1) introduction

Nous avons vu comment sont représentés les entiers relatifs au sein d'un ordinateur. Nous allons maintenant voir comment sont représentés les nombres réels, appelés ici nombres flottants.

2) représentation de la partie décimale d'un nombre

a) base 10 vers base 2

Partons tout de suite sur un exemple : comment représenter 5,1875 en binaire ?

Il nous faut déjà représenter 5, ça, pas de problème : 101

Comment représenter le ",1875" ?

- on multiplie 0,1875 par 2 : 0,1875 x 2 = 0,375. On obtient 0,375 que l'on écrira 0 + 0,375
- on multiplie 0,375 par 2 : 0,375 x 2 = 0,75. On obtient 0,75 que l'on écrira 0 + 0,75
  on multiplie 0,75 par 2 : 0,75 x 2 = 1,5. On obtient 1,5 que l'on écrira 1 + 0,5 (quand le résultat de la multiplication par 2 est supérieur à 1, on garde uniquement la partie décimale)
- on multiplie 0,5 par 2 : 0,5 x 2 = 1,0. On obtient 1,0 que l'on écrira 1 + 0,0 (la partie décimale est à 0, on arrête le processus)

On obtient une succession de "a + 0,b" ("0 + 0,375", "0 + 0,75", "1 + 0,5" et "1 + 0,0"). Il suffit maintenant de "prendre" tous les "a" (dans l'ordre de leur obtention) afin d'obtenir la partie décimale de notre nombre : 0011

Nous avons $101,0011_2$ qui est la représentation binaire de $5,1875_{10}$

b) base 2 vers base 10

Il est possible de retrouver une représentation décimale en base 10 à partir d'une représentation en binaire.

Partons de $100,0101_2$

Pas de problème pour la partie entière, nous obtenons "4". Pour la partie décimale nous devons écrire : $0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0,3125$. Nous avons donc $4,3125_{10}$

c) le cas particulier de $0,1_{10}$

Intéressons-nous à la conversion de $0,1_{10}$.

0,1 x 2 = 0,2 = 0 + 0,2

0,2 x 2 = 0,4 = 0 + 0,4

0,4 x 2 = 0,8 = 0 + 0,8

0,8 x 2 = 1,6 = 1 + 0,6

0,6 x 2 = 1,2 = 1 + 0,2

0,2 x 2 = 0,4 = 0 + 0,4

Nous constatons que la 6e ligne est identique à la 2e ligne. Le processus va donc se poursuivre sans jamais s'arrêter, car après le 0,2 x 2 = 0,4 = 0 + 0,4 nous allons retrouver :

0,4 x 2 = 0,8 = 0 + 0,8

0,8 x 2 = 1,6 = 1 + 0,6

0,6 x 2 = 1,2 = 1 + 0,2

et nous retombons donc sur 0,2 x 2 = 0,4 = 0 + 0,4…

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.

d) écrire un nombre binaire à l'aide de puissance de 2

En base dix, il est possible d'écrire les très grands nombres et les très petits nombres grâce aux "puissances de dix" (exemples $6,02.10^{23}$ ou $6,67.10^{-11}$). Il est possible de faire exactement la même chose avec une représentation binaire, puisque nous sommes en base 2, nous utiliserons des "puissances de deux" à la place des "puissances dix" (exemple : $101,1101.2^{10}$).

Pour passer d'une écriture sans "puissance de deux" à une écriture avec "puissance de deux", il suffit de décaler la virgule : $1101,1001 = 1,1011001.2^{11}$ pour passer de 1101,1001 à 1,1011001 nous avons décalé la virgule de 3 rangs vers la gauche d'où le 2^{11} (attention de ne pas oublier que nous travaillons en base 2 le 11 correspond bien à un décalage de 3 rangs de la virgule).

Si l'on désire décaler la virgule vers la droite, il va être nécessaire d'utiliser des "puissances de deux négatives" $0,0110 = 1,10.2^{-10}$, nous décalons la virgule de 2 rangs vers la droite, d'où le -10

3) représentation des flottants dans un ordinateur

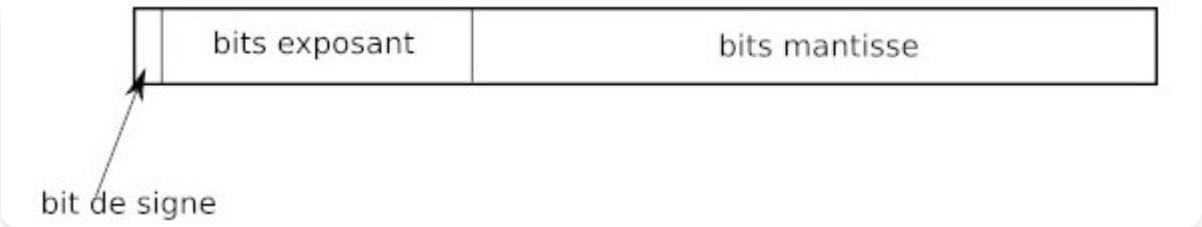
a) principe

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.

Nous allons étudier deux formats associés à cette norme : 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. Dans la suite nous travaillerons principalement sur le format 32 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)



Nous pouvons vérifier que l’on a bien 1 + 8 + 23 = 32 bits pour la simple précision et 1 + 11 + 52 = 64 bits pour la double précision.

