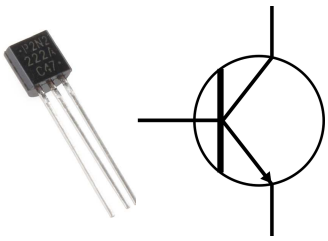
Chapitre II : Les entiers



À la base de la construction d'un ordinateur, il y a les transistors. Ces composants permettent de laisser passer ou non un courant électrique. Au passage du courant, on associe la valeur 1 et dans le cas contraire la valeur 0. Voici ce qui nous contraint à cette base 2.
Dans un ordinateur, toutes les informations (données ou programmes) sont ainsi représentées à l'aide de deux chiffres 0 et 1, appelés chiffres binaires ou *binary digits* en anglais (ou plus simplement ***bits***).

Dans la mémoire d'un ordinateur, ces bits sont regroupés en ***octets*** (c'est-à-dire des paquets de 8 bits, qu'on appelle *bytes* en anglais) puis organisés en *mots machines* de 2, 4 ou 8 octets.
Par exemple, un ordinateur dit de 32 bits manipule des mots de 4 octets ($4 \times 8 = 32$ bits) lorsqu'il effectue des opérations.

I - Les bases dans l'écriture des entiers naturels

Un nombre entier naturel écrit dans une base b est de la forme $\triangle\square Y \triangle X$ où chaque caractère est un « chiffre » dont le positionnement a une importance.
Cette écriture correspond alors au nombre égal à : $\triangle \times b^4 + \square \times b^3 + Y \times b^2 + \triangle \times b^1 + X \times b^0$.

Par exemple, en base 10 (celle que nous utilisons tous les jours) :
$23\,701 = 2 \times 10^4 + 3 \times 10^3 + 7 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$.

Remarque : Pour écrire un nombre en base b, on a besoin de b caractères distincts.
Ces caractères sont en premier lieu les chiffres que nous utilisons, puis si besoin des lettres, voire d'autres caractères.

Pour écrire un nombre en base 6, on se contenterait des chiffres allant de 0 à 5.
Par exemple 4012_6 vaut $4 \times 6^3 + 0 \times 6^2 + 1 \times 6^1 + 2 \times 6^0$.

a) Le binaire

C'est le système de comptage des ordinateurs. En binaire (base 2), on n'utilise que 2 chiffres : 0 et 1.

Voici des exemples de nombres écrits en base 2 : 1 ; 101 ; 1101101 ; 10000001 ; ...
Mais on peut aussi en énumérer dans l'ordre, en leur associant leur valeur en base 10 :

Nombres en écriture binaire	0	1	10	11	100	101	110	111	1000	...
Nombres en écriture décimale	0	1	2	3	4	5	6	7	8	...

Convertir de binaire en décimal :
C'est ce qui a été présenté au début : par exemple 1001101_2

☒ À compléter : vidéo 1 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/4a08458c-e9ad-4718-b439-2e3890d00c9c

Exercice du cours 1 : Faire de même pour les nombres binaires :
10010_2 ; 1111100_2 ; 10100101101_2

Remarque :
En python, cette conversion s'effectue directement : `int('10010',2)` ou plus simplement `0b10010`.

Convertir de décimal en binaire :
Une première méthode consiste à identifier les puissances de 2 présentes dans le nombre en commençant par les plus grandes.
Voici la liste des puissances successives de 2 (liste à connaître) : 1 ; 2 ; 4 ; 8 ; 16 ; 32 ; 64 ; 128 ; 256 ; 512 ; 1024 ; ...

Par exemple avec 185,

☒ À compléter : vidéo 2 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/dc9d700a-7005-4b1e-8323-d2ae7ee68c9a

Remarque : La plus grande puissance de 2 présente dans 185 étant $128 = 2^7$ donc 185 est écrit sur $7 + 1 = 8$ bits.

Cette méthode devient plus compliquée et pénible avec des nombres importants.
Il est dans ce cas plus pratique d'effectuer des divisions entières par 2 et de relever les restes successifs, sauf que ceci donnera les bits dans l'ordre inverse :

☒ À compléter : vidéo 3 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/6a005b90-ac45-4d12-a750-a5131766a584

Remarque : En python, cette conversion s'effectue directement : `bin(185)`.
Le résultat est sous la forme `'0b10111001'`, le 0b identifiant la base 2.

Exercice du cours 2 : Convertir 26 puis 302 en binaire.

b) L'hexadécimal

L'écriture d'un nombre en binaire nécessite de nombreux bits. La base 10 est plus condensée mais elle ne correspond pas à l'écriture des nombres sur un ordinateur. Il devient par contre pratique de choisir comme base une puissance de 2. La base $8 = 2^3$ est appelée la base **octale**. Nous allons plutôt nous intéresser à la base $16 = 2^4$ appelée base **hexadécimale**.

