# CHAPITRE 1 : ENVIRONNEMENT MATÉRIEL ET LOGICIEL D'UN SYSTÈME INFORMATIQUE

## I. Architecture simplifiée d'un ordinateur :

En première approche, un ordinateur est constitué d'un processeur qui effectue les traitements, d'une mémoire centrale où ce processeur range les données et les résultats de ces traitements et de périphériques permettant l'échange d'informations avec l'extérieur. Tous ces constituants sont reliés entre eux par l'intermédiaire d'un bus, qui est l'artère centrale et leur permet de s'échanger des données. Pratiquement, tous les ordinateurs actuels ont cette architecture, que ce soient les micro-ordinateurs personnels ou les gros ordinateurs des entreprises. Les différences résident essentiellement dans les performances des constituants ***(Fig. 1)***.

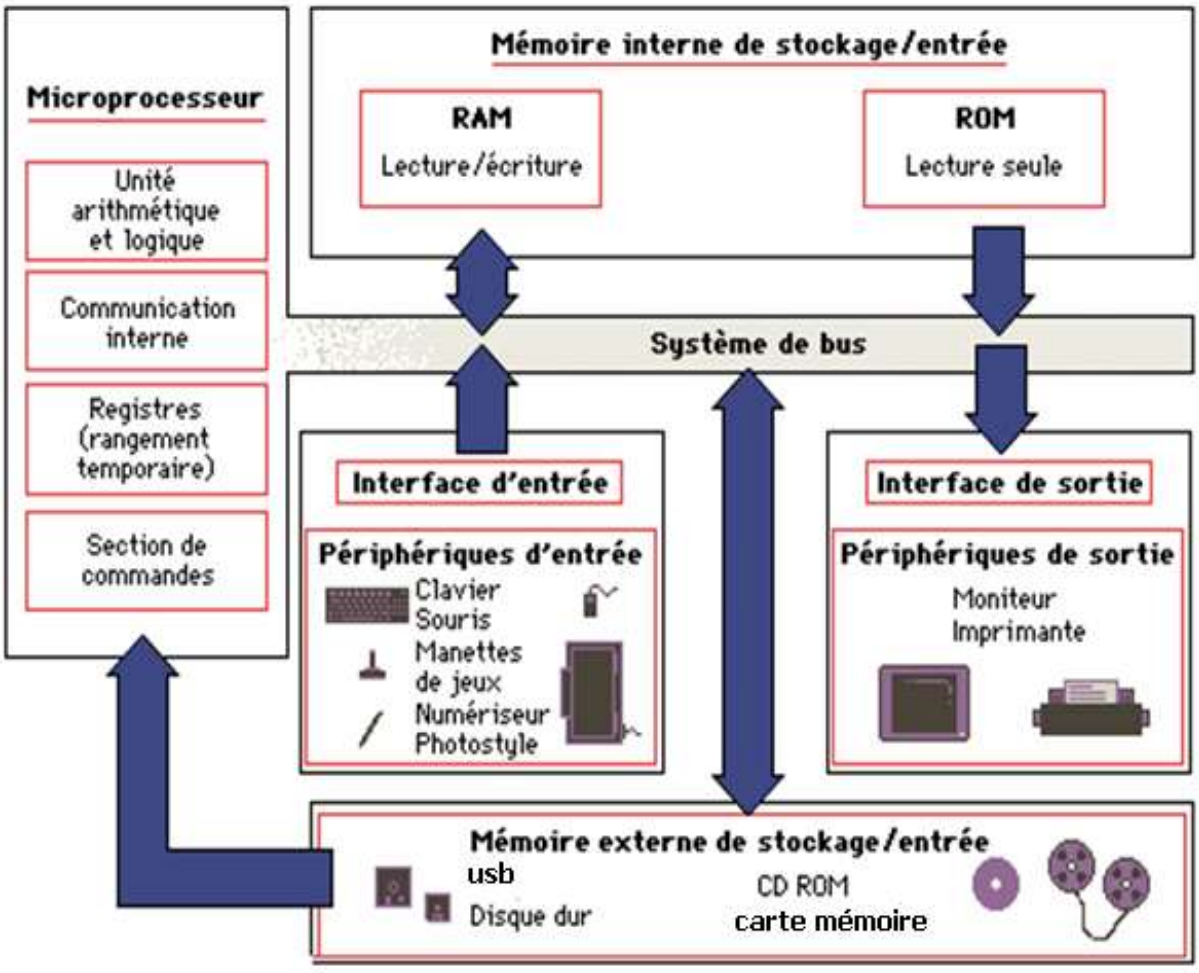



***Fig. 1 : Architecture générale d'un ordinateur***

Les informations traitées par les ordinateurs sont de différentes natures :

- nombres, texte,
- images, sons, vidéo,
- programmes, …

Dans un ordinateur, elles sont toujours représentées sous forme binaire (**BIT** : **B**inary dig**IT**) une suite de 0 et de 1.

Le codage de l'information 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.

Dans un ordinateur, un **mot mémoire** est l'unité de base manipulée par un microprocesseur. La taille d'un mot s'exprime en **bits** ou en **octets**, et est souvent utilisée pour classer les microprocesseurs (32 bits, 64 bits…). Toutes choses égales par ailleurs, un microprocesseur est d'autant plus rapide que ses mots sont longs, car les données qu'il traite à chaque cycle sont plus importantes.

