Codage des informations

1. Le codage de l'information

Il existe différents types d'informations dans le monde du numérique : du texte, des nombres, des sons, des images, des instructions...

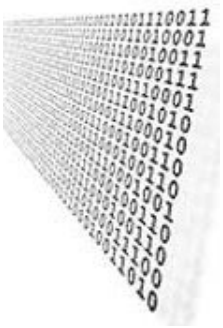
1.1. Types d'informations

Une information numérique peut être de *deux types* :
- une **instruction**, qui représente une opération réalisée par un organe de calcul (un microprocesseur par exemple) ;
- une **donnée**, sur laquelle des instructions pourront être faites.

Parmi les données, il existe des sous-catégories qui ne seront pas traitées de la même manière par les systèmes informatiques. Parmi ces données, on peut citer :
- des **données non numériques** (caractère alphanumérique) ;
- des **données numériques** :
  - entiers naturels (0 ; 1 ; 315 ...)
  - entiers relatifs (-1578 ; -15 ; -1 ...)
  - réels (3.1415 ; 4587.598 ...)

1.2. Représentation de l'information

Les informations numériques sont transmises par des **signaux électriques**. Afin d'avoir un langage universel, ces données sont représentées sous **forme binaire**, c'est à dire une *suite de 0 et de 1*.



L'information élémentaire est appelé **BIT** (BInary digiT).

Un **mot binaire** est un **regroupement de n bits**. Il permet d'obtenir 2^n combinaisons différentes.

Un regroupement de **8 bits** s'appelle **un octet**.

2. Codage des caractères alphanumériques

Les premiers codes ont été établis pour répondre au besoin de transmettre des `messages` rapidement en utilisant les moyens de communication disponibles à l'époque. L'alphabet a été codé au moyen de signaux à états finis : télégraphe Chappe (optique), code Morse (radio)...

Des systèmes de codage basés sur la présence ou l'absence d'un signal électrique (1 ou 0) sont apparus en même temps que l'informatique. On ne retiendra ici que les codes les plus utilisés actuellement : **ASCII** et **Unicode**.

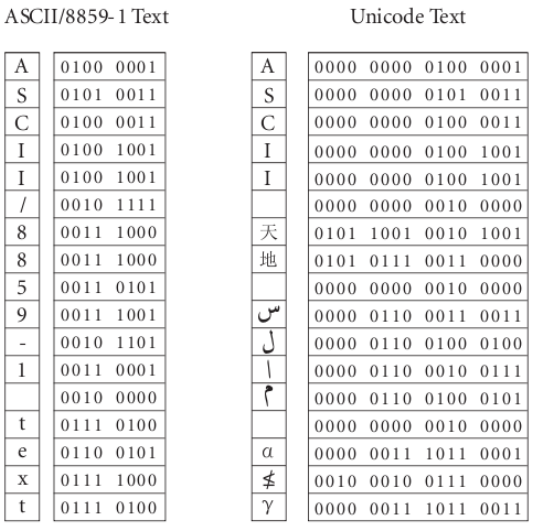
2.1. Code ASCII (American Standard Code for Information Interchange)

Le codage **ASCII** (normalisé en 1967) représente un jeu de **128** caractères au moyen de **7** signaux à 2 états. La table de correspondance est représentée ci-dessous, sous forme rectangulaire pour être plus compacte.

Ce code comprend 26 majuscules, 26 minuscules, 10 chiffres, 32 symboles, 33 codes de contrôle et un espace. Ce codage n'est réellement adapté qu'à la langue anglaise qui est dépourvue de toute *décoration* (ou signes diacritiques : accent, cédille, tilde, etc..) sur les caractères.

Des codes de commande pour les anciens terminaux complètent cette liste : seuls quelques uns sont encore utilisés aujourd'hui (NUL, BS, HT, LF, VT, FF, CR, ESC, DEL).

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	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	EOT	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



Ce code est très limité, mais il suffit à de nombreuses activités techniques dans le domaine du traitement de l'information (systèmes d'exploitation, claviers, imprimantes, internet, etc..).

C'est le codage utilisé en **langage C** pour les variables de type `char`, codé sur 8 bits (ou 1 octet).

