Codes & Numération

A. Les systèmes de numération

I.1 La représentation polynomiale

Si nous manipulons les nombres de manière intuitive, c'est la plupart du temps dans la base décimale, naturelle et universelle. Mais cela ne doit pas masquer la nature même de la numération qui peut prendre plusieurs formes, parmi lesquelles on trouve la théorie des ensembles et la représentation polynomiale.

La représentation polynomiale d'un nombre est sa représentation sous la forme sui vante :

$$a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + a_{n-3}b^{n-3} + \ldots + a_2b^2 + a_1b + a_0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \ldots + a_{-m}b^{-m}$$

où b est appelée la base.

Si la base 10 nous est familière, d'autres bases existent et les bases les plus utilisées en informatique sont les bases 10, 2, 8 et 16 appelées respectivement « décimale », « binaire », « octale » et « hexadécimale ».

I.2 Le système binaire

I.2.1 Introduction

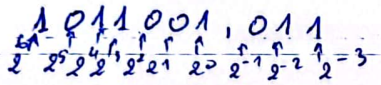
Le système décimal est malheureusement difficile à adapter aux mécanismes numériques, car il est difficile de concevoir du matériel électronique fonctionnant sur dix plages de tensions différentes.

On lui préférera donc le système binaire :

CamScanner

- base B=2 ;
- 2 symboles : {0,1} appelés « éléments binaires » ou « bits » (bit=*Binary digIT*) ;
- le système binaire est pondéré par 2 : les poids sont les puissances de 2 ;

Exemple :



- un ensemble de 8 bits est appelé « octet » (ou *byte*).

I.2.2 Comptage binaire

On présente les nombres binaires en général avec un nombre fixe de bits, nombre imposé par les circuits mémoires utilisés pour représenter ces nombres.

Suite des nombres binaires à 4 bits :

Poids :	2^3	2^2	2^1	2^0	B10
	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

CamScanner

Le bit le plus significatif (le bit le plus à gauche) est appelé « bit de poids fort »ou MSB (*Most Significant Bit*).

Le bit le moins significatif (le bit le plus à droite) est appelé « bit de poids faible »ou LSB (*Less Significant Bit*).

Si on utilise N bits, on peut représenter 2^N valeurs différentes de 2^0 à $2^N - 1$

Exemple

N= 8 ; $0 = (00000000)_2 \rightarrow 2^8 - 1 = 255 = (11111111)_2$

▸ **Remarque**

Comme l'on traite souvent en micro-informatique de nombres à 8 ou 16 éléments binaire , on se sert des systèmes :
- *octal : à base 8 ;*
- *hexadécimal : à base 16.*

I.3 Le système octal

- base B=8 ;
- 8 symboles : {0, 1, 2, 3, 4, 5, 6, 7} ;

L'intérêt de ce système est que la base 8 est une puissance de 2 (8= 2^3), donc les poids sont aussi des puissances de 2.

Chaque symbole de la base 8 est exprimé en binaire sur 3 bit :
$(ai)8 = bi_2\, bi_1\, bi_0$

Exemple :

$(523)_8 = (101\,010,011)_2$

I.4 Le système hexadécimal

- base B=16 ;

CamScanner

- 16 symboles :
  { 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , A , B , C , D , E , F } appelés « digits » ;
- chaque symbole est exprimé en binaire sur 4 bits ;

Exemple :

$(F3D,2)_{16} = (1111001111101,0010)_2$.

