Mathématiques appliquées à l'informatique
1 – Arithmétique : numération

Bertrand LIAUDET

SOMMAIRE

SOMMAIRE 1

CHAPITRE 1 - NUMERATION 3

0 - Préambule 3
- Principes 3
- Références 3
- Le problème : calcul d'un développement limité en C 3
- Les exercices 3

1 - Systèmes de numération des entiers naturels 6
- Numération 6
- Système de numération 6
- Système « naturel » ou système décimal : la base 10 6
- Numération romaine 6
- Numération babylonienne (environ 2000 avant JC) : base 60 7
- Base 2 : système binaire 7
- Base 8 : système octal 8
- Base 16 : système hexadécimal 8
- Généralisation et notation $(n)_p$ ou $n_{(p)}$ 8
- Remarques sur les bases 2, 8, 16 9
- Puissances de 2 remarquables 9

2 - Conversion d'une base à une autre 10
- Présentation 10
- Conversions vers la base 10 : abordées dans la présentation des bases 2, 8 et 16 10
- Conversions de la base 10 vers les bases 2, 8 ou 16 10
- Conversion de la base 2 à la base 16 et inversement 12
- Utilisations 12

3 - Numération des réels 13
- Propriété : écriture des réels en somme des puissances de la base 13
- Conversions vers la base 10 13
- Conversions de la base 10 vers la base 2 13
- Utilisation 13
- Opérations élémentaires 14

4 - Arrondi et précision **15**
5 - Codage de l'information dans les ordinateurs **17**
Notion de BIT 17
Représentation externe et représentation binaire 17
Représentation binaire et représentation physique 17
La numérisation 17
Notion d'octet 17
6 - Codage des entiers dans les ordinateurs **19**
Code binaire signé 19
Complément à 1 19
Complément à 2 : c'est la représentation dans les ordinateurs 20
Conclusion : représentation physique, représentation logique et optimisation 21
7 - Codage des nombres réels dans les ordinateurs **22**
Code à virgule fixe 22
Code à virgule flottante 22
Représentation dans les ordinateurs 23
Standard IEEE 754 (1985) – float et double 25
8 - Codage des caractères – ASCII – ISO 8859 – UNICODE – UTF8 **27**
Présentation 27
Le code ASCII 27
Les codes ASCII étendus : codes ISO 8859 28
L'UNICODE et UTF8 29

Dernière édition : février 2017

CHAPITRE 1 - NUMERATION

0 - Préambule

Principes

Partir d'une application pour remonter à l'objet mathématique théorique qu'on étudie abstraitement ensuite.

Ensuite on fait des exercices théoriques (des gammes), des exercices appliqués et des traductions algorithmiques.

Le but est de rentrer dans une logique de mathématiques appliquées, c'est-à-dire : les maths comme un outil et pas une abstraction creuse.

Objectif : ne pas avoir peur des maths ! Les voir comme un outils pratique qui permet de régler des problèmes efficacement.

Références

Mathématique pour l'informatique – BTS SIO – Dunod – 2015 : Chapitre 1, pp. 3-33.

Méthodes mathématiques pour l'informatique – IUT-Licence-Ecole d'ingénieurs-CNAM – Dunond 2013.

Le problème : calcul d'un développement limité en C

En mathématiques, un développement limité d'une fonction en un point est une approximation polynomiale de cette fonction, c'est-à-dire l'écriture de cette fonction sous la forme d'une somme allant à l'infini.

Par exemple : $\sin(x) = x - x^3 / 3! + x^5/5 ! - x^7 / 7! + x^9/9 ! \ldots$

Si on veut calculer sin(x) avec un ordinateur, le problème est de savoir quand on arrête la somme.

Autrement dit, à partir de quelle valeur de N, N ! est trop grand pour être représenté, ou 1/N ! est trop petit pour être représenté.

En langage C, il existe un fichier float.h qui définit 2 constantes :

FLT_EPSILON plus petit nombre e tel que $1.0 + e \neq 1.0$

FLT_MAX plus grand nombre représentable

On veut écrire un algorithme qui permettent de calculer sin(x).

Pour bien comprendre cet exercice, mieux vaut comprendre le standard IEEE 754

Les exercices

1 - Systèmes de numération des entiers naturels – 3p

1. Imaginez une version adaptée du système romain qui permettent d'écrire les entiers jusqu'à l'infini. Demandez vous comment écrire 10 mille, 100 mille, 1 million, 100 millions, 1 milliards.

Ecrivez alors 2000, 3212, 397 742, 253 521 430.

2. Additionner 4215 + 382 en système babylonien
3. Combien valent précisément et approximativement 2^8, 2^{10}, 2^{20}, 2^{30}

2 - Conversion d'une base à une autre – 3p

4. Ecrire en base 10 les nombres suivants : $101010_{(2)}$, $1F2_{(16)}$
5. Ecrire en base 2 les nombres suivants : 238, $30A_{(16)}$
6. Ecrire en base 16 les nombres suivants : 8251, $10010111_{(2)}$

7. Codez sous excel la méthode par la division euclidienne pour des entiers en base 10. Le principe de résolution consiste à avoir un tableau avec dividende, diviseur, quotient et reste (le diviseur correspond à la base cible).

Dans un second temps, on peut se doter d'une colonne avec le chiffre trouvé sous forme de caractère (pour gérer les chiffres de la base 16) et d'une colonne permettant de construire au fur et à mesure le résultat.

3 - Numération des réels – 2p

8. Soit 2 entiers en base 2 : a=10110 et b=1101. Calculer a+b, a-b, a*b
9. Soit 2 réels en base 2 : a=110,01 + b=100,1. Calculer a+b, a-b, a*b
10. Soit 2 entiers en base 16 : a=D1 et b=19. Calculer a+b, a-b, a*b

4 - Arrondi et précision – 1p

11. Quel est l'arrondi de $1101011_{(2)}$ à $100_{(2)}$ près ?
12. Quel est l'arrondi de $10{,}011_{(2)}$ à $0{,}1_{(2)}$ près ?

5-8 Codage de l'information dans les ordinateurs – 10p

13. Sachant qu'un livre de 200 pages contient environ 300 000 caractères espaces compris et qu'un caractère est codé sur un octet ; sachant que le traitement de texte triple en moyenne la taille du texte (1000 caractères deviennent 3000) ; calculer le nombre de livres que vous pourrez enregistrer sur une clé de 2 GO.

14. Donnez la traduction à laquelle correspond le mot de 4 octets codé en hexadécimal suivant : 49 55 50 31selon qu'on le lit comme :

- un entier signé,
- un entier représenté en complément à 2,
- un nombre représenté en virgule flottante simple précision suivant la norme IEEE 754,
- une suite de caractères ASCII (représentés chacun sur 8 bits, le bit de plus fort poids étant inutilisé et codé à 0)

15. Convertir le nombre décimal 8,625 en virgule flottante suivant la norme IEEE 754 simple précision.

16. Calcul de FLT_EPSILON

Le langage C définit une constante : FLT_EPSILON

C'est la différence entre 1 et la plus petite valeur représentable supérieure à 1.

