Chapitre III: Représentation de l'information
(Numération et codage de l'information)

1. Introduction
La structure interne de l'ordinateur est réalisée à partir de composants électroniques élémentaires. Chacun de ces composants ne peut prendre que deux états différents ; Il est sous tension ou hors tension (chargé ou non chargé, ouvert ou fermé, vrai ou faux, etc.)[1]. Par convention, la désignation de ces deux états est 1 pour le premier et 0 pour le deuxième. Cette désignation est à la base du Digit Binaire (*Binary digIT*) abrégé en bit.

L'information à représenter et à traiter par l'ordinateur étant sous plusieurs formes (nombres, textes, images, sons, vidéos, programmes, etc.) ; il fallait trouver un moyen de faire correspondre ces informations avec des formes en fonction du Digit Binaire ; c'est le rôle du codage.

2. Codage de l'information
2.1. Définition
En informatique, le codage de l'information (système de codage) permet d'établir une bijection ou une correspondance, non ambiguë, entre une représentation externe d'une information et une autre représentation interne (forme binaire ou code binaire) de la même information, suivant un ensemble de règles précises.

Exemple 1:

20 est la représentation externe, comme nous le connaissons, du nombre vingt ;
La représentation interne de 20, supportée par la machine, est une suite de 0 et de 1 : **10100**.

Exemple 2:

L'exemple suivant établie une bijection entre les dix chiffres décimaux et leurs représentations binaires sur quatre bits :

Chiffres décimaux		Codes binaires
0		0000
1		0001
2		0010
3	Système de codage →	0011
4		0100
5		0101
6		0110
7		0111
8		1000
9		1001

2.2. Etapes de codage de l'information
De la représentation externe de l'information, que nous connaissons, trois étapes essentielles doivent être accomplies pour obtenir la représentation interne. Ces trois étapes sont:

1. **Numérisation** : l'information est exprimée par une suite de nombres;
2. **Codage** : chaque nombre est codé sous une forme binaire (suite de 0 et 1) ;
3. **Représentation physique** : chaque élément binaire est représenté par un état physique.

2.3. Capacité de représentation
Lors d'un codage la capacité de représentation d'un ensemble (ou combinaison) de **n bits** est le nombre d'informations que cet ensemble peut coder.

Exemple:

Avec **un bit** nous pouvons représenter deux états: 0 et 1 ;
L'association de **deux bits** permet de représenter quatre états (2^2): 00, 01, 10 et 11 ;
De même qu'il possible de représenter 8 états différents (2^3) avec **trois bits** : 000, 001, 010, 011, 100, 101, 110 et 111 ;
Ainsi un ensemble de **n bits** peut permettre de représenter 2^n états différents.

[1] Selon la technologie sous laquelle il est fabriqué.

3. Systemes de numeration et representation des nombres

3.1. Notion de système de numération

Un système de numération se caractérise par une base et un nombre de symboles (chiffres et lettres) qu'il utilise pour exprimer des grandeurs.

Un système de numération à base B utilise les symboles compris entre 0 et B-1, il nécessite donc B symboles pour le représenter.

La base d'un système de numération est égale au nombre d'unité de rang n nécessaires pour constituer une unité de rang n+1.

Exemple :

Dans le système à base 10 ou système décimal, il faut :

- 10 unités pour constituer une dizaine ;
- 10 dizaines pour constituer une centaine ; etc.

3.2. Rang et Poids du symbole

Dans un système de numération l'importance d'un symbole varie en fonction de son rang (position) dans le nombre à partir de la droite et en commençant par 0. La base élevée au rang d'un symbole s'appelle Poids.

Exemple :

Soit le nombre 125 dans le système décimal :

1	**2**	**5**	← Symbole
2	1	0	← Rang
10^2	10^1	10^0	← Poids
100	20	5	← Valeur du symbole

3.3. Système décimal

C'est le système le plus connu. Dans ce système les symboles de représentation sont les chiffres de 0 à 9 (10 symboles) et la base est 10.

Exemple:

Soit le nombre 256 exprimé dans le système décimal. Ce nombre peut être exprimé aussi avec la décomposition 200 + 50 + 6, ou bien encore :

$$(2 * 10^2) + (5 * 10^1) + (6 * 10^0)$$

Base Rang Poids

Cette représentation nous permet de dire que le chiffre 2, par exemple, a un poids plus important que le chiffre 5. En effet, le chiffre 2 représente les centaines alors que le chiffre 5 représente les dizaines.

