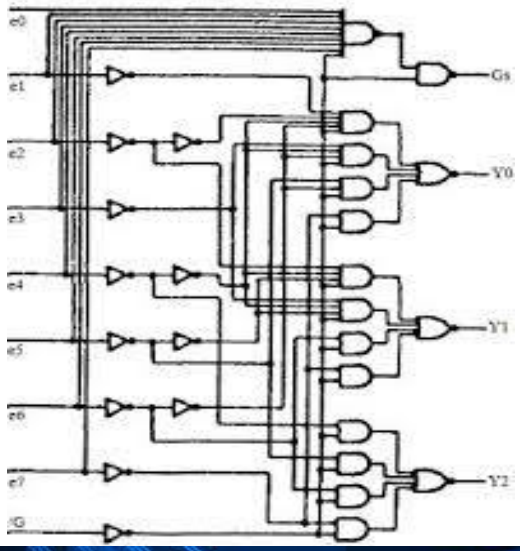
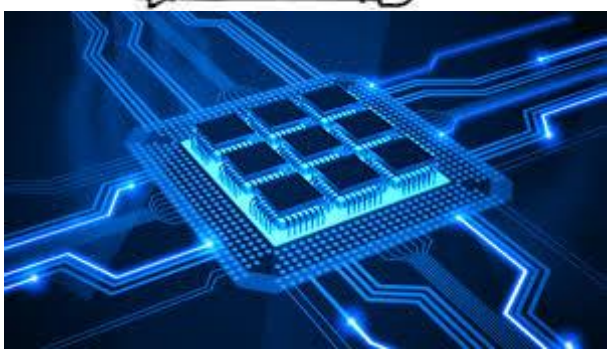
Université d'El-Oued
Faculté de Sciences exactes
1ère MI. LMD

Module :

Codage et Représentation de l'Information





Support de cours

L'Objectif :

Prendre connaissance des notions de base de l'informatique, le codage et la représentation de l'information..et des connaissances sur la théorie formelle basée sur l'Algèbre de Boole pour la synthèse des circuits.

Plan du cour :

Chapitre 0 : Introduction

Chapitre1 : Codification et représentation des nombres

- ***a. Les Entiers Positifs***
  - **Systèmes d'énumérations**
  - **Arithmétique**
- ***b. Les Entiers Négatifs***
  - **Représentation des nombres négatifs en SVA (signe et valeurs absolues)**
  - **Représentation des nombres négatifs en CP1 (Complément à 1)**
  - **Représentation des nombres négatifs en CP2 (Complément à 2)**
  - **Arithmétique**
- ***c. Les Nombres Réels***
  - **Représentation des nombres Réels en virgules fixe**
  - **Représentation des nombres Réels en virgules flottantes**
  - **Arithmétique**

Chapitre2 : Codification et représentation α-Numérique

a. Le code ASCII
b. Le code BCD
c. Le code Gray
d. L'Unicode

Chapitre 3 : Algèbre de Boole

a. Introduction
b. Terminologie
c. Les fonctions booléennes et leurs représentations
d. Théorèmes et postulat de l'algèbre de Boole
e. Opération de base
f. Evaluation des expressions booléennes
g. Les tables de vérité

Chapitre1 : Codification et représentation des nombres

1 : Les systèmes de numération

(أنظمة التعداد)

Introduction :

L'homme a besoin **d'un système de codage** pour **identifier**, **quantifier**, **qualifier** les objet, les lieux, les évènements…Ces codes lui permettent de **mémoriser**, **traiter** et **communiquer les informations** . Ils peuvent être :

- Le langage
- L'écriture
- Numérations
- Le graphisme
- ….

Chaque code respecte des règles. Par exemple, à l'écriture correspond:

- Une liste des symboles prédéfinis : L'alphabet
- Des règles d'utilisations des symboles :
  Plusieurs symboles (lettres) ⟶ mot
  Plusieurs mots ⟶ phrase
- Des règles de syntaxe pour ordonner les mots dans une phrase

1- Système de numération (أنظمة التعداد)

La numération est la science qui traite la dénomination et la représentation graphique des nombres.
Le problème posé est de représenter tous les entiers naturels et les décimaux à l'aide d'un ensemble fini de symboles (souvent des chiffres) rassemblés selon des règles (le code) pour former un nombre.

Le système de numération décrit la façon avec laquelle les nombres sont représentes ; et il est défini par :

- **Un alphabet A :** ensemble de symboles ou chiffres,
- **Des règles d'écritures des nombres :** Juxtaposition de symboles

Un système de numération positionnel pondéré à base **B** est défini sur **un alphabet** de **b chiffres** :

$$A = \{a_0, a_1, \ldots, a_{b-1}\} \text{ avec } 0 \leq a_k < B$$

- Dans une base B , on utilise b symboles (chiffres) distincts pour représenter les nombres.
- La valeur de chaque symbole (chiffres) doit être strictement inférieure à la base B.

Quelque soit le système de numération (base B), tout **entier positif X** (parfois réel aussi) s'écrire comme :

$$X = (a_n a_{n-1} \ldots\ldots a_2 a_1 a_0, a_{-1} a_{-2} a_{-3} \ldots a_{-K})_b = \left(\sum_{i=-k}^{n} ai.\, b^i\right)_{10}$$

avec $a_i \in (0,1,\ldots.b-1)$

- a_i : est un chiffre de l'alphabet de poids i (position i).
- a_0 : chiffre de poids 0 appelé le chiffre de poids faible (LSB = Low Significant Bit)
- a_n : chiffre de poids n appelé le chiffre de poids fort (MSB = Most Significant Bit)

Ex :

$$(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (5)_{10}$$

Il est important de connaître les différents systèmes car ils sont utilisés en informatique et plus généralement dans le traitement de l'information.

A) Le système décimal (النظام العشري)

C'est le plus utilisé et le plus connu et que nous utilisons tous les jours.

- Il est basé sur le nombre 10 qui est la base du système décimal.
- L'alphabet de ce système est compose de dix chiffres : A = {0,1,2,3,4,5,6,7,8,9}

  Les chiffres 0,1,2,3,4,5,6,7,8 et 9 soit dix caractères au total sont utilisés pour la représentation des nombres.
  Ces chiffres sont classés de droite à gauche "unités, dizaines, centaine, milliers ….".
- C'est un système positionnel. Chaque position possède un poids.

Par exemple le nombre N= 2356 dans ce système :

Par convention nous l'écrirons N=$(2356)_{10}$. L'indice '10' indique la base dans laquelle le nombre est écrit.

Décomposition du nombre dans sa base

Ce nombre N peut être écrit sous la forme suivante :

N = 2. 10^3 **+ 3.** 10^2 **+ 5.** 10^1 **+ 6.** 10^0 **= 2000+300+50+6 = 2356 .**

Chaque chiffre du nombre est à multiplier par **une puissance** de 10 : c'est ce que l'on nomme le **poids du chiffre.** L'exposant de cette puissance est nul pour le chiffre situé le plus à droite et s'accroît d'une unité pour chaque passage à un chiffre vers la gauche. Cette façon d'écrire les nombres est appelée **: système de numération de position**.

• Si, dans la vie courante, nous utilisons la base 10, les ordinateurs utilisent la base 2, car :

- **L'inconvénient de la base 10** est **la difficulté de stocker** et de transmettre **les 10 niveaux avec un signal**
- **L'avantage de la base 2** est **la facilité de stocker et de transmettre les 2 niveaux avec un signal**. Elle permet l'utilisation d'éléments électroniques bistables et une transmission fiable, même dans des environnements bruités et imprécis

B) Le système binaire(النظام الثنائي)

Introduction :

Les informations traitées par les ordinateurs sont de différentes natures :

- Nombres, texte,
- Images, sons, vidéo,
- Programmes, …

Mais , elles sont toujours représentées sous forme binaire (**une suite de 0 et de 1**) par ce que dans un circuit, on dispose de deux niveaux de voltage (Exemple 0V et 5V ou 5V et 12V) pour représenter toute information, qu'elle soit de nature logique ou numérique. On représente donc les nombres en base 2 **en associant par exemple la valeur binaire 0 au voltage 0V, et la valeur binaire 1 au voltage 5V**.

