Représentation des données:

Les nombres.

Question: quelle sont les représentations numériques de nos nombres dans leurs diversités?

Réponse: chaque ***type*** a son propre format.

Une remarque importante s'impose avant d'aller plus loin:

toute représentation doit être de nature à faciliter les traitements qu'on opère sur cette représentation. Si plusieurs façons de représenter existent, on ne retient que celle qui est appropriée.

Les types communs sont: *les entiers naturels, les entiers relatifs, et les réels*.

1) Représentation des **entiers naturels**:

la plus simple des représentations est celle qui consiste à écrire le nombre en base 2. elle s'offre parfaitement au calcul et à l'addition en particulier. Nous appellerons cette méthode, pour la distinguer des autres, la représentation en "***binaire pur***".

Inversement si une configuration binaire représente un entier naturel, alors cet entier est celui qu'on trouve en procédant à une conversion binaire – décimale.

Représentation des données:

Les nombres (suite).

Sur n bits on ne peut représenter, en ***binaire pur***, que les entiers naturels de l'intervalle $[0, 2^n - 1]$.

Exemples:

n	8	9	10	15	16	32
Intervalle	[0 , 255]	[0 , 511]	[0 , 1023]	[0 , 32767]	[0 , 65535]	[0, 4294967295]

Par ailleurs, si seulement n bits suffisent pour noter, en binaire pur, la valeur (donc la représentation) d'un entier naturel alors que nous disposons d'un espace sur m bits (m>n), les bits non significatifs sont mis à 0. la notion de "vide" n'existe pas dans le monde numérique.

Nous verrons que cette règle change pour les nombres relatifs.

L'addition en base binaire est similaire à l'addition en base dix et se fait conformément à la table donnée dans la diapositive suivante.

Il y va de même pour la multiplication.

Représentation des données:

L'arithmétique binaire.

Opérations arithmétiques en binaire (base 2):

Les opérations sur les entiers s'appuient sur les tables :

Addition

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	**1** 0

retenue

Multiplication

a	b	a*b
0	0	0
0	1	0
1	0	0
1	1	1

Représentation des données:

Les nombres (suite).

Exemples d'addition, en binaire pur, sur 8 bits.

	14	00001110		15	00001111		252	11111100
+			+			+		
	65	01000001		119	01110111		15	00001111
=	79	**0** 01001111	=	134	**0** 10000110	=	267	**1** 00001011

La retenue finale (ici en rouge) indique, **aussi**, le caractère correct ou pas du résultat sur 8 bits. C'est une forme de débordement (overflow).

Exemple de multiplication: 1010 $(10)_{10}$

* 1101 $*(13)_{10}$

1010

\+ 0000

\+ 1010

\+ 1010

= 10000010 $= (130)_{10}$

Représentation des données:

Les nombres (suite).

2) Représentation des **entiers relatifs ou signés**:

2.1) Représentation dite en *signe et valeur absolue* (sign and magnitude).

Sur n bits, 1 bit est réservé au signe, les autres bits, à la valeur absolue.

Exemple sur 8 bits, signe à gauche, (+) → 0, (-) → 1.

+15 → 00001111 **+32 → 00100000**

-15 → 10001111 **-32 → 10100000**

- ✓ Intervalle $[-2^{n-1}, +2^{n-1}]$
- ✓ Deux représentations de zéro **00000000** et **10000000** sur 8 bits.
- ✓ Le test par rapport à zéro est une opération courante.
- ✓ Arithmétique complexe: pour effectuer Z =X + Y ,

on regarde les signes de X et Y.

Si ils sont de même signe, Z garde ce même signe et les valeurs absolues s'ajoutent.

S'ils sont de signes opposés, il faut examiner le rapport entre les valeurs absolues etc. Bref, ce n'est pas simple.

Représentation des données:

Les nombres (suite).

2.2) La représentations en *notation complément à deux* ou *complément vrai*.

La compréhension de cette façon de représenter les nombres signés est relativement délicate en première approche. Commençons d'abord par l'illustrer en base dix.

Soit X et Y deux nombres décimaux codés, chacun, sur deux chiffres décimaux.

Les valeurs permises sont donc (00) (01) (02) (03) **…** (97) (98) (99).

Toutes les sommes 01+99, 02+98, 03+97 et d'une manière générale **(**X + complément à 10^2 de X**)**, donnent 10^2.

