Chapitre 13

Codage et représentation des données

13.1 Le binaire

Les ordinateurs et avant les calculateurs numériques ont été conçus autour des propriétés de la logique booléenne. La logique booléenne ou algèbre de Boole est une logique qui ne traite que des variables ne pouvant prendre que deux états (vrai/faux, 0/1, haut/bas, appuyé/relâché,...).

Un calculateur ne peut traiter que des valeurs binaires (vrai/faux, 0/1). Nous allons voir comment représenter les différentes informations (nombre — entiers naturels ou relatif, réels —, lettres, mots, son,...).

13.2 Représentation des nombres entiers - La base 2

Je vous invite à regarder la vidéo sur la numération vue par les shadoks :
https://www.youtube.com/watch?v=lP9PaDs2xgQ[1].

Les shadoks utilisent la base 4 avec les 4 symboles — Ga (0), Bu (1), Zo (2), Meu (3) —, nous utilisons naturellement la base 10 mais aussi les systèmes duodécimal (base 12) et sexagésimal (base 60) pour la mesure du temps.

Pour le système de numérotation Shadoks comme pour le nôtre, le principe d'écriture des nombres est la *numération positionnelle*, dans lequel c'est la position du symbole graphique qui donne sa valeur, ce système se différencie de la *numérotation additive* dans lequel c'est la forme du symbole qui donne sa valeur (numérotation égyptienne).

13.2.1 Principe de la numérotation positionnelle

Soit un nombre en base 10 : 123456.

Ce nombre se lit en tenant compte de la base et de la position (figure 13.1).

Chaque chiffre du nombre associé à son rang (de 0 à n-1) permet d'obtenir le poids correspondant dans le nombre, ainsi le chiffre 4 de rang 2 a une valeur de $4 \cdot 10^2 = 400$ dans le nombre total.

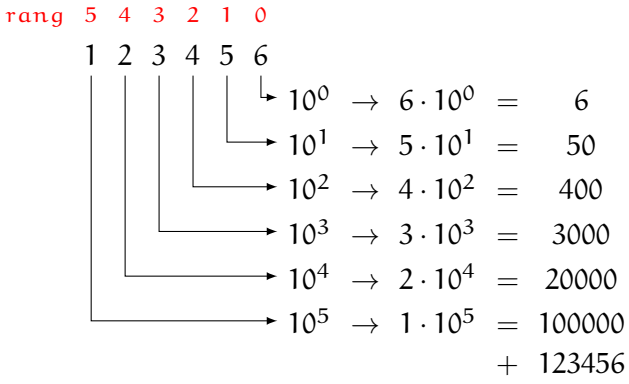


FIGURE 13.1 – Principe de la numération de position en base 10

1. Si le lien ne fonctionne pas, une recherche avec les mots clefs « Shadoks, base 4 » devrait retrouver la vidéo.

13.2.2 Transcodage base n vers base 10

À partir de cette description, il possible de convertir un nombre écrit dans une base n dans la base 10. La figure 13.2 décrit le principe du passage de la base Shadok (base 4) à la base 10.

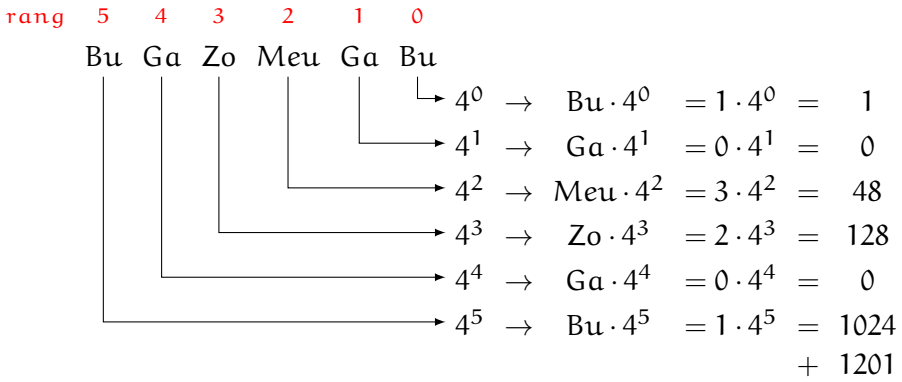


FIGURE 13.2 – Transcodage base 4 (Shadok) → base 10

Cette méthode permet bien sûr de passer de la base 2 à la base 10 (figure 13.3).

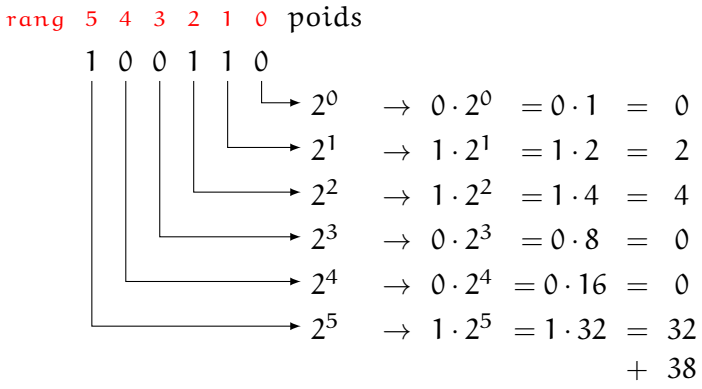
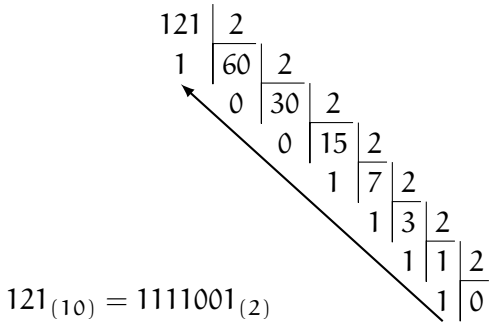


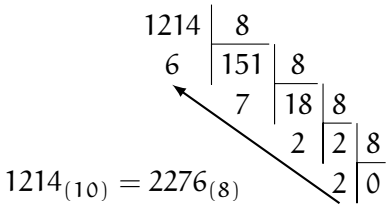
FIGURE 13.3 – Transcodage base 2 → base 10

13.2.3 Transcodage base 10 vers base n

Pour passer de la base 10 vers la base 2 il suffit de réaliser des divisions euclidiennes (division entière) successives du nombre à convertir puis du quotient de cette division jusqu'à ce que le quotiient soit nul. Le nombre binaire se lit alors du dernier reste vers le premier (figure 13.4a).