Règle générale

Pour tout nombre exprimé dans un système de base B, la valeur d'un symbole X de rang n, dans ce nombre, est égale à : $\mathbf{X} * B^n$.

3.4. Système binaire

Le système binaire est le système à base 2. Il utilise donc deux chiffres, 0 et 1, pour représenter un nombre quelconque. Un nombre binaire se présente donc sous la forme d'une combinaison de 0 et 1.

Exemple :

Soit le nombre binaire : 1101

Pour calculer la valeur de ce nombre dans le système décimal, nous procéderons comme suit :

$1101 = (1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 8 + 4 + 0 + 1 = 13.$

3.5. Système octal

Le système octal est le système à base 8. Il utilise les chiffres 0,1,2,3,4,5,6,7 pour représenter un nombre quelconque.

Exemple :

Soit le nombre octal : 123

Pour calculer la valeur de ce nombre dans le système décimal, nous procéderons comme suit :

$123 = (1 * 8^2) + (2 * 8^1) + (3 * 8^0) = 64 + 16 + 3 = 83.$

3.6. Système hexadécimal

Ce système est à base 16 ce qui veut dire qu'il utilise 16 symboles pour la représentation des nombres, ces symboles sont les chiffres décimaux (0..9) et les lettres de A à F.
10 dans le système décimal est représenté par A dans le système hexadécimal, 11 par B, 12 par C, 13 par D, 14 par E et 15 par F.

Exemple :

Soit le nombre hexadécimal : 2CB

Pour calculer la valeur de ce nombre dans le système décimal, nous procéderons comme suit :

$2CB = (2 * 16^2) + (C * 16^1) + (B * 16^0) = 2*16*16 + 12*16 + 11 = 715.$

3.7. Transcodage ou conversion de base

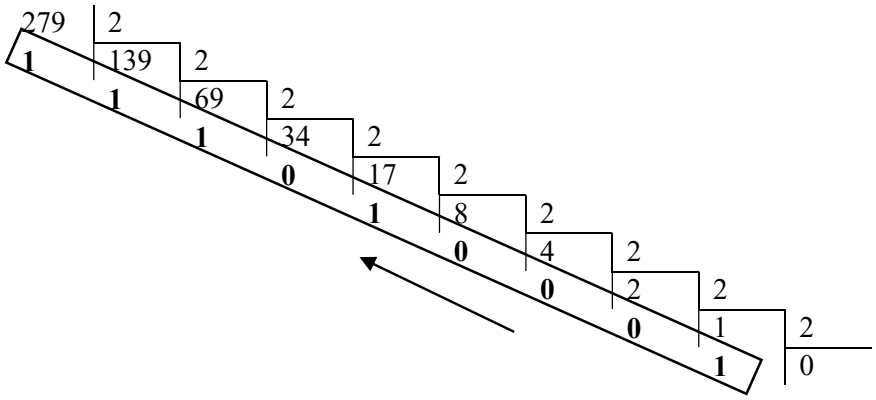
Le transcodage est l'opération qui permet de déterminer, pour tout nombre exprimé dans un système de numération de base quelconque, son équivalent dans un système de numération de base différente.

A. Du décimal au binaire

Le nombre binaire équivalent au nombre décimal à convertir, sera constitue par l'ensemble des restes trouvés en appliquant la méthode des divisions successives par 2 du nombre décimal à convertir jusqu'à ce que le quotient soit nul. Le dernier entre ces restes représente le chiffre binaire de rang le plus élevé.

Exemple :

Soit à trouver l'équivalent binaire du nombre décimal 279 :



Donc l'équivalent du nombre 279 en binaire est : **100010111** et se lit : $(279)_{10} = (100010111)_2$

B. Du binaire au décimal

Pour retrouver l'équivalant décimal d'un nombre binaire, il suffit de calculer la somme des poids binaires de tous les chiffres 1 du nombre binaire.

Exemple :

Soit à calculer l'équivalent décimal du nombre binaire: **1011**

1	**0**	**1**	**1**
3	2	1	0

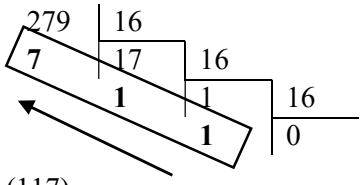
L'équivalent se calcule comme suit : $(1 * 2^0) + (1 * 2^1) + (1 * 2^3) = 1 + 2 + 8 = 11$, donc $(1011)_2 = (11)_{10}$

C. Du décimal à l'hexadécimal

Pour déterminer l'équivalent hexadécimal d'un nombre décimal, on utilise la même méthode que celle qui permet de passer du décimal au binaire, c.à.d. la méthode de la division successive et on changeant la base de 2 à 16.