Autrement dit, c'est la différence la plus petite qu'il puisse y avoir entre deux nombres X1 et X2. Donc, avec X1 ≥ X2 si X1-X2 < FLT_EPSILON, on peut considérer que X1 = X2.

En général, FLT_EPSILON = 1,19209 e^{-07}

La précision est de 6 chiffres : elle est fixée par une autre constante du C : FLT_DIG

On va calculer « logiquement » FLT_EPSILON

- ***Représentation de 1 en float***

  Convertissez 1 en float : $1_{(float)}$

- ***Représentation « logique » de 1 + FLT_EPSILON en float***

  A partir de la définition de FLT_EPSILON et $1_{(float)}$, déduisez $(1+FLT\_EPSILON)_{(float)}$

- ***Valeur de FLT_EPSILON en binaire***

  A partir de $(1+FLT\_EPSILON)_{(float)}$, déduisez $FLT\_EPSILON_{(2)}$

- ***Convertissez $FLT\_EPSILON_{(2)}$ en $FLT\_EPSILON_{(float)}$***

- ***Valeur de FLT_EPSILON en décimal***

  A partir de $FLT\_EPSILON_{(2)}$, calculez $FLT\_EPSILON_{(10)}$: vous pouvez utiliser une calculatrice !

- ***Valeur de FLT_EPSILON en binaire à partir de FLT_EPSILON en décimal***

  A partir de $FLT\_EPSILON_{(10)}$, calculez $FLT\_EPSILON_{(2)}$: vous pouvez utiliser une calculatrice ! Que constatez-vous ?

17. Convertissez le nombre 011001010 11100010 10101011 11000101 représenté sous forme d'un nombre entier binaire signé de 4 octets en un nombre réel représenté selon la norme IEEE 754 simple précision.

Convertissez le nombre réel obtenu en entier binaire signé codé sur 32 bits et comparez le résultat obtenu avec le nombre entier de départ.

Quelles conclusions en déduisez-vous ?

1 - Systèmes de numération des entiers naturels

Numération

Mode de représentation des nombres.

Système de numération

Un système de numération décrit la façon avec laquelle les nombres sont représentés.

Système « naturel » ou système décimal : la base 10

Présentation

Les entiers naturels sont 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, etc.

Les entiers s'écrivent avec des chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Les chiffres sont aux entiers ce que les lettres sont aux mots.

Le système de numération avec 10 chiffres, c'est le système décimal, ou système naturel.

La base 10 est la base « naturel ». 10 correspond au nombre de chiffres et au premier nombre à s'écrire avec 2 chiffres.

Système positionnel

Le système décimal est un **système positionnel** : Chaque position du chiffre dans le nombre possède un poids (unité, dizaine, centaine, etc.)

Un entier peut s'écrire sous la forme d'une addition

$4134=4x10^3 +1x10^2 +3x10^1 +4x10^0$

$(a_n, .., a_1, a_0) = a_n*10^n + .. + a_1*10^1 + a_0*10^0 = \Sigma_{i=0}^{n} a_i*10^i$

a_0 est le chiffre de poids faible. a_n est le chiffre de poids fort.

Avantages du système décimal : la base 10 permet de faire facilement les opérations en connaissant les tables d'opération élémentaires.

Numération romaine

Les nombres romains sont : I, II, III, IV, V, VI, VII, VIII, IX, X, XI, XII, etc.

Romain	I	V	X	L	C	D	M
Système décimal	1	5	10	50	100	500	1000

Système constitué de symbole et pas de chiffre progressant de 1 en 1.

On peut mettre jusqu'à 3 I, X, C, M à la suite. Pas plus.

Pour obtenir l'équivalent de 4 I, X, C, on les fait suivre respectivement par V, L, D : IV, XL, CD

Théoriquement, le plus grand nombre exprimable est : 3999 (MMMCMXCIX) car le symbole 5000 n'existe pas et ne peut pas être précédé d'un M.

http://chiffreromain.com

Défaut majeur : ce système n'est pas pratique pour faire des opérations.

Numération babylonienne (environ 2000 avant JC) : base 60

Les clous représentent les unités : 1

Les chevrons représentent les dizaines : <

On peut juxtaposer jusqu'à 9 clous.

On peut juxtaposer jusqu'à 5 chevrons : le système est à base 60.

Quand on arrive à 6 chevrons, on les remplace par un clou dans la colonne voisine à gauche : c'est un système en partie positionnel.

43 : <<<<111

73 : 1 <111 (la « dizaine » à 60 est décalé des « unités » : 60 + 10 + 3)

3673 : 1 1 <111 (60*60 = 3600 + 60 + 10 + 3)

7273 : 11 1 <111 (2*60*60 = 7200 + 60 + 10 + 3)

Ce système permet de faire des opérations :

			1	1	
		3	6	9	8
+		7	2	8	3
=	1	0	9	8	1

		1	<
	1	1	<<< 11111111
+	11	1	<< 111
=	111	111	1

Défaut majeur : il n'y a pas de chiffres, d'où la complexité des calculs.

La base 60 est trop grande pour avoir des tables faciles à apprendre.

Base 2 : système binaire

Présentation

2 chiffres : {0 ; 1 }

Base 2	0	1	10	11	100	101	110	111	1000	1001	1010	1011	Etc.
Base 10	0	1	2	3	4	5	6	7	8	9	10	11	Etc.

$1011 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = 8 + 0 + 2 + 1 = 11$

Conversion en base 10

Position	10	9	8	7	6	5	4	3	2	1	0
Valeur en base 10	1024	512	256	128	64	32	16	8	4	2	1
Nombre en base 2	1	0	1	1	0	1	1	0	1	1	1

$101\ 1011\ 0111_{(2)} = 1x2^{10} + 1x2^8 + 1x2^7 + 1x2^5 + 1x2^4 + 1x2^2 + 1x2^1 + 1x2^0$

$101\ 1011\ 0111_{(2)} = 1024 + 256 + 128 + 32 + 16 + 4 + 2 + 1 = 1463$

Base 8 : système octal

Présentation

8 chiffres : {0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 }

Base 8	0	1	..	7	10	11	..	17	20	21	..	27	Etc.
Base 10	0	1	..	7	8	9	..	15	16	17	..	23	Etc.

0, 1, .., 7, 10, 11, .., 17, 20, 21, .., 76, 77, 100, 101, etc.

