Chapitre1 :

Les Systèmes de Numération et de conversion

1. Quelques définitions :

DIGIT : Contraction de "digital unit" unité digitale. Un digit est un élément d'information numérique de base quelconque.
Exemple : Les nombres 1644 (base 10) et A84F (base 16) sont constitués chacun de 4 digits.

POIDS D'UN DIGIT : La valeur de chaque digit dépend de sa position. A chaque rang (position), est affecté un poids. Les positions des digits d'un nombre écrit en base B ont pour poids des puissances de B.

BIT : Contraction de "binary digit" digit binaire. Un bit ne peut prendre que deux états 0 ou 1.

Exemple : le nombre binaire 10100101 est constitué de 8 bits.

MSD : C'est le digit le plus significatif, de poids le plus fort (Most Significant Digit).

Exemple : pour le nombre A4F5, le MSB est un ….

LSD : C'est le digit le moins significatif, de poids le plus faible (Least Significant Digit).

Exemple : pour le nombre A4F5, le LSB est un ….

MOT : Un MOT est l'association (concaténation) de plusieurs digits ou bits (peut être aussi appelé courant un « nombre »)

- ⇒ Un mot de 4 bits s'appelle un quartet; exemple : 1010
- ⇒ Un mot de 8 bits s'appelle un octet; exemple : 1011 0110

BASE : Un nombre est écrit en base **B**, chacun de ses digits peut être écrit avec **B** symboles différents :
Valeurs en base 10 [*10 symboles*]: 0 1 2 3 4 5 6 7 8 9.
Symboles en base 16 [*16 symboles*]: 0 1 2 3 4 5 6 7 8 9 A B C D E F.

CAPACITE DE COMPTAGE : Avec N digits écrits en base B, on peut compter de 0 à $\mathbf{B^N}$-1, soit $\mathbf{B^N}$ nombres différents.
Exemple1 : avec un nombre de **3** digits en base **10,** on peut compter de 0 à 999 (**=10^3**), soit nombres différents.
Exemple2: avec un nombre de 4 digits en base 2, on peut compter de 0 à 1111 (**= 2^4),** Soit nombres différents.

2. Expression générale d'un nombre entier positif N (décomposition) :

De nombreux systèmes de numération sont utilisés en électronique numérique. Les plus courants sont les systèmes de numération suivants :

- Binaire (Base 2)
- Décimal (Base 10)
- Hexadécimal (Base 16)

2.1. Le système de numération « Décimal »

Le système de numération que nous employons couramment utilise **10** chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
On l'appelle pour cela "système décimal" ou système à base **10**.
Dans ce système, un nombre peut être décomposé en puissance de **10**.
Par exemple décomposons le nombre 546 :
546 = …….

- Le digit « 6 », situé au premier rang à partir de la droite a une valeur de 6
- Le digit « 4 », situé au deuxième rang a une valeur de 40.
- Le digit « 5 », situé au troisième rang a une valeur de 500.

Ainsi, chaque digit a un "poids" différent selon son rang :

- au premier rang **(rang de niveau 0)** : le poids est de 1 (ou 10^0),
- au deuxième rang **(rang de niveau 1)** : le poids est de 10 (ou 10^1),
- et au troisième rang **(rang de niveau 2)** le poids est de 100 (ou 10^2).

Le poids est la puissance nième de 10 (10^n) si on numérote les rangs de droite à gauche et en commençant par le **rang n° 0.**

Une multiplication (ou une division) par 10 consiste à déplacer le point (ou la virgule) d'une position vers la droite (ou vers la gauche). C'est d'ailleurs vrai pour toutes les bases et on peut le vérifier sur la relation (I.1).

Règles d'addition : On additionne 2 chiffres. Si le résultat est supérieur ou égal à B, on remplace le résultat R par R-B et on propage une retenue vers les chiffres de poids immédiatement supérieur.

Le système décimal est le système actuellement utilisé pour manipuler les nombres dans la vie de tous les jours.

	1	1	
		5	7
+		6	9
1		12	16
		-10	-10
	1	2	6

2.2. Généralisation : Décomposition d'un nombre

De façon générale, un nombre N quelconque peut être représenté dans une base B (système de numération en base B) par la juxtaposition suivante : $b_n b_{n-1} \ldots b_1 b_0, b_{-1} b_{-2} \ldots b_{-m}$

Les chiffres b_i sont tels que $0 \leq b_i \leq B-1$. On dit que b_i est le chiffre de *poids i*. La virgule sépare la *partie entière* (située à gauche) de la *partie fractionnaire* (située à droite). De façon générale, la relation suivante est vérifiée, quelle que soit la base utilisée :

$$N = b_n B^n + b_{n-1} B^{n-1} + \cdots + b_0 B^0 + b_{-1} B^{-1} + \cdots + b_{-m} B^{-m} \qquad \text{(I.1)}$$

Où **B est la base**, b est le chiffre de rang **n** et **n** représente le poids. Dans la **base B**, on a besoin de **B symboles** pour écrire tous les nombres.

Elle permet d'ailleurs de déterminer la représentation d'un nombre dans une base A lorsqu'on connaît sa représentation dans une base B. Il suffit de représenter les b_i et B dans la base A, d'effectuer les produits et les sommes correspondantes dans l'arithmétique de la base A et on obtient ainsi directement la représentation du nombre N dans la base A.

Exemple : On cherche la représentation en base 10 du nombre décimal 15,73. Il s'agit en fait d'une simple vérification de la formule.

$$N_{10} = 1 \,.\, 10^1 + 5 \,.\, 10^0 + 7 \,.\, 10^{-1} + 3 \,.\, 10^{-2} = 10 + 5 + 0{,}7 + 0{,}03 = 15{,}73$$

On cherche la représentation en base 10 du nombre 263,2 représenté en base 8.

$$N_{10} = 2 \,.\, 8^2 + 6 \,.\, 8^1 + 3 \,.\, 8^0 + 2 \,.\, 8^{-1} = 128 + 48 + 3 + 0{,}25 = 179{,}25$$

Remarque : Une partie fractionnaire exacte dans une base B ne conduit pas forcement à une partie fractionnaire exacte dans une base A.

