Architecture de Ordinateurs

Module 2

# Numération - Codage

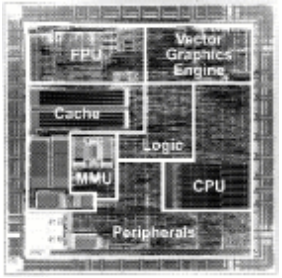
Dr. Yannick HERVE

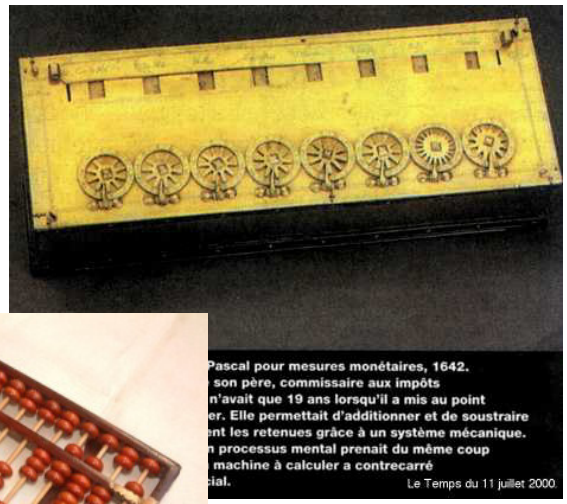
V 1.2

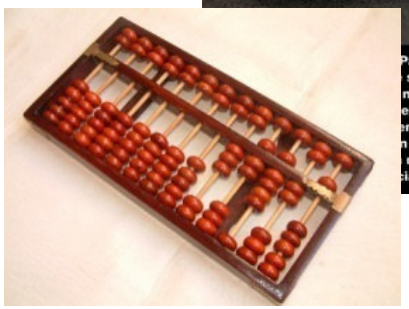
# Manipulation des valeurs par les ordinateurs

- Nombres entiers
- Nombres entiers signés
- Nombres à virgule
  - Virgule fixe
  - Virgule flottante

- Codes et codage



# Quelques définitions

**Alphabet** : ensemble de symboles utilisables

**Mot** : Séquence de symboles (alphabet) représentant une information ; Exemples : ordinateur, 2005, XXVIII

**Mot binaire** : mot constitué avec l'alphabet binaire {1, 0}
Exemple : 1100 1111 1010

**Quartet** (nibble) : mot binaire de longueur 4

**Octet** (byte) : mot binaire de longueur 8

**Code** : ensemble de mots auxquels on confère une signification (convention) pour représenter une catégorie de messages ou de concepts.

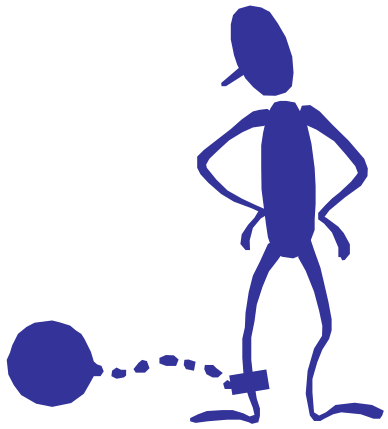
# Nombres en précision finie

Ordinateurs : nombres en précision finie et fixe

Exemple : ensemble des entiers positifs à trois chiffres décimaux

A = {000,001,002, ..., 999}  Card(A)=1000

Ne peut représenter :
- > 999
- < 0
- fractionnaires
- irrationnels
- complexes



Arithmétique sur Z : fermeture vis à vis de +,-,x
(pas de /)

Soient $i, j \in Z$

$i + j \in Z$

$i - j \in Z$

$i \times j \in Z$

$i / j \notin Z$ en général

Fermeture pas applicable en précision finie !

