

REPRÉSENTATION DES NOMBRES

1. Symboliser les états des circuits de la mémoire

La mémoire d'un ordinateur est structurée en *bits*, circuits électroniques, chacun de ces circuits ne pouvant être que dans deux états

sous tension, état symbolisé par le chiffre 1

hors tension, état symbolisé par le chiffre 0

Un circuit composé de plusieurs bits a un état symbolisé par un mot composé de 0 et de 1.

Un regroupement de 8 bits s'appelle *un octet*.

Exemple. Le mot 1001 symbolise un circuit composé de 4 circuits mémoire un bit respectivement dans les états 1, 0, 0 et 1.

2. Représentation des entiers naturels et relatifs

2.1 Écriture des entiers naturels en bases 10 et 2

Usuellement, nous travaillons avec les nombres écrits en base 10. Par exemple,

$$1349 = 1 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 9 \times 10^0$$

et plus généralement, pour tout $n \in \mathbb{N}$, l'égalité

$$n = \sum_{k=0}^p c_k 10^k = c_p \times 10^p + \dots + c_0 \times 10^0$$

se note $n = c_p \dots c_1 c_0$ où chaque c_k est compris entre 0 et 9.

Écrire n en base 2 consiste à le décomposer comme précédemment mais avec des puissances de 2 à la place des puissances de 10. On a le tableau suivant

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1

L'exemple précédent s'écrit

$$1349 = 1024 + 256 + 64 + 4 + 1$$

$$= 1 \times 2^{10} + 0 \times 2^9 + 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

On utilise la notation en soulignant avec 2 en indice

$$1349 = \underline{10101000101}_2$$

et plus généralement, pour tout $n \in \mathbb{N}$, l'égalité

$$n = \sum_{k=0}^{p'} c'_k 2^k = c'_p \times 2^p + \dots + c'_0 \times 2^0$$

se note $n = \underline{c'_p \dots c'_1 c'_0}_2$ (indice 2) où chaque c'_k est compris entre 0 et 1.

Méthode Pour passer de l'écriture en base 10 à celle en base 2, on effectue successivement des divisions euclidiennes par 2.

2.2 La base 2 dans les ordinateurs

Avec leurs circuits mémoire un bit, les ordinateurs comptent naturellement en base 2. Chaque circuit mémoire un bit représente un des deux chiffres de cette base : 0 ou 1.

2.3 Représentation des entiers relatifs : en complément à 2

On choisit de représenter les entiers relatifs en *complément à deux*.

Pour représenter sur N bits un entier relatif x on procède ainsi

- si x est positif ou nul alors c'est un entier naturel et il se représente comme vu en début de cours
- si x est strictement négatif alors on le représente comme $2^N + x$

Par exemple, sur $N = 3$ bits, ce choix donne les $2^N = 8$ représentations suivantes

Nombre codé	0	1	2	3	$2^3 + (-4) = 4$	$2^3 + (-3) = 5$	$2^3 + (-2) = 6$	$2^3 + (-1) = 7$
En binaire	000	001	010	011	100	101	110	111
En mémoire	0	1	2	3	-4	-3	-2	-1

On remarque que les entiers strictement négatifs ont un premier bit égal à 1 et que les entiers positifs ou nul ont un premier bit égal à 0 (cf feuille d'exercices).

3. Représentation des nombres à virgule

La notation binaire permet aussi de représenter les nombres à virgule.

Le principe s'inspire de celui de l'écriture scientifique des nombres en base 10.

3.1 Écriture scientifique en base 10

Tout nombre décimal s'écrit en base 10 sous la forme

$$(-1)^s \times m \times 10^{\text{exposant}}$$

où

- $(-1)^s$ donne le signe qui est $-$ ou $+$ suivant $s = 1$ ou $s = 0$,
- m est un nombre décimal avec un seul chiffre, compris entre 1 et 9, à gauche de la virgule,
- l'exposant est un entier relatif.

Exemple. On a $43,625 = (-1)^0 \times 4,3625 \times 10^1$. Cette écriture s'obtient ainsi

$$\begin{aligned}
 43,625 &= 4 \times 10^1 + 3 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3} \\
 &= \left(4 \times 10^0 + 3 \times 10^{-1} + 6 \times 10^{-2} + 2 \times 10^{-3} + 5 \times 10^{-4}\right) \times 10^1 \\
 &\quad \text{(On a mis le plus grande puissance de 10 en facteur)} \\
 &= (-1)^0 \times 4,3625 \times 10^1
 \end{aligned}$$

Ici $s = 0$, $m = 4,3625$ et $n = 1$.

3.2 L'analogie en base 2

Dans l'exemple précédent, les mêmes calculs, avec maintenant des puissances de 2 donnent

$$\begin{aligned} 43,625 &= 32 + 8 + 2 + 1 + 0.5 + 0.125 \\ &= 2^5 + 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3} \\ &= (2^0 + 2^{-2} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-8}) \times 2^5 \\ &= (-1)^0 \times \underline{1,01011101}_2 \times 2^5 \end{aligned}$$

On retrouve une expression de la forme

$$(-1)^s \times m \times 2^{\text{exposant}}$$

où

- $(-1)^s$ donne le signe est $-$ ou $+$ suivant $s = 1$ ou $s = 0$,
- m est un nombre binaire à virgule, compris entre 1 inclus et 2 exclu, appelé *mantisse*,
- l'exposant est un entier relatif.

