CHAPITRE 1 : REPRESENTATION DES DONNEES

1. Introduction

Les informations traitées par un ordinateur peuvent être de différents types (texte, nombres, etc.) mais elles sont toujours représentées et manipulées par l'ordinateur sous forme binaire. Toute information sera traitée comme une suite de 0 et de 1. L'unité d'information est donc le chiffre binaire (0 ou 1), que l'on appelle bit (pour binary digit, chiffre binaire).

Le codage d'une information consiste à établir une correspondance entre la représentation externe (habituelle) de l'information (le caractère A ou le nombre 36 par exemple), et sa représentation interne dans la machine, qui est une suite de bits.

On utilise la représentation binaire car elle est simple, facile à réaliser techniquement. Enfin, les opérations arithmétiques de base (addition, multiplication etc.) sont faciles à exprimer en base 2 (noter que la table de multiplication se résume à 0x0 = 0, 1x0 = 0 et 1x1 = 1).

2. Changements de bases

Avant d'aborder la représentation des différents types de données (caractères, nombres naturels, nombres réels), il convient de se familiariser avec la représentation d'un nombre dans une base quelconque (par la suite, nous utiliserons souvent les bases 2, 8, 10 et 16).

Habituellement, on utilise la base 10 pour représenter les nombres, c'est à dire que l'on écrit à l'aide de 10 symboles distincts, les chiffres.

En base **b**, on utilise b chiffres. Notons $\mathbf{a_i}$ la suite des chiffres utilisés pour écrire un nombre $\mathbf{x=a_n\ a_{n-1}\ \ldots\ a_1\ a_0\ .\ a_0}$ est le chiffre des unités.

- En décimal, b =10, ai $\in$ {0,1,2,3,4,5,6,7,8,9};
- En binaire, b =2, ai $\in$ {0,1}: 2 chiffres binaires, ou bits;
- En hexadécimal, b =16, ai $\in$ {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F} (on utilise les 6 premières lettres comme des chiffres).

2.1. Représentation des nombres entiers

En base 10, on écrit par exemple 1996 pour représenter le nombre

$$\mathbf{1996 = 1\ 10^3 + 9\ 10^2 + 9\ 10^1 + 6\ 10^0}$$

En général, en base **b**, le nombre représenté par une suite de chiffres $\mathbf{a_n\ a_{n-1}\ \ldots a_1\ a_0}$ est donné par :

$$a_n a_{n-1} \ldots a_1 a_0 = \sum_{i=0}^{n} a_i b^i$$

a_0 est le chiffre de poids faible, et an le chiffre de poids fort.

Exemple en base 2 :

$$(101)_2 = 1\ 2^2 + 0\ 2^1 + 1\ 2^0 = 4 + 0 + 1 = 5$$

La notation $(\ )_b$ indique que le nombre est écrit en base **b**.

2.2. Représentation des nombres fractionnaires

Les nombres fractionnaires sont ceux qui comportent des chiffres après la virgule.

Dans le système décimal, on écrit par exemple :

$$12{,}346 = 1\ 10^1 + 2\ 10^0 + 3\ 10^{-1} + 4\ 10^{-2} + 6\ 10^{-3}$$

En général, en base **b**, on écrit :

$$a_n\ a_{n-1} \ldots a_1\ a_0, a_{-1}\ a_{-2} \ldots a_{-p} = a_n\ b_n + a_{n-1}\ b_{n-1} + \ldots + a_0\ b_0 + a_{-1}\ b_{-1} + \ldots + a_{-p}\ b_{-p}$$

2.3. Passage d'une base quelconque à la base 10

Il suffit d'écrire le nombre comme ci-dessus et d'effectuer les opérations en décimal.

Exemple en hexadécimal :

$$(AB)_{16} = 10\ 16^1 + 11\ 16^0 = 160 + 11 = (171)_{10}$$

(en base 16, A représente 10, B 11, et F 15).

