1 Introduction

L'homme a besoin d'un système de codage pour identifier, quantifier, qualifier les objets, les lieux, les évènements… Ces codes lui permettent de mémoriser, traiter et communiquer les informations. Ils peuvent être :

- le langage
- l'écriture
- le graphisme
- la gestuelle
- …

Chaque code respecte des règles. Par exemple, à l'écriture correspond :

- Une liste de symboles prédéfinis : l'alphabet
- Des règles d'utilisation des symboles :
  plusieurs symboles (lettres) ➔ mot
  plusieurs mots ➔ phrase
- des règles de syntaxe pour ordonner les mots dans une phrase

⇨ Pour l'ordinateur c'est la même chose, mais son alphabet n'a que 2 symboles :

0 et 1

Pourquoi ?

- Raison technologique : il est simple de représenter en électronique un phénomène à 2 états.

Le courant passe ➔ 1
Le courant ne passe pas ➔ 0

Même si avez l'impression du contraire en utilisant certains logiciels ⇨ l'ordinateur ne comprend que les 0 et les 1. Cette logique est appelée le langage binaire.

2 Le langage binaire

- Le binaire possède un alphabet simplifié à deux symboles : **0 et 1**
- Sa base de numération est donc 2 (2 symboles).
- Normalisation d'écriture : n_2 (par exemple 101_2)

Un nombre en base 10, tel que nous l'employons d'habitude, devrait être noté sous la forme n_{10}, par exemple 135_{10}

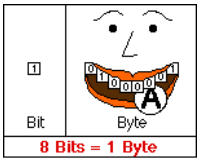
Attention 101_{10} # 101_2

Un peu de vocabulaire :

- Chaque élément binaire pouvant prendre la valeur 0 ou 1 est appelé un digit binaire (**BI**nary digi**T**)
- Une suite de **4** bits est appelée **quartet**
- Une suite de **8** bits est appelée **octet**.

Depuis 12/98, l'organisme international IEC (International Electrotechnical Commission), a définit les mesures suivantes :

1Ko = 1000 octets (2^{10} bits)
1Mo = 1000 Ko = 1000 x 1000 octets
1Go = 1000 Mo = 1000 x 1000 x 1000 octets
1To = 1000 Go = 1000 x 1000 x 1000 x 1000 octets



Mais de nombreux logiciels, parfois même certains systèmes d'exploitation, utilisent toujours la notation antérieure à 1998 pour laquelle 1Ko = 1024 octets (2^{10} bits)

Attention en anglais, octet se traduit par byte. Il faut donc absolument éviter la confusion entre bit et byte.

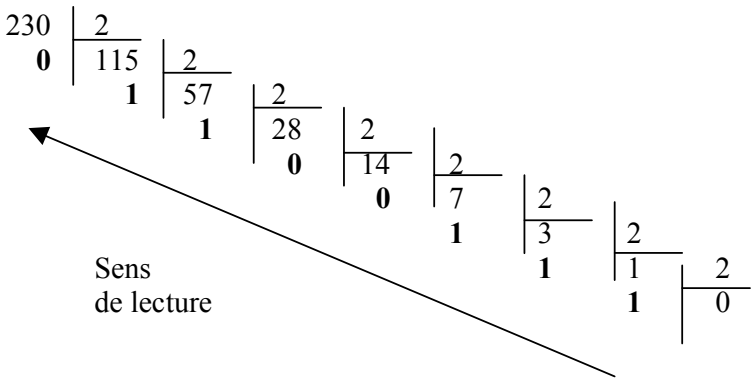
Donc 1 Ko = 1KB (Notez l'utilisation d'un B majuscule pour différencier Byte et bit)

3 Conversions binaires

3.1 Décimal → binaire par division successive :

On divise le nombre en base 10 par 2, puis on divise successivement le quotient de chaque division par 2 jusqu'à ne plus pouvoir diviser par 2. Le nombre binaire s'obtient en relevant le reste de chaque division en partant de la dernière division vers la première (sens de lecture vers le haut).

Ex : 230_{10} à convertir en base 2



Le résultat est donc : $230_{10} \rightarrow 11100110_2$

3.2 décimal → binaire par soustraction

Cette méthode consiste à retrancher du nombre la plus grande puissance de 2 possible, et ainsi de suite dans l'ordre décroissant des puissances. Si on peut retirer la puissance de 2 concernée, on note (1) sinon on note (0) et on continue de la même manière jusqu'à la plus petite puissance de 2 possible (2^0 pour les entiers).

