un nombre codé en virgule flottante sur 64 bits (qui représentait la vitesse horizontale de la fusée par rapport à la plate-forme de tir) était converti en un entier sur 16 bits. Malheureusement, le nombre en question était plus grand que 32768 (le plus grand entier que l'on peut coder sur 16 bits) et la conversion a été incorrecte.

6. La représentation en virgule flottante simplifiée

a. Introduction

Il arrive dans de nombreux domaines que l'intervalle des valeurs numériques pertinentes soit particulièrement étendu. L'astronomie en est un exemple extrême puisque certains calculs peuvent faire intervenir simultanément la masse du soleil (2.10^{30} kg) et la masse de l'électron (9.10^{-31}kg). Ces deux nombres diffèrent de plus de 60 ordres de grandeur (10^{60})!

Des calculs faisant intervenir ces nombres pourraient s'effectuer en précision multiple, avec par exemple des nombres de 62 chiffres. Tous les opérandes et tous les résultats seraient représentés par des nombres de 62 chiffres. Cependant, la masse du soleil n'est connue qu'avec une précision de 5 chiffres, et il n'y a en physique pratiquement aucune mesure que l'on puisse réaliser avec une précision de 62 chiffres. Une solution serait alors d'effectuer les calculs avec une précision de 62 chiffres et de laisser tomber 50 ou 60 d'entre eux avant d'annoncer les résultats, mais ceci est coûteux à la fois en espace mémoire et en temps de calcul.

En fait, ce qu'il faut est un système permettant de représenter des nombres, tel que la taille de l'intervalle des nombres "exprimables" soit indépendante du nombre de chiffres significatifs.

b. Principe de la représentation en virgule flottante

Le nombre *N* est représenté sous la forme :

exposant	mantisse

1ère approche :

Soit $N = a_3\, a_2\, a_1\, a_0$, a−−1$_3a$−2 a−3 : N peut se noter :

(a6 a5 a4 a3 a2 a1 a0).2^{-3}

Mantisse Exposant

exposant = −3

mantisse = $a_6\, a_5\, a_4\, a_3\, a_2\, a_1\, a_0$

Les valeurs de la mantisse et l'exposant seront notées en complément à 2 en mémoire du calculateur.

Exemple

Soit la mémoire de taille suivante :

4 bits	12 bits
exposant	mantisse

Coder la valeur 26,75 en virgule flottante.

$(26,75)_{10} = (11010,110)_2$

$(11010,11)_2 = (11010110).2^{-3}$

Exposant = -3 et mantisse= 11010110

1101	0000011010110

exp = -3 mantisse = 214

$26,75 = 214.2^{-3}$

- **2ème approche :**

C'est la méthode inverse de la précédente : on considère que le bit le plus à gauche de la mantisse a pour poids 2^{-1}.

Soit : $N = a_3\, a_2\, a_1\, a_0$, $a_{-1}\, a_{-2}\, a_{-3}$

N peut se noter :

(0, $a_{-1}\, a_{-2}\, a_{-3}\, a_{-4}\, a_{-5}\, a_{-6}\, a_{-7}$).$2^{-4}$

Mantisse Exposant

Exemple

Même exemple que précédemment :

$(26,75)_{10} = (11010,110)_2 \rightarrow (0,11010110).2^5$

0101	110101100000

Remarque

Les ordinateurs utilisent cette représentation avec 32 bits pour la mantisse et 8 bits pour l'exposant. En général, on utilise la représentation inverse, avec le bit le plus à gauche égal à 1*, soit une mantisse normalisée* $\Rightarrow 0,5 \leq M < 1$

7. La représentation IEEE 754

a. Présentation

Le standard IEEE 754 définit trois formats : les nombres en simple précision sur 32 bits, les nombres en double précision sur 64 bits, et les nombres en représentation intermédiaire sur 80 bits. La représentation sur 80 bits est principalement utilisée en interne par les processeurs pour minimiser les erreurs d'arrondi.

Un nombre *N* de 32 bits est représenté sous la forme :

s	exposant	mantisse

où le signe « s » est codé sur 1 bit, l'exposant est codé sur 8 bits en code relatif à 127, et la mantisse sur 23 bits.

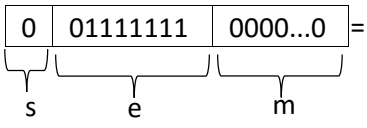
Un nombre de 64 bits (double précision) utilise la même représentation à ceci près que la taille de l'exposant est portée à 11 bits en code relatif à 1023, et celle de la mantisse à 52 bits.

Une mantisse normalisée commence toujours par un bit 1, suivi par la virgule, puis par le reste de la mantisse. Le bit initial, toujours présent et toujours à 1 dans une mantisse normalisée est implicite et non représenté. La valeur de la mantisse est appelée « significande » ; le significande a donc une valeur implicite $1 \leq x < 2$.

Exemple 2.10

$1 = 2^0 \times (1+0)$

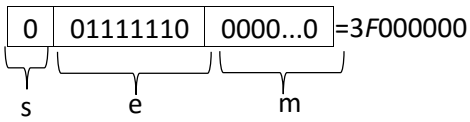
Le bit de signe sera 0, l'exposant, en code relatif à 127 sera représenté par 127 = 01111111, et le significande vaut 1, ce qui résulte en une mantisse dont tous les bits sont à 0. La représentation IEEE simple precision IEEE 754 du nombre 1 est donc :

Code(1)= 

= 3*F*800000

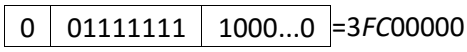
$-0.5 = 2^{-1} \times (1+0)$

Le bit de signe est 0, l'exposant, en code relatif à 127 est représenté par 127−1 = 01111110, et le significande vaut 1, ce qui résulte en une mantisse dont tous les bits sont à 0. La représentation IEEE simple précision IEEE 754 du nombre 0,5 est donc :

Code(0,5)= 

$-1.5 = 2^{0} \times (1+2^{-1})$

Le bit de signe est 0, l'exposant, en code relatif à 127 est représenté par 127 = 01111111, et le significande vaut 1,1, ce qui résulte en une mantisse dont le premier bit est à 1 et les 22 suivants à 0. La représentation IEEE simple précision IEEE 754 du nombre 1,5 est donc :

Code(1,5) = 

b. Nombres spéciaux

En arithmétique à virgule flottante on peut obtenir un résultat valable, ou alors rencontrer un problème de dépassement par valeur supérieure (*overflow*) lorsque le résultat est trop grand pour pouvoir être représenté, ou par valeur inférieure (*underflow*) lorsque le résultat est trop petit.

- **Dépassement par valeur inférieure**

Cette situation arrive lorsqu'un résultat est trop petit pour pouvoir être représenté. Le standard IEEE 754 résout partiellement le problème en autorisant dans ce cas une représentation dénormalisée. Une représentation dénormalisée est caractérisée par le fait d'avoir un code d'exposant complètement nul, ce qui est interprété comme une indication du fait que le bit de poids fort de la mantisse, implicite, est cette fois à 0 au lieu d'être à 1. De cette façon, le plus petit nombre « exprimable » est : $2^{-127} \times 2^{-23} = 2^{-150} \sim 10^{-45}$.

Cependant, il faut remarquer que plus le nombre représenté est petit, moins sa mantisse comportera de bits significatifs. Ce schéma permet une approche « douce » du phénomène de dépassement par valeur inférieure, en sacrifiant la précision lorsqu'un résultat est trop petit pour admettre une représentation normalisée.

