Chapitre 1: Systèmes de Numération et Codes

I. Introduction

Habituellement, on utilise le système décimal pour représenter les nombres, mais il est possible d'utiliser d'autres **systèmes de numération**. Nous nous intéressons dans ce chapitre aux systèmes de numération fréquemment rencontrés en technologie numérique. Il s'agit des systèmes **binaire**, **octal**, **décimal** et **hexadécimal**.

Avant de décrire ces systèmes, nous allons définir la notion de **base** d'un système de numération ainsi que le principe d'écriture d'un nombre dans un système de numération de base b quelconque.

1- Base d'un système de numération

La **base b** est définie comme étant le nombre de symboles différents utilisés pour représenter des nombres dans un système de numération de base b.

Le système décimal, par exemple, dispose de dix symboles (appelés chiffres) notés 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. **En décimal, b=10**.

2- Forme polynomiale

Tout nombre N peut être représenté dans un système de numération de base b sous la forme suivante :

$$(N)_b = a_0 b^0 + a_1 b^1 + \ldots\ldots\ldots\ldots + a_n b^n \qquad \text{avec } 0 \leq a_i \leq b-1$$

Cette forme est appelée ***forme polynomiale***

Soit l'écriture simplifiée :

$$(N)_b = (a_n \ldots\ldots a_0)_b$$

Le nombre **N** est représenté comme une séquence de symboles : $\mathbf{a_n \ldots\ldots a_0}$.

La notation $()_b$ indique que le nombre est écrit en base b. En décimal, on ne note pas d'indice.

Le poids (rang) de a_i est égal à i.

Par exemple, dans le système décimal, on écrit 154 pour représenter le nombre :

$$154 = 4.10^0 + 5.10^1 + 1.10^2$$

II- Systèmes de Numération

1. Système décimal (b = 10)

Le système décimal est composé de dix symboles (appelés chiffres) allant de 0 à 9.

Un nombre décimal s'écrit comme une séquence de chiffres : a_n …… a_0.

Le poids de chaque chiffre dépend de sa position dans le nombre décimal considéré. Il est égal à l'exposant de la base qui lui est associée dans la forme polynomiale du nombre considéré N.

Le chiffre de droite s'appelle le chiffre le moins significatif (à poids faible), celui de gauche s'appelle le chiffre le plus significatif (à poids fort). La séquence '541' et la séquence '145' ne représentent pas le même nombre décimal car les poids des chiffres ne sont pas les mêmes.

Selon la forme polynomiale, nous pouvons écrire :

$$N = \sum_{i=0}^{n} a_i 10i \text{ , où } a_i \in \{0,1,2,3,4,5,6,7,8,9\}$$

Exemple: $496 = 6 \times 10^0 + 9 \times 10^1 + 4 \times 10^2$

4 est le chiffre le plus significatif (poids : 2)

6 est le chiffre le moins significatif (poids : 0)

Remarque : Si N n'est pas entier, on traitera la partie entière de la manière indiquée ci-dessus. La partie fractionnaire sera traitée selon le même principe en utilisant les puissances négatives.

Exemple : $415{,}26 = 5 \times 10^0 + 1 \times 10^1 + 4 \times 10^2 + 2 \times 10^{-1} + 6 \times 10^{-2}$

De toute évidence, le système décimal est le plus familier de nous tous. Malheureusement, les circuits numériques ne fonctionnent pas en mode décimal, c'est à dire un mode qui nécessite dix niveaux de tension différents pour leur fonctionnement. C'est la raison pour laquelle la technologie numérique fait appel à un système qui n'utilise que deux niveaux de tension distincts. Ce système est appelé système binaire.

2. Système binaire (b = 2)

C'est le système le plus utilisé en électronique numérique. Il comprend deux symboles {0,1} appelés bits.

