# Architecture des ordinateurs

## Représentation des données

Anicet E. T. Ebou, ediman.ebou@inphb.ci

# Introduction

- Les interfaces d'entrées/sorties transforment les données intelligible vers/depuis une suite de 0 et de 1, c'est à dire une suite de **bits**.
- Le bit est l'unité fondamentale d'information dans l'ordinateur.
- Un mot de format **8 bits** est un **octet**.
- Les informations sont souvent représentées en binaire sous la forme de mots dont le format est multiple d'un octet (8 bits, 16 bits, 32 bits, …).

# 01 Représentation des nombres

# 1.1 Introduction

# Système de numération

Un système de numération est un ensemble de règles permettant de représenter les nombres.

Dans les systèmes numériques, on utilise principalement les systèmes suivants:

- Le système de numération **décimale** utilise les dix chiffres **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**.
- Le système de numération **binaire** utilise exclusivement les deux chiffres **0** et **1**. (que l'on appelle alors ***bit*** – binary digit).

# Système de numération

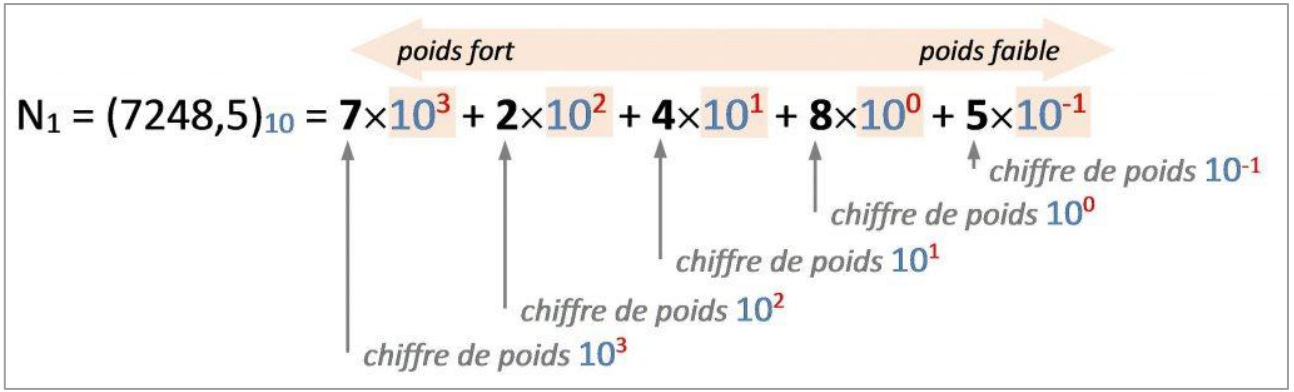
- Le système de numération **hexadécimale** utilise les seize chiffres **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.**

Par convention, l'écriture d'un nombre **N** s'effectue par la juxtaposition de **chiffres** possédant chacun un **poids** égal à une puissance entière de la base de numération. Les chiffres s'écrivent de la gauche vers la droite par valeur décroissante de leur poids.

# Système de numération

Le poids d'un chiffre dépend donc de son rang et du système de numération adopté :

En base 10 – écriture décimale :



# Système de numération

En bases 2 et 16 – écritures binaire et hexadécimale :

$$N_2 = (1\,0\,0\,1\,1\,0)_2 = \mathbf{1}\times 2^5 + \mathbf{0}\times 2^4 + \mathbf{0}\times 2^3 + \mathbf{1}\times 2^2 + \mathbf{1}\times 2^1 + \mathbf{0}\times 2^0 = (38)_{10}$$

$$N_3 = (E7B)_{16} = \mathbf{14}\times 16^2 + \mathbf{7}\times 16^1 + \mathbf{11}\times 16^0 = (3707)_{10}$$

