TD Corrigé : Représentation des flottants et norme IEEE754

Comment représenter des flottants de grande taille dans la mémoire d'un ordinateur ? Pour cela, il faut ruser un peu. Étudions cela…

Sur 32 bits, une idée est de prendre 1 bit pour écrire le signe, de réserver 8 bits pour coder la partie entière (non signée) et le reste pour coder la partie fractionnaire (après la virgule)

Signe Partie entière Partie fractionnaire

1) De combien de bits sera constitué la partie fractionnaire ?
   32 bits - 1 bit de signe - 8 bits de partie entière = 23 bits de partie fractionnaire
2) L'idée la plus naïve est de coder la partie entière comme nous l'avons fait avec les entiers.
   a. Quelle est la partie entière la plus grande que l'on peut écrire sur les 8 bits réservés ? Cela vous semble-t-il suffisant ? *1111 1111 = 255, ca va être vraiment limité pour coder la distance Terre Soleil*

   b. De la même manière, quelle serait la précision maximale que vous pourriez atteindre avec cette représentation ? Cela vous semble-t-il suffisant ?

   On met toutes les décimales à 0 sauf la dernière : on a alors une précision de 0,00…01 avec 22 zéros et 1 un. Ce nombre est $2^{-23} = 1{,}19 \times 10^{-7}$*, ce qui n'est pas tant que cela ! Un atome est plus petit…*

On se retrouve avec des nombres bien trop petits et/ou une précision assez limitée… Essayons d'améliorer cela !

3) Écrivez les nombres 3457064,21 et 0,0321 sous forme de notation scientifique.
Pour trouver une expression sous forme de notation scientifique, on va faire comme avec la base 2. On prend le chiffre non nul qui a la plus grand position. Cette position va être l'exposant de ma notation scientifique :
$3457064{,}21 = 3{,}45706421 \times 10^{6}$ *ou* $0{,}0321 = 3{,}21 \times 10^{-2}$

L'expression que vous avez trouvé est aussi appelée une notation mantisse-exposant.
D'une manière générale, en base 10, elle s'écrit : $\pm M \times 10^{e}$ avec $M \in [1,10[$ et $e \in \mathbb{Z}$.

4) **a.** À votre avis, que devient la notation mantisse-exposant en base 2 ?

Facile : on change tous les 10 en 2 : en base 2 : $\pm M \times 2^{e}$ avec $M \in [1,2[$ et $e \in \mathbb{Z}$.

b. À l'aide de puissance de 2, écrivez les nombres 110_2, 1001_2 puis $0{,}0101_2$ sous forme mantisse-exposant. *Pour simplifier, on écrira l'exposant de la puissance de 2 en base 10.*

$110_2 = 1{,}10 \times 2^2, \quad 1001_2 = 1{,}001 \times 2^3, \quad 0{,}0101_2 = 1{,}01 \times 2^{-2}$

Encore une fois, l'exposant correspond à la position du bit non nul le plus fort.

c. Que pouvez-vous en déduire sur la valeur de la mantisse M ? Est-il nécessaire de la stocker entièrement en base 2 ?

On voit que la mantisse commence toujours par un 1. Ce n'est d'ailleurs pas une surprise. Les nombres binaires commencent soient par un 1, soit par un 0. Mais la mantisse en base 2 est comprise entre 1 inclus et 2 exclus… Donc, la mantisse doit forcément commencer par un 1. ***On ne va pas stocker le 1,***

d. Proposez un procédé de stockage basé sur la notation mantisse-exposant. On s'aidera du tableau ci-dessous :

Signe | Exposant : max 2^{255} !! | Mantisse normalisée sur 23 bits.

e. Avec votre procédé de stockage, proposez la notation binaire sur 32 bits de -66,825.

Le signe : "-" : **1**

Convertissons 66,825 en base 2 : vous devez trouver 1000010,110100110011001100…

Mettons sous forme de mantisse-exposant : $1{,}00001011010011001100110 \times 2^6$

Mettons l'exposant en base 2 sur 8 bits : 6 = 00000110

On obtient : -66,825 = **1**00000110000010110100110011001100110

Là, on se dit qu'on est plutôt beau gosse. On a même réussi à économiser un bit dans la mantisse pour aller à une précision de 2^{-23} (comme avant en gros…).