Cette fois-ci, il faut 16 caractères différents pour représenter chacune des 16 valeurs :

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

La conversion de la base décimale à la base hexadécimale s'effectue selon la même méthode que pour le binaire. Il est cependant plus facile d'utiliser les divisions successives par 16.
Par exemple avec 1693 :

☒ À compléter : vidéo 4 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/10ac4700-1501-4ba0-b669-86472cbd34e0

Remarque : En python, cette conversion s'effectue directement : `hex(1693)`
Le résultat est sous la forme `'0x69d'`, le 0x identifiant la base 16.

La conversion de la base hexadécimale à décimale fonctionne toujours de la même façon. Par exemple pour $1CA_{16}$ qui contient 3 « chiffres », la plus grande puissance de 16 présente sera 16^2.

$1CA_{16} =$ .....

Remarque : En python, cette conversion s'effectue directement : `int('1CA',16)` ou plus simplement `0x1ca`.

Convertir de binaire en hexadécimal et inversement
On remarque que les 16 caractères de la base hexadécimale sont codés sur 4 bits.
Pour convertir un nombre écrit en binaire en base 16, on regroupe les bits par 4 en partant de la droite et transforme chaque paquets en hexadécimal.

Par exemple pour convertir $100\,1101\,0110_2$ en hexadécimal puis $B19_{16}$ en binaire :

☒ À compléter : vidéo 5 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/7d8c416d-8285-46b2-89a3-0632635aaeb0

Remarque : On aurait également pu mentionner la base $8 = 2^3$ ou base octale. Le principe est le même que pour la base 16. Pour ce qui est du passage du binaire à la base 8, il suffit cette fois-ci de faire des paquets de 3 bits en partant de la droite...

Exercice du cours 3 :

1. Convertir les nombres suivant en hexadécimal : 123 ; 8426.
2. De même avec les nombres $110\,0110\,0111_2$ et $1\,1010\,1001_2$.
3. Convertir en binaire les nombres suivants : 102_{16} ; $7C0_{16}$.

II - Nombres de bits nécessaires

Pour un nombre écrit en base 10, nous avons déjà vu combien de bits étaient nécessaires dans son écriture en base 2.
Soit n un entier, on cherche la plus grande puissance de 2 présente dans cet entier : on cherche ainsi un entier p tel que $2^p \leqslant n < 2^{p+1}$. Alors dans ce cas $p+1$ est le nombre de bits nécessaires à l'écriture en binaire du nombre n.
Par exemple pour 123 : $64 \leqslant 123 < 128 \iff 2^6 \leqslant 123 < 2^7$ donc 123 s'écrit avec 7 bits en binaire.

Exercice : Écrire un algorithme permettant de trouver le nombre de bits nécessaires à l'écriture d'un nombre entier n (sans utiliser la fonction bin).

Solution :

```
def nbBits(n):
    """cette fonction renvoie le nombre de bits nécessaires dans l'écriture de n en base 2
    On vérifie tout de même que n est un entier naturel"""
    assert (isinstance(n,int)), "Il me faut un entier"
    assert (n>=0), "L'entier doit être positif"
```

Cas de la somme de deux entiers

Exemple : Faisons tout d'abord la somme de deux nombres écrits en base 2 :

☒ À compléter : vidéo 6 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/bbfdd2b8-df21-4023-aaae-cc7f3934d234

$$\begin{array}{r} 1\,0\,0\,1\,0\,1\,1 \\ +\quad 1\,1\,0\,1\,0 \\ \hline \end{array} \qquad \begin{array}{r} 1\,0\,1\,1\,1\,0\,0 \\ +\quad 1\,1\,0\,1\,0\,1 \\ \hline \end{array}$$

Pour trouver le nombre de bits nécessaires pour écrire la somme de deux entiers naturels, on choisit tout d'abord le maximum du nombre de bits nécessaires pour chacun de ces deux nombres.
Le nombre cherché est alors égal à ce maximum ou à son suivant (dans le cas d'une retenue comme pour le deuxième exemple).

Définition du complément à 1

Un nombre étant écrit en binaire, son **complément à 1** est obtenu en inversant tous les bits (les 1 deviennent 0 et les 0 deviennent 1).

Le complément à 1 de $1011\,0100_2$ est .....

Propriété : n_1 et n_2 étant deux entiers naturel écrits en base 2 ($n_1 \geqslant n_2$), p_1 et p_2 représentent le nombre de bits nécessaires dans l'écriture de n_1 et n_2 respectivement.
Alors le nombre de bits nécessaires pour écrire $n_1 + n_2$ vaut p_1 si n_2 est inférieur ou égal au complément à 1 de n_1 et $p_1 + 1$ sinon.