Un nombre binaire s'écrit de la façon suivante :

$N = (a_n \ldots\ldots\ldots\ldots a_0)_2$ où $a_i \in \{0,1\}$

a_0 est le bit le moins significatif (**LSB** : **L**ow **S**ignificant **B**it).

a_n est le bit le plus significatif (**MSB** : **M**ost **S**ignificant **B**it)

Exemple : $(101010)_2$; $(10111)_2$

Selon la forme polynomiale, nous pouvons écrire :

$$N = \sum_{i=0}^{n} a_i 2i \text{ , où } a_i \in \{0,1\}$$

Exemple : $(10111)_2 = 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4$

3. Système octal (b = 8)

Ce système dispose de huit symboles : {0,1,2,3,4,5,6,7}.

De la même manière, un nombre octal s'écrit:

$N = (a_n \ldots\ldots\ldots\ldots a_0)_7$ où $a_i \in \{0,1,2,3,4,5,6,7\}$

Exemple: $(377)_8$; $(255)_8$

Selon la forme polynomiale, nous pouvons écrire :

$$N = \sum_{i=0}^{n} a_i \bullet 8^i \text{ , où } a_i \in \{0,1,2,3,4,5,6,7\}$$

Exemple : $(2145)_8 = 5 \times 8^0 + 4 \times 8^1 + 1 \times 8^2 + 2 \times 8^3$

4. Système hexadécimal (b = 16)

Ce système dispose de 16 symboles chiffrés de 0 à 9 plus les lettres majuscules A, B, C, D, E et F qui correspondent aux valeurs allant de 10 à 15.
Un nombre hexadécimal s'écrit :

$N = (a_n \ldots\ldots\ldots\ldots a_0)_H$ où $a_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

Exemple : $(1FF)_H$; $(1A2C)_H$

Selon la forme polynomiale, un nombre hexadécimal s'écrit :

$$N = \sum_{i=0}^{n} a_i \bullet 16^i \text{ , où } a_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$$

Exemple : $(1FF)_H = F \times 16^0 + F \times 16^1 + 1 \times 16^2$

III. Conversions

1. Conversion d'un nombre écrit en base quelconque vers le décimal

L'équivalent décimal d'un nombre N écrit dans une base b quelconque s'obtient par application directe de la forme polynomiale :

$$N = \sum_i a_i b^i$$

Exemple : convertir les nombres suivants en leur équivalent décimal (la base est indiquée en indice) :

a. $(10111)_2$ **b.** $(2145)_8$ **c.** $(1FF)_H$

a. $(10111)_2$ = $1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4$
= $1 \times 1 + 1 \times 2 + 1 \times 4 + 0 \times 8 + 1 \times 16$
= $1 + 2 + 4 + 0 + 16$
= 23

b. $(2145)_8$ = $5 \times 8^0 + 4 \times 8^1 + 1 \times 8^2 + 2 \times 8^3$

$= 5 \times 1 + 4 \times 8 + 1 \times 64 + 2 \times 512$
$= 5 + 32 + 64 + 1024$
$= 1125$

c. $(1FF)_H = F \times 16^0 + F \times 16^1 + 1 \times 16^2$
$= 15 \times 1 + 15 \times 16 + 1 \times 256$
$= 15 + 240 + 256$
$= 511$

2. Conversion d'un nombre décimal en binaire

Nous avons vu, d'après la forme polynomiale, qu'un nombre binaire s'écrit :

$$N = \sum_i a_i 2i$$

Le problème revient donc à déterminer les valeurs des bits a_i. Pour cela, il existe deux méthodes.

1) Première méthode :

Il s'agit d'une répétition de divisions par 2 jusqu'à ce que le quotient soit 0. Les restes des différentes divisions correspondent aux bits a_i à déterminer. On écrit le premier reste à la position du LSB (à droite) et le dernier reste à la position du MSB (à gauche).

Exemple: Convertir le nombre décimal 65 en binaire

Division	***Quotient***	***Reste***
65/2	32	$1 \rightarrow a_0$
32/2	16	$0 \rightarrow a_1$
16/2	8	$0 \rightarrow a_2$
8/2	4	$0 \rightarrow a_3$
4/2	2	$0 \rightarrow a_4$
2/2	1	$0 \rightarrow a_5$
1/2	0	$1 \rightarrow a_6$

$$65 = (1000001)_2$$

2) Deuxième méthode :

Cette méthode consiste à écrire le nombre décimal comme une somme de puissances entières de 2. Si un terme 2^k est présent dans la somme en question, on inscrit un '1' vis à vis de sa position, sinon on y inscrit un '0'.
Cette méthode nécessite la connaissance des différentes puissances entières de 2.

