Chapitre 3 – Représentation des données

I. Introduction

A. Types de données à représenter

Nous manipulons constamment des informations : nos conversations, nos messages, sont des informations, qu'elles soient de nature orale ou écrite ou sous forme de signes, etc.

Avec l'usage de l'informatique, la notion d'information est réduite à un ensemble de données.
Dans une entreprise, les personnels des différents services sont amenés à manipuler des données numériques :

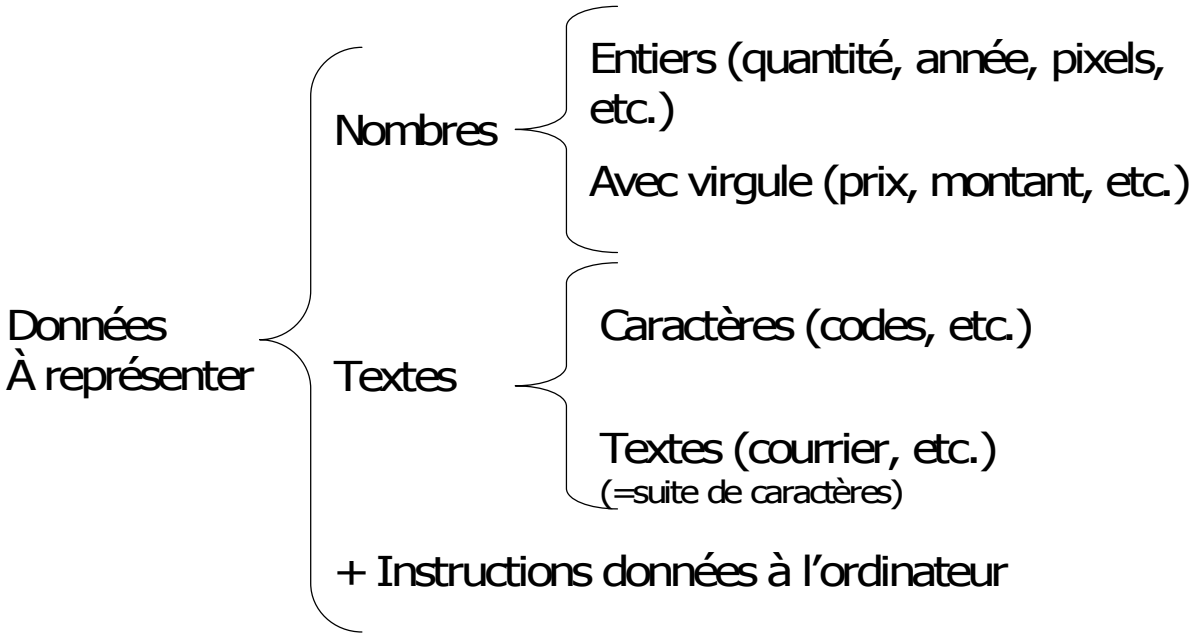
- les services commerciaux : gérer des fiches clients, les commandes, etc.
- les ateliers de productions : gérer des produits, des fabrications, etc.
- le secrétariat : courriers, présentations, agenda, etc.
- etc.

A notre domicile, nous manipulons des données numériques sous différentes formes :

- Images, sons, vidéos, etc.
- Courriers, messages (chat), etc.

L'analyse des données manipulées montre qu'elles sont en fait groupées en 2 familles principales :

- Les **nombres** : quantités, montants, pixels des images, échantillonnages des sons, etc.
- Les **textes** : caractères, rapports, courriers, etc.

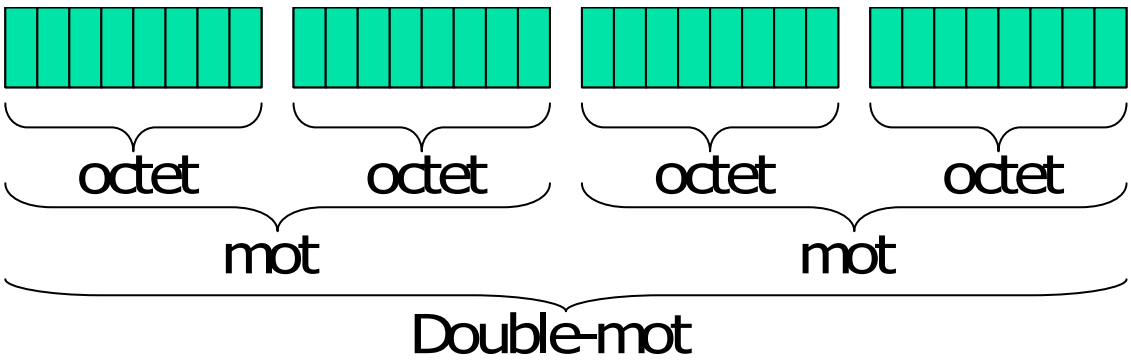


La représentation de ces données va nécessiter des méthodes de codage permettant leur traitement dans un « calculateur » muni d'un système de représentation binaire.
Nous verrons également que l'ordinateur nécessite également un codage pour identifier les instructions qu'il doit exécuter.