Cependant, sur deux chiffres 10^2 est équivalent à 00. cette addition est également appelée addition modulo 10^2. Ou bien addition sans tenir compte de la retenue finale.

Si on considère que 01 + 99 = 00, alors l'un est l'opposé de l'autre. C'est comme si l'un est positif , l'autre négatif. Pour simplifier, disons que 01 est positif et que 99 représente (-1). L'inverse est également valide. Dans cette logique 98 représente (-2), 97 représente (-3) . . . 53 représente (-47), 52 représente (-48), 51 représente (-49) et 50 représente en fait (-50).

Représentation des données:

Les nombres (suite).

En résumé tous les nombres de 00 à 49 représentent eux même et les nombres de 50 à 99 représentent des nombres négatifs (-x) ou x est le complément à 10^2 de cette configuration décimale. Intervalle des nombres qu'on peut représenter: [-50, +49].

X - Y = X + (-Y) = X + (Complément à 10^2 de Y). La soustraction se transforme en addition.

Prenons quelques exemples pour clôturer l'illustration:

Effectuons (-1) + (-1). (-1) est représenté par 99.

On fait 99 + 99 (modulo 10^2) = 98. 98 représente un nombre négatif (-x) où x est le complément à 10^2 de 98 donc x=2. 98 est l'écriture de (-2).

Autre exemple: (+20) + (-32). (+20) s'écrit 20 , -32 s'écrit 68. 20 + 68 = 88.

88 est un nombre négatif parce qu'il est dans l'intervalle [50, 99]. Il représente

(-x) avec x =complément de 88 vis-à-vis de 100. c'est 12. le résultat est donc (-12).

Deux cas de figure "bizarres":

(40) + (15) = (55). La somme de deux nombres positifs génère un résultat négatif!

(60) + (60) = (20). La somme de deux nombres négatifs engendre un résultat positif!

Cette dernière somme voulait en fait dire (-40) + (-40) qui devait donner (-80).

Ces deux cas bizarres sont deux cas de dépassement de capacité (overflow) qu'il faut ***détecter*** et ***signaler***.

Représentation des données:

Les nombres (suite).

En base deux maintenant.

a) ***Comment transformer une soustraction en une addition.***

Rappel:

2^n s'écrit 100…000, (1 suivi de n zéros).

$2^n - 1$ s'écrit 111…111 (concaténation de n un).

X et Y sont deux nombres binaires codés, chacun, sur n bits.

$X - Y = X - Y + 2^n$ modulo 2^n. Cette identité peut également s'énoncer:

$X - Y = X - Y + 2^n$. À condition d'ignorer la retenue. (retenue finale, cela s'entend).

$X - Y = X - Y + 2^n - 1 + 1$. À condition d'ignorer la retenue.

$X - Y = X + (2^n - 1 - Y) + 1$. À condition d'ignorer la retenue.

$(2^n - 1 - Y)$ s'écrit à partir de Y en inversant les bits.

$$\begin{array}{rl} & 1\ 1\ 1 \ \ldots \ 1\ 1\ 1 \\ - & y\ y\ \mathbf{0} \ \ldots \ \mathbf{1}\ y\ y \\ = & y'y'\mathbf{1} \ \ldots \ \mathbf{0}\ y'y' \end{array}$$

y′ signifie complément de **y**.

Terminologie: $y'y'\ y'$ **…** $y'y'y'$**.** Est appelé le **complément à 1 de Y**. On le note ici $\mathbf{Y_1}$.

$X - Y$ devient $X + Y_1 + 1$. À condition d'ignorer la retenue.

Terminologie: $Y_1 + 1$ est appelé **complément à deux de Y**. on le note ici $\mathbf{Y_2}$.

$X - Y = X + \mathbf{Y_2}$. À condition d'ignorer la retenue. La soustraction se transforme en addition.

Représentation des données:

Les nombres (suite).

b) ***Qui est positif et qui est négatif?*** (dans ce qui suit on ne va plus réitérer la condition sur la retenue. Cependant elle est toujours là, sous entendue.)

Du résultat $X - Y = X + \mathbf{Y_2}$, on peut tirer que: $X - X = 0 = X + X_2$.

De $X + X_2 = 0$, on déduit que si $X > 0$ alors $X_2 < 0$,

si $X < 0$ alors $X_2 > 0$.

