IAM Ouaga
Année universitaire : 2019 – 2020
Réseaux Télécoms et Informatique de Gestion
1ère année de licence (L1)
Enseignant : M. SANA

Bases des services réseaux

Annexe: Systèmes de numération, Arithmétique binaire & Codes

A. Systèmes de numération

A.1. Généralités

Nos calculs sont toujours effectués dans le système de numération décimal. Un micro-ordinateur ne peut pas utiliser ce système, les circuits qui le constituent ne permettent pas de distinguer 10 états. Les calculs effectués par la machine sont donc tributaires de la nature particulière des circuits. ceux-ci ne pouvant prendre que deux états représentés par 0 et 1, le système de numération utilisé ne comportera que deux symboles 0 et 1, c'est le système de numération binaire. Pour étudier les systèmes de numération utilisés dans la technique du calcul digital, nous allons nous référer à un système qui nous est familier : le système décimal.

A.2. Le système décimal

La base du système est 10: Elle représente le nombre de symboles dont on dispose pour représenter les nombres dans ce système. Les 10 symboles du système décimal sont les chiffres successifs allant de 0 à 9. Ces chiffrent représentent une quantité par eux-même mais peuvent en représenter une autre par la position qu'ils occupent dans un nombre. La place occupée par le chiffre dans un nombre représente son rang.

Exemple :

$$(9817,45)_{10} = 9.10^3 + 8.10^2 + 1.10^1 + 7.10^0 + 4.10^{-1} + 5.10^{-2}$$
$$= 9000 + 800 + 10 + 7 + 0,4 + 0,05$$

Cas général :

$$A = a_n a_{n-1} \ldots\ldots\ldots\ldots a_0 , a_{-1} \ldots\ldots\ldots\ldots a_{-n}$$
$$= a_n .10^n + a_{n-1}.10^{n-1} + \ldots\ldots\ldots\ldots + a_0 .10^0 + a_{-1} .10^{-1} + \ldots\ldots\ldots\ldots a_{-n} 10^{-n}$$
$$= \left(\sum_{i\,n}^{n} a_i .10^i \right)_{10}$$

$a_i \subset \{0, 1, \ldots\ldots\ldots 9\}$

Pour une base B quelconque : $= (\sum_{i=-n}^{n} a_i \, . B^i)$

$a_i \subset \{0, 1, \ldots\ldots\ldots\ldots B-1\}$

A.3. Le système binaire

La base de ce système est 2 (B =2). Cela signifie que l'on dispose que 2 symboles pour représenter des nombres: Ces symboles sont 0 et 1. Un nombre binaire sera donc constitué d'une suite de 0 et de 1.

On a donné à ces symboles le nom de bits, contraction de binary-digit.

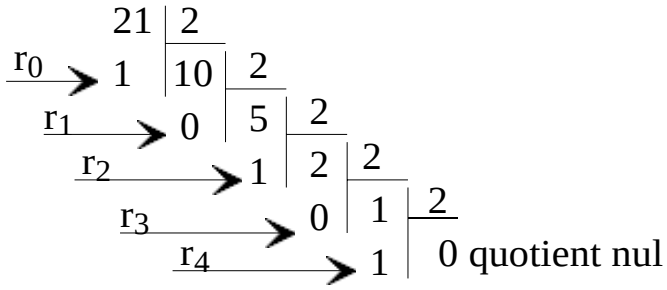
Exemple :

$$(1011,011)_2 = 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0 + 0.2^{-1} + 1.2^{-2} + 1.2^{-3}$$
$$= 8 + 0 + 2 + 1 + 0 + 0,25 + 0,125$$
$$= (11,375)_{10}$$

A.4. Conversion décimal-binaire

L'équivalent binaire d'un nombre décimal s'obtient en divisant successivement le nombre décimal par 2 jusqu'au moment où le quotient est nul. Ce dernier quotient et les restes des divisions successives, regroupés de droite à gauche, forment le nombre binaire cherché.