# Base d’un système de numération

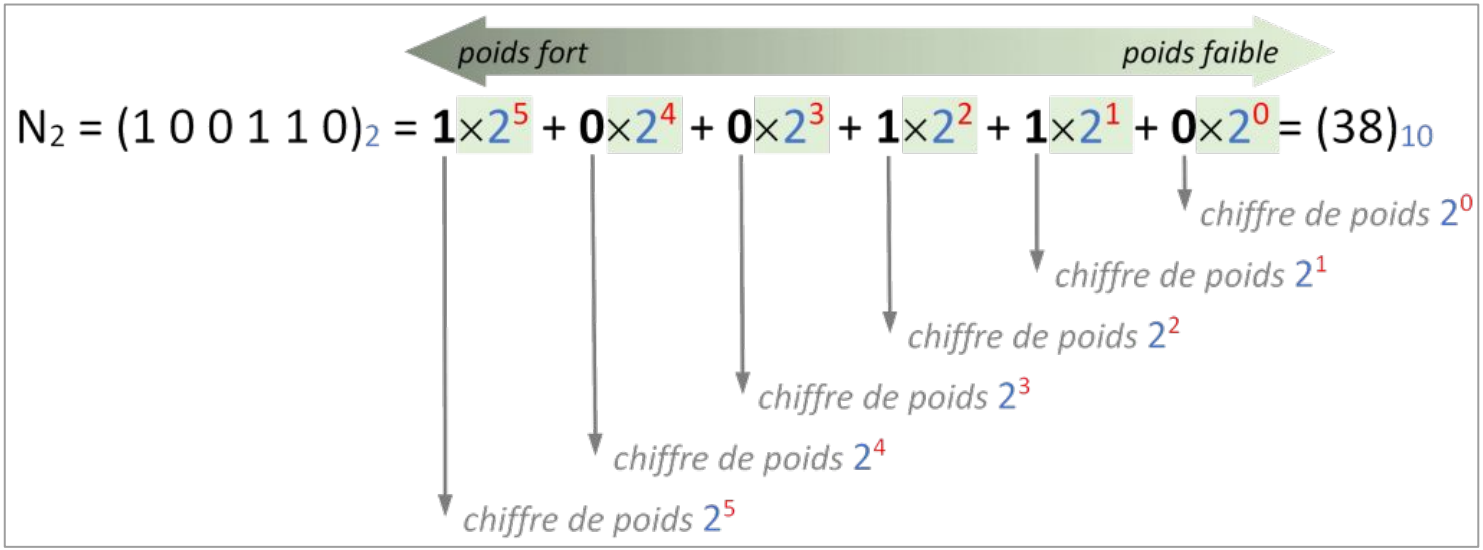
La base d’un système de numération indique le nombre de chiffres (c.a.d de symboles) distincts permettant l’écriture de n’importe quel nombre ℕ**aturel** dans ce système.

# 1.2

# Représentation des nombres entiers naturels

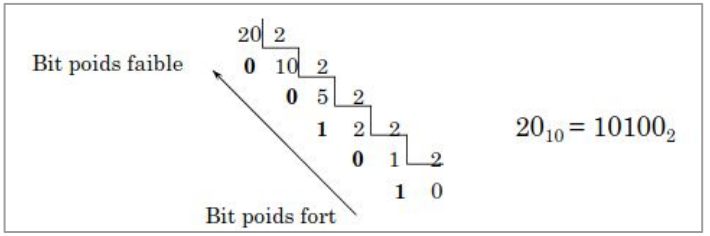
# Code binaire naturel

Les nombres entiers naturels sont représentés par le code binaire naturel qui utilise l'expression naturelle du nombre en base 2.



# Méthode de conversion

1. Réaliser la division euclidienne du nombre à coder par 2, mettre le reste à droite dans le tableau;
2. Recommencer la division par 2 avec le quotient à gauche du chiffre binaire précédent;
3. Continuer jusqu'à ce que le quotient ne soit plus divisible par 2.



# Arithmétique binaire: addition

Les ordinateurs ne connaissent pratiquement qu'une opération: l'addition.

Pour additionner 2 nombres en binaire, on procède comme en base 10.

Ainsi, on utilise ces 4 opérations références:

- 0 + 0 = 0
- 1 + 0 = 1
- 0 + 1 = 1
- 1 + 1 = 0 plus 1 de retenue, soit 10

puis on pose l'addition comme en base 10, avec le système de retenue.