Sur n bits on peut écrire toutes les configurations binaires suivantes:

(**0**00 … 000), (**0**00 … 001), (**0**00 … 010), (**0**00 … 011), **…** ,(**0**111 … 110), (**0**11 … 111)

Et

(**1**00 … 000), (**1**00 … 001), (**1**00 … 010), (**1**00 … 011), **…** ,(**1**111 … 110), (**1**11 … 111)

Si on **convient** que la première série de configurations représente des nombres entiers positifs pour lesquels on sait retrouver leurs équivalents en base dix, alors on peut trouver leurs opposés comme étant des nombres négatifs.

Représentation des données:

Les nombres (suite).

X binaire (X>0)	X décimal	– X binaire (-X<0)	– X Décimal
00000001	1	**1**1111111	–1
00000010	2	**1**1111110	–2
00000011	3	**1**111 1101	–3
00000100	4	**1**1111100	–4
…	…	…	…
01111110	126	**1**0000010	-126
01111111	127	**1**0000001	-127
n'existe pas	n'existe pas	**1**0000000	-128
00000000	0	**0**0000000	0

Si on maintient la convention suggérée à l'instant, selon laquelle les nombres de la première colonne de ce tableau représentent des nombres positifs, alors les nombres de la troisième colonne représentent des nombres négatifs. **Conclusion** le bit de poids fort indique le signe (0:positif et 1:Négatif). Cependant il fait partie, intégralement, de la valeur.

Représentation des données:

Les nombres (suite).

Voici maintenant une liste de questions qu'on peut se poser dans le contexte des nombres signés représentés dans la notation complément à deux.

a) Recherche de l'intervalle des nombres signés qu'on peut représenter sur n bits en notation complément à deux:

Le plus grand nombre positif est **0**1111… 111 c'est $2^{(n-1)}$-1.

Le nombre négatif le plus petit est **1**0000 … 000, il représente (-x) où cette configuration est le complément à deux de x. Le complément à deux de cette configuration est elle-même, c'est-à-dire que x = $2^{(n-1)}$. Donc le nombre négatif le plus petit est - $2^{(n-1)}$

L'intervalle recherché est [- $2^{(n-1)}$, + $2^{(n-1)}$ -1]. Dans le cas du tableau précédent, c'est [-$2^{(7)}$, +$2^{(7)}$-1] soit [-128, 127]. Tout autre nombre signé en dehors de cet intervalle ne tient pas sur 8 bits. Et si toutefois vous essayez de le caser, comme même, en le tronquant par exemple, sa valeur sera erronée.

L'arithmétique sur les nombre signés dans la notation complément à deux est ordinaire, à l'exception de la détection et du signalement du débordement.

Représentation des données:

Les nombres (suite).

b) Comment convertir des entiers décimaux signés (relatifs) en leurs équivalents binaires sur n bits.

Réponse: Il faut d'abord s'assurer qu'il est possible de les représenter, c'est-à-dire qu'ils appartiennent à l'intervalle vu précédemment.

Si le nombre est positif, il est codé comme on code un entier naturel en binaire pur.

S'il est négatif, c'est le complément à deux de sa valeur absolue qui le représente.

Exemple n=10 bits. Donc l'intervalle est [-512, +511].

Codez +254, +514, -1, -192, -520.

$(+254)_{10}$ → (**0**$0111111110)_2$. $(+514)_{10}$ impossible.

$(-1)_{10}$ → (**1**$111111111)_2$. $(-192)_{10}$ → (**1**$101000000)_2$. $(-520)_{10}$ impossible.

Représentation des données:

Les nombres (suite).

c) Quel est le nombre décimal signé représenté, dans la notation en complément à deux, par une certaine configuration binaire?

Réponse: On examine le bit de poids fort. S'il est à 0, il s'agit d'un nombre positif. Son équivalent décimal est la valeur obtenue en faisant la conversion du binaire pur vers le décimal.

Si le bit de poids fort est à 1, il s'agit alors d'un nombre négatif.

Par définition, cette configuration représente (-X, avec X>0) et c'est le X_2, de telle sorte que: $X + (X_2) = 0$.

Donc $X = -(X_2)$.

Or $-(X_2) = (X_2)_2$. Au final $\mathbf{X = (X_2)_2}$.

Exemples: que représentent $(00011111)_2$ et $(10000011)_2$.

$(00011111)_2 = (+31)$.