Exemple : Soit à convertir $(21)_{10}$



D'où $(21)_{10} = (10101)_2$

Vérification:

$$(21)_{10} = 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0$$
$$= 16 + 0 + 4 + 0 + 1$$
$$= (21)_{10}$$

Dans le cas de nombres fractionnaires on multiplie la partie fractionnaire par 2: Si après la multiplication un "1" apparaît à gauche de la virgule, on ajoute "1" à la fraction binaire en formation, si après la multiplication, c'est un "0" qui apparaît, on ajoute un "0".

Exemple : Soit à convertir $(0.4375)_{10}$

–e~r~~bit après la virgule

0.4375.2 = 0 .875
0.875.2 = 1 .750
0.750.2 = 1 .500
0.500.2 = 1 .000
0.000.2 = 0 .000

D'où $(0.4375)_{10} = (0,01110)_2$

Vérification :

$(0.4375)_{10} = 0.2^{-1} + 1.2^{-2} + 1.2^{-3} + 1.2^{-4}$
$= 0 + 0.25 + 0.125 + 0.0625$
$= (0.4375)_{10}$

A.5. Le système octal : (B = 8 et $a_i \in \{0, 1, ...7\}$)

Exemple :

$(476)_8 = 4.8^2 + 7.8^1 + 6.8^0$
$= 256 + 56 + 6$
$= (318)_{10}$

A.6. Conversion décimal-octal

Exemple : Soit à convertir $(4928)_{10}$

4928	8		
0 (r_0)	616	8	
0 (r_1)	77	8	
5 (r_2)	9	8	
1 (r_3)	1	8	
1 (r_4)	0 quotient nul		

D'où $(4928)_{10} = (11500)_8$

A.7. Conversion octal-binaire

La base du système octal étant égale à la puissance troisième de 2, l'équivalent binaire d'un nombre octal, s'obtient facilement en écrivant la suite des équivalents binaires de ses chiffres exprimés chacun au moyen de trois bits. ($B = 2^3 = 8$)

Exemple : $(4173)_8$

Chiffres octaux	**4**	**1**	**7**	**3**
Equivalent binaire	**100**	**001**	**111**	**011**

Ainsi $(4173)_8 = (100001111011)_2$

De même pour le passage inverse

Exemple :

$$(100\ 110\ 000\ 001)_2$$
$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$
$$(4 \quad 6 \quad 0 \quad 1)_8$$

A.8. Le système hexadécimal

On dispose de 16 symboles pour représenter un nombre hexadécimal. Ces symboles sont les dix chiffres du système décimal auxquels on a ajouté 6 signes supplémentaires. Par conversion il a été admis d'utiliser les six premières lettres de l'alphabet. Les 16 symboles du système sont alors : 0, 1, , 9, A, B, C, D, E et F.

Exemple :

$$(4CA2)_{16} = 4.16^3 + C.16^2 + A.16^1 + 2.16^0$$
$$= 16384 + 3072 + 160 + 2$$
$$= (19618)_{10}$$

A.9. Conversion décimal-hexadécimal

Exemple : Soit à convertir $(92628)_{10}$

92628	16	
r_0 → 4	5789	16
r_1 = D → 13	361	16
r_2 → 9	22	16
r_3 → 6	1	16
r_4 → 1	0	quotient nul

D'où $(92628)_{10} = (169D4)_{16}$

Application : Convertissez en hexadécimal ces nombres décimaux :

a. 75 b. 314 c. 2048 d. 25619 e. 4095

A.10. Conversion hexadécimal-binaire

La base du système hexadécimal étant égale à la puissance quatrième de 2, l'équivalent binaire d'un nombre hexadécimal s'obtient en écrivant la suite des équivalents binaires de chacun de ses signes exprimés au moyen de 4 bits. ($B = 2^4 = 16$)

Exemple : $(4CA2)_{16}$

Chiffres hexadécimaux	**4**	**C**	**A**	**2**
Equivalent binaire	**0100**	**1100**	**1010**	**0010**

D'où $(4CA2)_{16} = (0100110010100010)_2$

De même pour l'inverse

Exemple :

$$(1\underline{001}\ 1\underline{000}\ \underline{0001})_2$$
$$(\ 9 \quad 8 \quad 1\ )_{16}$$