- **Zéro**

Zéro est représenté sous la forme d'un nombre dénormalisé. Ceci résulte en deux représentations possibles pour zéro : l'une pour +0, l'autre pour −0. Ces représentations sont caractérisées par un bit de signe suivi par 31 zéros.

- **Dépassement par valeurs supérieures**

Le dépassement par valeurs supérieures ne peut pas être traité comme le dépassement par valeurs inférieures, et est indiqué par un code d'exposant dont tous les bits sont à 1, suivi par une mantisse dont tous les bits sont à 0. Ceci est interprété comme représentant l'infini. L'infini peut être positif ou négatif, en fonction de la valeur du bit de signe. L'infini peut être utilisé dans les calculs et les résultats correspondent au sens commun : $\infty+\infty=\infty$; $x/\infty= 0$; $x/0 =\infty$.

Not a Number (NaN)

Cependant, certaines opérations peuvent ne conduire à aucun résultat exprimable, comme $\infty/\infty=?$ ou $0x\infty=?$.

Le résultat de telle opération est alors indiqué par un autre code spécial : le code d'exposant a tous les bits à 1, suivi par une mantisse non nulle. Le « nombre » correspondant est appelé NaN (*Not a Number*) : c'est un nonnombre.

c. Résumé

- La valeur d'un nombre est indépendante de la base dans laquelle il est noté.
- Un nombre binaire peut avoir plusieurs valeurs différentes selon le système de représentation. Soit le nombre binaire $a_n a_{n-1} ... a_1 a_0$. Ce nombre vaut :
  - ✓ $a_n.2^n + a_{n-1}.2^{n-1} +...+ a_1.2+ a_0$ en représentation non signée
  - ✓ $-a_n.2^n + a_{n-1}.2^{n-1} +...+ a_1.2+ a_0$ en représentation signée complément à 2
  - ✓ $1- a_n.2^n + a_{n-1}.2^{n-1} +...+ a_1.2+ a_0$ en représentation signée complément à 1
  - ✓ $-1^{a_n} \times (a_{n-1}.2^{n-1} +...+ a_1.2+ a_0)$ en représentation module et signe
- Les opérations arithmétiques obéissent en binaire aux mêmes règles qu'en décimal, il suffit juste de se rappeler que la base de numération est 2 et non plus 10.

8. Opérations arithmétiques

a. L'addition

Il suffit de savoir que :

0+0=0

0+1=1

1+0=1

1+1=10

Et d'effectuer éventuellement une retenue comme dans le cas d'une addition décimal

Exemple

			1	0	1	1	0	1
+				1	0	0	1	0
			1	1	1	1	1	1

			1	1	0	1	1
+				1	1	0	1
			1	0	1	0	0	0

Remarque: L'addition s'effectue de la même manière dans les autres bases.

Exemple 2

	0	1	1	1	,	1	1
+	0	1	0	1	,	0	1
=	1	1	0	1	,	0	0

Exercice:

Effectuer les opérations suivantes:

$(37)_8 + (65)_8 + (116)_8 = (242)_8$

$(D5E)_{16} + (2F36)_{16} = (3C94)_{16}$

b. La soustraction

On peut opérer comme dans la soustraction décimale. Voilà ci-dessous la table de soustraction binaire:

0-0=0

0-1=1 avec un retenue de 1

1-0=1

1-1=0

Exemple:

	1		**1**	**1**			
	1	**0**	**1**	**0**	**0**	**1**	**1**
-		**1**	**0**	**1**	**1**	**0**	**1**
	0	**1**	**0**	**0**	**1**	**1**	**0**

Donc 1010011-101101 = 100110 en base 2

Remarque: la soustraction s'effectue de la même manière dans les autres bases.

Exemple 2

	1	0	1	,	0
-	0	1	1	,	1
	0	0	1	,	1

Exercice:

Effectuer les opérations suivantes:

$(137)_8 - (63)_8 = (54)_8$

$(F23)_{16} - (2A6)_{16} = (C7D)_{16}$

$(FD28)_{16} - (E5E)_{16} - (2F36)_{16} = (FD28)_{16} - [\,(E5E)_{16} + (2F36)_{16}\,] = (FD28)_{16}\ (3D94)_{16} = (BF94)_{16}$

c. La multiplication

La multiplication en binaire est très simple, voilà la table de multiplication:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Remarque: On doit bien tenir compte des décalages.

Exemple:

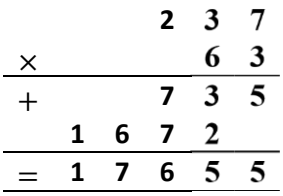
					1	**0**	**1**	**1**
x					**1**	**1**	**0**	**1**
				1	**0**	**1**	**1**	
					1	**1**		
+		**1**	**0**	**1**	**1**			
	1	**0**	**0**	**0**	**1**	**1**	**1**	**1**

Donc 1011 x 1101 = 10001111 en base 2

Exercice:

Effectuer les opérations suivantes:

$(237)_8 * (63)_8 = (17655)_8$



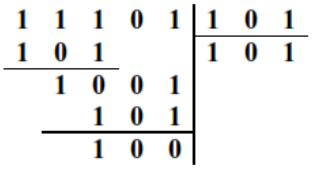
$(F3)_{16} * (206)_{16} = (1EBB2)_{16}$

d. La division

La division entre deux nombres binaires est identique à la division euclidienne.

A		B		C
0	÷	0	=	impossible
0	÷	1	=	0
1	÷	0	=	impossible
1	÷	1	=	1

Exemple:



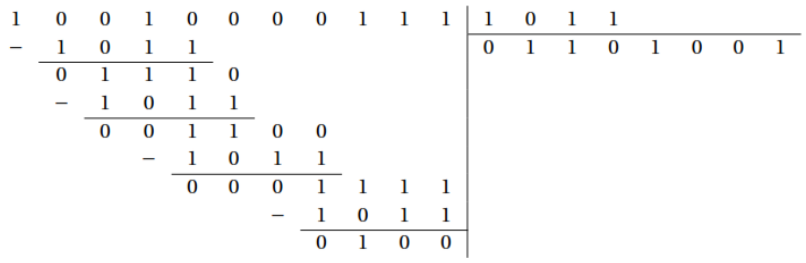
Il suffit en fait de soustraire 101 lorsqu'on le peut, et d'abaisser le chiffre suivant :

$$11101 = 101 \times 101 + 100$$

Exemple 2 :

La division du nombre (10010000111)2 par (1011)2 = (1101001)2 reste (100)2

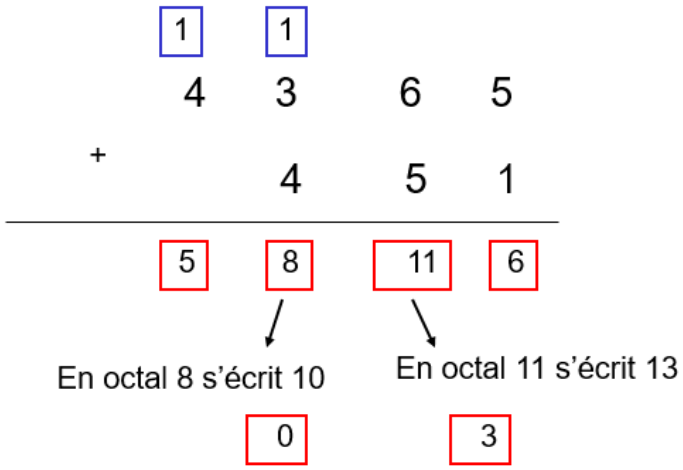
C’est-à-dire 1159/11 =105, reste 4.



