IF1a – Représentation des entiers

Dans un ordinateur, toutes les informations (données ou programmes) sont représentées à l'aide de deux chiffres 0 et 1, appelés chiffres binaires ou Binary Digits en anglais : *Bits*.

Dans la mémoire d'un ordinateur (RAM, ROM,…), ces chiffres binaires sont regroupés en *Octets*, c'est-à-dire par paquet de huit, appelés octets ou bytes puis organisés en mots machine ou *Words* de 2,4 ou 8 octets. Par exemple, une machine dite de 32 bits est un ordinateur qui manipule directement des mots de 4 octets : $4 \times 8 = 32$ bits.

Le fait de remplacer une donnée par un nombre s'appelle la numérisation. Il sera donc nécessaire d'inventer des encodages pour représenter ces informations.

Objectifs :
- Connaître différentes représentations des entiers naturels
- Faire la distinction entre entiers signés et les entiers non signés
- Comprendre la gestion des entiers multi-précision en Python

I – Encodage des entiers naturels

I-1) Écriture en base 10

Un nombre entier en base 10 est une séquence de chiffre entre 0 et 9. Pour calculer la valeur d'une séquence $c_{n-1} \dots c_i \dots c_1 c_0$, on affecte à chaque chiffre c_i le poids 10^i et on calcule la somme des termes :

$$N =$$

Prenons l'exemple de la séquence 63451 :

Séquence	6	3	4	5	1
Position	4	3	2	1	0
Poids	10^4	10^3	10^2	10^1	10^0

La valeur de la séquence est l'entier N calculé de la manière suivante :

$$N =$$

Dans une séquence $c_{n-1} \dots c_i \dots c_1 c_0$ de n chiffres, le chiffre c_{n-1} est celui dit de poids fort et le chiffre c_0 est celui dit de poids faible.

I-2) Écriture en base 2

Dans cette base, les chiffres (0 ou 1) d'une séquence sont associés à un poids 2^i d'une puissance de 2 qui dépend toujours de la position i des chiffres dans la séquence.

Par exemple, l'octet de bits 01001101 peut également être représenté en colonnes de manière suivante.

Séquence	0	1	0	0	1	1	0	1
Position	7	6	5	4	3	2	1	0
Poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Tel que : $N =$

Cet encodage d'entiers naturels permet de représenter des nombres de 0 à 2^{n-1}. Ainsi un entier peut représenter les entiers naturels de 0 à 255 et l'entier le plus grand représentable avec un mot de 16 bits est 65535.

I-3) Écriture en base 16

On peut utiliser de nombreuses base « b » mais avec la base 2 et la base 10, la plus utilisée est la base hexadécimale ou base 16. On utilise les chiffres pour les 10 premiers et les lettres A,B,C,D,E et F pour les 6 derniers.

Lettre	A	B	C	D	E	F
Valeur	10	11	12	13	14	15

La séquence 2A4D correspond à la représentation en colonnes suivantes.

Séquence	2	A	4	D
Position	3	2	1	0
Poids	16^3	16^2	16^1	16^0

La valeur de cette séquence correspond au nombre N suivant :

$N =$

La base 16 est souvent utilisée pour simplifier l'écriture de nombres binaires. En effet si on regroupe les nombres binaires par 4 on peut écrire le tableau de correspondance suivant.

Hexadécimal	0	1	2	3	4	5	6	7
Bits	0000	0001	0010	0011	0100	0101	0110	0111
Hexadécimal	8	9	A	B	C	D	E	F
Bits	1000	1001	1010	1011	1100	1101	1110	1111

Ainsi la séquence de bits 1010010111110011 correspond au nombre .

$$\overbrace{1010}\,\overbrace{0101}\,\overbrace{1111}\,\overbrace{0011}$$

II – Mots de taille fixe

II-1) Intérêt

Nous avons l'habitude d'écrire des nombres comme 235 et 7813. Si nous décidons d'utiliser exactement quatre chiffres pour représenter un nombre, nous les écrivons 0235, 7813. Les entiers naturels représentables avec quatre chiffres sont alors tous les nombres entiers allant de 0 à 9999, 10^4 nombres. La mémoire d'une machine ressemble à une feuille de papier quadrillées sur laquelle nous pouvons écrire un chiffre 0 ou 1 dans chaque petit carré. Chaque carré est numéroté, c'est son adresse.