Application : Convertissez en hexadécimal les nombres binaires :

a. 10110 b. 10001101 c. 100100001001 d. 1111010111 e. 1011111

A.11. Virgule fixe et virgule flottante

Dans un ordinateur les entiers sont représentés en virgule fixe ou fixée. La virgule binaire est fixée dans la mesure qu'elle se trouve à la fin de chaque mot binaire et ainsi chaque valeur représentée est un entier positif ou négatif.
Exemple de codification d'entiers relatifs sur un mot mémoire UNIVAC 1100 (le mot comprend 36 bits avec un bit de signe dans la position 35)

Les entiers positifs exemple : $(12)_{10} = (1100)_2$

0.	....	**0**	**1**	**1**	**0**	**0**

35 0

Les entiers négatifs exemple : $(-12)_{10}$. On utilise le complément à "1"

1.	....	**1**	**0**	**0**	**1**	**1**

35 0

L'entier 0

0.	....	**0**

35 0

Ou

1.	....	.....	**0**
35			0

En calcul scientifique, il est souvent nécessaire de calculer avec des nombres très grands ou très petits. Ainsi on représente le nombre par une mantisse et un exposant; et on parle de représentation en virgule flottante.

Exemple : $(4900000)_{10} = (0.49.10^{7})_{10}$

$$x = ((0, A)_{10} .10^{(E)})_{10}$$

De même

$$(0.00023)_{10} = (0.23.10^{-3})_{10}$$

Exemple de codification des nombres rationnels sur UNIVAC 1100

Soit $x = (0.1275)_{10} = (0.001100)_2 = (0.11)_2.2^{126-128} = (0.75)_{10}.2^{-2}$

0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	....	**0**
35	**34**	**Exposant**						**27**	**26**	**Mantisse**			**0**

signe

Soit $x = (12)_{10} = (1100)_2 = (0.11)_2.2^{132-128} = (0.75)_{10}.2^{4}$

0	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	....	**0**
35	**34**	**Exposant**						**27**	**26**	**Mantisse**			**0**

signe

Remarque

Les nombres rationnels binaires sont enregistrés en normalisés (le premier chiffre significatif de la mantisse est un "1" logique).

La codification des nombres rationnels négatifs se fait par complémentation à "1".

Application

Dans la plus part des micro-ordinateurs, les adresses des emplacements mémoires sont exprimés en hexadécimal. Ces adresses sont des nombres séquentiels qui identifient chacune des cases mémoires.

a. Un certain micro- ordinateur peut stoker des nombres de 8 bits dans chacune de ses cases mémoires. Si l'intervalle des adresses mémoires va de 0000_{16} à $FFFF_{16}$, dites combien cet ordinateur a de cases de mémoires.

b. Un autre micro-ordinateur possède 4096 emplacements en mémoire. Donnez l'intervalle de ses adresses exprimées en hexadécimal.

B. Arithmétique binaire

Quand un calculateur effectue une opération arithmétique, deux contraintes interviennent:

- les circuits travaillent sur des nombres qui ont toujours la même longueur (FORMAT); par exemple, dans une machine de 8 bits (ou 8 éléments binaires ou 8 e.b), le nombre 10111 doit être complété par zéros 00010111;
- les circuits ne manipulent que deux nombres à la fois; pour calculer S = X +Y +Z, ils effectuent X +Y = s puis s + Z = S.

En effectuant une addition, une retenue (**CARRY**) peut apparaître; dans une soustraction, il peut y avoir une retenue (**BORROW**); enfin si en manipulant des nombres trop grands pour le format, on obtient un dépassement de capacité (**OVERFLOW**)différent d'une retenue; il doit être signalé par un drapeau (**FLAG**).

B.1. Les quatre opérations

Les mêmes règles de calcul s'appliquent dans tous les systèmes de numération; l'arithmétique binaire à celle des nombres décimaux.