I.5 Conversion d'un système de numération à un autre

I.5.1 Base B vers base 10

$(an \ldots a0)B = an B^{n} + \ldots + a0 B^{0} = (a'_m \ldots a'_0)10$

Exemple

$(1001,1)_2 = (?)_{10} = 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1}$

$= (9,5)_{10} \;/\; (A12)_{16} = A\times16^2 + 1\times16^6 + 2\times16^0 = (2578)_{10}$

I.5.2 Base 10 vers base B

a. Première méthode

Elle consiste à soustraire successivement la plus grande puissance de *B*

Exemple

$(100)_{10} = (?)_2$

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

$100 = (2^6 = 64) + 36$

$36 = (2^5 = 32) + 4$

$4 = 2^2$

$(100)_{10} = (1100100)_2$

b. Deuxième méthode

Elle consiste à diviser par *B* autant de fois que cela est nécessaire pour obtenir un quotient nul. Ensuite on écrit les

restes dans l'ordre inverse de celui dans lequel ils ont été obtenus.

Pour la partie fractionnaire on multiplie par B (résultat nul ou selon la précision demandée)

Exemple

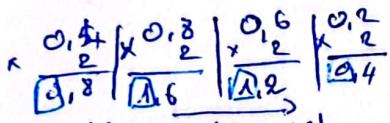
(20,4)10 = (?)2

$(20)_{10} = (?)_2$



$(20)_{10} = (10100)_2$

$(0,4)_{10} = (?)_2$



on doit chercher ... 0 si n'est pas possible on arrête après 4 fois par exemple pour donner 4 chiffres après la virgule !

$\Rightarrow (0,4)_{10} = (0,0110)_2$

$(20,4)_{10} = (10100,0110)_2$

I.5.3 Base 2^n vers base 2

Chaque symbole de la base $B = 2^n$ peut être représenté par n bits.

Exemple

$(3A9)_{16} = (?)_2$

$16 = 2^4 \Rightarrow$ chaque symbole est écrit en binaire sur 4 bits

$(3A9)_{16} = (0011\ 1010\ 1001)_2$

$(742,5)_8 = (?)_2$

$8 = 2^3$

$(742,5)_8 = (111\ 100\ 010,\ 101)_2$

I.5.4 Base 2 vers base 2^n

Il suffit de regrouper les bits par paquets de n bits.

Exemple

$(1011\,1011)_2 = (?)_8 = (?)_{16}$

$(10\,111\,011)_2 = (010\,111\,011)_8 = (133)_8$

$(10111011)_2 = (1011\,1011) = (5B)_{16}$

I.5.5 Base i vers base j

- si i et j sont des puissances de 2, on utilise la base 2 comme relais ;

base i → base 2 → base j

Exemple

Base 8 → Base 2 → base 16

- sinon, on utilise la base 10 comme relais.

base i → base 10 → base j

Exemple

Base 5 → Base 10 → Base 2

II.1 Représentation des nombres entiers positifs

Les nombres sont représentés en binaire sur n bits : n = nombre d'unités mémoires (n= 8, 16, 32, 64,...)

On peut représenter des nombres allant de 0 à $2^n - 1$.

II.2 Représentation binaire des entiers signés

Traditionnellement on met un signe «-» pour représenter les

CamScanner

nombres négatifs. Mais les systèmes logiques ne permettent de présenter qu'un des deux symboles « 0 » et « 1 », il faut chercher une convention pour remplacer le « – ».

II.2.1 Représentation module et signe

Solution la plus simple : on ajoute un bit à gauche du module pour le signe. Ainsi, un nombre commençant par un « 0 » sera positif, alors qu'un nombre commençant par un « 1 » sera négatif : 0→+ et 1→ -.

Exemple

avec 4 bits les valeurs vont -7 à +7

signe	module	valeur	signe	module	valeur
1	111	-7	0	111	7
1	110	-6	0	110	6
1	101	-5	0	101	5
1	100	-4	0	100	4
1	011	-3	0	011	3
1	010	-2	0	010	2
1	001	-1	0	001	1
1	000	0	0	000	0

Problème :

on a 2 representation differente pour 0 (0000 / 1000).

II.2.2 Représentation en complément restreint (complément à 1)

$\bar{A} = \overline{A}$: pour prendre l'inverse d'un nombre, il suffit de le complémenter (inversion de tous ses bits). Comme dans le cas précédent, la nature du premier bit donnera le signe : 0→+ et 1→ -.

