Chapitre I:

Systèmes de numération,

Codes et Arithmétique binaire

I.1. Systèmes de numération:

On appelle numération, la manière de représenter un nombre. On définit un système de numération par la base et les symboles utilisés.

Pour définir les systèmes de numération, nous allons commencer par étudier un système qui nous est familier: c'est le système décimal.

I.1.1. Système décimal:

La base du système décimal est 10. Elle représente le nombre de symbôles utilisés pour la représentation des nombres dans ce système. Les dix symbôles utilisés sont les chiffres de 0 à 9. Ils sont appelés **digits**.

Un nombre ***N*** peut être écrit dans le système décimal sous la forme générale suivante

$$N = (a_n\, a_{n-1} \ldots a_i \ldots a_0\, , a_{-1} \ldots a_{-m})_{10}$$

Avec

$$a_i \in \{0,1,\ldots 9\} \quad pour\ i = -m..n$$

L'addition des pondérations redonne la valeur du nombre

$$N = a_n .10^n + a_{n-1} .10^{n-1} \ldots + a_i .10^i + \ldots + a_0 .10^0 + a_{-1} .10^{-1} + \ldots + a_{-m} .10^{-m}$$

$$N = \sum_{i=-m}^{n} a_i .10^i$$

L'indice ***i*** est le rang du digit a_i, 10^i est le poids du digit a_i

Exemple:

Soit $N = (216,2)_{10}$

Digit	2	1	6	2
Rang	2	1	0	-1
Poids	10^2	10^1	10^0	10^{-1}
Pondération	200	10	6	0,2

I.1.2. Système binaire:

La base du système binaire est 2. Elle représente le nombre de symbôles utilisés pour la représentation des nombres dans ce système. Les deux symbôles utilisés sont 0 et 1. Ils sont appelés **bits**.

Un nombre ***N*** peut être écrit dans le système binaire sous la forme générale suivante

$$N = (a_n\, a_{n-1} \ldots a_i \ldots a_0\, , a_{-1} \ldots a_{-m})_2$$

Avec

$$a_i \in \{0,1\} \quad pour\ i = -m..n$$

L'addition des pondérations redonne la valeur du nombre

$$N = a_n.2^n + a_{n-1}.2^{n-1} \ldots + a_i.2^i + \ldots + a_0.2^0 + a_{-1}.2^{-1} + \ldots + a_{-m}.2^{-m}$$

$$N = \sum_{i=-m}^{n} a_i.2^i$$

L'indice ***i*** est le rang du bit a_i

2^i est le poids du bit a_i

Exemple:

Soit $N = (1101)_2$

Bit	1	1	0	1
Rang	3	2	1	0
Poids	2^3	2^2	2^1	2^0
Pondération	8	4	0	1

I.1.3. Système hexadécimal:

La base du système binaire est 16. Cela signifie qu'on dispose de 16 symbôles utilisés pour la représentation des nombres dans ce système. Ces symbôles sont les chiffres de 0 à 9 et les lettres A, B, C, D, E et F et sont appelés **signes**.

Un nombre ***N*** peut être écrit dans le système hexadécimal sous la forme générale suivante

$$N = (a_n\, a_{n-1} \ldots a_i \ldots a_0\, , a_{-1} \ldots a_{-m})_{16}$$

Avec $a_i \in \{0,1 \ldots 9, A, B, C, D, E, F\} \quad pour\ i = -m..n$

L'addition des pondérations redonne la valeur du nombre :

$N = \sum_{i=-m}^{n} a_i.16^i$ L'indice ***i*** est le rang du signe a_i ; 16^i est le poids du signe a_i

Exemple:

Soit $N = (64)_{16}$

Signe	6	4
Rang	1	0
Poids	16^1	16^0
Pondération	96	4

$N = 6 \times 16^1 + 4 \times 16^0 = (100)_{10}$

I.1.4. Système de base B quelconque:

La base du système est B. Cela signifie qu'on dispose de B symbôles utilisés pour la représentation des nombres dans ce système. Ces symbôles sont constitués par des chiffres et des.

