Faculté de Technologie
Département Socle Commun ST 2
Module : *Logique Combinatoire et Séquentielle*

Chapitre III : Systèmes de Numération et Codage.

I. Introduction:

Le terme numérique vient du principe de fonctionnement des ordinateurs pour exécuter diverses opérations dans des délais limités. Au début, les applications numériques se limitaient aux systèmes d'ordinateurs, aujourd'hui cette technologie est en pleine expansion et elle est employée dans une grande variété d'applications comme les systèmes de communication, la télévision, le radar et l'instrumentation médicale.

Malgré l'importance que revêt le système décimal, en raison de son utilisation universelle pour représenter les grandeurs du monde courant, le système de numération binaire est le plus important de ceux utilisés dans les circuits numériques.

Un système de Numération est un système qui permet de représenter un nombre N exprimé dans une base B quelconque (entière positive), écrit sous la forme :

$N = (a_n\ a_{n-1} \ldots a_1\ a_0\ a_{-1} \ldots a_{-m})_B$. ce qui correspond à la valeur dans la base 10: $N = \sum_{i=-m}^{n} a_i B^i$

Où : a_i : est un symbole représentant un chiffre entier de rang i, compris entre 0 et (B-1). Le poids d'un chiffre dépend de sa position dans le nombre.

Tel que : Si i=0 le chiffre correspondant est de poids le plus faible.

(LSB : abréviation de l'anglais "Less Significant Bit").

Si i=n le chiffre correspondant est de poids le plus fort.

(MSB : abréviation de l'anglais "Most Significant Bit").

Par exemple pour le système décimal, tout nombre N s'exprime à partir des dix chiffres : 0,1,2,3,.....,9, on dit alors que la base de numération est B=10.

Il faudra parfois convertir des valeurs décimales en valeurs binaires avant de pouvoir les traiter dans un circuit numérique. De même, il y aura des situations où des valeurs binaires données par un circuit numérique devront être converties en valeurs décimales pour qu'on puisse les lire (ex. la calculatrice).

Il existe deux autres systèmes de numération très répandus dans les circuits numériques. Il s'agit des systèmes de numération, octale et hexadécimal qui servent tous les deux au même but, c'est celui de constituer un outil efficace pour représenter de gros nombres binaires. Les bases des systèmes de numération qui font l'objet de ce cours sont représentées par le tableau suivant :

Système	Binaire	Octal	Décimal	Hexadécimal
Base	2	8	10	16
Chiffres	0,1	0,1,2,3,4,5,6,7	0,1,2,3,4,5,6,7,8,9	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Le présent chapitre se propose de vous montrer comment effectuer des conversions entre les différents systèmes, il se veut également une introduction à certains codes binaires utilisés pour représenter divers genres d'information. Ces codes agencent des 0 et des 1 de manière différente à celle du système binaire.

II. Système Binaire :

Dans le système binaire, tous les nombres sont exprimés à l'aide des chiffres 0 et 1. Ces deux chiffres sont appelés bits.
Le passage du système décimal au système binaire s'appelle le codage.
Le passage du système binaire au système décimal s'appelle le décodage.

II.1. Conversion Binaire-Décimal :

Pour convertir un nombre binaire en décimal, il suffit d'utiliser la base B=2 dans la relation précédente. $(N)_{10} = a_n.2^n + a_{n-1}.2^{n-1} + ... + a_0.2^0 + a_{-1}.2^{-1} + a_{-2}.2^{-2} + ... + a_{-m}.2^{-m}$

Exemple 1 : $(1\ 1\ 0\ 1\ 1)_2 = 1.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0 = (27)_{10}$.

Exemple 2 : $(1\ 0\ 1\ 1\ 0\ 1\ 0\ 1)_2 = 1.2^7 + 0.2^6 + 1.2^5 + 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0 = (181)_{10}$.

