Notion de nombre flottant

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.

Représentation de la partie décimale d'un nombre (conversion en binaire après la virgule)

Partons d'un exemple : comment représenter 5,1875 en binaire ?
Il nous faut déjà représenter 5, ça, pas de problème : 101
Comment représenter le ",1875" ?
La méthode est strictement l'opposé de celle qu'on utilise pour convertir la partie entière : au lieu de procéder par divisions successives par 2, on procédera par **multiplication successives par 2** :

-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 $(5,1875)_{10}$

Exercice :
Trouvez la représentation binaire de $(4,125)_{10}$

Conversion d'un nombre binaire réel en décimal :

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 allons faire comme pour la partie entiere : utiliser les puissances de 2 :
$0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0,3125$. Nous avons donc $(4,3125)_{10}$

Exercice : Trouvez la représentation décimale de $(100,001)_2$

Rq :
$2^{-1}=0.5$
$2^{-2}=0.25$
$2^{-3}=0.125$
$2^{-4}=0.0625$
etc

Exercice : Trouvez la représentation binaire de $(0,1)_{10}$
Que remarquez-vous ?

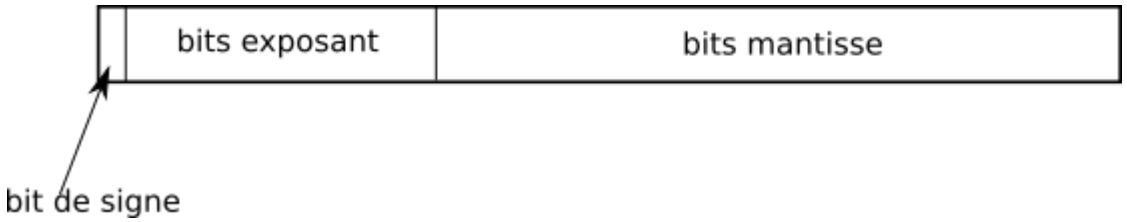
Dans l'exemple ci-dessus, nous remarquons que le processus de "conversion" ne s'arrête pas, nous obtenons : "0,0001100110011...", **le schéma "0011" se répète à "l'infini".** Cette caractéristique est très importante, nous aurons l'occasion de revenir là-dessus plus tard.

Utiliser les puissances de 2 en binaire comme on utilise les puissances de 10 en décimal

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 "$101{,}1101.2^{101}$").
Pour passer d'une écriture sans "puissance de deux" à une écriture avec "puissance de deux", il suffit de **décaler la virgule** : "$1101{,}1001 = 1{,}1011001.2^{11}$" (comme en vase 10 !) 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 gauche, 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"

Représentation des flottants dans un ordinateur :

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)



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 15 zéros à droite, ce qui donnera en fin de compte "**01011001**00000000000000")

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 le 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" = 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.$2^{10000010}$ "
Mantisse : "01011001000000000000000" (on a ajouté les zéros nécessaires à droite pour avoir 23 bits)
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 :
. Conversion en binaire : $(10)_{10} = (1010)_2$ et $(0,125)_{10} = (0,001)_2$ soit $(10,125)_{10} = \mathbf{(1010,001)_2}$
. 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.$2^{10000010}$**
.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" : **1100 0001 0010 0010 0000 0000 0000 0000**
Cette écriture étant un peu pénible, il est possible d'écrire ce nombre en hexadécimal : C1220000

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

Lire un nombre au format IEE 754 (travail inverse) :

Exemple : Soit le nombre flottant au format simple précision : 00111110100000000000000000000000
Soit en séparant bit de signe, exposant et mantisse : **0 01111101 00000000000000000000000**
. Ce nombre est positif (bit de signe à 0).
. Les 8 bits suivants nous donnent l'exposant décalé : $(01111101)_2 = (125)_{10}$.
Une fois le décalage supprimé : 125 - 127 = -2.
Les 23 bits suivants (la mantisse) sont uniquement des zéros, ce qui nous donne en fin de compte : $1,000.2^{-2}$. Ce qui donne, en base 10 également $(1,000.2^{-2})_{10}$ soit $(0,25)_{10}$.

Le cas des nombres à virgule sans fin :

Si on cherche la représentation au format simple précision de $(0,1)_{10}$ en binaire, on à un problème : comme déjà évoqué plus haut, nous nous retrouvons avec une "conversion" qui ne s'arrête jamais
0d0,1 = 0,0001100110011… = 1,10011001100… 2^{-4} => 1,10011001100… 2^{-4+127} = 1,10011001100… 2^{123}
=> 1,10011001100… $2^{1111011}$
(le schéma "0011" se répète à "l'infini"). On rappelle qu'en simple précision, la mantisse est limitée à 23 bits.
Vous devriez donc obtenir : **0**01111011100110011001100110011001100
ce qui est donc un nombre approximatif ! Eh oui, les machines calculent faux !

Exercice :

Soit le nombre flottant au format simple précision : 00111101110011001100110011001100. Trouvez la représentation en base 10 de ce nombre.

La réponse à la question posée ci-dessus est $(0,099999994)_{10}$, or, en toute logique, nous devrions trouver $(0,1)_{10}$. Cette "légère" erreur est logique quand on y réfléchit un peu. N'oubliez qu'à cause de la limitation de la mantisse à 23 bits, nous avons dû "tronquer" notre résultat (de toutes les façons, même avec une mantisse beaucoup plus grande, on aurait aussi eu le problème, car le schéma "0011" se répète à l'infini). Cette représentation avec un nombre limité de bits des nombres flottants est un problème classique en informatique. Cela peut entrainer des erreurs d'arrondi dans les calculs qui peuvent être très gênants si on n'y prend pas garde :

Experimentation :

À l'aide de Spyder, tapez dans la console : 0.1+0.2

En toute logique, nous devrions trouver 0.3 et pourtant la console affiche 0.30000000000000004. Ce problème est directement lié à la limitation du nombre de bits utilisé pour représenter les nombres flottants évoquée ci-dessus. Il existe des "astuces" pour éviter ce genre d'inconvénient, mais ce sujet ne sera pas abordé ici.

Exercice :

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