REPRESENTATION DES NOMBRES PAR L'ORDINATEUR

Il est possible de stocker différents types d'informations (textes, images, sons…) sur un ordinateur, sous réserve qu'on ait pu les traduire sous forme numérique. Nous avons déjà abordé le codage des images. Voyons alors comment cela se fait pour les nombres.

1) Circuits et états élémentaires

Pour décrire des quantités, nous utilisons généralement le **système décimal**, appelé aussi **écriture en base 10**, utilisant 10 chiffres : $0, 1, 2, \ldots, 9$. Dans cette écriture, tous les nombres peuvent se décomposer de façon unique **selon les puissances de 10**.
Par exemple, pour un nombre entier :

$$234\,056 = 2 \times 100\,000 + 3 \times 10\,000 + 4 \times 1000 + 0 \times 100 + 5 \times 10 + 6 \times 1$$
$$= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

Ou encore pour un nombre « à virgule » :

$$20{,}504 = 2 \times 10 + 0 \times 1 + 5 \times \frac{1}{10} + 0 \times \frac{1}{100} + 4 \times \frac{1}{1000}$$

$$= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

La machine, elle, ne dispose pas d'autant de chiffres, car sa seule manière de s'exprimer est électronique, par l'intermédiaire de mini-**circuits élémentaires**, ne pouvant être chacun que dans deux états (le courant passe ou ne passe pas) que l'on a **appelés 0 et 1** par commodité et cohérence des calculs. Ces valeurs sont appelées des *booléens* ou encore des ***bits***. Ainsi, toutes les informations (et donc en particulier les nombres) sont représentées à l'aide **de suites de 0 et de 1**, appelées *mots* et transcrites par des circuits composés d'autant de circuits élémentaires (1-bit) que de bits nécessaires.
Cependant, le principe de l'écriture des nombres reste le même, sauf qu'au lieu de ranger par 10, on range alors par 2 ! Et la valeur d'un chiffre au sein d'un mot va dépendre de sa position, comme en base 10… Cette écriture est appelée **écriture binaire** ou notation binaire ou encore **écriture en base 2**.

➔ Exercices **7.1** et **7.2** P**98**

2) De l'écriture en base 2 à l'écriture en base 10

Pour pouvoir communiquer, l'utilisateur et elles doivent être capables de passer d'un « langage » à l'autre.

A_2 signifie que le nombre A est écrit en base 2 (même principe pour la base 10), ce qui évite de confondre par exemple le nombre 110 en base 10 et en base 2.

Commençons par comprendre le passage **de l'écriture en base 2 à l'écriture en base 10.** Un petit exemple basique : $1101_2 = (1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3)_{10} = \ldots\ldots\ldots\ldots$

On s'aperçoit donc qu'il suffit de **décomposer le nombre binaire selon les puissances de 2** pour obtenir sa version décimale, mais en faisant attention à la position des chiffres : le bit le plus à droite est celui qui a le moins de « poids » et qui correspond donc à la puissance 0, son voisin à la puissance 1, et ainsi de suite jusqu'au bit le plus à gauche. Il est conseillé de commencer par la droite, afin d'éviter les erreurs de rang (de puissance), notamment pour les mots « longs ».

a) A votre tour d'écrire les nombres binaires suivants en base 10 :

Base 2	10	11	110	1000	10000	110011	11001101
Base 10							

b) → *Exercice* **7.17 P 103**

c) Lorsque l'ordinateur donne un nombre en base 10, il traduit la version binaire du nombre, qu'il a en mémoire, en suivant un **algorithme, dit « de décodage ».** Essayons d'en avoir un aperçu… (Il sera intéressant de le programmer lorsque l'occasion se présentera)

Pour s'y retrouver dans sa mémoire (succession quasi-ininterrompue de 0 et de 1 !), l'ordinateur regroupent les données par groupe de 8 bits, soit par …… Et afin que tous les nombres soient mémorisés de façon uniforme, tout en gardant la possibilité de coder de « grands » nombres, il les code généralement **tous sur 32 bits** (soit ……) **ou bien sur ……**

Quel est le plus grand entier que l'on puisse coder sur 32 bits ? ……

sur 64 bits ?……

sur n bits ?……

Supposons qu'on soit dans le cas d'un codage sur 32 bits. On peut donc considérer que chaque nombre entier est mémorisé via son écriture binaire à 32 chiffres, rangés dans une sorte de **tableau indexé *T*** (représentant la mémoire) .

Supposons également que la fonction $T[i]$ permette à la machine d'obtenir le chiffre de la colonne i du tableau .