Les ordinateurs modernes ou processeurs modernes utilisent généralement des données de 8, 16, 32 ou 64 bits, bien que d'autres tailles soient possibles. La nomenclature actuelle est comme suit :

- donnée de 8 bits : « **octet** », parfois abusivement « **byte** » ;
- donnée de 16 bits : « **word** » ou « **mot** »
- donnée de 32 bits : « **dword** » ou « **double mot** »
- donnée de 64 bits : « **qword** » ou « **quadruple mot** ».

## II. Représentation des nombres dans la mémoire de l'ordinateur

### II-1. Systèmes de numération :

On appelle système de numération tout système permettant d'écrire les nombres.

*Exemple :*

Le système décimal permet d'écrire un nombre à l'aide de dix chiffres 0 à 9. Tout nombre est représenté par une combinaison de ces dix symboles.

#### a. Définition d'une base :

La base d'un système de numération est la référence qui permet l'écriture d'un nombre. Dans le cas du système décimal, la base est **10**.

*Exemple :*

$$2148 = 2x10^3 + 1x10^2 + 4x10^1 + 8x10^0$$

En généralisant, un nombre A peut être exprimé dans une base b par n chiffres :

$$(A)_b = a_{n-1}\, a_{n-2} \ldots a_1\, a_0$$

- $a_i$ : est un chiffre de l'alphabet de poids i (position i).
- $a_0$ : chiffre de poids 0 appelé le chiffre de poids faible
- $a_{n-1}$ : chiffre de poids n-1 appelé le chiffre de poids fort

La valeur de A en base b est donnée par :

$$(A)_b = a_{n-1}.b^{n-1} + a_{n-2}.b^{n-2} + ... + a_0.b^0$$

*Remarque:*

La notation $(A)_b$ ou $A_b$ signifie que **A** est exprimé dans la base **b**.

### b. Système décimal (Base 10):

**C**'est le système de base 10 que nous utilisons tous les jours. Il comprend dix symboles différents: **(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)**

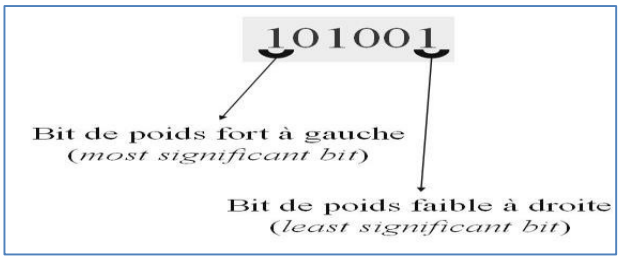
***Exemple :*** $(159)_{10}$.

