

Chap II : Codification

1. Le code binaire naturel

1.1. Présentation

C'est la représentation des nombres entiers avec seulement deux chiffres (base 2) : 01.

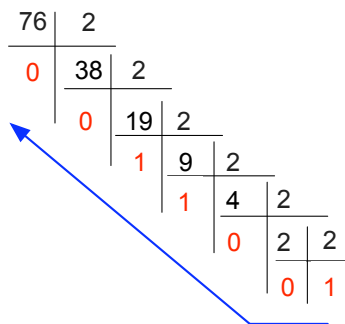
Un nombre N qui s'écrit $b_n b_{n-1} \dots b_1 b_0$ (avec $b=0$ ou $b=1$) vaut : $\sum_{i=0}^{i=n} b_i \times 2^i$.

Ainsi : (0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7...) devient (0 ; 1 ; 10 ; 11 ; 100 ; 101 ; 110 ; 111...)

Remarque : Les chiffres composants un nombre binaire s'appellent bit.

1.2. Conversion par divisions euclidiennes

Il s'agit de diviser récursivement N par 2, et de considérer le reste.



On a donc $76 = 100\ 1100_{(2)}$

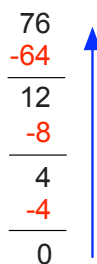
Vérification :

$$100\ 1100_{(2)} = 2^6 + 2^3 + 2^2 = 64 + 8 + 4 = 76$$

1.3. Conversion par soustractions

Il s'agit de soustraire récursivement à N la puissance de 2 immédiatement inférieure.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1



On a donc $76 = 2^6 + 2^3 + 2^2$

$$76 = 100\ 1100_{(2)}$$

1.4. Addition de deux nombres binaires

On procède comme avec les nombres décimaux. Il suffit de mettre les deux nombres l'un au dessus de l'autre, et on fait l'addition bit à bit. Attention, il ne faut pas oublier la retenue.

Exemple :

$$\begin{array}{r}
 1\ 1\ 1 \\
 0\ 1\ 1\ 1 \\
 +\ 0\ 1\ 0\ 1 \\
 \hline
 1\ 1\ 0\ 0
 \end{array}$$

1.5. Limite

Les nombres entiers dans les appareils numériques sont représentés avec un nombre n limité de bits (généralement 8 ou 16). Donc, on ne pourra représenter que les nombres compris entre 0 et 2^{n-1} .

Attention : si les nombres sont codés sur 8 bits, $1111\ 1111 + 1 = 0$! Il y a *débordement*.

2. Représentation des nombres entiers signés

2.1. Bit de signe

Pour compléter la représentation des entiers, il faut pouvoir écrire des entiers négatifs. On ajoute pour cela à la représentation un bit de signe, placé en tête. Un bit de signe 0 indique une valeur positive, un bit de signe positionné à 1 une valeur négative. Cette règle permet de rester cohérent avec le système de représentation des entiers positifs : il suffit d'ajouter un 0 en tête de chaque valeur.

- $0111_{(2\text{-bit-de-signé})} = +7$;
- $1111_{(2\text{-bit-de-signé})} = -7$;

Ce code n'est pas utilisé par le microprocesseur car la somme d'un chiffre positif et du même chiffre mais négatif ne donne pas 0.

2.2. Complément à deux

Finalement, il faut que $xxxx + (-xxxx) = 0000$. C'est le nombre que l'on appelle le complément à 2.

Exemple :

$$\begin{array}{r} 1 \quad 1 \quad 1 \quad 1 \\ \quad 0 \quad 1 \quad 1 \quad 1 \\ + \quad 1 \quad 0 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 0 \quad 0 \quad 0 \end{array}$$

Ainsi, $1001_{(C2)} = -7$

Comme précédemment, les nombres débutants par un 1, sont négatifs.

Comment obtenir le complément à 2 :

- On remplace tous les bits 0 par des 1 et vice-versa ;
- On ajoute 1, au nombre obtenu.

Remarque : Avec n bits, ce système permet de représenter les nombres entre -2^{n-1} et $2^{n-1}-1$.

2.3. Soustraction de deux nombres binaires

Pour soustraire deux nombres binaires, il suffit d'additionner au premier le complément à deux du second.

