A.V. Codage de l'information

A.V.1 Système de numération

Le système de numération le plus répandu aujourd'hui est le système **décimal**. Toutefois, dans le monde de l'automatique, on gère des informations électriques (courant présent ou non), de pression ou autre, qui sont généralement des variables TOUT OU RIEN, auxquelles on associe les variables 0 ou 1. C'est donc le système **binaire** qui sera utilisé. Il est donc nécessaire de savoir transcrire un code décimal en binaire et inversement.

A.V.1.a Digits et bases

Un nombre N est représenté par un ensemble de caractères à l'aide de **digits** dans une base b. On écrit

$$N_b$$

Les principaux systèmes de numération sont les suivants :

Nom du système	Nb de digits	Digits utilisés	Exemple : 1000
Binaire	2	0 1	$1111101000_{(2)}$
Ternaire	3	0 1 2	$1101001_{(3)}$
Quintale	5	0 1 2 3 4	$13000_{(5)}$
Octal	8	0 1 2 3 4 5 6 7	$1750_{(8)}$
Décimal	10	0 1 2 3 4 5 6 7 8 9	$1000_{(10)}$
Hexadécimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F	$3E8_{(16)}$

Dans le système binaire, on appelle

- Bit un chiffre binaire valant 0 ou 1
- Mot une suite de Bits
- Octet un mot de 8 Bits

A.V.2 Changements de bases

A.V.2.a Expression d'un même nombre dans deux bases différentes

Soit un nombre N_B constitué de p digits dans la base B et s'écrivant $N'_{B'}$ constitué de p' digits dans la base B' :

$$N_B = n_{p-1}n_{p-2}\dots n_1 n_{0_B} = n_{p-1}B^{p-1} + n_{p-2}B^{p-2} + \dots + n_1B^1 + n_0B^0$$

$$N_{B'} = n'_{p'-1}n'_{p'-2}\dots n'_1 n'_{0_{B'}} = n'_{p'-1}B'^{p'-1} + n'_{p'-2}B'^{p'-2} + \dots + n'_1B'^1 + n'_0B'^0$$

La conversion consiste à trouver les p' digits de $N_{B'}$ dans la base B'.

A.V.2.b Applications

Nous allons voir comment passer d'une base quelconque à la base 10 et de la base 10 à une base quelconque.

Ces passages sont simplifiés par le fait que :

- D'une base quelconque à la base 10, nous savons associer :
  - chaque digit à un nombre dans la base 10
  - chaque base B au nombre B puisqu'il est exprimé dans « notre base usuelle » 10
- De la base 10 à une base quelconque, il suffit d'appliquer la division euclidienne classique que nous connaissons dans la base 10

Pour un passage entre deux bases, on pourra simplement passer par la base intermédiaire 10.

A.V.2.b.i Base B quelconque ⟶ Base 10

Binaire ⟶ Décimal : $1111101000_{(2)}$

$$1111101000_{(2)} = 1.2^9 + 1.2^8 + 1.2^7 + 1.2^6 + 1.2^5 + 0.2^4 + 1.2^3 + 0.2^2 + 0.2^1 + 0.2^0$$

$$1111101000_{(2)} = 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 = 512 + 256 + 128 + 64 + 32 + 8$$

$$1111101000_{(2)} = 1.10^3 + 0.10^2 + 0.10^1 + 0.10^0$$

$$1111101000_{(2)} = 1000_{(10)}$$

Hexadécimal ⟶ Décimal : $3E8_{(16)}$

$$3E8_{(16)} = 3.16^2 + 14.16^1 + 8.16^0 = 3 * 256 + 14 * 16 + 8 = 768 + 224 + 8$$

$$3E8_{(16)} = 1.10^3 + 0.10^2 + 0.10^1 + 0.10^0$$

$$3E8_{(16)} = 1000_{(10)}$$

A.V.2.b.ii Base 10 ⟶ Base B quelconque

• **Principe**

La transformation dans ce sens n'est pas aussi simple. Dans ce cas, voici la méthode à adopter :