B. Les contraintes de codage : des cadres

L'ordinateur, machine avec une structure physique, par définition, possède des limites technologiques Ces limites ont amené ses concepteurs à définir des tailles de cellules permettant de contenir les données sous forme de chiffres binaires (*anglais : binary digits, BITS*). Les 'cadres' de stockage élémentaires sont les suivants :

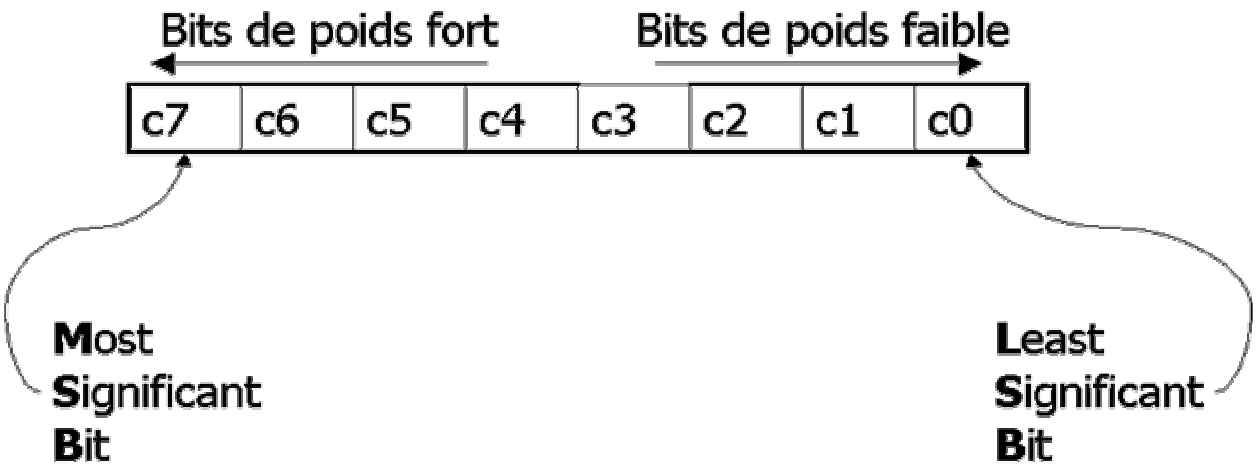
- **BIT** : information binaire élémentaire, capable de prendre 2 valeurs, 0 ou 1.
- **OCTET** (*anglais : byte, B*) : regroupe 8 bits
- **MOT** (*anglais : word, W*) : regroupe en général 2 octets
- **DOUBLE-MOT** (*anglais : double word, DW*) : regroupe 2 mots



Exemples de chiffres binaires (la séparation entre les groupes de 4 bits permet une meilleure lisibilité) :

- Bit : 1
- Octet : (1001 1101)
- Mot : (1000 1101 1111 0001)
- Double mot : (1000 1101 1111 0101 1010 1101 1011 0001)

L'octet (8 bits) est l'unité de base la plus utilisée pour définir le stockage et la manipulation des bits.



Comme nous l'avons étudié précédemment, **chaque bit** d'un octet (ou d'un mot, ou d'un double-mot) **possède un poids** dans un nombre binaire **en fonction de son rang** :

- **le plus à droite possède le poids le plus faible** (bit de poids faible, le moins significatif, ***anglais : LSB, Least Significant Bit***),
- **le plus à gauche le poids le plus fort** (bit de poids fort, le plus significatif, ***anglais : MSB, Most Significant Bit***).

C. Echelles des capacités de stockage des données

Les capacités des ordinateurs pour stocker les données utilisent des échelles de grandeurs basées sur l'octet (tout comme pour le mètre, le gramme, etc.).

Echelles utilisées en Informatique

Les échelles de grandeurs	Puissance de 2	octets
1ko (kilo octet)	10	1 024
1Mo (méga octet)	20	1 048 576
1Go (giga octet)	30	1 073 741 824
1To (téra octet)	40	1 099 511 627 776

Discussion avec les organismes de normalisation où :

échelles	Puissances de 10	nombres
1kilo	3	1 000
1Méga	6	1 000 000
1Giga	9	1 000 000 000
1Téra	12	1 000 000 000 000

Depuis 1998, il a donc été décidé de définir des unités binaires (kilo binaire, méga binaire, etc.) ce qui donne le système d'unités suivant :

- 1 kibioctet (Kio) = 2^{10} octets = 1024 octets
- 1 mébioctet (Mio) = 2^{20} octets = 1024 Kio
- 1 gibioctet (Gio) = 2^{30} octets = 1024 Mio
- 1 tébioctet (Tio) = 2^{40} octets = 1024 Gio