2.2. Unicode

Après des tentatives d'extension du code ASCII par les firmes informatiques, **Unicode** a été créé de manière concertée. Au départ, les caractères étaient codés sur 16 bits. Depuis, il a été étendu à 32 bits.

La première version a été développée en 1991. L'unicode est mis à jour régulièrement pour prendre en compte de nouveaux alphabets : 5.2 (2009), 6.0 (2012), 6.1 (2012), 6.3 (2013).

Dans la version 6.0 (janvier 2012), l'Unicode permettait de coder :
- 137 468 caractères à usage privé ;
- 109 242 lettres ou syllabes, chiffres ou nombres, symboles divers, signes diacritiques (accent...) et signes de ponctuation ;
- plusieurs centaines de caractères de contrôle ou modificateurs spéciaux.

Pour assurer la compatibilité, les codes des caractères du jeu ASCII sont les mêmes en ASCII qu'en Unicode, il suffit d'ajouter des zéros à gauche pour arriver à 16 bits.

3. Codage des données numériques

Selon le type de données numériques à traiter, on pourra utiliser des codages différents. Le codage est réalisé à l'aide de l'algorithme de conversion associé au type de la donnée. Les opérations arithmétiques sont ensuite effectuées en arithmétique binaire.

On rappelle ici l'addition et la multiplication en binaire :

0 + 0 = 0	0 x 0 = 0
0 + 1 = 1	0 x 1 = 0
1 + 0 = 1	1 x 0 = 0
1 + 1 = 0 (et une retenue)	1 x 1 = 1

Dans tous les cas, un **code pondéré**, c'est à dire que chaque symbole d'un nombre aura un poids spécifique dans l'écriture de celui-ci, sera utilisé.

3.1. Codage des entiers naturels

3.1.1 Base décimale

La base décimale (dite base 10) possède 10 chiffres {0,1,...,9}.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre. Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

	Centaine		Dizaine		Unité
$345 =$	3×10^2	+	4×10^1	+	5×10^0

On peut généraliser cette équivalence par l'expression : $n = \sum_{i=0}^{p-1} d_i \cdot 10^i$ avec $d_i < 10 \ \forall i$ pour un nombre n écrit avec p chiffres décimaux d_i.

3.1.2 Autres bases

En généralisant ce qui précède, on peut utiliser d'autres bases de numération pour des besoins particuliers.
Soit B la base de numération, soient b_i les chiffres utilisés pour cette base, on a bien évidemment : $b_i < B$, $\forall i$.

La décomposition d'un nombre dans la base B se traduit par : $n = \sum_{i=0}^{q-1} b_i \cdot B^i$ avec $b_i < B \ \forall i$

$$(b_{q-1}...b_1b_0)_B = b_{q-1} \times B^{q-1} + ... + b_1 \times B^1 + b_0 \times B^0$$

Nombre minimum de symboles pour le codage dans une base d'un nombre Le nombre q minimum de symboles nécessaires pour écrire n en base B est donné par l'inégalité :

$$q \geq \frac{\ln n}{\ln B}$$

Dynamique de codage Dans les machines de traitement de l'information, les nombres sont représentés par une certaine quantité de symboles (généralement binaires). Cette quantité est fixée lors de la conception des machines. Il en resulte une certaine étendue ou *dynamique de codage* liée à ce nombre de symboles représentant l'information à coder.

Dynamique de codage des entiers positifs :

Nombre de symboles	Dynamique de codage	Base 10	Base 2
1	0 à $(B^1 - 1)$	0 à 9	0 à 1
2	0 à $(B^2 - 1)$	0 à 99	0 à $(11)_2 = (3)_{10}$
4	0 à $(B^4 - 1)$	0 à 9 999	0 à $(1111)_2 = (15)_{10}$
8	0 à $(B^8 - 1)$	0 à 99 999 999	0 à $(1111\ 1111)_2 = (255)_{10}$
16	0 à $(B^{16} - 1)$	0 à $(10^{16} - 1)$	0 à $(65535)_{10}$
...	...	...	...
k	0 à $(B^k - 1)$	0 à $(10^k - 1)$	0 à $(2^k - 1)_{10}$

3.1.3 Base binaire / Codage binaire naturel

La base binaire (dite base 2) possède 2 symboles {0,1}.