# Arithmétique binaire: multiplication

Les principes sont les mêmes que ceux utilisés en base 10. On a donc les tableaux suivants pour l'addition et la multiplication:

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 10 |

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

Tables d'addition et de multiplication en base 2.

# Arithmétique binaire: soustraction

La soustraction c'est de l'addition. Soustraire un nombre c'est ajouter à ce nombre un nombre négatif. Si a et b sont des nombres binaires, a-b = a + (-b).

Faire le calcul de a - b, revient à faire l'addition de a + le complément à 2 (section suivante) de b.

# Arithmétique binaire: soustraction

Pour faire simple, il faut retenir :

- 0-0 = 0
- 0-1 = 1 (on retient 1)
- 1-0 = 1
- 1-1 = 0

# Arithmétique binaire: soustraction

**Exemple**: prenons (1001)2 (9 en base 10)

le complément à 1 (on inverse les bits) : 0110

le complément à 2 (on ajoute 1 au résultat) : 0110 + 1 = 0111

Alors pour faire 9-9, on ferra 1001+0111 = 0000.

Pour faire simple, il faut retenir :

- 0-0 = 0
- 0-1 = 1 (on retient 1)
- 1-0 = 1
- 1-1 = 0

# La division

Nous avons vu que la multiplication était basée sur une succession d’addition inversement la division va être basée sur une succession de soustraction et s’emploi de la même façon qu’une division décimal ordinaire.

# 1.3 Code hexadécimal

# Code hexadécimal

Le code hexadécimal permet une représentation simplifiée des nombres binaires, à l'usage des humains !

L'écriture et la lecture d'une succession de 0 et de 1 est fastidieuse pour l'homme et source d'erreurs.

Pour raccourcir les mots binaires, un groupe de 4 bits consécutifs est représenté par un chiffre hexadécimal.

# Code hexadécimal

```
# code binaire difficile à lire
000100111110010010101101


# conversion en hexadecimal
0001 0011 1110 0100 1010 1101
 1    3    E    4    A    D
```

# Code hexadécimal

- 4 bits permettent de coder 16 éléments;
- Pour coder 16 chiffres, on utilise ceux de la base 10 (0 à 9) auxquels on ajoute les 6 lettres de A à F: 0 1 2 3 4 5 6 7 8 9 A B C D E F.

# Conversion

Dans la pratique, la conversion binaire↔hexadécimal est très simple, il suffit d'utiliser un tableau de conversion chiffre hexadécimal ↔ code binaire.

| **hex** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **bin** | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |

| **hex** | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|
| **bin** | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

# Notation

Des notations sont utilisées, notamment dans les langages informatiques, pour différencier sans ambiguïté les nombres hexadécimaux des autres:

- notation préfixée: 0x123 (langage C et dérivés, Python), etc.
- notation suffixée: $123_{h}$, $123_{hex}$, $123_{(16)}$ (arithmétique)

1.4

# Représentation des nombres entiers relatifs

# Complément à deux

En informatique, le complément à deux est une méthode de représentation des entiers relatifs en binaire permettant d'effectuer simplement des opérations arithmétiques.

**Le complément à deux ne s'applique qu'à des nombres ayant tous la même longueur**: avec un codage sur n bits, cette méthode permet de représenter toutes les valeurs entières de $-2^{n-1}$ à $2^{n-1}-1$.

# Bit de signe

Le complément à deux opère toujours sur des nombres binaires ayant le même nombre de bits. Dans une telle écriture, le bit de poids fort (bit le plus à gauche) donne le signe du nombre représenté (positif ou strictement négatif). C'est le bit de signe.

Exemple, bit de signe en bleu:

- 00000010, 0 est le bit de signe +
- 10000010, 1 est le bit de signe -

# Problème de la représentation naïve

Une représentation naïve pourrait utiliser ce bit de poids fort comme marqueur du signe, les autres bits donnant une valeur absolue :

Dans les exemples ci-après, le bit de signe est représenté en bleu.

| Notation naïve | Décimal |
|---|---|
| 00000010 | +2 |
| 10000010 | -2 |

# Problème de la représentation naïve

Cette représentation possède deux inconvénients:

1. Le premier (mineur) est que le nombre zéro (0) possède deux représentations: 00000000 et 10000000 sont respectivement égaux à +0 et −0.
2. L'autre inconvénient (majeur) est que cette représentation impose de modifier l'algorithme d'addition; si un des nombres est négatif, l'addition binaire usuelle donne un résultat incorrect.