Ex : 230_{10} à convertir en base 2

De	230	on peut retirer	128	reste 102	(1)
De	102	on peut retirer	64	reste 38	(1)
De	38	on peut retirer	32	reste 6	(1)
De	6	on ne peut pas retirer	16	reste 6	(0)
De	6	on ne peut pas retirer	8	reste 6	(0)
De	6	on peut retirer	4	reste 2	(1)
De	2	on peut retirer	2	reste 0	(1)
De	0	on ne peut pas retirer	1	reste 0	(0)

Sens de lecture ↓

Le résultat est donc : $230_{10} \rightarrow 11100110_2$

Cette technique implique de connaître les valeurs décimales associées aux puissances de 2 :

Puissance	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Résultat	1024	512	256	128	64	32	16	8	4	2	1

3.3 Conversion binaire → décimal

On multiplie chaque bit par le chiffre 2 élevé à une puissance, croissant par pas de 1, comptée à partir de zéro en partant de la droite, puis on effectue la somme des résultats obtenus.

Soit 110011_2 à convertir en décimal :

1	1	0	0	1	1	
x	x	x	x	x	x	
2^5	2^4	2^3	2^2	2^1	2^0	
(32)	(16)	(8)	(4)	(2)	(1)	
=	=	=	=	=	=	
32	16	0	0	2	1	→ **51_{10}**

3.4 Notion de Poids binaire

Le **poids binaire** ou « poids du bit » est la puissance à laquelle est élevé le bit lors de la conversion binaire → décimal.

Ainsi, les bits situés à droite du nombre sont les bits de **poids faible** (leur puissance évolue à partir de zéro). Les bits situés à gauche sont les bits de **poids fort** (leur puissance est la plus élevée et va en décroissant).

Sur un octet, le bit situé à gauche est le bit de poids fort (2^7) le plus significatif ou **MSB** (Most Significant Bit) alors que le bit le plus à droite (2^0) est le bit de poids faible ou **LSB** (Least Significant Bit).

3.5 Exercices de conversion

- $127_{10} = 111\ 1111_2$
- $10\ 0110_2 = 38_{10}$
- $255_{10} = 1111\ 1111_2$
- $256_{10} = 1\ 0000\ 0000_2$
- $0010\ 1010_2 = 42_{10}$
- $1000\ 0000_2 = 128_{10}$
- $34_{10} = 10\ 0010_2$
- $1100\ 0120_2$ = Impossible en base 2
- Quel est le plus petit nombre que peut contenir un octet ?
- Quel est le plus grand nombre que peut contenir un octet (en décimal et en binaire) ?

3.6 Les opérations binaires

De la même façon qu'en base 10, il est possible d'effectuer des opérations (addition, soustraction, multiplication et division) directement en binaire suivant les mêmes principes.

Ex : Addition

```
  1 1
  1 0 1 1
+ 0 1 1 0
  1 0 0 0 1
```

Retenue

Ex : Division

```
  1 0 1 1 0 0   | 1 0 0
- 1 0 0         |-------
    1 1         | 1 0 1 1
    1 1 0       |
  - 1 0 0       | Sens de lecture →
      1 0 0     |
    - 1 0 0     |
        0       |
```

3.7 Les nombres fractionnaires

- La partie entière d'un nombre se traduit en mettant en œuvre des puissances **positives** de 2.
- Sa partie décimale se traduit en mettant en œuvre des puissances **négatives** de 2.

Le nombre binaire obtenu se présente sous la forme d'une partie entière située à gauche du point "décimal" (ou de la virgule), et d'une partie fractionnaire située à droite.

Ex. 100.01_2 est équivalent à 4.25_{10}

3.7.1 La conversion Décimal → Binaire

Il suffit de multiplier par 2, la partie entière ainsi obtenue représentant le poids binaire (1 ou 0). La partie fractionnaire restante est à nouveau multipliée par 2 et ainsi de suite jusqu'à ce qu'il n'y ait plus de partie fractionnaire ou que la précision obtenue soit jugée suffisante.

Soit 0.625_{10}

0.625 x 2 = 1.250	Le poids binaire est 1	→ 1 x 2^{-1}
0.250 x 2 = 0.500	Le poids binaire est 0	→ 0 x 2^{-2}
0.500 x 2 = 1.000	Le poids binaire est 1	→ 1 x 2^{-3}

Sens de lecture ↓

Quand il ne reste plus de partie fractionnaire, on s'arrête. Ainsi 0.625_{10} devient 0.101_2

Tableau des puissances de 2

2^{-n}	n	2^n
1	0	1
0.5	1	2
0.25	2	4
0.125	3	8
0.0625	4	16
0.03125	5	32
0.015625	6	64
0.0078125	7	128
0.00390625	8	256
0.001953125	9	512
0.0009765625	10	1024

3.7.2 La conversion Binaire → Décimal

Le principe est similaire :

- On utilise les puissances de 2 négatives en commençant à 2^{-1}
- On parcourt les bits de la gauche vers la droite.