Un nombre ***N*** peut être écrit dans le système hexadécimal sous la forme générale suivante

$$N = (a_n\, a_{n-1} \ldots a_i \ldots a_0\, , a_{-1} \ldots a_{-m})_B$$

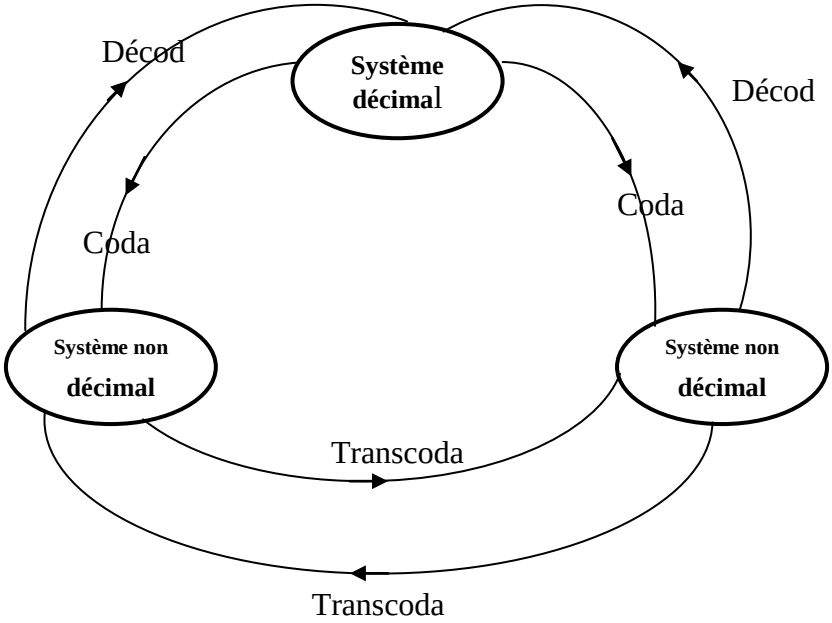
L'addition des pondérations redonne la valeur du nombre

$$N = \sum_{i=-m}^{n} a_i \,.\, B^i$$

I.2. Conversion des systèmes de numération:

Il existe trois types de conversions

- Conversion du système décimal en un autre système non décimal: cette opération s'appelle **le codage**
- Conversion d'un système non décimal au système décimal: cette opération s'appelle **le décodage**
- Conversion entre deux systèmes non décimaux: cette opération s'appelle **le transcodage**.

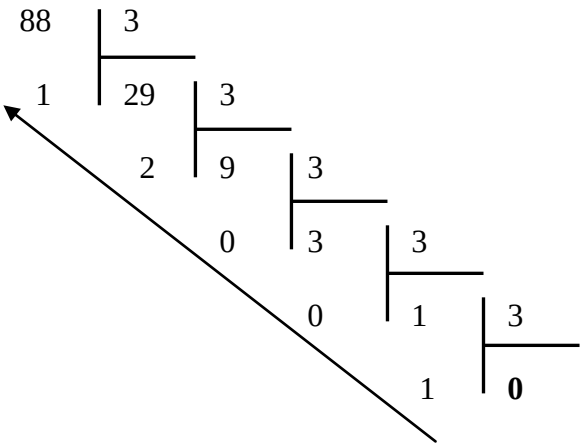


I.2.1. Codage d'un nombre:

Le codage d'un nombre N est la conversion de celui-ci du système décimal vers un système non décimal de base B.