9. Opération arithmétique en octal et hexadécimal

- Soit l’opération 4365 + 451 à effectuer en octal.

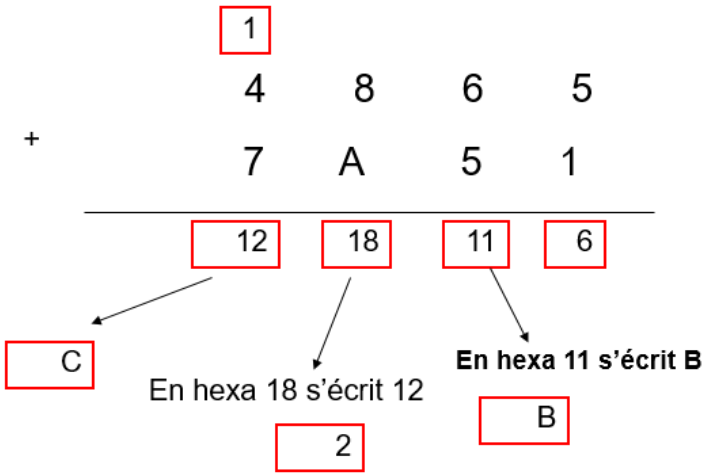
  On a :



Le résultat final est : $(\mathbf{5036})_8$

- Soit l’opération 4865 + 7A51 à effectuer en octal.

  On a :



Le résultat final est : $(\mathbf{C2B6})_{16}$

IV. LE CODAGE

1. Notion de codage

On appelle codage l'opération qui consiste à faire correspondre à tout caractère (lettre, chiffre, signe,...) un symbole où un ensemble de symboles particuliers appelés mot de code.

2. Les codes numériques pondérés:

a. Les codes binaires purs

Ceux sont des codes qui donnent à chaque combinaison une équivalence décimale et dans laquelle chaque rang d'élément binaire à un poids précis. Le code binaire naturel et ses dérivés

(octal et hexadécimal) répondent aux règles classiques de l'arithmétique des nombres positifs.

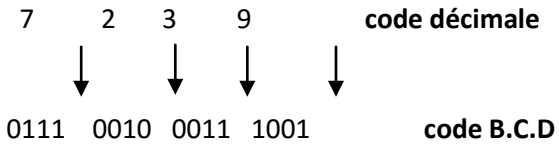
Exemple:

$(7)_{10}$ sur 8 bits = $(0000\ 0111)_2$

b. Les codes DCB (Décimal Codé Binaire)

Ce code DCB, en Anglais BCD (Binary Coded Decimal), consiste à représenter chaque chiffre d'un nombre décimal par son équivalent binaire sur 4 bits.

Le BCD s'appelle en français Décimal codé Binaire (DCB). Si on représente chaque chiffre d'un nombre décimal par son équivalent binaire, on obtient le code dit décimal codé binaire (abrégé dans le reste du texte par BCD). Comme le plus élevé des chiffres décimaux est 9, il faut donc 4 bits pour coder les chiffres. Illustrons le code BCD en prenant le nombre décimal 874 et en changeant chaque chiffre pour son équivalent binaire; cela donne:



De nouveau, on voit que chaque chiffre a été converti en son équivalent binaire pur. Notez qu'on fait toujours correspondre 4 bits à chaque chiffre. Le code BCD établit donc une correspondance entre chaque chiffre d'un nombre décimal et un nombre binaire de 4 bits. Évidemment, seuls les groupes binaires 0000 à 1001 sont utilisés. Le code BCD ne fait pas usage des groupes 1010, 1011, 1100, 1101, 1110 et 1111. Autrement dit, seuls dix des 16 combinaisons des 4 bits sont utilisés. Si l'une des combinaisons "inadmissibles" apparaît dans une machine utilisant le code BCD, c'est généralement le signe qu'une erreur s'est produite.

3. Les codes numériques non pondérés:

a. Le code Gray

Appelé aussi code binaire réfléchi, il appartient à la famille dite « codes à distance » de faite qu'une représentation codé ne diffère de celle qui la précède que par un bit comme le montre le tableau ci-dessous.

Code décimale	Code binaire	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
10	1010	1111
11	1011	1110

12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Intérêts et applications

- **Éviter des états transitoires**

Le fait de modifier plusieurs bits lors d'une simple incrémentation peut mener, selon le circuit logique, à un état transitoire indésirable dû au fait que le chemin logique de chaque bit dispose d'un délai différent. Cela apparait pour les capteurs de positions, par exemple sur des règles optiques. Ce code permet de contourner cet aléa en forçant la commutation d'un seul bit à la fois, évitant ainsi les états transitoires.

- **Modulo et roue codeuse**

On remarquera que le passage du maximum (*sept* sur 3 bits) à *zéro* se fait également en ne modifiant qu'un seul bit. Ceci permet par exemple d'encoder un angle, comme la direction d'une girouette : 0=*Nord*, 1=*Nord-Est*, 2=*Est*, ... 7=*Nord-Ouest*. Le passage de *Nord-Ouest* à *Nord* se fait également sans problème en ne changeant qu'un seul bit.

- **Souris**

Le décodage des signaux lumineux d'un axe de souris mécanique est un décodage de code de Gray à 2 bits (décodage différentiel dans ce cas, car ce que l'on veut obtenir n'est pas la valeur décodée mais les transitions ±1 mod 4 de la valeur décodée).

- **Tables de Karnaugh**

Le code Gray sert également dans les tables de Karnaugh utilisées lors de la conception de circuits logiques.

b. Codes alphanumériques (codage des caractères)

Les caractères également doivent être représentés en binaire de manière unique. La convention adoptée est d'associer à chaque caractère un nombre décimal et de convertir ce nombre en binaire. Il existe plusieurs normes:

Code ASCII:

Le code ASCII (ASCII = American Standard Code for Information Interchange) est initialement un code à 7 bits, qui permet le codage de 128 caractères. Il englobe des lettres, des chiffes, des signes de

ponctuations et un certain nombre de signaux de commande. Toutes ces correspondances sont fixées par l'American National Standards Institutes

Code ASCII (8 bits):

Ce code est normalisé ISO (International Standard Organisation). C'est une extension du code ASCII étendu. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles.

UNICODE (16 bits):

Ce code permet de représenter des caractères appartenant à plusieurs langues (arabes, hébreu, japonais, coréen,...) : 65536 caractères.

Code EBCDIC (IBM).

Le code EBCDIC (Extended Binary Decimal Interchange Code) est un code à 8 éléments binaires utiles, soit 256 combinaisons possibles.