Un état binaire est appelé (BIT : Binary digIT). **Un bit prend les valeurs 0 ou 1.**

- Un bit ne peut prendre que deux valeurs. Selon le contexte, numérique, logique (voir algèbre de Boole), électronique numérique, magnétique, ou optique, on les appelle « zéro » et « un » ce qui équivaut respectivement à « faux » et « vrai », « ouvert » et « fermé », « nord » et « sud », ou « noir » et « blanc »

Technologiquement parlant, il existe une grande variété de moyen d'encodage binaire de l'information, selon le support de stockage ou de transmission utilisé : les propriétés physiques telle que la polarisation magnétique, la charge, le courant ou la tension électrique, l'intensité lumineuse, sont couramment utilisées. L'essentiel est d'autoriser avec une très bonne fiabilité la distinction entre les deux états 0 et 1 de manière à limiter les erreurs.

Bit dans une mémoire :

Une mémoire vive est constituée de centaines de milliers de petits condensateurs emmagasinant des charges. Lorsqu'il est chargé, l'état logique du condensateur est égal à 1, dans le cas contraire il est à 0, ce qui signifie que chaque condensateur représente un bit de la mémoire. Etant donné que les condensateurs se déchargent, il faut constamment les recharger (le terme exact est rafraîchir, en anglais refresh) à un intervalle de temps régulier appelé cycle de rafraîchissement. Les mémoires DRAM nécessitent par exemple des cycles de rafraîchissement est d'environ 15 nanosecondes (ns). Chaque condensateur est couplé à un transistor (de type MOS) permettant de

« récupérer » ou de modifier l'état du condensateur. Ces transistors sont rangés sous forme de tableau (matrice), c'est-à-dire que l'on accède à une case mémoire (aussi appelée point mémoire) par une ligne et une colonne.

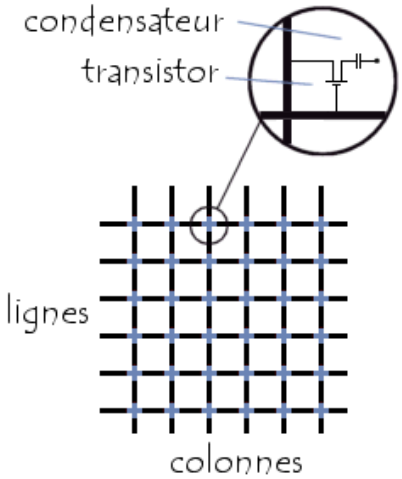


Fig 01 : *Bit dans une mémoire*

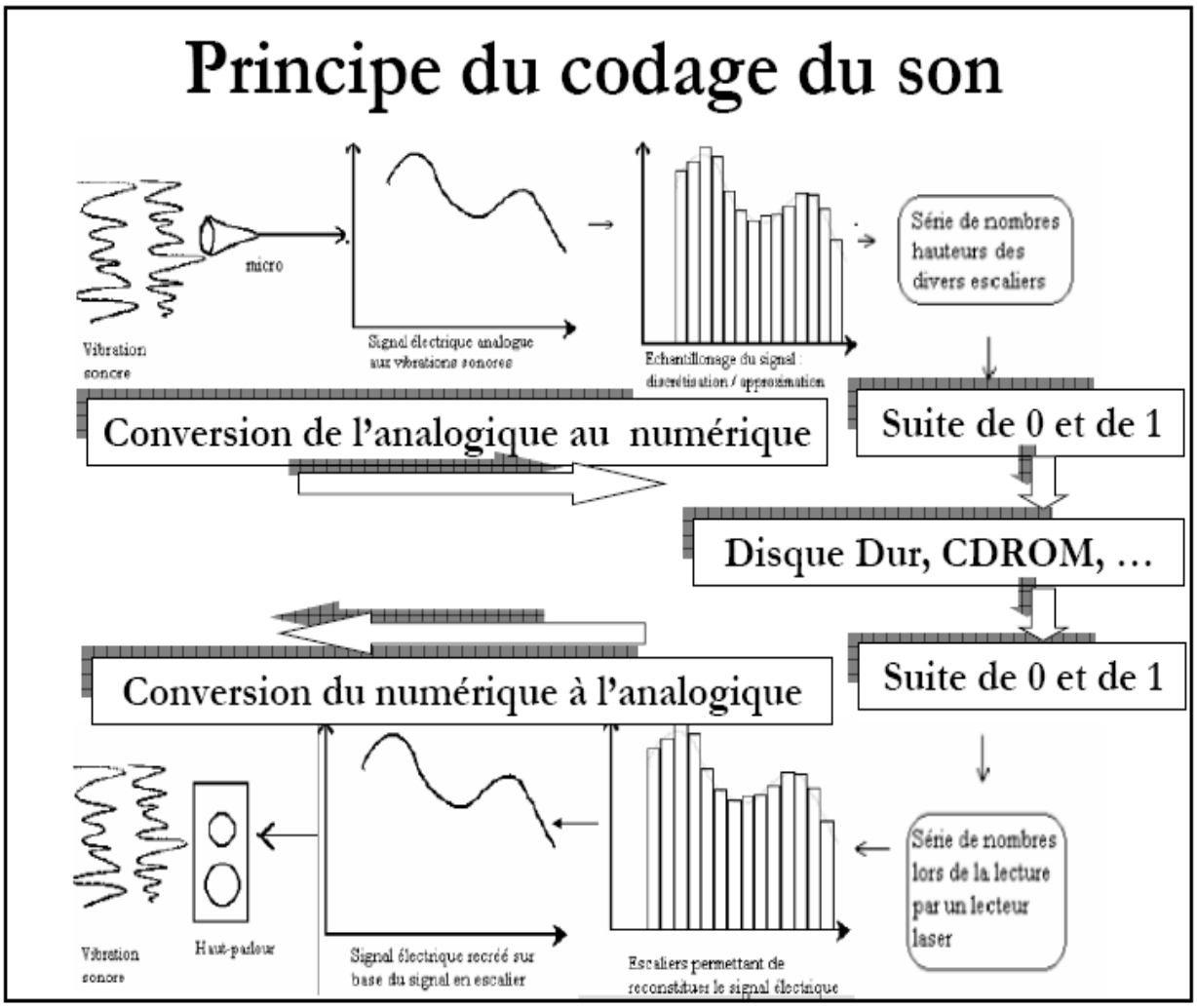


Fig 02 : *Exemple (le son) de principe de codage*

Codage de l'information :

Permet d'établir une correspondance qui permet sans ambiguïté de passer d'une représentation (dite externe) d'une information à une autre représentation (dite interne : sous forme binaire) de la même information, suivant un ensemble de règle précise.

En informatique, Le codage de l'information s'effectue principalement en trois étapes :

- L'information sera exprimée par une suite de nombres (Numérisation)
- Chaque nombre est code sous forme binaire (suite de 0 et 1)
- Chaque élément binaire est représente par un état physique

  Exemples :

  Charge électrique (RAM : Condensateur-transistor) : Charge (bit 1) ou non charge (bit 0)
  Magnétisation (Disque dur, disquette) : polarisation : Nord (bit 1) ou Sud (bit 0)
  Alvéoles (CDROM): réflexion (bit 1) ou pas de réflexion (bit 0)
  Fréquences (Modem) : dans un signal sinusoïdal : Fréquence f1 (bit 1) : Fréquence f2 (bit 0)

Dans le système binaire (ou base 2) utilises deux symboles (0 et 1); et chaque chiffre peut avoir 2 valeurs différentes : 0, 1

Tout nombre écrit dans ce système vérifie la relation suivante :

On écrit :

$$(a_{n-1}a_{n-2}\cdots a_1a_0)_2 = a_{n-1}.2^{n-1} + a_{n-2}.2^{n-2} + \cdots + a_1.2^1 + a_0.2^0$$

L'expression de droite dans la base 10

Exemples :

$(10\ 110)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
$(10\ 10)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$ donc : $(10110)_2 = (22)_{10}$

C)Le système octal :

Ce système de numération ayant (8) comme base8. Il faut noter que dans ce système nous n'aurons plus 10 symboles mais 8 seulement : 0, 1, 2, 3, 4, 5, 6, 7

Tout nombre écrit dans ce système vérifie la relation suivante :

On écrit :

$$(a_{n-1}a_{n-2}\cdots a_1a_0)_8 = a_{n-1}.8^{n-1} + a_{n-2}.8^{n-2} + \cdots + a_1.8^1 + a_0.8^0$$

L'expression de droite dans la base 10

Exemple :

On peut décomposer $(745)_8$ de la façon suivante :

$(745)_8 = 7 \times 8^2 + 4 \times 8^1 + 5 \times 8^0$

$(745)_8 = 7 \times 64 + 4 \times 8 + 5 \times 1$

$(745)_8 = 448 + 32 + 5$

$(745)_8 = (485)_{10}$

Ce système de numération est très peu utilisé de nos jours. Anciennement, il servait au codage des nombres dans les ordinateurs de première génération.