**S**i nous employons le système décimal avec aisance dans tous nos calculs, il n'en va pas de même pour les machines électroniques dans lesquelles ce système reste difficile à mettre en œuvre pour plusieurs raisons d'ordre technique. C'est un autre système, plus adapté, qui est implanté **(Le système binaire)**.

### c. Système binaire (Base 2) :

**C**e système est employé pour traduire les états d'un système logique, il comprend deux symboles **(0 et 1)**. Chacun d'eux est aussi appelé bit **(Binary digIT = BIT)**.

**L**e bit le plus à gauche d'un nombre binaire est appelé ***bit de poids fort*** (Le plus significatif / **M**ost **S**ignificant **B**it), et le bit le plus à droite est appelé ***bit de poids faible*** (Le moins significatif / **L**ess **S**ignificant **B**it)).




### d. Les autres systèmes de numération utilisées :

- **Système octal (Base 8) .**

**C**e système comprend 8 symboles (0 à 7). Autrefois très utilisé, il tend aujourd'hui à disparaître au profit de la base 16 suite à l'évolution technologique des composants (16 bits et +)

***Exemple :*** $(374)_8$.

- **Système hexadécimal (Base 16) .**

**L**e système hexadécimal comporte 16 symboles : les dix chiffres 0 à 9 et les six lettres A, B, C, D, E et F. Ce système est très répandu pour la simple raison qu'il permet de représenter les nombres binaires d'une manière plus compacte

***Exemple :*** $(AC53)_{16}$.

*Problème :*

Le problème de conversion se pose alors du moment que la machine et l'homme parlent deux langages différents : La machine ne peut accepter un nombre décimal et l'homme de son côté reste incapable d'interpréter un résultat fourni par la machine dans sa base sans faire appel à une gymnastique lourde d'esprit.
Par conséquent, les règles de passage entre la base décimale et n'importe quelle base b ont été définies :

## II-2. Conversion :

### a. Passage d'une base B à la base décimale :

Soit A un nombre exprimé dans une base B :

$(A)_B = a_{n-1}a_{n-2}...a_0$ avec $0 \leq a_i \leq B-1$

La valeur décimale du nombre A est calculée par :

$$(A)_{10} = a_{n-1}.B^{n-1} + a_{n-2}.B^{n-2} + ... + a_0.B^0$$

$$(A)_{10} = \sum_{i=0}^{n-1} a_i B^i$$

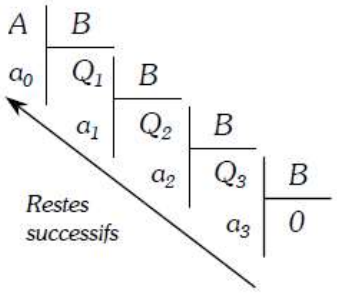
*Exemple :*

$(A2)_{16} = (\ ?\ )_{10} = 10x16^1 + 2x16^0 = 160 + 2 = (162)_{10}$
$(0101)_2 = 0x2^3+1x2^2+0x2^1+1x2^0 = (5)_{10}$

### b. Passage de la base décimale à une base B :

*Méthode :*

Diviser le nombre décimal à convertir (A) par la base B et conserver le reste de la division. Le quotient obtenu est divisé par B et conserver le reste. Il faut répéter l'opération sur chaque quotient obtenu. Les restes successifs sont écrits, en commençant par le dernier, de la gauche vers la droite pour former l'expression de $(A)_{10}$ dans le système de base B. Cette méthode est dite « **Méthode de la division successives** ».




$(A)_{10} = (a_3a_2a_1a_0)_B$

*Exemple :*

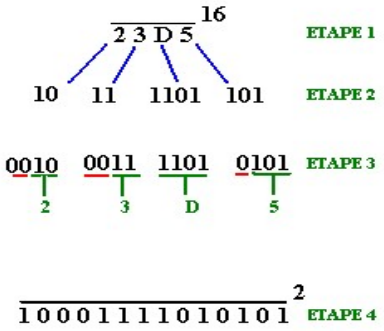
Convertir $A = (17)_{10}$ en binaire $A = (10001)_2$.

