Chapitre 8 (3b)

REPRÉSENTATION INTERNE DES DONNÉES

POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Représentation des nombres

- Notation positionnelle
  - La valeur de chacun des chiffres formant le nombre est pondérée par la valeur de la base choisie élevée à une puissance correspondant à la position occupée par le chiffre

    $4752 = 4\times10^3 + 7\times10^2 + 5\times10^1 + 2\times10^0$
- Système positionnel décimal (base dix)
  - Le plus utilisé par les humains.
  - Pas toujours pratique en informatique
    - L'élément de mémoire de base a deux valeurs possibles dans un ordinateur numérique standard.
    - Dix n'est pas une puissance de deux.
- La base 2
  - $10111_{\text{base } 2} = 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0 = 23_{\text{base } 10}$
  - Bit le plus significatif: bit ayant la pondération la plus élevée (à gauche)
  - Bit le moins significatif: bit ayant la pondération la moins élevée (à droite)

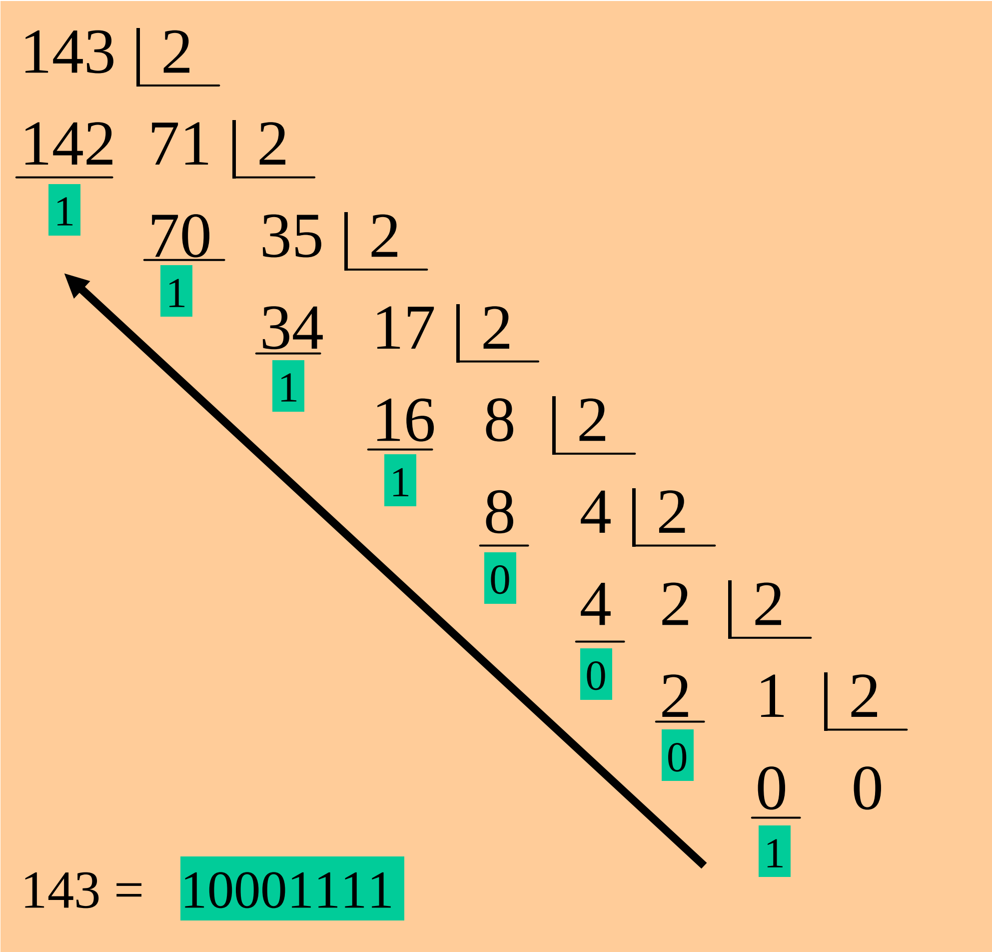
Conversion de base

- De base X vers base Y.
  - Il faut savoir calculer soit en base X ou en base Y
    - Sinon passer par une base dans laquelle on sait calculer
  - Les humains calculent généralement en base dix, pas les ordinateurs.
- Calculer dans la base destination (Y)
  - Tous les nombres sauf l'original seront en base Y
  - Se fait par multiplication par X et addition du chiffre
  - Exemple: base 2 vers base 10