Exemple : $0101_{(C2)} - 0011_{(C2)}$

Complément à 2 de $0011_{(C2)}$: $1100_{(C2)} + 1 = 1101_{(C2)}$

$0101_{(C2)} - 1101_{(C2)} = 0010_{(C2)}$

$$\begin{array}{r} 1 \quad 1 \quad \quad 1 \\ \quad 0 \quad 1 \quad 0 \quad 1 \\ + \quad 1 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 0 \quad 1 \quad 0 \end{array}$$

3. Représentation des nombres décimaux

3.1. Nombre à virgule fixe

Comme la définition des nombres binaires naturels, on peut définir un nombre décimal positif par une convention du même type, la virgule étant placée à un endroit fixe :

$$1010,1010_{(2)} = 2^3 + 2 + 2^{-1} + 2^{-3} = 8 + 2 + 0,5 + 0,125 = 10,625$$

Dans le sens inverse, prenons $N = 0,347$ (Pour la partie devant la virgule, il suffit de faire comme dans le paragraphe 2.2.

- $0,347 \times 2 = 0,694 < 1$, je pose 0 : $0,347 = 0,0_{(2)}$
- $0,694 \times 2 = 1,388 > 1$, je pose 1 et je le retranche : $0,347 = 0,01_{(2)}$
- $0,388 \times 2 = 0,766 < 1$, je pose 0 : $0,347 = 0,010_{(2)}$
- $0,766 \times 2 = 1,552 > 1$, je pose 1 et je le retranche : $0,347 = 0,0101_{(2)}$

Avec notre codage sur 8 bits on s'arrête là ; $0,347 = 0,0101_{(2)}$ à 2^{-4} près.

Ce codage est très peu utilisé car il ne permet pas d'avoir un grand rapport N_{\max}/N_{\min} .

3.2. Nombre à virgule flottante (float)

La norme IEEE définit le code de représentation d'un nombre décimal sur 32 bits (*single*) avec trois composantes :

- Le signe est représenté par un seul bit, le bit de poids fort (celui le plus à gauche) ;
- L'exposant est codé sur les 8 bits consécutifs au signe ;
- La mantisse sur les 23 bits restants.

signe	exposant	mantisse
1 bit	8 bits	23 bits
x	XXXX XXXX	XXX XXXX XXXX XXXX XXXX XXXX

On a :

$$N = (\text{signe}) \times 2^{\text{exposant}-127} \times 1, \text{ mantisse}_{(2)}$$

Attention :

- Signe = 1 : nombre négatif ;
- L'exposant 0 est interdit ;
- L'exposant 255 est interdit. On s'en sert toutefois pour signaler des erreurs, on appelle alors cette configuration du nombre NaN, ce qui signifie 'Not a Number'.

3.3. Conversion décimal vers binaire

Rien ne vaut un exemple, donc prenons $N = -6,625$.

1. Signe = 1 car le nombre est négatif.
2. On divise le nombre successivement par 2 jusqu'à obtenir un nombre du type 1,xxxx.

$$6,625/2 = 3,3125 [1]$$

$$3,3125/2 = 1,65625 [2] : \text{Exposant} = 2 + 127 = 129 = 128 + 1 = 1000\ 0001_{(2)}$$

3. On détermine la valeur binaire de la mantisse codée sur 23 bits.

$$0,65625 \times 2 = 1,3125 \Rightarrow 1$$

$$0,3125 \times 2 = 0,625 \Rightarrow 0$$

$$0,625 \times 2 = 1,25 \Rightarrow 1$$

$$0,25 \times 2 = 0,5 \Rightarrow 0$$

$$0,5 \times 2 = 1 \Rightarrow 1$$

Ainsi, la représentation de $-6,625$ est :

signe	exposant	mantisse
1 bit	8 bits	23 bits
1	1000 0001	101 0100 0000 0000 0000 0000

4. Le code binaire réfléchi (code Gray)

4.1. Présentation

C'est un codage qui a la particularité de ne changer qu'un bit lors du passage d'une valeur à sa suivante. De plus, on opère de telle manière que le bit modifié soit d'un poids le plus faible possible.

La constitution de cette table réclame l'exercice suivant :

- Écrire les deux premières valeurs : 0 et 1 ;
- Ajouter 1 devant les 2^1 suivants ;
- Par un effet miroir on complète les premiers bits ;
- Ajouter 1 devant les 2^2 suivants ;
- Par un effet miroir on complète les premiers bits ;
- Etc...

Decimal	GRAY
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Ainsi :

(0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7...) devient (000 ; 001 ; 011 ; 010 ; 110 ; 111 ; 101 ; 100...)