II. Codage des instructions

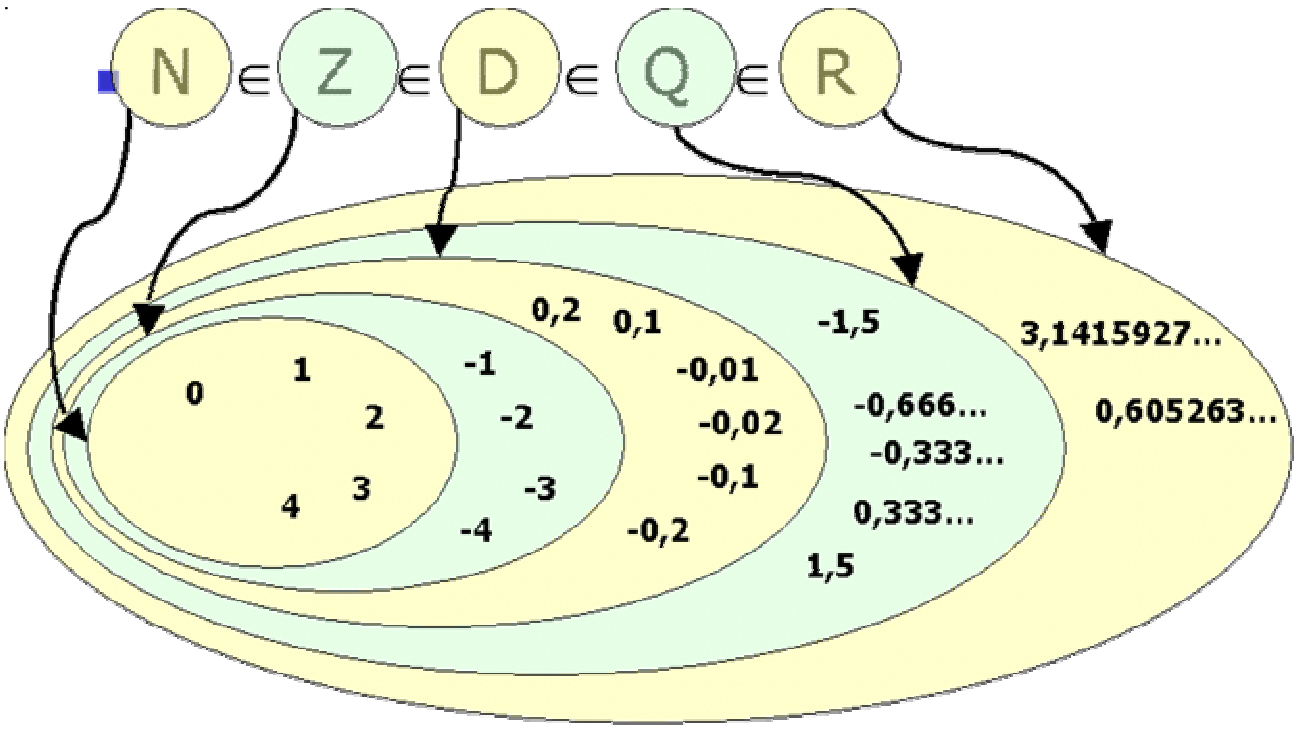
Les instructions permettent de demander à l'ordinateur d'exécuter certains traitement sur des données : un code instruction est donc une séquence binaire bien identifiée par l'ordinateur et qui lui permet de réaliser un certain traitement.
(➔ cette partie sera détaillée lors de l'étude du microprocesseur)

III. Codage des nombres entiers

Les nombres ont été les premiers éléments de données à être codés, l'ordinateur étant avant tout un calculateur (*anglais : computer*, *to compute*, calculer).

Il existe 2 grandes familles de nombres entiers, et une famille pour les nombres possédant une virgule :

- Les **nombres entiers naturels** (ensemble mathématique N) : nombres positifs {0, 1, 2, 3, etc.} (sauf les grands nombres)
- Les **nombres entiers relatifs** (ensemble mathématique Z) : inclus l'ensemble N et les nombres négatifs {..., -3, -2, -1, 0, 1, 2, 3, ...} (sauf les grands nombres)
- Les **nombres avec virgule et les grands nombres** (ensembles D, Q, et R et entiers en notation scientifique : mantisse X base^{exposant})



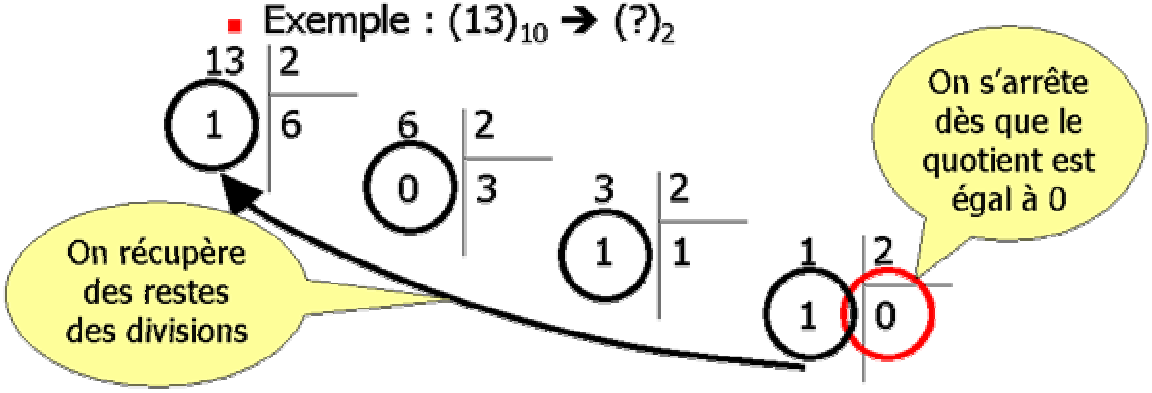
A. Nombres entiers naturels

Un nombre entier naturel va être simplement codé en binaire naturel, celui que nous avons utilisé dans la présentation des conversions entre bases de numération, du décimal au binaire
En fonction de sa grandeur, le codage d'un nombre entier naturel va nécessiter 1, 2 ou 4 octets et les 8, 16 ou 32 bits seront utilisés pour la représentation de ce nombre.