    $10111_{\text{base } 2} = 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0 = 23$

    $= ((((1\times2) + 0)\times2 + 1)\times2 + 1)\times2 + 1 = 23$
- Calculer dans la base source (X)
  - Tous les nombres sauf le résultat seront en base X
  - Se fait par division par Y, le reste est le chiffre
  - Exemple: base 10 vers base 2 (page suivante)

Conversion décimale-binaire



Base octale

- Base octale, base $8 = 2^3$ (existait avant l'informatique)
  - Chiffre entre 0 et 7
- Conversion de la base 2 à la base 8
  - Regrouper les bits par paquet de trois à partir du bit le moins significatif
  - Calculer la valeur associée à chaque paquet

$$1010100111 = 1 \quad 010 \quad 100 \quad 111$$
$$1 \quad 2 \quad 4 \quad 7 = 1247_{\text{base } 8}$$

Base hexadécimale

- Base hexadécimale, base $16 = 2^4$ (inventée pour l'informatique)
  - Chiffre de 0 à 9 et lettre de A à F (pour 10 à 15)
  - Chaque chiffre est la moitié d'un octet (appelé *nibble*)
- Conversion de la base 2 à la base 16
  - Regrouper les bits par paquet de quatre à partir du bit le moins significatif
  - Calculer la valeur associée à chaque paquet

1010100111 =	10	1010	0111	
	2	A	7	$= 2A7_{\text{base 16}}$

Représentation des entiers signés

- Le bit le plus significatif, le plus à gauche précise le signe du nombre: 0 indique un nombre positif

  1 indique un nombre négatif
- Exemple, représentation de 143 sur 16 bits

Bit de signe

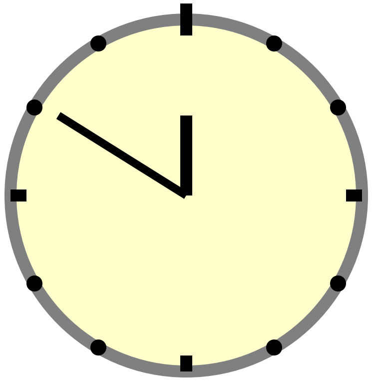
0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1

143

- **Attention: 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 <u>n’est pas</u> -143

Représentation des entiers signés: Complément à 2

- Analogie avec l'horloge:
  - « et 50 » ou « moins 10 » est la même heure, car les minutes sont modulo 60 (60 – 10 = 50)
- En informatique: puissance de 2
  - Sur 8 bits, « et 246 » = « moins 10 », car $2^8 - 10 = 246$.
  - Sur 16 bits « et 65526 » = « moins 10 », car $2^{16} - 10 = 65526$.
    - En base 2:
      10000000000000000
      – 1010
      = 1111111111110110



Représentation des entiers signés: Complément à 2

- Notons que: $2^n - x \;=\; (2^n - 1) - x + 1$

  $(2^n - 1) - x \;=\;$ inverser les bits de x, sur n bits (si $x < 2^n$)
- Donc, pour obtenir la représentation du nombre négatif il suffit d'inverser tous les bits du nombre positif et y ajouter 1.
- Par exemple, -143 s'obtient

	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	+ 143
	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0	Inversion des bits
+																1	Additionne 1
	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	1	= -143

Représentation des entiers signés: Complément à 2

- Représentation unique de 0
  - +0 = - 0 = 0000000…000
- Intervalle de la représentation
  - [-(2^{n-1}) , (2^{n-1}- 1)] où *n* est le nombre de bits utilisés
- Addition simple
  - Méthode apprise à l'école primaire pour additionner (mais en base 2)