(a) de la base 10 à la base 2

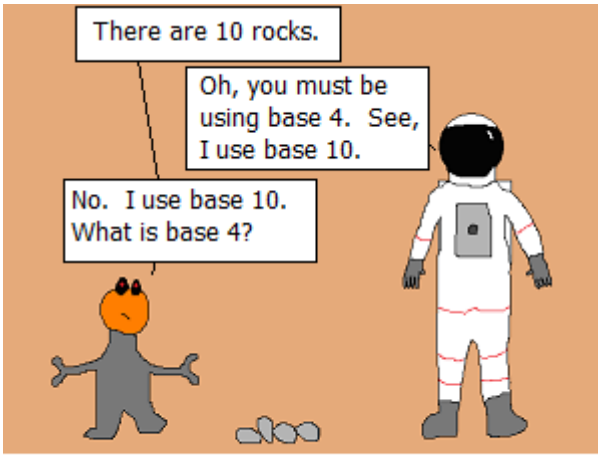


(b) de la base 10 à la base 8

FIGURE 13.4 – Transcodage

Cette procédure permet de convertir un nombre écrit en base 10 dans n'importe quelle base, il suffit pour cela de diviser par le nombre de base (figure 13.4b).

Si on souhaite utiliser une base supérieure à 10, il est nécessaire d'utiliser de nouveaux symboles pour décrire les chiffres supérieurs à 9, par convention, on utilise les lettres de l'alphabet, $10 \rightarrow A$, $11 \rightarrow B$, $12 \rightarrow C$,...



Every base is base 10.

13.2.4 Base 2 - Code binaire naturel

Les règles des quatre opérations de base (addition, soustraction, multiplication et division) restent exactement les mêmes qu'en notation décimale.

Pour une addition de deux variables binaires a et b on pose S, la somme et R la retenue. a et b ne peuvent prendre que les deux valeurs binaires 0 et 1, on peut donc poser les quatre opérations suivantes :

R	0	0	0	1
a	0	1	0	1
+ b	+ 0	+ 0	+ 1	+ 1
= R S	= 0 0	= 0 1	= 0 1	= 1 0

Les règles de la soustraction sont analogues tant que le résultat reste un nombre positif, nous verrons plus loin le codage des nombres négatifs.

La multiplication par 2 revient à décaler le nombre binaire vers la gauche et rajouter un 0

		1	0	1	1
×				1	0
=	1	0	1	1	0

Pour la division un décalage à droite correspond à une division par deux, le résultat est exact si le dividende est un entier pair.

13.2.5 La base 16-hexadécimal

La base 16 utilise les caractères numériques de la base 10 de 0 à 9 et est complétée par les caractères alphabétiques A, B, C, D, E, F afin d'obtenir les 16 caractères nécessaires pour écrire un nombre en base 16 (figure 13.1).

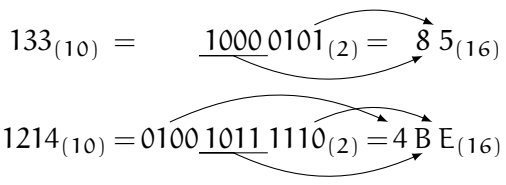


FIGURE 13.5 – Transcodage base 2 → base 16

Le système hexadécimal (la base 16) est très souvent utilisé en informatique car il permet une représentation rapide et courte des nombres binaires. En effet, on peut en regroupant les bits du nombre en base 2 par 4 en partant de la gauche et en complétant par des 0 à droite si nécessaire pour avoir un multiple de 4 bits, obtenir directement le code hexadécimal en remplaçant chaque groupe de 4 bits par le code hexadécimal correspondant (tableau 13.5).

base																
10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 13.1 – Tableau de conversion base 10, base 2, base 16

13.2.6 Code Gray

Le code Gray est un code binaire qui possède comme particularité de n'avoir qu'un bit qui change entre deux valeurs entières successives.

Du code binaire naturel au code Gray

Voir l'exercice Transcodage page 53 pour le principe du transcodage mais il est possible de déterminer directement le code Gray d'un code binaire naturel.

$$N_g = \frac{N_b \oplus 2 \cdot N_b}{2}$$

- Écrire le nombre en binaire naturel.
- Multiplier le nombre par 2 (décalage vers la gauche de chaque bit).
- Faire un OU exclusif bit à bit entre le nombre binaire et le nombre multiplié par 2.
- Diviser par 2 (décalage à droite de chaque bit) pour obtenir le code Gray :

Ainsi pour $56_{10} = 0111000_2$:

$$\begin{aligned} N_b &= 0111000 \\ 2 \cdot N_b &= 1110000 \\ N_b \oplus 2 \cdot N_b &= 1001000 \\ N_G = \frac{N_b \oplus 2 \cdot N_b}{2} &= 0100100 \end{aligned}$$

Décimal	Binaire pur				Binaire réfléchi			
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Table 13.2 – Code binaire naturel et code Gray

Ce n'est pas un code pondéré, il n'est donc pas possible de l'utiliser pour des calculs numériques. Il est principalement utilisé à chaque fois que l'on a besoin de réaliser un capteur absolu.

Capteur absolu : Avec ce type de capteur, la position est connue de manière absolue, en cas de coupure de courant, lors du redémarrage, la lecture de la position sur le capteur donne la position. Ce type de capteur se différencie des codeurs relatifs ou incrémentaux, pour lesquels, la position est connue uniquement par comptage des impulsions par rapport à une position de référence (origine).