Addition		**Soustraction**	
	retenue		***retenue***
0 + 0 = 0	0	0 - 0 = 0	0
0 + 1 = 1	0	0 - 1 = 1	1
1 + 1 = 0	1	1 - 1 = 0	0
1 + 0 = 1	0	1 - 0 = 1	0

Multiplication	**Division**		
		quotient	***reste***
0 . 0 = 0	0 : 0	impossible	
0 . 1 = 0	0 : 1	0	1
1 . 1 = 1	1 : 1	1	0
1 . 0 = 0	1 : 0	impossible	

Exemple

Addition						**Soustraction**					
Décimal	***Binaire***					***Décimal***	***Binaire***				
	r	+1	+1	+1			***r***	-1	-1		
11		1	0	1	1	19	1	0	0	1	1
+ 07	+	0	1	1	1	- 05	-		1	0	1
= 18	= 1	0	0	1	0	= 14	= 0	1	1	1	0

Multiplication	
Décimal	***Binaire***
11 x 5 = 55	1011 x 101 1011 1011. = 110111

Des circuits électroniques assez simples réalisent des additions. On ramène une soustraction à une addition à l'aide d'une représentation adéquate des nombres négatifs.

Pour faire une multiplication, il faut effectuer les produits partiels, des décalages et des additions. Quant à la division, elle n'est pas faite comme en numération décimale. Elle se ramène à une suite de multiplication et de comparaison.

A retenir

Les quatre opérations arithmétiques nécessitent trois opérateurs (additionneur, multiplicateur, comparateur) et un circuit de décalage.

B.2. Les nombres négatifs

Afin d'utiliser un additionneur comme soustracteur, il faut une représentation convenable des nombres négatifs.

a. Représentation avec valeur absolue et le signe (module & signe)

On adopte la convention 0 (+), 1 (-) pour le signe:

+35 = **0** 100011 - 35 = **1** 100011

il faut, pour le signe un traitement spécial, différent de celui des e.b de la valeur absolue, ce qui ne permet pas une utilisation commode de l'additionneur en soustracteur.

b. Représentation par le complément restreint (complément à 1)

On obtient le complément restreint CR(X) d'un nombre X en remplaçant ses 0 par des 1 et ses 1 par des 0. Si n est le nombre d'e.b de X, on a $X + CR(X) = 2^n - 1$ ainsi que le montre l'exemple suivant (n = 4):

X = 1101
CR(X) = 0010
X + CR(X) = 1111 = $2^4 - 1$

On déduit que : - X = CR (X) + 1 - 2^n, ce qui montre que CR (X) + 1 représente le nombre négatif - X à condition de ne pas tenir compte du chiffre de poids 2^n dans l'expression numérique de CR (X) + 1, c'est-à-dire de conserver les n e.b de poids S 2^{n-1}

La quantité CV (X) = CR (X) + 1 s'appelle le complément vrai. En écrivant :

+ 13	= 000	1101
CR(13) + 1	= 111	0011
13 + CV(13)	=.....000	0000

on a bien éliminé le terme de poids 24 et la relation précédente donne :

13 + CV(13) = 0000

ou -13 = CV(13). A condition de manipuler des nombres de longueur fixe.

c. Représentation par le complément vrai (Complément à 2)

C'est la représentation utilisée dans tous les calculateur. Nous venons de voir comment on aboutit naturellement à cette représentation des nombres négatifs.

CV(X) = CR(X) +1 × - X

+ 0	0	0000	- 0	0	0000
+ 1	0	0001	- 1	1	1111
+ 2	0	0010	-2	1	1110
.....	....	.....	.....	.....	.....
+ 15	0	1111	-15	1	0001
			-16	1	0000

e.b.de signe

Quelques remarques doivent être faites :

- on une seule représentation du zéro, ce qui n'est pas le cas avec le complément restreint;
- de la relation X + CR(X) = 2^n - 1 on déduit CV(X) = 2^n - X et CV[CV(X)] = X;
- la représentation de +16 (0 10000) ne convient pas car elle a 6 e.b. au lieu de 5; en outre, on obtient CV(16) = 101111+1 = 110000 qui a 6 e.b. et n'est pas contenue avec la représentation des nombres -1, ..., -15;
- Par contre la représentation (10000) de -16 convient car elle a 5 éléments binaires (e.b.), elle contenue avec les représentation des nombres -1,.....,-15 et parce qu'elle redonne la valeur absolue correcte:

CV[CV(16)] = CV(10000) = 01111 + 1 = 10000 = 16

B.3. Problèmes liés à la longueur des nombres

Nous venons de montrer que les circuits traitant des nombres de n e.b. (n = 8 par exemple) peuvent manipuler tous les nombres entre -2^{n-1} et $2^{n-1}-1$ (-128 et +127) à condition que les résultats partiels ne sortent pas de cet intervalle.

a. Addition de deux nombres positifs

En ajoutant 2 nombres > 0 (e.b. de signe, 0) on peut obtenir un résultat dont l'e.b. de signe est 1 (bien que le résultat soit positif) si le résultat est hors de l'intervalle autorisée (+127, -128 dans l'exemple suivant) :

```
+ 49   0   0 1 1 0 0 0 1          + 49   0   0 1 1 0 0 0 1
+ 33   0   0 1 0 0 0 0 1          +88    0   1 0 1 1 0 0 0
-------  ----------------         --------  ------------------------
+ 82   0   1 0 1 0 0 1 0          + 137  1   0 0 0 1 0 0 1
       +       82 ⟵                        ⟵ CV(119) = - 119
```

dépassement de capacité (8 ème e.b. à 1)

Les calculateurs utilisent un indicateur de dépassement de capacité (OVERFLOW FLAG) $f_d = 1$ si dans le résultat R l'e.b. de signe S_R est à 1 alors que dans les deux nombres positifs A et B les éléments binaires de signes S_A et S_B sont à 0. On écrit $f_d = \overline{S_A}.\overline{S_B}.S_R$

b. Addition de deux nombres négatifs

En additionnant deux nombres < 0 représentés par leurs compléments vrais (e.b. de signe, 1) on peut obtenir un résultat dont l'e.b. de signe (rang n) est 0 (bien que le résultat soit négatif). En outre il y a toujours une retenue car l'e.b. de rang (n+1) est toujours à 1.

```
 - 32      1   1 1 0 0 0 0 0          - 32      1   1 1 0 0 0 0 0
 - 31      1   1 1 0 0 0 0 1          -127      1   0 0 0 0 0 0 1
------     ---------------          ------  ----------------
 - 63   1  1   1 0 0 0 0 0 1          -159  1   0   1 1 0 0 0 0 1
retenue ⟵      CV(63) ⟵          retenue ⟵      ⟶ + 97 → FAUX
              -      63                       dépassement car 8ème e.b. à 0
```

Le 9ème e.b. est toujours à 1 de même que les 10ème, 11ème ... etc. Dans la représentation par le complément vrai; par exemple :

```
+ 11 : .....0 0 0 0 1 0 1 1
CV(X)..... 1 1 1 1 0 1 0 1
```

Ce 9ème e.b. est donc une retenue. Des deux exemples précédents, le premier donne un résultat correct, car il est dans l'intervalle (+127, -128); le second donne un résultat faux à cause du dépassement de capacité (-159 est hors de l'intervalle, +127, -128). L'indicateur de dépassement $f_d = 1$ car, dans le résultat, l'e.b. de signe S_R (8ème e.b.) est à 0 alors que dans les 2 nombres négatifs les e.b. de signe S_A et S_B (8ème e.b.) sont à 1; on écrit $f_d = S_A . S_B . \overline{S_R}$ avec la notation présentée à la fin du paragraphe précédent. On réunit ces 2 résultats en écrivant

$f_d = \overline{S_A} . \overline{S_B} . S_R$ ou $S_A . S_B . \overline{S_R}$

c. Addition d'un nombre positif et d'un nombre négatif

Cela revient à effectuer la soustraction A - B. Pour cela, on forme :

$$A + CV(B) = A + 2^n - B = 2^n + A - B$$

- Si A ≥ B on a A - B ≥ 0 et $A + CV(B) \geq 2^n$; donc :

$$A - B = A + CV(B) - 2^n$$

La soustraction se ramène à une addition dans laquelle on néglige la retenue qui apparaît ($n+1^{ième}$ e.b. de poids 2^n) : la présence de ce terme dans une comparaison de conclure à A ≥ B.