Complément à deux de $(10000011) = (01111101)_2 = (125)_{10}$.

Donc le deuxième configuration représente (-125).

Représentation des données:

Les nombres (suite).

d) Addition et débordement.

Soit à effectuer les opérations suivantes sur **8 bits**:

$(+33)_{10} + (+16)_{10}$. $(-33)_{10} + (-16)_{10}$. $(+27)_{10} + (-28)_{10}$. $(-16)_{10} + (+20)_{10}$.

$(+65)_{10} + (+70)_{10}$. $(-96)_{10} + (-49)_{10}$

$(+33)_{10}$	00100001	$(-33)_{10}$	11011111	$(+27)_{10}$	00011011		
$+ (+16)_{10}$	+ 00010000	$+(-16)_{10}$	+ 11110000	$+ (-28)_{10}$	+ 11100100		
$= (+49)_{10}$	= 00110001	$=(-49)_{10}$	= 1 11001111	$= (-1)_{10}$	= 11111111		

$(-16)_{10}$	11110000	$(+65)_{10}$	01000001	$(-96)_{10}$	10100000		
$+ (+20)_{10}$	+ 00010100	$+(+70)_{10}$	+ 01000110	$+(-49)_{10}$	+ 11001111		
$= (+4)_{10}$	= 1 00000100	$=(135)_{10}$	= 10000111	$=(-145)_{10}$	= 1 01101111		

Les deux dernières sommes ont généré un débordement (overflow). Il se produit un débordement lorsque les opérandes sont de même signe et le résultat est de signe opposé.

Représentation des données:

Les nombres réels.

Les formats examinés jusqu'à présent ne s'offrent pas aisément à la représentation des nombres "très grands" ou "très petits".

Exemple sur 32 bits:

- En naturel **[0, 4294967295] ≃ [0, 4.3*10^9]**.
- En signé complément à deux, partie entière seulement:

  [- 2147483648 , + 2147483648] ≃ [-2.15*10^9, +2.15*10^9].
- En signé complément à deux, 22 bits partie entière, 10 bits partie fractionnaire

  [-2097152.0 , +2097151.9990234375] ≃ [-2.1*10^6 , +2.1*10^6].

La différence entre deux valeurs successives est $2^{-10} \simeq 1/1000$

Il y'a un autre point qu'il faut connaitre, lorsqu'on veut représenter des nombres avec partie entière et partie fractionnaire: c'est le compromis intervalle-précision. Essayons de l'illustrer en base dix.

Soit 5 chiffres décimaux pour écrire des nombres.

Avec une répartition 4 chiffres pour la partie entière et 1 chiffre pour la partie fractionnaire on peut aller (en positif seulement) de 0000,0 à 9999,9 : intervalle grand, petite précision.

Inversement 1 chiffre pour partie entière et 4 chiffres pour partie fractionnaire: 0,0000 à 9,9999 petit intervalle, grande précision.

Le nombres de "nombres" pouvant être représentés est le même dans les deux cas: 10^5.

Représentation des données:

Les nombres réels (suite).

Si nous voulons représenter par exemple des nombres de l'ordre de mille milliards, soit $10^{12} = (10^3)^4 \simeq (2^{10})^4 = 2^{\mathbf{40}}$, il faut en moyenne 40 bits.

Si de plus nous voulons aussi des fractions de l'ordre de un mille milliardième, soit $10^{-12} = (10^3)^{-4} \simeq (2^{10})^{-4} = 2^{\mathbf{-40}}$,il nous faut encore 40 bits. Soit 80 bits en tout. Cela fait beaucoup.

D'où la notation appelée ***format virgule flottante*** (Floating-Point format).

L'appellation ***format virgule fixe*** est donné au format précédent. La virgule (ou le point décimal n'est jamais matérialisé au niveau de l'écriture en binaire.

La notation appelée ***format virgule flottante*** s'inspire de l'écriture dite scientifique des nombres: $\mathbf{\pm XX.XX*10^{YY}}$

Terminologie: Il y'a trois composants dans cette écriture:

- Le signe: **±**.
- La mantisse: **XX.XX**.
- L'exposant: $^{\mathbf{YY}}$.

Le qualificatif flottant vient du fait qu'on peut déplacer le point décimal et jouer sur l'exposant sans changer de valeur au nombre.

Exemple $3584{,}1 * 10^0 = 3{,}5841 * 10^3 = 0{,}35841 * 10^4$.

