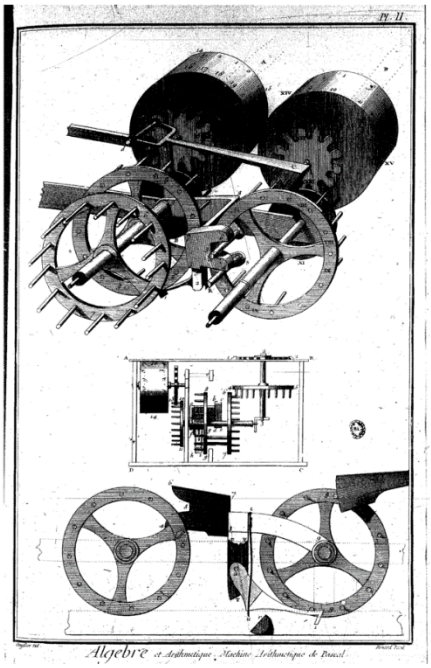
Le codage en binaire des nombres décimaux

En 1642, Blaise Pascal met au point une calculatrice afin d'aider son père dans les tâches de calculs qu'il avait à réaliser en qualité de surintendant des finances de Haute Normandie.

Source : Site Wikipédia CC BY-SA 3.0
Musée des arts et métiers Paris



Cette machine que l'on appelle la Pascaline est basée sur un système de roues dentées avec des sautoirs permettant d'effectuer des retenues à l'identique de celles que l'on met en œuvre dans les additions réalisées à la main. Pascal conçoit trois modèles :

- Un modèle classique destiné aux calculs en base 10 et qui en l'occurrence comportait des roues dentées de 10 dents ;
- Un modèle pour les calculs financiers prenant en compte le fait d'avoir un sol pour 12 deniers et une livre pour 20 sols et qui en conséquence comportait une roue de 12 dents et une de 20 dents ;
- Un modèle pour les mesures géométriques avec là-aussi des nombres de dents différents sur les roues dentées.

Source : Site Wikipédia Domaine Public
Planche de l'encyclopédie Diderot et D'Alembert

La Pascaline est donc une machine mécanique dont le fonctionnement relève du mouvement de roues dentées et sautoirs. Aujourd'hui que ce soit pour les calculatrices ou pour les ordinateurs, leur fonctionnement relève essentiellement du cheminement d'un courant électrique dans un circuit électronique. A l'identique des roues dentées de la Pascaline (comportant de 6 à 20 dents), le composant électronique élémentaire est le dipôle qui lui n'a que 2 « dents » : 0 et 1. De façon interne, la première opération qu'effectue un ordinateur lorsqu'on lui fournit un nombre est de le transcrire en une succession de 0 et de 1, c'est ainsi que l'ordinateur va utiliser tout ou partie de l'écriture binaire dudit nombre.

L'écriture en binaire des entiers

Nous proposons dans ce paragraphe de décortiquer le processus d'écriture en binaire d'un nombre entier. Pour ce faire, nous prenons l'exemple de 2023 que nous allons écrire en binaire (on dit aussi en base 2). L'idée est de réaliser à partir de 2023 des divisions successives par 2 jusqu'à épuisement.

On divise 2023 par 2 : 2023 = **1011** x 2 + **1**
On prend le quotient obtenu avant (**1011**) que l'on divise à son tour par 2 :

1011 = **505** x 2 + **1**

On prend le quotient obtenu avant (**505**) que l'on divise à son tour par 2 : **505** = **252** x 2 + **1**
On reproduit le processus : **252** = **126** x 2 + **0**
126 = **63** x 2 + **0**
63 = **31** x 2 + **1**
31 = **15** x 2 + **1**
15 = **7** x 2 + **1**
7 = **3** x 2 + **1**
On poursuit le processus jusqu'à obtenir un quotient égal à 1 : **3** = 1 x 2 + **1**
Recopions à présent la liste des restes obtenus au fur et à mesure et ajoutons à cette liste le dernier quotient 1, nous obtenons : **1 1 1 0 0 1 1 1 1 1 1**.
Il suffit à présent de recopier la liste à l'envers pour obtenir l'écriture en binaire de 2023, autrement dit : **1 1 1 1 1 1 0 0 1 1 1**.
L'entier 2023 s'écrit 11111100111 en base 2 ce qui correspond à l'égalité :

$$2023 = 1x2^{10} + 1x2^{9} + 1x2^{8} + 1x2^{7} + 1x2^{6} + 1x2^{5} + 1x2^{2} + 1x2 + 1$$

L'écriture d'un nombre décimal non entier

Nous vous proposons à présent de travailler sous Python, en téléchargeant l'environnement Spyder à l'adresse suivante : https://www.spyder-ide.org/ .