Addition en complément à 2

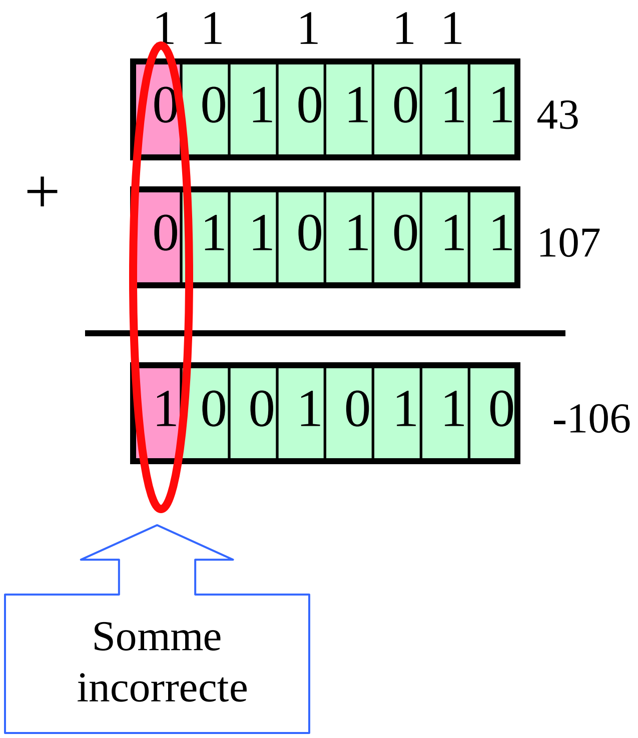
- Additionner normalement (bit à bit, avec retenue entre les bits)
- Ignorer la retenue finale
- Validité de la réponse
  - Si les deux opérandes sont de signe opposé, la réponse est exacte
  - Si les deux opérandes sont de même signe, mais que la réponse est de signe opposé, il y a débordement, la réponse est inexacte

		1		1		1	1		
	1	0	1	0	1	0	1	1	-85
+	0	0	1	0	1	0	1	1	43
	1	1	0	1	0	1	1	0	-42

				1		1	1		
	0	0	1	0	1	0	1	1	43
+	0	0	0	0	1	0	1	1	11
	0	0	1	1	0	1	1	0	54

Deux sommes **correctes**

Addition



Sur 8 bits l'intervalle des valeurs possibles est

$[-(2^{8-1}) , (2^{8-1}- 1)] = [-128, 127]$

Puisque 43 + 107 = 150 est une valeur à l'extérieur de l'intervalle, nous sommes en présence d'un cas de débordement.

La valeur obtenue -106 est inexacte.

Entier non signé

- L'entier non signé est un entier positif.
- Le bit de signe n'est plus nécessaire, tous les bits contribuent donc positivement au nombre. L'intervalle devient alors
  - $[0, 2^n - 1]$ où n est le nombre de bits
  - Par exemple, pour une représentation sur 8 bits
    - Entier non signé : $[0, 2^8 - 1] = [0, 255]$
    - Entier signé, complément à 2: $[-2^7, 2^7 - 1]=[-128,127]$

Les types entiers en C/C++

Type	Étendue	Espace
signed char	-128 ... 127	1 octet
short (int)	-32 768 ... 32 767	2 octets
int *(entier le plus standard)*	-2 147 483 648 ... 2 147 483 647	4 octets
long (int)	-2 147 483 648 ... 2 147 483 647	4 octets
long long (int)	-9 223 372 036 854 775 808 … 9 223 372 036 854 775 807	8 octets
unsigned char	0 ... 255	1 octet
unsigned short (int)	0 ... 65 535	2 octets
unsigned (int)	0 ... 4 294 967 295	4 octets
unsigned long (int)	0 ... 4 294 967 295	4 octets
unsigned long long (int)	0 ... 18 446 744 073 709 551 615	8 octets

Note: ces valeurs peuvent varier selon l'ordinateur ou le compilateur.

Opérations sur les bits en C/C++

Opérateur	Description	Exemple	
~	Effectue le complément à 1 en inversant tous les bits	~12	~12 =~(00001100) = (11110011) = 243 ou -13
&	Réalise un ET binaire sur les bits: 1&1 donne 1 1&0 donne 0 0&1 donne 0 0&0 donne 0	7 & 2	7 00000111 & & 2 00000010 ____ __________ 2 00000010
|	Réalise un OU binaire sur les bits: 1|1 donne 1 1|0 donne 1 0|1 donne 1 0|0 donne 0	7 | 3	7 00000111 | | 3 00000011 ____ ________ 7 00000111

Opérations sur les bits en C/C++ (suite)

Opérateur	Description	Exemple	
^	Réalise un OU-EXCLUSIF binaire sur les bits: 1^1 donne 0 1^0 donne 1 0^1 donne 1 0^0 donne 0	5 ^12	5 000000101 ^ 12 000001100 ___ _________ 9 000001001
>>	Réalise un décalage à droite du nombre de bits spécifié	7 >>2	7 >> 2 00000111 >> 2 _______________ 00000001
<<	Réalise un décalage à gauche du nombre de bits spécifié	11 << 3	11 << 3 00001011 << 3 _______________ 01011000