Représentation des données:

Les nombres réels (suite).

En binaire: $N = \pm M*2^{\pm E}$

Les nombres réels sont représentés selon le schéma suivant:

La mantisse et son signe (S) sont dans la notation signe et valeur absolue.

Avec la convention suivante pour le signe: S=0 pour les nombres positifs, 1 sinon.

L'exposant est celui de 2 et est souvent représenté en *excédent de* B. autrement dit ce qui est écrit dans le champs correspondant est le vrai exposant augmenté de B.

Ces exigences facilitent la comparaison des nombres réels.

Par ailleurs la mantisse est toujours normalisée, c'est-à-dire qu'elle est de la forme: $(\mathbf{1,bbb\ldots bbb})_2$ donc $1 \leq$ mantisse < 2.
Cette dernière exigence choisit une écriture parmi plusieurs et injecte le maximum de bits dans la partie fractionnaire pour plus de précision.

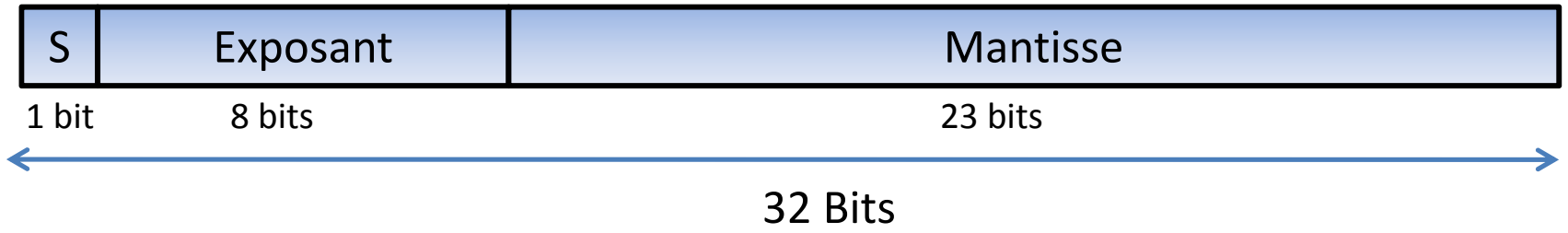
Représentation des données:

La norme IEEE 754.

La norme IEEE 754.

Ce standard a défini au moins deux formats, l'un pour la simple précision, l'autre pour la double précision.

a) Simple précision:



S=0, nombre positif; S=1, nombre négatif.

Exposant en ***excédent de 127***. *(les exposants possibles -126 à +127)*

Mantisse normalisée, sur 23 bits, toujours sous la forme 1.bbb…bbb (le 1 n'est pas écrit dans la configuration).

Représentation des données:

La norme IEEE 754, (simple précision) exemples.

1) Représentez, selon *le format IEEE 754, simple précision*, le nombre $(0,75)_{10}$.

$(0,75)_{10} = (0,11)_2 = (0,11)_2 * 2^0$.

Normalisation: $(0,11)_2 * 2^0 = (1,1)_2 * 2^{-1}$.

Exposant réel est -1. il est écrit en *excédent de 127:* -1 + 127 = 126.

Conversion de 126 sur 8 bits *:* $(126)_{10} = (01111110)_2$.

0	01111110	10000000000000000000000

réponse à donner: $(3F400000)_{16}$

2) Représentez, selon *le format IEEE 754, simple précision*, le nombre $(-10,125)_{10}$.

$(10,125)_{10} = (1010,001)_2 * 2^0$.

Normalisation: $(1010,001)_2 * 2^0 = (1,010001)_2 * 2^3$.

Exposant: 3 + 127 = 130.

1	10000010	01000100000000000000000

réponse: $(C1220000)_{16}$

Représentation des données:

La norme IEEE 754, (simple précision) exemples.

3) Quel est le nombre réel dont le code, selon le standard IEEE 754 simple précision est $(C3880000)_{16}$.

1	10000111	00010000000000000000000

L'exposant représenté est 135. Il est en *excédent de 127.* Donc l'exposant réel est: 135 – 127 = 8.