Soit $N_{(10)}$, changer de base revient à déterminer les n_i tels que :

$$N_{(10)} = n_{p-1}B^{p-1} + n_{p-2}B^{p-2} + \cdots + n_2B^2 + n_1B^1 + n_0B^0$$

$$N_{(10)} = n_{p-1}B^{p-1} + n_{p-2}B^{p-2} + \cdots + n_2B^2 + n_1B^1 + n_0$$

Sachant que les n_i ne sont pas divisibles par B, par définition.

On peut encore écrire ce nombre sous la forme :

$$N_{(10)} = B\left(n_{p-1}B^{p-2} + n_{p-2}B^{p-1} + \cdots + n_2B^1 + n_1\right) + n_0$$

$$N_{(10)} = BQ_0 + r_0$$

On voit que n_0, qui n'est pas divisible par B, est le reste dans la division euclidienne de $N_{(10)}$ par B.

On peut continuer ainsi :

$$N_{(10)} = B\left(B\left(n_{p-1}B^{p-3} + n_{p-2}B^{p-2} + \cdots + n_2\right) + n_1\right) + n_0$$

$$B\left(n_{p-1}B^{p-3} + n_{p-2}B^{p-2} + \cdots + n_2\right) + n_1 = BQ_1 + r_1$$

On voit que n_1, qui n'est pas divisible par B, est le reste dans la division euclidienne de Q_0 par B.

Il faut donc effectuer des divisions euclidiennes successives par B, base d'arrivée, jusqu'à ce que le quotient soit nul. Les restes, du premier au dernier, correspondent aux digits en ordre inverse.

• **Exemples**

Décimal ⟶ Binaire : $1000_{(10)}$

	1000	2									
$n_0 =$	0	500	2								
	$n_1 =$	0	250	2							
		$n_2 =$	0	125	2						
			$n_3 =$	1	62	2					
				$n_4 =$	0	31	2				
					$n_5 =$	1	15	2			
						$n_6 =$	1	7	2		
							$n_7 =$	1	3	2	
								$n_8 =$	1	1	2
									$n_9 =$	1	0

$$1000_{(10)} = 1111101000_{(2)}$$

Décimal ⟶ Octal : $1000_{(10)}$

	1000	8			
$n_0 =$	0	125	8		
	$n_1 =$	5	15	8	
		$n_2 =$	7	1	8
			$n_3 =$	1	0

$$1000_{(10)} = 1750_{(8)}$$

Décimal ⟶ Hexadécimal : $1000_{(10)}$

	1000	16		
$n_0 =$	8	62	16	
	$n_1 =$	$14 = E$	3	16
		$n_2 =$	3	0

$$1000_{(10)} = 3E8_{(16)}$$

A.V.3 Codes binaires utilisant 0 et 1

Il existe une infinité de codes utilisant 0 et 1 possibles correspondant à des organisations différentes des digits d'un même nombre. Les trois codes principaux qu'il faut connaître sont les codes :

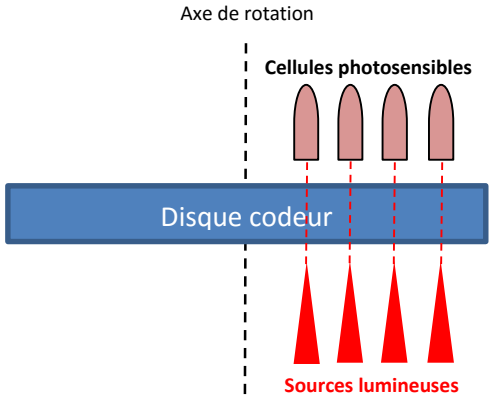
- Binaire naturel
- Binaire réfléchi ou Code Gray utilisé principalement pour coder des positions
- BCD (Binary Code Decimal) utilisé pour les calculatrices de poche