- Si A < B on a A - B < 0 et $A + CV(B) < 2^n$

$$A + CV(B) = 2^n - (B - A) = CV(B - A).$$

Il n y a pas de retenue (n $+1^{ième}$ e.b. de poids 2^n), ce qui, dans une comparaison, permet de conclure à A < B et dans une soustraction de prendre le complément vrai du résultat obtenu qui est CV(B-A); en effet, on a :

$$CV(CV(B-A)) = 2^n - CV(B-A) = 2^n - [2^n - (B-A)] = B-A$$

9		0	1 0 0 1	7		0	0 1 1 1
-5		1	1 0 1 1	-10		1	0 1 1 0
+4	1	0	0 1 0 0	- 3		1	1 1 0 1
rejeter		+	4				CV (3)
						-	3

C. LES CODES

C.1. Généralités

Le codage permet la spécification des caractères (par exemple caractère numérique ou alphanumérique) en utilisant d'autres symboles. Les codes ont été utilisés pour des raisons de sécurité; ce qui a permis la protection des messages secrets. Dans les ordinateurs la codification permet de spécifier les caractères en utilisant simplement les symboles binaires 0 et 1. Le choix parmi plusieurs codes binaires dépend de la fonction à accomplir.

Il y a des codes qui se prêtent bien pour des opérations arithmétiques. D'autres sont meilleurs parce qu'ils sont plus efficace en donnant plus d'information avec le minimum de bit. L'importance qu'apporte également les codes réside dans la détection et la correction d'erreur; en effet les codes donnent les possibilités à un calculateur de déterminer si un caractère qui a été codé et transmis a été reçu correctement; et s'il existe une erreur de la corriger. Le codage en lui même est un sujet très vaste, on présentera dans ce qui suit les codes les plus familiers.

C.2. Code binaire naturel, code 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; le tableau 1 présente le code binaire naturel pour des nombres compris entre 0 et 16.

Tableau 1

décimal	C.B.Naturel
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

Un nombre binaire à 16 bits (1110100101101110 par exemple), utilisé par un ordinateur, est difficilement lu et reconnu. En plus la conversion en décimal n'est

pas facile à obtenir. Afin de permettre une expression facile des nombres binaires on utilise le codage octal ou hexadécimal. La majorité des calculateurs utilisent en effet le codage hexadécimal pour représenter les nombres binaires.

Exemple: Soit à exprimer le nombre binaire suivant en octal et en hexadécimal:

1011110101000101 1101

Ce nombre est représenté en octal comme suit:

010	111	101	010	001	011	101
2	7	5	2	1	3	5

ou en hexadécimal:

1011	1101	0100	0101	1100
B	D	4	5	C

C.3. Décimal codé binaire BCD « Binary Coded Decimal »

a. BCD-(8421)

C'est un code utilisé pour exprimer en binaire un nombre décimal compris entre 0 et 9 (voir tableau 2)

Tableau 2

décimal	BCD (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ce code nécessite 4 bits pour représenter un caractère décimal, Il est moins efficace que le système décimal, mais il a l'avantage d'être sous forme binaire pour être facilement exploitable sur ordinateur.

Quelques exemples de ce code:

Décimal	BCD (8421)
0110 0010	62
0101 0011	53
0110 1001 0001	691
0011 0111 0101 1001	3759

Dans les opérations arithmétiques on rencontre des difficultés dans l'utilisation des codes BCD.

Exemple:

Décimal	Binaire	BCD(8421)	
8	1000	1000	
+ 7	+ 0111	0111	
15	1111	1111	Ce caractère n'est pas accepté dans le BCD (15: 0001 0101)

Un additionneur spécial est nécessaire pour assurer les opérations dans le code BCD.

b. Autres codes BCD

En plus du code BCD (8421) qui est le plus utilisé, il existe plusieurs autres codes utilisant des bits pour représenter les nombres décimaux. Ces codes peuvent être à 4 bits, 5 bits, 7 bits (biquinaire) et même à 10 bits. On présentera dans ce qui suit quelques codes à 4 et à 5 bits qui sont les plus utilisés.

b.l. BCD excédent 3 (exc3)

Il représente également les nombres décimaux de 0 à 9. Le tableau 3 illustre le code BCD exc3 qui est obtenu en ajoutant 3 (0011) à chaque valeur du code BCD (8421). Contrairement au code BCD (8421) l'excédent 3 est un code qui n'est pas pondéré.