D) Le système Hexadécimale :

Dans le système hexadécimal (ou base 16) utilise les 16 symboles suivant :

{0,1,2,3,4,5,6,7,8,9,A=$10_{(10)}$,B=$11_{(10)}$,C=$12_{(10)}$,D=$13_{(10)}$,E=$14_{(10)}$,F=$15_{(10)}$}

Tout nombre écrit dans ce système vérifie la relation suivante :

On écrit :

$$(a_{n-1}a_{n-2}\cdots a_1a_0)_{\mathbf{16}} = a_{n-1}\cdot\mathbf{16}^{n-1} + a_{n-2}\cdot\mathbf{16}^{n-2} + \cdots + a_1\cdot\mathbf{16}^{1} + a_0\cdot\mathbf{16}^{0}$$

L'expression de droite dans la base 10

Exemple :

On peut décomposer $(A8)_{16}$ de la façon suivante :

$(A8)_{16} = A.16^1 + 8.16^0 = 10.16 + 8.1 = (168)_{10}$

Ce système de numération est très utilisé dans les systèmes ordinateurs et micro ordinateurs ainsi que dans le domaine des transmissions de données.

Selon le contexte il peut être plus judicieux d'utiliser un code plutôt qu'un autre, il faut donc savoir comment passer de l'un à l'autre.

2-Transcodage (ou conversion de base) :
التحويلات

Le transcodage (ou conversion de base) est l'opération qui permet de passer de la représentation d'un nombre exprime dans une base à la représentation du même nombre mais exprime dans une autre base.

Par la suite, on verra les conversions suivantes :

- Décimale vers Binaire, Octale et Hexadécimale
- Binaire vers Décimale, Octale et Hexadécimale

A) Changement de base de la base 10 vers une base b :

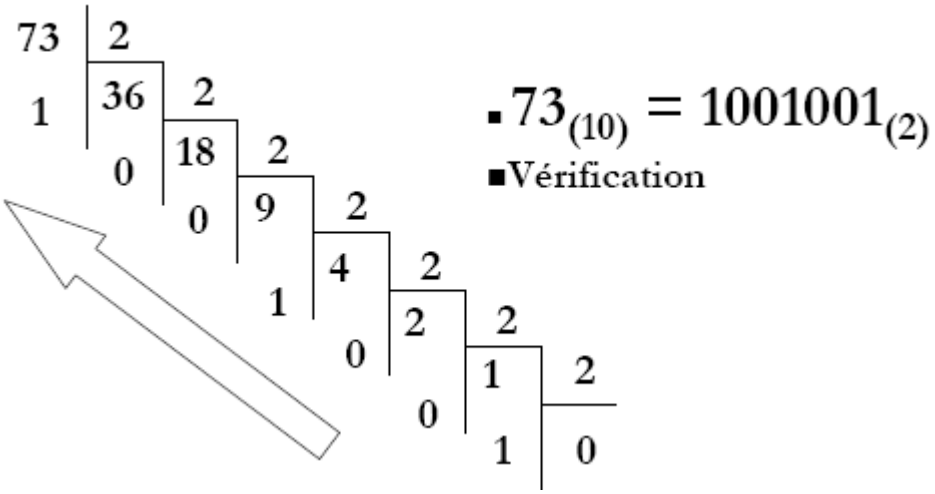
La règle à suivre est **les divisions successives** :

_ On divise le nombre par la base b
_ Puis le quotient (الحاصل)par la base b
_ Ainsi de suite jusqu'à l'obtention d'un quotient nul
_ La suite des restes (سلسلة البواقي) correspond aux symboles de la base visée.

- **<u>On obtient en premier le chiffre de poids faible et en dernier le chiffre de poids fort</u>**.

Exemple : décimale vers binaire

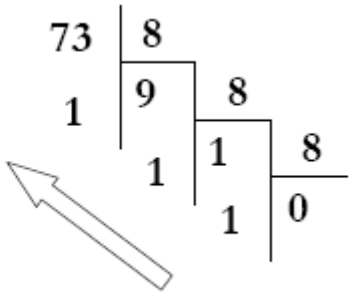
Soit N le nombre d'étudiants d'une classe représenté en base décimale par : $N = 73_{(10)}$
La représentation en Binaire?



Exemple : décimale vers Octale

Soit N le nombre d'étudiants d'une classe représenté en base décimale par : $N = 73_{(10)}$
La représentation en Octale?

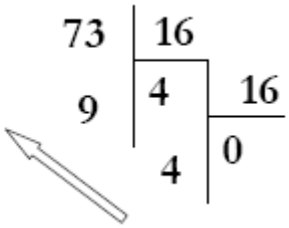
$\blacksquare 73_{(10)} = 111_{(8)}$ ■Vérification



Exemple : décimale vers Hexadécimale

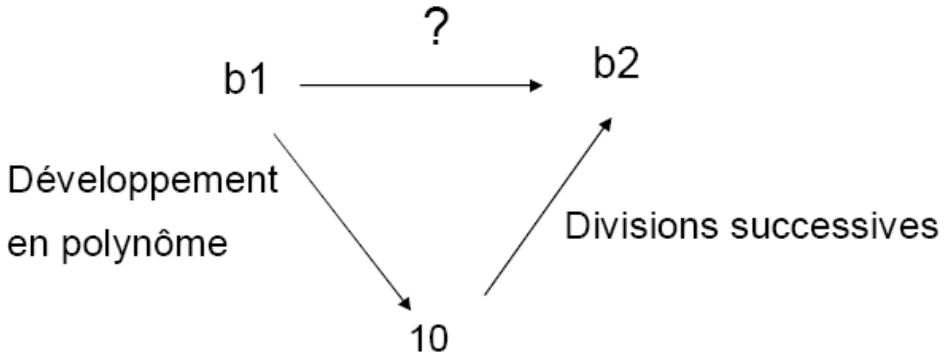
Soit N le nombre d'étudiants d'une classe représenté en base décimale par : N = 73(10)
La représentation en Hexadécimale?

$\blacksquare 73_{(10)} = 49_{(16)}$ ■Vérification



B) Conversion d'une base b1 à une base b2

Il n'existe pas de méthode pour passer d'une base b1 à une autre base b2 directement. L'idée est de convertir le nombre de la base b1 à la base 10 , en suit convertir le résultat de la base 10 à la base b2 .



- ***De la base binaire vers une base b***

Solution 1:

Convertir le nombre en base binaire vers la base décimale puis convertir ce nombre en base 10 vers la base b.