$1027 = 1x8^3 + 0x8^2 + 2x8^1 + 7x8^0 = 512 + 0 + 16 + 7 = 535$

Conversion en base 10

Position	4	3	2	1	0
Valeur en base 10	4096	512	64	8	1

Exemple :

Nombre en base 8	2	5	7	3	6

Calcul :

$25736_{(8)} = 2x8^4 + 5x8^3 + 7x8^2 + 3x8^1 + 6x8^0$

$25736_{(8)} = 8192 + 2560 + 448 + 24 + 6 = 11230$

Base 16 : système hexadécimal

Présentation

16 chiffres : {0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; A ; B ; C ; D ; E ; F}

Base 16	0	1	..	9	A	..	F	10	11	..	19	1A	Etc.
Base 10	0	1	..	9	10	..	15	16	17	..	25	26	Etc.

$101A = 1x16^3 + 0x16^2 + 1x16^1 + 10x16^0 = 4096 + 0 + 16 + 10 = 4122$

Conversion en base 10

Position	3	2	1	0
Valeur en base 10	4096	256	16	1

Exemple :

Nombre en base 16	2	A	8	F

Calcul :

$2A8F_{(16)} = 2x16^3 + Ax16^2 + 8x16^1 + Fx16^0$

$2A8F_{(16)} = 8192 + 2560 + 128 + 15 = 10895$

Généralisation et notation $(n)_p$ ou $n_{(p)}$

On peut écrire les nombres dans toutes sortes de bases à partir de 2.

On utilise la notation $(n)_p$ pour indiquer la base du nombre : $(57)_{10}$, $(1A)_{16}$, $(101101)_2$, $(7403)_8$

$(23)_{10} = (1011\ 1)_2 = (17)_{16}$

On écrit aussi $(17)_h$ à la place de $(17)_{16}$

On peut aussi écrire $17_{(16)}$

Quelle que soit la base, le système de numération est un système positionnel : chaque position du chiffre dans le nombre possède un poids.

Remarques sur les bases 2, 8, 16

Les bases les plus courantes, parce qu'utilisés en informatiques, sont les bases 2, 8 et 16.

2, 8 et 16 sont des puissances de 2 : $2 = 2^1$, $8 = 2^3$, $16 = 2^4$

Pour traduire de l'octal ou de l'hexadécimal en décimal, on peut travailler avec les puissances de 2.

Sachant que $(n^p)^q = n^{p*q}$, on a, par exemple : $16^3 = (2^4)^3 = 2^{4*3} = 2^{12} = 4096 = 2^{3*3} = (2^3)^4 = 8^4$

ou encore $8^3 = (2^3)^3 = 2^{3*3} = 2^9 = 512$.

Puissances de 2 remarquables

On a intérêt à connaître la série des puissances de 2, et particulièrement 2^8 et 2^{10}

2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{20}	2^{30}
2	4	8	16	32	64	128	256	512	1024	$\approx 10^6$	$\approx 10^9$

2 - Conversion d'une base à une autre

Présentation

Il y a 3 cas de conversions :

Des autres bases, et particulièrement 2, 8 et 16, vers la base 10

De la base 10 vers la les autres bases, et particulièrement 2, 8 et 16

Des autres bases que 10 entre elles (et particulièrement de 2 vers 16)

Convertisseur en ligne : ici ou là

Conversions vers la base 10 : abordées dans la présentation des bases 2, 8 et 16

Propriété : écriture des entiers en somme des puissances de la base

Un entier de n'importe quelle base peut s'écrire, en base 10, sous la forme d'une addition de puissances de la base :

$(a_n, .., a_1, a_0)_p = a_n*p^n + .. + a_1*p^1 + a_0*p^0 = \Sigma_{i=0}^{n} a_i*p^i$

Exemples

$(4327)_{10} = 4*10^3 + 3*10^2 * 2*10^1 + 7*10^0$

$(1010)_2 = 1*2^3 + 0*2^2 * 1*2^1 + 1*2^0$

$(9A2F)_{16} = 9*16^3 + 10*16^2 * 2*16^1 + 15*16^0$

Conversions de la base 10 vers les bases 2, 8 ou 16

Il existe 2 méthodes qui s'appuient sur la division euclidienne (division entière).

On part d'un nombre dans la base source : nbs dans la base bs, pour arriver à un nombre dans la base cible : nbc dans la base bc.

Division euclidienne

Les entiers naturels sont 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, etc.

Soit A, B, Q et R : 4 entiers naturels.

Effectuer la division euclidienne de A par B non nul c'est déterminer les uniques entiers Q (appelé quotient) et R (appelé reste) tels que :

$$A = B*Q + R \text{ avec } R < B.$$

On peut obtenir les résultats en posant la division ou avec une calculatrice.

Méthode par la puissance

On cherche la plus grande puissance de de la base cible qui est inférieure au nombre de la base source, et on effectue la division euclidienne du nbs par la puissance trouvée.

On recommence en remplaçant le nbs par le reste, jusqu'à ce que la puissance vaille 0 ou que le reste vaille 0.

On écrit alors le nombre par une addition de puissances de la base cible

Les facteurs des puissances sont les chiffres du nombre dans la base cible

➢ ***Exemples***

$(77)_{10}$ en base 2

Nbs	Puissance inférieure	Puissance supérieure	Division euclidienne
77	$2^6 = 64$	$2^7 = 128$	$75 = 2^6 * 1 + 13$
13	$2^3 = 8$	$2^4 = 16$	$11 = 2^3 * 1 + 5$
5	$2^2 = 4$	$2^3 = 8$	$5 = 2^2 * 1 + 1$
1	$2^0 = 2$	$2^1 = 2$	$1 = 1^1 * 1 + 0$

$(77)_{10} = 1*2^6 + 0*2^5 * 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = (1001101)_2$

$(1986)_{10}$ en base 16

Nbs	Puissance inférieure	Puissance supérieure	Division euclidienne
1986	$16^2 = 2^{4*2} = 256$	$16^3 = 2^{4*3} = 4096$	$1986 = 16^2 * 7 + 194$
194	$16^1 = 16$	$16^2 = 2^{4*2} = 256$	$194 = 16^1 * 12 + 2$
2	$16^0 = 1$	$16^1 = 16$	$2 = 16^0 * 2 + 0$

$(1986)_{10} = 7*16^2 + 12*16^1 * 2*16^0 = (7C2)_{16}$

Méthode par la division euclidienne

On effectue la division euclidienne du nombre de la base source, nbs, par la valeur de la base cible, bc. On recommence avec le quotient jusqu'à obtenir 0 comme quotient. Les restes obtenus, en partant du dernier (celui du quotient à 0) au premier, celui du dividende de départ (le nombre de la base source), forme le nombre dans la base cible, de gauche à droite.