Pour stocker ces nombres en machines, il faut inscrire les chiffres de chaque nombre et pour chaque nombre son nombre de chiffres.

Par exemple les nombres 2, 3 et 15 s'écrivent en base 2 :

$$0010, 0011 \; et \; 1111$$

Que l'on pourrait écrire sur la feuille quadrillée :

0	0	1	0	0	0	1	1	1	1	1	1

Mais aussi :

1	0	1	1	1	1	1	1

Mais lu par une autre personne ou machine ce chiffre peut s'interpréter de la façon suivante :

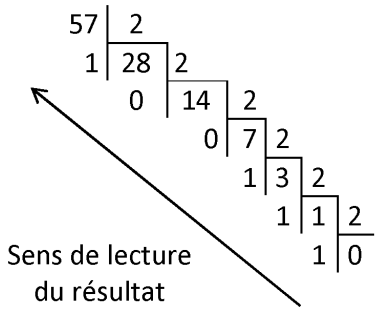
1	0	1	1	1	1	1	1

C'est-à-dire les nombres 7 et 15.

Afin d'enlever tout doute, on préfère donc travailler avec des entiers de taille fixe. Aujourd'hui les ordinateurs personnels travaillent sur des entiers de taille fixe à 64 bits.

II-2) Écriture en binaire

Pour les petits nombres il est facile d'obtenir son écriture en base 2. Pour les nombres plus élevés, il est pratique d'utiliser la division euclidienne. Prenons comme exemple le nombre 57.



Ainsi l'écriture en binaire de 57 en 8 bits est :

0	0	1	1	1	0	0	1

On rajoute les deux zéros au début de l'écriture pour avoir un entier de taille fixé à 8 bits.

II-3) Addition de nombres binaire

L'addition est l'une des opérations les plus simples pour un ordinateur. Il suffit de savoir exécuter les opérations 0+0, 1+0, 0+1 et 1+1 avec une éventuelle retenue.

Pour additionner deux nombres binaires on utilise les règles suivantes :

0+0	0
1+0 ou 0+1	1
1+1	10 ou 0 plus une retenue de 1

Exemple :

III – Représentation des entiers signés

III-1) Convention de la valeur signée

La représentation des entiers relatifs (signés) sur des emplacements ou mots de taille fixe a pu historiquement adopter la convention de la valeur signée.

Une chaîne de n bits $c_{n-1} \dots c_i \dots c_1 c_0$ représentant un entier, est interprétée de la façon suivante : le premier bit (ou chiffre binaire), c_{n-1}, est égal à 0 si l'entier est positif et à 1 sinon ; la suite $c_{n-2} \dots c_i \dots c_1 c_0$ code alors la valeur absolue. Le souci c'est qu'on a deux représentations pour zéro : 0000 et 1000 dans le cas d'une représentation sur 4 bits et en plus cela rend compliqué l'addition de nombres binaires. On est donc passé à la convention du complément à deux.

III-2) Convention du complément à 2

Pour pallier les inconvénients de la représentation précédente, on procède de la façon suivante :

- Un nombre positif ou nul est représenté par la chaîne $0c_{n-1} \dots c_i \dots c_1 c_0$;
- Pour un nombre strictement négatif, on inverse les chiffres de sa valeur absolue et on ajoute la valeur 1.
- Conséquences :
  - Le premier bit (bit de poids fort) est le bit de signe (0 si le nombre est positif, 1 sinon) ;
  - 0 n'a qu'une représentation ;
  - On représente ainsi les entiers de -2^{n-1} à $2^{n-1}-1$;

Prenons l'exemple de 5 ;

5	
Complément à 2 de 5	
-5	

D'où pour 4 bits le tableau suivant :

Entier relatif	Mot binaire	Entier relatif	Mot binaire
0	0000	-8	1000
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001

III-3) L'addition

L'avantage de la convention à deux en plus de ne pas avoir deux représentations pour zéro et de garder le même principe d'addition que pour les entiers non signés. Prenons l'exemple 14-12 et de 12-14.