Exemple :

$10010_{(2)} = ?_{(8)}$
$10010_{(2)} = 2^4+2_{(10)}=18_{(10)}=2*8^1+2*8^0{}_{(10)} =22_{(8)}$

Solution 2 :

- ***Binaire vers décimale*** **:** par définition $(\sum_{i=0}^{n-1} a_i\, b^i)$

Exemple :

Soit N un nombre représenté en binaire par : $N = 1010011101_{(2)}$ Représentation Décimale?
$N=1.\ 2^9+0.\ 2^8+1.\ 2^7+0.\ 2^6+0.\ 2^5+1.\ 2^4+1.\ 2^3+1.\ 2^2+0.\ 2^1+1.\ 2^0$
$=512 + 0 + 128 + 0 + 0 + 16 + 8 + 4 + 0 + 1 =669_{(10)}$ alors $1010011101_{(2)} = 669_{(10)}$

- ***Binaire vers octale :*** regroupement des bits en des **sous ensembles** de **trois bits** puis remplacer chaque groupe par le symbole correspondant dans la base 8.(Table1)

Correspondance
Octale \Binaire

Symbole Octale	suite binaire
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

.Table1 : *Correspondance Octale /Binaire*

Exemples : binaire vers octale :

Soit N un nombre représenté en base binaire par : N = 1010011101(2) ; La représentation Octale?

N = 001 010 011 101(2)
= 1 2 3 5 (8)
alors 1010011101(2)= 1235(8)

Octal vers binaire :

En octal chaque, symbole de la base s'écrit sur 3 bits en binaire. L'idée de base est de replacer chaque symbole dans la base octal par sa valeur en binaire sur 3 bits (faire des éclatement sur 3 bits).

Exemples :

(345)8=(011 100 101)2
(65,76)8=(110 101, 111 110)2
(35,34)8=(011 101 , 011 100)2

Remarque :
le remplacement se fait de droit à gauche pour la partie entière et de gauche à droite pour la partie fractionnelle .

- ***Binaire vers Hexadécimale :*** regroupement des bits en sous-ensembles de quatre bits puis remplacer chaque groupe par le symbole correspondant dans la base 16. (Table2)

Correspondance
Hexadécimale \Binaire

S. Hexad.	suite binaire	S. Hexad.	suite binaire
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

.Table2 : *Correspondance Hexadécimale/Binaire*

Exemple : binaire vers Hexadécimale :

Soit N un nombre représenté en base binaire par : N = 1010011101(2) ; La représentation Hexadécimale?

N = 0010 1001 1101(2)
= 2 9 D (16)
alors 1010011101(2)= 29D(16)

(110010100,10101) 2 = (0001 1001 0100,1010 1000) 2 = (194,A8) 16

Hexadécimale vers binaire :

En Hexa chaque symbole de la base s'écrit sur 4 bits. . L'idée de base est de replacer chaque symbole par sa valeur en binaire sur 4 bits (faire des éclatement sur 4 bits).

Exemple :

(345B) 16 = (0011 0100 0101 1011) 2
(AB3,4F6) 16 = (1010 1011 0011 , 0100 1111 0110) 2

Remarque :
le remplacement se fait de droit à gauche pour la partie entière et de gauche à droite pour la partie fractionnelle.

C) Cas d'un nombre réel

Un nombre réel est constitué de deux parties : ***la partie entière*** et ***la partie fractionnelle***.

- La partie entière est transformée en effectuant des divisions successives.
- La partie fractionnelle est transformée en effectuant des multiplications successives par 2 , la partie entière ainsi obtenue représentant le poids binaire (1 ou 0) La partie fractionnelle restante est à nouveau multiplier par 2 et ainsi de suite jusqu'à ce qu'il n'y ait plus de partie fractionnaire ou que la précision obtenue soit jugée suffisante.

Exemple : 35,625=(?)2 PE= 35 = (100011)2 PF= 0,625 = (?)2

0,625 * 2 = 1 ,25
0,25 * 2 = 0 ,5 ↓ Sens de lecture
0,5 * 2 = 1 ,0

(0,625)=(0,101)2 Donc 35,625=(100011,101)2

- Le même principe pour les autre bases (8,16,….)

Exemple : conversion de 171,3046875 en hexadécimal
Conversion de 171 : donne (AB)16
Conversion de 0,3046875
0,3046875 x 16 = 4,875 = 4 + 0,875
0,875 x 16 = 14,0 = 14 + 0
Donc (171,3046875)10 = (AB,4E)16

3- Codage des entiers signés en binaire

La vie n'étant faite que de chose positives, il a fallu introduire les nombres négatifs . Il existe au moins trois façons pour coder :

- **Code binaire signe (par Signe et Valeur Absolue)**
- **Code complément à 1**
- **Code complément à 2 (Utilisé sur ordinateur)**

Pour toutes ces solutions on aura toujours **un bit utilisé pour préciser le signe du nombre**

A) Binaire signe (Signe et Valeur absolue)

Le bit le plus significatif (le plus à gauche) est utilisé pour représenter le signe du nombre :

- Si le bit le plus fort = **1** alors **nombre négatif**
- Si le bit le plus fort = **0** alors **nombre positif**

Les autres bits codent la valeur absolue du nombre

Exemple 1 : si codage sur 4 bits
(**0**111)2 = 7 car bit de poids fort à 0
(**1**111)2 = -7 car bit de poids fort à 1

Exemple 2 : Sur 8 bits, codage des nombres -24 et -128 en (bs)
-24 est code en binaire signe par : 1 0 0 1 1 0 0 0 (bs)
-128 hors limite ⟶ nécessite 9 bits au minimum (128 = 10000000_2)

Codage sur p bits : Avec p bits, on code N valeurs avec :

$$0 \leqslant N \leqslant 2^p - 1$$

Codage sur p bits pour les entiers signés :

Pour p bits :

$$-(2^{p-1} - 1) \leqslant N \leqslant 2^{p-1} - 1$$

Pour un entier signé sur 16 bits :
Nombres positifs : 0XXXXXXXXXXXXXXX
Nombres négatifs : 1XXXXXXXXXXXXXXX
On a 15 bits pour coder la valeur absolue du nombre soit 2^{15} = 32768 valeurs possibles
Pour les positifs : de 0 à 32767
Pour les négatifs : de -0 à -32767

Cette méthode présente des inconvénients :

- Le zéro a 2 représentations distinctes 000…0 et 100…0, soit +0 et -0;
- Les tables d'additions et de multiplication sont compliquées , à cause du bit de signe qui doit être traité à part.

B) Code complément à 1 (Complément logique):

Aussi appelé Complément Logique (CL) ou Complément Restreint (CR) :

C) Les nombres positifs sont codes de la même façon qu'en binaire pure.

D) Un nombre négatif est codé en inversant chaque bit de la représentation de sa valeur absolue :

(0 → 1) (1→0)

Le bit le plus significatif est utilise pour représenter le signe du nombre :

E) Si le bit le plus fort = 1 alors nombre négatif

F) Si le bit le plus fort = 0 alors nombre positif

Exemples :

|-24|en binaire pur = 0 0 0 1 1 0 0 0 (2) puis
on inverse les bits = 1 1 1 0 0 1 1 1 (ca1)

G) Coder 100 et -100 par complément a 1 (ca1) sur 8 bits

100(10) = (01100100) (ca1)
-100(10) = (10011011) (ca1)

H) Décoder en décimal (11000111)(ca1) et (00001111)(ca1)

(11000111) (ca1) = -56(10)
(00001111) (ca1) = 15(10)

On peut constater que l'intervalle des entiers N qu'on peut représenter en complément à 1 est la même que pour la présentation (signe + valeur absolue) . Pour p bits , on a l'intervalle suivant:

$$-(2^{p-1} - 1) \leqslant N \leqslant 2^{p-1} - 1$$

Alors la représentation en Complément à 1 est symétrique , c'est-à-dire les mêmes nombres >0 et <0 sont représentables. Elle est donc facile à réaliser électroniquement .

En complémentà1:la retenue engendrée par l'addition des bits les plus à gauche est additionnée aux bits le plus à droite

Exemple (Représentation sur 7 bits + 1 bit de signe) :

```
  00001010          +10
  11111100          +(-3)
1 00000110          +7
+       1
  --------
  00000111
```

Limitations :

I) Deux codages différents pour 0 (+0 et -0)
Sur 8 bits : +0=0 0 0 0 0 0 0 0(ca1) et -0=1 1 1 1 1 1 1 1(ca1)

J) Multiplication et l'addition sont moins évidentes.