Sur A : 600 + 600 = 1200 $\notin$ A (overflow)
003 - 005 = -2 $\notin$ A (underflow)
050 * 050 = 2500 $\notin$ A (overflow)
007 / 002 = 3,5 $\notin$ A (arrondi)

# Conséquences pour les ordinateurs

Possibilités de faux résultats en conditions normales de fct. (pas de panne)

Dans A :
a = 700, b = 400, c = 300
a + (b - c) = (a + b) – c commutativité

700 + 100 = 800

Overflow -300 = OverFlow

Dans A :
a = 5, b = 210, c = 195
a x (b - c) = a x b - a x c distributivité

5 x 15 = 75

Overflow - 975 = OverFlow

Il faut connaître les méthodes de représentations pour prévoir les problèmes éventuels

# Représentation des nombres

Evolution : romaine, grèce (invention du zéro), inde, arabe

Principe de numération : Juxtaposition de symboles
appelés chiffres (caillou en arabe)

Système décimal : dix symboles {0,1,2, ...,9}
Nombre de symbole = Base de numération

Ecriture d'un nombre : position du chiffre détermine son poids

NUMERATION DE POSITION

$$1578 = 1.10^3 + 5.10^2 + 7.10^1 + 8.10^0$$

(en europe : 970 Gesbert d'Aurillac devenu en 999 Sylvestre II, relayé en 1202 par Fibonnacci)

Ecriture polynomiale

# Représentation des nombres

Soit une base b associée à b symboles $\{S_0, S_1, S_2, \ldots, S_{b-1}\}$

Un nombre positif N dans un système de base b s'écrit sous la forme polynomiale:

$$N = a_{n-1} \cdot b^{n-1} + a_{n-2} \cdot b^{n-2} + \cdots + a_1 \cdot b^1 + a_0 \cdot b^0 + a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + \cdots + a_{-m+1} \cdot b^{-m+1} + a_{-m} \cdot b^{-m}$$

La représentation simples de position est la suivante: $\left(a_{n-1}a_{n-2}a_1a_0, a_{-1}a_{-2}a_{-m+1}a_{-m}\right)$

$a_i$ est le chiffre de rang $i$ ($a_i$ appartient à un ensemble de $b$ symboles)

$a_{n-1}$ est le chiffre le plus significatif

$a_{-m}$ est le chiffre le moins significatif

$\left(a_{n-1}a_{n-2}\ldots a_0\right)$ partie entière

$\left(a_{-1}a_{-2}\ldots a_{-m}\right)$ partie fractionnaire $(<1)$

# Les bases usuelles

Système binaire (b=2)

$a_i \in \{0,1\}$

$a_{n-1}$ est le MSB (most significant bit)

$a_{-m}$ est le LSB (least significant bit)

Système octal (b=8)

$a_i \in \{0,1,2,3,4,5,6,7\}$

**Système décimal (b=10)**
**(base de l'école primaire )**

$a_i \in \{0,1,2,3,4,5,6,7,8,9\}$

Système hexadécimal (b=16)
(raccourci d'écriture de la base 2)

$a_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

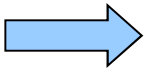
# Conversions entre bases

Base ***b*** vers base 10 : il suffit de substituer la valeur b dans l'expression polynomiale par la valeur de la base.

$$(F1C)_{16} = 15 \cdot 16^2 + 1 \cdot 16^1 + 12 \cdot 16^0 = (3868)_{10}$$

Base 10 vers base ***b*** : division successives du nombre décimal par ***b*** jusqu'à obtenir un quotient nul. Le nombre dans la base ***b*** correspond aux restes des divisions faites dans le sens inverse où ils ont été obtenus.

| Division | Quotient | Reste |
|---|---|---|
| 1836/7 | 262 | 2 ($a_0$) |
| 262/7 | 37 | 3 ($a_1$) |
| 37/7 | 5 | 2 ($a_2$) |
| 5/7 | 0 | 5 ($a_3$) |



$$(1836)_{10} = (5232)_7$$

Justification : 123/10 = 12,3
Quotient = 12, Reste = 3

# Partie fractionnaire

Pour la partie fractionnaire : algorithme de multiplication

Principe : En base 10 $0,xyz * 10 = x,yz = x + 0,yz$

| | | |
|---|---|---|
| | $0,xyz * 10 = x,yz$ | x |
| partie fractionnaire de x,yz | $0,yz * 10 = y,z$ | y |
| partie fractionnaire de y,z | $0,z * 10 = z$ | z |

# Conversion 10 vers B : exemple

$(0,45)_{10}$ en base 2 ?

| | |
|---|---|
| 0,45 * 2 = 0,90 | 0 |
| 0,90 * 2 = 1,8 | 1 |
| 0,8 * 2 = 1,6 | 1 |
| 0,6 * 2 = 1,2 | 1 |
| 0,2 * 2 = 0,4 | 0 |
| 0,4 * 2 = 0,8 | 0 |
| 0,8 * 2 = 1,6 | 1 |
| 0,6 * 2 = 1,2 .. ... | |