Exemple 1:

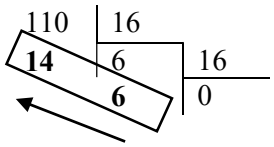
Déterminant l'équivalent hexadécimal du nombre décimal 279 :



D'où : $(279)_{10}=(117)_{16}$

Exemple 2:

Déterminant l'équivalent hexadécimal du nombre décimal 110 :



L'équivalent de 14 en hexadécimal est E d'où : $(110)_{10}=(6E)_{16}$

D. De l'hexadécimal au décimal

La méthode est la même que celle utilisée pour passer du binaire au décimal.

Exemple 1:

Calculant l'équivalent décimal du nombre hexadécimal :117

$(117)_{16}=(1*16^2) + (1*16^1) + (7*16^0) = 256 + 16 + 7 = 279$, donc : $(117)_{16}=(279)_{10}$

Exemple 2:

Calculant l'équivalent décimal du nombre hexadécimal :1DA

$(1DA)_{16}=(1*16^2) + (D*16^1) + (A*16^0) = (1*256) + (13*16) + (10*1) = 474$, donc : $(1DA)_{16}=(474)_{10}$

E. Du binaire à l'hexadécimal

La méthode de conversion consiste à découper, de droite à gauche, le nombre binaire en tranches de quatre chiffres, puis faire la correspondance de chaque groupe avec symbole hexadécimal.

Exemple :

Soit le nombre binaire: 00111101 en découpant ce nombre en groupes de quatre bits, on aura : 0011 1101 ce qui permet de faire la correspondance en hexadécimal :

0011	1101
3	D

D'où : $(00111101)_2 = (3D)_{16}$

F. De l'hexadécimal au binaire

Le passage de l'hexadécimal au binaire est effectuer en remplaçant chaque symbole de la représentation hexadécimale par sa valeur en binaire.

Exemple :

Soit le nombre hexadécimal 5C3A, son équivalent en binaire est établit comme suit :

5	C	3	A
0101	1100	0011	1010

D'où : $(5C3A)_{16} = (0101110000111010)_2$

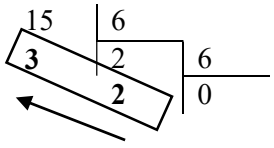
G. Du décimal à un système à base B

Pour réaliser cette conversion, nous devons accomplir les étapes suivantes :

- Diviser par B le nombre décimal à convertir, puis les quotients successifs obtenus, jusqu'à obtention d'un quotient nul,
- Le nombre recherché sera constitué par le regroupement de tous les restes obtenus, le premier de ces restes occupant le rang 0 et le dernier reste trouvé aura le rang le plus élevé.

Exemple :

L'équivalent du nombre décimal 15 dans le système à base 6 est calculé comme suit:



Donc : $(15)_{10}=(23)_6$

H. D'un système à base B au décimal

Pour convertir un nombre quelconque, d'un système à base B vers le système décimal, il faut suivre les deux étapes

suivantes :

- Multiplier chacun des symboles (chiffres) qui composent le nombre à convertir, par son poids ;
- Effectuer la somme des résultats obtenus.

Exemple :

Conversion du nombre 1142 de la base 6 vers la base 10 :

$(1142)_6=(1 * 6^3) + (1*6^2) + (4*6^1) + (2*6^0) = 216 + 36 + 24 + 2 = 278$ → $(1142)_6=(278)_{10}$

3.8. Opérations arithmétiques binaires

Les opérations arithmétiques usuelles (addition, soustraction, multiplication et division) dans les systèmes binaire, octal et hexadécimal se font selon les mêmes principes que ceux du système décimal. Dans ce qui suit nous allons voir comment se font ces opérations dans le système binaire[1].

A. Nombres entiers non signés

• Addition

L'addition des nombres binaires se fait en respectant les règles suivantes:

- 0 + 0 = 0
- 0 + 1 = 1 + 0 = 1
- 1 + 1 = 10, on écrit "0" et on **retient** (ou on reporte)[2] "1" sur le bit de rang supérieur (rang de gauche)
- 1 + 1 + 1 = 11, on écrit "1" et on retient "1" sur le bit de rang supérieur

Exemple:

Soit à additionner les nombres 101 et 111, on procède alors comme suit:

```
    1 1        Retenues
    1 0 1
+   1 1 1
= 1 1 0 0
```

Remarque:

S'il y a une retenue après l'addition des bits du rang le plus fort on dit qu'il y a un dépassement de capacité (débordement, *overflow*).