# Problème de la représentation naïve

| Décimal non signés | Addition en notation naïve | Décimal |
|---|---|---|
| 3 | 00000011 | 3 |
| + 132 | + 10000100 | + - 4 |
| = 135 | = 10000111 | = -1<br>= -7 (au lieu de -1) |

# Représentation des nombres en complément à 2

Pour remédier au problème posé par une représentation naïve, la notation en complément à deux est utilisée:

- Les nombres positifs sont représentés de manière usuelle.
- Les nombres négatifs sont obtenus en calculant l'opposé du nombre positif par deux opérations successives:
  - On inverse les bits de l'écriture binaire (opération binaire NON), on fait ce qu'on appelle le complément à un;
  - On ajoute 1 au résultat (les dépassements sont ignorés).

# Représentation des nombres en complément à 2

Cette opération correspond au calcul de $2^n - |x|$, où n est la longueur de la représentation et |x| la valeur absolue du nombre à coder. Ainsi, −1 s'écrit comme 256−1 = 255 = $11111111_2$, pour les nombres sur 8 bits.

Ceci est à l'origine du nom de cette opération : « complément à 2 puissance n », quasi-systématiquement tronqué en « complément à 2 ».

# Représentation des nombres en complément à 2

Les deux inconvénients précédents disparaissent alors. En effet, le calcul de l'opposé de 00000000 utilise le complément à 1: 11111111 qui après ajout de 1 redevient 00000000. De même, l'addition usuelle des nombres binaires fonctionne.

La même opération effectuée sur un nombre négatif donne le nombre positif de départ: $2^n - (2^n - x) = x$.

# Représentation des nombres en complément à 2

Par exemple, pour retrouver le codage binaire de (−4) sur 8 bits:

- on prend le nombre positif 4 : 00000100;
- on inverse les bits : 11111011;
- on ajoute 1 : 11111100.

# Représentation des nombres en complément à 2

Si l'on doit transformer un nombre en son complément à deux « de tête », un bon moyen est de garder tous les chiffres depuis la droite jusqu'au premier 1 (compris) puis d'inverser tous les suivants.

- Prenons par exemple le nombre 20 : 00010100.
- On garde la partie à droite telle quelle : (00010**100**).
- On inverse la partie de gauche après le premier un : **11101**100.
- Et voici −20 : 11101100.

# 1.5 Représentation des nombres réels

# Représentation de la partie décimale d'un nombre

La méthode est simple et ressemble à celle utilisée pour représenter la partie entière d'un nombre. A partir de la partie décimale du nombre à représenter:

- Multiplier le nombre par 2;
- Si le résultat est strictement inférieur à 1, alors on retient le chiffre 0, sinon on retient le chiffre 1;
- Recommencer à l'étape 1 avec la partie décimale du nombre obtenu à l'étape 2.

L'opération est terminée lorsque ce nombre est nul.

# Valeur approchée

En base 10, il est impossible de représenter certains nombres rationnels (par exemple ⅓ ≈ 0 ,33333333···). Cela concerne tous les nombres pour lesquels il n'existe pas de puissance de 10 par lesquels on peut les multiplier pour obtenir un entier.

De la même manière, en base 2 il est impossible de représenter les nombres pour lesquels il n'existe pas de puissance de 2 par lesquels on peut les multiplier pour obtenir un entier. Par exemple: 0,110 ≈ 0,0001100110011···, avec le motif 0011 qui se répète à l'infini.

# Virgule flottante

En base 10, il est possible d'écrire les très grands nombres et les très petits nombres en utilisant les puissances de 10: Par exemple $5{,}187 \cdot 10^{23}$

Il est possible de faire exactement la même chose avec une représentation binaire, en utilisant les puissances de 2: Par exemple $101{,}1101 \cdot 2^{10}$. (ATTENTION, l'exposant *10* est aussi exprimé en binaire !).