Un **nombre binaire** est une somme de termes où chacun est une **puissance de deux** multipliée par un symbole.

$$1001 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Le symbole le plus à gauche d'un nombre et qui a le plus de poids dans l'écriture d'un nombre binaire est appelé **bit de poids fort** ou **MSB** (Most Significant Bit). De la même manière, le symbole le plus à droite et qui a le moins de poids est appelé **bit de poids faible** ou **LSB** (Less Significant Bit).

3.1.4 Base hexadécimale

La base hexadécimale (dite base 16) possède 16 symboles {0,1,...,9,A,B,C,D,E,F} (où A=10, B=11,... F=15).

Un **nombre hexadécimal** est une somme de termes où chacun est une **puissance de seize** multipliée par un symbole.

$$D7A = D \times 16^2 + 7 \times 16^1 + A \times 16^0$$

La manipulation des nombres en base 2 est une nécessité pour les informaticiens, mais c'est une tâche lourde et propice aux erreurs.
La base hexadécimale (base 16) simplifie beaucoup la manipulation des machines.

La correspondance entre un groupe de 4 chiffres en base 2 et le chiffre hexadécimal correspondant est indiqué dans le tableau ci-après :

Base 2	Base 16	Base 10
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Base 2	Base 16	Base 10
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

3.2. Transcodage

Le transcodage est le fait de **convertir un nombre** d'une base vers une autre.

3.2.1 Base B vers Base décimale

Pour passer d'une base B vers la base décimale (base la plus utilisée par l'être humain), on utilise le principe de la décomposition d'un nombre dans cette base B (vu précédemment).

Ainsi, un nombre dans une base B quelconque s'écrira :

$$(b_{q-1}...b_1b_0)_B = b_{q-1} \text{ x } B^{q-1} + ... + b_1 \text{ x } B^1 + b_0 \text{ x } B^0$$

3.2.2 Base décimale vers Base B

Pour le transcodage inverse, de la base décimale vers une base B quelconque, on divise successivement le nombre à transcoder par B. Les calculs s'arrêtent lorsque le quotient arrive à 0.

$$\begin{array}{c|c} N & B \\ \hline a_0 & N_1 \end{array} \rightarrow \begin{array}{c|c} N_1 & B \\ \hline a_1 & N_2 \end{array} \rightarrow ... \begin{array}{c|c} N_{n-1} & B \\ \hline a_{n-1} & 0 \end{array}$$

Pour obtenir la valeur convertie, il faut ensuite récupérer l'ensemble des restes de ces divisions en partant du **dernier reste obtenu**, qui correspond alors au poids fort du nombre final (a_{n-1}), **jusqu'au premier reste** calculé qui correspond au poids faible (a_0).

3.3. Codage des entiers relatifs

3.3.1 Problématique

Lors de traitements arithmétiques, il est nécessaire de disposer des nombres entiers négatifs, pour que le résultat de certaines opérations puisse être représenté. Dans le cas d'une machine ayant un certain nombre de bits, et donc une certaine dynamique de codage en binaire naturel, la solution consiste à **sacrifier une partie de cet intervalle** pour la représentation des nombres négatifs. Généralement, on partage l'intervalle en deux morceaux égaux l'un pour les nombres positifs et la valeur zéro, et l'autre pour les nombres négatifs. Il y a donc une légère dissymétrie entre les nombres positifs et négatifs, sans grande importance en pratique.

3.3.2 Dynamique de codage des entiers relatifs

Dans le cas des machines les plus courantes, les dynamiques de codage deviennent :

Nombre de bits	Dynamique de codage
4	de -8 à +7
8	de -128 à +127
16	de -32768 à +32767
32	de -2 147 483 648 à +2 147 483 647

3.3.3 Code complément à 2

Le **code complément à 2** (C2 en abrégé) est un outil permettant de **coder des nombres entiers relatifs** tout en garantissant la dynamique de codage précédente et permettant les calculs arithmétiques entre deux nombres codés ainsi.

Le codage obtenu est alors **pondéré** : le bit de poids fort représente une valeur négative alors que les autres bits représentent des valeurs positives. Ce codage n'a donc de sens que pour un **nombre de bits convenu à l'avance** : le code C2 n'est pas extensible à volonté.

