Chapitre 12 : Les entiers

I) La numérisation

a) Représentation des nombres

Pour écrire des nombres entiers, nous sommes habitués à utiliser une notation de position en base 10 en utilisant des chiffres compris entre 0 et 9. Exemple : $3507 = 3 \times 100 + 5 \times 100 + 0 \times 10 + 7 \times 1$

Un entier naturel à $n+1$ chiffres peut donc s'écrire :

$$c_n c_{n-1} \dots c_1 c_0 = c_n 10^n + c_{n-1} 10^{n-1} + \dots + c_1 10^1 + c_0 * 10^0$$

En suivant le même principe, nous pouvons envisager une notation positionnelle en base b, où $b \geq 2$ est un entier en utilisant des chiffres c_i de 0 à $b-1$: $(a_n a_{n-1} \dots a_1 a_0)_b = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b^1 + a_0 b^0$

Si $b = 2$, on obtient le système binaire et les deux chiffres 0 et 1 sont appelés bits : $(c_n c_{n-1} \dots c_1 c_0)_b = c_n 2^n + c_{n-1} 2^{n-1} + \dots + c_1 2^1 + c_0 2^0$

c_0 est le poids du bit le plus faible (LSB) et c_n est le poids du bit le plus fort. Une suite composée de 0 et de 1 s'appelle un *mot* dont les *lettres* sont donc 0 ou 1 et *l'alphabet* est {0,1}

b) Méthode pour obtenir la représentation binaire

L'écriture en base 10 est constituée des restes obtenus dans les divisions euclidiennes successives par 10 jusqu'à obtenir un quotient nul, par exemple : $5326 = 6 \times 10^0 + 2 \times 10^1 + 3 \times 10^2 + 5 \times 10^3$

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

Le principe est le même en base 2 :

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

$11 = (1011)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^1 + 1 \times 2^0$ avec le bit de poids fort plus à gauche et le bit de poids faible à droite. Pour convertir en binaire un entier A sur python : $bin(A)$

c) Pourquoi le binaire ?

C'est ce système binaire qui est utilisé car il est simple à obtenir sur les différentes couches physiques :

- Dans les microprocesseurs ou clé USB, ces deux états sont électriquement simples à réaliser avec l'état haut (5V) ou bas (0V) d'une porte logique (transistor MOS),
- Sur le disque dure, il s'agit de l'orientation du moment magnétique de petites cellules
- Sur les CD, DVD, il s'agit de la présence ou de l'absence de trous
- Sur les fibres optiques, il s'agit de de jouer sur l'état de l'intensité lumineuse transmise

II) Représentation des entiers naturels en binaire

a) Les entiers naturels :

Avec n bits, on peut représenter 2^n nombres k compris entre 0 et $2^n - 1$ tels que : $k = \sum_{i=0}^{n-1} b_i 2^i$ *avec* $b_i \in \{0,1\}$.

b) Entiers relatifs ou signés

Pour décrire les entiers négatifs r avec n bits, on utilise le complément à 2 qui consiste à coder $r < 0$ de la manière suivante :

- On code $-r > 0$ sur les n bits
- On inverse les bits
- Et on ajoute 1

Par exemple, sur 6 bits : $-12 \rightarrow (001100)_2 \rightarrow (110011)_2 \rightarrow (110100)_2$

Ce qui permet de représenter les nombres compris entre -2^{n-1} et $2^{n-1} - 1$, le MSB est 0 pour les nombres positifs ou nuls et -1 pour les nombres négatifs. Exemple sur 8 bits :

uint 8	int 8
$0 \leq r \leq 255$	$-128 \leq r \leq 127$

c) Entiers-multi-précision

Avec le langage python, le nombre de bits avec lequel on souhaite faire la conversion numérique est paramétrable.

Par exemple en photo, on limite la numérisation d'un nombre à 8 bits en uint8 (les valeurs sont 256 périodiques et des risques de dépassement de capacités-overflow sont possibles) :

-256	…	….	-1	0	1…	….	…	255	256	…	…	511		
0			255	0				255	0			255		

- 255+1=0 en effet 1111 1111 + 1 = 0000 0000 avec la perte de bit
- 255+256=255
- 0-1 =255

Commenté [AM1]: Pour caster un nombre en entier, on utilise le mot cl $int(nbre)$

Commenté [AM2]: Opération qui consiste à passer du format analogique au format numérique

Commenté [AM5]: Attention à ne pas confondre la représentation des entiers sous python et sa représentation dans la mémoire.
Dans la mémoire se sont des mots de taille fixe (typiquement sur 64 bits).
Sous python plusieurs blocs peuvent être utilisés ce qui permet de manipuler de plus grand nombre que 2^{64}

Commenté [AM6]: Avec des entiers signés, on peut prévoir l'overflow en notant qu'il y a un pb si la somme de nombre de même signe donne un nombre de signe différent

Commenté [AM7]: Si on prend 3 sur $3bits$ avec un bit de signe
(1,0,1,1)
Et -3 serait associé (0,0,1,1)
La somme donnerait : $0 = (1110)$ ce qui est faux..
Le complément 2A résout ce problème et consiste graphiquement à : …

Commenté [AM8]: On donne les résultats suivant en binaire :
-1+0 = 0+1 =1
-0 + 0 =0 …

Commenté [AM9]: Donc avec des entiers signés, un système qui code sur 32 bits peut ainsi stocker le nombre maximum $2^{32} - 1$

Commenté [AM10]: Attention si la taille des entiers est limitée, alors on peut avoir des pb de bits perdus (par exemple lors d'une somme) : c'est le dépassement de capacité ou overflow

Commenté [AM3]: En cherchant à convertir 22, on s'aperçoit rapidement que cela revient à rajouter un zéro à droite. Cette propriété est vraie dans une base b …

Commenté [AM4]: Il y a aussi des raison mathématiques (par exemple en binaire, les additions conduisent à des retenues qui soient 0 ou 1), …