Exemple 3 : $(11.01)_2 = 1.2^1 + 1.2^0 + 0.2^{-1} + 1.2^{-2} = (3,25)_{10}$

II.2. Conversion Décimal-Binaire :

1. Conversion de la partie entière d'un nombre :

Il existe deux manières de convertir un nombre décimal en son équivalent binaire :

- **Méthode des soustractions successives :**

Cette méthode convient bien à la conversion des petits nombres. Pour utiliser cette méthode, on doit déterminer en premier lieu les valeurs successives des poids 2^i (i = 0, ……,n), on cherche ensuite le plus grand multiple contenu dans le nombre décimal à convertir et on le retranche du nombre. On recommence le processus avec le reste obtenu jusqu'à ce que l'on obtient "0" comme reste de la soustraction.
La valeur binaire correspondante sera "1" à la position des poids utilisés dans la soustraction, et "0" à la position des poids non utilisés.

Exemple1 : Convertir le nombre Décimal $(23)_{10}$ en Binaire, en utilisant la méthode des soustractions successives.

- **Méthode des divisions successives :**

La deuxième manière convient plus aux grandes valeurs. Elle consiste à diviser le nombre décimal à convertir, successivement par deux et à conserver le reste jusqu'à ce que le résultat de la division soit égal à "0".

Le nombre binaire correspondant sera la succession des restes obtenus, en commençant par le dernier, qui représentera le bit de poids le plus élevé de ce nombre.

Exemple : Convertir en Binaire le nombre décimal 26.

26 : 2 = 13 : 2 = 6 : 2 = 3 : 2 = 1 : 2 = 0

0 1 0 1 1

D'où $(26)_{10} = (11010)_2$.

2. Conversion de la partie fractionnaire :

On procède par des multiplications successives par "2" de la partie fractionnaire jusqu'à ce qu'on obtienne un nombre entier. Et à chaque multiplication, on prend en compte seulement la partie entière obtenue.

Remarque : Si après plusieurs multiplications, on ne trouve toujours pas de nombre entier, on arrête les calculs.

Exemple1 : Convertir en binaire le nombre décimal $(35,125)_{10}$.

Exemple2 : Convertir en binaire le nombre décimal 0,33.

On voit qu'après 8 multiplications on n'a toujours pas de nombre entier, si on arrête à ce stade; Le résultat trouvé est exact à 2^{-8} près.

On a alors : $(0,33)_{10} = (0.01010100)_2$.

Remarque : Très souvent un opérateur lit ou introduit des valeurs numériques dans une machine, et par conséquent la représentation binaire est assez mal commode. On utilise couramment d'autres modes de représentations d'une information binaire et qui sont les représentations: octale ; hexadécimale et enfin décimale codée binaire.

III. Relation entre la base 2 et les bases puissances de 2 ; $B=2^K$:

La règle générale ici est de faire des groupements de K bits en partant de la droite, puis convertir ces groupements au système de base $B = 2^K$. La conversion inverse se fait en convertissant chaque symbole à son équivalent binaire écrit sur K bits.

- Cas du système Octal $B = 8 = 2^3$; K=3 :

Soit le nombre binaire $(10111001)_2$, écrit en octal: $(271)_8$.

Exemple $(101001111100)_2 = (?)_8$.

Soit le nombre octal (6 1 5 3)$_8$, écrit en binaire : $(110001111011)_2$.

Exemple : (7 3 5 5 3)$_8$=(?)$_2$.

- Cas du système Hexadécimal : B=16=2^4 k=4:

Soit le nombre binaire $(1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1)_2$, écrit en hexadecimal : $(715)_{16}$.

Exemple $(1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1)_2 = (\ ?)_{16}$.

Soit le nombre hexadecimal $(\ 3\ \ 6\ \ \ 5\ \ \ B\ )_{16}$, écrit en binaire : $(11011001011011)_2$.

Exemple : $(\ A\ \ 3\ \ \ E\ \ 5\ )_{16}=(?)_2$.

- La conversion de l'hexadécimal à l'octal et inversement :

Dans ce cas, on fait deux conversions :

Hexadécimal ⟷ Binaire ⟷ Octal.