➢ ***Exemples***

$(77)_{10}$ en base 2

Dividende	Diviseur	Quotient	Reste
77	2	38	1
38	2	19	0
19	2	9	1
9	2	4	1
4	2	2	0
2	2	1	0
1	2	0	1

$(75)_{10} = (1001101)_2$

$(1986)_{10}$ en base 16

Dividende	Diviseur	Quotient	Reste
1986	16	124	2
124	16	7	12
7	16	0	7

$(1986)_{10} = (7C2)_{16}$

Conversion de la base 2 à la base 16 et inversement

Etant donné que $2^4=16$, les quatre premiers chiffres d'un nombre en base 2 correspondent au premier chiffre du même nombre en base 16, et ainsi de suite.

Idem entre la base 2 et la base 8 : $2^3=8$. Les 3 premiers chiffres d'un nombre en base 2 correspondent au premier chiffre du même nombre en base 8, et ainsi de suite.

On va donc utiliser un tableau de conversion entre les chiffres en base 16 et le nombre correspondant en base 2 :

Base 16	0	1	2	3	4	5	6	7
Base 2	0000	0001	0010	0011	0100	0101	0110	0111

Base 16	8	9	A	B	C	D	E	F
Base 2	1000	1001	1010	1011	1100	1101	1110	1111

- ***Notation des grands nombres en base 2***

  On regroupe les chiffres par paquets de 4 qui correspondent à de l'hexadécimal pour rendre la lecture plus facile en séparant les paquets par un tiret

- ***Exemples de conversion***

  $(7C2)_{16} = (0111\text{-}1100\text{-}0010)_2 = (11111000010)_2$

  $(1001101)_2 = (0100\text{-}1101)_2 = (4D)_{16}$

Utilisations

La base 2 est utilisée dans les ordinateurs

La base 16 permet de représenter l'information dans les ordinateurs.

Les calculatrices fournissent des fonctions qui permettent de passer d'une base à l'autre.

3 - Numération des réels

Propriété : écriture des réels en somme des puissances de la base

Un réel de n'importe quelle base peut s'écrire, en base 10, sous la forme d'une addition de puissances de la base :

$(a_n .. a_0 , b_1 .. b_m)_p = a_n*p^n + .. + a_1*p^1 + a_0*p^0 + b_1/p^1 + .. + b_m/p^m = \Sigma_{i=0}^{n} a_i*p^i + \Sigma_{i=1}^{m} b_i/p^i$

Conversions vers la base 10

$(4,25)_{10} = 4*10^0 + 2/10^1 * 5/10^2 = 4 + 0,2 + 0,05$

$(1.011)_2 = 1*2^0 + 0/2^1 + 1/2^2 + 1/2^4 = 1 + 0 + 0,25 + 0,125 = 1,375$

$(1.A2)_{16} = 1*16^0 + 10/16^1 * 2*16^2 = 1+ 5/8 + 2/256$

Conversions de la base 10 vers la base 2

On multiplie la partie décimale par 2. Et on répète l'opération avec la partie décimale du résultat de la multiplication.

Quand on arrive à 1, on s'arrête. Si on retombe sur une opération déjà faite, on s'arrête.

La suite des chiffres des unités des résultats donne la partie décimale en binaire, avec une série de chiffres répétés à l'infini si on est retombé sur une opération déjà faite pour finir.

- ***Exemples***

$(0,375)_{10} * 2 = 0,750$

$0,750 * 2 = 1,5$

$0,5 * 2 = 1,0$

$(0,375)_{10} = (0,011)_2$

$1/2^2 + 1/2^3 = 1/4 + 1/8 = 0,25 + 0,125 = 0,375$

$(0,3)_{10} * 2 = 0,6$

$0,6 * 2 = 1,2$

$0,2 * 2 = 0,4$

$0,4 * 2 = 0,8$

$0,8 * 2 = 1,6$

$0,6 * 2 = 1,2$

etc.

$(0,3)_{10} = (0,0[1001])_2$

Utilisation

La numération des réels est intéressante en base 2 car elle intervient dans les calculs des ordinateurs.

Opérations élémentaires

Addition, soustraction, multiplication, division des entiers et des réels se font en les posant de la même manière quelle que soit la base.

Mais il faut connaître les tables d'addition et de multiplication dans la base !

Ou alors convertir en base 10, faire l'opération en base 10, puis reconvertir dans la base d'origine.

En base 2, c'est facile !

En base 8 ou 16, c'est plus compliqué !

Extraits de tables d'addition et de multiplication en base 8 et 16 :

$(7)_8+(7)_8 = (16)_8$

$(7)_8*(7)_8 = (61)_8$

$(F)_{16}+(F)_{16} = (1E)_{16}$

$(F)_{16}*(F)_{16} = (E1)_{16}$

Exemples d'opérations en base 2

1+0=1	1+1=10	10+1 = 11	11+1=100	100+1=101	101+1=110	110+1=111
1-0=1	1-1=0	10-1 = 1	11-1=10	100-1=11	101-1=100	110-1=101

$^{1}\,^{1}1\ 1$	1 plus 1 égale 10, je pose 0 et je retiens 1
$+\quad 0\ 1$	0 plus 1 égale 1, plus1 égale 10, je pose 0 et je retiens 1
$=\ 1\ 0\ 0$	1 plus 1 égal 1

$1\ 1^{1}0$	1 ôté de 0 : impossible
$-\quad 1$	1 ôté de 10 égale 1et je retiens 1
$=\ 1\ 0\ 1$	0+1=1 ôté de 1 égale 0
	0 ôté de 1 égale 1

$1\ 1$	$1\ 0\ 1\ 0 \mid 11$	En 101 combien de fois 11 : 1 fois
$\times\ 1\ 1$	$1\ 0\ 0 \mid 11$	1 fois 11, 11, oté de 101 il reste 10
$1\ 1$	$1 \mid$	En 100 combien de fois 11 : 1 fois
$1\ 1\ .$		1 fois 11, 11, oté de 100 il reste 1
$=1\ 0\ 0\ 1$		

4 - Arrondi et précision

Présentation

On peut **arrondir** les réels à une certaine **précision.**

La précision « p » appartient à l'ensemble {… 100 ; 10 ; 1 ; 0,1 ; 0,01 ; .. }, cela quelle que soit la base. « p » c'est 1, ou 10, ou 0,1, etc.

Chercher l'arrondi d'un réel X à une certaine précision « p », c'est :

- D'abord : chercher **les 2 multiples de la précision les plus proches de X** tels que X soit compris entre ces deux multiples, autrement dit, trouver l'entier « n » tel que :

$$n*p \leq X < (n+1)*p$$

- Ensuite : **l'arrondi est alors le multiple le plus proche de X**. S'il existe 2 nombres possibles, on choisit **le plus grand par convention.**

En pratique

1) **Pour avoir la borne inférieure (BI) :** on ramène à 0 tous les chiffres de X de rang inférieur à la précision (par exemple, pour 5324 à une précision de 100, le rang c'est la centaine et la borne inférieure vaut 5300).
2) **Pour avoir la borne supérieure (BS) :** on ajoute la précision à la borne inférieure.
3) Pour trouver la valeur moyenne (VM) entre la BI et la BS, on divise « p » par 2.
4) Si BI + VW > X, l'arrondi vaut BI, sinon l'arrondi vaut BS (autrement dit BS – VM ≤ X).