- **Table du code ASCII standard (7 bits):**

	000	***001***	***010***	***011***	***100***	***101***	***110***	***111***
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}

1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Exemple

A=>$(65)_{ASCII}$ => $(01000001)_2$ => $(41)_H$

[=> $(91)_{ASCII}$ => $(01011011)_2$ => $(5B)_H$

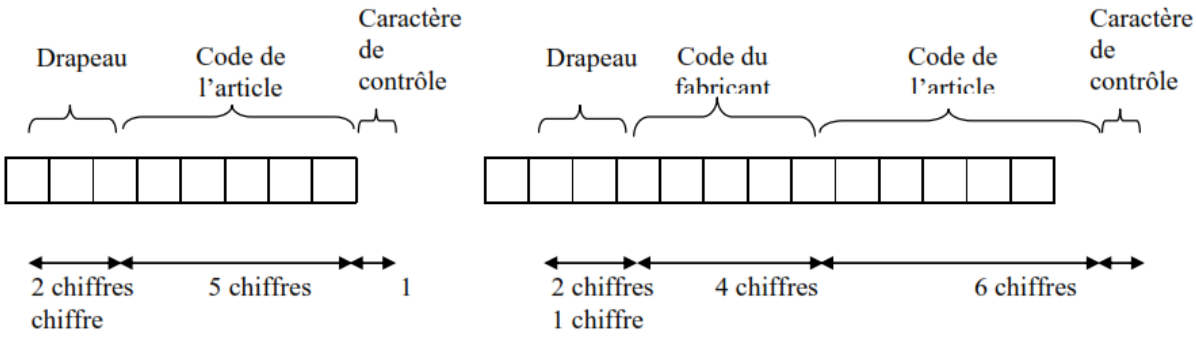
Pour le tester, il suffit de vous placer dans une zone de saisie ou fichier et de maintenir la touche **Alt** du clavier appuyé, puis de taper **65**, vous aurez la lettre **A** apparaître.

c. Le code à barres

Le code à barres qui figure sur la plupart des emballages des produits de consommation courante est la fiche d'identité, traduite en code, du produit sur lequel il est apposé. Il peut indiquer le pays d'origine, le nom du fabricant, celui du produit, sa référence.

Il permet de suivre la traçabilité du produit. Le code imprimé parfois directement sur l'emballage, se présente également sous la forme d'une étiquette rectangulaire collée. Il est composé de barres et d'espaces larges ou étroits dont le nombre correspond à un ensemble de données numériques ou alphanumériques.

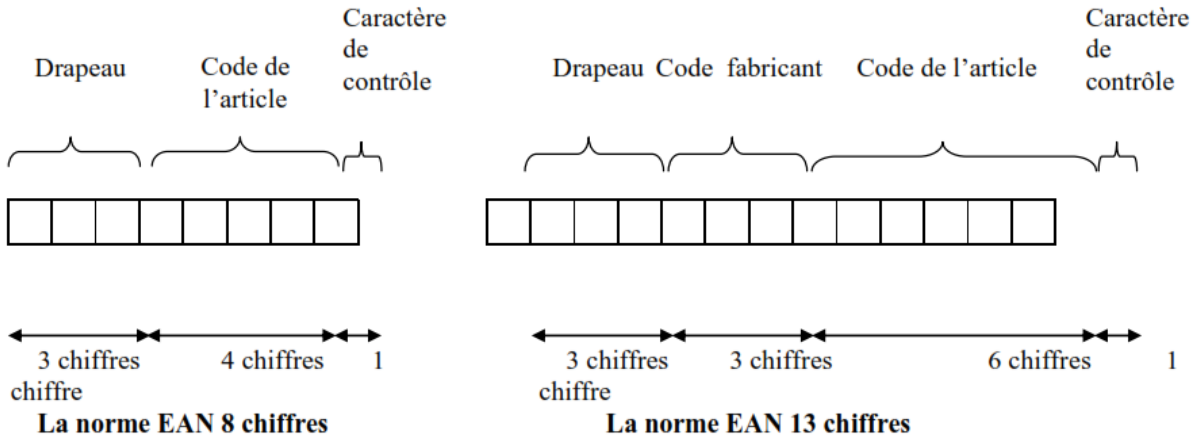
Ce marquage comporte un certain nombre de barres verticales, ainsi que des chiffres au nombre de 13. Le premier chiffre indique le pays d'origine, les 5 suivants sont ceux du code du fabricant, les 6 autres ceux du code de l'article, le 13^e est une clé de contrôle. Les barres sont le codage de ces chiffres sur 7 bits. A chaque chiffre est attribué un ensemble de 7 espaces, blanc ou noirs.



Remarque :

Dans le cas où le code pays comporte 3 caractères, le code fabricant ne comporte que 3 caractères.

Exemples :



La norme EAN 8 chiffres **La norme EAN 13 chiffres**

d. Les codes QR

Le **code QR** (en anglais ***QR Code***) est un type de code-barres en deux dimensions (ou code matriciel *datamatrix*) constitué de modules noirs disposés dans un carré à fond blanc. L'agencement de ces points définit l'information que contient le code.

Ce sont les nouveaux codes-barres du troisième millénaire. Alors que le code barre classique ne permet qu'un codage horizontal, le QR code est en deux dimensions et comprend donc plus d'informations.

Le QR Code a été inventé en 1994 par Denso Wave, une société japonaise qui travaillait pour Toyota. Le QR Code n'est toutefois pas le seul code-barres 2D existant. Un autre exemple est le Datamatrix, et son pendant payant le **Flashcode** français.

QR (abréviation de l'anglais *Quick Response*) signifie que le contenu du code peut être décodé rapidement après avoir été lu par un lecteur de code-barres, un téléphone mobile, un smartphone, ou encore une webcam. Son avantage est de pouvoir stocker plus d'informations qu'un code à barres[1], et surtout des données directement reconnues par des applications, permettant ainsi de déclencher facilement des actions comme :

- naviguer vers un site internet, visiter un site web ou mettre l'adresse d'un site en marque-page pour, par exemple, montrer un point géographique sur une carte, telle que OpenStreetMap, Google Maps ou Bing Maps ;
- regarder une vidéo en ligne ou un contenu multimédia ;
- se connecter à une borne Wi-Fi ;
- déclencher un appel vers un numéro de téléphone ou envoyer un SMS ;

- envoyer un courriel ;
- faire un paiement direct via son téléphone portable (Europe et Asie principalement) ;
- ajouter une carte de visite virtuelle (vCard, MeCard) dans les contacts, un rendez-vous ou un événement (iCalendar) dans l'agenda électronique ;
- afficher un texte ou rédiger un texte libre (sa version la plus grande permet d'inclure un texte d'environ 500 mots);
- etc.

Figure 1 : Code QR

e. Le code ISBN

L'ISBN (*International Standard Book Number*) est un numéro international qui permet d'identifier, de manière unique, chaque livre publié. Il est destiné à simplifier la gestion informatique des livres dans les bibliothèques, librairies, etc.

Le numéro ISBN-10 se compose de quatre segments, trois segments de longueur variable et un segment de longueur fixe, la longueur totale de l'ISBN comprend dix chiffres (le 1er janvier 2007, la longueur a été étendue à 13 chiffres en ajoutant un groupe initial de 3 chiffres).

Si les quatre segments d'un ancien code ISBN à 10 chiffres sont notés A - B - C - D :

- A identifie un groupe de codes pour un pays, une zone géographique ou une zone de langue.
- B identifie l'éditeur de la publication.
- C correspond au numéro d'ordre de l'ouvrage chez l'éditeur.
- D est un chiffre-clé calculé à partir des chiffres précédents et qui permet de vérifier qu'il n'y a pas d'erreurs. Outre les chiffres de 0 à 9, cette clé de contrôle peut prendre la valeur X, qui représente le nombre 10.

Calcul du chiffre-clé d'un numéro ISBN-10

- On attribue une pondération à chaque position (de 10 à 2 en allant en sens décroissant) et on fait la somme des produits ainsi obtenus.
- On conserve le reste de la division euclidienne de ce nombre par 11. La clé s'obtient en retranchant ce nombre à 11. Si le reste de la division euclidienne est 0, la clé de contrôle n'est pas 11 ($11 - 0 = 11$) mais 0.
- De même, si le reste de la division euclidienne est 1, la clé de contrôle n'est pas 10 mais la lettre X.