# Virgule flottante

Cette notation est appelée notation à virgule flottante car en effet, pour représenter un même nombre, la virgule peut être décalée à droite ou à gauche:

- en base 10, décaler la virgule vers la droite revient à multiplier le nombre par 10: exemple : $12{,}5819 \cdot 10^4 = 125{,}819 \cdot 10^3$
- en base 2, décaler la virgule vers la droite revient à multiplier le nombre par 2: exemple : $1110{,}1100101 \cdot 2^{1000} = 11101{,}100101 \cdot 2^{0111}$

# Virgule flottante

D’une manière générale, une représentation à virgule flottante consiste à représenter un nombre réel par:

- un signe (égal à −1 ou 1);
- une mantisse (aussi appelée significande);
- et un exposant (entier relatif, généralement borné).

Un tel triplet représente le nombre réel : $signe \times mantisse \times b^{exposant}$

# Représentation des flottants dans un ordinateur

L'IEEE 754 est une norme sur l'arithmétique à virgule flottante. Elle est la norme la plus employée actuellement pour le calcul des nombres à virgule flottante dans le domaine informatique.

La norme définit les formats de représentation des nombres à virgule flottante et valeurs spéciales (infinis et NaN - Not a number), en même temps qu'un ensemble d'opérations sur les nombres flottants.

La première version de cette norme date de 1985.

# Représentation des flottants dans un ordinateur

Selon la norme IEEE 754, le nombre réel à exprimer en binaire doit être écrit sous la forme: $\pm \times mantisse \times 2^{exposant}$

de sorte que la mantisse s'écrive 1,⋯

et que sa partie décimale et l'exposant puissent être représentée en code binaire naturel: $s \times 1,a_{22}a_{21}\cdots a_1a_0 \times 2^{b7b6\cdots b1b0}$

# Représentation des flottants dans un ordinateur

Il peut être exprimé en binaire, à l'aide de 32 bits, sous la forme:

| ***s*** | $b_7b_6 \cdots b_1b_0$ | $a_{22}a_{21} \cdots a_1a_0$ |
|---|---|---|
| Bit de signe:<br>● +: 0<br>● -: 1 | Bits d'exposant:<br>exposant = $(b_7b_6 \cdots b_1b_0)_2 - 127$<br><br>Soient des exposants $2^{-126}$ à $2^{127}$<br><br>Les valeurs 0000 0000 et 1111 1111 sont réservées à un autre usage. | Bits de mantisse:<br>$a_{22}a_{21} \cdots a_1a_0$<br><br>Correspondants aux chiffres binaires après la virgule, la partie entière étant toujours égale à 1 |

# Représentation des flottants dans un ordinateur

La représentation sur 32 bits de 0,1 s'obtient de la manière suivante:

$$0{,}1 = 1{,}6 \times 2^{-4}$$

$$-4 + 127 = 123 = 01111011_2$$

$$1{,}6 = 1{,}10011001100110011001100_2$$

$\Rightarrow$ 0 01111011 10011001100110011001100

# Représentation des flottants dans un ordinateur

La norme prévoit deux formats :

- Simple précision : 32 bits (8 bits d’exposant, 23 bits de mantisse)
- Double précision : 64 bits (11 bits d’exposant, 52 bits de mantisse)

# 02 Les caractères et les textes

# Charset et Encoding

En télécommunications et en informatique, un jeu de caractères codés (charset encoding en anglais) est un code qui associe un jeu de caractères d'un alphabet avec une représentation numérique pour chaque caractère de ce jeu.

- Le jeu de caractère est nommé charset (character set);
- Le code qui relie le chaque caractère à un nombre est nommé encoding.