Exemples en base 10

$N=(432,614)_{10}$

On cherche l'arrondi de $(432,614)_{10}$ à une **précision de 10.**

BI : 430. BS : 430+10=440.

VM : 10/2 = 5.

BI+VM= 430 + 5 = 435.

435 > 432,614, donc l'arrondi vaut 430

On cherche l'arrondi de $(432,614)_{10}$ à une **précision de 1.**

BI : 432. BS : 432+1=433.

VM : 1/2 = 0,5

BI+VM= 432+ 0,5 = 432,5

¬(432,5 > 432,614) , donc l'arrondi vaut 433 (effectivement, BS-VM = 433 – 0,5 = 432,5 ≤ X)

On cherche l'arrondi de $(432,614)_{10}$ à une **précision de 0,1.**

BI : 432,6. BS : 436+0,1=432,7.

VM : 0,1/2 = 0,05

BI+VM= 432,6 + 0,05 = 432,65

432,65 > 432,614, donc l'arrondi vaut 432,6

Exemples en base 2

Rappel de correspondances entre la base 2 et la base 10

Base 2	0	1	10	11	100	101	110	111	1000	1001	1010	1011	Etc.
Base 10	0	1	2	3	4	5	6	7	8	9	10	11	Etc.

N=$(101,10011)_2$

On cherche l'arrondi de $(432,614)_{10}$ à une **précision de 10.**

BI : 430. BS : 430+10=440.

VM : 10/2 = 5.

BI+VM= 430 + 5 = 435.

435 > 432,614, donc l'arrondi vaut 430

On cherche l'arrondi de $(101,10011)_2$ à une **précision de $(10)_2$**.

BI : 100. BS : 100+10=110.

VM : 10/10 = 1 : à noter qu'on divise par 2, mais en base 2, donc par 10

BI+VM= 100 + 1 = 101

¬ (101 > 101,10011), donc l'arrondi vaut 101(en effet, BS-VM = 101 – 1 = 100 ≤ 101,10011)

On cherche l'arrondi de $(101,10011)_2$ à une **précision de $(1)_2$**.

BI : 101. BS : 101+1=110.

VM : 1/10 = 0,1 : à noter qu'on divise par 2, mais en base 2, donc par 10

BI+VM= 101 + 0,1 = 101,1

¬ (101,1 > 101,10011), donc l'arrondi vaut 110 (en effet, BS-VM = 110 – 1 = 101 ≤ 101,10011)

On cherche l'arrondi de $(101,10011)_2$ à une **précision de $(0,1)_2$**.

BI : 101,1. BS : 101,1+0,1=110.

VM : 0,1/10 = 0,01 : à noter qu'on divise par 2, mais en base 2, donc par 10

BI+VM= 101,1 + 0,01 = 101,11

101,11 > 101,10011, donc l'arrondi vaut 101,1

5 - Codage de l'information dans les ordinateurs

Notion de BIT

Les informations traitées par les ordinateurs sont de différentes natures : nombres, texte, images, sons, vidéos, programmes, etc.

Dans un ordinateur, elles sont toujours représentées sous forme binaire : une suite de 0 et de 1.

On appelle chaque information élémentaire (un 0 ou un 1) BIT (Binary digIT = chiffre binaire).

Représentation externe et représentation binaire

Le codage permet de passer d'une représentation dite externe (la représentation compréhensible directement par un humain : nombres, texte, images, sons, vidéos) à représentation interne sous forme de BIT. On parle alors de représentation binaire.

Représentation binaire et représentation physique

La représentation binaire est une représentation logique : elle tient encore du discours.

Cette représentation logique se traduit physiquement par des états de la matière.

Il existe plusieurs solutions de représentation physique de la représentation binaire :

Pour le stockage

La charge électrique (RAM : Condensateur-transistor, sans électricité on perd la mémoire !) : chargé (bit 1) ou non chargé (bit 0)

La magnétisation (Disque dur, disquette) : polarisation Nord (bit 1) ou Sud (bit 0)

Les alvéoles (CDROM) : réflexion (bit 1) ou pas de réflexion (bit 0)

Pour la circulation

Les câbles électriques : niveau haut ou bas de tension, présence ou absence de lumière. Le débit d'information est donc le nombre de 0 et de 1 que l'on arrive à caser par seconde.

La numérisation

C'est la conversion des informations d'un support (texte, image, audio, vidéo) ou d'un signal électrique en données numériques que des dispositifs informatiques ou d'électronique numérique pourront traiter.

Notion d'octet

Présentation

Dans les ordinateurs, on travaille rarement directement au niveau du BIT. On travaille en général au niveau de l'octet.

Un octet c'est 8 BIT à la suite.

Un octet permet donc de coder 256 possibilités : 2^8

L'octet servira à coder les caractères alphanumériques et spéciaux (a, b, 1, 2, ;, -, +, etc.)

Notion de bit de poids fort ou faible

Dans un octet, le premier bit, dans une lecture de gauche à droite, qui représente 2^7, est appelé bit de poids fort.

Le dernier bit, qui représente 2^0, est appelé bit de poids faible.

ko, Mo, Go, To, etc.

un kilooctet = 1ko = 1000 octets

un megaoctet = 1Mo = 10^6 octets = 1 mega

un gigaoctet = 1Go = 10^9 octets = 1 giga

un teraoctet = 1To = 10^{12} octets = 1 tera = 1000 giga

un petaoctet = 1Po = 10^{15} octets

Attention : en 1998, l'IEC, organisme de standardisation, a fixé que un ko = 1000 octets et pas 1024 ou 2^{10} octets ! idem pour les suivants : un Mo = 10^6 octets et pas 2^{20}, un Go = 10^9 octets et pas 2^{30} octets.

Représentation d'un octet en hexadécimal

Un octet est souvent représenté sous la forme de 2 blocs de 4 bits, soit 2 blocs de 2^4=16 possibilités.

Le contenu d'un octet est donc représenté par deux entiers en base 16 : 4F, E6 par exemple.

4F, c'est 0100 et 1111 soit 1+2+4+8+64 = 79

E6, c'est 1110 et 0110 soit 2+4+32+64+128 = 230

Convertisseur en ligne : ici ou là

6 - Codage des entiers dans les ordinateurs

Les entiers se codent sur un ou plusieurs octets.

Si on code des entiers négatifs (entiers relatifs, Z), on utilise le bit de poids fort pour le signe.

Il existe 3 solutions de codages des entiers relatifs : le code binaire signé, le complément à 1 et le complément à 2.