C)Code complément à 2 (Complément arithmétique):

Aussi appelé Complément Vrai (CV) :

- Les nombres positifs sont codés de la même manière qu'en binaire pure.
- Un nombre négatif est code en ajoutant la valeur 1 a son complément à 1

Le bit le plus significatif est utilise pour représenter le signe du nombre

Exemple : -24 en complément a 2 sur 8 bits

24 est codé par 0 0 0 1 1 0 0 0(2)
-24 = 1 1 1 0 0 1 1 1(ca1)
donc -24 est codé par 1 1 1 0 1 0 0 0(ca2)

Etendu de codage :

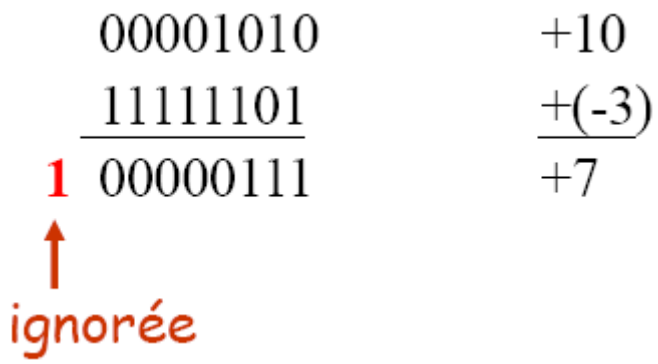
L'intervalle des entiers N qu'on peut représenter en complément à 2 est :

$$-(2^{p-1}) \leqslant N \leqslant 2^{p-1} - 1$$

Sur 1 octet (8 bits), codage des nombres de -128 a 127
+0 = 00000000 -0=00000000
+1 = 00000001 -1=111111111
… …
+127= 01111111 -128=10000000

En complément à 2 : le retenue est purement ignorée

Exemple (Représentation sur 7 bits + 1 bit de signe) :



Avantages :

Un seul codage pour 0. Par exemple sur 8 bits :
+0 est code par 00000000(ca2)
-0 est code par 11111111(ca1)
Donc -0 sera représenté par 00000000(ca2)

4) Arithmetique en base 2

Les opérations sur les entiers s'appuient sur des tables d'addition et de multiplication :

Addition

0	0	0
0	1	1
1	0	1
1	1	(1) 0

Retenu

Multiplication

0	0	0
0	1	0
1	0	0
1	1	1

Addition en complément à 2 :

La notation en complément à 2 et la notation en complément à 1 sont très semblables. Toutefois, la notation en complément à 2 jouit généralement de certains avantages quand vient le temps de construire des circuits.
Nous allons maintenant étudier comment les machines numériques additionnent et soustraient quand les nombres négatifs sont écrits dans la notation en complément à 2. Dans tous les cas étudiés, il est important que vous remarquiez que le bit de signe de chaque nombre est traité sur le même pied que les bits de la partie grandeur.

Cas 1: deux nombres positifs

L'addition de deux nombres positifs est immédiate. Soit l'addition de +9 et +4:

+ 9	0	1001	(cumulande)
+ 4	0	0100	(cumulateur)
+ 13	0	1101	(Somme)

Bit de signe

Remarquez que les bits de signe du cumulande et du cumulateur sont 0 et que celui de la somme est aussi 0, ce qui indique un nombre positif. Notez aussi qu'on a fait en sorte que le cumulande et le cumulateur aient le même nombre de bits. Il faut toujours s'assurer de cela dans la notation en complément à 2.

Cas 2: nombre positif et nombre négatif plus petit

Soit l'addition de +9 et de -4. Rappelez-vous que -4 est exprimé dans la notation en complément à 2. Donc +4 (00100) doit être converti en -4 :

+ 9	0	1001	(cumulande)
- 4	1	1100	(cumulateur)
+ 5 ~~1~~	0	0101	

Bit de signe

Dans ce cas-ci, le bit de signe du cumulateur est 1. Remarquez que les bits de signe sont aussi additionnés. En fait, un report est produit au moment de l'addition du dernier rang. Ce report est toujours rejeté d'où la somme finale de 00101, soit le nombre décimal +5.

Cas 3: nombre positif et nombre négatif plus grand.

Soit l'addition de -9 et de +4:

- 9	10111
+ 4	00100
- 5	11011

Dans ce cas-ci le bit de signe de la somme est 1, ce qui indique un nombre négatif. Comme la somme est un nombre négatif, la réponse est le complément à 2 de la grandeur exacte. Donc 1011 est en réalité le complément à 2 de la somme. Pour trouver la grandeur exacte de la somme, on doit prendre le complément à 2 de 1011, ce qui donne 0101 (5); la réponse est donc 11011 = -5.
On peu également vérifier le résultat en additionnant le poids de chaque bit, avec le bit de signe qui vaut -2^N, où N est le nombre de bits de grandeur.

$-1\cdot2^4+ 1\cdot2^3 +0\cdot2^2 +1\cdot2^1 +1\cdot2^0 = -16 + 8 + 2 + 1 = -5$

3-4.4 Cas 4: deux nombres négatifs

- 9	10111
- 4	11100
-13	⊣10011

ce report n'est pas pris en considération

Le résultat définitif est de nouveau négatif (-13).

Cas 5: nombres égaux et opposés

-9	10111
+9	01001
0	⊣00000

ce report n'est pas pris en considération

Le résultat est évidemment +0, comme on s'y attendait.

Soustraction: complément à 2

Une opération de soustraction qui porte sur des nombres exprimés dans la notation en complément à 2 est en réalité une opération d'addition qui diffère peu des cas examinés précédemment. Quand on soustrait un nombre binaire (le diminuteur) d'un autre nombre (le diminuande), la marche à suivre est comme suit:

a. Prendre le complément à 2 du diminuteur, y compris son bit de signe. Si ce dernier est un nombre positif, il deviendra un nombre négatif dans la notation en complément à 2. Si le diminuteur est un nombre négatif, la complémentation à 2 en fera un nombre positif écrit en grandeur exacte. Autrement dit, nous changeons le signe du diminuteur.
b. Après avoir complémenté à 2 le diminuteur, on l'additionne au diminuande. Le diminuande conserve sa forme initiale. Le résultat de cette addition représente la différence recherchée. Le bit de signe de la différence indique si là réponse est positive ou négative et si on est en notation binaire exacte ou en notation en complément à 2.
c. Rappelez-vous que les deux nombres doivent avoir le même nombre de bits.

Examinons la soustraction suivante: +9 - (+4).

diminuande (+ 9) 01001
diminuteur (+ 4) 00100

Changez le diminuteur pour sa version en complément à 2 (11100), ce qui représente 4. Maintenant additionnez-le au diminuande.

+ 9	01001
- 4	11100
+ 5	100101

La retenue est rejetée, le résultat est donc 00101 = +5

Quand on complémente à 2 le diminuteur, on obtient en réalité -4, de sorte qu'on additionne +9 à -4, ce qui est équivalent à soustraire de +9 le nombre +4. En définitive, toute opération de soustraction se résume à une addition lorsqu'on utilise la notation en complément à 2. **Cette caractéristique de la notation en complément à 2 explique pourquoi c'est la méthode la plus utilisée, puisqu'on peut additionner et soustraire en utilisant les mêmes circuits.**

Dépassement (overflow)

Débordement : ***la taille allouée (8, 16 ... bits) au codage d'un entier est trop petite pour coder ou stocker le résultat d'un calcul***

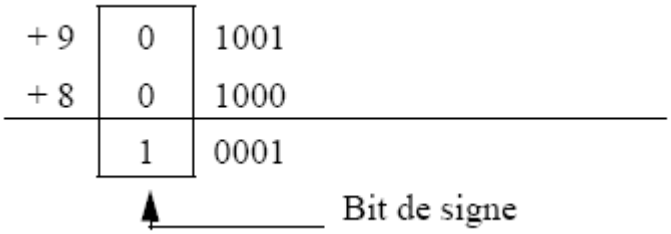
- ***On parle d'overflow quand il y a dépassement de la capacité pour représenter le résultat d'une somme***
- ***Cela peut se produire uniquement si les deux opérandes sont de mêmes signes***
- ***Quand il y a overflow la somme n'est pas de même signe que les opérandes***

- Considérons le cas du débordement : il ya débordement si : les opérandes X et Y sont de même signe , et le résultat est de signe différent :

Exemple

	0 10010			101110	
	0 1101	+13		1 0011	- 13
	0 1001	+ 9		1 0111	- 9
Sur 5 bits	1 0110	-10		0 1010	+ 10
Sur 6 bits	0 1 0110	+22		1 0 1010	- 22

Dans chacun des exemples d'addition et de soustraction que l'on vient d'étudier, les nombres que l'on a additionnés étaient constitués à la fois d'un bit de signe et de 4 bits de grandeur. Les réponses aussi comportaient un bit de signe et 4 bits de grandeur. Tout report fait sur le bit de sixième rang était rejeté. Dans tous les cas étudiés, la grandeur de la réponse ne dépassait jamais la capacité des 4 bits.Voyons l'addition de +9 à +8.