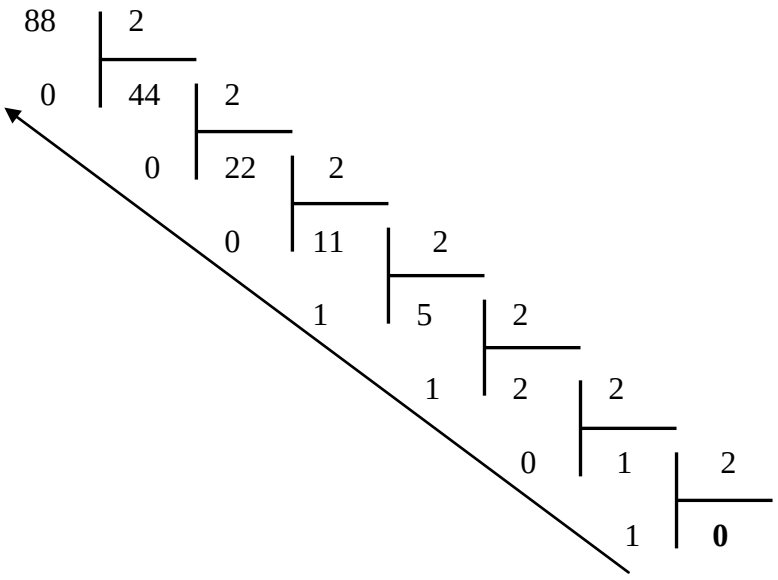
Il s'obtient en divisant successivement le nombre N par la base B jusqu'à ce que le quotient devienne nul. Le nombre recherché sera obtenu en regroupant de droite à gauche les restes successifs et en les écrivant de gauche à droite.

Exemple 1: coder le nombre 88 dans la base 3.



$(88)_{10} = (10021)_3$

Exemple 1: coder le nombre 88 dans la base 2.



$(88)_{10} = (1011000)_2$

I.2.2. Décodage d'un nombre:

Le décodage d'un nombre est l'opération inverse du codage. La somme des pondérations donne directement l'équivalent décimal du nombre.

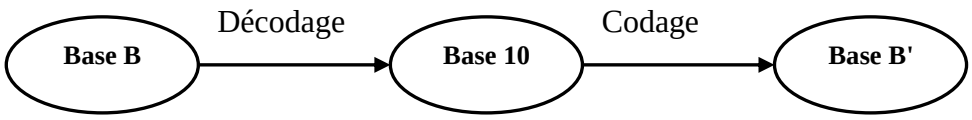
Exemple: décoder le nombre binaire $(10110)_2$

$(10110)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (22)_{10}$

$(10110)_2 = (22)_{10}$

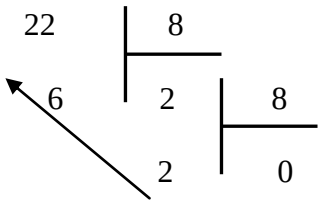
I.2.3. Transcodage d'un nombre:

Le transcodage d'un nombre est le passage entre deux systèmes non décimaux. Il s'effectue en passant par le système décimal. Cela signifie que si on veut convertir un nombre N d'une base B à une base B', on effectue le décodage puis le codage de ce nombre dans la base B'.



Exemple: Ecrire en octal le nombre binaire $(10110)_2$

$(10110)_2 = (22)_{10} = (26)_8$



Remarque:

Si on a un nombre écrit en binaire, pour le convertir en une base $B' = 2^k$, il suffit de regrouper chaque k bits ensemble en commençant de droite et en décodant chaque groupe.

Exemple: $B' = 8 = 2^3$ (k=3)

$(10110)_2 = (010110)_2 = 010 \quad 110 = (2 \quad 6)_8$

I.3. Les codes:

Coder une information c'est lui associer un symbôle ou une combinaison de symbôles qui permettent de la communiquer.

I.3.1. Codes binaire naturel, octal et hexadécimal:

Le système de numération binaire est considéré comme un code binaire naturel, il permet en effet de représenter des nombres sous forme binaire.

Pour des nombres binaires à un grand nombre de bits, la manipulation devient difficile. Pour cela, on utilise le code octal ou hexadécimal. La majorité des calculateurs utilisent en effet le codage hexadécimal pour représenter des nombres binaires.

I.3.2. Code BCD:

BCD : Binary Coded Decimal: Décimal Codé en Binaire

C'est le code le plus répandu. Dans ce code, chaque digit est représenté par une combinaison binaire de quatre bits.

- Les unités sont représentées par un quartet
- Les dizaines sont représentées par un quartet
- Les centaines sont représentées par un quartet
- Etc…

Ce code est aussi appelé code 8.4.2.1 qui sont les poids respectifs des bits de chaque quartet.

Exemple:

$(238)_{10}$ = 0010 0011 1000

I.3.3. Code binaire réfléchi ou code de GRAY:

On définit ce code de la façon suivante: à chaque augmentation d'une unité du chiffre décimal, un seul bit du nombre binaire équivalent change de valeur par rapport au nombre binaire précédant.