Coder un nombre entier naturel décimal en binaire : on utilise la technique étudiée dans les bases de numération (division par 2 ou méthode des poids).

Décoder un nombre entier naturel du binaire vers le décimal : on utilise la forme polynomiale (multiplication de chaque chiffre par son poids en décimal)

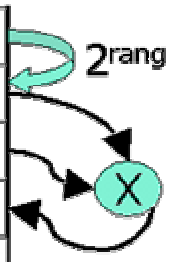
- Codage d'un entier naturel (décimal vers binaire)
  - Exemple : $(13)_{10}$ ➔ $(?)_2$



  - Résultat : $(0000\ 1101)_2$

- Décodage d'un entier naturel (binaire vers décimal)
  - Exemple : $(0000\ 1101)_2$➔ $(?)_{10}$

Rang	7	6	5	4	3	2	1	0
Poids	128	64	32	16	8	4	2	1
Symboles	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
Valeur	0	0	0	0	8	4	0	1
Résultat	Somme des valeurs = 13							



  - Résultat : $(13)_{10}$

B. Nombres entiers relatifs

Plusieurs techniques ont été utilisées pour coder les nombres nécessitant une position de signe. La plus utilisée est la technique du complément à 2.

Le codage d'un entier relatif sur 1 octet va utiliser 7 bits de l'octet pour représenter le nombre (chiffres binaires de c0 à c6) et le bit de poids fort (S) pour représenter le signe. Le bit de signe sera 0 pour un nombre positif et 1 pour un nombre négatif.
Si la grandeur de nombre à coder nécessite 2 ou 4 octets, c'est le bit de poids fort (le plus à gauche) qui représentera le signe.

S	c6	c5	c4	c3	c2	c1	c0

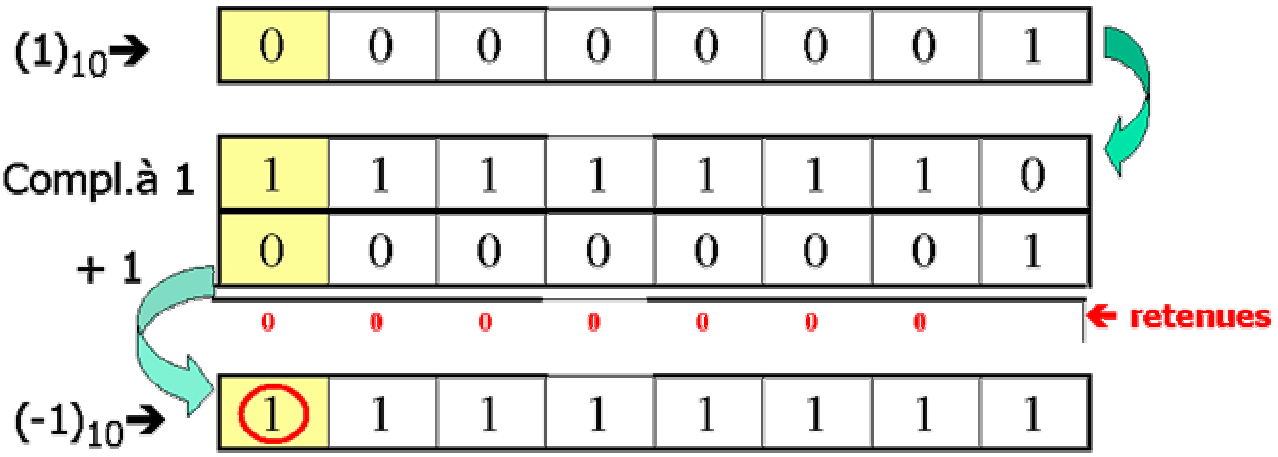
Pour **coder un nombre entier relatif décimal** en binaire :

- Le nombre à coder est positif : on le code comme un entier naturel
- Le nombre à coder est négatif : on code sa valeur absolue en binaire, comme un entier naturel, puis on réalise l'opération de complémentation à 2 : on inverse chaque bit (0 devient 1, 1 devient 0), puis on ajoute 1 au nombre binaire obtenu.