2.4. Passage de la base 10 vers une base quelconque

2.4.1. Nombres entiers

On procède par divisions successives. On divise le nombre par la base, puis le quotient obtenu par la base, et ainsi de suite jusqu'a obtention d'un quotient nul.

La suite des restes obtenus correspond aux chiffres dans la base visée, $a_0\ a_1 \ldots a_n$.

Exemple : soit à convertir $(44)_{10}$ vers la base 2.

$$44 = 22 * 2 + 0 \implies a_0 = 0$$

$$22 = 11 * 2 + 0 \implies a_1 = 0$$

$$11 = 2 * 5 + 1 \implies a_2 = 1$$

$$5 = 2 * 2 + 1 \implies a_3 = 1$$

$$2 = 1 * 2 + 0 \implies a_4 = 0$$

$$1 = 0 * 2 + 1 \implies a_5 = 1$$

Donc $(44)_{10} = (101100)_2$.

2.4.2. Nombres fractionnaires

On multiplie la partie fractionnaire par la base en répétant l'opération sur la partie fractionnaire du produit jusqu'a ce qu'elle soit nulle (ou que la précision voulue soit atteinte).

Pour la partie entière, on procède par divisions comme pour un entier.

Exemple : conversion de $(54{,}25)_{10}$ en base 2

Partie entière : $(54)_{10} = (110110)_2$ par divisions.

Partie fractionnaire :

$$0{,}25 * 2 = 0{,}50 \Longrightarrow a_{-1} = 0$$

$$0{,}50 * 2 = 1{,}00 \Longrightarrow a_{-2} = 1$$

$$0{,}00 * 2 = 0{,}00 \Longrightarrow a_{-3} = 0$$

2.5. Cas des bases 2, 8 et 16

Ces bases correspondent à des puissances de 2 (2^1, 2^3 et 2^4), d'où des passages de l'une à l'autre très simples. Les bases 8 et 16 sont pour cela très utilisées en informatique, elles permettent de représenter rapidement et de manière compacte des configurations binaires.

La base 8 est appelée notation octale, et la base 16 notation hexadécimale.

Chaque chiffre en base 16 (2^4) représente un paquet de 4 bits consécutifs. Par exemple :

$$\mathbf{(10011011)_2 = (1001\ 1011)_2 = (9B)_{16}}$$

De même, chaque chiffre octal représente 3 bits.

On manipule souvent des nombres formés de 8 bits, nommés octets, qui sont donc notés sur 2 chiffres hexadécimaux.

3. Codification des nombres entiers

La représentation (ou codification) des nombres est nécessaire afin de les stocker et manipuler par un ordinateur. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fixé.

3.1. Entiers naturels

Les entiers naturels (positifs ou nuls) sont codés sur un nombre d'octets fixé (un octet est un groupe de 8 bits). On rencontre habituellement des codages sur 1, 2 ou 4 octets, plus rarement sur 64 bits (8 octets, par exemple sur les processeurs DEC Alpha).

Un codage sur n bits permet de représenter tous les nombres naturels compris entre **0** et $\mathbf{2^n - 1}$. Par exemple sur **1 octet**, on pourra coder les nombres de **0 à 255** = $2^8 - 1$.

On représente le nombre en base 2 et on range les bits dans les cellules binaires correspondant à leur poids binaire, de la droite vers la gauche. Si nécessaire, on complète à gauche par des zéros (bits de poids fort).

3.2. Entiers relatifs

Il faut ici coder le signe du nombre. On utilise le codage en complément à deux, qui permet d'effectuer ensuite les opérations arithmétiques entre nombres relatifs de la même façon qu'entre nombres naturels.