$(0,45)_{10} = (0,0111001...)_2$

Une longueur finie en base 10 peut être infinie en base B
On conserve la précision relative $10^{-3}$ est approximée par $2^{-10}$

## Conversion : de $2^m$ vers 2 / 2 vers $2^m$

$2^m$ vers 2 : expansion d'un digit en m bits

2 vers $2^m$ : regroupement de bits par paquets de m

$$N = a_7.2^7 + a_6.2^6 + a_5.2^5 + a_4.2^4 + a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0$$

$$= (a_7.2^3 + a_6.2^2 + a_5.2^1 + a_4.2^0).2^4 + (a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0)$$

$$= (a_7.2^3 + a_6.2^2 + a_5.2^1 + a_4.2^0).16^1 + (a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0).16^0$$

$$N = b_1.16^1 + b_0.16^0$$

$$0 \le a_3.2^3 + a_2.2^2 + a_1.2^1 + a_0.2^0 \le 15$$

Ecriture de $(622,663)_8$ en base 2 et base 16 ?

# $(622{,}663)_8$ ?

```
 6    2    2  ,  6    6    3      base 8
110  010  010 , 110  110  011     base 2



 1 1001 0010 , 1101 1001 1        base 2
 1   9    2  ,  D    9    8       base 16
```

# Conversion base I vers base J

si $I = B^m$ et $J = B^n$

$B^m$ vers B puis B vers $B^m$

sinon

I vers 10 puis 10 vers J

# Représentations binaires

(nombres entiers naturels et relatifs, nombre fractionnaires)

| | | |
|---|---|---|
| Définitions : | format | nb de bit de utilisés |
| | convention | protocole de codage |
| Conséquences | dynamique | différence entre le max et le min |
| | résolution | différence entre deux consécutifs |

Exemple : format 8 bits
convention entiers positifs
dynamique $2^8$
résolution 1 (constante sur la dynamique)

$(255)_{10} = (1111\ 1111)_2$ $(7)_{10} = (0000\ 0111)_2$

Le format est important car les mots machines sont à taille finie et fixe

# Nombres signés : signe + module

Sur n bits on garde 1 bit pour indiquer le signe


Signe 1 bit — Module (positif) n-1 bits

Convention :

S=0 pour positif

S=1 pour négatif

Exemple sur 8 bits : -23 = $(1\ 0010111)_{2,S+M}$

Dynamique : $-(2^{n-1}-1)$ à $(2^{n-1}-1)$

Inconvénient : Deux représentations du zéro

Sur 4 bits +0 = 0000, -0 = 1000

Multiplications faciles

$N_1 * N_2$

$Abs(N_1) * Abs(N_2)$

$S = S_1$ xor $S_2$

Additions moins simples

Thermomètre de voiture : - 0°

# Nombres signés : complément restreint

(complément à B-1, ou complément à 1)

Def CR(X) = $\overline{X}$

Complément chiffre à chiffre
$(xyz)_b$ donne $(x'y'z')_b$
tel que $x + x' = b-1$

On a $X + CR(X) = b^n - 1$

En binaire 00110 donne 11001
et 00110 + 11001 = 11111

Dans le format considéré $2^n = 0$ Partie interprétée

sur 4 bits : $2^4 = 10000 = 0$

Avec : $X + CR(X) = 2^n - 1$
d'où : $CR(X) + 1 = -X$

# Nombres signés : complément vrai

(complément à B, complément à 2, 2*)

On a :

Avec : $X + CR(X) = 2^n - 1$

d'où : $CR(X) + 1 = -X$ On note $X^* = CR(X)+1 = -X$

Autre méthode : Calcul de l'opposé (sur n bits)

$N^* = (-N) = 2^n - N = [2^n-1] - N + 1$

$= [N + CR(N)] - N + 1 = CR(N) +1$

$N^* = CR(N) +1 = CV(N)$

# Complément à 2

| | | | | |
|---|---|---|---|---|
| Sur 4 bits : | 7 | 0111 | -7 | 1001 |
| | 6 | 0110 | -6 | 1010 |
| | ... | | | |
| | 0 | 0000 | -0 | 0000 |

Remarques : le bit de poids fort = signe (0:positif, 1:négatif)
0 n'a qu'un représentation
1000 jamais rencontré car
-(1000) = 0111 + 1 = 1000 (d'où (1000) = 0)
et 1000 + 0001 = 1001 = -7 (d'où (1000) = -8)