Le tableau ci-dessous présente le code de GRAY des nombres décimaux compris entre 0 et 9.

Décimal	Code binaire naturel	Code de GRAY
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

I.3.4. Conversion binaire naturel – binaire réfléchi:

On note par: $(B_k)_N$: le bit de rang k du nombre codé en binaire naturel;

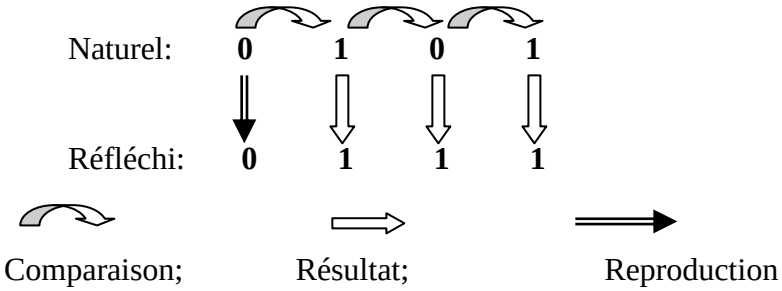
$(B_k)_R$: le bit de rang k du nombre codé en binaire réfléchi.

Le mécanisme de la conversion est basée sur la comparaison entre les bits du nombre écrit en binaire naturel telle que:

- On reproduit le bit de poids le plus fort
- Si les bits $(B_{n+1})_N$ et $(B_n)_N$ ont même valeur (0 ou 1), le bit correspondant en binaire réfléchi $(B_n)_R$ =0
- Si les bits $(B_{n+1})_N$ et $(B_n)_N$ ont des valeurs différentes, le bit correspondant en binaire réfléchi $(B_n)_R$ =1

Exemple:

Convertir en binaire réfléchi le nombre écrit en binaire naturel $(0101)_2$



I.3.5. Conversion binaire réfléchi – binaire naturel:

On note toujours par:

$(B_k)_N$: le bit de rang k du nombre codé en binaire naturel;

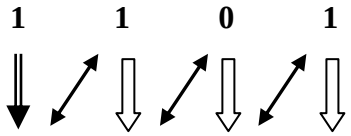
$(B_k)_R$: le bit de rang k du nombre codé en binaire réfléchi.

- On reproduit le bit de poids le plus fort
- Comparer le bit de rang n+1 du binaire naturel à celui de rang n du binaire réfléchi:
  - ❖ Si les bits $(B_{n+1})_N$ et $(B_n)_R$ ont même valeur (0 ou 1), le bit correspondant en binaire naturel $(B_n)_N$ =0
  - ❖ Si les bits $(B_{n+1})_N$ et $(B_n)_R$ ont des valeurs différentes, le bit correspondant en binaire naturel $(B_n)_N$ =1

Exemple:

Convertir en binaire naturel le nombre écrit en binaire réfléchi $(1101)_2$

Réfléchi:



Naturel: **1 0 0 1**

⟷ Comparaison; ⇨ Résultat; ⟹ Reproduction

I.3.6. Code de détection et de correction d'erreurs:

I.3.6.a. Détection d'erreurs:

Dans divers cas de transmissions de données, les informations binaires peuvent être erronées. Ce ci peut être dû à un mauvais fonctionnement d'un circuit électronique ou encore à la longue distance de transmission.

Ces erreurs doivent être détectées quand elles apparaissent au moment de la transmission des données. Une méthode simple de détection d'erreur s'appuie sur l'introduction d'un bit supplémentaire appelé bit de parité. Ce bit est transmis avec un mot numérique (groupe de bits) et aide à la détection d'erreurs possibles au cours de la transmission.

On distingue deux types de parité :

- La parité impaire génère un "1" ou un "0" de telle sorte à avoir dans l'ensemble (mot de données + bit de parité) un nombre impair de "1". Par exemple, dans le mot de données "10111011", le nombre des "1" est pair, le générateur de bit de parité impair va générer un "1".
- La parité paire génère un "1" ou un "0" de telle sorte à avoir dans l'ensemble (mot de données + bit de parité) un nombre pair de "1". Par exemple, dans le mot de données "10111011", le nombre des "1" est pair, le générateur de bit de parité impair va générer un "0".

