Représentation des entiers

Avez-vous déjà effectué l'opération $127 + 1$ avec des entiers de type `char` ? Si ce n'est pas le cas, sachez que l'on trouve le résultat surprenant de -128. De même, $127+2 = -127$, ce qui est tout aussi surprenant, mais au moins $127 + 2$ est égal à $(127 + 1) + 1$. Dans ce cours nous allons expliquer comment sont codés les entiers signés et non signés en machine ainsi que les concepts mathématiques qui permettent d'expliquer les opérations avec ces entiers.

1 Entiers non signés

1.1 Entiers naturels

Avant de parler de la représentation en machine des entiers, une petite piqure de rappel sur les outils mathématiques permettant de représenter les entiers naturels est nécessaire.

Théorème 1 (développement en base b). Soient $a, b \in \mathbb{N}^*$ avec $b \geq 2$. L'entier a peut s'écrire de manière unique sous la forme

$$a = \overline{a_{n-1} \dots a_0}^{(b)} = \sum_{i=0}^{n-1} a_i b^i \,,$$

avec $a_{n-1}, \dots, a_0 \in [\![0, b[\![$ et $a_{n-1} \neq 0$.

TD **Exercice 1.**

Comme vous le savez, nos ordinateurs ne manipulent que des uns et des zéros. Le développement en binaire est alors naturellement utilisé pour représenter les entiers en machine. Cependant, les nombres binaires sont relativement longs à écrire et le risque d'erreur est élevé [1], donc nous utilisons généralement l'hexadécimal pour décrire des quantités binaires. L'hexadécimal est le système de numération en base 16. On utilise les symboles `0`, …, `9`, `A`, …, `F` pour représenter les chiffres hexadécimaux. Voyons maintenant pourquoi il est simple de convertir un nombre binaire en hexadécimal et inversement.

Méthode 1 (conversion entre les bases b et b^p). Soient a et $p \geq 1$ deux entiers naturels. Alors $a = \overline{a_{n-1} \dots a_0}^{(b)} = \overline{c_{m-1} \dots c_0}^{(b^p)}$ si et seulement si

$$m = \left\lceil \frac{n}{p} \right\rceil \quad \text{et} \quad c_i = \overline{a_{p(i+1)-1} \dots a_{pi}}^{(b)} \,.$$

Exemple 1 (Conversion du binaire vers l'octal). On considère l'entier a donné par sa décomposition en binaire

$$a = \overline{a_{10} \dots a_0}^{(2)} = \overline{11\ 111\ 100\ 010}^{(2)} \,.$$

1. Pour un humain, bien entendu. Les machine ne se trompent pas, elles sont parfaites, elles.

Comme cet entier comporte $n = 11$ bits, son écriture en base $8 = 2^3$ contient $m = \lceil \frac{11}{3} \rceil = 4$ chiffres. En écrivant $a = \overline{c_3 \dots c_0}^{(8)}$, on a

$$c_3 = \overline{\underset{0}{a_{11}} a_{10} a_9}^{(2)} = 3\,, \quad c_2 = \overline{a_8 a_7 a_6}^{(2)} = 7\,, \quad c_1 = \overline{\underset{1}{a_5} a_4 a_3}^{(2)} = 4\,, \quad c_0 = \overline{\underset{0}{a_2} a_1 a_0}^{(2)} = 2\,.$$

On en déduit alors que $a = \overline{3742}^{(8)}$.

TD **Exercices 2 et 3.**

1.2 Entiers positifs sur n bits

Comme nous venons de le voir, l'écriture binaire permet de représenter exactement tous les entiers positifs et les algorithmes classiques d'addition et de multiplication s'adaptent à n'importe quelle base. Même s'il est possible de définir un type de données pouvant stocker exactement un entier de taille quelconque [2], on souhaite la plupart du temps connaître à l'avance la taille du type que l'on manipule. Ainsi, les entiers sont codés sur un nombre prédéfini de bits. On utilise par exemple un octet pour coder un type `char` et (généralement) quatre octets pour un type `int`. Voyons maintenant comment adapter les opérations sur des entiers de taille prédéfinie.

TD **Exercices 4 à 6.**

Résumé (entier non signés). Notons E l'ensemble des entiers naturels dont la décomposition binaire comporte au plus n bits.

- On a $E = [\![0, 2^n[\![$.
- Les algorithmes d'addition et de multiplication des entiers dans E sont les mêmes qu'avec des entiers dans $\mathbb{N}$ écrits en binaire, excepté qu'on ignore tous les bits dont l'indice est supérieur à n.
  Cela revient à additionner et à multiplier modulo 2^n.
- Si a et b sont dans E, les opérations $a + b$ et ab sont exactes, à condition que le résultat soit un entier représentable dans E, donc inférieur strictement à 2^n.
- Si le résultat d'une opération n'est pas représentable, on obtient l'unique entier dans E qui lui est congru modulo 2^n.

