Opérations flottantes

Alain MÉRIGOT

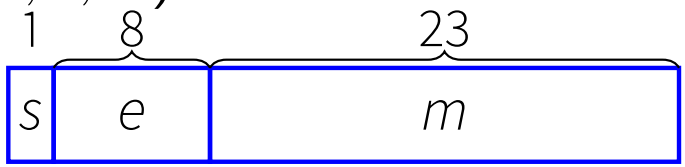
Université Paris Saclay

Nombres flottants

Rappels :

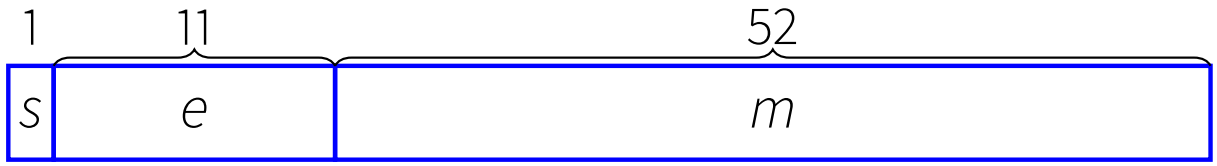
Nombre flottant = triplet (s, e, m)

En simple précision F :



$$F = -1^s \times 2^{e-127} \times 1.m$$

En double précision F :



$$F = -1^s \times 2^{e-1023} \times 1.m$$

Le bit en dernière position dans la mantisse est appelé *ulp* (*unit in the last place*).
Il détermine la *précision* des nombres (2^{-23} en SP, 2^{-52} en double)

Plan

Arrondi des nombres flottants

Le résultat d'un calcul doit être conforme à la norme IEEE-754
Les bits excédentaires seront supprimés : *arrondi du résultat*

La norme IEEE prévoit 4 modes d'arrondis :

- vers zéro : on supprime les bits à droite de l'ulp
- vers plus l'infini : on supprime les bits à droite de l'ulp et, si le résultat est positif et que certains bits supprimés étaient à 1, on ajoute 1 à l'ulp.
- vers moins l'infini : on supprime les bits à droite de l'ulp et, si le résultat est négatif et que certains bits supprimés étaient à 1, on ajoute 1 à l'ulp.
- au plus proche.

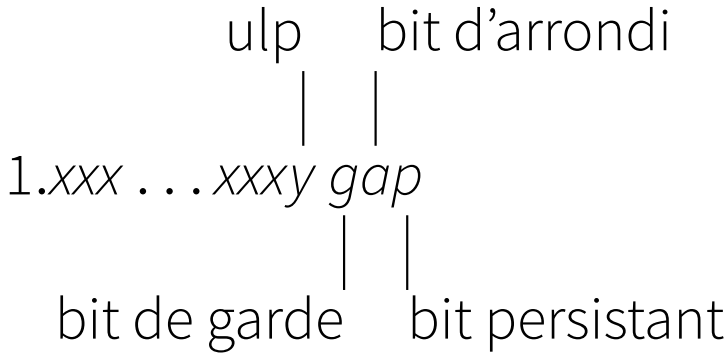
Exemple : arrondi de 1.788 sur deux digits décimaux

	vers 0	vers $+\infty$	vers $-\infty$	au plus proche
1.788	1.78	1.79	1.78	1.79
-1.788	-1.78	-1.78	-1.79	-1.79

Arrondi des nombres flottants (cont.)

arrondi au plus proche

Le cas des arrondis au plus proche implique de garder 3 bits à droite de l'ulp.
Ces bits sont nommés *g* (bit de *garde*), *a* (bit d'*arrondi*) et *p* (bit *persistant*).



Les bits de *garde* et d'*arrondi* nécessitent d'effectuer le calcul sur un nombre de bits suffisant.
Le bit *persistant* est particulier.
$p = 1$, si un des bits à sa droite est à 1.
Permet de discrimer les cas d'arrondi proche de 0.5ulp

Arrondi des nombres flottants (cont.)