Exemple : $(0{,}1)_9 = (?)_{10} = 1 \,.\, 9^{-1} = \frac{1}{9} = 0{,}111111\ldots$

Ainsi la représentation globale de N sera obtenue par juxtaposition de la représentation de la partie entière et de la représentation de la partie fractionnaire. En pratique, si on cherche par exemple la représentation en base B de $(15.73)_{10}$, on cherchera la représentation de 15 puis celle 0,73 et on juxtaposera les résultats obtenus comme le montre la figure I.1.

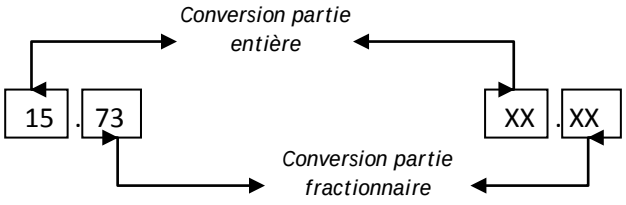


Figure I.1 : Illustration d'une conversion

2.3. Les autres bases de numération utilisées

A la place du décimal, nous pouvons utiliser la numération binaire, octale ou l'hexadécimale :

- **La base 2** (binaire) est employée pour traduire les états d'un système logique [0 ou 1, tout ou rien, juste ou faux…]
- **La base 8** (octal) autrefois très utilisée, elle tend aujourd'hui à disparaître au profit de la base 16 suite à l'évolution technologique des composants (16 bits et +)
- **La base 16** (hexadécimal) est apparue avec la logique microprogrammée et les microprocesseurs. Elle permet de traduire plus facilement un nombre binaire et autorise une représentation plus conviviale des grands nombres.
- La base 10 (décimal) est universellement employée par l'homme depuis qu'il sait compter sur ses doigts (10 doigts…)

2.3.1. Système binaire

Base : 2 *Chiffres binaires :* 0, 1

Chaque information traitée par l'ordinateur est définie par un ensemble fini d'informations simples à deux états possibles, représentées par un chiffre 1 ou 0 : des bits (BIT : Binary digIT).

On peut réaliser une conversion binaire décimale par application directe de la formule (I.1).

$$(101{,}11)_2 = 1.2^2 + 0.2^1 + 1.2^0 + 1.2^{-1} + 1.2^{-2}$$
$$= 4 + 0 + 1 + 0.5 + 0.25 = (5.75)_{10}$$

Règles d'addition : elles sont les mêmes qu'en base 10 mais on raisonne ici avec B=2.

```
 1  1  1
    1  0  1  1
+   1  1  1  0
 ---------------
    3  2  2  1
   -2 -2 -2
 ---------------
 1  1  0  0  1
```

Le système binaire est utilisé à l'intérieur des calculateurs et des systèmes logiques pour représenter les nombres. En effet, on peut faire fonctionner un transistor élémentaire dans 2 états stables bien distincts : bloqué ou saturé. Ainsi, l'état du transistor peut être représenté par un chiffre binaire, par exemple 0 s'il est bloqué et 1 s'il est saturé. Une association de transistors permet de représenter plusieurs chiffres binaires, c'est-à-dire un nombre.

- Avec un bit, on a deux valeurs possibles : 0 et 1,
- Avec 2 bits, on a 4 valeurs possibles : 00, 10, 10 et 11,
- Avec 3 bits, on a 8 valeurs possibles : 000, 001, 010, 011, 100, 101, 110, et 111.

2.3.2. Système octal et hexadécimal

Base : 8 Chiffres octaux : 0, 1, 2, 3, 4, 5, 6, 7
Base : 16 Chiffres hexadécimaux : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Les équivalents décimaux des lettres A, B, C, E, F sont respectivement 10, 11, 12, 13, 14, 15. Les systèmes octal et hexadécimal sont essentiellement utilisés pour manipuler les représentations binaires sous une forme plus condensée. Nous allons montrer sur un exemple comment chercher la représentation octale d'un nombre binaire.

Exemple : $\frac{1}{1}\ \frac{101}{5}\ \frac{011}{3}\ .\frac{010}{2}\ \frac{1}{4}\quad \frac{base\ 2}{base\ 8}$

On effectue des regroupements de 3 bits car $8=2^3$. Si on veut réaliser une conversion binaire-hexadécimal, il faut foire de groupements de 4 bits puise $16=2^4$. Attention, les b_i sont des chiffres hexadécimaux, c'est-à-dire que si l'équivalent décimal est compris entre 10 et 15, on utilise les lettres A à F.

Exemple : $\frac{110}{6}\ \frac{1011}{B}\ \frac{0101}{5}\ .\frac{0100}{4}\quad \frac{base\ 2}{base\ 16}$

Le passage de la base 16 (ou 8) vers la base 2 est immédiat. On remplace chaque chiffre hexadécimal (ou octal) par son équivalent binaire.

