Informatique PCSI
Les nombres entiers : représentation

1 Numérisation

Toutes les données utilisées dans une machine ont été numérisées, donc transformées en nombres entiers. Le nombre 91 peut représenter l'entier 91. Mais il peut aussi représenter, (en suivant le code ASCII), la lettre "a" ou une quantité de rouge dans une image en couleur. Numériser une information, c'est la convertir en une suite de 0 et de 1. Cette suite représente de manière exacte ou approchée une donnée.

Ces nombres doivent être conservés en mémoire et manipulés. Il est important de comprendre comment ces entiers sont représentés en machine et de distinguer cette représentation de celle utilisée par un langage de programmation particulier.

Pour écrire des entiers, nous sommes habitués à utiliser une notation de position avec le système décimal. Le zéro exprime le vide ou l'absence et permet de distinguer par exemple 35 de 305.

1.1 Représentation des nombres

Nous utilisons en permanence la notation positionnelle en base dix. Lorsque nous écrivons 357, que nous lisons « trois-cent-cinquante-sept », c'est une simplification de l'écriture $3 \times 100 + 5 \times 10 + 7$.

Cette notation est probablement née en Inde où elle était utilisée entre les années 600 et 700 dans des opérations arithmétiques. Elle est arrivée en Perse, lorsque le mathématicien al-Khwarizmi a traduit les chiffres en langue Arabe. Les traductions des ouvrages d'al-Khwarizmi en latin permettrons de répandre cette notation en Europe à partir des années 1200.

Un entier naturel peut donc s'écrire $c_n c_{n-1} \dots c_1 c_0 = c_n 10^n + c_{n-1} 10^{n-1} + \dots + c_1 10 + c_0$.

Avec la base dix, nous utilisons dix chiffres de 0 à 9. Il apparaît clairement que le rôle du zéro dans l'écriture positionnelle est capital. Sinon, comment distinguer 270 de 27, ou 7803 de 783 ?

En suivant le même principe, nous pouvons envisager une notation positionnelle en base b, où b est un entier quelconque avec $b \geq 2$, en utilisant b chiffres, de 0 à $b-1$.

$(a_n a_{n-1} \dots a_1 a_0)_b = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b + a_0$

Par exemple : $(2301)_4 = 2 \times 4^3 + 3 \times 4^2 + 0 \times 4 + 1 = 180$.

Une difficulté apparaît si $b > 10$, puisque nous ne disposons que de 10 chiffres dans notre écriture. En base seize par exemple, nous avons besoin de seize chiffres. Il a été choisi de compléter avec des lettres. Ainsi, les chiffres de zéro à quinze sont habituellement notés 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Si $b = 2$, on obtient le système binaire. Les deux chiffres 0 et 1 sont appelés bits, mot qui vient de l'anglais « binary digit ».

C'est le mathématicien Leibnitz qui est considéré à l'origine du développement du système binaire pour lequel il a précisé comment calculer en particulier pour les quatre opérations de base.

Certains des premiers ordinateurs calculaient en utilisant l'arithmétique décimale. C'est à partir des travaux de von Neumann dans les années 1940 que le calcul en binaire est devenu la norme.

1.2 Le binaire

L'utilisation du binaire a plusieurs raisons. Une première raison est qu'avec une machine utilisant des circuits électroniques, il est plus simple et plus sûr de ne considérer que deux états dans l'observation de ces circuits et donc de construire une machine qui n'utilise que deux symboles pour son fonctionnement. À ces deux états correspondent les deux chiffres 0 et 1. Une deuxième raison est que ces deux états ou ces deux chiffres peuvent correspondre aux deux valeurs d'une propriété, faux ou vrai, ce qui permet de faire le lien entre l'arithmétique binaire et le calcul logique. Ainsi les variables qui ne peuvent prendre que deux valeurs 0 ou 1 sont appelées des *variables booléennes* du nom du mathématicien britannique George Boole qui a développé au XIX^{e} une algèbre liée à la théorie des ensembles. Les trois opérations logiques addition, multiplication et négation correspondant à la réunion, l'intersection et le complémentaire sur l'ensemble des parties d'un ensemble de référence.

Les valeurs 0 et 1 correspondent aussi aux valeurs de la fonction caractéristique d'une partie A d'un ensemble E : $K_A(x) = 1$ si $x \in A$ et 0 sinon. Ainsi le produit de deux bits correspond à la conjonction de deux valeurs booléennes et à l'intersection de deux parties.

Voici quelques remarques complémentaires qui militent pour le binaire.