5) Quel problème rencontre-t-on si on essaie de donner la notation binaire sur 32 bits de 0,125 ?
$0{,}125 = 2^{-3}$ et on a envie de faire du binaire signé pour coder le négatif…

Oui, il nous faut du binaire signé. On aurait pu utiliser du binaire signé sauf qu'on ne l'a pas fait dans la norme IEEE754, utilisée à peu près partout dans le monde…

L'idée a été de **rajouter** $2^8 \div 2 - 1 = 127$ à toutes les valeurs des exposants. On décale donc tous nos exposants pour avoir des valeurs positives. 127 est appelé le biais.

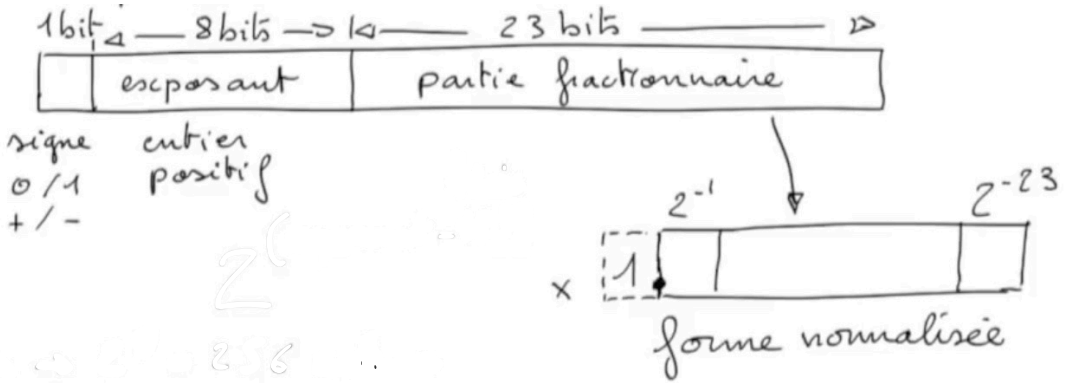
8 bits → $2^8 = 256$ valeurs
[0, 255]
− 127
[-127, 128]
[≃ 10^{-38}, 10^{38}]

Avec $0{,}125 = 2^{-3}$, l'exposant va être égal à $127 - 3 = 124$. On évite donc les négatifs.

6) Sur 8 bits, quel exposant écrire afin de coder 2^0 ? 2^1 ? 2^{128} ?

$$2^0 \rightarrow 2^{127}$$
$$2^1 \rightarrow 2^{128}$$
$$2^{128} \rightarrow 2^{255}$$

Une fois le biais pris en compte, c'est ça la norme simple précision IEEE754 pour les flottants.



7) Soit le nombre "-10,125" en base 10. Représentons-le en simple précision :
   - **a.** Écrire en base 2 le nombre $10{,}125_{10}$. **1010,00100000000000000000**
   - **b.** Écrire ce nombre sous forme mantisse-exposant puis décaler l'exposant de 127 unités.
   - **c.** Écrire l'exposant en base 2. **10000010**
   - **d.** Conclure sur la représentation du nombre les 32 bits — **11000001001000100000000000000000**
   - **e.** Écrire le nombre obtenu en hexadécimal — **C1220000, plus lisible**

C'est beau la norme IEEE754. Ca marche bien. On aime. Mais voilà…

8) Représenter le nombre "0,1" sous la norme IEEE754. **00111101110011001100110011001100**

9) Soit le nombre flottant au format simple précision : 00111101110011001100110011001100. Trouvez la représentation en base 10 de ce nombre. **0,099999994**

Petite démo Python : 0,2 + 0,1 = 0.30000000000000004

11) Déterminez la représentation au format simple précision d'un tiers (1/3) en binaire et en hexadécimal.

Pour aller plus loin :

- En notation mantisse-exposant, il est impossible de représenter le nombre 0. Pour pallier à ce problème, **l'exposant 0 est réservé.** Lorsque l'exposant vaut 0, nous pouvons avoir deux valeurs : +0 ou -0 (en fonction du signe).
- De même, **l'exposant 255 est réservé** pour des **valeurs spéciales** telles $+\infty$, $-\infty$ ou NaN.
  Lorsque, en simple précision, le nombre cherché est en-dehors de $[-1{,}7 \times 10^{38}; 1{,}7 \times 10^{38}]$, on atteint un infini : on met l'exposant à 255 et la mantisse à 0 pour signaler cela.
  Lorsque l'on réalise une opération interdite (division par 0, $\sqrt{-1}$ …), on renvoie une valeur spéciale appelée **NaN** (**N**ot **a N**umber) : on met l'exposant à 255 et la mantisse différente de 0 pour signaler ce cas.