Règle d'addition : on fait rarement des opérations en octal, un peu plus souvent en hexadécimal. De façon générale, on effectue mentalement la conversion des chiffres et l'opération en décimal, on retranche 8 ou 16 au résultat si nécessaire et on retranscrit le résultat en octal ou hexadécimal.

```
 1     1
   3   3   7
 + 6   2   3
 -------------
   9   6  10
  -8       -8
 -------------
 1 1   6   2    Base 8
```

```
   1
   3   A   6
 + B   D   4
 -------------
   4  10  10
 + 11  13
 -------------
  15  23
     -16
 -------------
   F   7   A    Base 16
```

Récapitulatif des bases :

Base	Système	Nbre. de Symboles	Symboles utilisés															
2	Binaire	2	0	1														
8	Octal	8	0	1	2	3	4	5	6	7								
10	Décimal	10	0	1	2	3	4	5	6	7	8	9						
16	Hexadécimal	16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Tableau de correspondance entre les différentes bases pour les premières valeurs

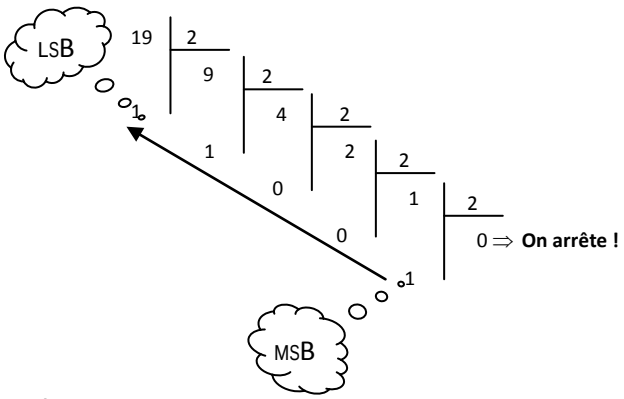
DECIMAL	BINAIRE	OCTAL	HEXADECIMAL
00	0000 0000	00	00
01	0000 0001	01	01
02	0000 0010	02	02
03	0000 0011	03	03
04	0000 0100	04	04
05	0000 0101	05	05
06	0000 0110	06	06
07	0000 0111	07	07
08	0000 1000	10	08
09	0000 1001	11	09
10	0000 1010	12	0A
11	0000 1011	13	0B
12	0000 1100	14	0C
13	0000 1101	15	0D
14	0000 1110	16	0E
15	0000 1111	17	0F
16	0001 0000	20	10
17	0001 0001	21	11
18	0001 0010	22	12

3. Conversion d'une base dans une autre (transcodage)

3.1. Conversion d'un nombre en décimal vers son équivalent en binaire [$(N)_{10}$ -> $(N)_2$]

La méthode consiste à répéter la division par 2 du nombre décimal à convertir et au report des restes jusqu'à ce que le quotient soit 0. Le nombre binaire résultant s'obtient en écrivant le premier reste à la position du bit de poids le plus faible (LSB = Least Significant Bit) et le dernier à la position du bit de poids le plus fort (MSB = Most Significant Bit).

Exemple : conversion du nombre décimal 19 en binaire $(19)_{10} = (.....)_2$?



D'où $(19)_{10}$ = (**10011**)$_2$

Exercice : Quel est le code binaire correspondant à $(65)_{10}$ et $(41)_{10}$?
$(65)_{10}$ = ( )$_2$
$(41)_{10}$ = ( )$_2$

3.2. Conversion d'un nombre en binaire vers son équivalent en décimal [$(N)_2$ -> $(N)_{10}$]

Il s'agit ici d'appliquer la formule donné au paragraphe 2.2 en prenant B= 2.

$(\mathbf{1101})_2 = \mathbf{1}x2^3 + \mathbf{1}x2^2 + \mathbf{0}x2^1 + \mathbf{1}x2^0$
$= \mathbf{1}x8 + \mathbf{1}x4 + \mathbf{0}x2 + \mathbf{1}x1 = (13)_{10}$

Table des puissances

Puissance de 2	Puissance de 8	Puissance de 10	Puissance de 16
$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 8$ $2^4 = 16$ $2^5 = 32$ $2^6 = 64$ $2^7 = 128$ $2^8 = 256$ $2^9 = 512$ $2^{10} = 1024$ $2^{11} = 2048$ $2^{12} = 4096$	$8^0 = 1$ $8^1 = 8$ $8^2 = 64$ $8^3 = 512$ $8^4 = 4096$ $8^5 = 32768$	$10^0 = 1$ $10^1 = 10$ $10^2 = 100$ $10^3 = 1000$ $10^4 = 10000$	$16^0 = 1$ $16^1 = 16$ $16^2 = 256$ $16^3 = 4096$ $16^4 = 65536$

3.3. Conversion d'un nombre en décimal vers son équivalent en octal ou hexadécimal [$(N)_{10}$ -> $(N)_8$ ou $(N)_{16}$]

Il s'agit ici d'appliquer la même méthode que pour le passage du décimal vers le binaire (§ 3.1) en divisant successivement le nombre décimal par 8 (conversion en octal) ou par 16 (conversion en hexadécimal).

Exercice : Convertir les nombres suivants :
$(1028)_{10}$ = ( )$_8$
$(61)_{10}$ = ( )$_{16}$
$(4095)_{10}$ = ( )$_8$
$(2748)_{10}$ = ( )$_{16}$

3.4. Conversion d'un nombre en octal ou hexadécimal vers son équivalent en décimal [$(N)_8$ -> $(N)_{10}$ ou $(N)_{16}$ -> $(N)_{10}$]

Il s'agit ici d'appliquer la même méthode que celle pour le passage d'un nombre binaire en décimal (§ 3.2), avec dans la formule du § 2.2 respectivement B=8 ou B=16.

Exercice : Convertir les nombres suivants :

$(1027)_8$ = ( )$_{10}$
$(61F)_{16}$ = ( )$_{10}$

3.5. Récapitulatif méthode à employer pour le transcodage

Base de départ	Base d'arrivée	Méthode de transcodage
Décimal	Binaire	Méthode de la division par 2 du nombre
	Octal	Méthode de la division par 8 du nombre
	Hexadécimal	Méthode de la division par 16 du nombre
Binaire	Décimal	$(N)_{10}= a_n \times B^n + a_{n-1} \times B^{n-1} + \ldots\ldots + a_1 \times B^1 + a_0 \times B^0$ avec B=2
Octal		$(N)_{10}= a_n \times B^n + a_{n-1} \times B^{n-1} + \ldots\ldots + a_1 \times B^1 + a_0 \times B^0$ avec B=8
Hexadécimal		$(N)_{10}= a_n \times B^n + a_{n-1} \times B^{n-1} + \ldots\ldots + a_1 \times B^1 + a_0 \times B^0$ avec B=16

3.6. Conversions directes binaire <=> hexadécimal

La conversion de la base 2 à la base 16 (et inversement) se fait aisément, la base 16 étant un multiple entier de la base 2. Elle permet de représenter sous une forme réduite un nombre binaire.
$2^4 = 16$ => un groupe binaire de 4 bits est **transcodable** directement en un digit hexadécimal.

