TPinfo : **Représentation des nombres**

Objectifs pédagogiques du TP :

- Représentation des entiers positifs sur des mots de taille fixe.
- Représentation des entiers signés sur des mots de taille fixe par complément à deux.
- Distinction entre nombres réels, décimaux et flottants.
- Représentation des flottants sur des mots de taille fixe : notion de mantisse, d'exposant.
- Précision des calculs en flottants.

I Codage binaire

I.1 Nombre entier

Un ordinateur manipule des informations binaires, sous forme de `0` ou `1`. Tout nombre entier se manipule alors naturellement en base 2, formant le système **binaire**. Les nombres que nous manipulons au quotidien sont écrits en base 10, formant le système **décimal**. Il nous faut donc être capables de **transcrire** les nombres du système décimal au système binaire.

Un même nombre entier N peut s'écrire comme une suite de coefficients $(d_n d_{n-1} \dots d_1 d_0)_{10}$ en base 10 ou $(b_m b_{m-1} \dots b_1 b_0)_2$ en base 2, tels que :

$$N = d_0 \times 10^0 + d_1 \times 10^1 + \cdots + d_{n-1} \times 10^{n-1} + d_n \times 10^n = b_0 \times 2^0 + b_1 \times 2^1 + \cdots + b_{m-1} \times 2^{m-1} + b_m \times 2^m.$$

- La **valeur** de chaque coefficient d_i (resp. b_j) est comprise entre 0 et 10 (resp. 2) : $\boxed{0 \leqslant d_i < 10}$ et $\boxed{0 \leqslant b_j < 2.}$
- Le nombre $n+1$ (resp $m+1$) de coefficient est déterminé par l'**exposant maximum** n (resp. m) de la décomposition dans la base choisie. On l'obtient rapidement en calculant la partie entière du logarithme dans la base correspondante : $\boxed{n = \lfloor \log(N) \rfloor}$ et $\boxed{m = \lfloor \log_2(N) \rfloor.}$

Remarque : En informatique, on appelle **bit** chaque information valant `0` ou `1`. Un **mot** est une suite de *bits*. Un **octet** est un mot de 8 *bits*.

Exemple ou exercice d'application – *En python*

Le langage `Python` permet de transcrire un nombre décimal `num` en binaire avec la fonction `bin(num)`. Réciproquement, on obtient la transcription en décimal d'un mot binaire `bin` en ajoutant le préfixe `0b`.

Les processeurs modernes fonctionnent en effectuant des calculs sur des mots de 32 ou de 64 *bits*.

1. Déterminer l'entier décimal maximal codable sur un mot de 32 puis 64 *bits*.

I.2 Nombres entiers relatifs `int` – représentation par complément à 2

Afin de représenter des nombres entiers relatifs sur un mot n de *bits*, il suffit naïvement d'**affecter un bit au signe**, en général le premier, la suite du mot de $n-1$ servant à coder la partie entière du nombre. Avec cette méthode, on peut coder des nombres N compris entre $-2^{n-1}-1$ et $+2^{n-1}-1$.

2. Déterminer tous les entiers relatifs codables naïvement en binaire sur un mot de taille $n = 4$.
3. En additionnant *bit* à *bit*, calculer $-4+3$. Le résultat est-il correct ?

Afin de palier ce problème et ne pas avoir à modifier l'algorithme d'addition qui fonctionne pour les nombres positifs, on choisit de modifier la représentation naïve des nombres négatifs en appliquant la **représentation par complément à** 2.

- Les nombres positifs sont codés normalement, avec un `0` initial.
- Les nombres négatifs, commençant par `1`, sont codés par deux opérations successives, correspondant en pratique au calcul de $2^n - |N|$:
  - — inversion de l'écriture binaire de $|N|$ (opération binaire `not`), réalisant ainsi un complément à 1,
  - — ajout de 1 au mot binaire. L'éventuellement dépassement en bout de mot est ignoré.

4. Reprendre les deux questions précédentes et conclure.

II Nombres à virgule flottante `float`

II.1 Introduction

Le langage `Python` est capable de stocker des **nombres à virgules**. Comme l'espace de stockage n'est pas infini, il n'est capable que de stocker des **nombres décimaux**. Le stockage de la plupart des nombres rationnels et, a fortiori, des nombres algébriques ou transcendants, s'accompagne forcément d'une **erreur d'arrondi**.