Code binaire signé

Principes sur un octet

Le bit de poids fort sert pour le signe (0 pour +, 1 pour -), les 7 autres bits servent pour la valeur absolue du nombre.

Sur un octet, on peut coder de -127 $(1111\ 1111)_{bs}$ à +127 $(0111\ 1111)_{bs}$

Autrement dit de -2^7-1 à $+2^7-1$

Avec ce système, on a 2 codages pour 0 : $(0000\ 0000)_{bs}$ et $(1000\ 0000)_{bs}$

Les opérations sont un peu compliquées : il faut distinguer entre le signe et le nombre et addition et soustraction diffèrent selon le signe (additionner un nombre négatif à un nombre positif revient à le soustraire).

Généralisation à n octets

Même principe : on peut coder de $-2^{8*n-1}-1$ à $+2^{8*n-1}-1$

- ***Soit sur 2 octets***

  de $-2^{8*2-1}-1$ à $+2^{8*2-1}-1$ soit de $-2^{15}-1$ à $+2^{15}-1$ de -32767 à +32767

  En code binaire signé, de $(1111\ 1111\ 1111\ 1111)_{bs}$ à de $(0111\ 1111\ 1111\ 1111)_{bs}$

Complément à 1

Exemple sur un octet

1. Pour les nombres positifs, on utilise le code binaire signé

   Donc $(0xxx\ xxxx)_{bs} = (0xxx\ xxxx)_{ca1}$ avec $x \in \{0, 1\}$

   - ***Par exemple***

     $(1)_{10} = (0000\ 0001)_{bs} = (0000\ 0001)_{ca1}$

2. Pour les nombres négatifs, de -1 à -127, on inverse les tous les bits du nombre positif correspondant.

   - ***Par exemple :***

     $(1)_{10} = (0000\ 0001)_{ca1}$. Donc $(-1)_{10} = (1111\ 1110)_{ca1}$

     $(127)_{10} = (0111\ 1111)_{ca1}$. Donc $(-127)_{10}\ (1000\ 0000)_{ca1}$

3. Avec ce système, on a 2 codages pour 0 : $(0000\ 0000)_{ca1}$ et $(1111\ 1111)_{ca1}$

Avantage : les opérations sont simplifiées

- ***Les additions sont simplifiées***

  On peut additionner tous les bits sans se soucier du signe.

- ***Par exemple :***

  $(1)_{10}+(-1)_{10} = (0000\ 0001)_{ca1}+(1111\ 1110)_{ca1} = (1111\ 1111)_{ca1} = (0)_{10}$

  $(69)_{10} = (0100\ 0101)_{ca1}$ Donc $(-69)_{10} = (1011\ 1010)_{ca1}$

  $(33)_{10} + (-69)_{10} = (0010\ 0001)_{ca1}+(1011\ 1010)_{ca1} = (1101\ 1011)_{ca1} = (0010\ 0100)_{bs} = (32+4)_{10} = (36)_{10}$

- ***La soustraction c'est l'addition de l'opposé.***

  L'opposé d'un nombre, c'est l'inversion de tous les bits.

  La soustraction c'est l'addition de l'opposé : 5-3 ⇔ 5+(-3)

Complément à 2 : c'est la représentation dans les ordinateurs

Exemple sur un octet

- ***Pour les nombres positifs, on utilise le code binaire signé donc le complément à 1.***

  Donc $(0xxx\ xxxx)_{bs} = (0xxx\ xxxx)_{ca1} = (0xxx\ xxxx)_{ca2}$ avec $x \in \{0, 1\}$

- ***Pour les nombres négatifs, on ajoute 1 au complément à 1 :***

  $(1)_{10} = (0000\ 0001)_{ca1}$ donc $(-1)_{10} = (1111\ 1110)_{ca1} = (1111\ 1111)_{ca2}$: 1111 1110 + 1

  $(127)_{10} = (0111\ 1111)_{ca1}$ donc $(-127)_{10} = (1000\ 0000)_{ca1} = (1000\ 0001)_{ca2}$: 1000 0000+ 1

- ***Avec ce système, on a 1 seul codage pour 0 : $(0000\ 0000)_{ca2}$***

  On a vu que $(1111\ 1111)_{ca2}$ vaut -1.

- ***On peut coder de -127 à +127 mais il nous reste une valeur de codage. Est-ce -128 ou +128 ?***

  On peut constater que :

  $(128)_{10} = (127)_{10} + (1)_{10} = (0111\ 1111)_{ca2} + (0000\ 0001)_{ca2} = (1000\ 0000)_{ca2}$

  $(-128)_{10} = (-127)_{10} + (-1)_{10} = (1000\ 0001)_{ca2} + (1111\ 1111)_{ca2} = (1000\ 0000)_{ca2}$

  Les deux opérations fonctionnent.

  Mais, pour les nombres positifs, on utilise le code binaire signé, dont le bit de poids fort vaut 0.

  Donc on va donc coder en complément à 2 de -128 à +127.

Avantage : les opérations sont simplifiées et il n'y a qu'un seul 0

Addition et soustraction sont simplifiées : elles sont indépendantes du signe.

Par exemple : 127 – 127 c'est $(0111\ 1111)_2 + (1000\ 0001)_{ca2} = (0000\ 0000)_{ca2}$

Le complément à 2 pour les entiers relatifs, c'est le mode de représentation utilisé dans les ordinateurs et les langages informatiques.

Conclusion : représentation physique, représentation logique et optimisation

On découvre avec le complément à 2 que la représentation logique de bas niveau est la plus éloignée de la représentation externe, c'est-à-dire en base 10.

C'est une représentation en base 2, avec un bit de signe et un complément à 2.

Cette représentation s'éloigne des mathématiques pour intégrer une problématique d'optimisation. Le but est de simplifier les opérations pour aller plus vite.

Cette représentation c'est celle qui est adaptée aux contraintes physique des ordinateurs pour optimiser les traitements.

7 - Codage des nombres réels dans les ordinateurs

Les réels se codent sur au moins 2 octets.

Il existe 2 solutions de codages des réels : le code à virgule fixe et un code à virgule flottante.

Code à virgule fixe

Le code à virgule fixe traduit les nombre écrit sous la forme : $\pm(a_n .. a_0 , b_1 .. b_m)_p$

- 1 bit de signe, n bits de partie entière, m bits de partie fractionnaire.
- Le bit de signe et les n bits de partie entières sont codés comme des entiers relatifs en complément à 2.
- Les m bits de partie fractionnaire sont codés comme des entiers positifs en simple code binaire.

La précision dépend de m et donc du nombre de bit dédié à la partie fractionnaire.

A l'usage, il faut connaître à l'avance le niveau de précision dont on aura besoin dans les programmes pour positionner au mieux la virgule. Comme en général, on ne peut pas déterminer le niveau de précision nécessaire, on préfère utiliser le code à virgule flottante.

