Numérique et sciences informatiques
Première

B. Représentation de l'information
5. Représentation des nombres entiers en mémoire

Dans cette séance, nous allons voir sous quelles formes les nombres traités par un ordinateur sont représentés dans sa mémoire. Nous allons d'abord parler d'un des deux types de nombres que nous avons rencontrés en Python : les nombres entiers relatifs associés au type **`int`**.

I Entiers positifs et place mémoire

I.1 Combien de bits pour un entier positif donné ?

Pour écrire un entier positif n en base 2, il faut disposer au minimum de k bits où k est un entier tel que

$$2^{k-1} \leq n < 2^k$$

• Pour déterminer le nombre de bits minimum k à partir de l'entier positif n, on peut utiliser la fonction logarithme de base 2, notée $\log_2$. Par définition, si x est un réel strictement positif, on a :

$$y = \log_2(x) \Leftrightarrow y = 2^x$$

• La fonction logarithme de base 2 possède les propriétés suivantes :

* $\log_2(2) = 1$
* si x et y sont deux réels strictement positifs, $\log_2(xy) = \log_2(x) + \log_2(y)$

• On a alors :

$$k = E\big(\log_2(n)\big) + 1 = \lceil \log_2(n) \rceil$$

$E(x)$ désigne la partie entière de x. k représente donc *l'arrondi à l'entier supérieur* de $\log_2(n)$ que l'on notera $\lceil \log_2(n) \rceil$

I.2 Combien de bits pour la somme de deux entiers ?

Supposons que nous avons de deux entiers positifs n_1 et n_2 et que nous souhaitons savoir combien il nous faut de bits au minimum pour exprimer la somme $n_1 + n_2$.

• Si n_2 est le plus grand des deux entiers , on aura :

$$n_2 \leq n_1 + n_2 \leq 2n_2$$

Donc le nombre K de bits minimum nécessaires pour exprimer $n_1 + n_2$ vérifiera les inégalités suivantes :

$$\lceil \log_2(n_2) \rceil \leq K \leq \lceil \log_2(2n_2) \rceil = \lceil \log_2(n_2) + \log_2(2) \rceil = \lceil \log_2(n_2) \rceil + 1$$

Il faut donc un bit de plus que le nombre de bits nécessaire pour représenter n_2.

I.3 Combien de bits pour le produit de deux entiers ?

Supposons maintenant que nous avons de deux entiers positifs n_1 et n_2 et que nous souhaitons savoir combien il nous faut de bits au minimum pour exprimer le produit $n_1 \times n_2$.

• Si n_2 est le plus grand des deux entiers , alors ce nombre de bits K' vérifie :

$$K' = \lceil \log_2(n_1 \times n_2) \rceil = \lceil \log_2(n_1) + \log_2(n_2) \rceil \leq \lceil \log_2(n_1) \rceil + \lceil \log_2(n_2) \rceil$$

Donc, dans le pire des cas, il faut la somme du nombre de bits significatifs de n_1 et du nombre de bits significatifs de n_2.

II Représentation des entiers relatifs par la méthode du complément à deux

II.1 Le problème à résoudre

On dispose d'un certain espace mémoire, exprimé en octets, pour représenter un nombre entier relatif et on souhaite pouvoir représenter *tous les entiers consécutifs* entre une valeur minimale et une valeur maximale de façon à pouvoir représenter à peu près autant de nombres négatifs que de nombres positifs. Bien entendu il faut aussi représenter le nombre 0.

Par exemple, supposons que l'on dispose de deux octets, c'est-à-dire de 16 bits, pour représenter les entiers. On dispose alors de $2^{16} = 65\,536$ combinaisons possibles de 0 et de 1. On pourrait alors représenter :

* tous les entiers positifs compris entre 0 et $32\,767$;
* tous les entiers négatifs compris entre $-32\,768$ et -1.

II.2 Une première méthode et ses inconvénients

• L'idée la plus immédiate consiste à réserver :
  * un bit pour le signe : par exemple, le bit plus à gauche appelé « bit de poids fort ». Les nombres positifs commencent par le bit 0 ; les nombres négatifs par le bit 1 ;
  * tous les bis qui suivent expriment la valeur absolue de l'entier à représenter.

• Cette méthodes possède deux inconvénients :
  * le nombre zéro est alors représenté de deux façons différentes. Sur un octet, il s'écrit « 00000000 » ou « 10000000 ».
  * on ne peut pas utiliser la technique de la retenue pour effectuer l'addition de deux nombres entiers. En effet, notamment pour de signes différents, cette technique ne donne pas le bon résultat.