Exemples :

$(8\ B\ 3\ F)_{16} \longrightarrow (?)_8$; $(737)_8 \longrightarrow (\ ?)_{16}$

IV. Différentes représentations des nombres entiers signés :

Les systèmes numériques traitent aussi bien les nombres négatifs que ceux positifs, d'où la nécessité d'adopter une certaine convention selon les méthodes suivantes:

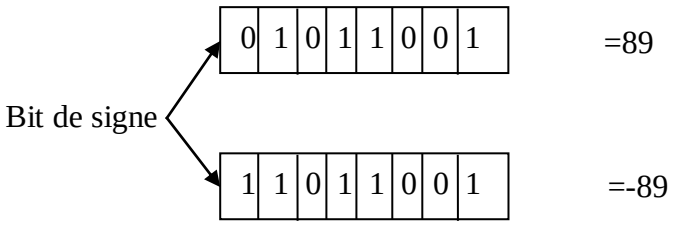
IV.1. Représentation signe-grandeur (Module et signe):

La solution la plus simple est d'ajouter un bit de signe à la représentation binaire de la valeur absolue du nombre. Ce bit a le poids le plus fort.

Si MSB = 0 ⟶ le nombre est positif.

Si MSB = 1 ⟶ le nombre est négatif.

Exemple 1 :



Avec n bits on peut représenter les nombres appartenant à la plage $-(2^{n-1}-1);+(2^{n-1}-1)$.

Exemple2 :

Représentation « signe-grandeur » des nombres binaires de 3 bits avec leurs équivalents décimaux. Le nombre zéro possède deux représentations distinctes +0 et –0.

Nombre binaire		
Signe	Valeur	Décimal
0	11	+3
0	10	+2
0	01	+1
0	00	+0
1	00	-0
1	01	-1
1	10	-2
1	11	-3

La représentation signe-grandeur est une représentation directe mais elle n'est généralement pas utilisée sur les ordinateurs à cause de la complexité qui matérialise cette notion.

L'inconvénient majeur de cette représentation est qu'il existe deux représentations différentes pour le zéro.

IV.2. Représentation en complément restreint (complément à 1):

Le complément restreint est obtenu en complémentant chacun des bits du nombre et ainsi, la somme du nombre et de son complément sera égale à 2^n-1.

Exemple :

Le complément à 1 de $(1101110)_2$ est $(0010001)_2$

1101110

0010001

1111111 ⟵ 127 ; 2^n-1 n=7

Remarque : même dans le cas du complément à 1, on a deux représentations pour le zéro.

IV.3. Représentation en complément vrai (complément à 2).

C'est la représentation la plus utilisée.

- Le complément à deux est obtenu en complémentant à 2 le premier bit non nul à partir de la droite et tous les autres à 1.
- Aussi il est égal au complément à 1 majoré de 1.

Remarques :

– Pour passer d'une valeur négative à une autre positive, on applique le complément à 2;

– une seule représentation pour le zéro ;

– avec des mots de n bits, on obtient 2^n valeurs différentes, de 0 à $2^{n-1}-1$ pour les valeurs positives, et de -1 à -2^{n-1} pour les valeurs négatives.

Exemple 1:

Représenter les nombres binaires de 3 bits en représentation complément vrai avec leurs équivalents décimaux.

Nombre binaire		Décimal
0	11	+3
0	10	+2
0	01	+1
0	00	0
1	11	-1
1	10	-2
1	01	-3
1	00	-4

Remarque : Pour les nombres positifs on remarque que le MSB=0 ; Tandis que pour les nombres négatifs le MSB=1

Exemple 2 :

+13 sur 8 bits : 00001101, -13 sur 8 bits : 11110011

V. Différentes représentations des nombres réels :

Il y a deux types de représentations :

- Représentation en virgule fixe.
- Représentation en virgule flottante.

1.Représentation en virgule fixe:

Un nombre fractionnaire en virgule fixe possède deux parties ; Partie entière et partie fractionnaire. La virgule est toujours positionnée à une position fixe entre les deux parties entière et fractionnaire.

Exemples : 01101010 = 106 ;

0110.1010 = 6,625.

01.101010 = 1,65625.

Remarques :

Cette représentation offre une utilisation limitée lorsqu'on traite des données de grandeurs différentes, car on doit prendre un grand nombre de bits de part et d'autre de la virgule, pour pouvoir représenter des grandeurs très faibles et des grandeurs très importantes.

