Conversion nombre en réel en flottant (32 bits)

Signe	Exposant	Mantisse
1 bit	8 bits	23 bits

Les étapes :

Étape	Description
1	Déterminer le bit de signe
2	Convertir partie décimal en binaire
3	Convertir la fraction en binaire
4	Calcul de l'exposant
5	Calcul de la mantisse

On va prendre comme exemple **12,625**

1er étape - Déterminer le bit de signe

Valeur du bit	Condition
0	Si le nombre est positif
1	Si le nombre est négatif

Dans notre exemple 15,625 >= 0 ➔ bit de signe = 0

2ème étape – Convertir partie décimal en binaire

La partie décimal de notre nombre est 12, converti en binaire : $12_{(10)} = 1010_{(2)}$

3ème étape – Convertir la fraction en binaire

Pour calculer la fraction, il faut convertir 0,625 en binaire. La conversion se fait comme pour un nombre entier normal, sauf qu'il faut utiliser les puissances de 2 négatives.

Tableau des puissances négatives :

Puissance	Résultat
2^{-1}	$\frac{1}{2} = 0{,}5$
2^{-2}	$\frac{1}{2^2} = 0{,}25$
2^{-3}	$\frac{1}{2^3} = 0{,}125$
2^{-4}	$\frac{1}{2^4} = 0{,}0625$
2^{-5}	$\frac{1}{2^5} = 0{,}03125$
…	…
2^{-23}	$\frac{1}{2^{23}} = 0{,}00000011920928955078125$

Dans notre exemple, on a un nombre « gentil » (ça me fait penser à KAT de dire ça, bref) car on voit qu'il pourra s'écrire : $1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$

Donc $0.625_{(10)} = 101_{(2)}$.

Remarque : dans notre cas, la conversion est exacte mais avec une autre fraction « méchante », par exemple 0.3, on n'arrivera jamais à approximer exactement la valeur avec des puissances négatives de 2. On prend alors les valeurs jusqu'à la 23ème puissance (taille de la mantisse).

Astuce : utiliser les puissances négatives de 2 devient rapidement un cauchemar surtout si l'on doit aller jusqu'à 2^{-23} qui donne un nombre ridiculement petit. Pour simplifier la conversion, on peut simplement multiplier par 2 (dans notre belle base 10) la fraction. Si le résultat est plus petit que 1, le bit est 0 et on continue à multiplier par 2 la nouvelle valeur, si le résultat est plus grand que 1, le bit est 1 et on soustrait 1 au résultat et on continue avec la nouvelles valeur. Enfin si le résultat est égal à 1, le bit est 1 et on stop la routine, si l'on tombe jamais sur 1 on arrête à la 23ème valeur (taille de la mantisse).

Dans notre exemple :

Calculs	Bits
$0{,}625 \cdot 2 = 1{,}25$	1
$(1{,}25 - 1) \cdot 2 = 0{,}5$	0
$0{,}5 \cdot 2 = 1$	1
stop	$101_{(2)}$

Et l'on retombe bien sur $0.625_{(10)} = 101_{(2)}$

Au final, on a **$12.625_{(10)} = 1010{,}101_{(2)}$**

4ème étape - Calcul de l'exposant

Jusqu'à maintenant, notre nombre binaire ne respecte pas la norme IEEE-754. Il faut donc normaliser ce nombre ce qui permettra de calculer la valeur de l'exposant.

Norme IEEE-754 : $flottant = signe \cdot exposant \cdot 1, mantisse$

On doit donc décaler la virgule de $1010{,}101_{(2)}$ vers la gauche jusqu'au dernier 1 qui est le bit caché.

Donc pour passer de $1010{,}101_{(2)}$ à $1{,}010101_{(2)}$ on a décalé la virgule de 3 positions donc multiplié par 2^3.

En rouge, c'est le bit caché qui est toujours à 1. En vert, c'est la mantisse « non complète ».

Pour calculer l'exposant, on sait que son décalage est de 127 (pour 32 bits évidemment). Ce qui veut dire que $2^0 = 127$ donc $2^3 = 127 + 3 = 130$. Il suffit donc de convertir $130_{(10)}$ en binaire donne $10000010_{(2)}$

Remarque : dans notre exemple, le décalage se fait sur la gauche car notre valeur décimale est plus grande que 1. Si la partie décimale se trouve entre [0 ; 1[. Exemple 0,3, notre partie décimale en binaire sera 0, donc on doit décaler la virgule sur la droite jusqu'au premier bit à 1 et par conséquent multiplier par $2^{-n\text{-décalage}}$.

5ème étape – Calcul de la mantisse

Toutes les étapes de calculs ont été faites, il ne reste plus qu'à écrire la mantisse avec une taille de 23 bits. Dans notre exemple, la mantisse est $010101_{(2)}$ et possède 6 bits, il suffit d'ajouter encore 17 bits à 0 (uniquement si le nombre est exactement approximé) pour remplir entièrement la mantisse. Si la mantisse contient déjà 23 bits, il n'y a rien à ajouter.

Finalement, on arrive au résultat suivant :

12,625		
0	10000010	01010100000000000000000
Signe (1 bit)	exposant (8 bits)	Mantisse (23 bit)

Autre exemple

-0,3 :

-0,3 < 0 ➔ bit de signe = 1

Partie décimale 0 en binaire 0.

Calcul de la fraction :

Calculs	Bits
$0{,}3 \cdot 2 = 0{,}6$	0
$0{,}6 \cdot 2 = 1{,}2$	1
$(1{,}2 - 1) \cdot 2 = 0{,}4$	0
$0{,}4 \cdot 2 = 0{,}8$	0
$0{,}8 \cdot 2 = 1{,}6$	1
$(1{,}6 - 1) \cdot 2 = 1{,}2$	1
stop	$010011\ldots_{(2)}$

On retombe sur nos pattes. On peut s'arrêter la suite est identique (0011).

Dans ce cas, on voit qu'il est impossible d'approximer exactement 0.3 avec des puissances négatives de 2. On garde les 23 premiers bits trouvés : $01001100110011001100110_{(2)}$

On a $0{,}3_{(10)} = 0{,}01001100110011001100110_{(2)}$

Pour normalisé ce nombre, il faut décaler la virgule vers la droite de 2 ou multiplier par 2^{-2}.

$0{,}01001100110011001100110_{(2)}$ ➔ ~~00~~1,$00110011001100110011_{(2)}$. En rouge, le bit caché. En vert, la mantisse « non complète ».

L'exposant vaut alors 127 + (-2) = 125. En binaire $125_{(10)} = 01111101_{(2)}$

Comme on a décalé la virgule de 2 vers la droite, on a « supprimé » les deux derniers bits de la mantisse. Faut donc recalculer les deux derniers bits. On a vu avant que 0.3 suivait une même suite (0011). Il suffit donc d'ajouter le bon bit de la suite. 0011-0011-0011-0011-0011-0 ➔ ajouter un 0 et un 1.

Donc la mantisse vaut : 00110011001100110011001

Finalement le résultat :

-0,3		
1	01111101	00110011001100110011001
Signe (1 bit)	exposant (8 bits)	Mantisse (23 bit)