Le bit de signe de la réponse est celui d'un nombre négatif, ce qui est manifestement une erreur. La réponse devrait être +17. Étant donné que la grandeur est 17, il faut plus de 4 bits pour l'exprimer, et il y a donc un dépassement (overflow) sur le rang du bit de signe. Une condition de dépassement donne toujours lieu à un résultat inexact, et on la détecte en examinant le bit de signe du résultat et en le comparant aux bits de signe des nombres additionnés. En additionnant deux nombres de signes différents, il ne peut pas y avoir de dépassement, par contre lorsqu'on additionne deux nombres de même signe, on a un dépassement si le signe du résultat est différent du signe des deux nombres additionnés. Dans un ordinateur, il existe un circuit spécialement conçu pour détecter les conditions de débordement et pour indiquer que la réponse est fausse.

Multiplication de nombres binaires :

On multiplie les nombres binaires de la même façon qu'on multiplie les nombres décimaux. En réalité, le processus est plus simple car les chiffres du multiplicateur sont toujours 0 ou 1, de sorte qu'on multiplie toujours par 0 ou par 1. Voici un exemple de multiplication de nombres binaires non signés.

			1	0	0	1		multiplicande = 9_{10}
		·	1	0	1	1		multiplicateur = 11_{10}
			1	0	0	1		
		1	0	0	1			produits partiels
	0	0	0	0				
1	0	0	1					
1	1	0	0	0	1	1		produit final = 99_{10}

Dans cet exemple, le multiplicande et le multiplicateur sont en notation binaire exacte et il n'y a pas de bit de signe. La marche à suivre est exactement la même que pour les multiplications décimales. D'abord, examinons le bit de poids le plus faible du multiplicateur; dans notre exemple il s'agit d'un 1. Ce 1 multiplie le multiplicande pour donner 1001, c'est notre premier produit partiel. Ensuite, examinons le deuxième bit du multiplicateur.
Il s'agit d'un 1, ce qui donne le second produit partiel. Notez qu'en écrivant le second produit partiel on le décale d'un rang vers la gauche par rapport au premier. Le troisième bit du multiplicateur est 0 et notre troisième produit partiel est 0000; ce troisième produit est aussi décalé d'un rang vers la gauche par rapport au produit partiel précédent. Le quatrième bit du multiplicateur est 1, de sorte que le dernier produit partiel est 1001, que l'on écrit à nouveau décalé d'un rang vers la gauche. On additionne ensuite les quatre produits partiels pour obtenir le produit final.
La plupart des machines numériques ne peuvent additionner que deux nombres binaires à la fois. C'est la raison pour laquelle les produits partiels d'une multiplication ne peuvent être additionnés ensemble en une seule fois. Ils

sont plutôt additionnés deux par deux, c'est-à-dire que le premier est additionné au second, que leur somme est additionnée au troisième et ainsi de suite.

Multiplication en complément à 2

Dans les machines qui utilisent la notation en complément à 2, la multiplication est effectuée de la façon décrite ci-dessus quand le multiplicande et le multiplicateur sont exprimés en notation binaire exacte.
-Si les deux nombres multiplier sont positifs, ils sont déjà dans cette notation et ils sont multipliés tels quels. Le produit résultant est évidemment positif et son bit de signe est 0.
-Quand les deux nombres sont négatifs, ils sont donc écrits dans la notation en complément à 2. Chacun de ces nombres est complémenté à 2 pour obtenir un nombre positif et ce sont les résultats de ces complémentations qu'on multiplie. Le produit est un nombre positif dont le bit de signe est 0.
-Quand un des nombres est positif et que l'autre est négatif, le nombre négatif est d'abord complémenté à 2 pour obtenir une grandeur positive. Le produit est exprimé selon la notation en grandeur exacte. Cependant, le produit doit être négatif car les nombres à multiplier sont de signes opposés. Par conséquent, on complémente à 2 le produit et on ajoute le bit de signe 1.

Division binaire

La division d'un nombre binaire (le dividende) par un autre (le diviseur) est identique à la division de deux nombres décimaux. En réalité, la division en binaire est plus simple puisque pour déterminer combien de fois le diviseur entre dans le dividende, il n'y a que 2 possibilités 0 ou 1.
Voici deux exemples de divisions:

Ex : Division

```
  101100  |  100
- 100     |------
   11     |  1011
   110
  -100
    100
   -100
      0
```

Sens de lecture →

Dans la plupart des ordinateurs modernes, les soustractions qui ont lieu durant une opération de division sont généralement des soustractions avec complément à 2, c'est-à-dire on complémente à 2 le soustracteur puis on effectue l'addition.
La division de nombres signés s'effectue de la même façon que la multiplication.
Les nombres négatifs sont complémentés pour obtenir des nombres positifs puis la division est effectuée. Si le dividende et le diviseur sont de signes opposés, le quotient est complémenté à 2 pour obtenir un nombre négatif, puis on lui ajoute un bit de signe de 1. Si le dividende et le diviseur ont le même signe, le quotient est laissé sous sa forme positive et on lui ajoute un bit de signe de 0.

Opérations arithmétiques avec les nombres signés (Résumé)

– On supposera que les nombres sont représentés en utilisant le système complément à 2

1 Addition : Quatre cas sont possibles :

1. Les deux nombres sont positifs
2. Le nombre positif est plus 'grand' que le nombre négatif
3. Le nombre négatif est plus 'grand' que le nombre positif
4. Les deux nombres sont négatifs

Les deux nombres sont positifs :

– Dans ce cas-là on effectue une addition binaire classique

Le nombre positif est plus 'grand' que le nombre négatif

– On effectue une addition de binaire classique
– On 'oublie' la dernière retenue (à gauche)
– La somme est positive

Le nombre négatif est plus 'grand' que le nombre positif

– On effectue une addition binaire classique
– La somme est négative et représentée directement dans le système complément à 2

Les deux nombres sont négatifs

– On effectue une addition binaire classique
– On oublie la dernière retenue la plus à gauche
– Le résultat est négatif et déjà représenté dans le système complément à 2 !

1.1 Overflow / dépassement de capacité :

– On parle d'overflow quand il y a dépassement de la capacité pour représenter le résultat d'une somme
– Quand il y a overflow la somme n'est pas de même signe que les opérandes
– Cela peut se produire uniquement si les deux opérandes sont de mêmes signes

2 Soustraction

– La soustraction est considérée comme un cas particulier de l'addition :
– A - B = A + (-B)
– -A - B = (-A) + (-B)
– On prend donc le système complément à deux pour représenter (-B)
– On effectue une addition

3 Multiplication

– Les deux nombres doivent être représentés dans une forme sans complément (i.e valeur absolue)
– On effectue la multiplication
– On décide du signe du résultat :
– Les opérandes sont de mêmes signes : le résultat est positif
– Les signes des opérandes sont de signes différents : le résultat est négatif
– Si le signe du résultat doit être négatif ; il faut le représenter avec son complément à 2

4 Division

Division

Les deux nombres doivent être représentés dans une forme sans complément

1. Déterminer si le dividende et le diviseur sont de mêmes signes ou de signes différents. Ceci va déterminer le signe du quotient ; Initialiser le quotient à zéro.
2. Soustraire le diviseur du dividende en utilisant l'addition avec complément à deux pour obtenir le premier reste partiel ; incrémenter le quotient de 1. Si le reste partiel est positif aller à l'étape trois. Si le reste partiel est zéro ou négatif la division est terminée.
3. Soustraire le diviseur du reste partiel et incrémenter le quotient de 1. Si le résultat est positif, répéter l'opération pour le reste partiel trouvé. Si le résultat est zéro ou négatif la division est terminée.

soustractions successives
exemple $15_{10}/3_{10}$

1111	11
01	101
11	
0	

```
  1111
- 11      1
  1100
- 11      10
  1001
- 11      11
  0110
- 11      100
  0011
- 11      101
  0
```

Conclusion :

En complément à 1 ou à 2 , les opérations arithmétique sont avantageuse. En effet , la soustraction d'un nombre se réduit à l'addition de son complément. Des circuits réalisant des soustractions sont inutile et il n'y a pas de traitement particulier pour le bit de signe.