A.V.3.a Code Gray ou binaire réfléchi

En binaire naturel, un ou plusieurs digits changent en même temps entre 2 nombres successifs :

Nombre décimal	Digits code Binaire			
	b_3	b_2	b_1	b_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

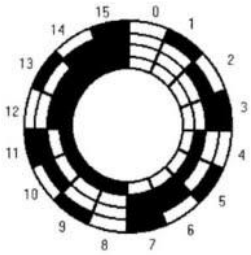
Prenons l'exemple du codage d'une position angulaire sur 16 pas par tour à l'aide d'un capteur incrémental composé d'un disque opaque composé de 4 « pistes » comportant des trous. On dispose face à chacune des pistes une source lumineuse et une cellule photosensible en face de l'autre côté du disque :



Le disque codeur est donc divisé en 4 pistes de 4 diamètres différents, puis on crée des trous lorsque l'on veut que l'information 1 transite, et on laisse la matière lorsque l'information ne doit pas passer. On peut par exemple prendre un disque en matériau transparent et coller un film opaque sur lequel on va graver (cases noires) les lieux où la lumière doit passer.

La première idée consisterait à créer des pistes selon le code binaire naturel. Ainsi, selon la position de 0 à 15, on crée des trous (cases noires) dans chaque piste lorsque le digit vaut 1 :

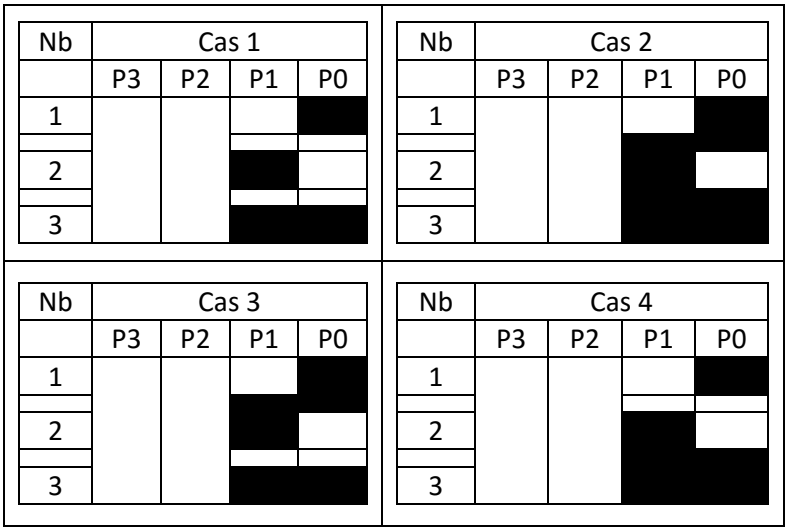
Nombre décimal	Pistes code Binaire			
	Piste 3	Piste 2	Piste 1	Piste 0
0				
1				■
2			■	
3			■	■
4		■		
5		■		■
6		■	■	
7		■	■	■
8	■			
9	■			■
10	■		■	
11	■		■	■
12	■	■		
13	■	■		■
14	■	■	■	
15	■	■	■	■



Ces pistes sont évidemment réparties circulairement sur le disque.

L'inconvénient de ce disque codeur est qu'il peut générer des erreurs du fait du changement de plusieurs bits entre deux positions successives.

En effet, imaginons deux légers décalages issus d'une fabrication imprécise des pistes autour de la position 2 et supposons que ces décalages induisent la présence des pistes suivantes (4 cas):



Pour ces 4 possibilités, l'enchaînement des positions mesurées va être le suivant :

Cas 1	Cas 2	Cas 3	Cas 4
1 – 0 – 2 – 0 – 3	1 – 3 – 2 – 3	1 – 3 – 2 – 0 – 3	1 – 0 – 2 – 3
Au lieu de 1 – 2 – 3			

Pour éviter cet inconvénient, on choisit de créer un nouveau code dans lequel un seul digit évolue à chaque changement de nombre, nommé **Code Gray**.