Remarque. Les nombres après la virgule dans la mantisse représentent les demis 2^{-1} , les quarts 2^{-2} , les huitièmes 2^{-3} , ...

Cette représentation s'appelle *écriture en virgule flottante normalisée*. La virgule est dite flottante car elle dépend a priori de l'exposant n . La normalisation provient du choix d'une mantisse m telle que $1 \leq m < 2$.

La représentation en machine va donc consister à représenter le signe via s , la mantisse m et l'entier relatif qui sert d'exposant.

3.3 Représentation en mémoire

On donne les caractéristiques **pour une représentation sur $N = 32$ bits**.

- On réserve 1 bit pour le signe s .
- On réserve 23 bits pour la mantisse m .

Comme cette mantisse a toujours le seul chiffre 1 avant la virgule, il est inutile de le représenter ce qui économise un bit.

On utilise donc les 23 bits pour représenter les chiffres après la virgule, c'est à dire les demis 2^{-1} , les quarts 2^{-2} , les huitièmes 2^{-3} , ...

- On prend 8 bits pour l'exposant n .

C'est un entier relatif, mais on décide **de ne pas le coder en complément à deux**.

On le représente comme l'entier naturel $n + 2^7 - 1 = n + 128 - 1 = n + 127$. Ce codage particulier s'appelle *127-excès*.

L'exposant n est dans $\{-127, \dots, -1, 0, 1, \dots, 128\}$.

Sa représentation est celles des entiers dans $\{0, 1, \dots, 255\}$.

En résumé

Pour $N = 32$ bits, la représentation en machine

$$\underbrace{s}_{\text{signe}} \underbrace{e_1 \dots e_8}_{\text{exposant}} \underbrace{m_1 \dots m_{23}}_{\text{mantisse}}$$

correspond au nombre réel

$$(-1)^s \times 2^{n-127} \times \left(1 + \sum_{i=1}^{23} m_i \frac{1}{2^i} \right)$$

où $e_1 \dots e_8$ est la représentation en base 2 de n et $m_1 \dots m_{23}$ est la représentation en base 2 des chiffres après la virgule de m .

Remarques. Les valeurs extrêmes $0 = \underline{00000000}_2$ et $255 = \underline{11111111}_2$ sont réservées pour les cas particuliers ci-dessous :

1. si la représentation de l'exposant est composée de 8 bits 0, alors on considère que le chiffre avant la virgule est un 0 implicite (au lieu de 1). Ces nombres ne sont plus normalisés, mais ils permettent d'augmenter le nombre de valeurs proche de zéro représentables.
2. L'autre valeur extrême, composée de 8 bits de valeurs 1 est utilisée pour $-\infty$, $+\infty$ et les *NaN* (Not a Number) qui signalent une impossibilité de calculer (comme par exemple une division par 0).

4. Des problèmes liés à ces représentations

4.1 Dépassements de capacité

Les nombres entiers ou réels représentables sont en nombre fini. Sur N bits, nous avons vu qu'on représente seulement les entiers relatifs compris entre -2^{N-1} et $2^{N-1} - 1$. Un dépassement par valeurs supérieures s'appelle un *overflow*.

Par exemple, sur $N = 3$ bits en représentation complément à deux le calcul posé $3 + 2$ donne une résultat négatif en additionnant bit à bit ! Il y a overflow car $5 > 3 = 2^{3-1} - 1$.

4.2 Arrondis

Dans les exemples précédents, les réels utilisés avaient une représentation exacte en écriture à virgule flottante normalisée.

C'est loin d'être le cas pour la plupart des nombres ! Par exemple 10^{-1} ne pose pas de problème en base 10, mais il a un développement infini et périodique en base 2.

Dans ce cas la représentation en écriture à virgule flottante donnera un arrondi du nombre initial ... et les arrondis sont source d'erreurs.

4.3 Comparaison à zéro

- Tout test d'égalité utilisant deux nombres a et b représentés en virgule flottante n'a pas de sens en général ! En effet, ces nombres ont pu subir des erreurs d'arrondis.
- On préférera un test de la forme $|a - b| < \varepsilon$ où ε sera proche de 0 de l'ordre des erreurs d'arrondi auxquelles on peut s'attendre.
- On parle du problème de la **comparaison à zéro** car tester l'égalité de deux nombres revient à comparer leur différence et 0.

À retenir : on ne peut pas comparer deux réels ou bien un réel et zéro.

4.4 Une erreur célèbre

Le 4 juin 1996 une fusée Ariane 5, lors de son premier lancement, a explosé 40 secondes après l'allumage. Coût : 500 millions de dollars ! Il s'agissait d'une erreur de programmation dans le système inertiel de référence. À un moment donné, un nombre codé en virgule flottante sur 64 bits (qui représentait la vitesse horizontale de la fusée par rapport à la plate-forme de tir) était converti en un entier sur 16 bits. Malheureusement ce nombre était plus grand que $2^{16-1} - 1$ et la conversion a été incorrecte.

4.5 Le saviez-vous ?

Il y a 10 sortes de personnes. Celles qui savent compter en base 2 et les autres.