Dans l'exemple précédant il y a une retenue sur l'addition effectuée sur les bits du troisième rang, donc il y a un débordement.

• Soustraction

De même que pour l'addition, la soustraction se fait en respectant les règles suivantes:

- 0 - 0 = 0
- 1 - 0 = 1
- 0 - 1 = 1, on écrit "1" et on retient "1" sur le bit de rang supérieur du diminuteur (rang de gauche)
- 1 - 1 = 0

Exemple:

L'exemple suivant effectue la soustraction entre deux nombres:

```
  1 0 0 1 0 1 1 0
- 1 1   1         Retenues
  0 1 1 0 1 1 0 0
= 0 0 1 0 1 0 1 0
```

• Multiplication

La multiplication binaire s'effectue selon le principe des multiplications décimales, c-à-d. qu'on multiplie le multiplicande par chacun des bits du multiplicateur. On décale les résultats intermédiaires obtenus et on effectue ensuite l'addition de ses résultats partiels.

Exemple :

L'exemple suivant effectue une multiplication des nombres $(1001)_2$ et $(1010)_2$:

[1] Etant donné que la quasi-totalité des opérations internes sur les ordinateurs se font sur ce système.

[2] Dans certains ouvrages une distinction d'appellation est faite entre la retenue en addition (**report**, *Carry*) et la retenue en soustraction (retenue, *Borrow*)

```
    1011
*   1010
-------
    0000
   1011.
  0000.
1011.
-------
1101110
```

• Division

Nous avons vu que la multiplication est basée sur une succession d'additions, inversement la division est basée sur une succession de soustractions et s'emploi de la même façon qu'une division décimale ordinaire.

Exemple :

L'exemple suivant effectue la division du nombre $(1111010)_2$ sur le nombre $(1011)_2$:

```
  1 1 1 1 0 1 0 | 1 0 1 1
- 1 0 1 1       | -------
  ---------     | 1 0 1 1
    1 0 0 0
-   0 0 0 0
  ---------
    1 0 0 0 1
-     1 0 1 1
  -----------
        1 1 0 0
-       1 0 1 1
  -------------
      reste : 1
```

B. Nombres entiers signés (ou relatifs)

Avec les nombres signés le fonctionnement des opérations arithmétiques déjà vues est étroitement lié à la technique avec laquelle ces nombres sont représentés. Généralement trois techniques sont proposées:

• Première technique

Le bit du rang le plus fort (MSB, Most Significant Bit) est utilisé pour coder le signe : 0 pour un entier positif, 1 pour un entier négatif. Le reste des bits, c-à-d les n-1 bits de rang faible (LSB, Less Significant Bit), est utilisé pour coder la valeur absolue du nombre.

Exemple :

Sur un octet le nombre décimal 66 est représenté comme suit :

$(+66)_{10} = (01000010)_2$ $(-66)_{10} = (11000010)_2$

Inconvénients

Le nombre décimal 0 a deux représentations différentes +0 $(00000000)_2$ et -0 $(10000000)_2$.

L'addition classique ne fonctionne plus car si nous prenons l'exemple de l'addition des nombres $(+3)_{10} + (-4)_{10}$ nous aurons pour résultat $(00000011)_2 + (10000100)_2 = (10000111)_2 = (-7)_{10}$.

• Deuxième technique, représentation en complément à un ou CA1

Un entier positif se représente avec le bit de poids fort à 0 et le reste des bits de poids faibles est utilisé pour coder la valeur absolue de ce nombre. Un entier négatif se représente en inversant les bits de l'entier positif correspondant.

Exemple:

$(+5)_{10} = (0101)_2$ et $(-5)_{10} = (1010)_2$

Inconvénients

On a toujours deux représentations pour le 0 décimal : $(00000000)_2$ et $(11111111)_2$.

L'addition classique[1] fonctionne partiellement (pas évidente), par exemple l'addition de $(+4)_{10}$ et $(+6)_{10}$ donne $(0100)_2 + (0110)_2 = (1010)_2$ qui égale au complément à un de $(-5)_{10}$.

La retenue joue un rôle important dans la détermination du résultat final de l'addition[2].

• Troisième technique, représentation en complément à deux ou CA2

La représentation en complément à deux est la même que celle en complément à un sauf que pour les entiers négatifs on rajoute un $(+1)_2$ à cette représentation.