Le disque de la figure 13.6 représente le disque d'un codeur de position sur 10 bits qui permet de différencier $2_{10} = 1024$ positions.

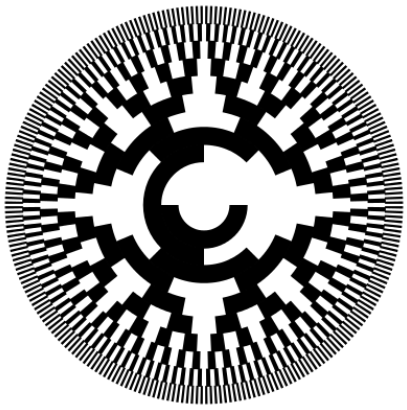
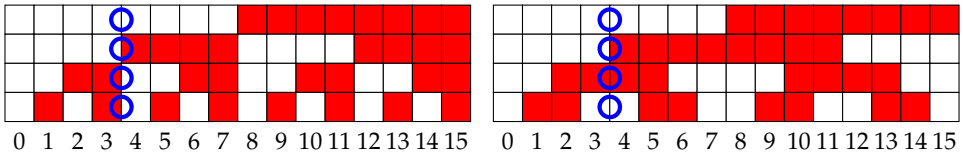


Figure 13.6 – Disque Gray 10 bits

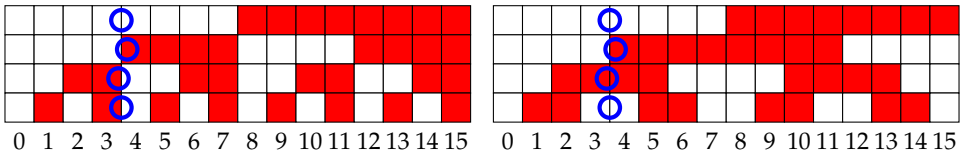
Ce code permet entre autres de réaliser des codeurs de position absolue sans aléas de commutation contrairement au codage absolu réalisé avec le code binaire naturel.

On remarque sur la figure 13.7 que lors d'un changement de position (de la position 3 à la position 4) 3 bits changent de valeur simultanément avec le codage binaire naturel alors qu'un seul change pour le codage Gray. Si le fonctionnement de tous les composants est parfait, alors dans les deux cas la transition s'effectue sans problèmes, or il est peu probable que les composants soient parfaits, au moindre défaut comme un décalage des capteurs, la position lue ne sera plus correcte. Cela ne se produit pas avec la règle utilisant le code Gray.



(a) Codage binaire naturel « parfait »

(b) Codage Gray « parfait »



(c) Codage binaire naturel « avec défaut »

(d) Codage Gray « avec défaut »

FIGURE 13.7 – Positionnement absolu linéaire

Tableau de Karnaugh

Le tableau de Karnaugh est généré à partir du code Gray.

On remarque sur la figure 13.8 que les différentes valeurs du code Gray de 0 à 15 sont toutes chaînées et obtenues en passant d'une case adjacente à l'autre. La dernière valeur 15 est adjacente à la première. On constate donc que sur un nombre fini de bits, le code Gray est un code cyclique.

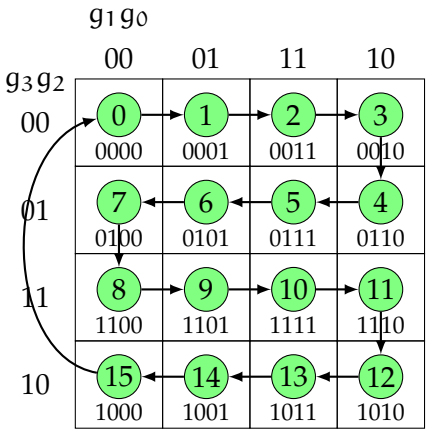


FIGURE 13.8 – Tableau de Karnaugh et code Gray

a) Code BCD

Travailler sur des nombres en binaire naturel est intéressant dans les calculateurs, car ces nombres sont pondérés, mais lorsqu'on veut une image rapide de l'équivalent décimal, on est amené à effectuer un transcodage long et fastidieux. Il est plus commode dans certaines applications, comme par exemple l'affichage en décimal du contenu de valeurs, d'utiliser la représentation BCD.

Le BCD (Binary Coded Decimal, ou Décimal Codé en Binaire en français DCB) est le code décimal le plus utilisé en électronique. Il contient des mots-code qui sont la traduction en binaire naturel (sur 4 bits) de chacun des dix chiffres du système décimal. Chaque élément binaire d'un mot-code a un poids comme en binaire naturel : 8 4 21. Le BCD est donc un code pondéré.

Décimal	BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

TABLE 13.3 – Tableau BCD

Conversion décimal/BCD

La conversion entre un nombre décimal et son code BCD est réalisée terme par terme, à chaque terme, on associe la tétrade (4 bits) correspondante en code binaire naturel. La conversion dans l'autre sens est réalisée en regroupant les bits 4 par 4 en partant du poids le plus faible (à droite) et en rajoutant si nécessaire des zéros à gauche. Chaque tétrade est ensuite convertie en décimal.

Le code BCD n'est pas un code optimal, il utilise 4 bits soit $2^4 = 16$ combinaisons pour coder 10 valeurs.

Décimal	2	9	5
	↕	↕	↕
BCD	0010	1001	0101

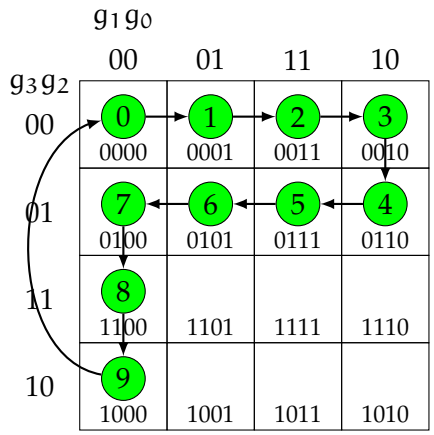
FIGURE 13.9 – Conversion Décimal / BCD

b) Code de Glixon

Le code de Glixon est un code décimal cyclique qui possède la propriété du code Gray de ne changer que d'un seul bit entre deux valeurs successives mais aussi d'être cyclique sur 10 valeurs, la valeur 9 précède la valeur 0.