Exemple: Convertir le nombre décimal 36 en binaire.

$36 = 32 + 4$ (la plus grande puissance entière de 2 contenue dans 36 est 32)

$= 2^5 + 2^2$

Notons que les termes 2^4, 2^3, 2^1, 2^0 sont absents. Nous en concluons que $a_4=a_3=a_1=a_0=0$. Donc $36 = (100100)_2$

3. Conversion d'un nombre décimal en octal

Même principe qu'avant, sauf qu'au lieu de diviser par 2, on divise par 8.

Exemple: Convertir le nombre décimal 65 en octal

Division	***Quotient***	***Reste***
65/8	8	$1 \rightarrow a_0$
8/8	1	$0 \rightarrow a_1$
1/8	0	$1 \rightarrow a_2$

$$65 = (101)_8$$

4. Conversion d'un nombre décimal en hexadécimal

De la même manière, par une suite de divisions successives par 16, on convertit un nombre décimal en hexadécimal.

Exemple: Convertir le nombre décimal 65 en hexadécimal

Division	*Quotient*	*Reste*
65/16	4	$1 \rightarrow a_0$
4/16	0	$4 \rightarrow a_1$

$$65 = (41)_H$$

$65 = (1000001)_2 = (101)_8 = (41)_H$

N.B. :

- Plus la base est grande, moins il faut de coefficients pour représenter le même nombre.
- Le système hexadécimal offre une représentation plus compacte.

5. Conversion binaire vers octal

On passe facilement du binaire à l'octal en groupant les bits par blocs de trois en allant vers la gauche puis on fait correspondre à chaque bloc son équivalent décimal. Le tableau suivant donne la correspondance entre les chiffres décimaux allant de 0 à 7 et leur équivalent binaire sur 3 bits :

Décimal	**Binaire**
0	$(000)_2$
1	$(001)_2$
2	$(010)_2$
3	$(011)_2$
4	$(100)_2$
5	$(101)_2$
6	$(110)_2$
7	$(111)_2$

Exemple: Convertir les nombres binaires suivants en octal :

a) $(100111010001)_2$; b) $(101110011101)_2$ c) $(10111)_2$

a) $(\underline{100}\ \underline{111}\ \underline{010}\ \underline{001})_2 = (4721)_8$

b) $(\underline{101}\ \underline{110}\ \underline{011}\ \underline{101})_2 = (5635)_8$

c) $(\underline{10}\ \underline{1\ 11})_2 = (27)_8$

6. Conversion octal vers binaire

Pour cela, il suffit d'associer à chaque chiffre du nombre octal son équivalent binaire sur 3 bits.

Exemple: $(402)_8 = (100\ 000\ 010)_2$

7. Conversion binaire vers hexadécimal

On passe du binaire à l'hexadécimal en groupant les bits par blocs de quatre en allant vers la gauche puis on fait correspondre à chaque bloc son équivalent hexadécimal. Le tableau suivant expose la correspondance entre les caractères hexadécimaux allant de 0 à F et leur équivalent binaire sur 4 bits :

Hexadécimal	**Binaire**
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100

D	1101
E	1110
F	1111

Exemple : Convertir les nombres binaires suivants en hexadécimal :

a) $(1100111010001)_2$; b) $(101011001101)_2$

a) $(\underline{1100}\ \underline{1101}\ \underline{0001})_2 = (CD1)_H$

b) $(\underline{1010}\ \underline{1100}\ \underline{1101})_2 = (ACD)_H$

8. Conversion hexadécimal vers binaire

Cela consiste à associer à chaque caractèret du nombre hexadécimal son équivalent binaire sur 4 bits.