Les nombres réels

- Les nombres réels s'expriment principalement sous deux formes:
  - La notation décimale
  - La notation scientifique
- La notation scientifique est celle qui permet un format de mémorisation standardisé. Il incorpore trois éléments: la mantisse, la base et l'exposant. Par exemple, $2,7 \times 10^5$
  - 2,7 est la mantisse
  - 10 est la base
  - 5 est l'exposant

Les nombres réels

- La mantisse doit respecter la règle:
  - $1 \leq |\text{Mantisse}| < \text{Base}$

- Ainsi
  - $0{,}1245 \times 10^{-23}$ est inacceptable
  - $12{,}45 \times 10^{-25}$ est inacceptable
  - $1{,}245 \times 10^{-24}$ est la seule et unique représentation acceptable
- Connaissant la base, il est possible de représenter le nombre à l'aide du triplet:
  - (Signe, Exposant, Mantisse)

Norme IEEE754

- Un nombre réel binaire est mémorisé en respectant la norme IEEE754

	Nombre bit du signe	Nombre de bits de l'exposant	Nombre de bits de la mantisse
Simple précision 32 bits	1	8	23
Double précision 64 bits	1	11	52

Norme IEEE754

- Règles
  - Le bit de signe est à 0 pour un nombre positif ou à 1 pour un nombre négatif
  - La mantisse est représentée par notation positionnelle. Le premier bit de la partie entière est sous-entendu puisque toujours 1.
  - L'exposant est représenté par excès de $2^{n-1} - 1$. Il s'agit de l'astuce utilisée pour combler l'absence du signe de l'exposant. En simple précision l'excès est de 127 tandis qu'en double précision l'excès est de 1023.

IEEE754 : Exemple

- Conversion de 132.147 en format simple précision = 01000011000000100001001011010010

Partie entière= 132		**Partie fractionnaire= .147**	
nbre	nbre modulo 2	nbre	Partie entière de (2×nbre)
132	0	0.147	0
66	0	0.294	0
33	1	0.588	1
16	0	0.176	0
8	0	0.352	0
4	0	0.704	1
2	0	0.408	0
1	1	0.816	1
0		0.632	1
		0.264	0
		0.528	1
		0.056	0
		0.112	0
		0.224	0
		0.448	0
		0.896	1
		0.792	1

Dernier bit est arrondi à 1

$$\overbrace{10000100}^{132} \,.\, \overbrace{0010010110100001}^{147}\boxed{1}$$

$1.\,00001000010010110100010 \times 2^7$

Signe = 0

Exposant = 7 + 127 = 134

= 10000110

Mantisse = 00001000010010110100010

S	Exposant	Mantisse
0	10000110	00001000010010110100010

IEEE754: Exceptions

- Dans les deux format, la norme prévoit des cas spéciaux:

Signe	Exposant	Mantisse
1 ou 0	0000…00	0000…000

Zéro (positif et négatif)

Signe	Exposant	Mantisse
1 ou 0	0000…00	xxxx…xx (!=0)

Dénormalisé (underflow)

Signe	Exposant	Mantisse
1 ou 0	1111…11	000…000

Infinis ou débordement (overflow) +1/+0 = +inf

Signe	Exposant	Mantisse
1 ou 0	1111…11	xxxx…xx (!=0)

NAN (Not A Number)
0/0 = -NAN , sqrt(-1) = -NAN
Les seules valeurs différentes d'elles-mêmes.