On peut retrouver le code de Glixon à partir du code Gray, il est identique jusqu'à la valeur 8, seule la valeur 9 diffère, elle est telle que entre 8 et 9 et entre 9 et 0 un seul bit varie.

On peut s'appuyer sur le tableau de Karnaugh à 4 variables pour le déterminer.



(a) Tableau de Karnaugh et code de Glixon

Décimal	BCD	Glixon
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 0 0 0

(b) Code BCD et de Glixon

FIGURE 13.10 – Code Glixon

c) Code p parmi n

Les codes « p parmi n » correspondent à des codes pour lesquels n bits sont à 1 parmi n bits.

On distingue plusieurs codes « p parmi n »,

Les codes « 2 parmi 5 », les mots-code comprennent 5 bits dont 2 sont à 1 (et les 3 autres à 0). Il existe plusieurs codes « 2 parmi 5 », et les plus utilisés sont le code « 8 4 2 1 0 » et le code « 7 4 2 1 0 ». Ces deux codes sont pondérés, la liste des poids figurant dans la dénomination du code.

Les codes « 2 parmi 5 » font partie des codes spécialement conçus pour la transmission de l'information et pour la détection des erreurs. En effet, si on reçoit un nombre codé en « 2 parmi 5 », pour détecter une éventuelle erreur dans ce nombre il suffit de compter le nombre de 1 logique présent dans chacun des groupes de 5 bits. Si un groupe ne présente pas deux 1 logiques, on peut en déduire avec certitude qu'il est erroné.

Remarque : contrairement à d'autres codes plus perfectionnés , les codes « 2 parmi 5 » permettent de détecter une erreur, mais ne permettent pas de la corriger. De plus, si lors de la transmission, 2 bits de valeurs différentes changent simultanément d'état, aucune erreur ne pourra être détectée à l'arrivée.

Décimal	Code « 2 parmi 5 » 8 4 2 1 0	Code « 2 parmi 5 » 7 4 2 1 0
0	1 0 1 0 0	1 1 0 0 0
1	0 0 0 1 1	0 0 0 1 1
2	0 0 1 0 1	0 0 1 0 1
3	0 0 1 1 0	0 0 1 1 0
4	0 1 0 0 1	0 1 0 0 1
5	0 1 0 1 0	0 1 0 1 0
6	0 1 1 0 0	0 1 1 0 0
7	1 1 0 0 0	1 0 0 0 1
8	1 0 0 0 1	1 0 0 1 0
9	1 0 0 1 0	1 0 1 0 0

TABLE 13.4 – Tableau de conversion Décimal vers « 2 parmi 5 »

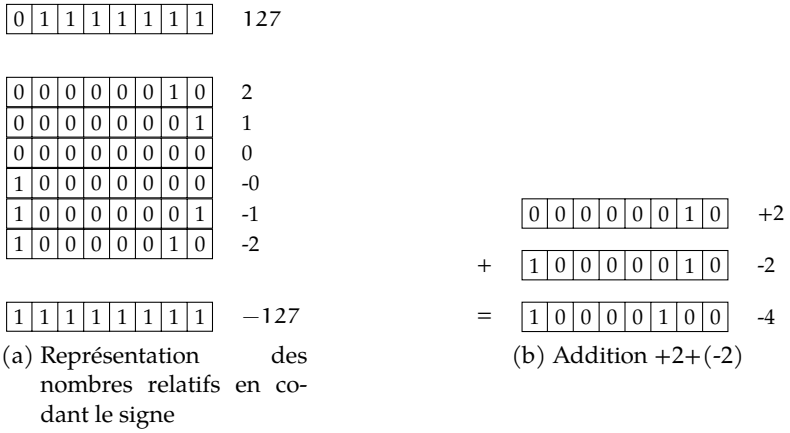
Le code « 3 parmi 5 » est utilisé par la poste pour imprimer le code barre nécessaire aux lecteurs optiques.

13.3 Représentation des nombres entiers relatifs

La première solution envisageable pour coder un entier relatif est de dédier un bit pour le codage du signe puis de coder sur les autres bits la valeur absolue. La figure 13.11a présente le codage des nombres relatifs de -127 à +127 sur 8 bits, le signe est codé sur le bit de poids fort.

Cette représentation, et c'est le cas pour toutes les représentations des nombres relatifs, nécessite de préciser sur combien de bits doit être fait le codage du nombre. On remarquera aussi que **0** a deux représentations possibles.

Cette représentation, si elle est facile à mettre en œuvre, ne permet pas d'utiliser les règles de l'addition binaire pour obtenir un résultat correct (figure 13.11b). Ce codage n'est en général pas utilisé, à cause des problèmes d'arithmétique qu'il pose.



(a) Représentation des nombres relatifs en codant le signe

(b) Addition +2+(-2)

FIGURE 13.11 – Représentation des nombres relatifs en codant le signe + valeur absolue

On constate donc qu'un codage doit respecter plusieurs critères :

— les entiers positifs doivent respecter le code binaire naturel (base 2)

— les règles de l'addition binaire doivent être respectées, aussi bien pour les nombres positifs que les négatifs.

Pour cela :

— le codage des nombres est défini sur un nombre **n** de bits donnés (8, 16, 32,...) ;

— les nombres entiers positifs sont codés en code binaire naturel sur **n-1** bits (7, 15, 31,...) ;

— les nombres entiers négatifs sont codés en complément à 2.