Tableau 3

décimal	BCD(8421)	Excèdent 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

Exemple: représentation de quelques nombres en excédent 3

Décimal	Excédent 3
5	1000
30	0110 0011
24	0101 0111
361	0110 1001 0100
8496	1011 0111 1100 1001

Le code excédent 3 permet d'assurer les opérations arithmétique d'une façon plus facile (en comparaison avec le code BCD (8421)).

Exemple1 :

```
   4  0111
 + 2  0101
 ------------
   6  1100     (pas de retenue
 -      11      on retranche donc 3)
 ------------
      1001
```

Exemple 2:

```
   3              0110
 + 9            + 1100
 ----           ------------
  12             1 0010      (Retenue = l
                 + 0011            on ajoute 3)
                ------------
                  0101
```

Il existe donc quelques règles à suivre pour accomplir l'opération d'addition (exemple: ajouter 3 à chaque digit s'il existe une retenue), Ces étapes lorsqu'ils sont programmées peuvent être assurées automatiquement sur le calculateur, ce qui rend l'excédent 3 plus apprécié pour les opérations arithmétiques.

b.2. Codes BCD à 4 bits

Ces codes sont très nombreux, quelques exemples de codes pondérés sont représentés dans le tableau 4.

Tableau 4

Décimal	4221	5421	7421	74-2-1	84-2-l
0	0000	000	000	0000	000
1	0001	001	001	0111	111
2	0010	010	010	0110	110
3	0011	011	011	0101	101
4	0110	100	100	0100	100
5	0111	000	101	1010	011
6	010	001	110	001	010
7	011	010	111	000	001
8	110	011	001	111	000
9	111	100	010	110	111

Exemple 1: Soit à convertir en décimal le nombre suivant (représenté dans le code BCD (5421)): 0011 1000 1010

0011 = 0 x 5 + 0 x 4 + 1 x 2 + 1 x 1 = 3
1000 = 1 x 5 + 0 x 4 + 0 x 2 + 0 x 1 = 5
1010 = 1 x 5 + 0 x 4 + 1 x 2 + 0 x 1 = 7

En se référant au tableau 4 on peut vérifier:

$$(0011\ 1000\ 1010)_{5421} = (357)_{10}$$

Exemple 2: Soit à convertir en décimal le nombre suivant (représenté dans le code BCD(74-2-1)): 0101 0110 1111

0101 = 0 x 7 + 1 x 4 - 0 x 2 - 1 x 1 = 3
0110 = 0 x 7 + 1 x 4 - 1 x 2 - 0 x 1 = 2
1111 = 1 x 7 + 1 x 4 - 1 x 2 - 1 x 1 = 8

En se référant à au tableau 4 on peut vérifier:

$$(0011\ 1000\ 1010)_{74\text{-}2\text{-}1} = (328)_{10}$$

b.3. Codes BCD à 5 bits

Dans le tableau 5 on présente deux codes BCD à 5 bits; le premier code, appelé 2 parmi 5, est un code pondéré qui est utilisé pour la détection d'erreur dans les systèmes de communication. Le deuxième code (51111) est un code pondéré; il a l'avantage d'être auto-complémenté.

Tableau 5

Décimal	2 parmi 5	51111
0	00011	0000
1	00101	00001
2	00110	00011
3	01001	00111
4	01010	01111
5	01100	10000
6	10001	11000
7	10010	11100
8	10100	11110
9	11000	11111

Exemple: Soit à exprimer le nombre 285 dans les codes:

a) 2 parmi 5

$(285)_{10}$ = 00110 10100 01100

b) BCD (51111)

$(285)_{10}$ = 00011 11110 10000

C.4. Les codes alphanumérique

En plus des codes binaires spécifiant les nombres décimaux de 0 à 9, il existe plusieurs codes qui peuvent représenter à la fois les caractères numériques et alphabétiques. Ces codes sont appelés alphanumériques. parmi ces codes le ASCII (American Standard Code for Information Interchange) qui est le plus utilisé.

ASCII est un code standard à 7 bits (généralement le 8ème bit pour la parité). Ce code permet la spécification des nombres décimaux, des caractères alphabétiques et quelques caractères spéciaux. Dans le tableau 6 figure tous les caractères du code ASCII et leur valeur correspondante en binaire et en hexadécimale.