Cette connaissance a priori existe pour certaines applications, mais dans le cas général ce n'est pas le cas. Pour pallier à ce manque de flexibilité, le concept de virgule flottante a été introduit.

Code à virgule flottante

Le code à virgule flottante traduit les nombre écrit sous la forme : $\pm\ m\ .\ b^{e}_{(b)}$

- $\pm$ est le signe
- m est la mantisse. C'est un réel positif. $m \in [1 ; 10 [$. m est codé en virgule fixe. 10 est la valeur maximum, quelle que soit la base.
- e est l'exposant. C'est un entier relatif codé en complément à 2.
- b est une base b : 2, 8, 10, 16,… Dans les ordinateurs, c'est 2.

Remarque : la puissance de l'exposant correspond à la base. On peut donc se passer d'écrire la base.

Exemples

Base 10 : $5{,}425x10^{2}$. Le nombre vaut : 542, 25 : en virgule fixe on a 2 chiffres après la virgule.

Base 2 : $1{,}01x2^{-1}$. Le nombre vaut 0,101 : en virgule fixe on a 3 chiffres après la virgule.

Base 16 : $A{,}0B4x16^{-2}$. Le nombre vaut 0,0A0B4 : en virgule fixe on a 5 chiffres après la virgule.

En virgule fixe : $0{,}25_{(10)} = 0{,}01_{(2)}$

En virgule flottante : $2{,}5x10^{-1} = 1{,}0x2^{-2}$

Représentation dans les ordinateurs

Répartition des bits

Signe	Exposant biaisé	Mantisse normalisée
1 bit	p bits	q bits

Notion de mantisse normalisé

La mantisse est un réel positif. En base 2, $m \in [1 ; 2[$

Donc, la mantisse est de la forme : 1, …

Donc, la forme du code à virgule flottante devient : $\pm 1,M \cdot 2^E$

Le premier 1 devient une valeur par défaut et M est la mantisse normalisée.

M est donc un entier positif : son codage est donc simplifiée : on n'a plus besoin de binaire signé ni ne complément à 1 ou à 2.

Notion d'exposant biaisé

- ***Présentation***

Dans la forme : $\pm m \cdot b^e_{(b)}$, « e » est l'exposant (petit e). C'est un entier relatif.

Dans la forme : $\pm 1,M \cdot 2^E$, « E » (grand e) est l'exposant biaisé.

Le biais « b » est tel que $e + b = E$ avec $E \geq 0$.

- ***Précisions – première approche***

On peut considérer en première approche que « e » est codé en binaire signé.

Donc si « e » est codé sur p bits :

$e \in [-2^{p-1}-1 ; 2^{p-1}-1]$.

Par exemple, sur 8 bits (standard float), $e \in [-2^7-1 ; 2^7-1]$ soit [-127 ; 127]

Pour passer de « e » à « E », on ajoute $2^{p-1}-1$ à « e » (la valeur absolue de la borne inférieure).

Biais = $2^{p-1}-1$

$E \in [0; +2^p-2]$.

Par exemple, sur 8 bits, le biais vaut 127 et $E \in [0 ; +254]$

- ***Exemple 1 : l'exposant « e » est codé sur 4 bit.***

Quel est l'ensemble de définition de e, de E et combien vaut le biais?

$e \in [-2^{4-1}-1 ; 2^{4-1}-1]$ c'est-à-dire [-7 ; +7]

Le biais vaut 7

$E \in [0 ; +14]$.

- ***Exemple 2 : On a $(e)_{10} = (-3)_{10}$. Combien vaut $(E)_{vf}$ avec un exposant codé sur 4 bits ?***

Le biais vaut $(7)_{10}$.

Donc $E = (-3)_{10} + (7)_{10} = (4)_{10} = (0100)_{vf}$

Précisions – 2ème approche

- ***infini et NaN***

  Si E ∈ [0 ; +254] sur 8 bits, il reste une valeur possible : 255 : que des 1 sur tous les octets de l'exposant de façon générale.

  Ce cas permet de coder deux cas particuliers :

  - L'infini : dans ce cas la mantisse vaut 0
  - NaN (not a number, c'est-à-dire les erreurs) : dans ce cas la mantisse est différente de 0

- ***Cas du 0***

  Avec une mantisse normalisée, on a un 1 par défaut. On ne peut donc pas coder le 0 avec la mantisse.

  On va donc utiliser une valeur spéciale de E pour coder le 0 : ce sera 0.

- ***Tous les cas***

  E ∈ [1 ; +254] : permet de coder les nombres

  E = 255 : permet de coder l'infini (mantisse=0) et Nan (mantisse !=0)

  E = 0 : permet de coder le 0

Exemple

Que vaut en base 10 le codage suivant : 1 0 1 0 0 1 0 1 avec 3 bits d'exposant biaisé ?

- ***signe***

  1 : négatif

- ***exposant***

  nombre de bits = p = 3

  $E = (010)_{vf} = (2)_{10}$

  $Biais = 2^{p-1}-1 = (3)_{10}$

  $e + b = E$ donc $e = (-1)_{10}$

- ***mantisse***

  Mantisse normalisée = 0101 donc mantisse = $(1,0101)_2$

- ***nombre en base 2***

  Nombre en base 2 = $-1,0101 \times 2^{-1}$

  Sachant que $(a_n .. a_0 , b_1 .. b_m)_p = a_n*p^n + .. + a_1*p^1 + a_0*p^0 + b_1/p^1 + .. + b_m/p^m = \Sigma_{i=0}^{n} a_i*p^i + \Sigma_{i=1}^{m} b_i/p^i$

  $(-1,0101)_2 \times 2^{-1} = -0,10101_{(2)}$

  $= -(1/2^1 + 1/2^3 + 1/2^5)_{10} = -(1/2+1/8+1/32)_{10} = -0,65625_{(10)}$

Standard IEEE 754 (1985) – float et double

Simple précision : sur 32 bits (4 octets)

Signe	Exposant biaisé	Mantisse normalisée
1 bit	8 bits biais=2^7-1=127 e ∈ [-126 ; +127] E ∈ [1 ; +254]	23 bits M ∈ [0 ; 2^{23}] mantisse : 1 bit de plus m ∈ [1 ; 2^{24}]

2^{24} = 16 777 216 : on a donc 8 chiffres significatifs en base 10

2^{-126} = 1.1754944 e^{-38}

2^{+127} = 1.7014118 e^{+38}

Type C ou java : float

Double précision : sur 64 bits (8 octets)

Signe	Exposant biaisé	Mantisse normalisée
1 bit	11 bits (biais=2^{10}-1=1023)	52 bits
1 bit	11 bits biais=2^{10}-1=1023 e ∈ [-1022 ; +1023] E ∈ [1 ; +2046]	52 bits M ∈ [0 ; 2^{52}] mantisse : 1 bit de plus m ∈ [1 ; 2^{53}]

2^{53} = 9.0071993 e^{+15}: 0n a donc 16 chiffres significatifs en base 10

2^{-1022} = 2.225074 e^{-308}

2^{+1023} = 8.988466 e^{+307}

Type C ou java : double

Valeurs particulières

Exposant	**Mantisse**	**Valeur**
0	0	0
Que des 1	0	+ou- infini
Que des 1	Tout sauf 0	NaN : not a number : error !