IEEE754: Caractéristiques

- Réel maximal

Signe	Exposant	Mantisse
1 ou 0	1111…10	1111…11

$= 1.111..11 \times 2^{(111\ldots10) - \text{Excès}}$

$1.111..11 = 1 + 2^{-1} + 2^{-2} + 2^{-3} + \ldots + 2^{-M}$

où M=nbre de bits de la mantisse

La M ième somme partielle de la série géométrique avec $|c|<1$ est

$$1 + c^1 + c^2 + c^3 + \ldots + c^M = \frac{1 - c^{M+1}}{1 - c}$$

Dans notre cas, $c = ½$

$$1.111..11 = \frac{1 - (½)^{M+1}}{½} = 2 - (½)^M$$

$$= 2 - 2^{-M}$$

$111\ldots10 - \text{Excès} = [(2^E-1)-1] - (2^{E-1}-1)$

où E=nbre de bits de l'exposant

$111\ldots10 - \text{Excès} = (2^E-2) - (2^{E-1}-1)$

$= 2(2^{E-1}-1) - (2^{E-1}-1)$

$= (2^{E-1}-1)$

Finalement,

$1.111..11 \times 2^{(111\ldots10) - \text{Excès}}$

$= (2 - 2^{-M}) \times 2^{(2^{E-1}-1)}$

IEEE754: Caractéristiques

- Réel normalisé minimal

Signe	Exposant	Mantisse
1 ou 0	0000…01	0000…00

$= 1.0\times 2^{(0000\ldots01)\text{ - Excès}}$

0000…01 – Excès = 1 - (2^{E-1}-1)

où E=nbre de bits de l'exposant

0000…01 – Excès = 2 - 2^{E-1}

- Réel dénormalisé minimal non nul

Signe	Exposant	Mantisse
1 ou 0	0000…00	0000…01

$= \underline{\mathbf{0.}}0000\ldots01\times 2^{\underline{\mathbf{1 - Excès}}}$

$= \quad 2^{(2 - 2^{E-1} - M)}$

Finalement,

$1.0\times 2^{(0000\ldots01) – \text{Excès}}$

$= \; 1.0 \times 2^{(2 - 2^{E-1})}$

$= \quad 2^{(2 - 2^{E-1})}$

IEEE754: Caractéristiques

- La granularité est la différence existant entre un nombre réel et le nombre réel suivant
- La granularité correspond tout simplement au dernier bit de la mantisse

Nombre réel :

S	Exposant	Mantisse
0	01111101	10111101110111011101100

Nombre réel suivant:

S	Exposant	Mantisse
0	01111101	10111101110111011101101

- La granularité = 2^{-M} où M est le nombre de bits de la mantisse

IEEE754: Caractéristiques

	Simple précision	Double précision
Réel maximal	$3{,}4 \times 10^{38}$	$1{,}8 \times 10^{308}$
Réel minimal normalisé	$1{,}2 \times 10^{-38}$	$2{,}2 \times 10^{-308}$
Granularité	$5{,}96 \times 10^{-8}$	$1{,}11 \times 10^{-16}$

Les types réels en C/C++

Type	**Étendue**	**Chiffres significatifs** *(base dix)*	**Espace**
float	$\pm 1{,}17549 \times 10^{-38} \ldots \pm 3{,}40282 \times 10^{38}$	~7,2	4 octets
double *(réel le plus standard)*	$\pm 2{,}22507 \times 10^{-308} \ldots \pm 1{,}79769 \times 10^{308}$	~15,9	8 octets
long double *(sur VisualC 2010)*	*identique à* double		
long double *(sur GCC / Clang)*	$\pm 3{,}36210 \times 10^{-4932} \ldots \pm 1{,}18973 \times 10^{4932}$	~19,2	12 octets (10+2 octets)

Note: ces valeurs peuvent varier selon l'ordinateur ou le compilateur. Particulièrement le « long double ».

Conversion implicite

(version plus complète)

	long double	double	float	unsigned long	long	unsigned	int, short, char, bool
long double	long double						
double		double					
float			float				
unsigned long				unsigned long			
long					long		
unsigned						unsigned	
int, short, char, bool							int

Conversion implicite

Expression	Type	Expression	Type
c-s/i	int	u+c-i	unsigned
c-5	int	u+c-i-u_long	unsigned long
u/2.0-i	double	7*5*u_long	unsigned long
c + 1.24	double	long_d+c-24	long double
d*3	double	f+3*s-i	float
4+i/u_long	unsigned long	long_d-f*2	long double

Les caractères

- Les nombres constituent le langage des ordinateurs. Pour obtenir une lettre il faut un système de conversion du nombre à la lettre.
- Création du code ASCII (American Standard Code for Information Interchange).
- Le jeu de caractères ASCII standard correspond à un tableau de correspondance qui associe un nombre entier à une lettre, un chiffre, un symbole ou un code de contrôle.