Lorsqu'un mot est reçu, sa parité est testée (paire ou impaire: choix fait à l'émission) et il est accepté comme correct s'il satisfait le test. Ce simple test n'est pas suffisant pour localiser le bit erroné; en plus deux erreurs dans un même mot passent inaperçues.

I.3.6.b. Détection et correction d'erreurs:

Le bit de parité permet de détecter l'erreur de transmission mais ne permet pas sa correction. Pour se faire, une méthode très utilisée consiste à ajouter un mot de parité.

Considérons un mot binaire à quatre bits $\mathbf{b_3\ b_2\ b_1\ b_0}$

La parité doit être déterminée pour les trois groupes de 3 bits

P_1 détermine la parité de **b_3 b_2 b_1**

P_2 détermine la parité de **b_3 b_1 b_0**

P_3 détermine la parité de **b_2 b_1 b_0**

L'information à transmettre sera alors **P_3 P_2 P_1 b_3 b_2 b_1 b_0**

A la réception, il peut y avoir des erreurs dans n'importe quel bit parmi le mot à sept bits. Si par exemple, la parité n'est pas vérifiée pour **P_2** et **P_3** alors l'erreur s'est produite dans **b_0**.

Le tableau suivant résume les possibilités d'erreur pour différents cas de parité non vérifiée.

Parité non vérifiée	Erreur
P_2 P_3	**b_0**
P_1 P_2 P_3	**b_1**
P_1 P_3	**b_2**
P_1 P_2	**b_3**
P_3	**P_3**
P_2	**P_2**
P_1	**P_1**

Exemple:

Soit à détecter l'erreur dans un mot binaire: 1 1 1 0 1 0 0 (Parité impaire).

1 1 1 0 1 0 0

P_3 P_2 P_1 b_3 b_2 b_1 b_0

En utilisant les définitions des parités de **P_3, P_2,** et **P_1**, on trouve **P_1=0**, **P_2=1** et **P_3=0**. La parité n'est pas vérifiée pour **P_1** et **P_3**. D'après le tableau, l'erreur se trouve dans le bit **b_0**.

I.4. Arithmétique binaire:

Les diverses opérations arithmétiques qui interviennent dans les ordinateurs et les calculatrices portent sur des nombres exprimés en notation binaire. Dans cette partie, nous allons nous concentrer sur les principes de base qui nous permettent de comprendre comment les machines numériques (c'est-à-dire les ordinateurs) réalisent les opérations arithmétiques de base en essayant de montrer comment effectuer manuellement ces opérations.

I.4.1. Addition binaire:

L'addition de deux nombres binaires est parfaitement analogue à l'addition de deux nombres décimaux. En fait, l'addition binaire est plus simple puisqu'il y a moins de cas à apprendre.

On commence par additionner les chiffres du rang de poids faible, les chiffres du deuxième rang sont ensuite additionnés, et ainsi de suite. Les mêmes règles s'appliquent à l'addition binaire. Cependant, il n'y a que quatre cas, qui peuvent survenir lorsqu'on additionne deux chiffres binaires et cela quel que soit le rang. Ces quatre cas sont:

0 + 0 = 0

1 + 0 = 1

1 + 1 = 10 = 0 + report de 1 sur le rang de gauche

1 + 1 + 1 = 11 = 1 + report de 1 sur le rang de gauche

Le dernier cas ne se produit que lorsque, pour un certain rang, on additionne deux 1 plus un report de 1 provenant du rang de droite.

Exemple:

```
      1
    0 1 1 0
+   1 1 0 1
= 1 0 0 1 1
```

0110 + 1101 = 10011

Remarque:

L'addition est l'opération arithmétique la plus importante dans les systèmes numériques. Les opérations de soustraction, de multiplication et de division effectuées par les ordinateurs ne sont essentiellement que des variantes de l'opération d'addition.

I.4.2. Addition en BCD:

De nombreux ordinateurs représentent les nombres décimaux au moyen du code BCD. Rappelons que ce code fait correspondre à chaque chiffre décimal un code de 4 bits compris entre 0000 et 1001. Deux cas peuvent survenir quand on additionne deux chiffres décimaux.