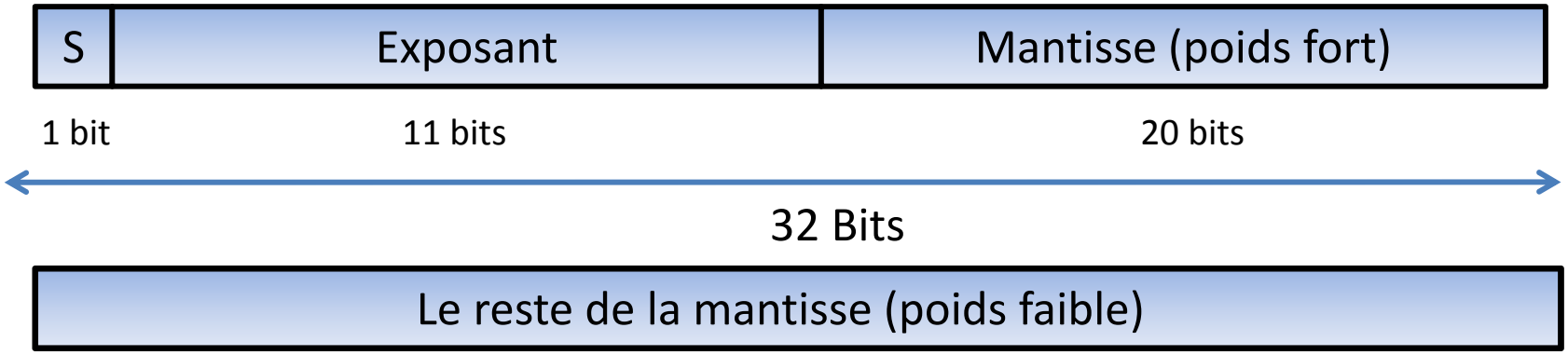
La mantisse est $(1,0001)_2$.

Le Nombre est donc $-(1,0001)_2 * 2^8 = -272$.

Représentation des données:

La norme IEEE 754, (simple précision) exemples.

b) Double Précision:



Exposant en ***excédent de 1023***.

Mantisse normalisée, sur 52 bits, toujours sous la forme 1.bbb…bbb (le 1 n'est pas écrit dans la configuration).

Représentation des données:

La norme IEEE 754,

Les cas particuliers:

Exposant	Mantisse	Signification
0	**0**	L'écriture de zéro
0	Autre que **0**	Mantisse non normalisée
Entre **1** et **254**	Quelconque	Nombre classique Normalisé
255	**0**	**±** Infinité
255	Autre que **0**	**NaN** (**N**ot **a** **N**umber)

Représentation des données:

Les caractères.

Question: Comment représenter le texte, du moins dans sa version simple, (une seule police, une seule couleur, une seule taille, etc.)?

Réponse : les codes de caractères.

Cette partie n'a pas la prétention de discuter dans les détails les différentes propositions normalisées, mais de présenter succinctement une technique de codage et citer une autre à cause de son universalité.

L'un des premiers codes proposé pour coder les caractères est:

ASCII = **A**merican National **S**tandard **C**ode for **I**nformation **I**nterchange.

Défini dans le document: **ANSI document X3.4-1977**

Caractéristiques:

- code sur 7 bits.
- 8ieme bit non utilisé (ou utilisé comme bit de parité).
- $2^7 = 128$ codes
- Deux types de codes:
  - 95 sont dits codes "Graphique" (affichables sur une console)
  - 33 sont dits codes de "Controle"

Représentation des données:

Les caractères: La table ASCII.

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

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

Le bit le plus significatif.

Le bit le moins significatif.

Exemple., ‘A’ = 1000001

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

Représentation des données:

Les caractères: Le code ASCII, exemple.

		Binary		Hexadecimal		Decimal
H	=	01001000	=	48	=	72
e	=	01100101	=	65	=	101
l	=	01101100	=	6C	=	108
l	=	01101100	=	6C	=	108
o	=	01101111	=	6F	=	111
,	=	00101100	=	2C	=	44
	=	00100000	=	20	=	32
w	=	01110111	=	77	=	119
o	=	01100111	=	67	=	103
r	=	01110010	=	72	=	114
l	=	01101100	=	6C	=	108
d	=	01100100	=	64	=	100

Représentation des données:

Les caractères: Unicode

Code sur 16 bits.

`http://www.unicode.org`

`Extrait, tiré de la norme:`

…contains 38,887 distinct coded characters …

These characters cover the principal written languages of the Americas, Europe, the Middle East, Africa, India, Asia, and Pacifica.