Ex : Convertir 0.101_2 en base 10

.	1	0	1
	x	x	x
	2^{-1}	2^{-2}	2^{-3}
	(0.5)	(0.25)	(0.125)
	=	=	=
	0.5 +	0 +	0.125 = 0.625_{10}

3.7.3 Exercices de conversion de nombres fractionnaires

10101.01_2 = 21.25_{10} 1+4+16 . 0+0.25

$24,625_{10}$ = 11000.101_2 1x16+1x8+0x4+0x2+0x1 . 1x0.5+0x0.25+1x0.125

4 Le langage hexadécimal

Le langage binaire s'il est compréhensible par la machine (et peut être appelé à ce niveau langage machine) est difficilement « assimilable » par l'homme dès lors qu'il est questions de grandes séries binaires. On utilise donc un autre système de notation, le système hexadécimal – base 16.

En notation hexadécimale, on utilise un alphabet comportant 16 symboles :

Base 16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

4.1 Conversions décimal - hexadécimal

Décimal → Hexa

On passe d'un nombre en base 10 à un nombre en base 16 par divisions successives. On note les restes des divisions successives puis on lit ces restes en « remontant ».

Ex : convertir en base 16 le nombre 728 en base 10

728	/	16	=	45	reste	8
45	/	16	=	2	reste	13 = D en hexa
2	/	16	=	0	reste	2

donc $728_{10} = 2D8_{16}$

Hexa → Décimal

Même principe que dans la conversion binaire, mais cette fois, il s'agit des puissances de 16.

Ex : $13D_{16}$ à convertir en base 10

$1 \times 16^2 + 3 \times 16^1 + D \times 16^0 \rightarrow 1 \times 256 + 3 \times 16 + 13 \times 1 \rightarrow 317_{10}$

4.2 Passage direct binaire / hexadécimal

Si l'on représente les 16 symboles de l'alphabet hexadécimal en binaire, on constate que l'on utilise pour chacun d'eux un maximum de 4 bits :

B 16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B 2	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

On peut donc passer du binaire à l'hexadécimal en décomposant le nombre binaire en bloc de 4 bits, en partant de la droite (bits dits de poids faible) et en restituant sa valeur hexadécimale à chacun de ces blocs.

Exemple : conversion de 732_{10}

$732_{10} \rightarrow 10\ 1101\ 1100_2$

$10\ 1101\ 1100_2 \rightarrow 2DC_{16}$

$2DC_{16} \rightarrow 2 \times 256 + D(13) \times 16 + C(12) \times 1 \rightarrow 732_{10}$

Le passage de l'hexadécimal en binaire peut donc se faire de la manière inverse, c'est-à-dire en convertissant les chiffres qui composent le nombre hexadécimal en leur équivalant binaire.

Ainsi $10C_{16} \rightarrow 0001\ 0000\ 1100_2$

5 TD Numération binaire et hexadécimale

1) Convertir en binaire les nombres 397_{10}, 133_{10}, 110_{10}
puis en décimal les nombres 101_2, 0101_2, 1101110_2
et vérifier en convertissant pour revenir à la base d'origine.

2) Effectuer les opérations suivantes et vérifier les résultats en procédant aux conversions nécessaires.
- a) 1100 + 1000
- b) 1001 + 1011
- c) 1100 – 1000
- d) 1000 – 101
- e) 1 + 1 + 1 + 1

3) Réaliser les opérations suivantes et vérifier les résultats en procédant aux conversions nécessaires.
- a) 1011 x 11
- b) 1100 x 101
- c) 100111 x 0110

4) Réaliser les opérations suivantes et vérifier les résultats en procédant aux conversions nécessaires.
- a) 100100 / 11
- b) 110000 / 110

5) Convertir en binaire 127.75_{10} puis 307.18_{10}

Vous pourrez constater, à la réalisation de cet exercice, que la conversion du .18 peut vous entraîner « assez loin ». C'est tout le problème de ce type de conversion et la longueur accordée à la partie fractionnaire dépendra de la précision souhaitée.

6) Convertir en hexadécimal
a) 3167_{10} b) 219_{10} c) 6560_{10}

7) Convertir en décimal
a) $3AE_{16}$ b) FFF_{16} c) $6AF_{16}$

8) Convertir en base 16
a) 128_{10} b) 101_{10} c) 256_{10} d) 110_2 e) 1001011_2

9) Convertir en base 10
a) $C20_{16}$ b) $A2E_{16}$

10) Convertir en base 2
a) $F0A_{16}$ b) $C01_{16}$

6 Correction TD Numération binaire et hexadécimale

1) Convertir en binaire les nombres 397_{10}, 133_{10}, 110_{10}
puis en décimal les nombres 101_2, 0101_2, 1101110_2
et vérifier en convertissant pour revenir à la base d'origine.