> Le code complément à deux du nombre entier négatif $-a$ sur n bits s'obtient en codant en binaire naturel le nombre $2^n - a$.

Soit le nombre $a = 81$, déterminons le code complément à 2 de $b = -a$ sur 8 bits.

a=81	0	1	0	1	0	0	0	1
$b = 2^8 - 81 = 175$	1	0	1	0	1	1	1	1

Vérifions maintenant que $a + b = 0$:

a		0	1	0	1	0	0	0	1
b	+	1	0	1	0	1	1	1	1
R	=	1	0	0	0	0	0	0	0

On obtient $R = 100000000_2$ en binaire sur 9 bits, si on considère que le résultat est sur 8 bits (on ne prend pas en compte la retenue qui déborde les 8 bits) alors $R = 00000000_2$. Le résultat est correct.

Quelques remarques :

— Dynamique du codage : le codage en complément à deux permet de coder sur 8 bits les nombres relatifs de -128 à + 127 soit en généralisant sur n bits de $[-2^{n-1}, 2^{n-1} - 1]$.

— L'entier 0 s'écrit $0000\ldots0000_{C2}$.

— L'entier -1_{10} s'écrit $1111\ldots1111_{C2}$.

— Tous les nombres positifs commencent par un 0, le plus grand entier positif sur n bits s'écrit : $01111\ldots1111_{C2} = +2^{n-1} - 1$.

— Tous les nombres négatifs commencent par un 1, le plus petit entier négatif sur n bits s'écrit : $100\ldots.00_{C2} = -2^{n-1}$.

13.3.1 Les limites du codage des nombres relatifs

Les limites du codage des nombres relatifs sont principalement dues à l'overflow (dépassement de capacité), lors d'un calcul, une addition de deux nombres positifs ou négatifs peut entraîner un dépassement de capacité, celui-ci peut être détecté en regardant le signe du résultat par rapport au signe des deux opérandes (deux nombres positifs donnent un résultat négatif et réciproquement).

Remarque : le dépassement de capacité est un problème important des langages informatiques, dans le cas de Python, celui-ci augmente dynamiquement la place occupée par les variables numériques en fonction des calculs.

13.4 Représentation des réels

Avec les réels, se posent de nouveaux problèmes de codage :

— Comment représenter les nombres à virgule ?

— Quelles sont les limites de la représentation des réels ?

13.4.1 Les nombres à virgule

Soit un nombre réel $r = 57{,}625$, le codage de la partie entière ne pose pas de problèmes particuliers, il suffit de convertir en binaire naturel :

$57_{10} = 00111001_2$ avec la méthode par divisions successives. Pour la partie fractionnaire 0,625 il est nécessaire d'adapter la procédure.

On sait que :

$$0{,}625 = 6 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3}$$

On retrouve la même forme que pour la partie entière hormis que les puissances de 10 sont négatives. On peut par analogie écrire en utilisant les puissances négatives de 2 ce qui permet de déduire le code binaire

$$\begin{aligned} 0{,}625_{10} &= 1 \times 0{,}5 + 0 \times 0{,}25 + 1 \times 0{,}125 \\ &= 1 \times \frac{1}{2} + 0 \times \frac{1}{4} + 1 \times \frac{1}{8} \\ &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \end{aligned}$$

$$\text{finalement : } 0{,}625_{10} = 0{,}101_2.$$

La méthode ci-dessus n'est pas systématique, il est préférable d'utiliser la procédure ci-dessous :

1. multiplier la partie fractionnaire par 2,
2. conserver la partie entière du résultat,
3. si la partie fractionnaire est nulle (ou si la précision est suffisante) construire le code binaire en concaténant les parties entières dans l'ordre du calcul,
4. sinon recommencer du début.

Pour $0{,}312\,5_{10}$:

				partie entière		partie fractionnaire
0,312 5 × 2 =	0,625 0	=	0	+	0,625
0,625 × 2 =	1,250	=	1	+	0,250
0,250 × 2 =	0,500	=	0	+	0,500
0,500 × 2 =	1,000	=	1	+	0,000

code binaire $0{,}312\,5_{10} = 0{,}010\,1_2$

Il est souvent impossible d'obtenir le code binaire exact (avec un nombre fini de termes) d'un nombre relatif, il est ainsi impossible de coder 0,1 en binaire. Il est donc nécessaire de limiter le calcul jusqu'à une précision donnée.

Tous les calculs sur les nombres réels sont nécessairement arrondis.

Savoir coder la partie fractionnaire d'un nombre à virgule ne suffit pas pour coder tous les nombres à virgule en binaire. Il reste à définir comment coder la virgule, en effet, il est facile d'écrire sur une feuille

$$3{,}312\,5_{10} = 11{,}010\,1_2$$

mais pour un système informatique cela n'a pas de sens.

La solution consiste à écrire le chiffre sous la forme scientifique :

$$-3{,}312\,5 = -0{,}331\,25 \times 10^1$$

Sous cette forme on constate qu'il suffit de mémoriser les chiffres après la virgule, la mantisse, le signe et l'exposant pour avoir une représentation du nombre dans la base 10 puisque tous les nombres vont commencer par 0.

C'est cette méthode que l'on va adapter pour coder les réels en binaire naturel.

Pour convertir un nombre réel, il faut au préalable l'écrire sous la forme

$$\text{s1,mmmm} \times 2^{\text{eee}}$$

avec s le signe, m, la mantisse et e l'exposant.

Par exemple, pour une représentation sur 8 bits le format est le suivant :

s	e	e	e	m	m	m	m

.