Le nombre 11 étant premier, une erreur portant sur un chiffre entraînera automatiquement une incohérence du code de contrôle. La vérification du code de contrôle peut se faire en effectuant le même calcul sur le code ISBN complet, en appliquant la pondération 1 au dixième chiffre de la clé de contrôle (si ce chiffre-clé est X, on lui attribue la valeur 10) : la somme pondérée doit alors être un multiple de 11.

Exemple

Pour le numéro ISBN (à 9 chiffres) 2-35288-041, quelle est la clé de contrôle ?

Code ISBN	2	3	5	2	8	8	0	4	1
Pondération	10	9	8	7	6	5	4	3	2
Produit	20	27	40	14	48	40	0	12	2

La somme des produits est **203**, dont le reste de la division euclidienne par 11 est **5**.

La clé de contrôle est donc 11 - 5 = **6**. L'ISBN au complet est : 2-35288-041-**6**.

La vérification de la clé complète à 10 chiffres donne la somme pondérée 203 + 6 = 209, qui est bien un multiple de 11.

Depuis le 1er janvier 2007, les ISBN sont passés à 13 chiffres

Pourquoi une réforme du système ISBN ?

- Pour augmenter la capacité de numérotation du système ISBN qui était devenu insuffisant, notamment en raison de l'augmentation du nombre de publications électroniques.
- Pour rendre l'ISBN complètement compatible avec l'EAN-13 qui sert à la génération des codes-barres :

Qu'est-ce qui a changé ?

- On fait précéder l'ISBN du préfixe « 978 » et on recalcule la clé de contrôle.
- L'ISBN à 13 chiffres est identique à l'EAN-13 servant à la génération du code à barres et figurant sous celui-ci.
- Le préfixe « 979 » sera introduit quand les segments ISBN existants seront épuisés.
- Les segments identifiant les éditeurs ne demeureront pas les mêmes lorsque le préfixe passera en 979.
- Depuis le 1er janvier 2007, l'ISBN à 13 chiffres est utilisé pour toutes les commandes et toutes les transactions commerciales, manuelles ou électroniques.

- Depuis le 1er janvier 2007, les codes à barres sont surmontés de l'ISBN à 13 chiffres segmenté (avec des tirets) alors que l'EAN-13 correspondant est indiqué en dessous des codes à barres (sans tiret ni espace).

Algorithme permettant de générer l'EAN Bookland à partir de l'ISBN

1. Garder les 9 premiers chiffres de l'ISBN.

2. Ajouter le préfixe Bookland de l'EAN.

3. Entrer/lire les facteurs de pondération constants associés à chaque position de l'EAN (1 3 1

3 1 3 1 3 1 3 1 3).

4. Multiplier chaque chiffre par le facteur de pondération qui lui est associé.

5. Diviser la somme par le nombre du module (10) pour trouver le reste.

6. Si le reste vaut 0, le numéro de contrôle est aussi 0. Sinon, le numéro de contrôle est 10

moins le reste trouvé. Ajouter ce suffixe.

Convertissons l'ISBN 0-8436-1072-7 :

ISBN				0	8	4	3	6	1	0	7	2	7
Ajout du préfixe 978	9	7	8	0	8	4	3	6	1	0	7	2	
Facteurs	1	3	1	3	1	3	1	3	1	3	1	3	
Produits	9	21	8	0	8	12	3	18	1	0	7	6	

f. Code de Hamming

Un code de Hamming permet la détection et la correction automatique d'une erreur si elle ne porte que sur une lettre du message. Un code de Hamming est **parfait**, ce qui signifie que pour une longueur de code donnée, il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens, son rendement est maximal.

Structure d'un code de Hamming

- les m bits du message à transmettre et les n bits de contrôle de parité.
- longueur totale : 2^n-1
- longueur du message : $m = (2^n-1)-n$
- on parle de code $x-y$: x est la longueur totale du code ($n+m$) et y la longueur du message (m)
- les bits de contrôle de parité C_i sont en position 2^i pour $i = 0, 1, 2,...$
- les bits du message D_j occupe le reste du message.

7	6	5	4	3	2	1
D_3	D_2	D_1	C_2	D_0	C_1	C_0

Structure d'un code de Hamming 7−4

Exemples de code de Hamming

- un mot de code 7−4 a un coefficient d'efficacité de 4/7 = 57 %
- un mot de code 15−11 a un coefficient d'efficacité de 11/15 = 73 %
- un mot de code 31−26 a un coefficient d'efficacité de 26/31 = 83 %

Retrouver l'erreur dans un mot de Hamming

Si les bits de contrôle de parité C_2, C_1, C_0 ont tous la bonne valeur, il n'y a pas d'erreurs ; sinon la valeur des bits de contrôle indique la position de l'erreur entre 1 et 7. Le code de Hamming présenté ici ne permet de retrouver et corriger qu'une erreur.

Pour savoir quels bits sont vérifiés par chacun des bits de contrôle de parité, il faut construire le tableau ci-dessous.

On numérote les lignes de 1 à $x = 2^n - 1$ dans la colonne de droite (prenons par exemple $x = 7$), puis on convertit chaque nombre en binaire et l'on écrit chaque bit dans les colonnes de gauche. On colorie de la couleur de C_i les nombres de droite s'il y a un 1 dans dans la colonne C_i . Par exemple, 5 sera coloré en vert et en rouge, car sur la ligne du 5, il y a un 1 dans les colonnes C_2 et C_0.

C_2	C_1	C_0	décimal
0	0	1	1 •
0	1	0	2 •
0	1	1	3 • •
1	0	0	4 •
1	0	1	5 • •
1	1	0	6 • •
1	1	1	7 • • •

- C_2 (en vert) colore les bits 4, 5, 6, 7. Ce sont les bits qu'il vérifie.
- C_1 (en bleu) vérifie les bits 2, 3, 6, 7.
- C_0 (en rouge) vérifie les bits 1, 3, 5, 7.
- On constate que chaque bit de données est coloré d'une manière différente. Cela permettra de retrouver la position d'une erreur.

Exemple d'un code de Hamming 7-4

On souhaite envoyer le message 1010. Complétons le mot de Hamming correspondant :

7	**6**	**5**	**4**	**3**	**2**	**1**
1	0	1		0		

C_0 vaut 0 pour pouvoir rendre pair 1+1+0 (les bits d'indices 7, 5, 3).

C_1 vaut 1 pour pouvoir rendre pair 1+0+0 (les bits d'indices 7, 6, 3).

C_2 vaut 0 pour pouvoir rendre pair 1+0+1 (les bits d'indices 7, 6, 5).

7	6	5	4	3	2	1
1	0	1	**0**	0	**1**	**0**

Imaginons que l'on reçoive le mot 0010010 (le bit de poids fort a été altéré).

C_0 a la mauvaise valeur, car 0+1+0+0 est impair, donc il y a une erreur en position 7, 5, 3 ou 1.

C_1 a la mauvaise valeur, car 0+0+0+1 est impair, donc il y a une erreur en position 7, 6, 3 ou 2.

C_2 a la mauvaise valeur, car 0+0+1+0 est impair, donc il y a une erreur en position 7, 6, 5 ou 4.