En base deux, une multiplication est guère plus compliquée qu'une addition.

Le passage d'une base $b = 2$ à une base $b = 2^p$ est toujours simple. Cela consiste à regrouper les bits par p bits. Par exemple, $(a_n a_{n-1} \dots a_1 a_0)_2 = (A_q \dots A_1 A_0)_{2^2}$ avec $A_k = a_{2k+1} a_{2k}$.

Dans une addition en base b, les retenues éventuelles valent toujours un. Mais dans une multiplication en base b, il y a $b - 1$ retenues possibles. En base deux, la seule retenue possible vaut 1 comme pour l'addition.

Donc, que ce soit pour le fonctionnement de la machine ou pour la simplicité des calculs, le binaire semble être un bon choix. Il permet de réaliser un lien entre la physique, le calcul numérique, la logique booléenne et la théorie des ensembles.

1.3 Vocabulaire

Une suite composée de 0 et de 1 s'appelle un *mot*. Les chiffres 0 et 1 sont les *lettres* et l'ensemble $\{0, 1\}$ est l'*alphabet*. La taille d'un mot s'exprime en bits. Un mot de huit bits est un *octet* (ou byte). Un mot est l'*unité de base* pour la machine ou le processeur. On compte en bits ou en octets. Avec seize bits, soit deux octets, on parle de word, avec trente-deux bits, soit quatre octets, de dword (pour double word) et avec soixante-quatre bits, soit huit octets, de qword (pour quadruple word). Suivant les architectures des ordinateurs, la taille du *mot* ou du *word* change. Dans les machines récentes un word a une taille de 64 bits. Dans les architectures plus anciennes un word a une taille de 32 bits.

Le type char, qui permet de représenter un caractère dans certains langages, utilise huit bits depuis le début des années 1970 et les premiers processeurs Intel. Ce type permet de représenter 256 caractères ($256 = 2^8$), en particulier ce que l'on trouve sur les touches d'un clavier usuel.

Note : en Python `ord("a")` vaut 91 et `chr(91)` vaut "a".

2 Entiers naturels

2.1 Mots de taille fixe

Nous avons l'habitude d'écrire des nombres comme 8, 17, 235, 7813. Si nous décidons d'utiliser exactement quatre chiffres pour représenter un nombre, nous les écrivons 0008, 0017, 0235, 7813. Les entiers naturels représentables avec quatre chiffres sont alors tous les nombres entiers allant de 0 (0000) à 9999, soit $10^4 = 10000$ nombres.

La mémoire d'une machine ressemble à une feuille de papier quadrillées sur laquelle nous pouvons écrire un chiffre 0 ou 1 dans chaque petit carré. Chaque carré est numéroté, c'est son adresse. Les machines numériques n'utilisent que le binaire pour stocker des données sous forme de suites composées uniquement de 0 et de 1. Nous pouvons écrire en binaire des nombres comme 1, 11, 101, 1011, soient 1, 3, 5, 11 en décimal. Pour stocker ces nombres en machines il faudrait inscrire les chiffres de chaque nombre et pour chaque nombre, son nombre de chiffres. Sinon il est impossible de savoir ce que représente 10111 écrit sur une ligne de la feuille.

1	0	1	1	1

Ce sont les nombres 1011 et 1, les nombres 101 et 11 ? De plus si un nombre occupe deux cases et que nous n'avons plus besoin de ce nombre, nous pouvons l'effacer mais nous ne disposons que de deux cases contigües pour écrire un nouveau nombre. Il est donc plus efficace de décider que tous les nombres sont stockés avec exactement le même nombre de chiffres. Par exemple si nous décidons d'utiliser des mots de taille quatre bits, alors 1, 3, 5, 11 s'écrivent respectivement 0001, 0011, 0101, 1011. Et nous avons ci-dessous les nombres 1011 et 0001 sans ambigüité possible.

1	0	1	1	0	0	0	1

Avec quatre bits, nous pouvons écrire seulement $2^4 = 16$ nombres. Avec un octet, nous pouvons représenter $2^8 = 256$ nombres. C'est suffisant pour coder les différents caractères figurant sur un clavier d'ordinateur. Les calculs étant effectués par un processeur, la taille des mots est liée à la capacité du processeur. Actuellement les calculs avec des nombres sur 64 bits sont courants.

Le langage Python gère des entiers plus longs, et leur taille n'est limitée que par la capacité de la mémoire.

