Codage des nombres réels

Objectifs pédagogiques :

- ✓ Savoir que la notion de nombre flottant en informatique est une représentation approximative des nombres réels des mathématiques
- ✓ Calculer sur quelques exemples la représentation de nombres réels : 0.1, 0.25 ou 1/3.

Dans l'activité précédente, nous avons appris comment représenter les entiers naturels et relatifs au sein d'un ordinateur. Nous allons maintenant découvrir comment sont représentés les nombres réels, appelés en informatique nombre flottants.

Représentation des nombres réels : notion de codage en virgule fixe

> Rappel : en notation décimale, les nombres réels s'écrivent sous la forme d'une **partie entière** et d'une **partie fractionnaire** séparées par une virgule. Un nombre réel peut se décomposer en une somme de puissance de dix.

Exemple : $(1106,578)_{10} = 1 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 6 \times 10^0 + 5 \times 10^{-1} + 7 \times 10^{-2} + 8 \times 10^{-3}$

Partie entière — Partie fractionnaire

On définit une notation semblable pour tout nombre binaire représentant un réel.

Exemple : $(1010,101)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 10 + \frac{1}{2} + \frac{1}{8} = (10,625)_{10}$

On peut démontrer rigoureusement que tout nombre réel positif peut être décomposé en binaire de la manière précédente. Il reste à décrire le signe, ce qui peut être fait par un bit particulier (bit de signe) ou par une convention de type complément à 2 (voir activité précédente).

Le codage en virgule fixe s'écrit donc sous la forme : **(partie entière en binaire, partie fractionnaire en binaire)$_2$**.

Exemple : traduire en binaire (codage en virgule fixe) le nombre $(78,1875)_{10}$

Partie entière : 78

78	2
0	39 | 2
1	19 | 2
1	9 | 2
1	4 | 2
0	2 | 2
0	1

Résultat : $(78)_{10} = (1001110)_2$

Partie fractionnaire : 0,1875

0.1875 x 2 = 0,375 < 1 → 0 ($x\ 2^{-1}$)
0,375 x 2 = 0,75 < 1 → 0 ($x\ 2^{-2}$)
0,75 x 2 = 1,5 > 1 → 1 ($x\ 2^{-3}$)
0,5 x 2 : 1,0 < 1 → 1 ($x\ 2^{-4}$)

Arrêt du processus (partie décimale nulle)

Résultat : $(0,1875)_{10} = (,0011)_2$

$(78,1875)_{10} = (1001110,0011)_2$

Q1. Trouvez la représentation binaire (codage en virgule fixe) de $(54,125)_{10}$
Q2. Trouvez la représentation binaire de $(0,1)_{10}$. Que remarquez-vous ?
Q3. Trouvez la représentation décimale de $(100,001)_2$

Remarque : dans le cas de la question **Q2**, nous remarquons que le processus de "conversion" ne s'arrête pas, nous obtenons : "0,0001100110011...", le schéma "0011" se répète à "l'infini". Cette caractéristique est très importante, nous aurons l'occasion de revenir là-dessus plus tard.

> Inconvénients

Compte-tenu de la constitution interne des systèmes informatiques à base de **transistors**, les données sont représentées en **binaire** par des « **mots** » d'un certain nombre fixé de **bits**. Les **architectures 32 bits** (simple précision) et **64 bits** (double précision) s'imposent chez les fondeurs de **microprocesseurs**.

Pour simplifier, imaginons un microprocesseur 16 bits. Pour représenter les nombres réels en binaire sur cette machine, on pourrait réserver un espace (de 8 bits par exemple) pour la **partie entière** du nombre et un autre espace (de 8 bits par exemple) pour la **partie fractionnaire**. Par exemple : **XXXXXXXX, XXXXXXXX**. C'est la représentation en virgule fixe (« fixed point »). Ce mode de représentation des réels n'a que l'avantage de la simplicité. En effet, il n'est plus employé vu ses inconvénients… Ainsi l'espace réservé à la partie fractionnaire limite le nombre de bits réservés à la partie entière. Ce qui est gênant pour représenter de très grands nombres, pour lesquels la partie fractionnaire est généralement peu signifiante. Inversement, la précision sur de très petits nombres est limitée par le manque d'espace dans la partie fractionnaire alors que pour ces nombres, la partie entière ne contient que des zéros. Espace mal utilisé dans les deux cas. La représentation dite "à virgule flottante" (« Floating Point ») permet une bien meilleure préservation des chiffres significatifs autant pour les grands que pour les petits nombres, comme nous allons le voir ci-après.