[1] En complément à un l'addition se fait selon des règles précises en fonction des signes des nombres à additionner.

[2] Par exemple s'il y a une retenue dans une addition de deux nombres de signes opposés, le résultat est un nombre positif et on ajoute la retenue pour avoir sa valeur.

Exemple:
Pour le nombre $(+5)_{10}$ la représentation en complément à un est égale à $(0101)_2$ et elle est la même en complément à deux. Pour le nombre $(-5)_{10}$ la représentation en complément à un est égale à $(1010)_2$. La représentation à deux est égale à : $(1010)_2+(0001)_2=(1011)_2$

Avantages
Le 0 décimal possède une seule représentation.
L'addition en représentation complément à deux revient à une simple addition binaire, on ne garde jamais la retenue (c-à-d lecture directe du résultat en complément à deux).

- **Conclusion sur la représentation des nombres entiers**

La représentation des nombres entiers est limitée par la taille du mot mémoire (ou nombre de bits manipulé au même temps) qui leur est affectée. La représentation la plus souvent utilisée est celle du complément à deux qui présente plusieurs avantages par rapport aux autres représentations.
Dans tous les cas il y aura toujours des opérations, dont le résultat n'est pas représentable, qui peuvent engendrer des résultats aberrants ou empêcher la poursuite des calculs.

C. Nombres réels
Pour représenter les nombres réels deux formes de codage sont utilisées : la représentation avec virgule fixe et la représentation avec virgule flottante.

- **Première forme, la représentation avec virgule fixe**

Cette représentation a été utilisée sur les premières machines, elle reflète la forme de représentation des nombres réels dans le système décimal que nous connaissons. La représentation avec virgule fixe possède une partie entière et une partie décimale séparées par une virgule. La position de la virgule est fixe d'où le nom de cette représentation.

▪ **Représentation:**
Un nombre réel X dans le système décimal est représenté, sous forme virgule fixe, sur n bits en binaire comme suit : $(X)_{10} = (a_{k-1}\, a_{k-2} \ldots a_1\, a_0, a_{-1}\, a_{-2} \ldots a_{-p})_2$
Où :
n= k+p,
a_{k-1} est le chiffre binaire de poids fort (MSB) et a_{-p} est le chiffre binaire de poids faible (LSB),
k est le nombre de chiffres binaires avant la virgule et p est le nombre de chiffres binaires après la virgule (précision).

Exemple :
Sur 8 bits le nombre réel $(3,125)_{10}$ est représenté par 0011,0010 en binaire avec k=4 et p = 4.

▪ **Conversion du décimale en binaire:**
La conversion d'un nombre réel décimal en nombre binaire se fait en suivant les deux étapes suivantes :
- La partie entière est codée sur k bits (division successive par 2),
- La partie fractionnaire ou décimale est codée sur p bits en la multipliant par 2 successivement jusqu'à ce que la partie fractionnaire soit nulle ou le nombre de bits p (précision) est atteint.

Exemple :
1. Convertissons le nombre réel $(3,347)_{10}$ en binaire sur 8 bits avec une précision de 5 bits (p=5):
   - La partie entière $(3)_{10}$ se transforme en binaire en $(11)_2$
   - La partie fractionnaire s'obtient selon le calcul 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 = 0,0**1**0...
0,766*2 = **1**,552	0,347 = 0,0**1**01...
0,552*2 = **1**,104	0,347 = 0,0**1**011...
0,104*2 = **0**,208	0,347 = 0,0**1**0110...
0,208*2 = **0**,416	0,347 = 0,0**1**01100...
0,416*2 = **0**,832	0,347 = 0,0**1**011000...
0,832*2 = **1**,664	0,347 = 0,0**1**0110001...
0,664*2 = **1**,328	0,347 = 0,0**1**01100011...
0,328*2 = **0**,656	0,347 = 0,0**1**011000110...

Atteinte de la précision désirée

Ici on arrête le calcul comme indiqué (atteinte de la précision désirée) et on obtient le nombre $(011,01011)_2$.

2. Convertissons le nombre réel (3,25) en binaire sur 8 bits avec une précision de 5 bits (p=5):

- La partie entière $(3)_{10}$ se transforme en binaire en $(11)_2$
- La partie fractionnaire s'obtient selon le calcul suivant :

0,25*2=**0**,50 0,25=0,**0**…
0,50*2=**1**,00 0,50=0,0**1** Partie fractionnaire égale à 0

Ici on arrête le calcul comme indiqué (Partie fractionnaire égale à 0) et on obtient le nombre $(011,01000)_2$.