- *Entiers positifs ou nuls* **:** On représente le nombre en base 2 et on range les bits comme pour les entiers naturels. Cependant, la cellule de poids fort est toujours à 0 : on utilise donc n-1 bits.
Le plus grand entier positif représentable sur n bits en relatif est donc 2^{n-1} -1.
- *Entiers négatifs* **:** Soit x un entier positif ou nul représenté en base 2 sur n-1 bits

$$x = \sum_{i=0}^{n-2} \alpha_i 2^i, avec \alpha_i \in \{0,1\}$$

et soit

$$y = \sum_{i=0}^{n-2} (1 - \alpha_i) 2^i + 1$$

On constate facilement que, $x + y = 2^{n-1}, \forall \alpha_i$

Or sur n bits, 2^{n-1} est représenté par n-1 zéros, donc on à x+y=0 modulo 2^{n-1}, ou encore y=-x .y peut être considéré comme l'opposé de x.

La représentation de –x est obtenue par complémentation à 2^{n-1} de x. On dit ***complément à deux***.

Pour obtenir le codage d'un nombre x négatif, on code en binaire sa valeur absolue sur n-1 bits, puis on complémente (ou inverse) tous les bits et on ajoute 1.

Exemple : soit à coder la valeur -2 sur 8 bits. On exprime 2 en binaire, soit 00000010. Le complément à 1 est 11111101. On ajoute 1 et on obtient le résultat : 1111 1110.

Remarques :

- le bit de poids fort d'un nombre négatif est toujours 1;
- sur n bits, le plus grand entier positif est $2^{n-1} – 1 = 011… 1$;
- sur n bits, le plus petit entier négatif est -2^{n-1}.

4. Représentation en virgule flottante (Norme IEEE 754) :

Elle consiste à représenter les nombres sous la forme :

$N = M * B^E$ ‖ M : mantisse
‖ B : base (2, 8, 10, 16…)
‖ E : exposant

Un flottant est stocké selon la norme IEEE 754, analogue à la représentation scientifiques des nombres fractionnaires (tel que 2.5 10-5).

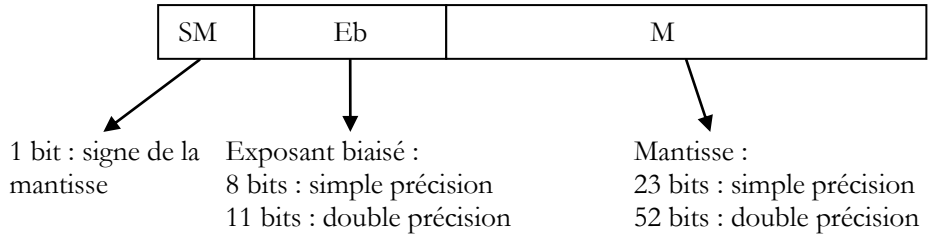


Figure : *représentation en virgule flottante.*

La représentation IEEE code séparément le signe du nombre, l'exposant, et la mantisse (la suite de bits après la virgule), le tout sur 32 bits.

En simple précision, **32** bits sont employés pour la représentation. Ainsi, le nombre $\mathbf{(-1)^{signe} \times 1,f \times 2^{e-127}}$ est représenté sous la forme suivante :

signe	**e**	**f**

31 30 23 22 0

Par exemple :

- 1 10000001 01000000000000000000000 représente :

  Bit du signe = 1 ➔ nombre négatif

  e - 127 = $(10000001)_2$ - 127 = 129 - 127 = 2

  f = $(0,01)_2$ = 0,25

  Donc le nombre représenté est $-1,25 \times 2^2 = -5$.

- +0,25 = $(0,01)_2$ est représenté par :

  Nombre positif ➔ Bit du signe = 0

  $(0,01)_2 = 1,0 \times 2^{-2} = 1,0 \times 2^{125-127}$

  Donc +0,25 est représenté par 0 01111101 00000000000000000000000

Remarques :

- Les exposants 00000000 et 11111111 sont interdits :

  L'exposant 00000000 signifie que le nombre est dénormalisé ;

  L'exposant 11111111 indique que l'on n'a pas affaire à un nombre (On note cette configuration NaN(Not a Number) et on l'utilise pour signaler des erreurs de calculs, comme par exemple une division par 0)

- Les plus petit exposant est donc -126 et le plus grand +127

5. Représentation des caractères

Les caractères sont des données non numériques : il n'y a pas de sens à additionner ou multiplier deux caractères. Par contre, il est souvent utile de comparer deux caractères, par exemple pour les trier dans l'ordre alphabétique.