La procédure est la suivante :

1. Convertir le nombre réel la partie entière et la partie fractionnaire séparément sans tenir compte du signe
$$-2{,}25_{10} = -10{,}11$$
2. Décaler le nombre binaire obtenu pour le mettre sous la forme s1,mmmm $\times 2^{eee}$
$$-2{,}25_{10} = -1{,}011 \times 2^{1}$$
3. Le nombre réel est codé avec les conventions suivantes :
   — le signe est codé en binaire positif=0, négatif=1, ici $s = 1$,
   — le **1** avant la virgule n'est pas codé (puisqu'il est toujours présent),
   — à l'exposant est ajoutée la valeur $2^{3-1} - 1 = 3$ puis codé sur 3 bits (ici : $e + 3 = 4 = 100$),
   — la mantisse est complétée à droite par des 0 : $m = 0110$.
4. On obtient finalement :

1	1	0	0	0	1	1	0

5. Pour prendre en compte les exposants négatifs, on choisit non pas de coder l'exposant en complément à deux, mais plutôt de décaler la valeur de $+2^{n-1} - 1$ (avec n, le nombre bits pour coder de l'exposant). Par convention les valeurs extrêmes de l'exposant 000...000 et 111...111 ne sont pas utilisés et sont réservés pour des valeurs particulières.
6. Il n'est pas possible avec cette représentation de coder le nombre **num0.0** pour résoudre ce problème, on choisit de le représenter avec l'exposant nul et la mantisse nulle.
   0,0 se code

0	0	0	0	0	0	0	0

   mais aussi

1	0	0	0	0	0	0	0

   sur 8 bits.
7. L'exposant 111...111 est utilisé pour représenter $+\infty$ (ou $-\infty$ si le signe est négatif).
8. Pour passer du code binaire à la représentation décimale, on utilisera la formule suivante :
$$(-1)^s \times 2^{e-(2^{n-1}-1)} \times (1 + m)$$
   avec e : l'exposant, m : la mantisse, s : le signe et n : le nombre de bits de l'exposant.

13.5 Représentation des caractères

13.5.1 Code ASCII

Le principe est simple, à un caractère il suffit de faire correspondre un ensemble de bits. Cela serait élémentaire si tout le monde écrivait avec l'alphabet anglais ! En fait la représentation des caractères est un problème crucial de l'informatique et surtout de la transmission des informations, d'ailleurs un des premiers codes binaires de représentation des caractères créé fut le code Morse permettant de communiquer par les premiers fils télégraphiques des messages textuels.

Le code ASCII (Américan Standard Code for Information Interchange) fut la première norme à définir un codage des caractères pour les systèmes informatiques dans les années 60, ce code à l'origine a été défini sur 7 bits [2] (table 13.5). Ce codage comme son nom l'indique est une norme américaine, ne permet de coder que 95 caractères imprimables (les 32 premiers sont des caractères de contrôles comme DEL : effacer, BS : Back Space, CR : Retour Chariot,...). Ce codage ne permet que d'écrire en anglais (aucun accent).

2. Le 8e bit était utilisé pour un contrôle d'erreur par bit de parité

hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
hex bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0 000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1 001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2 010	SP	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
3 011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4 100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5 101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6 110	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7 111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

NUL : Null
SOH : Start of heading
STX : Start of text
ETX : End of text
EOT : End of transmission
ENQ : Enquiry
ACK : Acknowledge
BEL : Bell
BS : Backspace
TAB : Tabulation horizontale
LF : Line Feed- saut de ligne
VT : Vertical tabulation- tabulation verticale
FF : Form feed
CR : Carriage return- retour à la ligne
SO : Shift out
SI : Shift in
DLE : Data link escape
DC1 : Device control 1
DC2 : Device control 2
DC3 : Device control 3
DC4 : Device control 4
NAK : Negative acknowledgement
SYN : Synchronous idle
ETB : End of transmission block- fin de bloc de transmission
CAN : Cancel- annulation
EM : End of medium- fin du médium
SUB : Substitute- substitut
ESC : Escape- caractère d’échappement
FS : File separator- séparateur de fichier
GS : Group separator- séparateur de groupe
RS : Record separator- séparateur d’enregistrement
US : Unit separator- séparateur d’enregistrement
SP : Space- espace
DEL : Touche de suppression

Table 13.5 – Table ASCII sur 7 bits

13.5.2 Norme ISO-8859-1

Au début chaque fabricant de matériel (IBM, Microsoft [3] et Apple [4]) a proposé une table de caractères plus complète sur 8 bits pour représenter et coder les autres caractères européens. Malheureusement cela rendait impossible la communication directe entre systèmes différents.

La norme ASCII a rapidement évolué vers la norme ISO-8859-1 (appelé aussi latin-1 ou ANSI) pour prendre en compte les spécificités des autres langues européennes (presque !), avec une table qui permet le codage sur 8 bits de 256 caractères de cette table a connu un certain succès car elle a été choisie pour le codage des pages web (pour la norme HTML4). Cette norme a évolué encore récemment vers ISO-8859-15 pour prendre compte en quelques caractères oubliés ou nouveaux comme le symbole Euro €.

Mais si cette norme peut suffire pour la majorité des langues européennes, elle ne permet pas du tout de coder les alphabets orientaux et africains.

13.5.3 Unicode - UTF-8

Pour résoudre ce problème, les différents acteurs du secteur informatique des fabricants de matériel aux éditeurs de logiciel se sont réunis en consortium pour établir un nouvelle norme de codage des caractères. Le consortium Unicode a pour objectif ambitieux de remplacer à terme les codages de caractères existants. Il travaille aux travaux de normalisation de l’ISO dans ce domaine pour la norme ISO/CEI 10646.

Le standard Unicode et la norme ISO/CEI 10646 définissent les caractères sur 4 octets (32 bits) soit la possibilité de coder $2^{32} = 4\,294\,967\,296$ caractères.

Pour éviter d’avoir à coder tous les caractères sur 4 octets (ce qui multiplierait par 4 la place nécessaire au stockage d’un fichier texte), Unicode a prévu un codage UTF-8 qui permet de coder les caractères sur des séquences de 1 à 4 octets.