Dynamique sur n bits : $-(2^{n-1}-1)$ à $(2^{n-1}-1)$

# Nombres signés : binaire décalé sur m bits

(ou excédent $2^{m-1}$)

On stocke les nombres de m bits comme

$$N_{stocké} = N_{xs} = N_m + 2^{m-1}$$

(translation de la demi-dynamique)

Exemple sur 8 bits : $N_{xs} = N_8 + 128$

| Valeur à coder : | -128 | ... | 0 | ... | 127 |
|---|---|---|---|---|---|
| | ↓ | | ↓ | | ↓ |
| Valeur stockée : | 0 | | 128 | | 255 |

Opération à la restitution

$N_{lu} = N_{xs} - 128$

Avantage : on garde la relation d'ordre

on peut effectuer les comparaisons facilement

Remarque : identique au 2* au signe près

# Nombres signés : comparaison

| $N_{10}$ | $N_2$ | $(-N)_{S+M}$ | $(-N)_{2,CR}$ | $(-N)_{2*}$ | $(-N)_{2,XS8}$ |
|---|---|---|---|---|---|
| mêmes positifs | | | | | positifs différents |
| 0 | 0000 | 1000 | 1111 | 0000 | 1000 |
| 1 | 0001 | 1001 | 1110 | 1111 | 0111 |
| 2 | 0010 | 1010 | 1101 | 1110 | 0110 |
| 3 | 0011 | 1011 | 1100 | 1101 | 0101 |
| 4 | 0100 | 1100 | 1011 | 1100 | 0100 |
| 5 | 0101 | 1101 | 1010 | 1011 | 0011 |
| 6 | 0110 | 1110 | 1001 | 1010 | 0010 |
| 7 | 0111 | 1111 | 1000 | 1001 | 0001 |
| 8 | 1000 | .... | .... | (1000) | 0000 |

Remarque :
relation d'ordre
signe du zéro
symétrie
gestion retenues

# 2* : propriétés

Propriétés : pas de gestion de retenue intermédiare
détection simple d'overflow

sur 4 bits : (-7 à +7)

| | |
|---|---|
| 3 | 0011 |
| + 6 (9 = 0F) | 0110 = 1001 |
| - 5 | 1011 |
| = 4 | = 1 0100 |

↑hors format

X + (- Y) = N avec X > N > -Y
4 + 5 = 9 (of) 0100 + 0101 = 1001
-4 - 5 = -9 (of) 1100 + 1011 = 0111

Note : modification de signe
Indicateur d'overflow :
(dans les microprocesseurs)

$$Fd = S_a.S_b.\overline{S_r} + \overline{S_a}.\overline{S_b}.S_r$$

# Nombres non entiers

Dans un calculateur : nombre sous format déterminé (entier, virgule fixe, virgule flottante ...)

TOUT EST QUESTION DE CONVENTION

Quoi associer à 1101100011100110 ?

Caractère ASCII, pixel d'une image, nombre entier 2* nombre fractionnaire ... ?

Dans l'ordinateur (le système numérique) il n'y a pas de virgule

# Virgule fixe

Par convention on place la virgule quelque part et on interprète

$2^{n-1}$ $2^{0}$ avant de placer la virgule

MSB xxxxxx , xxxx LSB

$2^{n-1-k}$ $2^{0}$,$2^{-1}$ $2^{-k}$ avec la virgule au rang k

Dynamique : $2^{n-1-k}$

Résolution : $2^{-k}$ # 0

# Virgule fixe : analyse

Bon format pour l'addition :

```
   12,32
+  23,45
=  35,77   La virgule reste fixe
```

Mauvais format pour la multiplication :

```
    13,4
 *  10,1
=  135,34   dépassement à gauche (overflow)
           dépassement à droite (arrondi)
```

# Virgule fixe : solution

Problèmes réglés si les nombres sont inférieurs à 1 :

$$\begin{array}{r} 0{,}87 \\ * \ 0{,}74 \\ \hline = \ 0{,}64_{38} \end{array}$$

- On place la virgule toujours à gauche
- On utilise un autre groupe de bit pour connaître la position de la virgule

⟹ Format virgule flottante

# Format virgule flottante

$N = M.b^E$

M = mantisse en 2* de forme 0,xxx
b = base de l'exponentiation (2 ou 16)
E = exposant en binaire décalé