▪ **Conversion du binaire en décimale:**

La transformation d'un nombre réel binaire vers la base décimale suit la règle générale $\sum_{i=-p}^{k-1} a_i b^i$ avec b=2.

Exemple :

$$(110,101)_2 = 1*2^2+1*2^1+0*2^0+1*2^{-1}+0*2^{-2}+1*2^{-3}=4+2+0+0,5+0,125=(6,625)_{10}$$

• **Deuxième forme, la représentation avec virgule flottante**

C'est la représentation la plus utilisée actuellement sur les ordinateurs.

▪ **Représentation:**

Le codage d'un nombre réel décimal en binaire, format virgule flottante, revient à coder un signe, une mantisse et un exposant représentés comme suit : $(X)_{10} = (\pm M*2^E)_2$

Où M est la mantisse (qui peut être en virgule fixe) et E est l'exposant (qui est signé).

Exemple :

Soit à coder en binaire, format virgule flottante, le nombre réel $(3,25)_{10}$, plusieurs représentations peuvent être données :

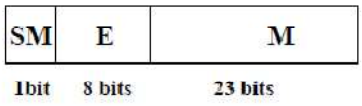
$(3,25)_{10} = (\mathbf{1,101} * 2^{\mathbf{1}})_2$, **Exposant**
$(3,25)_{10} = (\mathbf{110,1} * 2^{\mathbf{-1}})_2$, etc.

Mantisse

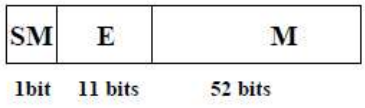
▪ **Normalisation:**

La représentation en virgule flottante décrite plus haut peut engendrer plusieurs valeurs de mantisses et d'exposants du même nombre réel, d'où la nécessité d'une normalisation[1]. Le standard de la normalisation le plus connu est celui de la norme IEEE 754[2] :

Simple précision sur 32 bits :
1 bit de signe de la mantisse
8 bits pour l'exposant
23 bits pour la mantisse



Double précision sur 64 bits :
1 bit de signe de la mantisse
11 bits pour l'exposant
52 bits pour la mantisse



3.9. Codage des caractères

Nous avons vu plus haut comment coder et représenter les nombres en langage machine. Ces représentations sont exclusivement destinées aux calculs arithmétiques. Cependant, pour mémoriser et transmettre des textes, l'ordinateur doit aussi utiliser des caractères alphanumériques : Alphabétiques (A -Z, a-z), numériques (0,..,9), ponctuation, spéciaux (&, $, %,..)…etc. Pour coder ces caractères on associe à chacun d'entre eux, dans une table dite de correspondance, un code binaire.

Actuellement il existe plusieurs standards de codage (encodage pour certains, norme pour d'autres). La majorité de ces standards ont les mêmes 128 premiers caractères qui correspondent au codage ASCII.

A. ASCII[3] (American Standard Code for Information Interchange)

Le code ASCII (prononcé généralement **aski,** figure 1) est l'un des plus anciens codes utilisés en informatique. Il définit un jeu de 128 caractères numérotés de 0 à 127 et répartis comme suit :

- Les codes 48 à 57 chiffres dans l'ordre (0,1,…,9) ;

[1] Cette normalisation est faite sur la mantisse (codée sur q bits qui égale à n-p-1, et la virgule est placée après le bit à 1 ayant le poids fort) et l'exposant (codé sur p bits et biaisé pour être positif c-à-d rajout de 2^{p-1}-1).
[2] Cette norme est donnée à titre d'exemple et l'étudiant n'est pas censé la connaitre.
[3] http://www.asciitable.com/

- Les codes 65 à 90 représentent les majuscules (A,..., Z) ;
- Les codes 97 à 122 représentent les minuscules (il suffit de modifier le 6ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale);
- Le reste des codes est dédié aux caractères de control.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 1: Codage ASCII

(Source: suivre lien 1, http://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/seq7_codage_caracteres.html ou **suivre lien 2**, http://www.asciitable.com/**)**

Le jeu de ces 128 caractères est codé sur sept bits, pour compléter l'octet de codage le dernier bit (de poids fort) est mis à zéro.

B. ASCII étendu

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Le code ASCII a donc été étendu à huit bits pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu) ce qui a donné naissance à plusieurs codages.

Exemple :

ISO-8859-1 est l'ASCII avec les caractères accentués d'Europe occidentale.