Exemple

avec 4 bits :

$(+5)_{10} = (0101)$

$(-5)_{10} = (1010)$

CamScanner

Problème :

2 representation differentes pour le 0

II.2.3 Représentation en complément vrai (complément à 2)

C'est la représentation la plus utilisée. Le bit le plus à gauche est encore le bit de signe : 0→+ et 1→ -.

$$\begin{aligned} -A &= \overline{A}+1 \\ \overline{A} &= \overline{a_{n-1}\,a_{n-2}\cdots a_0}^{\,1} \\ A+\overline{A} &= 11\ldots1 \\ 1+\overline{A}+A &= 1\underbrace{00\ldots0}_{0}{}^{2} \end{aligned}$$

$\Rightarrow -A = \overline{A}+1$ est appelé complément à 2

$(9) = (01001)_{c\grave{a}1} \quad \begin{array}{r} 10110 \\ +\quad\ 1 \\ \hline 10111 \end{array} \quad (-9) = (10111)_{c\grave{a}2}$

▸Remarques

- *pour passer d'une valeur négative à une valeur positive, on applique aussi 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

$n = 8 \Rightarrow 2^8$ valeurs $\begin{cases} 0 \to 2^7 - 1 = 127 \\ -1 \to -2^7 = -128 \end{cases}$

- *nb ≥ 0 : bit de signe 0, nb<0 : bit de signe 1.*
- *pour représenter un nombre positif sur une mémoire de taille donnée, on complète les cases vides de gauche par des 0 ; pour représenter un nombre négatif sur une mémoire de taille donnée, on complète les cases vides de gauche par des 1 ;*

Exemple

$(+13)_{10}$ sur 8 bits ? $(13)_{10} = (01101)$ → 0 0 0 0 1 1 0 1

$(-9)_{10}$ " ? $(-9)_{10} = (10111)_{c\grave{a}2}$ → 1 1 1 0 1 1 1

II.3 Arithmétique binaire

II.3.1 Addition

L'addition en binaire se fait avec les mêmes règles qu'en

décimal : on commence par additionner les bits de poids faibles ; on a des retenues lorsque la somme de deux bits de même poids dépasse la valeur de l'unité la plus grande (dans le cas du binaire : 1) ; cette retenue est reportée sur le bit de poids plus fort suivant.

La table d'addition binaire est la suivante : $A+B=C$

A	B	C	R (retenue, carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

7,75 + 5,25 = 13,00

0111,11 + 0101,01 = 1101,00 < 0 ? Le résultat est incorrect car on a dépasser la capacité on ne peut pas représenter 13 dans 4 bits.

Exemple

+3 / -6 / -3

$(3)_{10} = (0011)$ $(6)_{10} = (0110)$ ca1 → 1001 → $(1010)_{ca2} = (-6)$

0011 + 1010 = 1101 < 0

ca1 → 0010 → 0011 = (3)10 → $(1101)_{ca2} = (-3)_{10}$

La retenue de la deuxième opération indique un dépassement de capacité (*overflow*) : le bit de signe est à 1 alors qu'il aurait dû être à 0 (addition de deux nombres positifs).

II.3.2 Soustraction

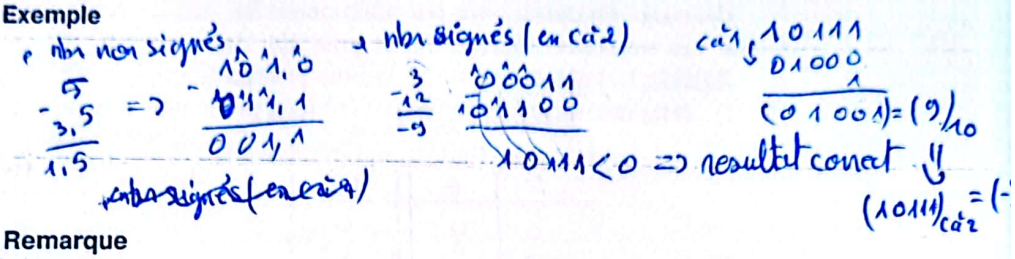
Dans la soustraction binaire, on procède comme en décimal. Quand la quantité à soustraire est supérieure à la quantité dont on soustrait, on emprunte 1 au voisin de gauche. En binaire, ce 1 ajoute 2 à la quantité dont on soustrait, tandis qu'en décimal il ajoute 10.