On doit donc impérativement connaître (ou trouver) le nombre de bits à employer pour établir le code C2 d'un nombre N.

Dans tous les cas, le nombre sera codé de la façon suivante :

$$N = -b_{q-1} \cdot 2^{q-1} + \sum_{i=0}^{q-2} b_i \cdot 2^i$$

où b_i représente un bit (0 ou 1) de poids i.

3.3.4 Algorithme de calcul du complément à 2

Il existe une méthode systématique pour le calcul du complément à 2 d'un nombre.

Si nécessaire, rechercher le nombre minimal de bits p pour coder le nombre n en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **Si** $N \geq 0$ (nombres de 0 à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à p bits (en complétant à gauche par des 0). Le bit de poids fort est égal à 0.
- **Si** $N < 0$ (nombres de -2^{p-1} à -1) :
  1. coder $|N|$ en binaire en complétant à gauche par des 0 pour obtenir un code sur p bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter 1 au résultat (**complèment à deux** ou C2)

Une autre méthode de codage peut être utilisée pour les nombres négatifs : additionner 2^q à N, puis coder le nombre obtenu en binaire naturel.

3.4. Codage des nombres réels

3.4.1 Problématique

Nous avons vu précédemment que la **représentation binaire** d'un nombre était **limitée** sur une plage de valeurs appelée la **dynamique de codage**. Sur cet intervalle de valeurs, on pouvait alors coder tous les nombres entiers présents, car ils sont en nombre limité.

Sur l'ensemble des réels, la problématique est tout autre. En effet, sur le même intervalle que précédemment, il existe une infinité de nombres réels. La représentation binaire entrainera alors systèmatiquement une **imprécision** dans le codage.

Deux possibilités existent alors pour coder des nombres réels en binaire :
- en **virgule fixe** : la partie entière et la partie réelle seront toujours codées sur le même nombre de bits ;
- en **virgule flottante** : le nombre à coder sera ramené à un certains nombres de chiffres significatifs et un exposant (norme IEEE 754).

3.4.2 Codage en virgule fixe

On considère un dénominateur *implicite* 2^p commun à toute la représentation. Le code équivalent est généralement appelé Q_p.

Il est alors possible de représenter le nombre codé par la relation suivante :

$x =$	a_{n-1}	+	a_{n-1}	+ ... +	a_1	+	a_0
	x		x		x		x
Rang	$p^{n-1-p-1}$		p^{n-p-2}	...	2^{-p+1}		2^{-p}
Puissance	n-1-p-1		n-p-2	...	-p+1		-p

La méthode de codage d'un réel x sur n bits en code Q_p peut être décrite par :
- multiplier x par 2^p ;
- ne conserver que la partie entière du résultat précédent et la coder en code C2.

Cette méthode de codage est souvent utilisée dans les applications de traitement audio ou vidéo "grand public", la majorité des processeurs de traitement du signal étant basée sur cette technique moins couteuse que le codage en virgule flottante.

3.4.3 Codage en virgule flottante : norme IEEE754

En codant séparément des chiffres significatifs et un exposant, le tout en mode binaire, le codage en virgule flottante des nombres réels présente un énorme avantage pour les calculs scientifiques : les débordements de capacité sont bien moins fréquents, car l'étendue de l'intervalle codé est bien plus grand. On peut mener les calculs sans être obligé de surveiller les dépassements à chaque étape ! De plus, par construction, la précision relative des nombres est constante et indépendante de leur magnitude, car ils ont un nombre de bits significatifs constant.

Cette méthode assure une utilisation optimale des bits consacrés à la représentation des nombres. Cependant, la complexité des opérateurs permettant les opérations de base est incomparablement plus grande que dans le cas de la représentation en virgule fixe. Il a fallu attendre longtemps avant que les machines de traitement de l'information aient accès à des *coprocesseurs numériques* spécialisés dans ces opérations.

Régles de codage IEEE 754 simple précision (32 bits) : Un bit est réservé pour le signe S de la **mantisse** (0 pour positif, 1 pour négatif), 8 bits pour l'**exposant** E et 23 bits pour la partie fractionnaire de la mantisse F.