Les caractères, appelés symboles alphanumériques, incluent les lettres majuscules et minuscules, les symboles de ponctuation (& ~ , . ; # " - etc...), et les chiffres.

Un texte, ou chaîne de caractères, sera représenté comme une suite de caractères.

Le codage des caractères est fait par une table de correspondance indiquant la configuration binaire représentant chaque caractère. Les deux codes les plus connus sont l'EBCDIC de IBM (en voie de disparition) et le code ASCII (American Standard Code for Information Interchange).

Le code ASCII représente chaque caractère sur 7 bits (on parle parfois de code ASCII étendu, utilisant 8 bits pour coder des caractères supplémentaires).

Notons que le code ASCII original, défini pour les besoins de l'informatique en langue anglaise, ne permet la représentation des caractère accentués (é, è, à, ù, ...), et encore moins des caractères chinois ou arabes. Pour ces langues, d'autres codages existent, utilisant 16 bits par caractères.

La table.1 donne le code ASCII. A chaque caractère est associé une configuration de 8 chiffres binaires (1 octet), le chiffre de poids fort (le plus à gauche) étant toujours égal à zéro. La table indique aussi les valeurs en base 10 (décimal) et 16 (hexadécimal) du nombre correspondant.

Plusieurs points importants à propos du code ASCII :

- Les codes compris entre 0 et 31 ne représentent pas des caractères, ils ne sont pas affichables. Ces codes, souvent nommés caractères de contrôles sont utilisés pour indiquer des actions comme passer à la ligne (CR, LF), émettre un bip sonore (BEL), etc.
- Les lettres se suivent dans l'ordre alphabétique (codes 65 à 90 pour les majuscules, 97 à 122 pour les minuscules), ce qui simplifie les comparaisons.
- On passe des majuscules aux minuscules en modifiant le 5ième bit, ce qui revient à ajouter 32 au code ASCII décimal.
- Les chiffres sont rangés dans l'ordre croissant (codes 48 à 57), et les 4 bits de poids faibles définissent la valeur en binaire du chiffre.

Table1 : le code ASCII

Décimal	Hexa	Binaire	Caractère	Décimal	Hexa	Binaire	Caractère
0	0	00000000	NUL	32	20	00100000	ESPACE
1	1	00000001		33	21	00100001	!
2	2	00000010	STX	34	22	00100010	"
3	3	00000011	ETX	35	23	00100011	#
4	4	00000100	EOT	36	24	00100100	$
5	5	00000101		37	25	00100101	%
6	6	00000110	ACK	38	26	00100110	&
7	7	00000111	BEL	39	27	00100111	'
8	8	00001000		40	28	00101000	(
9	9	00001001		41	29	00101001	)

10	A	00001010	LF	42	2A	00101010	*
11	B	00001011		43	2B	00101011	+
12	C	00001100		44	2C	00101100	,
13	D	00001101	CR	45	2D	00101101	-
14	E	00001110		46	2E	00101110	.
15	F	00001111		47	2F	00101111	/
16	10	00010000		48	30	00110000	0
17	11	00010001		49	31	00110001	1
18	12	00010010		50	32	00110010	2
19	13	00010011		51	33	00110011	3
20	14	00010100	NAK	52	34	00110100	4
21	15	00010101		53	35	00110101	5
22	16	00010110		54	36	00110110	6
23	17	00010111		55	37	00110111	7
24	18	00011000		56	38	00111000	8
25	19	00011001		57	39	00111001	9
26	1A	00011010		58	3A	00111010	:
27	1B	00011011		59	3B	00111011	;
28	1C	00011100		60	3C	00111100	<
29	1D	00011101		61	3D	00111101	=
30	1E	00011110		62	3E	00111110	>
31	1F	00011111		63	3F	00111111	?