# Charset et Encoding

Le code ASCII [aski:] s'est imposé au début de l'ère informatique pour coder 128 caractères (lettres, chiffres et autres symboles).

| Bits | $b_6b_5b_4$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|
| $b_3b_2b_1b_0$ | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0000 | 0 | (NUL) | (DLE) | (SP) | 0 | @ | P | ` | p |
| 0001 | 1 | (SOH) | (DC1) | ! | 1 | A | Q | a | q |
| 0010 | 2 | (STX) | (DC2) | " | 2 | B | R | b | r |
| 0011 | 3 | (ETX) | (DC3) | # | 3 | C | S | c | s |
| 0100 | 4 | (EOT) | (DC4) | $ | 4 | D | T | d | t |
| 0101 | 5 | (ENQ) | (NAK) | % | 5 | E | U | e | u |
| 0110 | 6 | (ACK) | (SYN) | & | 6 | F | V | f | v |
| 0111 | 7 | (BEL) | (ETB) | ' | 7 | G | W | g | w |
| 1000 | 8 | (BS) | (CAN) | ( | 8 | H | X | h | x |
| 1001 | 9 | (HT) | (EM) | ) | 9 | I | Y | i | y |
| 1010 | 10 | (LF) | (SUN) | * | : | J | Z | j | z |
| 1011 | 11 | (VT) | (ESC) | + | ; | K | [ | k | { |
| 1100 | 12 | (FF) | (FS) | , | < | L | \ | l | \| |
| 1101 | 13 | (CR) | (GS) | - | = | M | ] | m | } |
| 1110 | 14 | (SOH) | (RS) | . | > | N | ^ | n | ~ |
| 1111 | 15 | (SI) | (US) | / | ? | O | _ | o | (DEL) |

# Quand la table ASCII ne suffit plus

Pour coder un texte dans un alphabet plus complexe et pour échanger des informations sur l'Internet, il faut impérativement disposer d'un charset plus complet.

Par exemple, en français les caractères é, è, ç, à, ù, ô, æ, œ, sont fréquemment utilisés alors qu'ils ne figurent pas dans la table ASCII.

# Quand la table ASCII ne suffit plus

Il va donc falloir étendre la table ASCII pour pouvoir coder les nouveaux caractères.

Les mémoires devenant plus fiables et, de nouvelles méthodes plus sûres que le contrôle de parité ayant été inventées, le 8$^{ième}$ bit a pu être utilisé pour coder plus de caractères.

# Quand la table ASCII ne suffit plus

On élimine ainsi l'inconvénient très gênant de ne coder que les lettres non accentuées, ce qui peut suffire en anglais, mais pas dans les autres langues (comme le français et l'espagnol par exemple). On a pu aussi rajouter des caractères typographiques utiles comme des tirets de diverses tailles et sortes.

# De la difficulté de convenir d'une norme

Le fait d'utiliser un bit supplémentaire a bien entendu ouvert des possibilités mais malheureusement tous les caractères ne pouvaient être pris en charge. La norme ISO 8859–1 appelée aussi Latin-1 ou Europe occidentale est la première partie d'une norme plus complète appelée ISO 8859 (qui comprend 16 parties) et qui permet de coder tous les caractères des langues européennes.

# De la difficulté de convenir d'une norme

Cette norme ISO 8859–1 permet de coder 191 caractères de l'alphabet latin qui avaient à l'époque été jugés essentiels dans l'écriture, mais omet quelques caractères fort utiles (ainsi, la ligature œ n'y figure pas). Dans les pays occidentaux, cette norme est utilisée par de nombreux systèmes d'exploitation, dont Linux et Windows. Elle a donné lieu à quelques extensions et adaptations, dont Windows-1252 (appelée ANSI) et ISO 8859-15 (qui prend en compte le symbole € créé après la norme ISO 8859-1).

# De la difficulté de convenir d’une norme

C’est source de grande confusion pour les développeurs de programmes informatiques car un même caractère peut être codé différemment suivant la norme utilisée.