Voir ici ou là et encore ici

Exemple

Ecrire $35,5_{(10)}$ en float.

$35_{(10)} = 32+2+1 = 2^5+2^1+2^0 = 10\ 0011_{(2)}$

$0,5_{(10)} = 0,1_{(2)}$ (Méthode : 0,5 * 2 = 1)

$35,5_{(10)} = 10\ 0011,1_{(2)} = 1,000\ 111*2^5{}_{(2)}$

Mantisse = 1,000 111. Mantisse normalisée = 000111 puis des 0 pour remplir les 23 bits.

Exposant = $5_{(10)}$. Biais = 127. Exposant biaisé : 5 + 127 = 132 = 128 + 4 = $(1000\ 0100)_2$

Signe	Exposant biaisé	Mantisse normalisée	Valeurs
1 bit	8 bits (biais=2^7-1=127)	23 bits	
0	5+127=132=128+4=1000 0100 $_{(2)}$	00011100000000000000000	1,000 111*$2^5{}_{(2)}$

$35,5_{(10)}$ s'écrit donc en float sur 4 octets :
0100 0010 – 0000 1110 – 0000 0000 – 0000 0000

8 - Codage des caractères – ASCII – ISO 8859 – UNICODE – UTF8

Présentation

Il s'agit de coder les caractères alphanumériques : les chiffres (0,1,...,9), les lettres minuscules et majuscules (a,...z, A,...,Z), les signes de ponctuation (. , ; / = * - …), des symboles usuels (%, @, $, &, etc.), des caractères spéciaux (passage à la ligne, delete, bip sonore, etc.)

Cet ensemble de caractères contient : 10 chiffres, 52 lettres, une 20aine de signe de ponctuation ou de symboles usuels, ce qui fait déjà plus de 80 caractères.

Avec 6 bits, on peut coder $2^6 = 64$ caractères : c'est insuffisant. Avec 7 bits, on peut coder $2^7 = 128$ caractères. C'est suffisant.

Il existe finalement trois codes : le code ASCII, ASCII étendu ou ISO et l'UNICODE.

- Le code **ASCII** codifie 128 caractères sur 7 bits.
- Les codes **ASCII étendus ou codes ISO 8859** permettent de codifier 256 caractères sur 8 bits. Il en existe de nombreux, ce qui est source de problèmes avec les accents particulièrement.
- L'**UNICODE** permet de normaliser tous les caractères possibles, actuellement 128 172. Il est indépendant de la représentation physique (le nombre d'octets).

Le code ASCII

Présentation

Le code ASCII est un code standard qui définit la valeur du jeu courant de caractères sur 7 bits.

Sur un octet, il restera le bit de poids fort.

Les caractères sont donc codés de 0 à 127 :

- 0 à 31 : des caractères spéciaux
- 32 à 47 : des symboles et de la ponctuation : (espace, ! , « , #, etc.)
- 48 à 57 : les chiffres dans l'ordre : (0, … , 9)
- 58 à 64 : des symboles et de la ponctuation : (: , ; , < , etc.)
- 65 à 90 : les alphabets majuscules (A,...,Z)
- 97 à 122 : les alphabets minuscule (a,...z)
- 123 à 127 : des symboles et de la ponctuation : ([, \ , etc.)

Avantages

Il permet aux machines de communiquer entre elles, quels que soient les processeurs et quels que soit les système d'exploitation.

Défaut

American Standard Code for Information Interchange" signifie en français "Code américain normalisé pour l'échange d'information".

C'est un code basé sur l'alphabet américain : alphabet latin sans accents.

Table ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Les codes ASCII étendus : codes ISO 8859

Présentation

L'objectif est d'élargir le code ASCII pour permettre par exemple l'introduction d'accents.

Le 8ème bit de l'octet du caractère va permettre cela. C'est ce qu'on appelle le code ASCII étendu. C'est le code ISO 8859 qui normalise ça.

ISO signifie International Organization for Standardization (le sigle ne correspond pas !)

Comme il y a beaucoup de système d'accentuation dans les langues latines, et qu'il y a aussi des langues non latines, le code ISO 8859 se décline en :

ISO 8859-1 (ou latin-1 ou européen occidental) : pour la plupart des langues européennes.

ISO 8859-2 (ou latin-2 ou européen central) : pour le le croate, le tchec, le hongrois, etc.

…

ISO 8859-6 : pour l'arabe

…

ISO 8859-15 : révision du 8859-1

…

https://fr.wikipedia.org/wiki/ISO/CEI_8859

Avantage

Intègre le standard ASCII.

Permet de standardiser les accents.

Défaut

Il y a plusieurs standards ! Et des révisions dans les standards.

L'UNICODE et UTF8

UNICODE

http://www.tuteurs.ens.fr/faq/utf8.html#unicode

Le Standard Unicode est produit par une organisation à but non lucratif (le Consortium Unicode) ayant pour objectif d'attribuer un numéro à tout caractère utilisé dans une langue humaine.

Un caractère Unicode est un caractère défini dans le Standard Unicode. On y fait référence par son numéro écrit en hexadécimal précédé de «U+». Par exemple, la lettre latine « a » correspond à U+0061.

Unicode couvre actuellement 128 172 caractères.

http://www.unicode.org/charts/

UTF 8

L'UTF8 est le système d'encodage qui transforme une code Unicode en octets : de 1 à 4 octets.

L'UTF8 est compatible avec les caractères ASCII. Particulièrement, les caractères de 32 à 127 seront codés sur 1 octet.

Avantage

Un seul encodage pour toutes les langues : c'est LE système universel.

Inconvénients

La taille des caractères n'est pas fixe, ce qui complique certains calculs.

Usages

Généralisé ! Dans les sites web, sur les terminaux d'ordinateur, etc.

- ***Aparté technique :***

L'encodage des terminaux windows fonctionne avec des « page de code ». La page de code n°850 est celle d'Europe occidentale. On peut voir le n° avec la commande chcp.

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

Sous linux, la commande xxd permet de voir l'encodage d'un texte : xxd-entrée-texte-entrée-ctrl-d. Sont affichés les caractères du texte en hexadécimal avec un 0a à la fin pour le line feed. Si les caractères accentués sont sur 2 octets, c'est que l'encodage est en UTF8.