1.3 Pratique

Langage C (hexadécimal). Les entiers de C peuvent être donnés directement en hexadécimal en commençant par `0x`. Pour afficher un entier en hexadécimal on utilise les marqueurs `%x` ou `%X` dans la fonction `printf`.

```
unsigned int n = 0xC0FFEE;
printf("%d = %x = %X = %08X\n", n, n, n, n);
```

TD **Exercice 7.**

2. Enfin, limité par la taille de votre RAM, voir disque dur… Normalement, c'est suffisant.

Langage C (opérations binaires). Les opérations binaires disponibles en C sont les suivantes :

- `~a` : « non » logique bit-à-bit ;
- `a & b` : « et » logique bit-à-bit ;
- `a | b` : « ou inclusif » logique bit-à-bit ;
- `a ^ b` : « ou exclusif » logique bit-à-bit ;
- `a << n` : décalage de `n` bits à gauche ;
- `a >> n` : décalage de `n` bits à droite ;

Exemple 2 (décalages binaires). Le décalage d'un entier de `k` bits vers la gauche consiste à ajouter `k` zéros à la fin de sa décomposition binaire. Par exemple

$$\texttt{101010 << 3} = \texttt{101010}\,\texttt{000} = \texttt{101010000}\,.$$

Si l'entier est codé sur n bits, l'opération de décalage à gauche supprime les bits dont l'indice est supérieur à n. Le décalage d'un entier de `k` bits vers la droite supprime ses `k` derniers bits. Par exemple,

$$\texttt{101010000 >> 3} = \texttt{101010} \quad \text{et} \quad \texttt{101010 >> 3} = \texttt{101}\ .$$

TD **Exercices 8 et 9.**

2 Entiers signés

L'objectif de cette section est de découvrir comment sont codés les entiers signés (c'est à dire positifs ou négatifs). En effet, il existe peu d'opérations mathématiques implémentées en dur sur le processeur, faute de place. Il est donc nécessaire de définir une norme pour la représentation des entiers signés qui reste compatible avec l'addition.

Une représentation naïve des entiers signés consisterait à réserver le bit de poids fort afin d'indiquer le signe. Un entier sur huit bits aurait donc un premier bit de signe suivi de sept bits de valeur. Par convention, le 0 indiquerait un entier positif, le 1 un entier négatif. Ainsi, l'entier `10001101` sur huit bits représenterait l'entier -13 en décimal alors que l'entier `00001101` représenterait l'entier 13.

TD **Exercice 10.**

L'exercice précédent nous montre que cette représentation naïve n'est pas compatible avec l'addition déjà implémentée sur les processeurs. Il est donc nécessaire d'en trouver une autre, moins évidente certes, mais plus « vraie ».

Définition 1 (entiers signés). Soit n dans $\mathbb{N}^*$. Tout entier relatif a dans $[\![-2^{n-1}, 2^{n-1}[\![$ peut être codé sur n bits en utilisant :

- la représentation binaire de a si $a \geq 0$;
- la représentation binaire de $2^n - |a|$ si $a < 0$.

TD **Exercice 11.**

Proposition 1 (calcul de l'opposé). Soient n dans $\mathbb{N}^*$ et a un entier relatif dans $[\![-2^{n-1}, 2^{n-1}[\![$. On note c le codage sur n bits de a. Alors le codage de $-a$ s'obtient en calculant $\sim c + 1$ où $\sim c$ désigne l'inversion de tous les bits de c.

Exemple 3. Considérons les entiers signés codés sur $n = 4$ bits. On peut donc représenter les entiers dans l'intervalle $[\![-7, 8]\!]$. L'opération $7 - 4$ s'effectue en fait comme l'addition de 7 avec -4. Cela permet d'implémenter un seul algorithme pour les additions et les soustractions. On calcule donc l'opposé de 4 en inversant ses bits puis en ajoutant 1. Comme $4 = \overline{0100}$, on a $\sim 4 = \overline{1011}$ puis

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

Ainsi $-4 = \overline{1100}$. On effectue ensuite l'addition $7 + (-4)$ via l'algorithme classique, en tronquant le résultat sur 4 bits.

$$\begin{array}{rcccc} & 0 & 1 & 1 & 1 \quad = 7 \\ + & 1 & 1 & 0 & 0 \quad = -4 \\ \hline (1) & 0 & 0 & 1 & 1 \quad = 3 \end{array}$$

On retrouve alors $7 - 4 = 3$.

TD **Exercices 12 et 13.**

Résumé (entiers signés). Notons E l'ensemble des entiers relatifs représentables sur n bits.

- On a $E = [\![-2^{n-1}, 2^{n-1}[\![$.
- Les algorithmes d'addition et de multiplication des entiers dans E sont les mêmes qu'avec des entiers non signés.
- Le codage de chaque entier signé positif est identique à son codage non signé. Le codage de chaque entier signé négatif lui est congru modulo 2^n.
- Si a et b sont dans E, les opérations $a + b$ et ab sont exactes, à condition que le résultat soit un entier représentable dans E.
- Si le résultat d'une opération n'est pas représentable, on obtient l'unique entier dans E qui lui est congru modulo 2^n.
- Si c désigne le codage d'un entier, le codage de son opposé est $(\sim c) + 1$.

TD **Exercice 14.**