Voici deux tableaux présentant côte à côte ces deux encodages:

# De la difficulté de convenir d'une norme

**Windows-1252 (CP1252)**

| | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 1x | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 2x | SP | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 3x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4x | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5x | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 6x | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7x | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | DEL |
| 8x | € | | ‚ | ƒ | „ | … | † | ‡ | ˆ | ‰ | Š | ‹ | Œ | | Ž | |
| 9x | | ‘ | ’ | “ | ” | • | – | — | ˜ | ™ | š | › | œ | | ž | Ÿ |
| Ax | NBSP | ¡ | ¢ | £ | ¤ | ¥ | ¦ | § | ¨ | © | ª | « | ¬ | | ® | ¯ |
| Bx | ° | ± | ² | ³ | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ | ¾ | ¿ |
| Cx | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï |
| Dx | Ð | Ñ | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û | Ü | Ý | Þ | ß |
| Ex | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï |
| Fx | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù | ú | û | ü | ý | þ | ÿ |

**ISO/CEI 8859-15**

| | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x | *non utilisé* | | | | | | | | | | | | | | | |
| 1x | | | | | | | | | | | | | | | | |
| 2x | | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 3x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4x | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5x | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 6x | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7x | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | |
| 8x | *non utilisé* | | | | | | | | | | | | | | | |
| 9x | | | | | | | | | | | | | | | | |
| Ax | | ¡ | ¢ | £ | € | ¥ | Š | § | š | © | ª | « | ¬ | | ® | ¯ |
| Bx | ° | ± | ² | ³ | Ž | µ | ¶ | · | ž | ¹ | º | » | Œ | œ | Ÿ | ¿ |
| Cx | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï |
| Dx | Ð | Ñ | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û | Ü | Ý | Þ | ß |
| Ex | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï |
| Fx | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù | ú | û | ü | ý | þ | ÿ |

# Quand le net s’affole…

Nous avons tous un jour reçu un courriel bizarre ou lu une page web telle que celle-ci :

*Prenons lâ€™exemple typique de la lumiÃ¨re Ã©mise par un phare maritime : elle est dâ€™abord indivisible, son coÃ»t de production Ã©tant alors indÃ©pendant du nombre d’utilisateurs ; elle possÃ¨de une propriÃ©tÃ© de non-rivalitÃ© (elle ne se dÃ©truit pas dans l’usage et peut donc Ãªtre adoptÃ©e par un nombre illimitÃ© d’utilisateurs) ; elle est Ã©galement non excluable car il est impossible dâ€™exclure de lâ€™usage un utilisateur, mÃªme si ce dernier ne contribue pas Ã son financement.*

# Quand le net s’affole…

Bien que ce soit de moins en moins fréquent, on trouve parfois des phrases dans lesquelles certains caractères sont remplacés par d’autres (connus sous le nom de *mojibakes*) qui n’ont rien à voir et qui empêchent la lecture et la compréhension du texte. Il s’agit ici d’un problème d’encodage et de décodage. La personne qui écrit le texte utilise une norme différente de celle utilisée par celui qui le lit !

# Et l’Unicode vint…

La globalisation des échanges culturels et économiques a mis l’accent sur le fait que les langues européennes coexistent avec de nombreuses autres langues aux alphabets spécifiques voire sans alphabet. La généralisation de l’utilisation d’Internet dans le monde a ainsi nécessité une prise en compte d’un nombre beaucoup plus important de caractères (à titre d’exemple, le mandarin possède plus de 5000 caractères !).

# Et l’Unicode vint…

Une autre motivation pour cette évolution résidait dans les possibles confusions dues au trop faible nombre de caractères pris en compte ; ainsi, les symboles monétaires des différents pays n’étaient pas tous représentés dans le système ISO 8859-1, de sorte que les ordres de paiement internationaux transmis par courrier électronique risquaient d’être mal compris.

# Et l’Unicode vint…

La norme Unicode a donc été créée pour permettre le codage de textes écrits quel que soit le système d’écriture utilisé.

On attribue à chaque caractère un nom, un numéro (appelé point de code) et un bref descriptif qui seront les mêmes quelle que soit la plate-forme informatique ou le logiciel utilisés.

# Et l’Unicode vint…

Un consortium composé d’informaticiens, de chercheurs, de linguistes et de personnalités représentant les États ainsi que les entreprises s’occupe donc d’unifier toutes les pratiques en un seul et même système : l’Unicode.