Écrivons le nombre binaire $C_2C_1C_0$ où C_i vaut 0 si le bit de contrôle C_i a la bonne valeur et 1 sinon. On obtient ici 111, ce qui correspond à 7 en binaire. Le bit erroné est le numéro 7.

Que se passe-t-il si c'est un des bits de contrôle qui est altéré ? Imaginons que l'on ait reçu 1010011 (cette fois-ci, c'est le bit de poids faible qui a été altéré).

C_0 a la mauvaise valeur, car 1+1+0+1 est impair. Il y a une erreur en position 7, 5, 3 ou 1. C_1 a la bonne valeur, car 1+0+0+1 est pair. Il n'y a pas d'erreur en position 7, 6, 3 et 2 C_2 a la bonne valeur, car 1+0+1+0 est pair. Il n'y a pas d'erreur en position 7, 6, 5 et 4. Ici, $C_2C_1C_0$ vaut 001. Le bit erroné est donc le numéro 1.

g. Le codage du son

Nous le savons tous : les ordinateurs sont depuis longtemps capables d'enregistrer et de jouer de la musique. Et indépendamment même des ordinateurs, le son est devenu « digital ». Qu'est-ce que cela signifie ? Que le son est désormais, lui aussi, codé sous forme d'une suite d'informations binaires. Mais pour comprendre comment, il faut comprendre ce qu'est un son, et comment il était enregistré et restitué dans les appareils traditionnels.

Le son est une onde qui fait vibrer l'air transmettant son énergie de molécules en molécules. Autrement dit, le son est une sensation auditive provoquée par une vibration de l'air, c'est-à-dire une suite de surpressions et de dépressions de l'air par rapport à une moyenne qui est la pression atmosphérique. Plus une onde sonore vibre rapidement plus le son est aigu.

L'arrivée de l'informatique et du stockage d'information sous forme numérique a entraîné une véritable révolution dans le domaine musical. Cette révolution a commencé avec le CD audio, puis avec la compression des fichiers audios, puis les lecteurs dits MP3 et continue de nos jours avec l'intégration de la composante numérique dans le monde de la Hi-Fi et dans les lecteurs multimédias.

Il y a pour le grand public plusieurs sources possibles pour obtenir de la musique sous forme numérique.

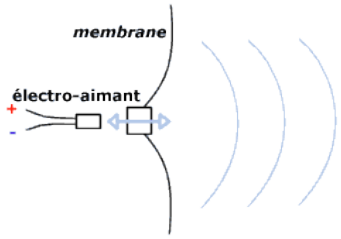
1. Sources analogiques
    1. Disque microsillon
    2. Cassette audio
    3. Radio
2. Sources numériques
    1. CD audio et ses dérivés
    2. Magasin de musique en ligne
    3. webradio et Diffusion audionumérique
    4. streaming

Le son existe grâce à trois éléments :

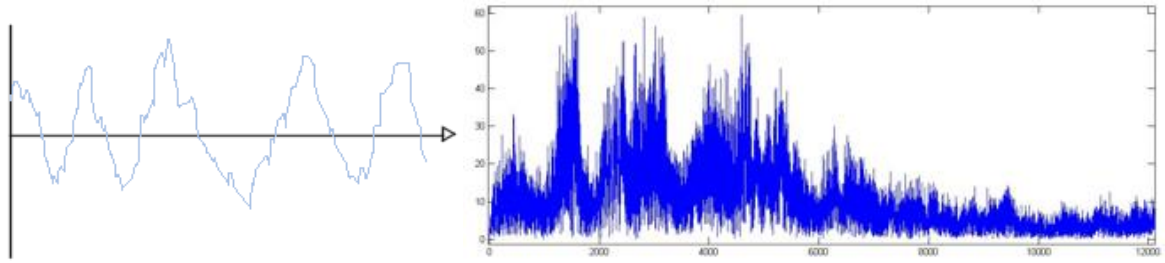
- la source qui produit le bruit : vibration d'un corps solide, liquide ou gazeux (membrane d'un haut-parleur, corde vocale, corde d'un instrument de musique, etc.),
- le milieu qui transmet le son : propagation d'ondes sonores semblables aux vibrations de la source,
- le récepteur : l'oreille qui le récepteur de l'ouïe perçoit des vibrations entre 16 Hz et 20 000 Hz (1Hz = 1 oscillation par seconde).

La façon la plus simple de reproduire un son actuellement est de faire vibrer un objet. De cette façon un violon émet un son lorsque l'archet fait vibrer ses cordes, un piano émet une note lorsque l'on frappe une touche, car un marteau vient frapper une corde et la fait vibrer.

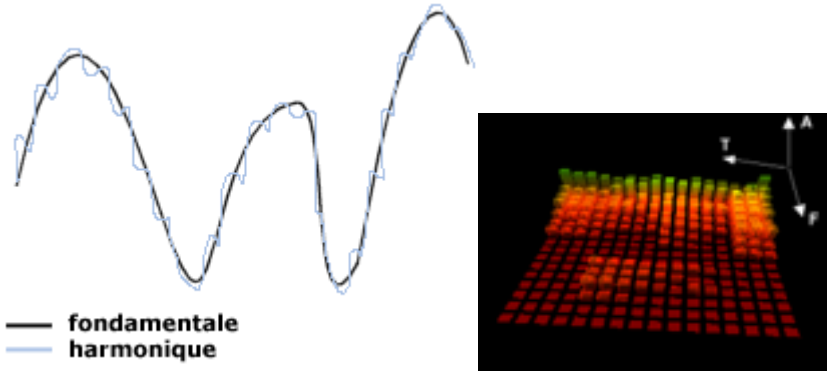
Pour reproduire des sons, on utilise généralement des haut-parleurs. Il s'agit en fait d'une membrane reliée à un électroaimant, qui, suivant les sollicitations d'un courant électrique va aller en avant et en arrière très rapidement, ce qui provoque une vibration de l'air situé devant lui, c'est-à-dire du son !



De cette façon on produit des ondes sonores qui peuvent être représentées sur un graphique comme les variations de la pression de l'air (ou bien de l'électricité dans l'électroaimant) en fonction du temps. On obtient alors une représentation de la forme suivante :



Cette représentation d'un son est appelée **spectre de modulation d'amplitude** (modulation de l'amplitude d'un son en fonction du temps). Le **sonogramme** représente par contre la variation des fréquences sonores en fonction du temps. On peut remarquer qu'un sonagramme présente une fréquence fondamentale, à laquelle se superposent des fréquences plus élevées, appelées harmoniques.



C'est ce qui permet d'arriver à distinguer plusieurs sources sonores : les sons graves auront des fréquences basses, et les sons aigus des fréquences élevées.