Si l'on trouve presque toujours la complémentation, le choix entre complément à 1 ou à 2 , chaque méthode ses avantages et ses inconvénients, est une question de compromis et dépend du constructeur .

Représentation des nombres réels

Les formats de représentations des nombres réels sont :

Format virgule fixe

- Utilisé par les premières machines
- Possède une partie 'entière' et une partie 'décimale' séparés par une virgule. La position de la virgule est fixe .

•Exemple : $54,25_{(10)}$; $10,001_{(2)}$; $A1,F0B_{(16)}$

- Etant donné une base b, un nombre x est représenté, en format virgule fixe, par :
  - $x = a_{n-1}a_{n-2}\ldots a_1a_0,a_{-1}a_{-2}\ldots a_{-p}\ (b)$
  - a_{n-1} est le chiffre de poids fort (MSB)
  - a_{-p} est le chiffre de poids faible (LSB)
  - n est le nombre de chiffre avant la virgule
  - p est le nombre de chiffre après la virgule
  - La valeur de x en base 10 est : $x = \sum_{-p}^{n-1} a_i b^i\ (10)$
  - Exemple :

$$101,01_{(2)} = 1.2^2+0.2^1+1.2^0+0.2^{-1}+1.2^{-2} = 5,25_{(10)}$$

Inconvénient de la méthode (virgule fixe) :

Problème de gestion de la virgule notamment dans les multiplications (pour les additions et soustraction pas problème , la position de la virgule ne change pas);

Format virgule flottante (utilise actuellement sur machine)

Le codage en complément à deux sur n bits ne permet de représenter qu'un intervalle de 2^n valeurs. Pour un grand nombre d'applications, cet intervalle de valeurs est trop restreint. La représentation à virgule flottante (floating-point) a été introduite pour répondre à ce besoin. Pour des mots de 32 bits, la représentation en complément à deux permet de coder un intervalle de 2^{32} valeurs tandis que la représentation à virgule flottante permet de coder un intervalle d'environ 2^{255} valeurs.

Le comité IEEE (Institue of Electrical and Electronics Engineers) a définie le standard IEEE-754 pour les calculs arithmétique flottants , afin que les programmes aient un comportement identique d'une machine à l'autre. Plus précisément, les buts de cette norme sont :

• de définir la représentation des nombres,
• de définir le comportement en cas de dépassement de capacité (exceptions),

• de garantir un arrondi exact pour les opérations élémentaires (+,-,*,/,√).

Afin de représenter les très grands ou très petits nombres sans s'encombrer d'une floppée de zéros, nous utilisons généralement une méthode assez pratique appelée ***notation scientifique.***

Celle-ci consiste à écrire les nombres sous la forme:

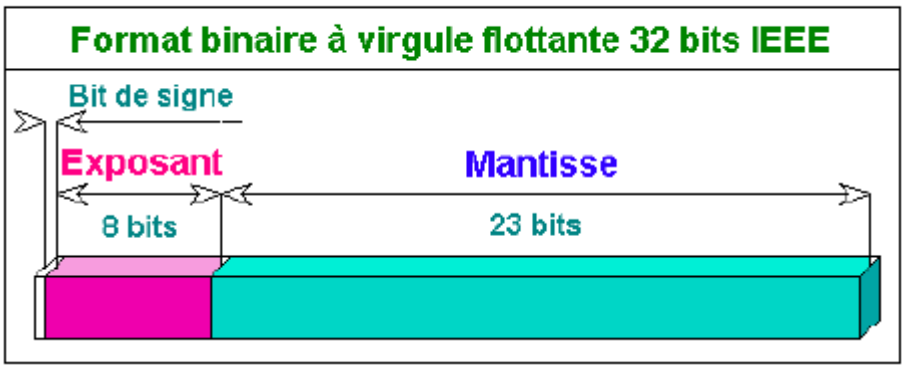
$$valeur = signe \times mantisse \times 10^{\pm\, Exposant\ réel}$$

Bien évidemment, la chose reste tout à fait possible en binaire, la notation devient:

$$valeur = signe \times mantisse \times 2^{\pm\, Exposant\ réel}$$

La norme IEEE 754 normalise cette représentation. Elle distingue 2 représentations, celle en simple précision (sur 32 bits), et celle en double précision (sur 64 bits).

Représentation en simple précision :



Chaque nombre réel peut s'écrire sous la forme :

$$valeur = signe \times 1,mantisse \times 2^{décalage}$$

$$= signe \times 1,mantisse \times 2^{Exposant\ -calage}$$

avec :

calage = $2^{k-1} - 1$

k nombre de bits attribués à l' Exposant

Exposant = Exposant réel + calage.

Un nombre flottant normalisé a une valeur v donnée par la formule suivante : $v = s \times 2^e \times m$.

- s = ±1 représente le signe (selon le bit de signe) ;
- e est l'exposant avant son décalage de 127 ;
- m = 1+mantisse représente la partie significative (en binaire), d'où $1 \leq m < 2$ (mantisse étant la partie décimale de la partie significative, comprise entre 0 et 1)

Explication pour le le décalge :

Avec 4bits , par exemple , on peut représenter 2^4 = 16 valeurs de Exposant (*e*) , qui vont de 0 à 15 . On peut faire correspondre les 8 premières valeurs (de 0 à 7) à un exposant < 0 , et les 8 suivantes (de 8 à 15) à un exposant ≥ 0 . Un exposant nul est donc représenté par la valeur 8 , un exposant égale à +1 par la valeur 9 , un exposant égale à -1 par la valeur 7 . On dit que le biais est égale à 8 . C'est la valeur qu'il faut soustraire à l'exposant biaisé (de 0 à 15) pour obtenir l'exposant réel (de – 8 à 7).

Exemple 1: Traduisons en binaire format flottant simple précision *32 bits* le nombre : - **1039,0**

Intéressons nous tout d'abord de sa valeur absolue 1039,0.
Pour traduire ce nombre (il est entier dans ce premier exemple) en binaire :

$1039_{\text{décimal}} = 0000\ 0100\ 0000\ 1111_{\text{binaire}}$

Nous constituons la mantisse : **1,** ***mantisse***

0000 0100 0000 1111 = 1,**00 0000 1111** . 2^{10} *(2^{10} opère un décalage de dix chiffres vers la droite après la virgule)*

Nous étendons la partie fractionnaire à 23 bits

1,00 0000 1111 = 1,00 0000 1111 0000 0000 0000 0
soit, rangé autrement : 1,000 0001 1110 0000 0000 0000
mantisse sur 23 bits = **000 0001 1110 0000 0000 0000** *(on ne mémorise pas le 1 implicite d'avant la virgule)*

Nous constituons le calage IEEE en simple précision 8 bits : $2^{8-1} - 1 = \mathbf{127}$

Nous constituons l'exposant : **exposant** = 10 + calage = **137**

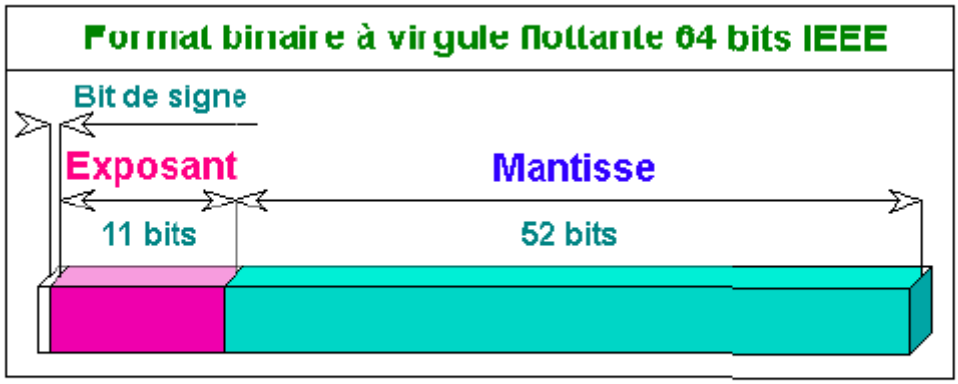
$137_{\text{décimal}} = 1000\ 1001_{\text{binaire}}$

Voici le résultat : bit de signe - exposant - mantisse

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

En hexadécimal C4 81 E0 00

Représentation en double précision :



On utilise la même méthode que pour le codage en simple précision, sauf que l'exposant est codé sur 11 bits et la mantisse sur 52 bits.