Exemple : $(1FA2)_H = (\mathbf{0001}1111\mathbf{1010}0010)_2$

9. Conversion en complément à 1

On passe facilement d'un nombre binaire en son complément à 1 en inversant tousde les bits (les '0' deviennent des '1' et les '1' des '0').

Nombre binaire	*Complément à 1*
$(101)_2$	$(010)_2$
$(0111)_2$	$(1000)_2$

10. Conversion en complément à 2

Le complément à 2 d'un nombre binaire est tout simplement son complément à 1 auquel on additionne 1. L'addition en binaire se fait de la même manière que l'addition décimale avec la table suivante :

0 + 0 = 0 + retenue de 0

0 + 1 = 1 + retenue de 0

1 + 1 = 0 + retenue de 1

1 + 1 + 1 = 1 + retenue de 1

Exemple :

Nombre binaire	*Complément à 2*
$(101)_2$	$(011)_2$
$(0111)_2$	$(1001)_2$

III. Ecriture des nombres signés

Les circuits numériques n'assimilent pas les signes '+' et '-' tels quels. D'où la nécessité d'adopter une certaine convention pour représenter les nombres binaires signés. Pour cela, on ajoute un bit supplémentaire appelé bit de signe et on attribue au nombre positif le bit de signe '0' et au nombre négatif le bit de signe '1'.

Les nombres signés sont représentés selon trois notations :

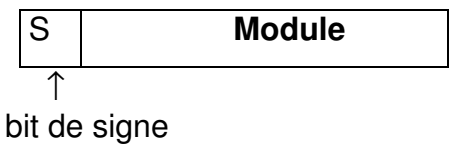
- Notation module plus signe,
- Notation complément à 1
- Notation complément à 1.

Les nombres positifs ont la même représentation dans les trois notations.

1. Notation module plus signe

Un nombre binaire signé comprend un bit de signe (0 pour les nombres positifs et 1 pour les nombres négatifs) et n bits indiquant le module. Ainsi, si un nombre N est codé sur n+1 bits, alors ce nombre est compris entre $\mathbf{-(2^n-1)}$ et $\mathbf{+(2^n-1)}$.

Avec 5 bits, N est compris entre -15 et 15.



Exemple :

+12 → 0 1100

-12 → 1 1100

Dans la notation 'Module plus Signe', le zéro a deux représentations différentes (le bit de signe S prend 0 ou 1).

2. Notation complément à 1

Pour représenter un nombre N négatif en notation complément à 1, il suffit d'attribuer au bit de signe la valeur de 1 et transformer le module en son complément à 1.

Exemple :

-12 → 1 0011 notation en complément à 1

+12 → 0 1100

3. Notation complément à 2

Pour représenter un nombre négatif en notation complément à 2, il suffit d'attribuer au bit de signe la valeur de 1 et de transformer le module en son complément à 2.

Exemple :

-12 → 1 0100 notation en complément à 2

+12 → 0 1100

Cas particulier du complément à 2 :

L'équivalent décimal du nombre binaire N=1 00...00 est -2^n .

Par conséquent, avec n+1 bits, on peut représenter des nombres signés allant de -2^n à $+(2^n -1)$.

IV. Codes

Jusqu'ici, nous n'avons utilisé que le code binaire naturel. Plusieurs codes sont utilisés en techniques numériques, nous citons entre autres :

- Le code DCB ;
- Le code binaire réfléchi ;
- Le code ASCII (American Standard Code for Information Interchange)
- Le code de parité.

1. Code DCB

Le code DCB signifie **D**écimal **C**odé **B**inaire. Chaque chiffre du nombre décimal est codé individuellement en son équivalent binaire sur quatre bits (quartet), ce qui n'est pas le cas pour le code binaire naturel où on convertit le nombre décimal dans son intégralité.

Exemple :

Nombre décimal	*code DCB*	*Code binaire*
127	$(000100100111)_{DCB}$	$(1111111)_2$
255	$(001001010101)_{DCB}$	$(11111111)_2$
64	$(01100100)_{DCB}$	$(1000000)_2$

Remarques :

- Le code DCB est un code non pondéré. Il n'obéit pas à la Forme Polynomiale.
- Dans le code DCB, il faut plus de bits pour exprimer le même nombre, qu'en code binaire.
- Le code DCB n'utilise que dix quartets parmi 16. Si l'un des quartets interdits (1010,1011,1100,1101,1110,1111) se manifeste dans un calculateur utilisant le code DCB, c'est alors un signe d'erreur.