### c. Conversion d'un nombre hexadécimal en binaire :

Chaque symbole du nombre écrit dans le système hexadécimal est remplacé par son équivalent écrit dans le système binaire sous 4 bits.

*Exemple* :

Convertir le nombre hexadécimal suivant en binaire $(23D5)_{16}$




### d. Conversion d'un nombre binaire en hexadécimal :

C'est l'inverse de la précédente. Il faut donc regrouper les 1 et les 0 du nombre par 4 en commençant par la droite, puis chaque groupe est remplacé par le symbole hexadécimal correspondant.

*Exemple :*

Convertir $(1\ 1000\ 0110\ 1111)_2$ en système hexadécimal :

$(1\ 1000\ 0110\ 1111)_2 = 0001\ 1000\ 0110\ 1111 = (1\ 8\ 6\ F)_{16}$.

### e. Conversion d'un nombre binaire en octal :

Le passage du système binaire au système octal se fait par groupement de 3 bits.

*Exemple :*

- ✓ Soit $A = (1100110)_2$ un nombre écrit dans la base binaire.
  Convertir A en système octal :

$$(A)_2 = \underset{\downarrow\ \downarrow\ \downarrow}{1\,100\,110} \quad \text{donc} \quad (A)_8 = 146$$
$$1\ \ 4\ \ 6$$

- ✓ Soit $A = (253)_8$ un nombre écrit dans la base octale.
  Convertir A en système binaire :

$$(A)_8 = 253 \quad \text{d'où} \quad (A)_2 = 100101011$$

## II-3. Représentation des nombres entiers en mémoire :

### a. Représentation d'un entier naturel :

Un entier naturel est un nombre entier positif ou nul. Le choix à faire (c'est-a-dire le nombre de bits à utiliser) dépend de la fourchette des nombres que l'on désire utiliser. Pour coder des nombres entiers naturels compris entre 0 et 255, il nous suffira de 8 bits (un octet)

car $2^8=256$. D'une manière générale un codage sur **n** bits pourra permettre de représenter des nombres entiers naturels compris entre **0** et $2^n-1$.

*Exemples :*

Sur un **octet :**
$(9)_{10} = (00001001)_2$, $(128)_{10} = (10000000)_2$, etc.

### b. Représentation d'un entier relatif :

Un entier relatif est un entier pouvant être négatif. Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif, et il faut de plus que les règles d'addition soient conservées. L'astuce consiste à utiliser un codage que l'on appelle **complément à deux**. Cette représentation permet d'effectuer les opérations arithmétiques usuelles naturellement.

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

  D'une manière générale le plus grand entier relatif positif codé sur **n** bits sera $2^{n-1}-1$.

- **Un entier relatif négatif** sera représenté grâce au codage en complément à deux.

**Principe du complément à deux :**

1. Ecrire la valeur absolue du nombre en base 2. Le bit de poids fort doit être égal à 0.
2. Inverser les bits : les 0 deviennent des 1 et vice versa. On fait ce qu'on appelle le complément à un.
3. On ajoute 1 au résultat (les dépassements sont ignorés).

Cette opération correspond au calcul de $2^n-|x|$, où n est la longueur de la représentation et $|x|$ la valeur absolue du nombre à coder.
Ainsi sur un octet -1 s'écrit comme $2^8-1=255=11111111$

**Exemple 1:**

$-6 = (0110)_2$ avec précision de 4 bits
Complément à 1 : 1001
Complément à 2 : 1001 + 1 = 1010
La représentation binaire de -6 sur 4 bits est donc 1010.