Le passage d'un signal analogique à un signal numérique se fait via une **conversion analogique - numérique**, par **échantillonnage** (prises des valeurs du signal analogique à intervalles de temps constants) et par **quantification** de chacune des valeurs échantillonnée

Echantillonnage du son

Pour pouvoir représenter un son sur un ordinateur, il faut arriver à le convertir en valeurs numériques, car celui-ci ne sait travailler que sur ce type de valeurs. Il s'agit donc de relever des petits échantillons de son (ce qui revient à relever des différences de pression) à des intervalles de temps précis. On appelle cette action **l'échantillonnage** ou la **numérisation du son**. L'intervalle de temps entre deux échantillons est appelé taux d'échantillonnage. Etant donné que pour arriver à restituer un son qui semble continu à l'oreille il faut des échantillons tous les quelques 100 000èmes de seconde, il est plus pratique de raisonner sur le nombre d'échantillons par seconde, exprimés en **Hertz** (*Hz*). Voici quelques exemples de taux d'échantillonnage et de qualités de son associées :

Taux d'échantillonnage	Qualité du son
44 100 Hz	qualité CD
22 000 Hz	qualité radio

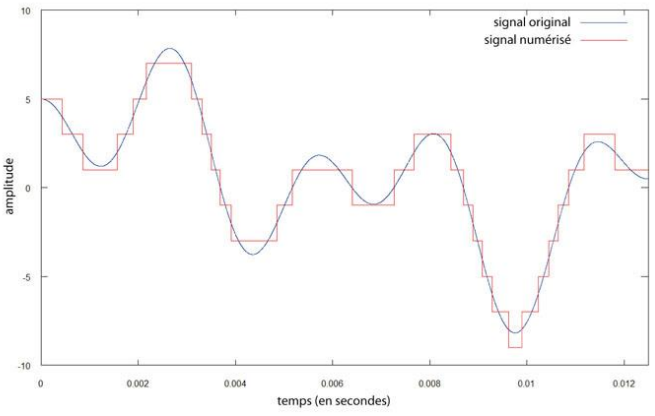
8 000 Hz	qualité téléphone

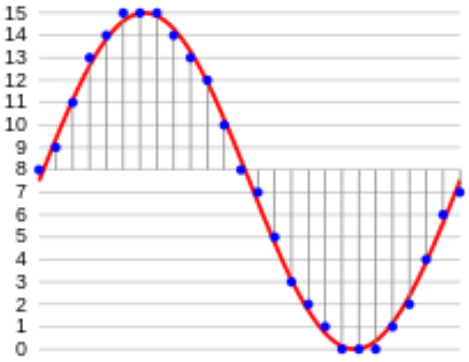
La valeur du taux d'échantillonnage, pour un CD audio par exemple, n'est pas arbitraire, elle découle en réalité du théorème de Shannon. La fréquence d'échantillonnage doit être suffisamment grande, afin de préserver la forme du signal. Le théorème de Nyquist - Shannon stipule que la fréquence d'échantillonnage doit être égale ou supérieure au double de la fréquence maximale contenue dans ce signal. Notre oreille perçoit les sons environ jusqu'à 20 000 Hz, il faut donc une fréquence d'échantillonnage au moins de l'ordre de 40 000 Hz pour obtenir une qualité satisfaisante. Il existe un certain nombre de fréquences d'échantillonnage normalisées :

- **32 kHz** : pour la radio FM en numérique (bande passante limitée à 15 kHz)
- **44.1 kHz** : pour l'audio professionnelle et les compact-disques
- **48 kHz** : pour les enregistreurs numériques multipistes professionnels et l'enregistrement grand public (DAT, MiniDisc...)

Représentation informatique du son

A chaque échantillon (correspondant à un intervalle de temps) est associée une valeur qui détermine la valeur de la pression de l'air à ce moment, le son n'est donc plus représenté comme une courbe continue présentant des variations mais comme une suite de valeurs pour chaque intervalle de temps :





L'ordinateur travaille avec des bits, il faut donc déterminer le nombre de valeurs que l'échantillon peut prendre, cela revient à fixer le nombre de bits sur lequel on code les valeurs des échantillons.

- Avec un codage sur 8 bits, on a 2^8 possibilités de valeurs, c'est-à-dire 256 valeurs possibles
- Avec un codage sur 16 bits, on a 2^{16} possibilités de valeurs, c'est-à-dire 65536 valeurs possibles

Avec la seconde représentation, on aura bien évidemment une qualité de son bien meilleure, mais aussi un besoin en mémoire beaucoup plus important.

Enfin, la stéréophonie nécessite deux canaux sur lesquels on enregistre individuellement un son qui sera fourni au haut-parleur de gauche, ainsi qu'un son qui sera diffusé sur celui de droite.

Un son est donc représenté (informatiquement) par plusieurs paramètres :

- la fréquence d'échantillonnage
- le nombre de bits d'un échantillon
- le nombre de voies (une seule correspond à du mono, deux à de la stéréo, et quatre à de la quadriphonie)

Mémoire requise pour stocker un son

Il est simple de calculer la taille d'une séquence sonore non compressée. En effet, en connaissant le nombre de bits sur lequel est codé un échantillon, on connaît la taille de celui-ci (la taille d'un échantillon est le nombre de bits...).

Pour connaître la taille d'une voie, il suffit de connaître le taux d'échantillonnage, qui va nous permettre de savoir le nombre d'échantillons par seconde, donc la taille qu'occupe une seconde de musique. Celle-ci vaut : Taux d'échantillonnage **x** Nombre de bits

Ainsi, pour savoir l'espace mémoire que consomme un extrait sonore de plusieurs secondes, il suffit de multiplier la valeur précédente par le nombre de secondes :
Taux d'échantillonnage **x** Nombre de bits **x** Nombre de secondes

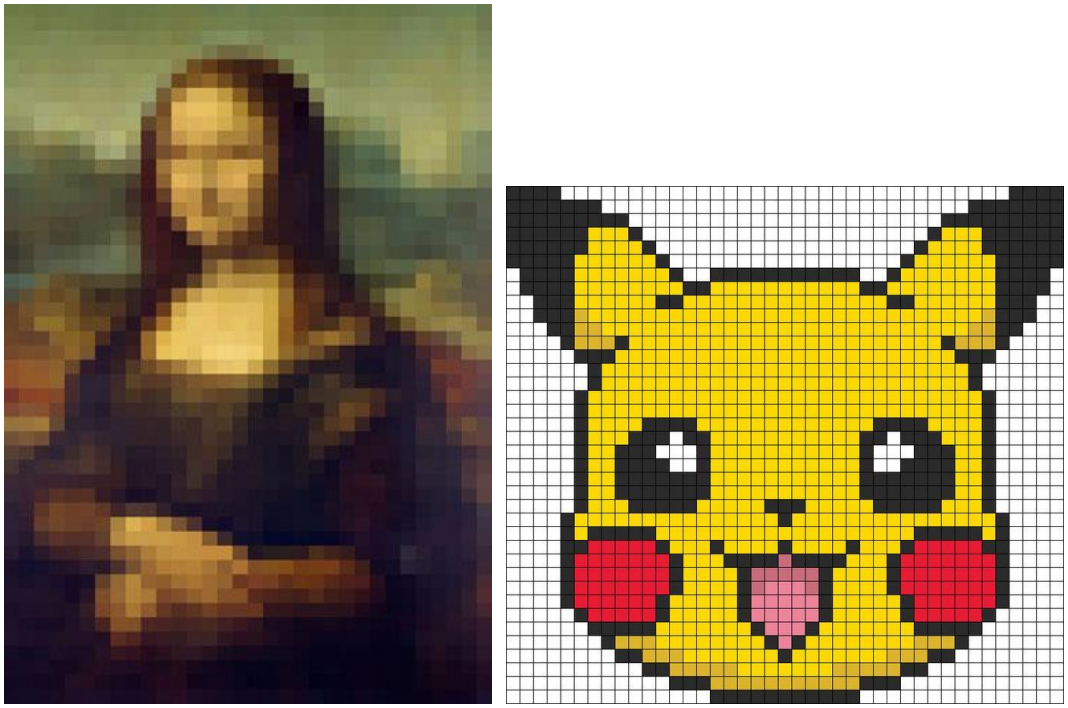
Enfin, la taille finale de l'extrait est à multiplier par le nombre de voies (elle sera alors deux fois plus importante en stéréo qu'en mono...).