2. Code binaire réfléchi

Dans le code binaire réfléchi (code Gray), deux représentations codées successives ne diffèrent que d'un seul bit.

En observant le tableau n°1 qui donne l'équivalent binaire des chiffres allant de 0 à 3 sur deux bits, on remarque que lorsqu'on passe de la représentation

binaire du chiffre ’1’ à celle de ‘2’ ; deux bits changent. Ceci n’est pas possible en code binaire réfléchi où un seul bit change seulement.

Décimal	Binaire
0	$(00)_2$
1	$(01)_2$
2	$(10)_2$
3	$(11)_2$

Tableau n°1

Le tableau n°2 expose la correspondance entre le décimal, le binaire et le binaire réfléchi sur 2 bits :

Décimal	Binaire naturel	Binaire réfléchi
0	$(00)_2$	$(00)_2$
1	$(01)_2$	$(01)_2$
2	$(10)_2$	$(11)_2$
3	$(11)_2$	$(10)_2$

Tableau n°2

Le tableau n°3 expose la correspondance entre le décimal, le code binaire et le code binaire réfléchi sur 3 bits :

Décimal	binaire	binaire réfléchi
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Le tableau n°4 expose la correspondance entre le code décimal, le code binaire et le code binaire réfléchi sur 4 bits :

Décimal	binaire	binaire réfléchi
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
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Le code binaire réfléchi est un code non pondéré ; il n'obéit pas à la forme polynomiale. Ce code est d'une grande utilité dans les conversions d'une grandeur analogique en une grandeur numérique.

3. Code ASCII 7 bits

Le code ASCII signifie **American Standard Code for Information Interchange,** il a été mis au point par l'organisation de normalisation

américaine appelée ANSI. Ce code constitue une norme universelle pour l'échange d'information entre le micro-ordinateur et ses périphériques (imprimante, clavier, écran, etc.).

Le code ASCII est un code alphanumérique. Il permet de représenter les chiffres, les lettres majuscules et les lettres minuscules ainsi que des caractères spéciaux (? ! + - : / # @ & ….).

Le code ASCII 7 bits fait correspondre à chaque caractère alphanumérique un code binaire sur 7 bits permettant ainsi de représenter au maximum 128 caractères différents.

Le tableau ci-dessous nous donne une partie du code ASCII où les 7 bits sont désignés par $b_6, b_5, b_4, b_3, b_2, b_1, b_0$:

				b_6	0	0	1	1	1	1
				b_5	1	1	0	0	1	1
				b_4	0	1	0	1	0	1
b_3	b_2	b_1	b_0	**HEX**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	**0**	*SP*	*0*	*@*	*P*		*p*
0	0	0	0	**1**	*!*	*1*	*A*	*Q*	*a*	*q*
0	0	1	0	**2**	*"*	*2*	*B*	*R*	*b*	*r*
0	0	1	1	**3**	*#*	*3*	*C*	*S*	*c*	*s*
0	1	0	0	**4**	*$*	*4*	*D*	*T*	*d*	*t*
0	1	0	1	**5**	*%*	*5*	*E*	*U*	*e*	*u*
0	1	1	0	**6**	*&*	*6*	*F*	*V*	*f*	*v*
0	1	1	1	**7**	*‘*	*7*	*G*	*W*	*g*	*w*
1	0	0	0	**8**	*(*	*8*	*H*	*X*	*h*	*x*
1	0	0	1	**9**	*)*	*9*	*I*	*Y*	*i*	*y*
1	0	1	0	**A**	*	*:*	*J*	*Z*	*j*	*z*
1	0	1	1	**B**	*+*	*;*	*K*	*[*	*k*	*{*
1	1	0	0	**C**	*,*	*<*	*L*	**	*L*	*|*
1	1	0	1	**D**	*-*	*=*	*M*	*]*	*m*	*}*
1	1	1	0	**E**	*.*	*>*	*N*	*^*	*n*	*~*
1	1	1	1	**F**			*O*	*_*	*o*	*DEL*

Tableau n°1

- SP : Espace
- Les codes de valeurs inférieurs à 32 (en décimal) sont réservés pour coder des caractères de contrôle qui ne sont pas imprimable.