On stocke la chaine de bit **ME** dans le calculateur

Exemple : codage de PI sur 5 chiffres de mantisse
et 2 chiffres d'exposant (en décimal)

$0{,}3141.10^1 = 0{,}0003.10^4$ !!! PI*10000=3

On dit qu'un flottant est normalisé quand le premier chiffre significatif est juste derrière la virgule (précision maximum)

Assure unicité et précision maximale

# Virgule flottante : calcul/stockage

Multiplication : $M_1.b^{E_1} * M_2.b^{E_2} = M_1.M_2.b^{(E_1+E_2)}$

Addition :

dénormalisation du plus petit nombre (vers la droite)

$$M_1.b^{E_1} + M_2.b^{E_2} = M_1.b^{(E_1-E_2)}.b^{E_2} + M_2.b^{E_2}$$

$$= (M_1.b^{(E_1-E_2)} + M_2).b^{E_2}$$

puis renormalisation

Si $E_2 > E_1$

$\Longrightarrow$ il faut comparer facilement
Exposant codé en binaire décalé

| $S_M$ | EXPOSANT | MANTISSE |
|---|---|---|

# Virgule flottante : IEEE 754/854

Norme internationale : IEEE 754 flottant sur 32 bits
(simple précision (32), sp étendue (≥43), double précision (64), dp étendue ≥79)
IEEE 854 flottant généralisé

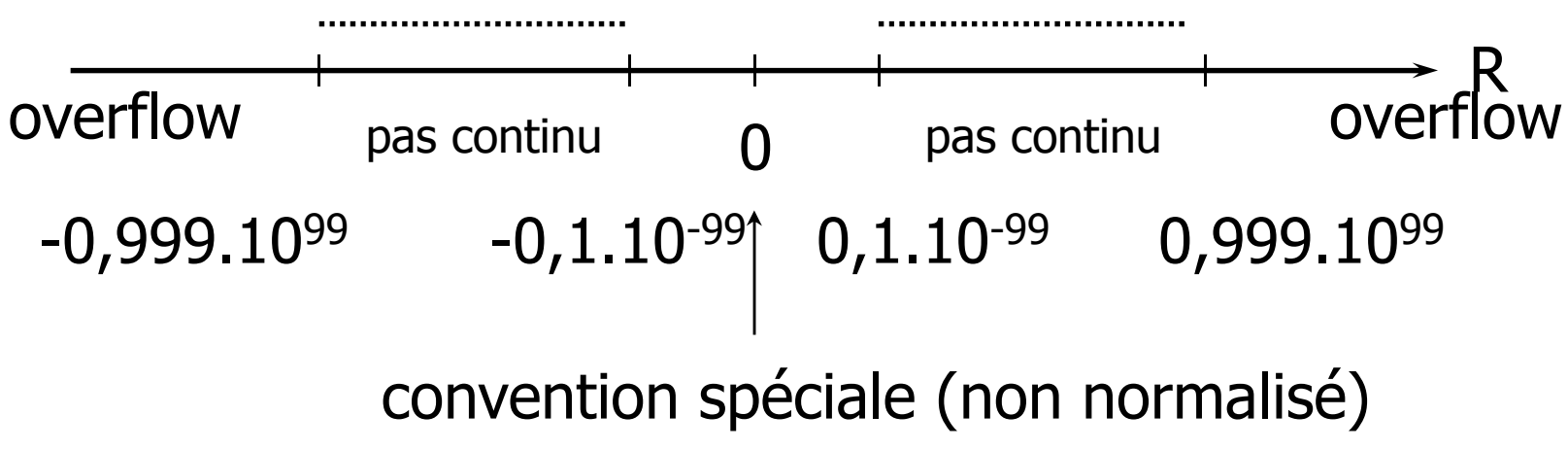
$b_{31}$ ..... ..... $b_0$
signe mantisse, exposant, mantisse
1 bit 8 bit 23 bits

Le bit de signe est 1 pour négatif et 0 pour positif
La mantisse vaut toujours 1,xxxx et on ne stocke que xxxx
L'exposant est en excédent 127
La valeur 0 correspond à des 0 partout (en fait $1,0.2^{-127}$)

Exemples : 1 10000011 11000000000000000000000 = $-1,75.2^4 = -28$
0 01111111 00000000000000000000000 = $1,0.2^0 = 1$

# Virgule flottante : performances

Exemple : (10) Mantisse 3 chiffres $0{,}999 > |M| > 0{,}1$
Exposant 2 chiffres -99 à 99




ON NE MANIPULE JAMAIS L'ENSEMBLE DES REELS

# Codage

Code : correspondance conventionnelle
entre un objet et un mot binaire

Analytique : code calculable
Conventionnel : pas de correspondance logique

# Code analytique : BCD

## Attention : BCD = Décimal Codé en binaire

Utilisé quand la précision doit être la même en binaire et en décimal (comptabilité, fiscalité, banque …)

Utilisé dans les machines à calculer.