Considérons le petit problème suivant :
 	
Combien puis-je acheter des bonbons à 10 cents l'un avec un billet de 10 € ?
 	
Vous pouvez copier-coller le programme ci-contre d'en la fenêtre d'édition du logiciel Spyder.
 	
L'instruction bonbons() permet lancer son exécution.
	`def bonbons():`
	`  # On part d'un nombre de bonbons égal à 0`
	`  # et d'une somme à payer égale à 0 (en euro)`
	`  nombredebonbons =0`
	`  sommeapayer =0`
	`  # Tant que la somme à payer est différente`
	`  #  de 10 euros, on parcourt la boucle`
	`  while sommeapayer != 10:`
	`  sommeapayer = sommeapayer + 0.1`
	`  nombredebonbons=nombredebonbons+1`
	`  print(nombredebonbons, sommeapayer)`
	
	`bonbons()`

Ce programme tourne sans s'arrêter et ne renvoie pas le résultat escompté. Au cours de ses calculs, le logiciel n'arrive pas à identifier une éventuelle somme à payer égale à 10 €. Nous allons expliquer ce phénomène plutôt interpelant.

Si l'écriture des entiers en binaire ne pose pas de problèmes, il n'en va pas de même de l'écriture des entiers décimaux non entier. On retombe un peu dans les mêmes problèmes que lorsque l'on souhaite par exemple écrire 1/3 en écriture décimale où l'on obtient 1,33333333…. avec des 3 à l'infini. Dans l'écriture en binaire, le problème est d'envergure car d'entrée : **0,1 pose « problème »**.

Lorsque l'on propose à l'ordinateur de réaliser le calcul suivant : 0,1 + 0,1 à l'aide de l'instruction print(0.1+0.1), le logiciel renvoie bien le résultat attendu : 0.2. Par contre, le calcul 0,1+0,1+0,1 renvoie étonnement le résultat 0.30000000000000004.

```
print(0.1+0.1)
print(0.1+0.1+0.1)
```

Nous allons écrire 0.1 en binaire. Pour ce faire, nous effectuons cette fois-ci une succession de multiplications par 2 jusqu'à obtenir 0 (ou plus exactement dit essayer d'obtenir 0).

0,1 x 2 = **0**,2 on garde 0.

0,2 x 2 = **0**,4 on garde 0.

0,4 x 2 = **0**,8 on garde 0.

0,8 x 2 = **1**,6 on garde 1.

0,6 x 2 = **1**,2 on garde 1.

0,2 x 2 = **0**,4 on garde 0. On remarque que cette ligne est identique à celle encadrée plus haut.

…

Ces calculs nous permettent d'écrire que 0,1 s'écrit 0,00011001100110011…. en base 2. L'ordinateur ne peut conserver cette écriture à l'infini et ne prend en compte qu'une partie de celle-ci.

La norme EEE 754 simple précision

L'enregistrement d'un nombre sur un ordinateur est régi par une norme dont l'une d'entre elles (très courante) est la norme EEE 754 qui enregistre les nombres sur 32 bits. Plus exactement dit, les nombres sont réduits à la partie que l'on enregistre de **leur mantisse**. Pour 0,1 voici comment est conservée son écriture en base 2. On écrit le développement sous la forme suivante :

$$0{,}00011001100110011\ldots = 1{,}1001100110011\ldots \times 2^{-4}.$$

L'enregistrement d'un nombre suivant la norme EEE 754 se fait en trois composantes :

- Le signe : 0.1 est positif. On met la valeur 0 dans le bit qui conserve le signe.
- L'exposant : il vaut est (-4). On lui ajoute 127. Cela donne 123 que l'on écrit en base 2, soit : 1111011.
- La mantisse : elle est constitué de la partie située à droite de la virgule 1001100110011… On ne conserve que 23 valeurs, soit :

10011001100110011001100.

Signe	Exposant								Mantisse																							
1	0	1	1	1	1	0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0

On remarque que – en simple précisions - l'ordinateur conserve pour 0,1 uniquement l'écriture en binaire suivante :

0,00011001100110011001100110 1

alors que l'écriture de 0,1 se poursuit à l'infini :

0,0001100110011001100110011001100110011001100110011001100110011001100…

Conclusion

Lorsque l'on effectue des calculs que ce soit avec une calculatrice ou un ordinateur, Il convient d'évoluer toujours avec précaution. On doit ainsi prendre en compte les erreurs liées à la place mémoire forcément finie qui conduit à ce que certains nombres soient tronqués. En résumé, il est utile de toujours procéder avec ces outils en tant que roseau pensant.

Jean-Alain Roddier
IA-IPR de mathématiques