2. Représentation en virgule flottante :

La représentation en virgule fixe ne permet pas de manipuler des nombres très grands où des nombres très petits. La représentation appropriée est celle en virgule flottante qui emploie la norme IEEE 754. Dans une première étape, on considère les nombres décimaux comme : $N = (-1)^s .M.B^E$

Avec : *s :* le signe, M: la mantisse, E: l'exposant.

On normalise ensuite cette représentation en imposant à la mantisse d'être comprise entre 1 et 2 (c'est ce qu'on fait en base 10 en imposant à la mantisse d'être entre 1 et 10).

Puisque la mantisse commence forcément par 1, on n'a pas besoin de représenter le dit 1.

Remarque : on n'a pas de représentation de 0.

(Si E>0 le nombre représenté est très grand) ;

(Si E<0 le nombre représenté est très petit).

Le standard IEEE 754 définit trois formats : les nombres en simple précision sur 32 bits, les nombres en double précision sur 64 bits, et les nombres en représentation intermédiaire sur 80 bits (utilisée par les processeurs pour minimiser les erreurs d'arrondi).

La norme internationale de représentation en virgule flottante est IEEE 754 flottant sur 32 bits est représentée par :

b_{31}	b_{30}	b_{29}	…….	b_{24}	b_{23}	b_{22}	….....	b_1	b_0

Le bit b_{31} : le bit de signe mantisse.

b_{30} ……b_{23} : L'exposant.

b_{22}…….b_0 : La mantisse.

Le bit de signe mantisse est : 1 pour une mantisse négative.

0 pour une mantisse positive.

- La mantisse vaut toujours 1,xxx et on ne stocke que xxx sur b_{22}……b_1b_0.
- L'exposant est en excédent 127.

Pour les doubles c'est pareil, sauf que l'exposant est codé sur 11 bits et la mantisse sur 52 bits, ce qui nous donne bien 64 bits avec le bit de signe.

Exemples:

1. 1 10000011 11000000000000000000000 = ?
2. 0 01111111 00000000000000000000000 = ?

VI. Les opérations arithmétiques en binaire :

VI.1. L'addition:

Elle obéit aux règles suivantes :

0+0=0 ; 1+0=1 ; 0+1=1 ; 1+1=0 avec une retenue 1.

Exemple 1 : 1101(13)+1001(9)=10110(22).

10101(21)+11100(28)=110001(49)

Exemple 2 : additionner les deux nombres hexadécimaux :

```
  1010|0001|1111    (A1F)
  1011|0010|0101    (B25)
 ---------------------  --------
1|0101|0100|0100    1544
1|  5  |  4  |  4
```

Si on fait la sommation en hexadécimal :

F+5=4 et une retenue=1 ;

1+2+1(r)=4 et une retenue =0 ;

A+B=5 (A+B=F+6=5) et une retenue =1 .

VI.2. La soustraction :

La soustraction binaire obéit aux règles suivantes :

0-0=0 ; 1-0=1 ; 0-1=1 avec un report de 1 ; 1-1=0.

Exemple 1 : 10-9

```
1010
1001
-------
0001
```

pas de report, alors le résultat est positif

Exemple 2 : 9-10

```
  1001
  1010
 -------
 11111        report =1  le nombre est négatif
```

VI.2.1. La soustraction en complément à 2 (c à 2) :

Cette méthode permet de transformer une soustraction en binaire, en une addition et de pouvoir déterminer le signe du résultat. En représentant les valeurs négatives par leur c à 2 et le bit de signe étant égale à 1. La valeur absolue du résultat s'obtient en c à 2.

Le signe : Le signe est obtenu par addition de la retenue de l'addition des bits de poids le plus fort avec la somme des bits de signe. La retenue de cette somme est ignorée.