**Exemple 2:**

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

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

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

**Exemple 3:**

Sur 4 bits

| *Décimal* | *Code compl. à 2* | *Décimal* | *Code compl. à 2* |
|---|---|---|---|
| 0 | 0000 | –8 | 1000 |
| 1 | 0001 | –7 | 1001 |
| 2 | 0010 | –6 | 1010 |
| 3 | 0011 | –5 | 1011 |
| 4 | 0100 | –4 | 1100 |
| 5 | 0101 | –3 | 1101 |
| 6 | 0110 | –2 | 1110 |
| 7 | 0111 | –1 | 1111 |

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

## II-4. Représentation des nombres réels en mémoire :

En base 10, l'expression 652,375 est une manière abrégée d'écrire :

$$6\cdot10^2 + 5\cdot10^1 + 2\cdot10^0 + 3\cdot10^{-1} + 7\cdot10^{-2} + 5\cdot10^{-3}.$$

Il en va de même pour la base 2. Ainsi, l'expression 110,101 signifie :

$$1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3}$$

### a. Conversion de binaire en décimal

On peut ainsi facilement convertir un nombre réel de la base 2 vers la base 10.

**Exemple :**

$$110{,}101_2 = 1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} = 6{,}625_{10}.$$

### b. Conversion de décimal en binaire

Le passage de base 10 en base 2 est plus subtil. Par exemple : convertissons 13,375 en base 2.

- ✓ La partie entière se transforme comme suite $13_{10} = 1101_2$
- ✓ On transforme la partie décimale selon le schéma suivant :

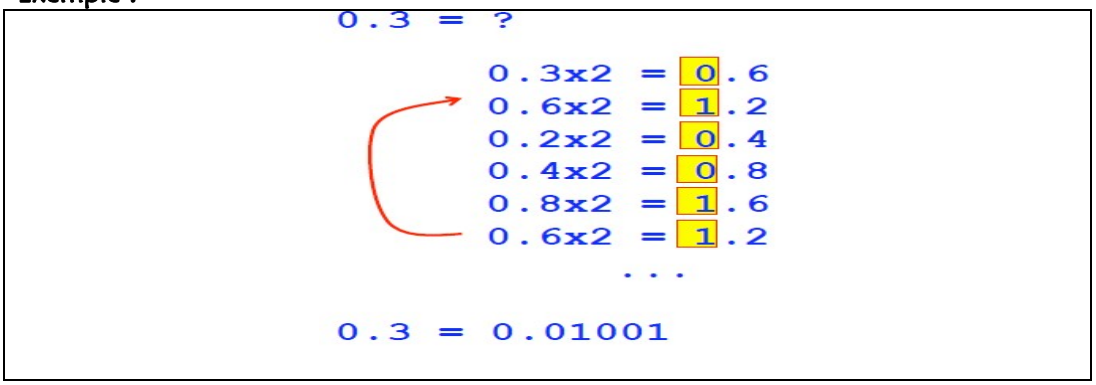
```
0.375 = ?

        0.375x2 = 0.75
        0.75x2 = 1.5
        0.5x2 = 1.0

0.375 = 0.011
```

- ✓ Donc la représentation en binaire de $(13{,}375)_{10}$ est $(1101{,}011)_2$:

Exemple :

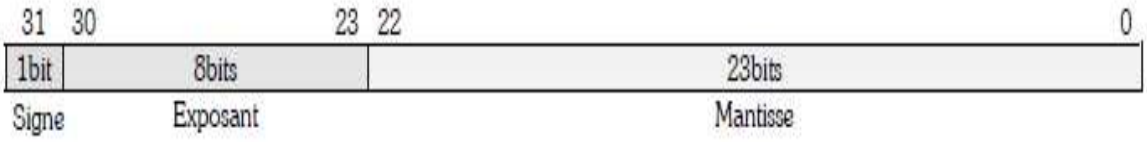



### c. Codification des nombres réels (norme IEEE 754)

Du moment que les nombres réels peuvent prendre des valeurs très grandes ou très petites, ils sont représentés en virgule flottante (floating point) selon le format normalisée IEEE.

Il existe deux formats IEEE pour les nombres à virgule flottante : le format 32 bits (simple précision) et le format 64 bits (double précision) :

Format 32 bits : Cette norme se propose de coder le nombre sur 32 bits et définit trois composantes :




- le signe est représenté par un seul bit, le bit de poids fort (bit 31)
- l'exposant est codé sur les 8 bits consécutifs au signe. Il est égal à **exposant+127**
- la mantisse (les bits situés après la virgule) sur les 23 bits restants

**Encodage en IEEE754 par l'exemple :**

Essayons d'encoder le nombre +16,5 en simple précision.