• Par exemple, en, supposant que l'on code sur un octet, si on veut calculer $-6 + 3$.
On coderait en binaire de la façon suivante :

-6 par $1000\ 0110$ et 3 par $0000\ 0011$

L'addition avec retenue donnerait :

$$\begin{array}{r} 1000\ 0110 \\ +\ 0000\ 0011 \\ \hline 1000\ 1001 \end{array}$$

Le nombre $1000\ 1001$ correspond à -9 alors que l'on doit obtenir -3.

II.3 La méthode utilisée en pratique : la méthode du complément à deux

Pour des raisons de commodité des calculs, on a donc adopté la "méthode du complément à deux".

a) La méthode de représentation

On procède de la façon suivante :

• On représente *tout entier positif* comme on le représenterait dans la méthode précédente : le premier bit prend la valeur 0 et les bits suivants expriment la valeur absolue de cet entier

(c'est-à-dire lui-même puisqu'il est positif). Par exemple, le nombre 7 est représenté sur un octet de la façon suivante « $0000\ 0111$ ».

• On représente *tout entier négatif* en utilisant l'algorithme suivant :
(a) représenter sa valeur absolue avec la technique précédente ;
(b) construire la représentation par « négation bit à bit » ;
(c) ajouter 1 à cette représentation en propageant la retenue.

• Par exemple, on détermine la représentation binaire du nombre -12 de la façon suivante :
(a) sa valeur absolue est 12 donc elle se représente par « $0000\ 1100$;
(b) la représentation obtenue par négation bit à bit est « $1111\ 0011$ » ;
(c) si on ajoute 1 en propageant la retenue, on obtient « $1111\ 0100$ »
Ainsi l'entier -12 est donc représenté par « $1111\ 0100$ ».

• Partant de la représentation par « complément à deux » d'un nombre négatif on peut aussi retrouver la représentation de son nombre opposé (donc positif) en appliquant le même algorithme :
* écrire la représentation par négation bit à bit
* lui ajouter 1 en propageant la retenue.

• Par exemple, en partant de la représentation de -12, soit « $1111\ 0100$ ».., on obtient :
* « $0000\ 1011$ » par négation bit à bit ;
* « $0000\ 1100$ » après avoir ajouté 1 et propagé la retenue.
Or nous avons vu que « $0000\ 1100$ » est bien la représentation du nombre 12.

• On peut ainsi remarquer que la représentation de « -0 » par la méthode du complément à 2 du nombre 0 donne la même représentation binaire que « $+0$ ». En effet le complément à 2 de « 00000000 » donne « 1111111 » puis « 00000000 » après ajout de 1 (en ignorant la retenue finale). Ainsi il n'y a qu'une seule représentation du nombre 0.

• Selon cette technique le bit de poids fort sera toujours :
* à 0, si le nombre est positif ;
* à 1 si le nombre est négatif.

• Si on utilise cette technique par exemple avec deux octets, on peut ainsi représenter :
* tous les entiers positifs compris entre 0 à $32\ 767$;
* tous les entiers négatifs compris entre -32768 et -1.

• D'une façon générale, si on dispose de n bits pour représenter les entiers relatifs avec la "méthode du complément à deux", on représente :
* les entiers positifs de 0 à $2^{n-1} - 1$;
* les entiers strictement négatifs de -2^{n-1} à -1.

II.4 Addition et soustraction avec la représentation par complément à deux

• Vérifions que l'on peut additionner deux nombres représentés ainsi en utilisant la « technique de la retenue ». Par exemple $-12 + 7$ se calcule ainsi :

$$\begin{array}{r} 1111\ 0100 \\ +\ 0000\ 0111 \\ \hline 1111\ 1011 \end{array}$$

• Le nombre représenté par « $1111\ 1011$ » est négatif puisque son bit de poids fort est 1.

• Pour vérifier quel est le nombre représenté par « $1111\ 1011$ », on applique l'algorithme direct :
(a) on applique la négation bit à bit et on obtient « $0000\ 0100$ » ;

(a) on ajoute 1 à cette représentation et on obtient « $0000\ 0101$ » ;

(c) Puisque « $0000\ 0101$ » représente l'entier positif 5, « $1111\ 1011$ » représente son opposé, donc l'entier -5.

On a donc obtenu le résultat attendu.

• Pour soustraire l'entier a par l'entier b, il suffit d'additionner a et l'opposé de b :

$$a - b = a + (-b)$$