L’Unicode est une table de correspondance Caractère-Code (Charset), et l’UTF-8 est l’encodage correspondant (Encoding) le plus répandu.

# Et l’Unicode vint…

Maintenant, par défaut, les navigateurs Internet utilisent le codage UTF-8 et les concepteurs de sites pensent de plus en plus à créer leurs pages web en prenant en compte cette même norme; c’est pourquoi il y a de moins en moins de problèmes de compatibilité.

# L’encodage UTF-8

L’encodage UTF-8 utilise 1, 2, 3 ou 4 octets en respectant certaines règles:

- Un texte en ASCII de base (appelé aussi US-ASCII) est codé de manière identique en UTF-8.
- On utilise un octet commençant par un bit 0 à gauche (bit de poids fort).

# L’encodage UTF-8

Exemple de codage UTF-8

| Type | Caractère | Point de code (hexadécimal) | Valeur scalaire | | Codage UTF-8 | |
|---|---|---|---|---|---|---|
| | | | décimal | binaire | binaire | hexadécimal |
| Texte | A | U+0041 | 65 | 1000001 | 01000001 | 41 |
| | é | U+00E9 | 233 | 00011 101001 | 11000011 10101001 | C3 A9 |

https://fr.wikipedia.org/wiki/Point_de_code

# L’encodage UTF-8

- Les octets ne sont pas remplis entièrement.

  Les bits de poids fort du premier octet forment une suite de bits indiquant le nombre d’octets utilisés pour coder le caractère.

  Les octets suivants commencent tous par le bloc binaire 10.

# L’encodage UTF-8

| Caractères codés | Représentation binaire UTF-8 | Signification |
|---|---|---|
| U+0000 à U+007F | 0xxxxxxx | 1 octet, codant 7 bits |
| U+0080 à U+07FF | 110xxxxx 10xxxxxx | 2 octets, codant 11 bits |
| U+0800 à U+FFFF | 1110xxxx 10xxxxxx 10xxxxxx | 3 octets, codant 16 bits |
| U+10000 à U+FFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx | 4 octets, codant 21 bits |

# L’encodage UTF-8

Dans la norme ISO 8859-1 le «**é**» est codé sur 1 octet : 1110 1001,

En UTF-8 on le code sur **deux octets** en respectant les précisions apportées dans le tableau ci-dessus. Les bits imposés sont **110** pour le 1er octet et **10** pour le deuxième, le code du «**é**» est écrit en commençant par la droite et l’octet de gauche est rempli par des zéros (en *italique*). Voilà ce que l’on obtient : **110***000*11 **10**101001.

On pourra remarquer que le codage ISO s’inscrit bien dans le codage UTF-8.

# L’encodage UTF-8: remarques

- Ce codage permet de coder tous les caractères de la norme Unicode.
- Les caractères Unicode sont la plupart du temps représentés en hexadécimal. Par exemple : 1110 1001 qui est le code binaire du « é » en ISO 8859-1 devient $E9_h$ ce que l’on peut retrouver dans le tableau donné plus haut. Il faut noter que la notation en binaire est très peu utilisée sauf par les électroniciens et par ceux qui travaillent en langage machine.

# L’encodage UTF-8: remarques

- Le système de codage UTF-8 permet d’encoder un même caractère de plusieurs manières. Ceci peut poser un problème de sécurité car un programme détectant certaines chaînes de caractères (pour contrer des injections dans les bases de données par exemple), s’il est mal écrit, pourrait alors accepter des séquences nuisibles. En 2001 un virus a ainsi attaqué des serveurs http du web.

# L’encodage UTF-8: remarques

- Par exemple, le symbole € pourrait être codé sur 4 octets (forme super longue) de la manière suivante : 11110000 10000010 10000010 10101100. Si elle n’est pas rejetée ou remise sous forme standard ce codage ouvrira une brèche potentielle de sécurité par laquelle on pourra faire passer un virus.

# Bibliographie


- C. Faury, Représentation des données, Lycée Blaise Pascal.
- A. Benhadid, Architecture des ordinateurs, benhadid.github.io