La table de soustraction binaire est la suivante : $A-B=C$

A	B	C	R (retenue, Borrow)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

CamScanner

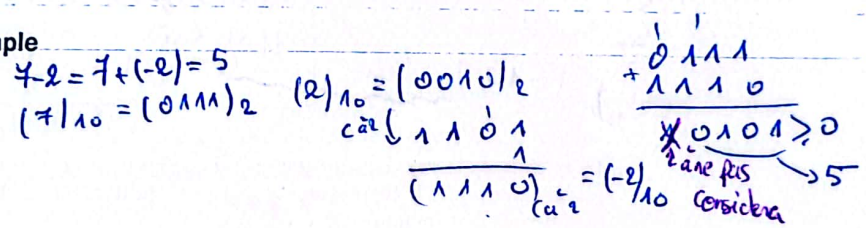
Exemple



▸Remarque

On peut utiliser le complément à 2 de la valeur à soustraire puis on additionne. Dans ce cas, il faut complémenter le retenue (carry) *pour obtenir la retenue soustractive* (borrow). *Cela se passe de cette manière dans certains calculateurs.*

Exemple



On ne tient pas compte de la retenue. (1) à ne pas considérer

II.3.3 Multiplication

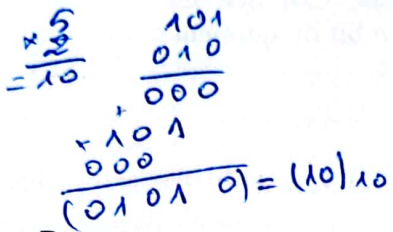
La table de multiplication en binaire est très simple : A x B = C

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

La multiplication se fait en formant un produit partiel pour chaque digit du multiplieur (seuls les bits non nuls donneront un résultat non nul). Lorsque le bit du multiplieur est nul, le produit partiel est nul, lorsqu'il vaut un, le produit partiel est constitué du multiplicande décalé du nombre de positions égal au poids du bit du multiplieur.

CamScanner

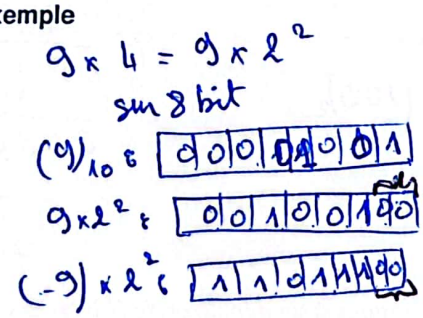
Exemple



▸ **Remarque**

La multiplication binaire par 2^N, se résume à un décalage de N bits vers la gauche. On introduira donc à droite N zéro.

Exemple



II.3.4 Division

La table de division binaire est la suivante :

A		B		C
0	/	0	=	impossible
0	/	1	=	0
1	/	0	=	impossible
1	/	1	=	1

CamScanner

La division binaire s'effectue à l'aide de soustractions et de décalages, comme la division décimale, sauf que les digits du quotient ne peuvent être que 1 ou 0. Le bit du quotient est 1 si on peut soustraire le diviseur, sinon il est 0.

Exemple

Division du nombre $(10010000111)_2$ par $(1011)_2 = (1101001)_2$ reste $(100)_2$, c'est-à-dire 1159/11 105, reste 4.