Décimal	Hexa	Binaire	Caractère	Décimal	Hexa	Binaire	Caractère
64	40	01000000	@	96	60	01100000	`
65	41	01000001	A	97	61	01100001	a
66	42	01000010	B	98	62	01100010	b
67	43	01000011	C	99	63	01100011	c
68	44	01000100	D	100	64	01100100	d
69	45	01000101	E	101	65	01100101	e
70	46	01000110	F	102	66	01100110	f
71	47	01000111	G	103	67	01100111	g
72	48	01001000	H	104	68	01101000	h
73	49	01001001	I	105	69	01101001	i
74	4A	01001010	J	106	6A	01101010	j
75	4B	01001011	K	107	6B	01101011	k

76	4C	01001100	L	108	6C	01101100	l
77	4D	01001101	M	109	6D	01101101	m
78	4E	01001110	N	110	6E	01101110	n
79	4F	01001111	O	111	6F	01101111	o
80	50	01010000	P	112	70	01110000	p
81	51	01010001	Q	113	71	01110001	q
82	52	01010010	R	114	72	01110010	r
83	53	01010011	S	115	73	01110011	s
84	54	01010100	T	116	74	01110100	t
85	55	01010101	U	117	75	01110101	u
86	56	01010110	V	118	76	01110110	v
87	57	01010111	W	119	77	01110111	w
88	58	01011000	X	120	78	01111000	x
89	59	01011001	Y	121	79	01111001	y
90	5A	01011010	Z	122	7A	01111010	z
91	5B	01011011	[	123	7B	01111011	
92	5C	01011100	\	124	7C	01111100	|
93	5D	01011101	]	125	7D	01111101	
94	5E	01011110	^	126	7E	01111110	~
95	5F	01011111	_	127	7F	01111111	

6. Les additionneurs:

3.1. Le demi-additionneur:

Le demi-additionneur est un circuit qui réalise l'addition de deux bits. La somme s'obtient normalement sur deux digits binaires, S pour le poids faible, R pour le poids fort ou retenue. Du tableau de vérité de la figure 15, on déduit les équations :

$$S = A \oplus B = A.\bar{B} + \bar{A}.B = (A+B).(\bar{A}+\bar{B})$$
$$R = A.B$$

entrées		somme	
A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Fig.15 : Tableau de vérité

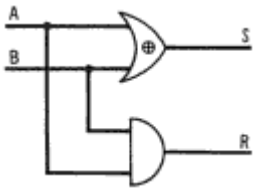


Schéma du demi-additionneur

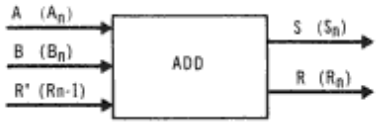
6.1. L'additionneur complet:

Le demi-additionneur correspondait à l'addition de deux bits isolés. Pour réaliser l'addition des deux nombres binaires, il faut tenir compte lors de l'addition de deux bits de rang n de la retenue de rang n-l. L'opérateur qui réalise l'addition de deux bits

binaires, en tenant compte d'une éventuelle retenue en entrée, s'appelle additionneur complet. Il possède trois entrées A. B et R' et deux sorties S et R (fig. 16).

entrées			sorties	
A	B	R'	R	S
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

tableau de vérité



symbolisation

Fig.16 : L'étage additionneur

Exercice:
Calculer

```
  10011101
+ 00111001
=
```

ce qui donne en décimale

```
  157
+  57
= 214
```

7. Les soustracteurs:

Au même titre que nous avons défini le demi-additionneur nous pouvons définir le demi-soustracteur qui retranche le digit B du digit A (fig. 17).

différence : $D = A \oplus B$
retenue : $R = \bar{A} . B$

A	B	R	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

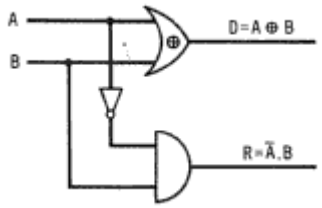


Fig.17 : Le demi-soustracteur