1. Encoder en binaire le nombre à virgule : +16,5 = $(10000,1)_2$
2. Transformer en notation scientifique (souvenez-vous que décaler la virgule revient à multiplier/diviser par 2) $(10000,1)_2 = (1,00001*2^4)_2$
3. Identifier les champs :
   signe = 0 (positif)
   exposant = 4
   mantisse = 00001 (on ne prend pas le 1,)
4. Encoder l'exposant 4 doit être codé sur 8 bits par excès de 127 => on encode 131 = $(1000\ 0011)_2$
5. Résultat final (En contrôle, écrire tous les 0 à la fin pour faire 32 bits) :

| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

6. En Héxa (ne pas oublier les 0 à la fin pour faire 32 bits) : 41 84 00 00

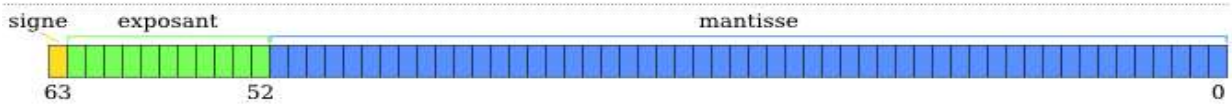
**Remarques**

- L'exposant 00000000 signifie que le nombre est dénormalisé (trop petit)
- L'exposant 11111111 signifie un dépassement de capacité (nombre trop grand : NaN (Not a Number))

Par conséquent, le plus petit exposant possible est 01 (m=–126) et le plus grand est FE (m=+127).

**Format 64 bits :** Cette norme se propose de coder le nombre sur 32 bits et définit trois composantes :

- le signe est représenté par un seul bit, le bit de poids fort (bit 63)
- l'exposant est codé sur les 11 bits consécutifs au signe. Il est égal à **exposant+1023**
- la mantisse (les bits situés après la virgule) sur les 52 bits restants




## III. limite de la représentation en mémoire :

### III-1. Les erreurs d'arrondi :

La représentation interne d'un nombre est limitée par le nombre de bits utilisés pour le stockage. Par suite, les erreurs de calcul de l'ordinateur peuvent être plus ou moins importantes suivant les machines. Il ne s'agit donc pas de bugs de votre machine ! Il faut savoir qu'un résultat affiché n'est pas celui qui est stocké dans les entrailles du microprocesseur.

Prenons le cas d'une calculatrice ou d'un micro-ordinateur de poche affichant 10 chiffres significatifs :

| Tapons : | On obtient : |
|---|---|
| 17/13 | 1.307692308 |
| 17/13-1 | 0,3076923077 |
| (17/13)*10-13 | 0,0769230769 |
| (17/130)*100-130 | 0,769230769 |

On voit que le dernier chiffre (8) obtenu est un arrondi (800) de 769 : plus clairement $8.10^{-9}$ est un arrondi d'affichage d'un calcul interne égal à $7,69.10^{-9}$. Ce qui signifie que, par précaution, ces ordinateurs n'affichent pas deux chiffres risquant d'être entachés d'erreurs : affichage de 10 chiffres significatifs en calculant avec 12.

### III-2. Dépassement de capacité (les "overflow")

Sur 4 bits, on veut soustraire 2 à 5 :

- Nous ferons l'addition de +5 (0101) et -2 (1110)
- Soit l'addition:

0101 + 1110 = 10011

- **Sans tenir compte du 5ème bit,** le résultat (0011) est positif et égal à +3

Une erreur a été effectuée par dépassement de la capacité du code (codage sur 4 bits), ce type d'erreur est appelée **Overflow**

## IV. Le codage des caractères :

L'homme utilise le plus souvent des informations sous forme de texte, de chiffres ou de symboles, c'est ce qu'on appelle en informatique des caractères (A,B,C,...,1,2,3,...,(,@,&,...). Cependant, la mémoire de l'ordinateur ainsi que les supports d'informations ne peuvent supporter que les valeurs 0 et 1. Il a donc fallu trouver un codage qui permettrait de représenter n'importe quel caractère sous forme d'un nombre binaire afin de faciliter son stockage et sa transmission.