— Si le 1er octet commence par 0, alors on lit directement la table ASCII :

3. Codage Windows-1252http://fr.wikipedia.org/wiki/Windows-1252
4. Codage MacRoman http://fr.wikipedia.org/wiki/MacRoman

0	x	x	x	x	x	x	x	ainsi A=	0	0	1	0	1	0	1	0

— Si le 1er octet commence par 110, alors il est nécessaire de lire deux octets pour obtenir le code du caractère, l'octet suivant commençant lui par 10 :

1	x	x	x	x	x	x	x	1	0	x	x	x	x	x	x

ainsi le caractère é=

1	1	0	0	0	0	1	1	1	0	1	0	1	0	0	1

Le code du caractère est construit en concaténant les 5 derniers bits du 1er octet et les 6 derniers du 2nd soit 11 bits, ici : 00011101001.

— Si le 1er octet commence par 1110, il faut lire le code sur 3 octets, les deux octets qui suivent commencent par 10 soit 16 bits :

1	1	1	0	x	x	x	x	1	0	x	x	x	x	x	x	1	0	x	x	x	x	x	x

— Si le 1er octet commence par 11110, il faut lire le code sur 4 octets (les trois octets qui suivent commencent par 10), soit 22 bits :

1	1	1	1	0	x	x	x	1	0	x	x	x	x	x	x	1	0	x	x	x	x	x	x	1	0	x	x	x	x	x	x

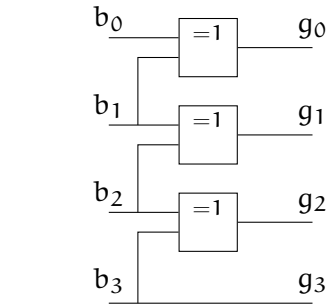
Le codage UTF-8 permet donc de coder $2^{22} = 4\,194\,304$ qui ne sont pas encore tous définis.

13.6 Feuille de travaux dirigés n°13

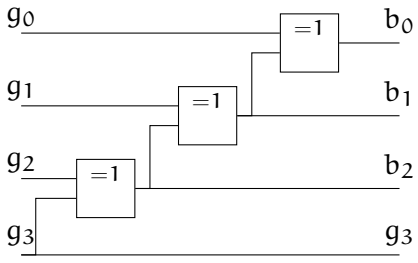
Exercice 7 - Transcodage

Corrigé page 56

Q1. En utilisant le schéma de la figure 13.12a compléter le tableau 13.6. Que fait ce circuit ?
Q2. Déterminer les équations logiques des g_i en fonction de b_i. Proposer une forme générale de la relation.
Q3. Vérifier que le schéma de la figure 13.12b permet la conversion dans l'autre sens.
Q4. Déterminer les équations logiques des b_i en fonction de g_i. Proposer une forme générale de la relation.



(a) Circuit A



(b) Circuit B

FIGURE 13.12 – Circuits de transcodage

Décimal	Binaire pur				- - - - - - - - - -			
N	b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0
0	0	0	0	0	–	–	–	–
1	0	0	0	1	–	–	–	–
2	0	0	1	0	–	–	–	–
3	0	0	1	1	–	–	–	–
4	0	1	0	0	–	–	–	–
5	0	1	0	1	–	–	–	–
6	0	1	1	0	–	–	–	–
7	0	1	1	1	–	–	–	–
8	1	0	0	0	–	–	–	–
9	1	0	0	1	–	–	–	–
10	1	0	1	0	–	–	–	–
11	1	0	1	1	–	–	–	–
12	1	1	0	0	–	–	–	–
13	1	1	0	1	–	–	–	–
14	1	1	1	0	–	–	–	–
15	1	1	1	1	–	–	–	–

TABLE 13.6 – Tableau de conversion à compléter

Exercice 8 - Contrôle de position

Extrait de Mines-Ponts 2002

Corrigé page 56

Présentation

Le déplacement des masses du cogite est contrôlé par un ensemble de trois détecteurs qui lisent les pistes d'un disque de codeur.

Afin de contrôler chaque 1/10 de tour de la poulie, les trois détecteurs lisent 4 pistes angulaires adjacentes situées sur la poulie (noir = 1, blanc = 0) (Figure 13.13a).

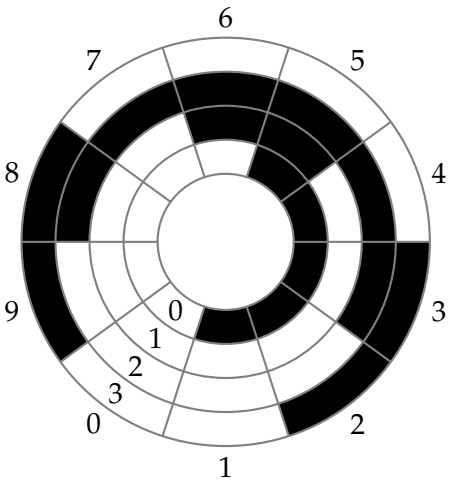
La poulie d'entraînement des câbles a un rayon de 0,795 m.

Ces trois détecteurs : $A(a_0,a_1,a_2,a_3)$, $B(b_o,b_1,b_2,b_3)$ et $C(c_o,c_1,c_2,c_3)$ sont formés de quatre cellules photoélectriques.

Par exemple : a_o, b_o, c_o lisent la même piste. La valeur des bits d_i du détecteur de position $D(d_0,d_1,d_2,d_3)$ se construit à la majorité des valeurs des bits a_i, b_i et c_i des détecteurs A, B et C. Ce système permet au calculateur de gérer les aléas de passage d'une position à une autre, de faciliter la maintenance du système et de réaliser un contrôle de la mesure par surabondance des données.

En cas de désaccord sur un bit en position i, le bit e ; d'un mot $E(e_0,e_1,e_2,e_3)$ est placé à 1.

Q1. Combien de tours correspondent à un déplacement de 32 m ? Quelle sera alors la précision de la mesure ?
Q2. La position est-elle connue de manière absolue ou relative ? Que peut-on faire pour obtenir une position absolue sur la totalité du déplacement ?
Q3. Compléter le tableau de la figure 13.13b.