Méthode : 0n divise le nombre binaire en tranches de 4 bits (à partir du LSB). Chacun des quartets est ensuite converti en un digit hexadécimal par simple sommation pondérée.

Exemple 1 :

$2^3\ 2^2\ 2^1\ 2^0 \quad 2^3\ 2^2\ 2^1\ 2^0 \quad 2^3\ 2^2\ 2^1\ 2^0$

$(111000110101)_2 =$ 1 1 1 0 | 0 0 1 1 | 0 1 0 1

$(1110\ 0011\ 0101)_2 =$ **(E 3 5)$_{16}$**

Exemple 2 :

$(D4C7)_{16} =$ D 4 C 7

1101 0100 1100 0111 = $(1101\ 0100\ 1100\ 0111)_2$

3.7. Conversion Décimal <=> code BCD (Décimal Codé Binaire)

Le code BCD est utilisé pour les afficheurs lumineux, son principe repose sur le codage de chaque digit décimal (chiffre) en son équivalent en binaire sur 4 bits (et inversement).

Exemple :

$(1\ 2\ 7)_{10} = (0001\ 0010\ 0111)_{BCD}$

Exercice : Effectuer les transcodages suivants :

$(5\ 7\ 6)_{10}$ = ( )$_{BCD}$

$(9\ 9)_{10}$ = ( )$_{BCD}$

$(1000\ 0011\ 0110)_{BCD}$ = ( )$_{10}$

Exercice : Combien faut-il de bits pour représenter un nombre décimal de 5 chiffres dans le code BCD ?

3.8. Conversion des nombres fractionnaires

Les nombres fractionnaires sont les nombres qui comportent une partie décimale (après la virgule) non nulle. Dans la machine, tout nombre est codé avec un nombre fini de chiffres ; il n'est pas possible de représenter tout les rationnels et fortiori tous les réels.
Pour la partie purement fractionnaire, le passage se fait par l'addition de puissances négative de B.

$$N_B = (a_n a_{n-1} \dots a_0 a_1, a_{-1} a_{-2} \dots a_{-m})_B$$
$$= a_n B^n + \cdots a_1 B^1 + a_0 + a_{-1} B^{-1} + a_{-2} B^{-2} + \cdots + a_{-m} B^{-m}$$

Exemple:

Conversion de $(1.01)_2$ en base 10:

$(1.01)_2 = 1*2^0 + 0*2^{-1} + 1*2^{-2} = (1.25)_{10}$

La conversion se fait, ici par des multiplications successives par la base des nombres purement fractionnaires. Cet algorithme doit s'arrêter des qu'on obtient une partie fractionnaire nulle ou bien quand le nombre de bits obtenus correspond à la partie fractionnaire, cherché s'obtient en lisant les parties entières de la première vers la dernière obtenue.

Exemple : $(14.2)_{10} = (?)_2$

On prend la partie fractionnaire.

0.2 *2 = 0.4 = [0] + 0.4

0.4 *2 = 0.8 = [0] + 0.8

0.8 *2 = 1.6 = [1] + 0.6

0.6 *2 = 1.2 = [1] + 0.2

0.2 * 2= … boucle

$(14)_{10} = (1110)_2$

et

$(0.2)_{10} = (00110011\dots)_2$

$(14.2)_{10} = (1110.00110011\dots)_2$

4. Représentation d'un nombre signé

4.1. Dans le système binaire

Nous savons qu'avec un mot de n bits nous pouvons représenter un entier dont la valeur est comprise entre 0 et 2^n-1. Le complémentaire d'un mot de n bits est obtenu en prenant le complément de chacun des n bits. Ainsi, si nous sommons un nombre et sont complément nous obtenons un mot dont tous les bits sont à 1. C'est-à-dire :

$$A + \bar{A} = 2^n - 1$$

Nous pouvons encore écrire :

$$-A = \bar{A} + 1 - 2^n$$

Mais sur n bits l'entier 2^n est identique à 0 : $2^n \equiv 0$

C'est-à-dire qu'il est possible d'écrire un nombre entier négatif comme le ``*complément à 2''* de sa valeur absolue :

$$-A = \bar{A} + 1$$

Nous reviendrons sur les divers codages des entiers signés plus tard. Nous pouvons utiliser cette propriété pour écrire la soustraction de deux mots de n bits sous la forme suivante :

$$A - B = A + \bar{B} + 1 - 2^n \equiv A + \bar{B} + 1 = A + (CP\ à\ 2)\ de\ B\ \ sur\ (n\ bits)$$

Avec ce mode de représentation, les soustractions se ramènent à des additions. On représente un nombre *positif* en ajoutant devant l'amplitude *un bit de signe égal à 0.*

Exemple : 5.5 : 101.1

+5.5 : **0**101.1

On a mis le bit de signe en caractère gras pour le différencier des autres. On représente un nombre négatif en prenant le complément à 2 (Cp à 2) du nombre positif correspondant. On obtient alors un bit de signe égal à 1 (qui indique qu'on a affaire à un nombre négatif). Pour obtenir le complément à 2, on prend le complément à 1 (Cp à1) et on ajoute un 1 au bit de poids le plus faible (avec propagation des retenues). Pour obtenir le Cp à 1 d'un nombre, on remplace chaque bit par son complément, c'est-à-dire qu'un 0 est remplacé par un 1, et un 1 par 0.