$(1159)_{10} = (10010000111)_2$

$(10)_{10} = (1011)_2$

$(10010000111)_2 / (1011)_2$

$= (01101001)_2$ et reste $(100)_2$

```
1001000011 | 1011
-0000      | 01101001
 10010
-1011
 00111
-1011
 00011
-1011
 00001
-1011
 00100
```

▸ **Remarque**

La division binaire par 2^N, se résume à un décalage de N bits vers la droite. En arithmétique signée, il faut penser à recopier à gauche le bit de signe autant de fois que nécessaire.

Exemple

8/4 ; $4 = 2^2$

$(8)_{10} =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

$(8/4)_{10} = (2)_{10} \rightarrow$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

$(-16)_{10} / (4)_{10}$; $4 = 2^2$

$(16)_{10} = (010000)_2$

101111

$+ \quad 1$

110000

$(-16)_{10}$: | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

$(-16)/2^2 = -4$: | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

CamScanner

II.4 En résumé

– La valeur d'un nombre est indépendante de la base dans laquelle il est noté.

– Un nombre binaire peut avoir plusieurs valeurs différentes selon le système de représentation. Soit le nombre binaire $ana_{n-1} \ldots a_1 a_0$. Ce nombre vaut :

- $a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
  en représentation non signée
- $-a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
  en représentation signée complément à 2
- $1 - a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
  en représentation signée complément à 1
- $-1^{a_n} \times (a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0)$
  en représentation module et signe

– Les opérations arithmétiques obéissent en binaire aux mêmes règles qu'en décimal, il suffit juste de se rappeler que la base de numération est 2 et non plus 10.

CamScanner

II.4 En résumé

– La valeur d'un nombre est indépendante de la base dans laquelle il est noté.

– Un nombre binaire peut avoir plusieurs valeurs différentes selon le système de représentation. Soit le nombre binaire $anan{-}1 \ldots a1a0$. Ce nombre vaut :

- $a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
  en représentation non signée
- $-a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
  en représentation signée complément à 2
- $1 - a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
  en représentation signée complément à 1
- $-1^{a_n} \times (a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0)$
  en représentation module et signe

– Les opérations arithmétiques obéissent en binaire aux mêmes règles qu'en décimal, il suffit juste de se rappeler que la base de numération est 2 et non plus 10.

C. Les codes numériques

III.1 Codes numériques pondérés

III.1.1 Code binaire pur

Code pondéré par des puissances de 2. Utilisé en arithmétique binaire. Ses dérivées sont le code octal et le code hexadécimal.

CamScanner

III.1.2 Code DCB (Décimal Codé Binaire)

chaque chiffre décimal (0, 1, . . . , 9) est codé en binaire avec 4 bits. Code pondéré avec les poids 1, 2, 4, 8, 10, 20, 40, 80, 100,... Plus facile pour coder des grands nombre, il est surtout utilisé pour l'affichage des nombres.

▸ **Remarque**

Ne pas confondre DCB et code binaire pur : quand on code selon le code binaire pur on prend le nombre dans son intégralité et on le convertit ; par contre, quand on code en DCB on code chaque chiffre indépendamment les uns des autres.

Exemple

$(137)_{10} = (0001\ 0011\ 0111)_{DCB} = (000100110111)_2$

1 3 7

III.1.2.a Addition en DCB

L'addition de deux nombres codés en DCB ne pose pas de problème tant que le résultat est inférieur ou égal à 9 :

en DCB

$$\begin{array}{r} 02 \\ +\ 05 \\ \hline = 07 \end{array} \Rightarrow \begin{array}{r} 0000\ 0010 \\ +\ 0000\ 0101 \\ \hline 0000\ 0111 \end{array}$$