La taille en bits d'un extrait sonore est ainsi égale à :

Taux d'échantillonnage x **Nombre de bits** x **Nombre de secondes** x **Nombre de voies**

h. Le codage de l'image

Le principe de la numérisation de l'image ressemble donc à celui du son : on va découper l'image en petits carrés appelés pixels, et on va assigner à chacun de ces carrés une couleur parmi un nombre limité. Plus les carrés seront denses, et donc petits, et plus le nombre de couleurs sera élevé, plus l'œil humain finira par voir des courbes là où il n'y a que des marches d'escalier, et des dégradés de couleurs là où il n'y a qu'une juxtaposition. Si le découpage est trop grossier, ou les couleurs trop peu nombreuses, comme c'était le cas dans les débuts de l'informatique, l'image était manifestement dégradée, et on parle dans ce cas de pixellisation.



Images pixellisées

La qualité de l'image va donc dépendre :

- Du nombre de pixel contenu dans l'image.

- Du nombre de bits utilisé pour coder la couleur de chaque pixel.

Nb de bits	Nb de couleurs possibles
1	2
4	16
8	256
16	65 536
24	16 777 216
32	4 194 967 296

- **La définition** d'une image est le nombre de pixels qui la constitue. On peut aussi indiquer la résolution d'une image en donnant le nombre de pixel sur la largeur et le nombre de pixel sur la hauteur (exemple : les téléviseurs fullHD ont une définition de 1920x1080 c'est-à-dire 2 073 600 pixels)

- **La résolution** est le lien entre le nombre de pixels de l'image et sa taille. Elle s'exprime en point par pouce (dot per inch→dpi) ou en pixel par pouce (pixel per inch→ppi).

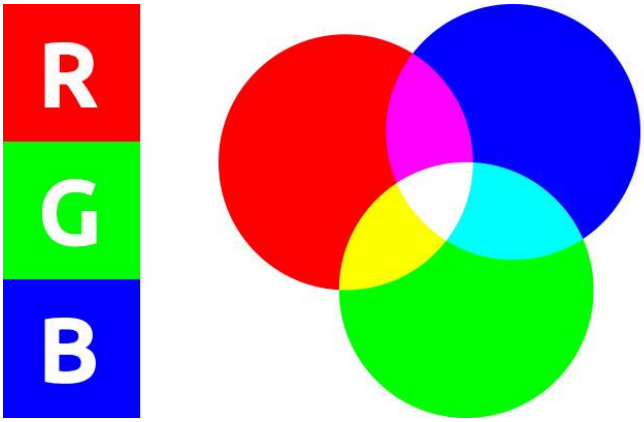
Remarque : 1pouce=2.54cm

Exemple :

Le codage RVB (ou RGB) 24bits :

On peut obtenir n'importe quelle couleur en utilisant une combinaison des couleurs rouges vertes et bleu.

Le code RVB va coder le niveau d'intensité de chacune de ces couleurs sur un octet (256 niveau d'intensité par couleur). Le premier octet représentant le niveau de rouge, le deuxième octet le niveau de vert et le troisième octet le niveau de bleu.



Quelques couleurs et leur codage RVB :

rouge	255.0.0
magenta	255.0.255
noir	0.0.0
marron	88.41.0

vert	0.255.0
cyan	0.255.255
blanc	255.255.255
orange	237.127.16

bleu	0.0.255
jaune	255.255.0
gris	128.128.128
violet	102.0.153

- **Le codage de la vidéo :**

Une vidéo étant une succession d'images, il suffit de coder toutes les images d'une vidéo les une à la suite des autres.

Remarque : pour qu'une vidéo soit fluide il faut qu'elle soit constituée d'au moins 24 images par seconde.

i. Compression numérique

- **Présentation :**

Nous avons vu que les principaux intérêts d'une information numérique par rapport à une information analogique était :

- Copie, stockage et transmission de l'information sans perte de qualité.
- Possibilité d'être exploitée et traitée par un système informatique.

Parmi les traitements que l'on peut effectuer sur un signal numérique, il y en a un très intéressant qui est la compression.

La compression consiste diminuer le nombre d'octets sur lequel l'information est codée.

La compression peut se faire sans perte de qualité (compression non destructive ou lossless) ou avec perte de qualité (compression destructive ou lossy).

Avantage :

- On a besoin de moins de bande passante pour transmettre le signal (dans un même laps de temps)
- On utilise moins d'espace mémoire pour son stockage.

Inconvénient :

- Les données nécessitent un traitement supplémentaire avant de pouvoir être exploitées (utilisation d'un codec).
- Beaucoup de compression se font avec perte de qualité du signal d'origine (compression destructive).

- **RLE (Run Length Encoding) :**

C'est le principe de compression le plus simple, il consiste à repérer dans un fichier une suite consécutive d'une même donnée et de remplacer cette suite par la donnée répétée avec son nombre de répétition (identifié par un caractère spécial).

Dans le cas d'une image ou d'une vidéo très souvent nous avons des suites d'informations qui se répètent.

Exemple :

Si on grossit le 2 premières lignes de l'image de droite, on obtient :

Si on part du bord haut gauche de l'image nous avons :

- 34 pixels noirs consécutifs (32 sur la première ligne+ 2 sur la deuxième ligne)

- Puis 4 pixels rouges consécutifs.

- Puis 1 pixel noir.

- Puis 6 pixels rouges consécutifs…

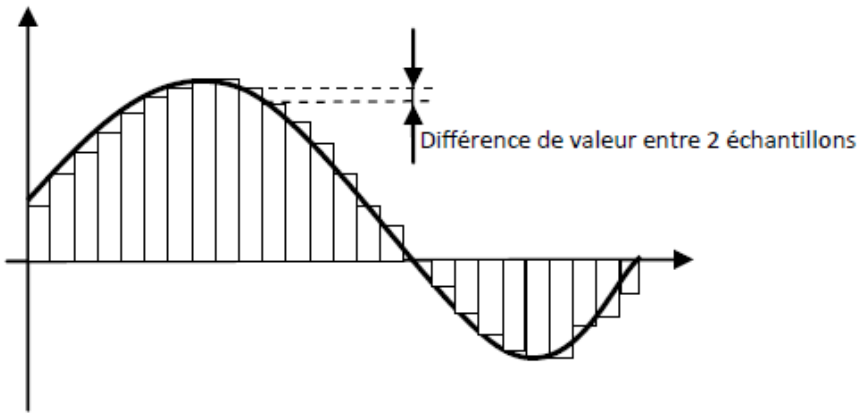
Au lieu d'écrire 34 fois d'affilé le code correspondant à la couleur noire, il suffit de l'écrire une seule fois en indiquant le nombre de fois qu'il se répète (même remarque pur le 4 pixels rouges qui suivent).

- **Huffman :**

Le code Huffman consiste à coder des éléments (caractères, pixels..) sur un nombre de bits variable. Les éléments qui apparaissent le plus souvent sont codés sur un petit nombre de bits alors que les éléments qui apparaissent plus rarement sont codés sur un plus grand nombre de bits.

- **Le codage différentiel :**

En général, 2 échantillons ou 2pixels qui se suivent ont des valeurs relativement proches. Il vaut mieux alors coder la différence de valeurs entres les 2 échantillons plutôt que la valeur de l'échantillon.


Différence de valeur entre 2 échantillons