Nombre décimal	Digits code Gray			
	g_3	g_2	g_1	g_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

On crée les 2 premières lignes avec 0 puis 1 sur la colonne b_0
Pour les 2^1 lignes suivantes :
- on effectue une symétrie des 2^1 lignes précédentes
- on ajoute des 1 à la colonne immédiatement à gauche

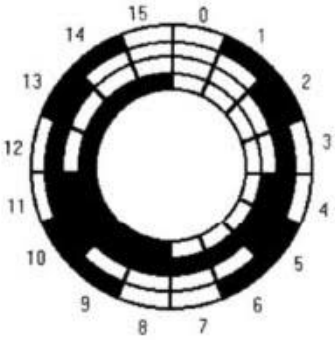
Pour les 2^2 lignes suivantes :
- on effectue une symétrie des 2^2 lignes précédentes
- on ajoute des 1 à la colonne immédiatement à gauche

Pour les 2^3 lignes suivantes :
- on effectue une symétrie des 2^3 lignes précédentes
- on ajoute des 1 à la colonne immédiatement à gauche

…

On obtient ainsi les pistes suivantes :

Nombre décimal	Pistes code Gray			
	g_3	g_2	g_1	g_0
0				
1				■
2			■	
3			■	
4		■	■	
5		■	■	■
6		■		■
7		■		
8	■	■		
9	■	■		■
10	■	■	■	
11	■	■	■	
12	■		■	
13	■		■	■
14	■			■
15	■			



Le gros avantage de ce code est que même s'il y a des défauts de fabrication ne dépassant pas la taille du codage d'un nombre, il n'y a aucuns risques d'erreurs de lecture de la position :

- Si deux positions différentes se chevauchent quelque peu, on ne pourra pas passer par une position intermédiaire fausse
- Pour éviter qu'il existe une zone blanche entre 2 plages, il suffira d'agrandir légèrement (valeur des défauts possibles) les deux plages afin de provoquer un chevauchement volontaire initial

Dans ces 2 cas, mais comme avec le codage en binaire naturel (à la différence d'éviter de fausses positions), les défauts induiront une légère erreur de connaissance sur la position exacte du disque codeur.

Il faudra toutefois avoir un transcodeur Gray – Binaire afin de traduire les signaux reçu codés en code Gray en binaire naturel. La table de transcription est donc la suivante, pour les 16 premiers nombres :

Nombre décimal	Digits code Binaire				Digits code Gray			
	b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0
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
Ainsi de suite								

A.V.3.b Code BCD

Le code BCD (Binary Code Decimal) pour décimal codé binaire, est un code qui permet de s'affranchir de la transposition Binaire – Décimal qui s'avère assez lourde dès lors que les nombres sont grands (grand nombre de Bits en binaire).

Ainsi, au lieu de coder le nombre 1000 ainsi :

$$1000_{(10)} = 1111101000_{(2)}$$

Un nombre exprimé dans la base 10 est composé de digits compris entre 0 et 9. On va coder chacun de ces digits sur 4 bits (il faut coder 10 possibilités, 3 bits n'en offrent que 8, 4 en offrent 16).

Ainsi, pour 1000, on va coder 1 puis 0 puis 0 puis 0 sur 4 bits :

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

$$1000_{(10)} = 0001000000000000_{(BCD)}$$

La lecture d'un nombre dans le code BCD s'avère bien plus simple pour nous :

$$1000100101010001_{(BCD)}$$

1	0	0	0	1	0	0	1	0	1	0	1	0	0	0	1
8				9				5				1			

$$1000100101010001_{(BCD)} = 8951_{10}$$

Un décodeur devra donc lire les Bits 4 par 4 et transcrire chacun des chiffres entre 0 et 9 puis les assembler.

La table de transcription est donc la suivante :

Digits Base 10	Digits code BCD			
d_0	d'_3	d'_2	d'_1	d'_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