4.2. Codeur de position

Ce code est surtout utilisé pour des capteurs de positions absolue, comme les règles optiques. Il permet une résolution double de la règle avec la même précision de gravure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CODE GRAY															

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CODE BINAIRE NATURELLE															

5. Le code binaire BCD

Il suffit de remplacer chaque chiffre décimal par son image binaire de quatre bits.

Exemple :

$$\begin{matrix} 1 & 2 & 8 & (10) \\ \underbrace{} & \underbrace{} & \underbrace{} & \\ 0001 & 0010 & 1000 & (BCD) \end{matrix}$$

6. Le code hexadécimal

6.1. Présentation

La représentation binaire des nombres amène vite à des représentations de nombre très longue.

$$1024 = 10\ 0000\ 0000_{(2)}$$

Pour simplifier la représentation des nombres tout en se rapprochant de la représentation binaire, on utilise la représentation hexadécimale.

C'est la représentation des nombres avec 16 chiffres (base 16) : 0123456789ABCDEF.

Un nombre N qui s'écrit $h_n h_{n-1} \dots h_1 h_0$ vaut :

$$\sum_{i=0}^{i=n} h_i \times 16^i = \sum_{i=0}^{i=n} h_i \times (2^4)^i$$

Ainsi :

$$1024 = 200_{(16)}$$

Remarque :

Avec la base 16 un nombre est représenté avec 4 fois moins de chiffres qu'avec la base 2.

6.2. Division euclidienne

Il s'agit de diviser récursivement N par 16, et de considérer le quotient et le reste.

Exemple :

$$\begin{array}{r|l} 76 & 16 \\ -64 & 4 \\ \hline 12 & \\ \leftarrow & \end{array}$$

On a donc :

$$76 = 4 \times 16 + 12$$

$$76 = 4C$$

6.3. Tableau des 19 premiers nombres

Décimal	Binaire	Hexadécimal	Décimal	Binaire	Décimal
0	0000	0	10	1010	A
1	0001	1	11	1011	B
2	0010	2	12	1100	C
3	0011	3	13	1101	D
4	0100	4	14	1110	E
5	0101	5	15	1111	F
6	0110	6	16	1 0000	10
7	0111	7	17	1 0001	11
8	1000	8	18	1 0010	12
9	1001	9	19	1 0011	13

6.4. Conversion Hexa/Binaire

Comme $16=2^4$, le technique consiste à grouper les bits par quatre et d'utiliser le tableau ci-avant.

Exemple :

$$\underbrace{11}_3 \quad \underbrace{1010}_A \quad \underbrace{1100}_C \quad \underbrace{0001}_1 \quad (2)$$

(16)

7. Le code ASCII

7.1. Présentation

La norme ASCII (American Standard Code for Information Interchange « Code américain normalisé pour l'échange d'information ») est la norme de codage de caractères en informatique la plus connue et la plus largement compatible. C'est également la variante américaine du codage de caractères ISO/CEI 646. ASCII contient les caractères nécessaires pour écrire en anglais. Elle a été inventée par l'américain Bob Bemer en 1961. Elle est à la base de nombreuses autres normes comme ISO 8859-1 et Unicode qui l'étendent.

L'ASCII définit 128 caractères numérotés de 0 à 127 et codés en binaire de 0000000 à 1111111. Sept bits suffisent donc pour représenter un caractère codé en ASCII. Toutefois, les ordinateurs travaillant presque tous sur huit bits (un octet) depuis les années 1970, chaque caractère d'un texte en ASCII est stocké dans un octet dont le 8^e bit est 0.

Les caractères de numéro 0 à 31 et le 127 ne sont pas affichables ; ils correspondent à des commandes de contrôle de terminal informatique. Le caractère numéro 32 est l'espace. Les autres caractères sont les

chiffres arabes, les lettres latines majuscules et minuscules et quelques symboles de ponctuation.(D'après WIKIPEDIA)

7.2. Table des 128 caractères ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					eot			bel	bs		lf		ff	cr		
1												esc				
2	sp	!	”	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z					
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	del

Ce tableau n'est pas à apprendre par cœur il est toutefois trois points de repère simples :

- Le zéro est codé $30_{(16)}$ et les autres chiffres suivent ;
- Le « A » majuscule est codé $41_{(16)}$ et les autres lettres suivent ;
- Le « a » minuscule est codé $61_{(16)}$ et les autres lettres suivent.

Cette façon de coder les lettres n'est pas anodine. Elle permet, par exemple, le classement alphabétique car le codage de A est un nombre inférieur à celui qui code B donc est classé avant.