Exemple 1 : 12-15=12+(-15)

```
(12)   0  1100
(-15)  1  0001
       1  1101      ⟶   Complément à 2 de (-3)10
```

Exemple 2 : (-8) +(-5)

```
(-8)          1  1000
(-5)          1  1011
          1   1  0011      ⟶   Complément à 2  1101 (-13)10
          ↑   à ignorée
```

VI.2.2. La soustraction en complément à 1 (c à 1):

Les nombres négatifs sont remplacés par leurs c à 1 et précédés de 1. Le signe du résultat provient de l'addition de la retenue de la somme des MSB avec les bits de signe, puis la retenue de l'opération est additionnée aux LSB du résultat de l'addition des valeurs absolues. Si le résultat est négatif on prend le c à 1 de la somme des valeurs absolues.

Exemple : $(18)_{10}$ - $(5)_{10}$

```
(18)          0   10010
(-5)          1   11010
          1 0   01100
          +  ⟶  1
              0   01101      ⟶(13)10 : valeur positive
```

$(5)_{10}$ - $(18)_{10}$

```
0  00101
1  01101
1  10010  ⟶  valeur négative, il faut la complémenter à 1
0  01101  →      (-13)10
```

VII. Le codage des nombres :

La création des codes répond à plusieurs impératifs, parmi lesquels on trouve :

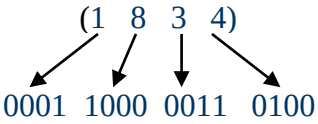
1. Celui de faciliter le traitement de l'information et de permettre dans certains cas l'économie de matériel (ex : le code excédant 3 et le code Aïken).

2. Assurer une protection de l'information contre d'éventuelles perturbations (code 'p' parmi 'n').

1. Les codes DCB :

Ce sont les codes décimaux codés binaires. Le nombre décimal est codé chiffre par chiffre et sa représentation binaire s'obtient en juxtaposant les combinaisons de quatre digits.

Exemple :



2. Code « excédent 3 » :

Ce code est très utilisé dans les systèmes traitant l'information en décimal, à cause de la simplification qu'il apporte du point de vue de la réalisation des circuits, relatifs aux opérations arithmétiques. Il est formé de la même manière que le code DCB mais on ajoute systématiquement 3 à chaque chiffre. Pour les nombres de 0 à 9 on aura ainsi :

Décimal	Code DCB	Code excédant 3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

3. Code GRAY : (Code cyclique progressif ou code réfléchi)

Dans ce code, un seul bit change de valeur lorsqu'on passe d'une combinaison à une autre (propriété d'adjacence).

Nombre	Code GRAY
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

4. Code Aïken :

Ce code ne permet que la représentation des chiffres décimaux de '0' à '9' par les 5 premiers et les 5 derniers chiffre du code binaire.

Nombre décimal	Code Aïken
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

5. Codes 'p' parmi 'n' :

Chaque combinaison de n chiffres utilisée, contient un nombre 'p' de '1' et (n-p) de '0'. Ce sont des codes pondérés.

Exemple : Code '2' parmi '5'

	Code '2' parmi '5'					
Décimal	Poids	7	4	2	1	0
0		1	1	0	0	0 (*)
1		0	0	0	1	1
2		0	0	1	0	1
3		0	0	1	1	0
4		0	1	0	0	1
5		0	1	0	1	0
6		0	1	1	0	0
7		1	0	0	0	1
8		1	0	0	1	0
9		1	0	1	0	0

(*) ne vérifie pas la pondération

Les codes 'p' parmi 'n' permettent un contrôle de parité comme pour le cas '2' parmi '5'. le nombre de 1 est pair, ce qui permet de détecter à la réception s'il y a un nombre non pair et de détecter l'erreur. Il est souvent utiliser dans les centraux téléphoniques.

VIII. Les codes alphanumériques :

Un ordinateur est plus utile lorsqu'il a la possibilité de traiter en plus de l'information numérique, l'information non numérique. De ce fait il doit reconnaître des codes qui correspondent à des nombres, des lettres, des signes de ponctuations et des caractères spéciaux. Ces codes sont appelés alphanumériques.

Le code alphanumérique le plus répandu est le code ASCII (American Standard Code for Information Interchange), prononcé "aski"; ce code est sur 7 bits ce qui permet de représenter 128 éléments codés. Il existe d'autre variantes du codes ASCII étendu sur 8 bits suivant la langue utilisée.