Bit	31	30..23	22..0
Dénomination	S	E	F

La formule permettant de trouver la valeur décimale à partir de la représentation IEEE est :

$$v = (-1)^S \cdot 2^{E-127} \cdot F$$

où E et F sont des représentations binaires et $0 < E < 255$.

Cas spéciaux :
- Si $E = 255$ et $F \neq 0$ alors v est NaN (Not a Number : code illicite) ;
- Si $E = 255$ et $F = 0$ alors v est $\pm\infty$ selon la valeur de S ;
- Si $E = 0$ et $F \neq 0$ alors v est un nombre dénormalisé dont la valeur est : $v = (-1)^S \cdot 2^{E-126} \cdot F$

Il existe également un format double précision (64 bits).

3.5. Autres codes

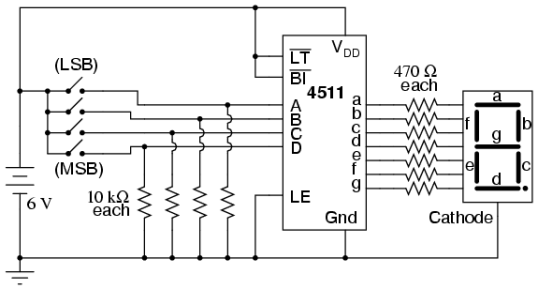
En dehors des codes purement arithmétiques, il existent de nombreux autres codes ayant un domaine d'application particulier. Nous n'en retiendrons que deux :
- le code BCD ;
- le code Gray.

3.5.1 Code BCD (binary coded decimal : décimal codé binaire)

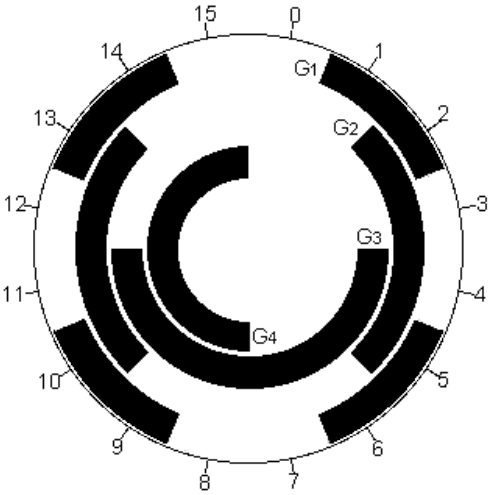
Chaque chiffre de la représentation décimale est codé en binaire naturel sur 4 bits.

Décimal	2	6	9
BCD	0010	0110	1001

Ce code est encore utilisé pour communiquer avec des dispositifs décimaux, comme certains afficheurs.



3.5.2 Code Gray



Le code **Gray** ou *binaire réfléchi* a été initialement créé pour résoudre les problèmes liés aux codeurs de position absolue.

Le passage d'une position à l'autre n'entraine alors le **changement que d'un seul bit**, évitant ainsi des codes intermédiaires fugitifs (dûs aux temps de réaction des capteurs, par exemple).

Les convertisseur le code binaire en code Gray et inversement sont très simples.

Si l'on désigne par b_n un bit quelconque en code binaire et par g_n le bit recherché en code Gray, on a alors : $g_n = b_n \oplus b_{n+1}$

Pour la conversion du code Gray en binaire, on a : $b_n = g_n \oplus b_{n+1}$

Méthode pour construire une table de code Gray à 2^n éléments

La méthode peut être décrite par l'algorithme suivant :

1. si $n = 2$, la table T_2 est composée des éléments 0 et 1 ;
2. si $n > 2$, la table T_n se construit à partir de la table T_{n-1}de taille $n - 1$ de la manière suivante :
   - on recopie la liste des $n - 1$ éléments de T_{n-1} ;
   - on ajoute la liste des $n - 1$ éléments de T_{n-1} en ordre inverse (en miroir) ;
   - on préfixe les $n - 1$ premiers éléments de 0 et les $n - 1$ éléments suivants de 1.

La méthode est illustrée ci-dessous pour les tables T_2, T_4 et T_8.

T_2	T_4	T_8
0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100