Représentation des nombres réels : notion de codage en virgule flottante

Notation scientifique (rappel) : afin de représenter les nombres réels très grands ou très petits, on utilise couramment dans le système de **numération décimal** une représentation appelée **notation scientifique**.
Son format est du type :

$\pm\ a \times 10^n$ avec
- $\pm$ **signe**
- **a** : **mantisse**, nombre décimal appartenant à [1 ;10[
- **n** : **exposant**, entier relatif (positif ou négatif)

Exemples : – 0,006234 s'écrit $-6{,}234 \times 10^{-3}$
42789,97 s'écrit $4{,}278997 \times 10^{4}$

Document 1 : notation scientifique d'un nombre réel

On peut transposer ce mode de représentation des réels dans le système de numération décimal au système de numération binaire. Il suffit pour cela de remplacer la puissance de 10 par une puissance de 2.

Exemples : $(1010\ 1101{,}1011\ 0110)_2 = (1{,}010\ 1101\ 1011\ 0110 \times 2^7)_2$

$(0{,}0010\ 1101)_2 = (1{,}01101 \times 2^{-3})_2$

En binaire, la notation scientifique est du type :

$\pm\ a \times 2^n$ avec
- $\pm$ **signe**
- **a** : **mantisse**
- **n** : **exposant**

Q4. Imaginons, pour simplifier, un microprocesseur à l'architecture 16 bits ou les 8 premiers bits de gauche sont réservés à la partie entière et les 8 bits suivants à la partie fractionnaire (codage en virgule fixe) de tout nombre réel exprimé en binaire. Peut-on dans ce processeur écrire par exemple le nombre réel binaire suivant :

$(0{,}000000010110011)_2$?

> Intérêt du codage en virgule flottante

Imaginons maintenant un microprocesseur à l'architecture 16 bits ou les **8 premiers bits** à gauche sont réservés à l'**exposant** et les **8 autres** à la **mantisse** comme ci-après.

Codage en virgule flottante (16 bits) : XXXXXXXXXXXXXXXX

Exposant Mantisse

Comment dans cette nouvelle architecture, le nombre $(0{,}000000010110\ 011)_2 = (1{,}0110011 \times 2^{-8})_2$ pourra-t-il s'écrire ? Le problème revient à rechercher l'écriture de l'exposant $(-8)_{10}$ en binaire.

- $(8)_{10} = (00001000)_2$
- Complément à 1 : $(11110111)_2$
- Complément à 2^8 (on ajoute 1) : $(11111000)_2$

Donc $(-8)_{10} = (11111000)_2$

Le nombre $(0{,}000000010110\ 011)_2$ s'écrit donc 1111100010110011 dans un format à 16 bits (8 bits d'exposant et 8 bits de mantisse) à virgule flottante. Ce format ne correspond à aucune norme : il a été choisi arbitrairement dans un but pédagogique pour la simplicité des explications.

Q5. Comment écrira-t-on dans le format précédent le nombre binaire $(110001{,}01)_2$?
Q6. Même question pour $(0{,}0000010101011)_2$?

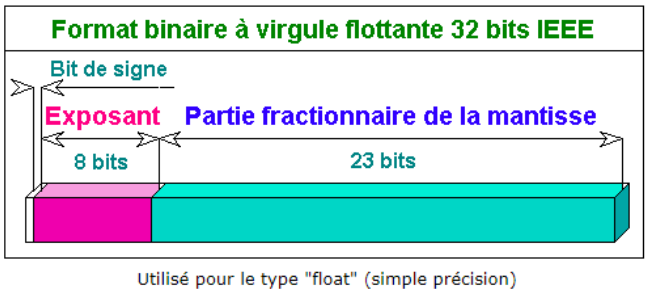
La norme IEEE 754 est la norme la plus employée pour la représentation des nombres à virgule flottante dans le domaine informatique. La première version de cette norme date de 1985.

La norme IEE 754 pour le codage en virgule flottante

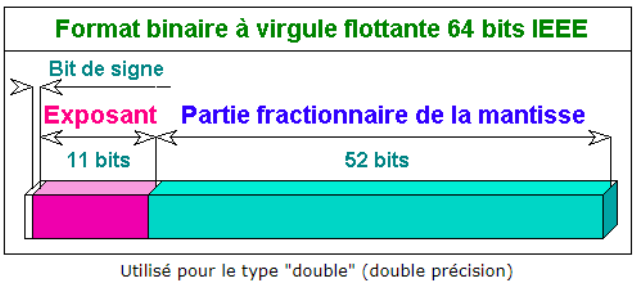
Les deux formats associés à la **norme IEE 754** sont : le format dit "simple précision" et le format dit "double précision". Le format "**simple précision**" utilise **32 bits** pour écrire un nombre flottant alors que le format "**double précision**" utilise **64 bits**.