Par exemple, avec la somme de $1101\,1001_2$ et $10\,0011\,1101_2$, puis la somme de $111\,0001_2$ et $1\,1000_2$

☒ À compléter : vidéo 7 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/c9634b1f-69e2-44d0-bc68-5071c8098c8c

Cas du produit de deux entiers

Exemple : Faisons le produit de 10110_2 par 1100_2 en posant l'opération :

$$\begin{array}{r} 1\,0\,1\,1\,0 \\ \times\quad 1\,1\,0\,0 \\ \hline 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 1\,0\,1\,1\,0 \\ 1\,0\,1\,1\,0 \\ \hline 1\,0\,0\,0\,0\,1\,0\,0\,0 \end{array}$$

Le premier nombre contient 5 bits donc la plus grande puissance de 2 présente dans son écriture décimale est 2^4. Le second contient 4 bits et la plus grand puissance de 2 présente est 2^3.
Le produit des deux nombres contiendra donc la puissance $2^4 \times 2^3 = 2^{4+3} = 2^7$ et ainsi au moins 8 bits. Mais à cause de la retenue (comme dans l'exemple), on peut avoir un bit de plus.

Propriété : Deux nombres n_1 et n_2 contiennent respectivement p_1 et p_2 bits dans leur écriture binaire, alors l'écriture binaire du produit $n_1 \times n_2$ contient $p_1 + p_2$ ou $p_1 + p_2 - 1$ bits.

III - Écriture des entiers relatifs

a) Méthode naïve

☒ Vidéo explicative 8 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/198583fc-e762-4407-8481-19aa75d2316c

La représentation des entiers relatifs est plus délicate. L'idée principale est d'utiliser le bit le plus à gauche (appelé *bit de poids fort*) pour le signe : 0 indique un entier positif et 1 un entier négatif.
De la sorte, le codage des entiers positifs ne serait pas différent.
Par exemple sur 8 bits (1 octet), 58 devient 00111010_2.
On pourrait alors penser que -58 soit de la forme 10111010. Il n'en est rien car avec cette idée, on aurait deux représentations possibles de 0 : 00000000_2 et 100000000. Par ailleurs, l'addition d'entiers de signes contraires ne fonctionnerait pas correctement :
13 étant représenté par 00001101_2, -13 le serait par 10001101_2.

La somme des deux donnerait :

$$\begin{array}{rcccccccc} & & & & {\scriptstyle 1} & {\scriptstyle 1} & & {\scriptstyle 1} & \\ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ + & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ \hline & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}$$

on obtiendrait alors -26 et non 0...

b) Complément à deux

Dans cette méthode, le *bit de poids fort* est toujours utilisé comme bit de signe.
La représentation des nombres positifs ne change pas, c'est pour les entiers négatifs que nous utiliserons le complément à 2.
Ceci s'obtient simplement en deux étapes :

- on inverse les bits du nombre positif (complément à 1) ;
- on ajoute 1 au résultat (en omettant l'éventuelle retenue finale).

☒ À compléter : vidéo 9 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/4631fe38-8d2b-49da-a057-186af252f142

Exercice du cours 4 : Comment sont encodés en complément à 2 les nombres -1 et -128 sur 8 bits ?

Propriété : Avec cette représentation (encodage), sur un octet (8 bits), on peut représenter tous les entiers positifs allant de $0000\,0000_2 = 0$ à $0111\,1111_2 = 127$ (le bit de poids fort étant nécessairement égal à 0) et tous les entiers négatifs allant de -128 à -1.

☒ Vidéo explicative 10 avec le lien :
https://peertube.lyceeconnecte.fr/videos/watch/f1417517-9f60-477a-9d1d-0be282c2c856

Remarque : Avec cet encodage, on remarque que sur 8 bits (un octet), -128 est encodé par 10000000, -127 par 10000001, -126 par 10000010 et ainsi de suite en ajoutant 1 au fur et à mesure.
L'encodage de -2 sera 11111110, -1 ce sera 11111111 et en ajoutant 1 (en omettant la retenue), on retrouve 00000000 qui est bien l'encodage de 0.

Sur un octet, on peut donc encoder tous les entiers allant de $-128 = -2^7$ à $127 = 2^7 - 1$, ce qui représente 256 nombres ($256 = 2^8$).

Généralisation : Sur n bits, on peut encoder les 2^n entiers relatifs compris entre -2^{n-1} et $2^{n-1} - 1$.
Sur 16 bits, cela représente tous les entiers de l'intervalle $[-32768\,;\,32767]$ et sur 32 bits, tous les entiers de l'intervalle $[-2\,147\,483\,648\,;\,2\,147\,483\,647]$.

Exercice du cours 5 :

1. Donner la représentation en complément à 2 et sur 8 bits de -42 et 97.
2. Que valent en base 10 les octets signés 11100111 et 11000010 ?