0 7

Par contre, dès que le résultat est supérieur à 9, il faut apporter une correction en additionnant 6, de manière à obtenir une réponse valide :

$$\begin{array}{r} 06 \\ +\ 04 \\ \hline 10 \end{array} \Rightarrow \begin{array}{r} 0000\ 0110 \\ +\ 0000\ 0100 \\ \hline 0000\ 1010 \\ +\ 0110 \\ \hline 0001\ 0000 \end{array}$$

non valide en DCB

3

La correction est ici évidente, puisque la valeur obtenue est invalide en codage DCB. L'exemple suivant est moins évident :

09 + 08 = 17 => 0000 1001 + 0000 1000 = 0001 0001 résultat incorrect

1 0001 0 0111

Dans ce dernier exemple, la correction est due au fait qu'il a eu débordement sur les 4 bits de poids faible du mot DCB : il faut donc apporter une correction sur ces 4 bits de poids faible.

Note :

- *lorsque le résultat de l'addition est inférieur ou égal à 9, on ne change pas le résultat ;*
- *lorsque le résultat de l'addition est supérieur à 9, on ajoute 6 au résultat pour obtenir la valeur exacte ;*
- *lorsqu'il y a une retenue auxiliaire (ou décimale) (*auxiliary *ou* decimal carry*), on ajoute également 6 au résultat obtenu, même si la valeur est inférieure à 9.*

Les propriétés énoncées ci-dessus pour les chiffres des unités sont évidemment valables pour les dizaines, les centaines, etc. La correction à apporter sera alors (selon les circonstances) 6, 60, 66, etc.

III.1.2.b Soustraction en DCB

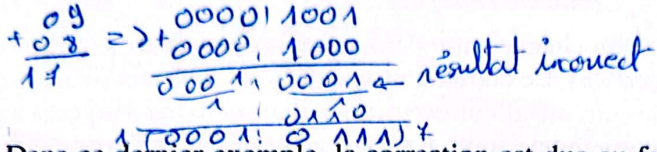
La soustraction en DCB se comporte exactement comme l'addition, au signe près.

Note :

- *lorsque le résultat de la soustraction est inférieur ou égal à 9, on ne change pas le résultat ;*
- *lorsque le résultat de la soustraction est supérieur à 9, on soustrait 6 au résultat pour obtenir la valeur exacte ;*
- *lorsqu'il y a une retenue soustractive (*borrow*), on soustrait également 6 au résultat obtenu, même si la valeur est inférieure à 9.*

CamScanner

La correction est ici évidente, puisque la valeur obtenue est invalide en codage DCB. L'exemple suivant est moins évident :



Dans ce dernier exemple, la correction est due au fait qu'il a eu débordement sur les 4 bits de poids faible du mot DCB : il faut donc apporter une correction sur ces 4 bits de poids faible.

Note :

- *lorsque le résultat de l'addition est inférieur ou égal à 9, on ne change pas le résultat ;*
- *lorsque le résultat de l'addition est supérieur à 9, on ajoute 6 au résultat pour obtenir la valeur exacte ;*
- *lorsqu'il y a une retenue auxiliaire (ou décimale) (* auxiliary *ou* decimal carry*), on ajoute également 6 au résultat obtenu, même si la valeur est inférieure à 9.*

Les propriétés énoncées ci-dessus pour les chiffres des unités sont évidemment valables pour les dizaines, les centaines, etc. La correction à apporter sera alors (selon les circonstances) 6, 60, 66, etc.

III.1.2.b Soustraction en DCB

La soustraction en DCB se comporte exactement comme l'addition, au signe près.