I.4.2.a. La somme de deux digits est inférieure ou égale à 9:

Dans ce cas, aucune somme de deux chiffres décimaux ne dépasse 9; donc, il n'y a pas eu de reports décimaux. Dans cette situation l'addition BCD est un processus direct équivalent à l'addition binaire normale.

Exemple:

	253			**0010**	**0101**	**0011**
+	**421**		**+**	**0100**	**0010**	**0001**
=	**674**		**=**	**0110**	**0111**	**0100**
				6	**7**	**4**

I.4.2.b. La somme de deux digits est supérieure 9:

En additionnant deux digits codés en BCD, on peut avoir six représentations interdites ou non valides. Ce sont: 1010, 1011, 1100, 1101, 1110, 1111. Ces représentations n'existent pas en code BCD.

Dans un tel cas, il faut corriger la somme en additionnant 6 (0110) afin de prendre en considération le fait qu'on saute six présentations codées non valides.

Exemple:

47	**0100**	**0111**	**BCD de 47**
+ 35	**+ 0011**	**0101**	**BCD de 35**
= 82	**= 0111**	**1100**	**Somme non valide dans le 1er chiffre**
	+ 1	**0110**	**Additionner 6 pour corriger**
	= 1000	**0010**	**Somme BCD exacte**

I.4.2.c. Récapitulation de l'addition en BCD:

- Addition binaire ordinaire des représentations BCD de tous les rangs.
- Pour les rangs où la somme est égale ou inférieure à 9, aucune correction ne s'impose et la somme est une représentation BCD valide.
- Quand la somme des deux chiffres est supérieure à 9, on ajoute une correction de 0110 pour obtenir la représentation BCD exacte. Il se produit toujours un report sur le chiffre de rang immédiatement à gauche, soit lors de l'addition initiale (première étape) ou de l'addition de la correction.

I.4.3. Multiplication binaire:

On multiplie les nombres binaires de la même façon qu'on multiplie les nombres décimaux. En réalité, le processus est plus simple car les chiffres du multiplicateur sont toujours 0 ou 1, de sorte qu'on multiplie toujours par 0 ou par 1.

La multiplication binaire se base sur les quatre opérations suivantes:

- 1 x 1 = 1
- 1 x 0 = 0
- 0 x 1 = 0
- 0 x 0 = 0

Exemple:

```
        1 0 0 1          multiplicande = (9)10
      ·   1 0 1 1        multiplicateur = (11)10
-------------------
=       1 0 0 1
      1 0 0 1
    0 0 0 0
  1 0 0 1                produits partiels
-------------------
=  1 1 0 0 0 1 1         produit final = (99)10
```

I.4.4. Soustraction binaire:

La soustraction d'un nombre binaire (le diminuteur) d'un autre nombre (le diminuande) est semblable à la soustraction décimale et fait intervenir un emprunt de 1 dans le cas où un bit du diminuteur est supérieur à celui de même rang du diminuande. Cet emprunt sera ajouté au bit du rang suivant du diminuteur.

Elle repose sur les quatre opérations suivantes:

- 0 – 0 = 0
- 1 – 0 = 1
- 0 – 1 = 1 on emprunt 1
- 1 – 1 = 0

Exemple:

```
   1  0  0  1  1          19
-  0  1  1  1  0       -  14
=  0  0  1  0  1       =  05
```

I.4.5. Division binaire:

La division d'un nombre binaire (le dividende) par un autre (le diviseur) est identique à la division de deux nombres décimaux. En réalité, la division en binaire est plus simple puisque pour déterminer combien de fois le diviseur entre dans le dividende, il n'y a que 2 possibilités 0 ou 1.

Exemple:

```
 1  0  0  1  |  1  1        9/3 = 3
-   1  1     |  1  1
 0  0  1  1
-0  0  1  1
 0  0  0  0
```

```
 1  0  1  0 , 0  |  1  0  0        10/4 = 2,5
-1  0  0         |  1  0, 1
 0  0  1  0
-   0  0  0
      1  0  0
      1  0  0
      0  0  0
```