Facilité de conversion pour l'affichage (humains)

Le code BCD encode le nombre à représenter de façon très directe : Chaque chiffre est encodé sur 4 bits. Les possibilités binaires de 10 à 15 ne sont pas utilisées.

Peut être stocké et transmis en chaîne séquentiellement

| 9 | 6 | 5 |
|---|---|---|
| 1001 | 0110 | 0101 |

Addition BCD

| | | |
|---|---|---|
| 6 | 0110 | |
| +6 | +0110 | addition binaire |
| =12 | =1100 > 1001 test | |

Si test =oui ajustement +6

= 0001 0010 lu en BCD

# Code analytique : Gray

Evite les problèmes de transition
Un seul bit modifié d'une code au suivant (adjacence)

De plus, ce code est cyclique et à symétries multiples

000
001
011
010
110
111
101
100

Construction au tableau noir

# Codes analytiques (p parmi n)

| Chiffre | Télécommunications 01236 | POSTNET 74210 | IBM 7070, 7072, 7074 01234 |
|---|---|---|---|
| 0 | 01100 | 11000 | 01100 |
| 1 | 11000 | 00011 | 11000 |
| 2 | 10100 | 00101 | 10100 |
| 3 | 10010 | 00110 | 10010 |
| 4 | 01010 | 01001 | 01010 |
| 5 | 00110 | 01010 | 00110 |
| 6 | 10001 | 01100 | 10001 |
| 7 | 01001 | 10001 | 01001 |
| 8 | 00101 | 10010 | 00101 |
| 9 | 00011 | 10100 | 00011 |
| A | N/A | N/A | 1—10 |
| - | N/A | N/A | 1—01 |
| + | N/A | N/A | 0—11 |

Chaque nombre décimal codé sur n bits, dont p valent 1 et n-p valent 0

Permet de détecter jusqu'à une erreur.

Exemple : 2 parmi 5

# Autres codes

Codes à bit de parité : dans ces codes un bit est rajouté à l'information transmise de sorte que le nombre total de 1 soit pair (ou impair), selon le type convenu de parité. Il y a plusieurs définitions possible de la parité (longitudinale, transversale, orthogonale).

Codes à redondance : des codes où l'information est transmise en plusieurs exemplaire. On augmente ainsi la probabilité de détecter une erreur de transmission et de pouvoir éventuellement la corriger (Hamming).

# CODES CONVENTIONNELS

(ASSOCIATION **OBJET⇔CODE** PAR CONVENTION)

# Code Baudot (5 bits) : historique

## Code télégraphique Alphabet International (AI) nº 1 ou Alphabet International (AI) nº 2 ou code CCITT nº 2

(remplace le morse : de 1000 à 3000 mots par heure)