Les opérations arithmétiques en virgule flottante :

Pour la multiplication; il suffit d'additionner les exposants, de multiplier les mantisses et de renormaliser le résultat si nécessaire .

Exemple : $(0,2*10^{-3})*(0,3*10^{7})$ = ?

Addition des exposants : -3+7= 4
Multiplication des mantisses : 0,2*0,3=0,06
Résultat avant normalisation : $0,06*10^{4}$
Résultat normalisé : $0,6*10^{3}$

Pour la division , il faut soustraire les exposants et diviser les mantisses , puis normaliser si nécessaire.

La addition , exige que les exposants aient la même valeur ; on est donc obligé de dénormaliser la plus petite valeur pour amener son exposant à la même valeur que celui du plus grand nombre , Après avoir additionné les mantisses, une renormalisation peut s'avérer nécessaire.

La soustraction s'effectue comme l'addition, sauf que l'on doit effectuer la soustraction et non plus l'addition des mantisses.

Les différents codages

Le code BCD

Pour éviter une procédure complexe de conversion, on utilise souvent un codage chiffre à chiffre du nombre décimal.

Le BCD n'est pas un système de numération mais un code.

BCD est l'appellation anglaise (Binary Code Decimal). En français, on l'appelle code DCB (Décimal Codé en Binaire).

C'est donc un code qui permet de remplacer directement chaque chiffre d'un nombre décimal par son équivalent binaire sur 4 bits (quartet).
Le code BCD est utilisé dans les systèmes d'affichage de chiffres décimaux.

Décimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Exemple : $(129)_{10} = (\ ?\ )_{BCD}$

1				2				9			
8	4	2	1	8	4	2	1	8	4	2	1
0	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**

$(129)_{10} = (\ \mathbf{0001\ 0010\ 1001}\ )_{BCD}$

Comme les chiffres décimaux vont de 0 à 9, on utilisera les quartets de $0000_{(2)}$ à $1001_{(2)}$.
Les quartets $1010_{(2)}$, $1011_{(2)}$, $1100_{(2)}$, $1\ 101_{(2)}$, $1110_{(2)}$ et $1111_{(2)}$ ne sont donc jamais utilisés. Il faut en tenir compte pour les opérations arithmétiques.
Pour l'addition ;il faut ajouter 6 chaque fois que le résultat est > 9 .
Exemple :

Décimal	binaire	BCD
15	01111	0001'0101
+ 18	+ 10010	+ 0001'1000
--------	----------	----------
33	100001	0010'1101 > 9
	(=33)	+ 0110 + 6

		0011'0011 (=33)

Pour la soustraction , il faut retrancher 6 chaque fois que le résultat est < 0

Les opérations arithmétiques sont donc assez compliquées. Par contre , les opérations d'entrées/sorties sont faciles : chaque entité BCD est directement associée à un caractère .

Attention : ne pas confondre binaire et BCD.

Le code ASCII

L'un des codes les plus connus des utilisateurs de PC est le code ASCII (American Standard Code for Information Interchange). Il a été défini en 1963 aux Etats-Unis.
C'est un code à 7 bits autorisant donc le codage de 128 caractères parmi lesquels, tous les caractères alphanumériques utilisés en anglais. Chaque symbole d'un texte est représenté par une certaine chaîne de bits, un code

- Caractères anglais (lettre majuscules/minuscules), chiffres de 0 à 9
- Caractères de ponctuation, arithmétique, …
- Certains caractères de contrôle (fin de ligne, tabulateur, etc)

Le caractère A est codé 41 en hexadécimal et donc 01000001 en binaire. Son code décimal est 65.
Le caractère Z est codé 5A en hexadécimal et donc 01011010 en binaire. Son code décimal est 90.
Le caractère a est codé 61 en hexadécimal et donc 01100001 en binaire. Son code décimal est 97.

Les codes 0 à 31sont traditionnellement réservés à des caractères de contrôle dont l'existence est historique. Les plus connus d'entre eux sont certainement le 8 (BACKSPACE) et le 13 (CARRIAGE RETURN).

Le caractère $ est codé 24 en hexadécimal et donc 00100100 en binaire. Son code décimal est 36.

ASCII TABLE

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]

Comme la plupart des ordinateurs traitent les bits par paquets de huit ou plus, le huitième bit est, soit inutilisé, soit utilisé comme bit de parité pour le contrôle lors de la transmission , soit utilisé pour coder un maximum de 128 caractères supplémentaires tels des caractères graphiques et des caractères nationaux (caractères accentués). On parle alors de code ASCII étendu. . Mais, malheureusement, il n'y a pas une implémentation unique de cette solution.

Remarque : les chiffres sont codés de telle sorte que le quartet de poids faible représente la valeur du chiffre dans le système binaire.

	b7	b6	b5	b4	b3	b2	b1	b0
5	0	0	1	1	0	1	0	1

Tableau Code ASCII étendu, 8 bits

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	□	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	□	□
9	□	‘	’	“	”	•	–	—	˜	™	š	›	œ	□	□	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Conversion nombre ASCII en binaire :

1- ASCII à décimal : utiliser la table ASCII
2- Puis conversion de décimal à binaire.

Extended Binary Coded Decimal Interchange Code

L'**Extended Binary Coded Decimal Interchange Code** (EBCDIC) est un mode de codage des caractères sur 8 bits , soit 256 combinaisons possibles. créé par IBM à l'époque des cartes perforées.

Unicode

Unicode est une norme informatique, développée par le Consortium Unicode, qui vise à permettre le codage de texte écrit en donnant à tout caractère de n'importe quel système d'écriture un nom et un identifiant numérique, et ce de manière unifiée, quelle que soit la plate-forme informatique ou le logiciel. La dernière version, Unicode 6.1.0, est publiée depuis le 31 janvier 2012

Le standard Unicode est constitué d'un répertoire de plus de 110 000 caractères couvrant 100 écritures, d'un ensemble de tableaux de codes pour référence visuelle, d'une méthode de codage et de plusieurs codages de caractères standard, d'une énumération des propriétés de caractère (lettres majuscules, minuscules, symboles, ponctuation, etc.) d'un ensemble de fichiers de référence des données informatiques, et d'un certain nombre d'éléments liés, tels que des règles de normalisation, de décomposition, de tri, de rendu et d'ordre d'affichage bidirectionnel (pour l'affichage correct de texte

contenant contenant à la fois des caractères d'écritures droite à gauche, comme l'arabe et l'hébreu, et de gauche à droite).

Le **code de Gray**

Egalement appelé **binaire réfléchi**, est un type de codage binaire permettant de ne modifier qu'un seul bit à la fois quand un nombre est augmenté d'une unité. Le nom du code vient de l'ingénieur américain Frank Gray qui publia un brevet sur ce code en 1953, mais le code lui même est plus ancien; il a notamment été utilisé dans le code Baudot.

Méthode et codage (Gray)

Codage décimal	Codage binaire naturel	Codage Gray ou binaire réfléchi
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100