Exemple :

+ 5.5 :	**0**	101.1	
- 5.5 :	**1**	010.0	Cp à 1 de +5.5
		+ 1	
	1	010.1	Cp à 2 de +5.5

Exemple :

+ 5 :	**0**	101	
- 5 :	**1**	010	Cp à 1 de +5
		+ 1	
	1	011	Cp à 2 de +5

En résumé, pour représenter un nombre signé N,

a) *On convertit son amplitude $|N|$ en binaire dans le format proposé,*
b) *On ajoute un bit de signe égal à 0 pour représenter un nombre positif $+|N|$,*
c) *On prend le complément à deux du nombre positif pour représenter le nombre négatif $-|N|$.*

En conséquence, on additionne les deux nombres, bits de signe inclus, avec propagation des retenues et on ignore la dernière retenue après addition des bits de signe.

+ 13	**0**	01101	- 13	**1**	10011	+ 11	**0**	01011
+11	**0**	01011	+11	**0**	01011	-13	**1**	10011
+24	**0**	11000	-2	**1**	11110	-2	**1**	11110

-13	**1**	10011	+ 13	**0**	01101	-11	**1**	10101
-11	**1**	10101	-11	**1**	10101	+13	**0**	01101
-24	**1**	01000	+2	**0**	00010	+2	**0**	00010

Pour obtenir l'équivalent décimal d'un nombre binaire signé, on applique la relation (I.1) de définition, s'il s'agit d'un *nombre positif.*

Ainsi, l'équivalent décimal de **0** 101.1 est égal à $1.2^2+1.2^0+1.2^{-1}=+5.5$
S'il s'agit d'un *nombre négatif*, on a 2 possibilités :

- Prendre le Cp à 2 pour obtenir le nombre positif correspondant (c'est-à-dire la valeur absolue), calculer l'équivalent décimal et le faire précéder du signe moins.

Exemple :	**1**	01000	N
	0	10111	Cp à 1 de N
		+ 1	
	0	11000	Cp à 2 de N = - N

$-N=1.2^4+1.2^3=16+8=24$ d'ou $N=-24$

- Appliquer la relation (I.1) de définition mais en affectant le bit de signe d'un poids égal à -2^n, n étant le rang occupé par le bit de signe.

Si le format général de N est le suivant, $\boldsymbol{s_n}$représentant le bit de signe : $N = \boldsymbol{s_n} a_{n-1} a_{n-2} \dots a_0, a_{-1} \dots a_{-m}$
L'équivalent décimal est donné par la relation (I.2) :

$$N = -2^n . \boldsymbol{s_n} + a_{n-1} . 2^{n-1} + \cdots a_{-m} . 2^{-m} \quad (I.2)$$

$$ou \quad N = -2^n . s_n + \sum_i a_i 2^i$$

$$\boldsymbol{Exemple}: \quad 1\ 01000 = -2^5+2^3 = -32+8 = -24$$

La relation (I.2) peut aussi s'appliquer aux nombres positifs car le bit de signe est alors égal à 0 on retrouve bien l'équation (I.1). Elle permet également de calculer rapidement le plus

grand nombre positif et le plus grand nombre négatif représentables dans un format déterminé.

Remarque importante :

Lorsqu'on additionne deux nombres signés, le résultat peut "tomber" en dehors de la plage des nombres représentables. Si on additionne par exemple 24 et 13, les deux nombres étant représentés sur 5 bits plus un bit de signe, on obtient **1**00101. Le résultat est manifestement erroné puisque le bit de signe est égal à 1. En fait, le résultat est correct est égal 37, c'est-à-dire **0**100101. On constate effectivement qu'il n'est pas représentable sur 6 bits.

Lorsque cette erreur apparaît, la retenue C_{n-1}avant addition des bits de signe et la retenue C_n après addition des bits de signe sont complémentaire. L'indicateur Overflow, qui indique ce type d'erreur, est égal à 0 lorsque les deux retenues sont identiques à 1 lorsqu'elles sont différentes. On l'obtient en réalisant la fonction ou exclusif (cf. chapitre 2) entre C_{n-1} et C_n. On peut donc détecter facilement cette situation et recommencer l'opération en représentant les données avec un nombre de bits plus important.

Voyons comment modifier le format d'un nombre quand on connaît déjà sa représentation dans un format déterminé. Représentons les nombres +3 et -3 dans les formats suivants :

Signe	Partie entière	Partie fractionnaire	Représentation du nombre	
			+3	-3
1 bit	2 bits	0 bit	**0**11	**1**01
1 bit	3 bits	1 bit	**0**011.0	**1**101.0
1 bit	4 bits	2 bits	**0**0011.00	**1**1101.0

On constate facilement qu'il suffit de recopier le bit de signe pour réaliser une extension du côté des bits de poids fort et d'ajouter des 0 pour réaliser une extension du côté des bits de poids faible. On peut retrouver facilement cette règle à partir de l'équation (I.2).

4.2. Problème de format

On constate aisément que le nombre de bits nécessaires pour représenter un nombre donné augmente très rapidement avec ce nombre. Il faut par exemple 5 bits pour représenter les nombres entiers compris entre -16 et +15 et 20 bits pour représenter ceux qui sont compris approximativement entre -500.000 et + 500.000.