C. Unicode

Les diverses extensions du code ASCII présentent l'inconvénient d'être incompatibles entre elles, et le plus souvent d'être spécialisées pour un jeu de caractères lié à une langue. Comment alors coder dans un même document des textes rédigés avec des alphabets aussi divers que les alphabets latin, cyrillique, grec, arabe, ... (figure 2)?

arabe : السلم	français : paix	grec : ειρήνη
hébreu : שלום	japonais : へいわ	russe : мир
symbole : ☮	tchèque : mír	thai : ความสงบสุข

Figure 2: Diversité de l'alphabet

(Source: suivre lien, http://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/seq7_codage_caracteres.html**)**

La réponse est venue vers les débuts des années 1990 avec l'apparition du standard Unicode. Ce standard explique comment représenter le texte. Il contient une liste de plus de 110000 caractères (avec la septième version et encore plus avec la huitième[1]). Chaque caractère est représenté par un nom et un numéro (appelé point de code) exprimé souvent sous la forme U+XXXX (U pour Unicode et XXXX le représentant en hexadécimale).

[1] http://www.unicode.org

Exemple 1:

Caractère	Nom	Numéro (décimal)
A	LATIN CAPITAL LETTER A	U+0041 (65)
é	LATIN SMALL LETTER E WITH ACUTE	U+00E9 (233)
œ	LATIN SMALL LIGATURE OE	U+0153 (339)
ε	GREEK SMALL LETTER EPSILON	U+03B5 (949)
и	CYRILLIC SMALL LETTER I	U+0438 (1080)
ש	HEBREW LETTER SHIN	U+05E9 (1513)
س	ARABIC SMALL SEEN	U+0633 (1587)
ฆ	THAI CHARACTER KHO KHWAI	U+0E04 (3588)
へ	HIRAGANA LETTER HE	U+3078 (12408)
€	EURO SIGN	U+20AC (8364)
☮	PEACE SYMBOL	U+262E (9774)

Exemple de la représentation de quelques symboles dans Unicode.
(Source: suivre lien, http://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/seq7_codage_caracteres.html)

Exemple 2:

Représentation de l'alphabet arabe dans Unicode.
(Source : suivre lien, https://fr.wikipedia.org/wiki/Table_des_caract%C3%A8res_Unicode_(0000-0FFF))

Rôle de l'Unicode

Unicode se contente de recenser, nommer les caractères et leur attribuer un numéro. Mais il ne dit pas comment ils doivent être codés en informatique. Plusieurs codages des caractères Unicode existent et sont les Format de Transformation Universels (Universal Transformation Format, UTF):

- UTF-32 qui code chaque caractère sur 32 bits (soit quatre octets) ;
- UTF-16 qui code chaque caractère sur 16 ou 32 bits (soit deux ou quatre octets) ;
- UTF-8 qui code chaque caractère sur 8, 16, 24 ou 32 bits (soit un, deux, trois ou quatre octets).

Le format le plus couramment utilisé, notamment pour les pages Web, est UTF-8. Un texte en UTF-8 est simple, il est partout en ASCII (donc un octet est utilisé pour économiser l'espace utilisé), et dès qu'on a besoin d'un caractère appartenant à l'Unicode deux, trois ou quatre octets sont utilisés (en fonction du caractère Unicode utilisé).

D. Autres codage

Plus haut nous avons cité les standards de codage de texte (c-à-d les plus souvent utilisés), d'autres codages, qui sont à l'origine de ces standards, comme le DCB (Binary Coded Decimal, Décimal Codé en Binaire) ou l'EBCDIC (Extended Binary Coded Decimal Information Code, Code Etendu de l'Information Décimale Codée en Binaire) qui sont à l'origine de l'ASCII ou encore l'ANSI (American National Standards Institute) qui est un dérivé de l'ASCII, sont généralement cités dans la littérature.

3.10. Codage des couleurs

Les couleurs en informatique s'obtiennent à partir des trois couleurs de base: le rouge, le vert et le bleu. Chacune de ces trois couleurs a une luminosité plus au moins forte qui est déterminée par la valeur d'un octet. Cette valeur de luminosité peut donc aller de : 00000000 (la plus claire) à 11111111 (la plus foncée).

Pour coder la couleur d'un pixel (PICture ELement, qui est la plus petite unité affichable à l'écran), il faut utiliser trois octets qui définissent chacun la luminosité des couleurs rouge, verte et bleue (couleur à 24 bits). Trois octets sont bien longs à écrire : on préfère utiliser un nombre hexadécimal.