Exemple :

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	1

Quel est le nombre binaire stocké ? ……

Que vaut $T[4]$?…… $T[18]$?……

Dans la traduction en écriture décimale, à quelle puissance de 2 sera associé chacun d'eux ? ……

A quelle puissance de 2 sera associé $T[i]$ de façon générale ?……

Compléter l'algorithme suivant, afin qu'il renvoie l'écriture décimale d'un nombre binaire codé sur 32 bits et fourni sous la forme d'un tableau T :

```
Variables
entier i,n ;  tableau T de taille 32
Entrée
Saisir ……
Initialisation
…… prend la valeur ……
Traitement et sortie
 Pour i de ……. à ……  faire
    n prend la valeur ……
 Fin Pour
Afficher  ……
```

3) De l'écriture en base 10 à l'écriture en base 2

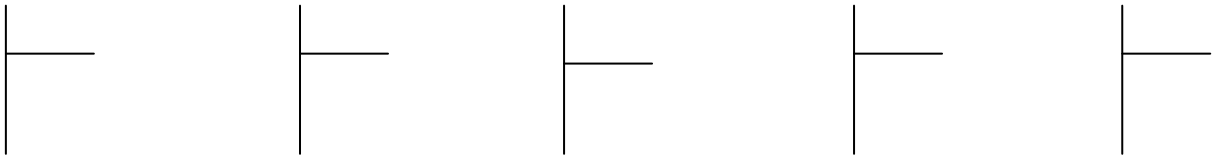
Le **problème inverse** se pose aussi. Comment traduire la base 10 en base 2 afin que l'ordinateur puisse le manipuler ?

Ecrire les nombres entiers suivants comme somme de puissances de 2, et en déduire leur écriture binaire :

- 15_{10} =
- 29_{10} =
- 158_{10} =

Assez laborieux dès que les nombres sont « un peu grands » !

Considérons de nouveau le nombre 29. Poser la division de 29 par 2. Récupérer le quotient, et poser sa division par 2 puis recommencer, jusqu'à ce que le quotient soit 0. (barres de division prévues à cet effet)



Que constatez-vous ?

Comment expliquer cela ?

Récapituler la méthode la plus efficace à appliquer à un nombre en base 10 pour obtenir son écriture binaire :

➔ Exercices **7. 11, 7. 12, 7. 13, 7. 14, 7. 19 p 102**

Pour cette « traduction », l'ordinateur suit également un algorithme. Essayons là aussi d'en avoir un aperçu…
Cette fois, le nombre en base 10 est fourni, et on veut remplir le tableau de 32 bits, afin de stocker ces bits pour les afficher (ou les renvoyer) ultérieurement.
On suppose que l'ordinateur dispose d'une fonction *reste*(n,d) qui fournit le reste de la division entière de n par d et d'une fonction *quotient*(n,d) qui en fournit le quotient.
Compléter l'algorithme suivant afin qu'il affiche l'écriture binaire du nombre n en base 10 fourni :

```
Variables
entiers i,j,n ;  tableau T de taille 32
Entrée
Saisir ……
Initialisation
  Pour i de 1 à 32 faire
    ……
  Fin Pour
  j prend la valeur……
Traitement et sortie
 …… (quotient(……)≠ 0) faire
    ……. prend la valeur reste(……
    n  prend la valeur quotient(……
    j prend la valeur……
 Fin ……
 Pour i de ... à ... faire
    Afficher  T[……]
 Fin Pour
```

4) Représentation des nombres relatifs

Un ordinateur est appelé à manipuler d'autres types de nombres que les entiers naturels, parmi lesquels tout d'abord les **entiers relatifs** ! Comment l'ordinateur peut-il comprendre et coder la notion de signe ?!

Il faut ajouter une information supplémentaire. La première idée est généralement de **consacrer un bit au signe**, puisqu'heureusement, il n'y a que 2 signes possibles, et donc on peut représenter l'un par 0 et l'autre par 1. Et pour que ce soit plus naturel, on peut lui consacrer le bit le plus à gauche sur les 32, si on code en 32 bits, par exemple. Ainsi, on décide que les nombres positifs commencent par 0 et que la partie numérique du nombre est lue sur les 31 bits restants. Même principe pour les nombres négatifs.

A l'aide de ce codage, donner l'écriture binaire des nombres suivants :

- -20_{10} =
- 6_{10} =
- 0_{10} =

Combien de nombres peuvent ainsi être codés ?.....

Et avec n bits ?.....

Que constate-t-on dans le cas de 0_{10} ?.....
.....

On a donc préféré une autre méthode :

On va *représenter un entier relatif par un entier naturel.*

Par exemple sur 8 bits : $10_{10} = 00001010_2$. L'entier est positif, le bit de gauche est 0.

Lorsque l'entier x est négatif, on va le représenter comme $x + 2^8$.

Par exemple *si* $x = -10_{10}$, $-10 + 256 = 246$ qui s'écrit 11110110_2 donc $-10_{10} = 11110110_2$

On dit que l'**on prend le complément à 2^n** . Voici une astuce : *pour représenter un entier négatif en binaire, on inverse tous les bits, on ajoute 1.*

- Coder en binaire : -1094_{10} =.....
  Coder -1_{10}=.....
- A quel nombre décimal correspond le code suivant ?
  10000111_2 = ...
- Avec 8 bits quel est le plus grand entier positif que l'on puisse coder ?
  .....
- Le plus petit entier négatif?
  .....
- Avec 16 bits ?
  .....
- Avec n bits ?
  .....

➔ Exercices **7.26, 7.28, 7.30, 7.32 p 105 – 107**

Pour s'amuser un peu :
Représenter les entiers 27 et 79 en binaire sur 8 bits. Ajouter les deux nombres binaires obtenus (on pourra poser l'addition comme pour les nombres décimaux, mais attention, 2 n'existe pas en binaire). Quel est le nombre entier obtenu ?
Recommencer la même opération avec 96 et 48. Pourquoi le résultat est-il négatif ?