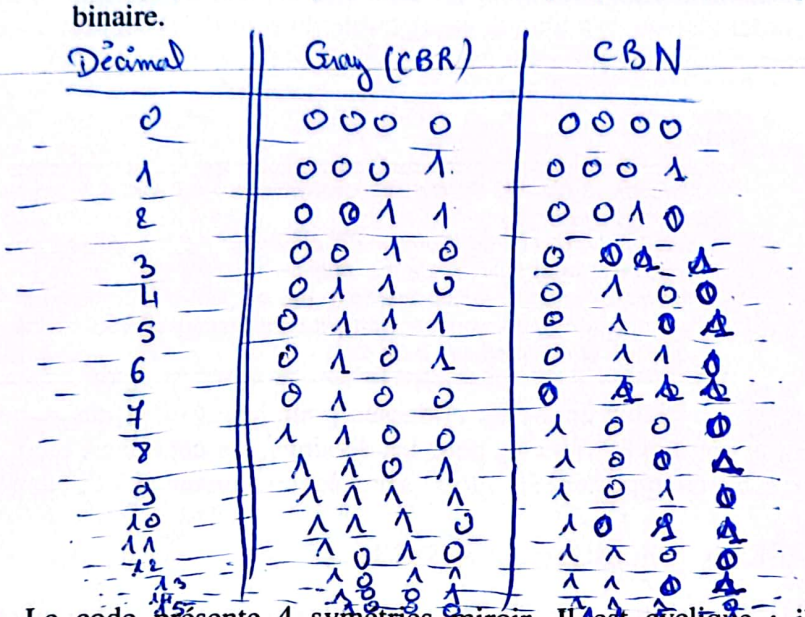
Note :

- *lorsque le résultat de la soustraction est inférieur ou égal à 9, on ne change pas le résultat ;*
- *lorsque le résultat de la soustraction est supérieur à 9, on soustrait 6 au résultat pour obtenir la valeur exacte ;*
- *lorsqu'il y a une retenue soustractive (* borrow*), on soustrait également 6 au résultat obtenu, même si la valeur est inférieure à 9.*

CamScanner

III.2 Codes numériques non pondérés : *Code de Gray (binaire réfléchi)*

Un seul bit change entre deux nombres consécutifs (notion d'adjacence). Ce code est utilisé dans les tableaux de Karnaugh, dans des circuits d'entrée/sortie et dans certains convertisseurs analogique/numérique. Il ne convient pas pour l'arithmétique binaire.



Décimal	Gray (CBR)	CBN
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 1	0 0 1 0
3	0 0 1 0	0 0 1 1
4	0 1 1 0	0 1 0 0
5	0 1 1 1	0 1 0 1
6	0 1 0 1	0 1 1 0
7	0 1 0 0	0 1 1 1
8	1 1 0 0	1 0 0 0
9	1 1 0 1	1 0 0 1
10	1 1 1 1	1 0 1 0
11	1 1 1 0	1 0 1 1
12	1 0 1 0	1 1 0 0
13	1 0 1 1	1 1 0 1
14	1 0 0 1	1 1 1 0
15	1 0 0 0	1 1 1 1

Le code présente 4 symétries miroir. Il est cyclique : il se referme sur lui-même.

III.2.1 Conversion du code binaire naturel vers binaire réfléchi

Pour convertir un nombre en code binaire naturel (CBN) vers un nombre en code binaire réfléchi (CBR), il faut ajouter le CBN trouvé à lui-même décalé d'un rang vers la gauche, sans tenir compte de l'éventuelle retenue et en abandonnant dans le résultat le bit de poids faible.

CamScanner

→ addition sans tenir compte de retenue
→ comparaison si (=) → 0
(≠) → 1

Exemple

$(87)_{10} = (1010111)_{CBN} = (1111100)_{CBR}$

1010111
1010111
1111100

$(1010)_{CBN} = (?)_{CBR}$

1010
1010
1111

$(1010)_{CBN} = (1111)_{CBR}$

méthode pratique : $(1010)_{CBN} = (?)_{CBR}$

1010

1 1 1 1

III.2.2 Conversion du code binaire réfléchi vers binaire naturel

méthode pratique :

$(1111)_{CBR} = (?)_{CBN}$

1111

1 0 1 0

$\Rightarrow (1111)_{CBR} = (1010)_{CBN}$

$(0100)_{CBR} = (?)_{CBN}$

0100

0 1 1 1

$\Rightarrow (0100)_{CBR} = (0111)_{CBN}$

CamScanner