On voit immédiatement qu'on va être obligé de manipuler des nombres ayant des longueurs très différentes si on ne trouve pas un autre mode de représentation. La solution consiste à utiliser des formats fixes comportant une mantisse et un exposant :

$$n = m.B^e \; ; \begin{cases} m: \text{mantisse} \\ B: \text{base } (2, 8, 10, 16, \ldots) \\ e: \text{exposant} \end{cases}$$

4.2.1. Dans le système décimal

Cette technique est déjà utilisée en décimal dans la notation dite scientifique (ou ingénieur) : $n=m.10^e$
Par exemple, le nombre +.535E+05 (qui correspond à 53 500) comporte les éléments suivants :

- *La mantisse* : +.535 dont le premier chiffre doit être obligatoirement différent de zéro.
- *La lettre E* qui sépare la mantisse de l'exposant
- *L'exposant* : +05

Si on veut augmenter la précision, il suffit d'augmenter le nombre de chiffres significatifs de la mantisse. Si on passe à 3 chiffres, on pourra utiliser les plages suivantes :

[-.9999E+99, -.1000E-99] et [+.1000E-99, +.9999E+99].

Si on veut augmenter la plage des nombres représentables, il suffit d'augmenter le nombre de chiffres significatifs de l'exposant. Si on passe à trois chiffres au lieu de 2, les plages deviennent les suivantes :

[-.999E+999, -.100E-999] et [+.100E-999, +.999E+999].

4.2.2. Dans le système binaire

On utilise la même technique que dans le système décimal, mais on travaille en base 2. La norme flottante IEEE 754 définit un format standard qu'on retrouve dans pratiquement tous les ordinateurs depuis 1980. Elle a largement amélioré la simplicité du portage des programmes flottants et la qualité de l'arithmétique des ordinateurs.
Un nombre flottant est stocké selon la norme IEEE 754 d'une façon analogue à la représentation scientifique des nombres fractionnaires (tel que $2.5\ 10^{-5}$).

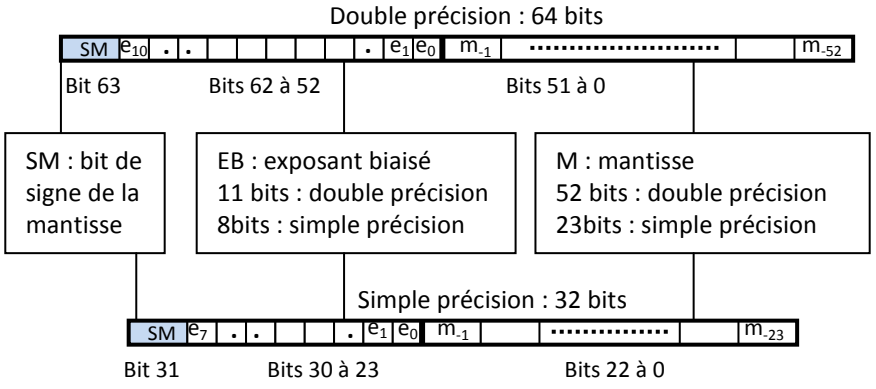


***Figure I.3** : Représentation en virgule flottante*

SM représente le signe de N. SM est égal à 0 si le nombre N est positif ou nul, et SM est égal à 1 si le nombre N est négative.
$(e_7 e_6 \dots e_1 e_0)$ est un nombre non signé e, avec 0<e<255 si le nombre flottant N est normalisé.
L'exposant est en réalité représenté avec un format de type excédent 127 (ou excédent 1023 en double précision), c'est-à-dire que si E représente la valeur réelle de l'exposant, on a les relations suivantes :

$$e = E + 127 \quad \text{ou} \quad E=e-127$$

Dans ce mode de représentation qui ne comporte pas de "signe visible", un exposant positif est toujours supérieur à un exposant négatif, ce qui facilite les opérations de comparaison de deux nombres. Par exemple, l'exposant -35 sera représenté par +92 et l'exposant +35 par +162. La valeur réelle de l'exposant peut donc varier de -126 (si e est égal à 1) à +127 (si e est égal à 254).
$m_{-1} m_{-2} \dots m_{-22} m_{-23}$ représente la partie fractionnaire de la mantisse non signé. La norme IEEE 754 précise que la partie entière est toujours égale à 1 pour les nombres normalisés et en conséquence elle n'est pas représentée. Ainsi, la mantisse M comporte en réalité 24 bits en simple précision :

$$1, m_{-1} m_{-2} \dots m_{-22} m_{-32}$$

et sont équivalent décimal est égal à :

$$M = 1 + m_{-1}.2^{-1} + m_{-2}2^{-2} + \cdots + m_{-22}2^{-22} + m_{-23}2^{-23}$$

En conséquence, l'équivalent décimal de N est égal à :

$$N = (-1)^s . M . 2^E = (-1)^s . (1 + m) . 2^{e-127}$$

$$avec \ m = \sum_{i=-1}^{i=-23} m_i . 2^i \quad et \quad e = \sum_{i=0}^{i=7} e_i . 2^i$$

La formule précédente n'est valable que pour les nombres normalisés. Elle ne permet pas de représenter certaines valeurs particulières comme le nombre 0 ou bien les quantités +∞ ou -∞. Les différents cas de figure prévus par la norme IEEE 754 sont récapitulés dans le tableau suivant :

Simple précision sur 32 bits		***Double précision sur 64 bits***		***Objet représenté***
e : 8 bits	*m : 23 bits*	*e : 11 bits*	*m : 52 bits*	
1 à 254	quelconque	1 à 2046	quelconque	Flottant normalisé
0	0	0	0	Nombre 0
0	≠0	0	≠0	Nombre dénormalisé
255	0	2047	0	+∞ ou -∞ (suivant s)
255	≠0	2047	≠0	NaN (Not a Number)

Exemple : Représentation de -1.75 en IEEE754

Traduire 1.75 en base 2 :

$1.75 = (1.11)_2$

Ecriture normalisée :

$(1.11)_2 = (0.111 * 2^1)_2$

EB=1 + 127 = 128.