2.2 Méthodes

Sous forme de tableau, nous obtenons pour le nombre 8050 :

...	$10000 = 10^4$	$1000 = 10^3$	$100 = 10^2$	$10 = 10^1$	$1 = 10^0$
...		8	0	5	0
...	0	8	0	5	0

L'utilisation du chiffre 0 est déterminante.

L'écriture de 5326 en base dix est constituée des restes obtenus dans les divisions euclidiennes successives par 10 jusqu'à obtenir un quotient nul.

5326	10		
6	532	10	
	2	53	10
		3	5	10
			5	0

En base deux le principe est le même qu'en base dix.

Par exemple, l'écriture de 11 en base deux, ou l'écriture binaire de 11, est constituée des restes obtenus dans les divisions euclidiennes successives par 2 jusqu'à obtenir un quotient nul.

11	2		
1	5	2	
	1	2	2
		0	1	1
			1	0

On obtient l'écriture en base deux en lisant les restes de bas en haut, soit 1011_2, soit $r_4r_3r_2r_1$.

Écrivons les calculs sous une autre forme :
$11 = 5 \times 2 + 1 = (2 \times 2 + 1) \times 2 + 1 = ((1 \times 2) \times 2 + 1) \times 2 + 1 = 1 \times 2^3 + 1 \times 2 + 1$
d'où, $11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$.
Nous pouvons utiliser un tableau comme en base dix :

...	$16 = 2^4$	$8 = 2^3$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
...		1	0	1	1

Note :

- le bit (ou chiffre) le plus à gauche est appelé bit de poids fort ;
- le bit (ou chiffre) le plus à droite est appelé bit de poids faible.

En informatique, tout est écrit en base deux. Les données écrites sont souvent regroupées en base huit ou seize. La présentation en base seize améliore grandement la lisibilité.

Base deux	$\longleftrightarrow$	Base seize
1101	$\longleftrightarrow$	D
0011	$\longleftrightarrow$	3
1101 0011	$\longleftrightarrow$	D3

Justification de la méthode
Le nombre seize s'écrit 10000 en base deux et 10 en base seize : $10000_2 = 10_{16}$;
donc $(a_7a_6a_5a_4a_3a_2a_1a_0)_2 = (a_7a_6a_5a_4)_2 \times 10000_2 + (a_3a_2a_1a_0)_2 = (A_1)_{16} \times 10_{16} + (A_0)_{16}$
d'où : $(a_7a_6a_5a_4a_3a_2a_1a_0)_2 = (A_1A_0)_{16}$.

Remarque : pour multiplier par deux un nombre écrit en base deux, il suffit d'ajouter un zéro à droite du nombre : $1011_2 \times 10_2 = 10110_2$. Cette propriété est vraie dans une base b quelconque : le produit d'un nombre par b s'obtient en ajoutant un zéro à droite du nombre.

2.3 Représentation en machine

Un entier naturel s'écrit sans signe et on parle d'*entier non signé*.

Dans une machine, si on utilise un octet pour représenter un entier naturel, on peut représenter les entiers naturels de 0 (0000 0000 en base deux) à 255 (1111 1111 en base deux). Le nombre 45 par exemple est représenté par 0010 1101.

Si on utilise seize bits, soit deux octets, on peut représenter les entiers naturels jusqu'à 65535 (1111 1111 1111 1111 en base deux) et dans ce cas 45 est représenté par 0000 0000 0010 1101.

Avec n bits, on peut représenter les nombres entre 0 et $2^n - 1$, c'est-à-dire tout nombre k qui s'écrit :

$$k = \sum_{i=0}^{n-1} b_i 2^i \quad \text{avec } b_i \in \{0, 1\}$$

En général, un entier est codé sur 4 octets, (soit 32 bits), ou sur 8 octets, (soit 64 bits).

Addition de nombres binaires
L'addition est l'une des opérations les plus simples pour un ordinateur. Il suffit de savoir exécuter les opérations $0 + 0$, $1 + 0$, $0 + 1$ et $1 + 1$ avec une éventuelle retenue.

Pour additionner deux nombres en binaire on peut poser l'addition comme on le fait en base dix, avec le système de retenue.

On utilise les résultats : $0 + 0 = 0$, $1 + 0 = 0 + 1 = 1$ et $1 + 1 = 10$.

Par exemple :

	1	1	0	1	treize
+	1	0	0	1	neuf
1			1		retenues
1	0	1	1	0	vingt-deux

Attention, si la taille des entiers est limitée, par exemple avec quatre bits, alors dans l'addition ci-dessus le bit à gauche est perdu. Donc pour la machine, la somme de 1101 et de 1001 vaut 0110. Autrement dit, si nous demandons à la machine de calculer $13 + 9$, elle nous répond 6 !