Code ASCII

Lettres minuscules

a = 97	**b** = 98	**c** = 99	**d** = 100	**e** = 101
f = 102	**g** = 103	**h** = 104	**i** = 105	**j** = 106
k = 107	**l** = 108	**m** = 109	**n** = 110	**o** = 111
p = 112	**q** = 113	**r** = 114	**s** = 115	**t** = 116
u = 117	**v** = 118	**w** = 119	**x** = 120	**y** = 121
z = 122				

a = 61_{hex}

Lettres majuscules

A = 65	**B** = 66	**C** = 67	**D** = 68	**E** = 69
F = 70	**G** = 71	**H** = 72	**I** = 73	**J** = 74
K = 75	**L** = 76	**M** = 77	**N** = 78	**O** = 79
P = 80	**Q** = 81	**R** = 82	**S** = 83	**T** = 84
U = 85	**V** = 86	**W** = 87	**X** = 88	**Y** = 89
Z = 90				

A = 41_{hex}

H	**e**	**l**	**l**	**o**
01001000	01100101	01101100	01101100	01101111
72	101	108	108	111
48_{hex}	65_{hex}	$6C_{hex}$	$6C_{hex}$	$6F_{hex}$

Code ASCII

- Le chiffre a aussi son correspondant en caractère

chiffres	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
valeur ASCII	48	49	50	51	52	53	54	55	56	57
	30_{hex}	31_{hex}	32_{hex}	33_{hex}	34_{hex}	35_{hex}	36_{hex}	37_{hex}	38_{hex}	39_{hex}

Le nombre 107 en complément à deux

0	1	1	0	1	0	1	1

s’écrit en caractères ASCII

1	**0**	**7**
0 0 1 1 0 0 0 1	0 0 1 1 0 0 0 0	0 0 1 1 0 1 1 1
49	48	55
31_{hex}	30_{hex}	37_{hex}

UNICODE Réf. fr.wikipedia.org

- Unicode, dont la première publication remonte à 1991, a été développé dans le but de remplacer l'utilisation de pages de code nationales.
- Une **page de code** est un standard informatique national/local qui vise à donner à tout caractère national/local un numéro. Cependant les pages de code ayant été développées sur des bases nationales, elles ne permettent pas facilement l'échange de document entre différents pays. Les pages de code les plus connues sont :
  - page de code 437 américain, graphique (original IBM PC / MS-DOS)
  - page de code 850 multilingue européen
  - Latin-1 (ISO 8859-1) / « ANSI » Windows-1252
  - VISCII vietnamien
  - Windows-1258 (vietnamien)

- Tous les systèmes les plus utilisés dans le monde sont représentés, ainsi que des règles sur la sémantique des caractères, leurs compositions et la manière de combiner ces différents systèmes (par exemple, comment insérer un système d'écriture de droite à gauche dans un système d'écriture de gauche à droite)

UNICODE

- Points de code (numéro des « caractères »)
  - ASCII de 0 à $7F_{hex}$ (7bits)
  - Pages de code, généralement 8bits
  - Unicode1.1 de 0 à $FFFD_{hex}$ (16bits) 1993
  - Unicode actuellement de 0 à $10FFFF_{hex}$ (~21bits) (pas tous définis)
- Numéros donnés aux symboles/sens et non à leurs représentation graphique (*glyphes*)
  - Un même numéro peut avoir plusieurs glyphes:
    - 0072 (r) devant différentes lettres *ra re ri ro ru rh rs*
  - Un glyphe peut avoir plusieurs numéros:
    - é = 00E9 (é) ou 0065 0301 (é)
- Un point de code est utilisé pour une seule signification
  - Une police de caractères grecques n'a pas le 'α' à la place du 'a'

UNICODE

- Représentations:
  - UTF-8 (1 à 4 octets/code), UTF-16 (2 ou 4 octets/code), UTF-32
- La version actuelle est 6.3 de septembre 2013
  - Définit 110 187 « caractères » (incluant les directives).
- Problèmes actuels:
  - L'unification des différentes langues CJK a des problèmes à cause de calligraphies légèrement différentes
    - Utiliser une police faite pour la langue particulière.
  - Les polices OpenType ne peuvent avoir plus de 65 536 glyphes.
  - Certains programmes ne supportent pas les codes de plus de 16 bits.

Caractères 200 à 309 pour la police Arial Unicode

	0	1	2	3	4	5	6	7	8	9
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ	Ā	ā	Ă	ă
260	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č
270	Ď	ď	Đ	đ	Ē	ē	Ĕ	ĕ	Ė	ė
280	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ	Ġ	ġ
290	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī
300	Ĭ	ĭ	Į	į	İ	ı	Ĳ	ĳ	Ĵ	ĵ