Que cela soit en simple précision ou en double précision, la **norme IEEE754** utilise :

- **1 bit de signe** (1 si le nombre est négatif et 0 si le nombre est positif)
- **des bits consacrés à l'exposant** (8 bits pour la simple précision et 11 bits pour la double précision)
- **des bits consacrés à la mantisse** (23 bits pour la simple précision et 52 bits pour la double précision)



Utilisé pour le type "float" (simple précision)



Utilisé pour le type "double" (double précision)

Document 2 : la norme IEE 754 (32 bits et 64 bits)

Simulateur : IEE – 754 Floating Point Converter JavaScript

Pour écrire un nombre en virgule flottante en respectant la norme IEEE 754, il est nécessaire de commencer par écrire le nombre sous la forme **1,XXXXX x 2^{n}** (avec **n** l'exposant), il faut obligatoirement qu'il y ait un seul chiffre à gauche de la virgule et il faut que ce chiffre soit un "1". Par exemple le nombre "1010,11001" devra être écrit "1,01011001 x 2^{3}". Autre exemple, "0,00001001" devra être écrit "1,001 x 2^{-8}".
La partie "XXXXXX" de "1,XXXXX x 2^{n}" constitue la partie fractionnaire de la mantisse (dans notre exemple "1010,11001 = 1,01011001 x 2^{3}" la partie fractionnaire de la mantisse est "01011001"). Comme la partie fractionnaire de la mantisse comporte 23 bits en simple précision (32 bits), il faudra compléter avec le nombre de zéro nécessaire afin d'atteindre les 23 bits (si nous avons "01011001", il faudra ajouter 23 - 8 = 15 zéros à droite, ce qui donnera en fin de compte "01011001000000000000000").

ATTENTION : aucun bit pour le signe de l'exposant **n** n'a été prévu dans le norme IEEE 754, une autre solution a été choisie.

Pour le format simple précision (32 bits), 8 bits sont consacrés à l'exposant, il est donc possible de représenter 256 valeurs, nous allons pouvoir représenter des exposants compris entre $(-126)_{10}$ et $(+127)_{10}$ (les valeurs -127 et +128 sont des valeurs réservées, nous n'aborderons pas ce sujet ici). Pour avoir des valeurs uniquement positives, il va falloir procéder à un décalage : **ajouter systématiquement 127 à la valeur de l'exposant**. Prenons tout de suite un exemple (dans la suite, afin de simplifier les choses nous commencerons par écrire les exposants en base 10 avant de les passer en base 2 une fois le décalage effectué) :

Repartons de "**1010,11001**" qui nous donne "**1,01011001 x 2^{3}**", effectuons le décalage en ajoutant 127 à 3 : "**1,01011001 x 2^{130}**", soit en passant l'exposant en base 2 : "**1,01011001 x $2^{10000010}$**" car $(130)_{10} = (10000010)_{2}$. Ce qui nous donne donc pour "1010,11001" une partie fractionnaire de mantisse égale à "**01011001000000000000000**" (en ajoutant les zéros nécessaires à droite pour avoir 23 bits) et un exposant "**10000010**" (même si ce n'est pas le cas ici, il peut être nécessaire d'ajouter des zéros pour arriver à 8 bits, ATTENTION, ces zéros devront être rajoutés à gauche).

Remarque : pour le **format double précision** le **décalage** est de **1023** (il faut systématiquement ajouter 1023 à l'exposant afin d'obtenir uniquement des valeurs positives).

Exemple 1: Traduire en binaire format flottant simple précision 32 bits
le nombre : **- 1039,0**

1) Signe : 1
2) Traduire en binaire $(1039)_{10} = (0000\ 0100\ 0000\ 1111)_{2}$
3) Constituez la mantisse : **1, mantisse × 2^{n}**

0000 0100 0000 1111 = 1,00 0000 1111 × 2^{10}

(décalage de dix chiffres vers la droite après la virgule)

4) Constituez le calage IEEE en simple précision **8 bits** : $2^{8-1} - 1 = 127$
5) Constituez l'exposant : 0000 0100 0000 1111 = 1,00 0000 1111 × 2^{10}

exposant = 10 + calage = 10 +127= 137

4) Exprimer l'exposant en binaire $(137)_{10} = (1000\ 1001)_{2}$
5) Etendre la partie fractionnaire à 23 bits

mantisse sur 23 bits = **000 0001 1110 0000 0000 0000**

$(-1039,0)_{10} = (1100\ 0100\ 1000\ 0001\ 1110\ 0000\ 0000\ 0000)_{2}$
En hexadecimal C4 81 E0 00

Q7. Déterminez la représentation au format simple précision de $(10,125)_{10}$ en binaire et en hexadécimal.
Q8. Déterminez la représentation au format simple précision de $(0,25)_{10}$ en binaire et en hexadécimal.

Q9. Déterminez la représentation au format simple précision de $(0,1)_{10}$ en binaire. Qu'observez-vous ?
Q10. Soit le nombre flottant au format simple précision : 00111101110011001100110011001100. Trouvez la représentation en base 10 de ce nombre.

Q11. Dans une console PYTHON tapez l'opération suivante : 0.1 + 0.2. Qu'observez-vous ? Expliquez.
Q12. Déterminez la représentation au format simple précision d'un tiers (1/3) en binaire et en hexadécimal.

La **Pascaline**, initialement dénommée machine d'arithmétique puis roue pascaline, est une calculatrice mécanique inventée par **Blaise Pascal** et considérée comme la première machine à calculer.