Deux plages de 28 caractères et 4 codes de contrôle

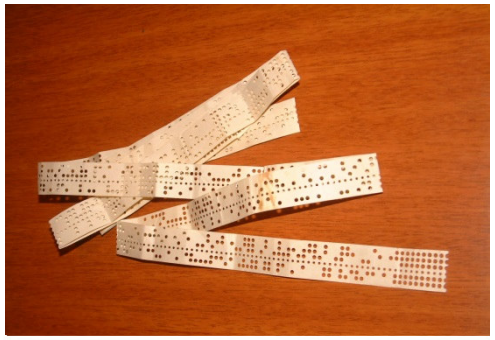
NULL : repos ou espace

LTRS : ➔ mode lettres

FIGS : ➔ mode chiffres

DEL : annule le précédent

**Baudot ➔ baud**
(unité de débit)



1874, Code Baudot

| | A | É | E | I | O | U | Y | B | C | D | F | G | H | J | FI | DEL | K | L | M | N | P | Q | R | S | T | V | W | X | Z | t | LT | NU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | & | 2 | o | 5 | 4 | 3 | 8 | 9 | 0 | f | 7 | h | 6 | GS | | ( | = | ) | n° | % | / | - | ; | ! | ' | ? | , | : | . | RS | LL |
| V | | | | | | | | | | | | | | | | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | |
| IV | | | | | | | | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | | | | | | | | | | |
| III | | | | ● | ● | ● | ● | ● | ● | ● | ● | | | | | | | | | ● | ● | ● | ● | ● | ● | ● | ● | ● | | | | |
| II | | ● | ● | ● | ● | | | | | ● | ● | ● | ● | | | | | ● | ● | ● | ● | | | | | | ● | ● | ● | ● | | |
| I | ● | ● | | | ● | ● | | | ● | ● | | | ● | ● | | | ● | ● | | | ● | ● | | | ● | ● | | | ● | ● | | |

# Code EBCDIC

Extended Binary Coded Decimal Interchange Code

Porté par IBM

(extension du code 5 bits, 1964, 6 versions)

# Code ASCII

## American Standard Code for Information Interchange (1960)

| Dec | Hex | Name | Char | Ctrl-char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Null | NUL | CTRL-@ | 32 | 20 | Space | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | Start of heading | SOH | CTRL-A | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | Start of text | STX | CTRL-B | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | End of text | ETX | CTRL-C | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | End of xmit | EOT | CTRL-D | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | Enquiry | ENQ | CTRL-E | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | Acknowledge | ACK | CTRL-F | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | Bell | BEL | CTRL-G | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | Backspace | BS | CTRL-H | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | Horizontal tab | HT | CTRL-I | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | Line feed | LF | CTRL-J | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | Vertical tab | VT | CTRL-K | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | Form feed | FF | CTRL-L | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | Carriage feed | CR | CTRL-M | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | Shift out | SO | CTRL-N | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | Shift in | SI | CTRL-O | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | Data line escape | DLE | CTRL-P | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | Device control 1 | DC1 | CTRL-Q | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | Device control 2 | DC2 | CTRL-R | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | Device control 3 | DC3 | CTRL-S | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | Device control 4 | DC4 | CTRL-T | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | Neg acknowledge | NAK | CTRL-U | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | Synchronous idle | SYN | CTRL-V | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | End of xmit block | ETB | CTRL-W | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | Cancel | CAN | CTRL-X | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | End of medium | EM | CTRL-Y | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | Substitute | SUB | CTRL-Z | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | Escape | ESC | CTRL-[ | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | File separator | FS | CTRL-\ | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | Group separator | GS | CTRL-] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | Record separator | RS | CTRL-^ | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | Unit separator | US | CTRL-_ | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | DEL |

7 bits

# Code ANSI (Latin 1 Windows)

## American National Standards Institute

Ref : http://ascii-table.com/ansi-table.php

Poids fort

Poids faible

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

Extension de l'ASCII
➔ 8 bits

# Code UNICODE

## Lié à ISO/CEI 10646 (évolution UTF-8)

Un répertoire de 137 929 caractères (une centaine d'écritures)
Un ensemble de tableaux de codes pour référence visuelle
Une méthode de codage et plusieurs codages de caractères
Une énumération des propriétés de caractère
lettres majuscules, minuscules, APL, symboles, ponctuation, etc.
APL : langage de description de traitement de l'information
Un ensemble de fichiers de référence des données informatiques

et Un certain nombre d'éléments liés : Règles de normalisation, de décomposition, de tri, de rendu et d'ordre d'affichage bidirectionnel