1	10000000	11100000000000000000000
SM	EB	M

5. Représentation par les codes

5.1. Généralité

Les constructeurs de machines logiques et en particulier d'ordinateurs sont confrontés à différents problèmes du point de vue de la représentation des symboles ou des nombres.
Supposons par exemple qu'un utilisateur tape sur le clavier d'un ordinateur un nombre présenté sous la forme suivante : +.750E-03. Comme le PC ne comprend que les symboles binaires, il faudra associer à chaque caractère une combinaison binaire appelée *mot de code*. Cette opération est réalisée par le terminal de façon électronique et on la désigne sous le nom de *codage*. L'opération inverse, qui consiste à retrouver un caractère à partir de sa représentation binaire, est appelée *décodage*. Le nombre proposé sera donc représenté dans le calculateur par une suite ordonnée de mots de codes stockés dans leur ordre d'arrivée. L'ensemble de tous les symboles du clavier du terminal constitue un *code alphanumérique.*

Le code alphanumérique normalisé le plus utilisé s'appelle le code ASCII (American Standard Code for Information Interchange).

Table des caractères ASCII

Table ASCII standard (codes de caractères de 0 à 127)

000		(nul)	016	▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	–	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

On se rend compte facilement que la suite ordonnée de mots de code, qui représente le nombre +0.750E-02, n'est pas très utilisable en arithmétique binaire. En effet, ce nombre est représenté par 10 mots de code (un par caractère), c'est-à-dire par 80 bits, si on considère qu'il faut un octet pour représenter un caractère. Si l'opérateur avait tapé sur le clavier le même nombre sous la forme .0075, on au aurait utilisé seulement 5 mots de code, c'est-à-dire la moitié de ce qu'on a utilisé précédemment. Si on garde cette procédure, les nombres sont représentés par des mots de longueur variable et cela ne facilite pas du tout les opérations arithmétiques. Dans les calculateurs, on préfère représenter les nombres par des mots de longueur fixe (8 bits ou multiples de 8 bits par exemple). On est donc obligé d'établir une correspondance entre une suite ordonnée de mots de code représentant les caractères

successifs composant un nombre et un autre mot de code de longueur fixe choisi dans un ensemble susceptible de représenter toutes les valeurs numériques possibles. Cette opération de changement de représentation ou de passage d'un code à un autre code s'appelle un *transcodage*.

Il existe plusieurs types de codes susceptibles de représenter les différentes valeurs numériques. Nous avons déjà vu le système de numération binaire particulièrement bien adapté aux opérations arithmétiques. Nous verrons aussi les *codes décimaux – Codés – Binaires* (DCB en français, ou BCD Binary Coded Decimal dans la littérature anglo-saxonne), avec lesquels on représente chaque chiffre décimal par un mot de code, ce qui facilite grandement les opérations de transcodage pour les entrées/sorties. Nous parlerons également des *codes continus*, dans lesquels à chaque valeur numérique est associé un mot de code, les séquences de mots de code utilisées étant très avantageuses pour la réalisation de codeurs ou la représentation des fonctions logiques.

En général, on cherche à représenter les nombres en utilisant un nombre minimal d'éléments binaires de façon à réduire le coût et la complexité des équipements et de traitements ultérieurs. C'est une solution satisfaisante pour représenter l'information, mais pas pour la transmettre. En effet, si on envoie correctement un mot de code sur un canal de transmission, ce mot de code peut être altéré à cause des défauts de l'équipement, du bruit ou des perturbations atmosphériques rencontrées au cours de la transmission. Le mot de code reçu peut donc comporter des erreurs. On peut détecter et parfois corriger ces erreurs, supplémentaires (bits de redondance) dont la valeur est directement fonction de l'information. Les codes associés à ces éléments binaires sont appelés *codes détecteurs et/ou correcteurs d'erreurs*. Il est bien évident que le nombre de bits de redondance qu'il faut introduire est fonction croissante du nombre d'erreur qu'on désire corriger simultanément. Il faudra souvent trouver un compromis entre le taux de sécurité qu'on se fixe pour la transmission et le nombre total de bits à transmettre (bits d'information et bits de redondance).

5.2. Les codes décimaux codés binaires

5.2.1. Introduction

Les codes DCB permettent d'associer un mot de code à chaque digit décimal. Comme il faut représenter 10 digits différents (0, 1, …, 8, 9), il est nécessaire d'utiliser un code à 4 bits pour obtenir au moins 10 combinaisons différentes ($2^3<10<2^4$). En réalité, avec 4 bits, on peut avoir 16 mots de code différents et un code DCB est constituer de 10 mots de code choisis parmi les 16 possibles. Il y a C_{16}^{10} combinaisons différentes obtenues en choisissant 10 mots de code parmi les 16 et il y a P_{10}permutations possibles des 10 mots de code à l'intérieur de chaque combinaison. Le nombre total des codes DCB théoriquement concevables est donc égal à :

$$P_{10}C_{16}^{10} = 10!\ \frac{16!}{10!\,(16-10)!} = 30\ Milliards$$

C'est un nombre extrêmement important et parmi tous ces codes, on en retenu quelques uns qui offrent des caractéristiques intéressantes : ce sont les codes pondérés et les codes symétriques.

5.2.2. Les codes pondérés

Dans les codes pondérés, un poids est associé à chaque bit du mot de code. Si les poids p_1, p_2, p_3, p_4 sont associés aux bits b_1, b_2, b_3, b_4 respectivement, la valeur d du digit représenté par le mot de code $b_1b_2b_3b_4$ est donnée par :

$d=p_1b_1 + p_2b_2 + p_3b_3 + p_4b_4$.

Le code correspondant est généralement appelé code $p_1p_2p_3p_4$.

Exemple : Le mot de code 0110 de code 3321 représente le digit décimal 5.

Le code pondéré le plus connu est le code DCBN (décimal codé binaire naturel) ou code 8421 qui utilise les 10 premiers mots de code de la numération binaire naturelle. C'est le code le plus utilisé en particulier pour les entrées/sorties d'informations numériques des appareils de mesure (voltmètres numériques, fréquencemètres, générateurs…). Il est représenté sur la figure I.4.