Pour écrire un nombre flottant en respectant la norme IEEE754, il est nécessaire de commencer par écrire le nombre sous la forme $1,XXXXX.2^{e}$ (avec e 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 $1,01011001.2^{11}$. Autre exemple, 0,00001001 devra être écrit $1,001.2^{-101}$.

La partie XXXXXX de $1,XXXXX.2^{e}$ constitue la mantisse (dans notre exemple 1010,11001 la mantisse est 01011001). Comme la mantisse comporte 23 bits en simple précision, 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)

Notre première intuition serait de dire que la partie “exposant” correspond simplement au “e” de 1,XXXXX.2^e (dans notre exemple 1010,11001, nous aurions 11). En faite, c’est un peu plus compliqué que cela. En effet, comment représenter les exposants négatifs ? Aucun bit pour le signe de l’exposant n’a été prévu dans la norme IEEE754, une autre solution a été choisie :

Pour le format simple précision, 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 $1,01011001.2^3$, effectuons le décalage en ajoutant 127 à 3 : $1,01011001.2^{130}$, soit en passant l’exposant en base 2 : $1,01011001.2^{10000010}$. Ce qui nous donne donc pour 1010,11001 une mantisse 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).

À noter que 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)

Nous sommes maintenant prêts à écrire notre premier nombre au format simple précision :

Soit le nombre -10,125 en base 10 représentons-le au format simple précision :

nous avons $10_{10} = 1010_2$ et $0,125_{10} = 0,001_2$ soit $10,125_{10} = 1010,001_2$
Décalons la virgule : $1010,001 = 1,010001.2^3$, soit avec le décalage de l’exposant $1,010001.2^{130}$, en écrivant l’exposant en base 2, nous obtenons $1,010001.2^{10000010}$

Nous avons donc : notre bit de signe = 1 (nombre négatif), nos 8 bits d’exposant = 10000010 et nos 23 bits de mantisse = 01000100000000000000000

Soit en “collant” tous les “morceaux” :

11000001001000100000000000000000

Cette écriture étant un peu pénible, il est possible d’écrire ce nombre en hexadécimal : C1220000

b) problème de l’arrondi

Revenons au cas de $0,1_{10}$:

nous avons vu plus que la représentation en binaire est infinie :

$0{,}0001100110011\ldots_2$, le schéma “0011” se répète à “l’infini”

La mémoire d’un ordinateur n’étant pas infinie, nous allons devoir nous limiter à 32 bits (en simple précision) ou à 64 bits (en double précision) pour la représentation de $0{,}1_{10}$

Nous obtenons donc la représentation suivante (simple précision) :

$0{,}1_{10}$ => 00111101110011001100110011001100

Si nous procédons à la conversion inverse (conversion en base 10 du nombre flottant ci-dessus) nous n’obtenons pas 0,1 mais 0,099999994 !

Cette “légère” erreur était prévisible, elle est due à la limitation imposée (32 bits dans cet exemple).

Cette représentation avec un nombre limité de bits des nombres flottants est un problème classique en informatique. Cela peut entraîner des erreurs d’arrondi dans les calculs qui peuvent être très gênants si on n’y prend pas garde. Il faut donc être très méfiant quand on utilise des flottants pour des calculs.

Prenons un exemple simple :

```
a = 0.1
b = 0.2
c = a + b
if c == 0.3:
    print("pas de problème")
else :
    print("oups...??!")
```

Le résultat attendu après l’exécution de ce programme peut paraître évident : nous devrions voir s’afficher le message “pas de problème”…pourtant c’est le message “oups…??!” qui s’affiche !

Comme nous l’avons vu, la représentation de 0.1 en mémoire pose un problème, le résultat du calcul *a* + *b* pose donc aussi un problème (nous n’obtenons pas exactement 0.3), donc *c == 0.3* renvoie *False*…

On évitera donc de procéder à des comparaisons de flottants dans les programmes.

activité 7.1

Trouvez la représentation binaire de $4{,}125_{10}$

activité 7.2

Trouvez la représentation binaire de $0{,}25_{10}$

activité 7.3

Trouvez la représentation binaire d'un tiers (1/3)

activité 7.4

Trouvez la représentation décimale de $100{,}001_{2}$

activité 7.5

Trouvez la représentation décimale de $1{,}101_{2}$

activité 7.6

Trouvez la représentation binaire de $0{,}1_{10}$

activité 7.7

À l'aide de Spyder, tapez dans la console : 0.1 + 0.2
Quel résultat obtenez-vous ?

Ce qu'il faut savoir

- la représentation en machine des nombres réels (on parle souvent en informatique de nombres flottants) diffère de la représentation en machine des entiers
- la norme IEEE 754 est la norme la plus employée pour la représentation des nombres à virgule flottante dans le domaine informatique.
- il existe 2 formats associés à la norme IEEE 754 : le format simple précision (le nombre est représenté sur 32 bits) et le format double précision (le nombre est représenté sur 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)
- à cause de la limitation de la taille de mantisse on peut dans certains cas avoir des erreurs d'arrondies, par exemple 0.1 + 0.2 n'est pas égal à 0.3. On évitera de tester l'égalité entre 2 flottants (par exemple 0.1 + 0.2 == 0.3 retourne Faux !)

Ce qu'il faut savoir faire

- vous devez être capable de trouver la représentation en binaire d'un réel (par exemple 0.1, 0.25 ou encore 1/3)
- vous n'avez pas à savoir écrire un nombre flottant en utilisant la norme IEEE754, vous devez juste connaitre les grands principes de cette norme (bit de signe, mantisse, exposant)