# Unicode + transformation = UTF

## Universal Character Set Transformation Format (UTF8/16/32)

### Codage : de 1 à 4 octets (extrait de la table d'interprétation)

| Caractères codés | Représentation binaire UTF-8 | Premier octet valide (hexadécimal) | Signification |
|---|---|---|---|
| **U+0000 à U+007F** | 0xxxxxxx | 00 à 7F | 1 octet, codant 7 bits |
| **U+0080 à U+07FF** | 110xxxxx 10xxxxxx | C2 à DF | 2 octets, codant 11 bits |
| **U+0800 à U+0FFF** | 11100000 101xxxxx 10xxxxxx | E0 (le 2e octet est restreint de A0 à BF) | 3 octets, codant 16 bits |
| **U+1000 à U+1FFF** | 11100001 10xxxxxx 10xxxxxx | E1 | |
| **U+2000 à U+3FFF** | 1110001x 10xxxxxx 10xxxxxx | E2 à E3 | |
| **U+4000 à U+7FFF** | 111001xx 10xxxxxx 10xxxxxx | E4 à E7 | |
| **U+8000 à U+BFFF** | 111010xx 10xxxxxx 10xxxxxx | E8 à EB | |
| **U+C000 à U+CFFF** | 11101100 10xxxxxx 10xxxxxx | EC | |
| **U+D000 à U+D7FF** | 11101101 100xxxxx 10xxxxxx | ED (le 2e octet est restreint de 80 à 9F) | |
| **U+E000 à U+FFFF** | 1110111x 10xxxxxx 10xxxxxx | EE à EF | |

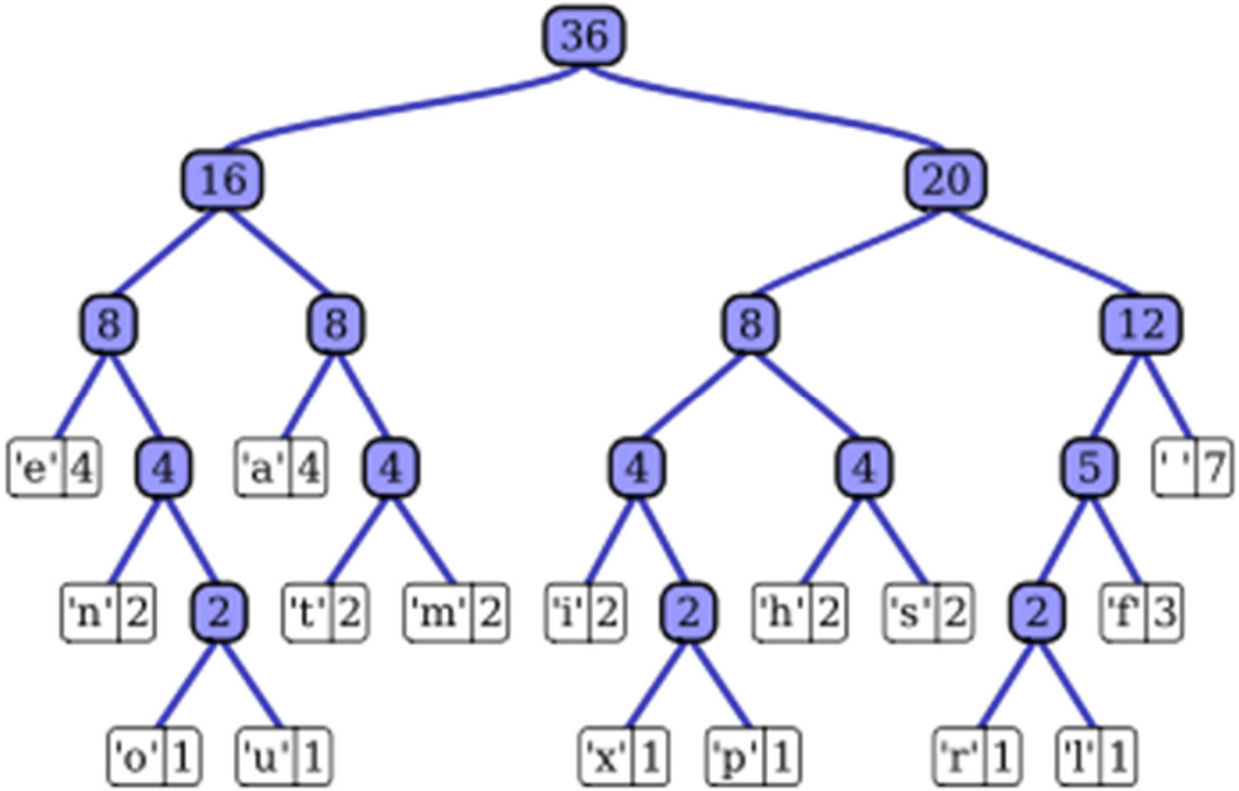
# Code de Huffmann

Longueur du code associé à la probabilité d'apparition : dynamique

« *this is an example of a huffman tree* ».

- Analyse
- Construction table
- Codage

En cas de transmission il faut envoyer la table




« *Wikipédia* » : 101 11 011 11 100 010 001 11 000

24 bits au lieu de 63 en ASCII