3 Entiers signés

Considérons un ordinateur qui utilise l'arithmétique décimale et des nombres écrits avec quatre chiffres. Dix-milles nombres sont donc représentables pas des mots de 0000 à 9999. Si nous souhaitons représenter des nombres négatifs, nous pouvons envisager plusieurs possibilités.

On pourrait représenter les nombres de $+0$ à $+9999$ et les nombres de -0 à -9999. On utilise un symbole de plus et il y a deux zéros. Mais si nous n'autorisons que quatre chiffres, un chiffre est monopolisé pour le signe et il n'en reste que trois pour la valeur absolue du nombre.

Une autre possibilité serait de translater tous les nombres n de -5000 à 4999 par $n + 5000$. Une difficulté majeure avec ce choix est alors de programmer des opérations.

Si nous effectuons $9999 + 0001$ en éliminant le bit de retenue, nous obtenons 0000. Autrement dit : $0000 - 0001 = 9999$. De même $0000 - 0002 = 9998$, etc. Nous pourrions donc dire que les nombres positifs sont représenté par les nombres de 0000 à 4999 et les négatifs par les nombres de 5000 à 9999. Tous les nombres commençant par 5, 6, 7, 8, ou 9 sont négatifs et représentent les nombres de -5000 à -1. Les négatifs n sont représentés pas $n + 10^4$.

Ce type de représentation est celui utilisé avec les angles sur le cercle trigonométrique gradué de degré en degré. Les angles de -180 à 179 correspondent à des angles de 0 à 359 : les angles α positifs à α, les angles α négatifs à $\alpha + 360$. Ainsi l'angle -1 correspond à 359.

Remarque : on remplace chaque chiffre c par le chiffre $9 - c$ dans un nombre à quatre chiffres, puis on lui ajoute 1 ; par exemple $n = 3415$ donne 6584 puis $p = 6585$; alors $n + p = 10000$, soit 0 en ne gardant que quatre chiffres. Donc p représente $-n$. C'est *la représentation en complément à* 10^4.

- **Représentation en complément à deux des entiers signés**.

De manière générale, avec n bits, la représentation en machine d'un entier négatif r, est l'écriture binaire de la différence $2^n - (-r)$. Cette représentation s'appelle le complément à 2^n, souvent abrégé en complément à deux. Pratiquement, pour trouver la représentation d'un entier négatif r, on prend l'écriture binaire de $-r$ (qui est un entier positif), on inverse les bits de cette écriture et on ajoute 1. Inverser les bits et ajouter 1 sont des opérations simples pour la machine. Dans toutes ces écritures, la taille des mots, le nombre de bits utilisés, est fixée.

Exemple avec une taille de six bits, pour obtenir le codage de -12 :
001100 (codage de 12 en binaire sur 6 bits)
110011 (complément à un, on inverse chaque bit)
110100 (on ajoute 1) : représentation de -12 en complément à deux sur 6 bits.

Examinons la capacité de ce système. Avec n bits, on peut représenter les nombres compris entre -2^{n-1} et $2^{n-1} - 1$. Les nombres négatifs ont tous le bit de poids fort égal à 1.

Les écritures binaires des entiers naturels de 0 à $2^{n-1} - 1$ représentent les entiers relatifs positifs ou nul correspondants. Les écritures binaires des entiers naturels de 2^{n-1} à $2^n - 1$ représentent les entiers relatifs négatifs de -2^{n-1} à -1.

Ainsi avec un octet, soit huit bits, on peut écrire les entiers naturels n de 0 à 255 et donc représenter les entiers relatifs de $-2^7 = -128$ à $2^7 - 1 = 127$. Les nombres n de 0 à 127, (de 0000 0000 à 0111 1111 en base deux), servent à représenter les entiers relatifs positifs ou nul r avec $n = r$ et les nombres n de 128 à 255, (de 1000 0000 à 1111 1111 en base deux), représentent les entiers relatifs négatifs r avec $n = r + 256$. Avec quatre octets, on peut écrire les entiers naturels n de 0 à $2^{32} - 1$ et donc représenter les entiers relatifs de -2^{31} à $2^{31} - 1$.

Remarque : -1 est représenté par le nombre $2^n - 1$ écrit en binaire, donc par une suite de 1.

Résumé de la représentation avec trois bits :

codage binaire	000	001	010	011	100	101	110	111
entiers non signés	0	1	2	3	4	5	6	7
entiers signés	0	1	2	3	-4	-3	-2	-1