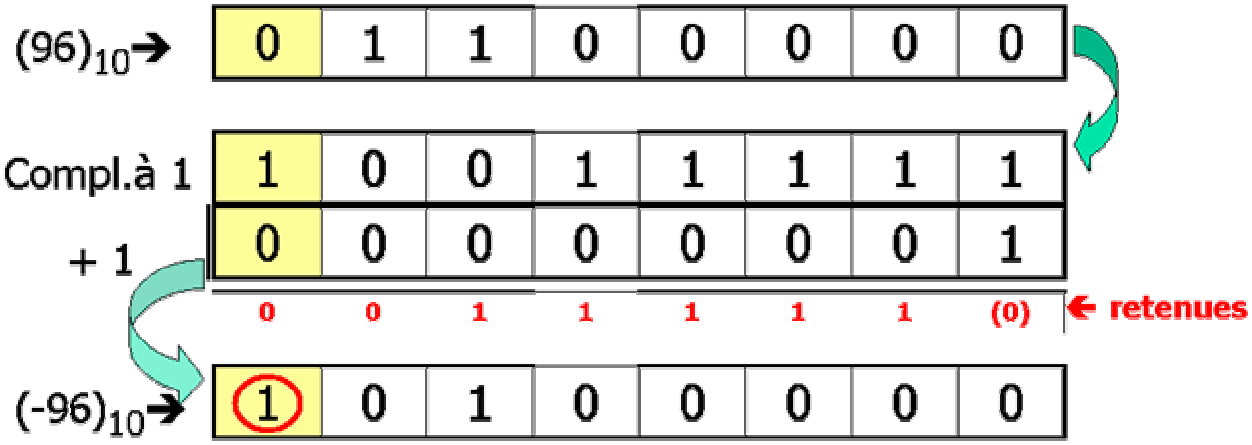
Pour **décoder un nombre entier relatif binaire** en décimal :

- Si le bit de poids fort est à 0 (le plus à gauche): ce nombre est positif, on le décode comme un entier naturel
- Si le bit de poids fort est à 1 : ce nombre est négatif ; on réalise l'opération de complément à 2 (inversion de chaque bit, puis ajout de 1) puis on utilise la forme polynomiale pour calculer la valeur de ce nombre (qui sera donc négatif)

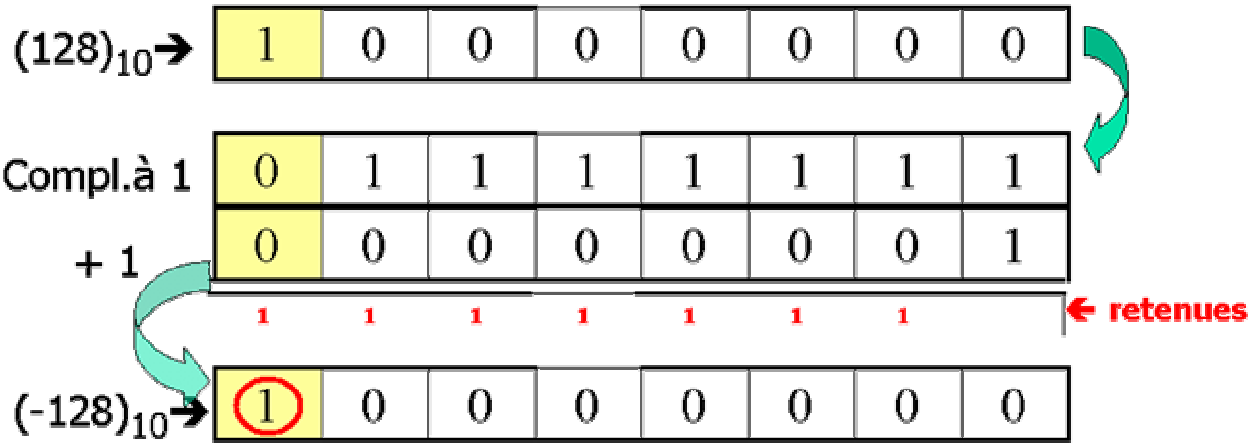
- Exemple :
  - $(-1)_{10}$ → $(?)_2$ en COMPLEMENT À 2 :



- Exemple :
  - $(-96)_{10}$ → $(?)_2$ en COMPLEMENT À 2 :



- Exemple :
  - $(-128)_{10}$ → $(?)_2$ en COMPLEMENT À 2 :



