Représentation des données

[Mise à jour le 4/6/2021]

Sources

- Numérique et sciences informatiques 1er - ellipses

1. Introduction

Les données et les programmes stockés dans la mémoire des machines numériques sont représentés à l'aide de deux chiffres : **0** et **1**.

0 et **1** sont appelés chiffres binaires (*Binary Digits*) ou **bits**.

Ces chiffres binaires sont regroupés en paquets de **8 bits** (***octets*** ou ***bytes***) puis couramment organisés en **mots** de 2, 4 ou 8 octets. Une machine dite 32 bits manipule des mots de 4 octets lorsqu'il effectue des opérations. Ce regroupement de bits est à la base de la représentation des entiers, des réels et des caractères.

Il est nécessaire d'utiliser des codes (**encoder**) pour représenter des **entiers**, des **décimaux** et des **caractères**.

2. Encodage des entiers naturels

2.1 Écriture en base 2

Une séquence de chiffres binaires peut s'interpréter comme un nombre écrit en base 2. Dans cette base, les chiffres **0** et **1** d'une séquence sont associés à un **poids** $\mathbf{2^i}$ qui dépend de la position i des chiffres dans la séquence.

Une séquence de **n bits** b_i, $b_{n-1},b_{n-2},\ldots,b_1,b_0$ correspond au nombre décimal N tel que : $$N_{10} = b_{n-1}2^{n-1}+b_{n-2}2^{n-2}+ \ldots +b_12^1+b_02^0 = \displaystyle\sum_{i=0}^{i=n-1}b_i2^i$$

Cet encodage permet de représenter des entiers dans l'intervalle [$\mathbf{0}$,$\mathbf{2^n - 1}$]

Exemples

- Un octet (n = 8) permet de coder tous les nombres entre 0 et 255
- $10011011_2 = 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 128 + 16 + 8 + 2 + 1 = 155_{10}$

2.2 Écriture en base 16

La base 16 ou hexadécimale est souvent utilisée pour simplifier l'écriture des nombres binaires.

On passe d'un nombre en base 16 à un nombre en base 2 en regroupant les chiffres binaires par 4.

2.3 Boutisme

Sources

- Boutisme sur Wikipédia
- endianness sur Academic.com

La représentation en machine des entiers naturels sur des mots de 2, 4 ou 8 octets se heurte au problème de l'ordre dans lequel ces octets sont organisés en mémoire. Ce problème est appelé le **boutisme** (ou endianness).

- **Big endian**

Le gros boutisme consiste à placer l'**octet de poids fort en premier**, c'est-à-dire à l'adresse mémoire la plus petite.

- **Little endian**

Le petit boutisme consiste à placer l'**octet de poids faible** en premier. C'est-à-dire à l'adresse mémoire la plus petite.

Bien que le boutisme soit géré au niveau du système d'exploitation, il peut être nécessaire d'en tenir compte lorsqu'on accède aux **octets en mémoire** ou lors d'**échanges sur un réseau**.

2.4 Bases 2,10 et 16 en Python

Voir Variables, types numériques et entrées / sorties dans la console

3. Encodage des entiers relatifs (complément à 2)

Ressource

- La notation en complément à 2 sur SlidePlayer

*Le code complément à 2 permet d'effectuer simplement des **opérations arithmétiques** sur les entiers relatifs codés en binaire. Il opère toujours sur des nombres binaires ayant le **même nombre de bits** (par commodité, celui-ci est généralement un multiple de 4).* Wikipédia

Le code complément à 2 permet de représenter des entiers dans l'intervalle [$\mathbf{-2^{n-1}}$,$\mathbf{2^{n-1} - 1}$]. Le bit de **poids fort** (bit le plus à gauche) ou **MSB** (Most Significant Bit) donne le **signe** du nombre représenté (**0 ⇒ positif** ou **1 ⇒ négatif**).

Exemple
- Un octet (n = 8) permet de coder en complément à 2 tous les nombres entre -128 et 127.
- Le nombre 01110001_2 est positif et égal à 89_{10}
- Le nombre 11111111_2 est négatif et égal à -1_{10}

Obtention d'un nombre binaire en complément à 2

Avant de convertir un nombre décimal en binaire complément à 2, il faut s'assurer de représenter ce nombre avec suffisamment de bits. $\mathbf{-2^{n-1} \leq -N}$ avec N>0

Exemple : nombre de bits nécessaires pour coder $\mathbf{-125000_{10}}$
$-2^{n-1} \leq -125000$
$\Rightarrow 2^{n-1} > 125000$
$\Rightarrow \log_{10}(2^{n-1}) > \log_{10}(125000)$ or $\log_{10}(a^b) = b*\log_{10}(a)$
$\Rightarrow n-1 > \log_{10}(125000)/\log_{10}(2)$
$\Rightarrow n > (\log_{10}(125000)/\log_{10}(2)) + 1 \Rightarrow n > 17{,}93 \Rightarrow$ **n=18**

Vérification : pour n = 18, $N_{10} \in [-2^{17}, 2^{17}-1]$ soit $\mathbf{-131072 \leq N_{10} \leq 131071}$

4. Représentation approximative des nombres réels

4.1 Nombres décimaux

L'encodage des nombres ***flottants*** est inspiré de l'écriture scientifique des nombres décimaux qui se compose d'un **signe** (+ ou -), d'un nombre décimal *m* appelé ***mantisse***, compris dans l'intervalle **[1, 10[** et d'un entier relatif *n* appelé exposant. L'écriture scientifique d'un nombre décimal est de la forme $\pm m \times 10^n$.

On remarquera que **le nombre 0 ne peut pas être représenté par cette écriture**.

4.2 Norme IEEE 754

Ressource

- IEEE 754