(a) Codeur

d_3	d_2	d_1	d_0	secteur	k_3	k_2	k_1	k_0
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1
1	0	0	1	2	0	0	1	0
				3	0	0	1	1
				4	0	1	0	0
				5	0	1	0	1
				6	0	1	1	0
				7	0	1	1	1
				8	1	0	0	0
				9	1	0	0	1

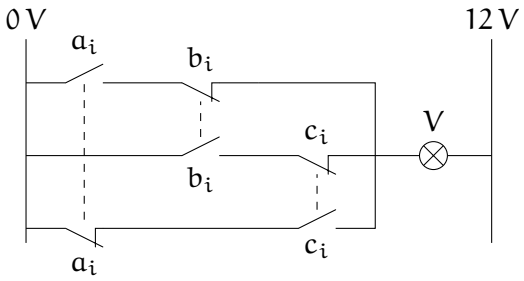
(b) Tableau

FIGURE 13.13 – Codeur optique de position

Q4. Déterminer l'expression de $d_i = f(a_1, b_i, c_i)$. Câbler d_i en schéma avec des portes logiques NON, ET et OU.

Q5. Montrer que $e_1 = g(a_i, b_i, c_i)$ peut s'écrire $e_i = a_i \oplus b_i + b_i \oplus c_i + c_i \oplus a_i$.

On souhaite installer un voyant V sur le tableau de commande. Le schéma proposé est le suivant :



Q6. Vérifier que ce schéma permet l'éclairage du voyant dès que
$e_i = 1$.

On désire afficher la valeur lue par D sur un pupitre indépendant du calculateur (en cas de dysfonctionnement de celui-ci). Pour cela, il est nécessaire de transcoder D en binaire naturel (soit K ce mot de 4 bits), déterminez K : (k_0,k_1,k_2,k_3) en fonction de (d_0,d_1,d_2,d_3) (Figure 13.13b).

Q7. Écrire ces fonctions de la manière la plus condensée possible.

Q8. On se propose d'affiner la précision en utilisant un codage sur 20 secteurs angulaires. Proposez une extension du codage précédent en complétant le tableau ci dessous.

nb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
d_0	0	1	1	1	1	1	0	0	0	0										
d_1	0	0	0	0	0	1	1	0	0	0										
d_2	0	0	0	1	1	1	1	1	1	0										
d_3	0	0	1	1	0	0	0	0	1	1										

Exercice 9 - Horloge binaire

Corrigé page 58

Sur l'horloge murale de la figure 13.14a, les heures et les minutes sont codées graphiquement.

Q1. Sur combien de bits sont codées les heures et les minutes. Quel est le code utilisé ?

Q2. Est-il possible de coder les heures et minutes sur moins de bits ?

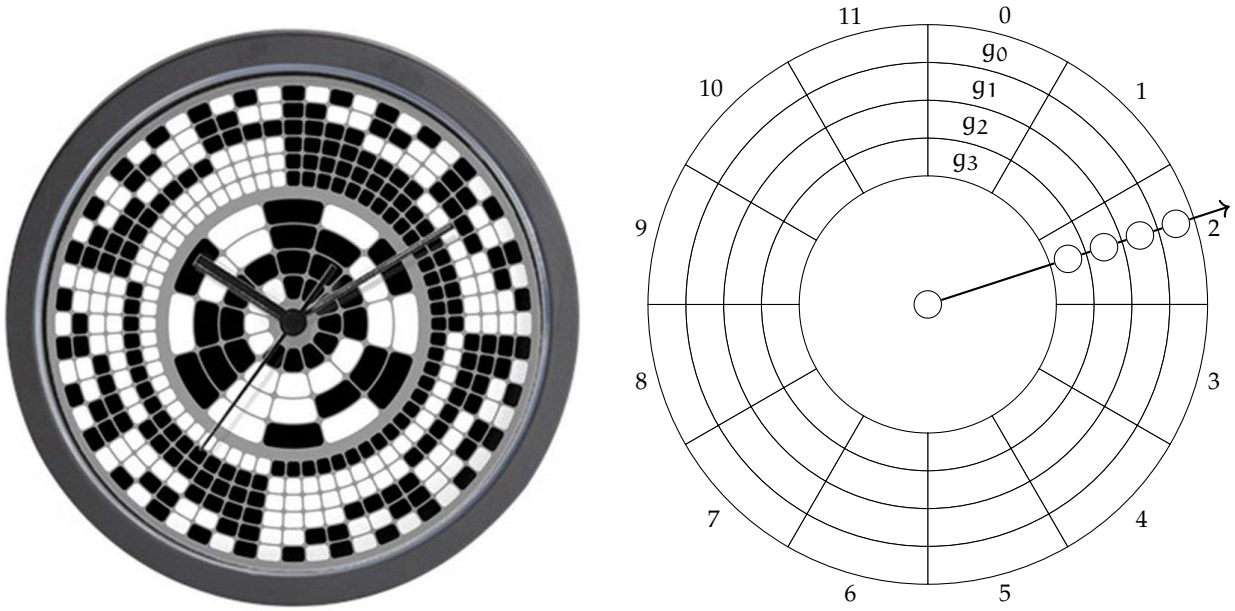
Q3. Donner l'heure indiquée par les aiguilles.

Q4. Cette horloge est-elle analogique ou numérique ?

On souhaite faire évoluer cette horloge en installant sur les aiguilles des photodétecteurs qui vont lire le code binaire par réflexion.

On ne s'intéresse dans un premier temps qu'à la gestion des heures.

Q5. Justifier que le code binaire naturel n'est pas un code utilisable dans ce contexte. Quelles doivent être les propriétés du code à utiliser ?
Q6. Proposer un code qui satisfasse vos critères.
Q7. Dessiner le code sur l'horloge (figure 13.14b).



(a) Horloge binaire (b) Schéma à compléter

Figure 13.14 – Horloge numérique