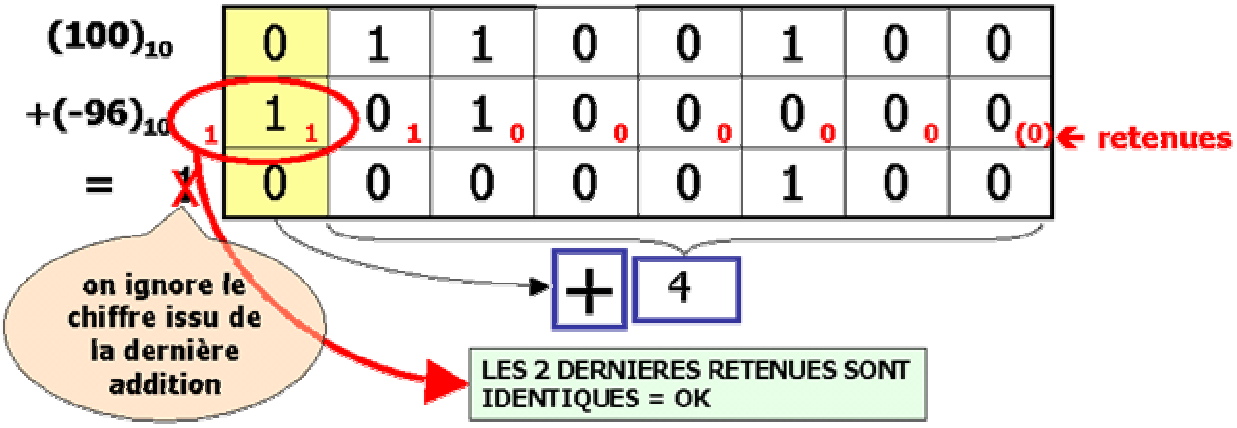
-128 est la plus petite valeur négative que l'on peut coder sur un octet. Coder -129 nécessitera 2 octets, la position de signe sera le bit de poids fort, celui le plus à gauche).

AVANTAGES DE LA CODIFICATION DU COMPLEMENT A 2

Les avantages liés à l'utilisation de cette technique de codification des nombres négatifs sont :

- Une seule valeur du 0
- Simplification des opérations arithmétiques : la soustraction disparaît au profit de l'addition ;

- $(100)_{10}$ ➔ $(0110\ 0100)_2$
- $(-96)_{10}$ ➔ $(1010\ 0000)_2$



PLAGES DE VALEURS ADMISSIBLES POUR LES ENTIERS

En fonction de la taille de la zone de stockage (octet, mot ou double-mot) et de la présence ou non d'une position de signe (entiers naturels ou entiers relatifs), on pourra coder les plages de valeurs suivantes :

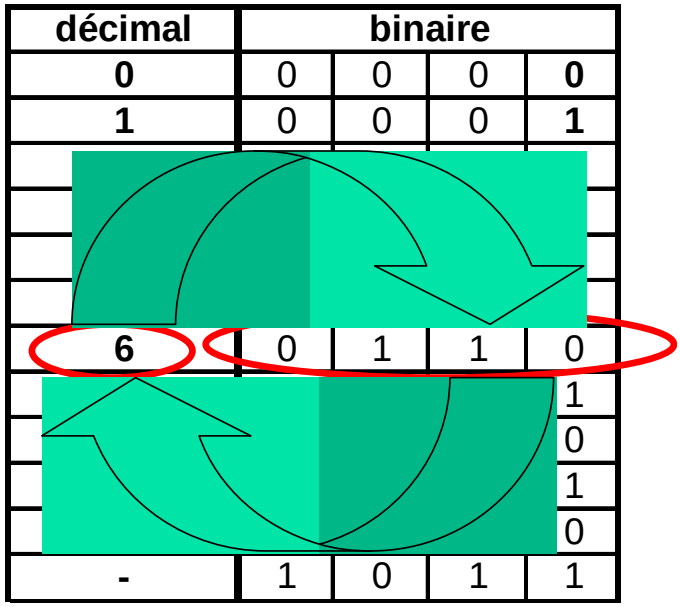
Taille du mot en octets (bytes)	nombre de bits	intervalle de valeurs
1	8	[0;+255]
1	1(S) + 7	[-128;+127]
2	16	[0;+65 535]
2	1(S) + 15	[-32 768;+32 767]
4	32	[0;4 294 967 295]
4	1(S) + 31	[-2 147 483 648;+2 147 483 647]

Les nombres entiers plus grands nécessiteront un codage sous la forme :

MANTISSE X BASE $^{\text{exposant}}$

C. Codage DCB

Le codage Décimal Codé Binaire (*anglais : BCD, Binary Coded Decimal*), permet la représentation de nombres grâce à un tableau de conversion. Le signe, puis chaque chiffre du nombre, sont codés sur 4 bits :



Par exemple : le nombre $(-1256)_{10}$ sera codé :

$$(1011\ 0001\ 0010\ 0101\ 0110)_{DCB}$$

IV. Codage des nombres fractionnaires

La représentation des nombres fractionnaire a utilisé plusieurs techniques de codification, celles les plus utilisées étant la représentation d'une position de virgule fixe et celle de la définition d'un format à virgule flottante.

A. Virgule fixe

La virgule fixe n'est pas définie dans le codage du nombre lui-même mais dans les programmes qui utilisent ce nombre. On parle aussi de virgule virtuelle.
Par exemple : le nombre 12345 avec une description 999V99 vaudra : 123,45.