On remarque ainsi que le premier résultat est bien 2 et que le second est bien -2.

IV – Entiers multi-précision

IV-1) Programmation

La plupart des processeurs calculent avec des nombres binaires de taille limitée à 32 ou 64 bits. Avec certains langages, le nombre d'octets avec lequel on travaille sur les entiers peut être précisé (un octet, deux octets, quatre octets…). Les calculs sont en général gérés directement par le processeur. Il y a certains risques qu'il faut connaître. Pour simplifier, supposons que les entiers sont codés sur quatre bits.

Si le langage nous permet de travailler sur des entiers non signés, nous avons à notre disposition les nombres de 0 à 15. Si notre programme est amené à demander le calcul 14+3, le résultat de l'addition sera 1.

14	1	1	1	0
3	0	0	1	1
Résultat	0	0	0	1

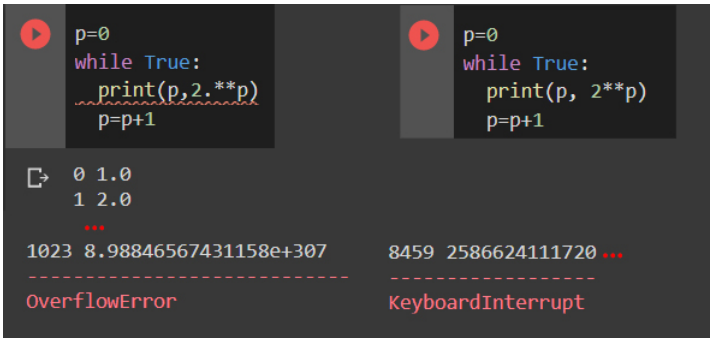
Avec des entiers signés, on aura le même problème. Pour traiter ce genre de problème, par exemple le dépassement de capacité dans une addition, on peut faire une remarque simple sur les entiers signés. Si les deux opérandes sont du même signe, il y a dépassement si le résultat n'est pas du même signe. Par exemple, avec quatre bits, 6+2 donne un résultat négatif :

6	0	1	1	0
2	0	0	1	0
Résultat	1	0	0	0

Si les deux opérandes ne sont pas du même signe, il n'y a jamais de dépassement.

IV-2) En Python

Comme nous l'avons remarqué dans ce qui précède une représentation des entiers sur 64 bits, avec la convention du complément à 2, permet de représenter tous les entiers n de -2^{63} à $2^{63}-1$. Cependant le programme suivant va avoir un comportement non attendu sous Colab en python 3x avec un ordinateur 64 bits.



Le script de gauche termine sur une erreur : comme nous le verrons dans la section suivante, le flottant $2.^{1024}$ n'est pas représentable : la représentation des flottants sur 64 bits est limitée, comme celle des entiers. Le script de droite, quant à lui, travaille avec des entiers. On attendrait qu'il s'arrête à la 64ème ligne.

Or, ce n'est pas ce qui se passe. En effet, nous l'avons stoppé au clavier pour p=8459. Nous sommes là dans le domaine des entiers multi-précision que sait gérer la classe « int » de Python.

IV-3) Représentation des entiers multi-précision

Comme la taille des nombres que l'on représente n'est pas limitée a priori il faudra les représenter par des listes dont la longueur varie avec celle du nombre.

- On écrit les entiers dans une base p:

$$X = \sum_{i=0}^{n-1} c_i p^i \text{ où } 0 \leq c_i < p$$

- On représente X par la liste des chiffres $[c_0, c_1, \ldots, c_{n-1}]$. Ainsi, en base 10, X = 30987 serait représenté par la liste $[7, 8, 9, 0, 3]$.
- Les chiffres $(c_i)_i$ sont stockés dans des registres machine et on choisit la base p le plus grand possible qui permettra ce stockage. Cela dépend évidemment du processeur.
- Il ne reste plus qu'à choisir les algorithmes pour réaliser additions, soustractions et multiplications…