397	:	2	=	198	reste	1
198	«	«	=	99	«	0
99	«	«	=	49	«	1
49	«	«	=	24	«	1
24	«	«	=	12	«	0
12	«	«	=	6	«	0
6	«	«	=	3	«	0
3	«	«	=	1	«	1
1	«	«	=	0	«	1

Le résultat se lit en remontant : 1 1000 1101
Vérification

	Puiss2	8	7	6	5	4	3	2	1	0
		256	128	64	32	16	8	4	2	1
	x	1	1	0	0	0	1	1	0	1
		256	128				8	4		1

Soit 397_{10}

133_{10} = 1000 0101 = 128 + 4 + 1
110_{10} = 110 1110 = 64 + 32 + 8 + 4 + 2

$101_2 = 4 + 1 = 5_{10}$
0101_2 = idem, le zéro devant un nombre n'est pas significatif, en décimal ou en binaire
$1101110_2 = 64 + 32 + 8 + 4 + 2 = 110_{10}$

2) Effectuer les opérations suivantes et vérifier les résultats en procédant aux conversions nécessaires.

a) 1100 + 1000 = 10100
b) 1001 + 1011 = 10100
c) 1100 – 1000 = 0100
d) 1000 – 101 = 0011
e) 1 + 1 + 1 + 1 = 100 (en décomposant les additions)

3) Réaliser les opérations suivantes et vérifier les résultats en procédant aux conversions nécessaires.

a) 1011 x 11 = 10 0001
b) 1100 x 101 = 11 1100
c) 100111 x 0110= 1110 1010

4) Réaliser les opérations suivantes et vérifier les résultats en procédant aux conversions nécessaires.

a) 100100 / 11 = **1100**

```
1       0       0       1       0       0               11

-       1       1                                       1
-----------------------
0       0       1       1                               1
        -       1       1
        ---------------------
                0       0       0                       0
                0       0       0       0               0
```

b) 110000 / 110 = **1000**

5) Convertir en binaire 127.75_{10} puis 307.18_{10}

127.75_{10}

Partie entière : 111 1111

Partie fractionnaire : .75 x 2 = 1.50
.50 x 2 = 1.00

il ne reste plus de décimale, donc on s'arrête. ➔ **111 1111.11**

307.18_{10}

Partie entière : 1 0011 0011

Partie fractionnaire : .18 x 2 = 0.36
.36 x 2 = 0.72
.72 x 2 = 1.44
.44 x 2 = 0.88
.88 x 2 = 1.76
.76 x 2 = 1.52
.52 x 2 = 1.04 etc

➔ 1 0011 0011.0010 111

6) Convertir en hexadécimal

a) 3167_{10} = C5F

3167	:	16	=	197	reste	15	soit	F
197	:	16	=	12	reste	5	soit	5
12	:	16	=	0	reste	12	soit	C

b) 219_{10} = DB (=13 x 16 + 11)

c) 6560_{10} = 19A0 en base 16

7) Convertir en décimal

a) $3AE_{16}$ = 3 x 16^2 + A (10) x 16 + E (14) x 1
= 3 x 256 + 10 x 16 + 14 = 942

b) FFF_{16} = 15 x 256 + 15 x 16 + 15 = 3840+240+15 = 4095

c) $6AF_{16}$= 6 x 256+ 10 x 16 + 15 = 1536+160+15= 1711

8) Convertir en base 16

a) 128_{10}

128 / 2 = 64 reste = 0
64/2 = 32 = 0
32/2 = 16 = 0
16/2 = 8 = 0
8/2 = 4 = 0
4/2 =2 = 0
2/2 =1 = 0
1/2 = 0 = 1 ➔ 1000 0000 ➔ 80_{16}

b) $101_{10} = 65_{16}$
c) $256_{10} = 100_{16}$
d) $110_2 = 6_{16}$ en complétant le bloc de 4 bits = 0110
e) $1001011_2 = 4B_{16}$

9) Convertir en base 10

a) $C20_{16}$

On reconstitue le bloc de 4 bits ➔ 1100 0010 0000
Et on convertit le nombre binaire obtenu en décimal, 32 + 1024 + 2048 = 3104

b) $A2E_{16}$ ➔ 1010 0010 1110 ➔ 2+4+8+32+512+2048 = 2606 en base 10

10) Convertir en base 2

a) $F0A_{16}$ = 1111 0000 1010
b) $C01_{16}$ = 1010 0000 0001