B. Virgule flottante, format IEEE-754

La technique de codification des nombres à virgule flottante est la plus utilisée. Elle permet la représentation des nombres dans un grand intervalle de valeurs, avec cependant un inconvénient majeur : l'approximation du codage (et donc une perte d'information au décodage pour les nombres très grands ou très petits).
Ce codage utilise une structure bien déterminée :

- Sur 4 octets : nombres en simple précision (1 bit de signe, 8 bits pour l'exposant et 23 bits pour la mantisse)
- Sur 8 octets : nombres en double précision (1 bit de signe, 11 bits pour l'exposant et 52 bits pour la mantisse)

PRINCIPE DE CODAGE (simple précision)

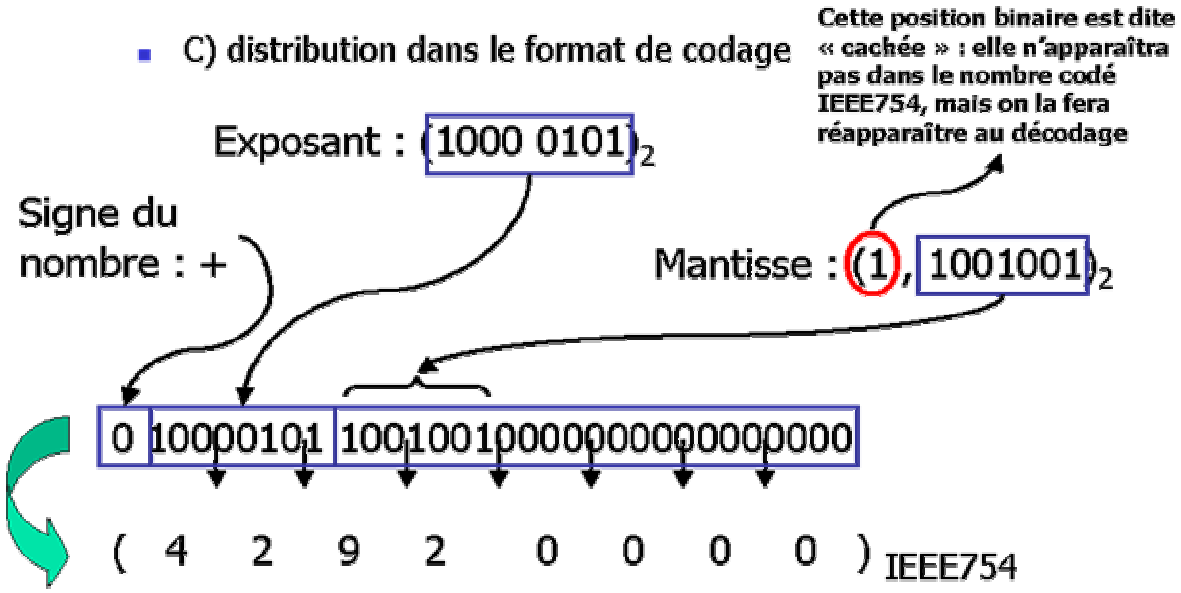
- Exemple : $(100,5)_{10}$ ➔ $(?)_{IEEE754}$

- Conversion en binaire :
  - $(100,5)_{10}$ ➔ $(1100100,1)_2$

a) Forme « Mantisse*Base^{Exposant} », soit en binaire « 1,...*$2^{Exposant}$ »
$(1100100,1)_2$ ➔ $(1,1001001)_2 * 2^6$

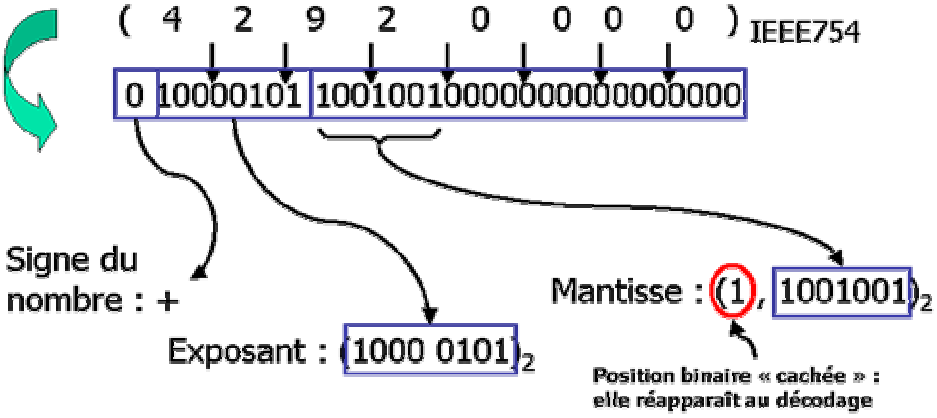
b) Conversion de l'exposant par rapport à 127 :
$(127+6)=133$ ➔ $(1000\ 0101)_2$

- C) distribution dans le format de codage



PRINCIPE DE DECODAGE

- Décodage



- Conversion de l'exposant par rapport à 127 :
  - $(1000\ 0101)_2$ ➔ 133
  - 133-127 = 6, valeur de l'exposant
- Forme « Mantisse*Base^{Exposant} », soit en binaire « 1,...*$2^{Exposant}$ »
  - $(1,1001\ 001)_2 * 2^6$ ➔ $(110\ 0100,1)_2$
- Conversion en décimal :
  - $(110\ 0100,1)_2$ ➔ $(100,5)_{10}$

V. Représentation et codage des caractères

Le codage des caractères nécessite un tableau de correspondance entre une séquence binaire (8 bits, par exemple) et le caractère que cette séquence va coder. Il existe de nombreuses tables de codification dont les principales sont ASCII et Unicode.