5.2.3. Les codes symétriques

Ces codes ont été élaborés de façon à obtenir très simplement le complément à 9 d'un chiffre décimal. Ce complément à 9 peut en effet intervenir dans certaines méthodes de traitement des nombres négatifs comme nous l'avons vu au paragraphe 2.1.

Parmi les codes les plus utilisés, on peut citer le *code AIKEN* (qui est également un code pondéré 2421) et *le code Excédent 3*. Le code Excédent 3 n'est pas un code pondéré, mais se déduit facilement de la numérotation binaire naturelle en ajoutant 3 à chaque chiffre. Par exemple, dans le code excédent 3, le chiffre 4 est représenté par le mot de code 0111, qui correspond au chiffre 7, c'est-à-dire 4+3 en numération binaire naturelle. Les codes AIKEN et Excédent 3 sont représentés sur la figure I.4.

Binaire Naturel	DCBN 8421	AIKEN 2421	Excédent 3 ou XS3
0000	0	0	
0001	1	1	
0010	2	2	
0011	3	3	0
0100	4	4	1
0101	5		2
0110	6		3
0111	7		4
1000	8		5
1001	9		6
1010			7
1011		5	8
1100		6	9
1101		7	
1110		8	
1111		9	

Figure. I.4. : *Codes pondérés et symétriques*

5.2.4. Arithmétique des codes DCB

Le code DCB le plus utilisé est le code DCBN. En effet, dans certain nombre de petites machines à calculer, comme l'entrée (clavier) et l'affichage des valeurs numériques s'effectuer en décimal, il est tout naturel d'utiliser le code DCBN pour représenter les chiffres. Il est alors plus intéressant d'effectuer les opérations arithmétiques directement en DCBN plutôt qu'en binaire naturel afin d'éviter deux transcodages consécutifs (DCBN → binaire à l'entrée puis binaire → DCBN avant l'affichage).
En décimal en additionne les chiffres de même rang. Si la somme de deux chiffres est égale ou supérieure à 10 pour obtenir le bon résultat et on ajoute 1 (la retenue) au chiffre de poids supérieur.

Le résultat de l'opération va donner un nombre compris entre 0 et 19. Il faut apporter une correction seulement pour les nombres compris entre 10 et 19. La procédure est simple :

Pour retrancher 10, il suffit d'additionner 6 puis de retrancher 16 (c'est-à-dire qu'on laisse tomber le bit de poids 2^4, c'est-à-dire le cinquième bit).

Nombre à corriger	+6	-16
10 → 1010	16 → 10000	0 → 0000
19 → 10011	25 → 11001	9 → 1001

On obtient bien le résultat désiré. Ainsi dans la pratique, s'il y a une correction à apporter, on ajoute +6 (0110), on ignore le cinquième bit du résultat et on propage une retenue vers l'étage suivant.

Exemple	Milliers	Centaines	Dizaines	Unités
1100	**1**	**1**		
4386	0100	0011	1000	0110
+ 2893	+ 0010	+1000	+ 1001	+ 0011
7279	0111	1100	10001	1001
		+ **0110**	+ **0110**	
		10010	10111	
		0010	0111	
	7	2	7	9

On utilise la technique de complément à 10 de façon à transformer la soustraction en une addition décimal.

Exemple		243		**0**243	
		- 386		+**9**614	Cp à 10 de 386
	0	0010	0100	0011	
	1	0110	0001	0100	
	1	1000	0101	0111	
On obtient	**9**	8	5	7	soit -143

Exemple		243		**0**243	
		- 111		+**9**889	Cp à 10 de 386
	1	1	1		
	0	0010	0100	0011	
	1	1000	1000	1001	
	0	1011	1101	1100	
	+	0110	0110	0110	
On obtient	**0**	1	3	2	soit +132

5.3. Les codes continus

Un *code continu* est un code dans lequel deux mots de code consécutifs sont adjacents c'est-à-dire, qui ne diffèrent entre eux que par un seul bit. Si de plus le premier mot est adjacent au dernier, le code est dit code continu cyclique ou *code cyclique*. Parmi les codes continus, les plus utilisés sont les *codes réfléchis* qui offrent des perspectives intéressantes pour la simplification des fonctions logiques.

Le code réfléchi est également cyclique et on le désigne généralement sous le nom de *code GRAY*.
Le code Gray est très utilisé dans les codeurs linéaires ou rotatifs puisqu'il s'agit d'un code continu cyclique. Il est également utilisé pour la représentation des fonctions logiques par les tables de Karnaugh en vue de faciliter leur simplification ultérieure.

Valeur décimal	Binaire naturel	Binaire réfléchi (GRAY)
	ABCD	ABCD
0	0000	0 0 0 0
1	0001	0 0 0 1
2	0010	0 0 1 1
3	0011	0 0 1 0
4	0100	0 1 1 0
5	0101	0 1 1 1
6	0110	0 1 0 1
7	0111	0 1 0 0
8	1000	1 1 0 0
9	1001	1 1 0 1
10	1010	1 1 1 1
11	1011	1 1 1 0
12	1100	1 0 1 0
13	1101	1 0 1 1
14	1110	1 0 0 1
15	1111	1 0 0 0

5.4. Le Code Barre

Ce principe de codage, apparu dans les années 80, est largement utilisé sur les produits de grande consommation, car il facilite la gestion des produits.

Le marquage comporte un certain nombre de barres verticales ainsi que 13 chiffres :

- Le 1er chiffre désigne le pays d'origine : 3 = France, 4 = Allemagne, 0 = U.S.A, Canada etc. …
- Les cinq suivants sont ceux du code « fabricant », - Les six autres sont ceux du code de l'article, - Le dernier étant une clé de contrôle

Les barres représentent le codage de ces chiffres sur 7 bits, à chaque chiffre est attribué un ensemble de 7 espaces blancs ou noirs.