Exemple : Un message transmit ("START 1." par exemple) se présente au niveau de l'ordinateur en code ASCII de la manière suivante:

S	T	A	R	T	1	.
1010011	1010100	1000001	1010010	1010100	0110001	0101110

Plus simplement on peut représenter en hexadécimal

S	T	A	R	T	1	.
53	54	41	52	54	31	2E

Tableau 6

C.5. Codes de détection et de correction d'erreurs

Dans divers cas de transmission de données, les informations binaires peuvent être erronées. Ceci peut être dû à un mauvais fonctionnement d'un circuit électronique ou encore à la longue distance de transmission. La détection et la correction d'erreurs est un domaine d'étude très vaste dans les transmissions de données.

a. Bit de parité

La détection d'erreurs la plus populaire est celle utilisant un bit de parité.

Exemple:

bit de parité (impaire)	bit de parité (paire)
010001 1	010101 1
donnée	donnée

Lorsqu'un mot est reçu, sa parité est testé (paire ou impaire: choix fait à l'émission) et il est accepté comme correct s'il satisfait le test. Malheureusement ce simple test n'est pas suffisant pour localiser le bit erroné; en plus deux erreurs dans un même mot passent inaperçues. Le tableau 7 présente le code BCD (8421) avec les bits de parité pour les deux cas (paire et impaire).

Tableau 7

Décimal	BCD 8421	Parité Paire	Impaire
0	0000	0000 0	0000 1
1	0001	0001 1	0001 0
2	0010	0010 1	0010 0
3	0011	0011 0	0011 1
4	0100	0100 1	0100 0
5	0101	0101 0	0101 1
6	0110	0110 0	0110 1
7	0111	0111 1	0111 0
8	1000	1000 1	1000 0
9	1001	1001 0	1001 1

Exemple: soit à déterminer les erreurs dans les valeurs suivantes:

Mots	Bit de parité	Type de parité
1001	0	impaire
11010	0	impaire
100101	0	paire
1010	0	paire
1001100	1	paire

Dans les exemples précédents, le premier et le troisième sont incorrects.

b. Détection et correction d'erreur

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

Exemple 1:

bit de parité

01111001
00000111
10001111
01101110
00110001
01011000
11110001
11111000 ← mot de parité

Exemple 2: Détermination et correction d'erreur

0111 1 001
0000 0 111
1000 1 111
0110 1 110
0011 1 001 ← erreur de parité détecté
0101 1 000
1111 0 001
1111 1 000

erreur de parité détecté ↑

c. Détection et correction d'erreur sur un seul mot binaire

Considérons un mot binaire à 4 bits

b3 b2 bl b0

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

Pl détermine la parité de **b3 b2 bl**
P2 détermine la parité de **b3 bl b0**
P3 détermine la parité de **b2 bl b0**

L'information à transmettre sera alors : **P3 P2 P1 b3 b2 bl b0**

A la réception il peut y avoir des erreurs dans n'importe quel bit parmi le mot à 7 bits.

Si par exemple la parité n'est pas vérifiée pour P2 et P3 alors l'erreur s'est produite dans b0.

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

Parité non vérifiée	Erreur
P2 P3	b0
Pl P2 P3	bl
Pl P3	b2
Pl P2	b3
P3	P3
P2	P2
Pl	Pl

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

Solution: 1 1 1 0 1 0 0
P3 P2 Pl b3 b2 bl b0

En utilisant les définitions des parités de P3 , P2 et Pl on trouve: Pl = 0, P2 = 1 et P3 = 0. La parité n'est pas vérifiée pour Pl et P3. D'après le tableau l'erreur se trouve dans b0.

C.6. Code de gray

Il est dit également code binaire réfléchi; il est surtout utilisé pour coder des positions. on rencontre ses applications dans le domaine optique et mécanique. Le code de Gray est un code non pondéré et il est caractérisé par le changement d'un seul bit entre deux mots successifs.

Le tableau 8 présente le code de Gray pour des valeurs comprises entre 0 et 12.

Tableau 8

décimal	C.B.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
10	1010	1111
11	1011	1110
12	1100	1010