REPRÉSENTATION DES NOMBRES ENTIERS

À la fin de ce chapitre, je sais :

- ☞ encoder un nombre entier non signé en binaire
- ☞ encoder un nombre entier signé en complément à deux

A Encoder un entier naturel

Les ordinateur encodent les entiers en binaire sur des groupes de bits. Selon le processeur ou le circuit, les machines sont capables de proposer différentes tailles de regroupements de bits appelés *mots*. Les plus courants sont 8, 16, 32 et 64 bits.

■ **Définition 1 — Mot.** Un mot de taille n est un regroupement de n bits.

Sur p bits, on peut coder un nombre entier naturel $0 \leqslant a < 2^p$. C'est à dire que l'on peut utiliser la plage de nombre allant de 0 à 2^{p-1} exactement.

R La base 2 n'utilise que les chiffres 0 et 1, ce qui correspond également à la capacité de stocker une information selon deux états physiquement différents de la matière, chose bien maîtrisée en électronique.

Nombre de bits	Minimum	Maximum	Nombres de valeurs
8	0	255	256
16	0	65535	65536
32	0	4294967295	plus de 4 milliards
64	0	18446744073709551615	plus de 18 milliards de milliards

TABLE 1 – Plage de valeurs atteintes et nombre de valeurs encodées selon la taille des entiers non signés sur 8 , 16, 32 ou 64 bits

■ **Définition 2 — Entiers non signés.** Les électroniciens et informaticiens désignent par le terme entiers non signés les entiers naturels représentés en machine par un mot.

■ **Exemple 1 — Addition sur des entiers non signés.** Pour additionner deux nombres entiers non signés, on utilise des circuits nommés additionneurs qui additionnent les bits entre eux en signalant s'il existe une retenue [a]. Par exemple, le résultat de $1+1$ s'écrit $1_2+1_2 = 10_2$. Pour trouver le bit des unités, une simple porte de type ou exclusif suffit : $1 \oplus 1 = 0$. Pour trouver la retenue, on utilise une porte AND : $1 \wedge 1 = 1$. Cette retenue peut être propagée dans un autre additionneur. Au final, le circuit électronique exécute l'opération que nous effectuerions à la main comme suit :

$$\begin{array}{r} \text{Bit de retenue } 10110100 \\ \text{Opérande a } 01011010_2 \\ \text{Opérande b } +01001011_2 \\ \hline \text{Résultat } =10100101_2 \end{array}$$

Ces opérations ont été intensivement optimisées : les processeurs actuels sont capables d'additionner plusieurs nombres entiers de 64 bits en un seul cycle d'horloge, notamment grâce aux instructions vectorielles.

a. en anglais, la retenue se dit Carry, c'est pourquoi on note souvent ce bit C.

■ **Définition 3 — Dépassement de capacité.** Lors d'un calcul sur un type représenté sur n bits, il est possible que le résultat d'une opération soit trop grand pour être codé sur n bits. Dans ce cas, on dit qu'il y a dépassement de capacité : on ne peut plus représenter le résultat à l'aide de ce type de données.

■ **Exemple 2 — Dépassement de capacité.** On effectue des calculs avec des entiers non signés codés sur 8 bits et on souhaite additionner 168 et 192 :

$$168 + 192 = 360 > 255 = 2^8 - 1$$

Donc, on aboutit à un dépassement de capacité car on ne peut pas représenter 360 sur 8 bits. Le processeur est sensé le signaler, mais les langages peuvent se comporter différemment selon le compilateur face à cette situation.

$$\begin{array}{r} \text{Bit de retenue } 00000000 \\ \text{Opérande a } 10101000 \\ \text{Opérande b } +11000000 \\ \hline \text{Résultat } =01101000_2 \end{array}$$

Le résultat présenté par le processeur sera donc 104 et le bit de retenu sera égal à 1, ce qui signifie un dépassement de capacité. Le résultat ne peut pas être utilisé pour le calcul standard.

R Si l'objectif est de faire des calculs modulo 256 sur des entiers non signés, alors il est possible, dans certains langages, d'utiliser le dépassement de capacité et les opérateurs arithmétiques dans cet objectif.

On observe en effet que, sur l'exemple précédent, le résultat vaut $01101000_2 = 104_{10}$ et que

$$168 + 192 = 360 = 104 \bmod 256$$

L'opération et le dépassement de capacité engendre un calcul modulo 2^8 : on a juste *effacé* les bits de poids fort du résultat, le bits qui permettent d'atteindre une valeur supérieure à 255.