### IV-1. Le code ASCII (American Standard Code for Information Interchange) :

Le code **ASCII** était parmi les premiers codes utilisés pour ce propos. C'est un code universel qui fait correspondre à chaque caractère un code sur 7 ou 8 bits (**voir Annexe 1**).

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 8 bits pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...). Cette norme s'appelle ISO-8859 et se décline par exemple en ISO-8859-1 lorsqu'elle étend l'ASCII avec les caractères accentués d'Europe occidentale, et qui est souvent appelée Latin-1 ou Europe occidentale.

### IV-2. l'UNICODE

Il existe d'autres normes que l'ASCII, comme **l'Unicode** par exemple, qui présent l'avantage de proposer une version unifiée des différents encodages de caractères complétant l'ASCII mais aussi de permettre l'encodage de caractères autres que ceux de l'alphabet latin. **Unicode** définit des dizaines de milliers de codes, mais les 128 premiers restent compatibles avec ASCII (**voir Annexe 2**).

### IV-3. Unicode dans la pratique: UTF-8

Généralement en Unicode, un caractère prend 2 octets. Autrement dit, le moindre texte prend deux fois plus de place qu'en ASCII. C'est du gaspillage.

De plus, si on prend un texte en français, la grande majorité des caractères utilisent seulement le code ASCII. Seuls quelques rares caractères nécessitent l'Unicode.

On a donc trouvé une astuce: l'**UTF-8** :

Un texte en UTF-8 est simple: il est partout en ASCII, et dès qu'on a besoin d'un caractère appartenant à l'Unicode, on utilise un caractère spécial signalant "attention, le caractère suivant est en Unicode".

# ANNEXE 1 : TABLE ASCII

le code ASCII

### ASCII control characters

| | | |
|---|---|---|
| 00 | NULL | (Null character) |
| 01 | SOH | (Start of Header) |
| 02 | STX | (Start of Text) |
| 03 | ETX | (End of Text) |
| 04 | EOT | (End of Trans.) |
| 05 | ENQ | (Enquiry) |
| 06 | ACK | (Acknowledgement) |
| 07 | BEL | (Bell) |
| 08 | BS | (Backspace) |
| 09 | HT | (Horizontal Tab) |
| 10 | LF | (Line feed) |
| 11 | VT | (Vertical Tab) |
| 12 | FF | (Form feed) |
| 13 | CR | (Carriage return) |
| 14 | SO | (Shift Out) |
| 15 | SI | (Shift In) |
| 16 | DLE | (Data link escape) |
| 17 | DC1 | (Device control 1) |
| 18 | DC2 | (Device control 2) |
| 19 | DC3 | (Device control 3) |
| 20 | DC4 | (Device control 4) |
| 21 | NAK | (Negative acknowl.) |
| 22 | SYN | (Synchronous idle) |
| 23 | ETB | (End of trans. block) |
| 24 | CAN | (Cancel) |
| 25 | EM | (End of medium) |
| 26 | SUB | (Substitute) |
| 27 | ESC | (Escape) |
| 28 | FS | (File separator) |
| 29 | GS | (Group separator) |
| 30 | RS | (Record separator) |
| 31 | US | (Unit separator) |
| 127 | DEL | (Delete) |