Dans le système décimal, on décompose simplement les nombres à virgule comme :

- une partie entière qui s'écrit comme une somme de puissances de 10.
- une partie décimale, après la virgule, qui s'écrit comme une somme de puissances de $1/10 = 10^{-1}$

Ex : $5,375 = 5 + 0,375 = 5 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$.
Dans le système binaire, on tient le même raisonnement mais avec des puissance de 2 et de $1/2 = 2^{-1}$.
Ex : $101,011_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 4 + 0 + 1 + 0 + \frac{1}{4} + \frac{1}{8}$.
Remarque : Il y a concordance entre les parties entières et décimales entre les deux bases.

II.2 Nombre à virgule flottante

• La représentation en machine des nombres à virgule flottante (couramment appelés flottants, ou `float`) est issue de la notation scientifique, en décomposant en quatre parties : le **signe**, la **caractéristique**, la **mantisse** et la **puissance**.

Ex : $-1234,5 = -1,2345 \times 10^3$ se décompose :

$-1,2345 \times 10^3$			
$-$	1	2345	3
Signe	Caractéristique	Mantisse	Puissance

Remarques : Le signe ne peut prendre que deux valeurs. La caractéristique est un entier compris entre 1 et 9.

• Le même principe s'applique pour les `float` **en représentation binaire**, avec des puissance de 2 au lieu de 10, puisque l'écriture scientifique décimale se transpose parfaitement aux mots binaires. *Ex :* $1100_2 = 1,1_2 \times 2^3$.

5. Déterminer l'écriture binaire du nombre $1234,5$ et en déduire sa décomposition sous forme de signe, caractéristique, mantisse et puissance.

• Comme la caractéristique ne peut pas être nulle, elle doit nécessairement valoir 1 en binaire. Il n'est alors pas nécessaire de stocker cette information, qui est un implicite en binaire. Il reste donc à stocker le signe, la mantisse et la puissance. Le choix est fait de les représenter dans le sens suivant :

Représentation	Signe	Exposant	Mantisse
32 *bits*	1 *bit*	8 *bits*	23 *bits*

6. Déterminer la plage de valeurs des nombres flottants stockés sur 32 *bits* ainsi que l'écart minimal entre deux mantisses consécutives. En déduire l'écart minimal entre deux `floats` consécutifs pour un exposant n donné et donc le nombre de chiffres significatifs pertinents dans l'écriture décimale d'un `\float`.
7. Est-il possible de stocker la valeur en $\mathrm{m.s^{-1}}$ de la vitesse de la lumière sur un `float32` ?
8. Déterminer les caractéristiques des `float` de `Python` à l'aide de la méthode `float_info` du module `sys` (à importer).

Exemple ou exercice d'application – *En python*

La version 3 de `Python` est capable de dépasser automatiquement le format 64 *bits* pour le stockage des entiers en passant sur un format `long`.

On peut toutefois choisir la taille du codage grâce au module `numpy`, grâce aux fonctions `np.int8()`, `np.int16()`, `...` pour les entiers et `np.float16()`, `...` pour les flottants.

9. Observer le dépassement de capacité en ajoutant des entiers "trop grands" pour un codage sur 8 *bits*.
10. Calculer le carré de la distance Terre-Lune `d = np.array(384000000)`. Comment faire pour modifier le calcul afin que le résultat soit acceptable ?

II.3 Erreur d'arrondi et test d'égalité

Hormis la limitation naturelle de l'espace de stockage, il apparait que la plupart des nombres décimaux usuels ne présentent pas un développement binaire fini. C'est par exemple le cas de $1/10 = 0,1$, valeur assez courante en calcul numérique ! Ex : Calculer la somme `0.1+0.2`.

Lors d'un calcul unique, cette erreur d'arrondi peut être bénigne, mais ce n'est pas le cas lorsqu'on l'ajoute de nombreuses fois. C'est pour cela que les pas dans les calculs numérique doivent être choisis comme des puissances négatives de 2.

11. Écrire un programme qui somme un grand nombre de fois un pas valant `pas = 0.1` puis une puissance négative de 2. Commenter.

Ainsi, pour savoir si deux nombres flottants `a` et `b` sont égaux, on doit prendre en compte l'éventuelle erreur d'arrondi. On préférera comparer la différence `abs(a - b)` à une précision arbitraire ε donnée.

12. Écrire un programme qui permet de vérifier si la somme de `a = 0.1` et `b = 0.2` est bien égale à `c = 0.3`.

III Transcription

13. Écrire une fonction `bin2num(bin)` qui renvoie un entier contenant la transcription décimale du mot binaire `bin` (écrit comme une chaine de caractère).
14. Écrire une fonction `num2bin(nombre)` qui renvoie une chaine de caractère contenant la transcription binaire de l'entier `nombre`. Il s'agit d'effectuer des divisions successives par 2.
15. Écrire une fonction `float2bin(flot)` qui renvoie une chaine de caractère contenant la transcription binaire du nombre flottant `flot`.