B Encoder un entier relatif

a Approche naïve

Si l'on cherche à représenter un entier relatif et qu'on adopte une approche naïve, on peut imaginer utiliser un bit pour désigner le signe de ce nombre. Par exemple, 0 pour le signe positif et 1 pour le signe négatif. S'en suivrait alors la représentation binaire de la valeur absolue du nombre $|a|$. Cette méthode possède néanmoins de nombreux inconvénients :

1. le nombre 0 pourrait être représenté par deux symboles différents, positif ou négatif, ce qui n'est pas souhaitable. Il est toujours préférable d'avoir une unicité lors de la représentation d'un objet.
2. l'algorithme de l'addition ne s'appliquerait qu'à des entiers de même signe et on devrait donc utiliser une autre algorithme, et donc un autre circuit électronique, pour les entiers relatifs. Or il est intéressant de pouvoir appliquer toujours le même algorithme quelle que soit la donnée, c'est-à-dire d'utiliser les mêmes circuits électroniques.

On utilise donc un encodage particulier pour représenter les entiers négatifs, encodage dit **complément à deux**.

b Complément à deux (puissance n)

■ **Définition 4 — Complément à deux.** Pour représenter les entiers relatifs en mémoire sur n bits avec la convention du complément à deux à la puissance n, on encode :

- les nombres positifs de 0 à $2^{n-1} - 1$ par leur valeur en binaire,
- les nombres **négatifs** a par les nombres positifs $2^n + a$.

Le premier chiffre d'un nombre négatif en complément à deux est donc toujours 1. Cela limite la représentation des nombres à la plage $-2^{n-1} \leqslant a \leqslant 2^{n-1} - 1$. Par contre, on peut utiliser les mêmes circuits pour additionner ou multiplier des nombres positifs ou négatifs.

■ **Exemple 3 — Encodage sur 8 bits en complément à 2.** Sur 8 bits, on peut donc normalement inscrire les entiers relatifs allant de $-128 \to 11111111_2$ à $127 \to 01111111_2$.

Pour écrire le nombre -67_{10}, on calcule $2^8 - 67 = 189$ qui s'écrit 10111101_2.

M **Méthode 1 — Trouver rapidement le complément à 2^n d'un nombre négatif** Pour cela il suffit de :

1. Prendre la valeur absolue du nombre en binaire,
2. Prendre le complément à un de ce nombre,
3. Ajouter un.

Par exemple :

$$|-121| = 121 = 01111001_2 \rightarrow_{\text{(complément à un)}} 10000110_2 \rightarrow_{+1} 10000111_2 = (-121)_{10}$$

R Le complément à un d'un nombre binaire s'obtient en remplaçant les 0 par des 1 et les 1 par des 0.

L'encodage complément à deux permet de donner une représentation **unique** sur n bits à tout nombre entier relatif appartenant à $[\![-2^{n-1}, 2^{n-1}-1]\!]$. Ces encodages sont dits **signés** ou **signed** en anglais. Ils sont déclinés pour des nombres de bits allant de 8 à 64 sur la plupart des architectures (cf. tableau 2). Ils permettent également de réaliser des opérations d'addition, de soustraction, de multiplication et de division facilement.

Nombre de bits	**Intervalle accessible (signé)**	**Intervalle accessible (non signé)**
8	$[\![-128, 127]\!]$	$[\![0, 255]\!]$
16	$[\![-32768, 32767]\!]$	$[\![0, 65535]\!]$
32	$[\![-2147483648, 2147483647]\!]$	$[\![0, 4294967295]\!]$
n	$[\![-2^{n-1}, 2^{n-1}-1]\!]$	$[\![0, 2^n-1]\!]$

TABLE 2 – Plage d'entiers accessibles en fonction du nombre de bits de la représentation signée.

c Opérations en complément à deux

■ **Exemple 4 — Addition en complément à deux.** Calculons $113 + (-91)$ comme le ferait un ordinateur. Ces deux nombres sont encodables sur 8 bits, car compris dans l'intervalle $[\![-2^7, 2^7-1]\!] = [\![-128, 127]\!]$.

$113 = (01110001)_2$ et $-91 = (10100101)_2$. On additionne de manière classique et on obtient : $133 + (-91) = (00010110)_2 = 22_{10}$

$4 = (00000100)_2$ et $-125 = (10000011)_2$. On additionne de manière classique en tronquant le résultat à 8 bits et on obtient : $4 + (-125) = (10000111)_2 = -121_{10}$

■ **Définition 5 — Dépassement de capacité.** Lorsque le résultat d'une opération sort de l'intervalle de représentation possible, c'est-à-dire $a \otimes b \notin [\![-2^{n-1}, 2^{n-1}-1]\!]$, alors le résultat n'est plus valide et on dit qu'on a dépassé les capacités de stockage du système.

■ **Exemple 5 — Addition et dépassement.** Si on utilise un encodage signé des entiers sur 8 bits pour effectuer $72 + 59$ alors il advient un dépassement de capacité. Le résultat obtenu sur 8 bits est `0b10000011` qui représente -125, ce qui n'est pas possible puisque les deux opérandes sont positives. Les processeurs savent détecter ces situations.