Pour coder la couleur d'un pixel, on utilise un nombre hexadécimal à six chiffres sous le format 0xRRVVBB où 0x représentent le hexadécimal, RR correspondent à la luminosité du rouge, VV à celui du vert et les deux derniers BB au bleu.

Exemple :

0xFF0000 : avec une intensité maximale sur le rouge ;
0x00FF00 : avec une intensité maximale sur le vert ;
0x0000FF : avec une intensité maximale sur le bleu ;
0xFFFFFF : avec une intensité maximale sur les trois couleurs on obtient le blanc.

3.11. Codage des images

A. Qualité ou définition de l'image

Une image est constituée d'un ensemble de pixels répartis sur une grille (figure 3). Deux nombres sont importants pour décrire cette grille : le nombre de pixels en largeur et le nombre de pixels en hauteur. Plus ces nombres sont élevés, plus la surface de chaque pixel est petite et plus l'image est proche de l'originale (augmentation de la définition).

Figure 3: Répartition des pixels sur la grille.

Exemple :

Une image possédant 800 pixels en largeur et 600 pixels en hauteur aura une définition notée 800x600 pixels.

B. Profondeur de l'image

La profondeur ou la dynamique d'une image est le nombre de bits utilisés pour coder la couleur de chaque pixel.

Exemple :

Image en noir et blanc → 1 bit pour chaque pixel ;
Image avec 256 couleurs → 1 octet pour chaque pixel ;
Image en couleur vrai (True Color : 16 millions de couleurs) → 3 octets (24 bits) pour chaque pixel.

C. Poids de l'image

Le poids d'une image (exprimé en Ko ou en Mo) est égal à son nombre de pixels (définition) multiplié par sa profondeur.

Exemple :

800x600x3 octets ≈ 1,44 Moctets est le poids d'une image qui possède une définition de 800x600 et une profondeur de 3 octets.

3.12. Codage des sons

Pour coder ou numériser le son deux phases sont essentielles l'échantillonnage et quantification (figure 4).

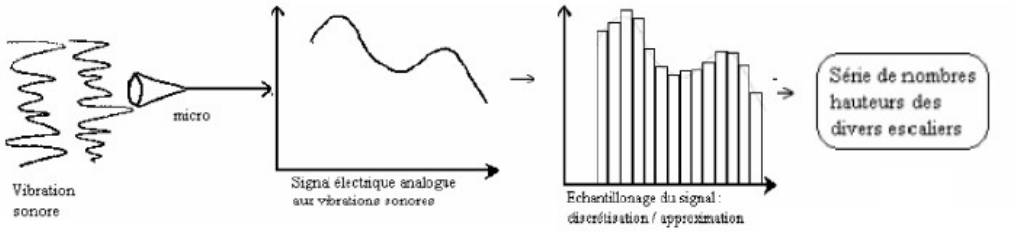


Figure 4: Etapes pour coder le son.

A. Echantillonnage (figure 5)

Représente le passage du temps continu au temps discret.

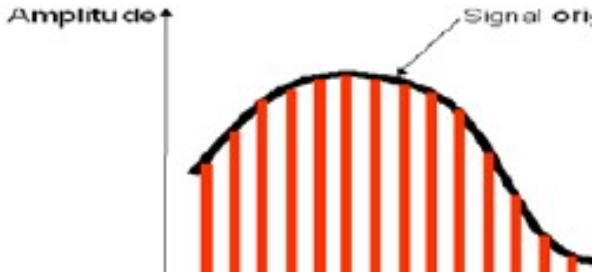


Figure 5: Echantillonnage.

B. Quantification (figure 6)

Représente le passage des valeurs continues aux valeurs discrètes.

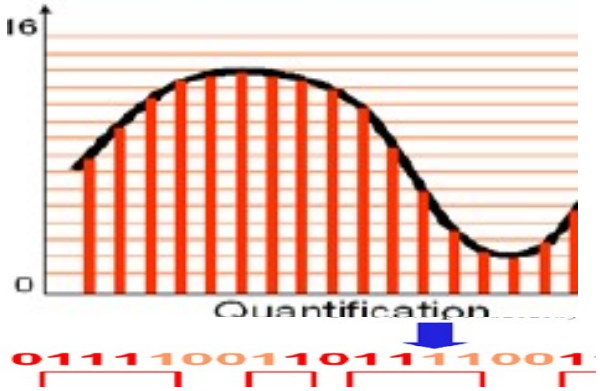


Figure 6: Quantification.