A. ASCII

C'est le système de codification le plus utilisée jusqu'à présent (*anglais : American Standard Code for Information Interchange*).
Il a été mis en œuvre dans les premiers ordinateurs pour coder les caractères alphanumériques utilisés alors en informatique (caractères anglo-saxons). La codification nécessitait alors 7 bits pour représenter 128 caractères (de 0 à 127 en décimal), le 8ème bit étant toujours à 0 :

- Caractères de contrôle des transmissions informatique
- Lettres : **A** à **Z** et **a** à **z** (A=65, a=97 soit A+32, 0=48, etc.)
- Chiffres : **0** à **9**
- Autres symboles (espace=20, Carriage Return =13, Line Feed=10, etc.)

La généralisation de l'utilisation de l'informatique a nécessité l'extension de cette table en utilisant le 8ème bit, afin de pourvoir coder les caractères européens (et les autres...) :

- Les pages de code (par exemple 850 utilisée sous DOS, en ligne de commande) associent spécifiquement par pays des numéros aux caractères spéciaux
- Les extensions iso-8859-1, ou latin-1, ou ANSI, permettent la représentation les caractères utilisés en Europe (lettres accentués, en particulier), etc.

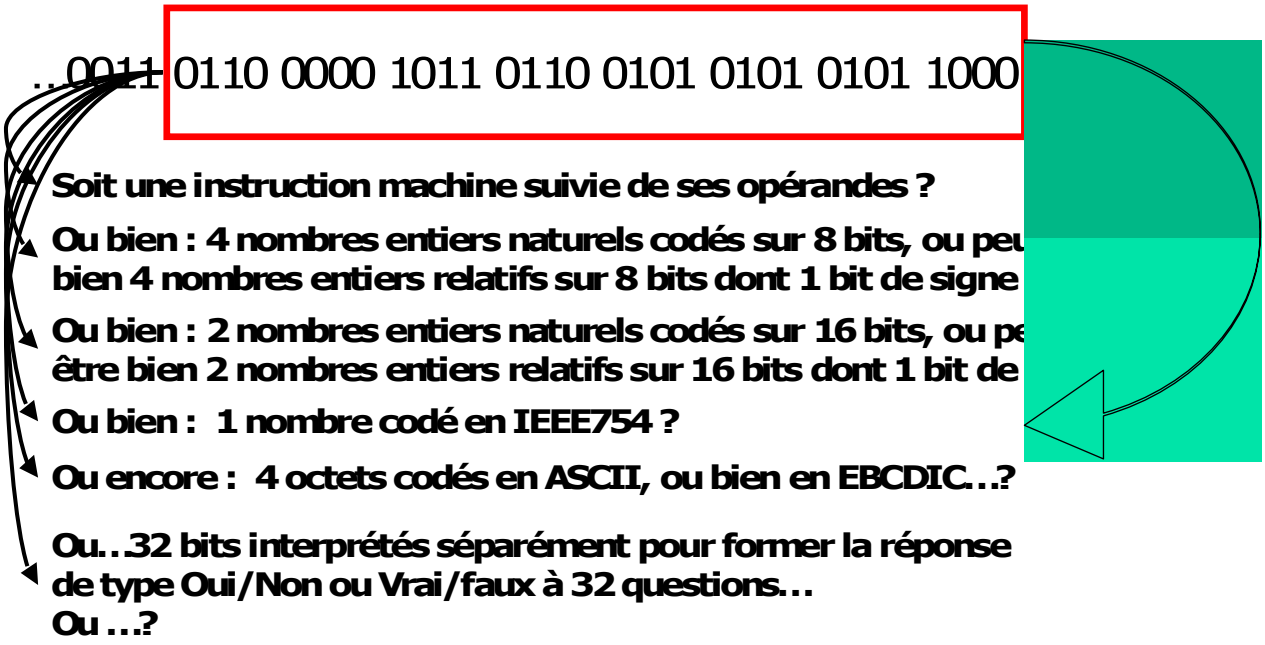
B. UNICODE

Unicode est une norme informatique, développée par le Consortium Unicode Site Web : http://www.unicode.org, dont l'objectif est d'attribuer à chaque caractère de tous les systèmes d'écritures, un nom et un identifiant numérique.

- UTF-8 : permet le codage des caractères sur un nombre de bits variables ; il est compatible avec l'ASCII de base ; chaque caractère est codé par une suite de 1 à 4 octets
- UTF-16 : chaque caractères est codé par 1 ou 2 mots de 16 bits
- UTF-32 : chaque caractères est codé sur 32 bits,

VI. Conclusion

Toute séquence binaire dans l'ordinateur peut représenter n'importe lequel des types de données présentés ci-dessus.
Le décodage passe donc forcément par la connaissance du type de données représenté par cette séquence binaire.



JE DOIS CONNAITRE LE « CADRE » ET « LE CODAGE ASSOCIÉ À CETTE SEQUENCE » POUR POUVOIR L'INTERPRETER, LA DECODER, LUI DONNER UN SENS