Exemple : le code ASCII de ‘ A’ est : 1000001 = $(41)_H$

Le code ASCII 7 bits ne reconnaît pas les lettres accentuées comme dans le latin. Pour contourner cette limitation, ce code ASCII 7 bits a été étendu à 8 bits pour coder 256 caractères différents.

4. Code de parité

Le code de parité fait partie des codes détecteurs d'erreurs qui sont utilisés dans le contrôle de transmission d'information entre un émetteur et un récepteur. En effet, une erreur éventuelle sur un bit modifie le code transmis en un code erroné, ce qui entraîne une dégradation de la qualité de l'information transmise. Il faut donc utiliser des codes permettant de détecter ou même de corriger des erreurs éventuels. Il en existe plusieurs mais ils se basent tous sur l'introduction de bits supplémentaires calculés à partir des bits d'information.

Le code de parité est le code détecteur d'erreurs le plus simple. Il consiste à introduire un seul bit supplémentaire (p) appelé bit de parité.
Dans le code de parité paire, p prend 'o' ou '1' de telle manière que le nombre total de '1' soit pair (y compris le bit de parité).

Exemple 1 : associer un bit de parité paire p aux codes ASCII suivants :

	p	Code ASCII
A	**0**	1000001
D	**0**	1000100
F	**1**	1000110
4	**1**	0110100

Dans le code de parité impaire, p prend 'o' ou '1' de telle manière que le nombre total de '1' soit impair (y compris le bit de parité).

Exemple 2 : associer un bit de parité impaire p aux codes ASCII suivants :

	P	Code ASCII
A	**1**	1000001
D	**1**	1000100
F	**0**	1000110
5	**1**	0110101

Principe :

L'émetteur détermine le bit de parité sur le caractère qu'on veut transmettre. Ce bit est envoyé au récepteur. Le récepteur recalcule le bit de parité sur le code reçu et le compare au bit qu'il avait reçu auparavant. Si le bit de parité reste inchangé, il ne y'a donc pas d'erreur. Si le bit de parité a changé; cela veut dire qu'il y a une erreur. Comme ce code ne permet pas de corriger l'erreur, le récepteur va demander de retransmettre le code en question.

Supposons qu'on veut transmettre le code ASCII 7 bits du caractère '4' selon le code de parité paire. L'émetteur détermine le bit de parité paire p qui vaut '1'. Celui-ci est envoyé au récepteur.

Si il y a une erreur simple (sur un seul bit) et que le code reçu est par exemple '10100100' au lieu de '10110100'. Le récepteur recalcule le bit de parité sur le code reçu, il trouvera un '0'. Donc, le bit de parité a changé; ce qui indique qu'il y a une erreur puisque avant la transmission, l'émetteur et le récepteur ont convenu un bit de parité paire p égal à '1'. Comme ce code est incapable de corriger l'erreur, il faut alors retransmettre du caractère '4'.

Si il y a une erreur double (sur deux bits) lors de la transmission du caractère 4 et que le code reçu est par exemple '10100111' au lieu de '10110100'. Le récepteur recalcule le bit de parité sur le code reçu, il trouvera '1'. Donc, le bit de parité n'a pas changé et pourtant il y a erreur. Le code de parité ne permet pas donc de détecter que des erreurs simples (erreurs n'affectant qu'un seul bit) car le bit de parité va changer. Par contre, il ne peut pas détecter une erreur

double (sur 2 bits) car le bit de parité reste inchangé. Donc ce code ne peut être utilisé que dans des cas de transmission où le taux d'erreur est très faible ; à titre d'exemple entre un ordinateur et une imprimante.