### ASCII printable characters

| | | | | | |
|---|---|---|---|---|---|
| 32 | space | 64 | @ | 96 | ` |
| 33 | ! | 65 | A | 97 | a |
| 34 | " | 66 | B | 98 | b |
| 35 | # | 67 | C | 99 | c |
| 36 | $ | 68 | D | 100 | d |
| 37 | % | 69 | E | 101 | e |
| 38 | & | 70 | F | 102 | f |
| 39 | ' | 71 | G | 103 | g |
| 40 | ( | 72 | H | 104 | h |
| 41 | ) | 73 | I | 105 | i |
| 42 | * | 74 | J | 106 | j |
| 43 | + | 75 | K | 107 | k |
| 44 | , | 76 | L | 108 | l |
| 45 | - | 77 | M | 109 | m |
| 46 | . | 78 | N | 110 | n |
| 47 | / | 79 | O | 111 | o |
| 48 | 0 | 80 | P | 112 | p |
| 49 | 1 | 81 | Q | 113 | q |
| 50 | 2 | 82 | R | 114 | r |
| 51 | 3 | 83 | S | 115 | s |
| 52 | 4 | 84 | T | 116 | t |
| 53 | 5 | 85 | U | 117 | u |
| 54 | 6 | 86 | V | 118 | v |
| 55 | 7 | 87 | W | 119 | w |
| 56 | 8 | 88 | X | 120 | x |
| 57 | 9 | 89 | Y | 121 | y |
| 58 | : | 90 | Z | 122 | z |
| 59 | ; | 91 | [ | 123 | { |
| 60 | < | 92 | \ | 124 | \| |
| 61 | = | 93 | ] | 125 | } |
| 62 | > | 94 | ^ | 126 | ~ |
| 63 | ? | 95 | _ | | |

### Extended ASCII characters

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 128 | Ç | 160 | á | 192 | └ | 224 | Ó |
| 129 | ü | 161 | í | 193 | ┴ | 225 | ß |
| 130 | é | 162 | ó | 194 | ┬ | 226 | Ô |
| 131 | â | 163 | ú | 195 | ├ | 227 | Ò |
| 132 | ä | 164 | ñ | 196 | ─ | 228 | õ |
| 133 | à | 165 | Ñ | 197 | ┼ | 229 | Õ |
| 134 | å | 166 | ª | 198 | ã | 230 | µ |
| 135 | ç | 167 | º | 199 | Ã | 231 | þ |
| 136 | ê | 168 | ¿ | 200 | ╚ | 232 | Þ |
| 137 | ë | 169 | ® | 201 | ╔ | 233 | Ú |
| 138 | è | 170 | ¬ | 202 | ╩ | 234 | Û |
| 139 | ï | 171 | ½ | 203 | ╦ | 235 | Ù |
| 140 | î | 172 | ¼ | 204 | ╠ | 236 | ý |
| 141 | ì | 173 | ¡ | 205 | ═ | 237 | Ý |
| 142 | Ä | 174 | « | 206 | ╬ | 238 | ¯ |
| 143 | Å | 175 | » | 207 | ¤ | 239 | ´ |
| 144 | É | 176 | ░ | 208 | ð | 240 | ≡ |
| 145 | æ | 177 | ▒ | 209 | Ð | 241 | ± |
| 146 | Æ | 178 | ▓ | 210 | Ê | 242 | ‗ |
| 147 | ô | 179 | │ | 211 | Ë | 243 | ¾ |
| 148 | ö | 180 | ┤ | 212 | È | 244 | ¶ |
| 149 | ò | 181 | Á | 213 | ı | 245 | § |
| 150 | û | 182 | Â | 214 | Í | 246 | ÷ |
| 151 | ù | 183 | À | 215 | Î | 247 | ¸ |
| 152 | ÿ | 184 | © | 216 | Ï | 248 | ° |
| 153 | Ö | 185 | ╣ | 217 | ┘ | 249 | ¨ |
| 154 | Ü | 186 | ║ | 218 | ┌ | 250 | · |
| 155 | ø | 187 | ╗ | 219 | █ | 251 | ¹ |
| 156 | £ | 188 | ╝ | 220 | ▄ | 252 | ³ |
| 157 | Ø | 189 | ¢ | 221 | ¦ | 253 | ² |
| 158 | × | 190 | ¥ | 222 | Ì | 254 | ■ |
| 159 | ƒ | 191 | ┐ | 223 | ▀ | 255 | nbsp |