Tableau pour déterminer si l'arrondi au plus proche se fait

- par troncation des bits excédentaires
- ou s'il faut en plus ajouter 1 à l'ulp

bit de garde	bit d'arrondi	bit persistant	valeur numérique	résultat arrondi
0	x	x	$0.gap < 0.5$ ulp	troncation
1	1	x	$0.gap \geq 0.75$ ulp	ajouter 1 à l'ulp
1	0	1	$0.gap = 0.5$ ulp $+ \epsilon$	ajouter 1 à l'ulp
1	0	0	$0.gap = 0.5$ ulp	vers la valeur paire

Pour éviter de faire une erreur d'arrondi systématiquement dans le même sens dans le cas où $gap = 100$ ($O.5$ ulp)
$\rightarrow$ arrondi « vers la valeur paire » (*ties to even*) de telle sorte que le résultat arrondi ait 0 en ulp.

- si ulp$= 0$, arrondi vers la valeur inférieure (troncation)
- si ulp$= 1$, arrondi vers la valeur immédiatement supérieure en ajoutant $+1$ à l'ulp.

Arrondi des nombres flottants (cont.)

Mode d'arrondi par défaut en C/C++ :

- au plus proche pour les calculs et les conversions `double` → `float`
- vers zéro pour les conversions flottant vers entier (`(int)2.9f` → 2, `(int)-3.6` → −3)

L'arrondi fait que les calculs flottants ne sont **ni associatifs, ni distributifs**

- $(a+b)+c \neq a+(b+c)$, car, $(a+b)+c \equiv$ `round(round(a+b)+c)` ce qui peut être différent de `round(a+round(b+c))` $(a+(b+c))$.
- $a \times (b+c) \neq a \times b + a \times c$ car `round(a*round(b+c))` peut être différent de `round(round(a*b)+round(a*c))`.

De même, certaines relations mathématiques peuvent ne pas être vérifiées ($a \times 1/a \neq 1.0$, $\sqrt{a \times a} \neq a$, etc)

A cause des arrondis, il est *fortement* déconseillé de tester dans un programme l'égalité de deux nombres flottants. Il est préférable de tester si la valeur absolue de leur différence est *suffisamment petite*.

Plan

Addition flottante

Soit à additionner deux nombres flottants normalisés $A = (s_a, e_a, m_a)$ et $B = (s_b, e_b, m_b)$ et le résultat $A + B = (s, e, m)$
Soit n_b le nombre de bits des mantisses.

Différentes étapes du calcul :

1. Permuter les nombres A et B si $e_a < e_b$ pour que A soit le nombre de plus grand exposant. e_a est (temporairement) l'exposant e du résultat.
2. Faire apparaitre les bits cachés de m_a et m_b ($n_b + 1$ bits)
3. Mettre les mantisses en complément à deux.
   Comme $-2 < m_a, m_b < +2$, $-4 < m_a + m_b < +4$, le résultat en C_2 et les opérandes nécessitent 3 bits à gauche de la virgule. ($n_b + 3$ bits)
4. Décaler *arithmétiquement* m_b de $e_a - e_b$ pas vers la droite.
   Pour faire l'arrondi, on doit garder les 3 bits *gap* à droite de l'ulp pour B et rajouter 3 zéros à A ($n_b + 6$ bits)
   Le bit p de B est obtenu en faisant le **OU** des bits éliminés

Addition flottante (cont.)

5. Ajouter les mantisses

   Repasser en forme signe/valeur absolue.

   Le résultat est la somme S.

6. Renormaliser S dans les cas suivants :

   a) si $2.0 \leq |S| < 4.0$, décaler S d'un cran vers la droite
   Ajouter 1 à e.
   (Peut arriver si A et B sont de même signe)

   b) si $|S| < 1.0$, chercher la position p du premier 1 de S.
   Décaler S de p pas vers la gauche.
   Soustraire p à e.
   (Peut arriver si A et B sont de signe contraire).

   Dans les deux cas, mettre à jour les bits pour les arrondis.

7. Arrondir le résultat pour le repasser à n_b bits, par troncation ou en ajoutant 1 à l'ulp si nécessaire.

8. Dans le cas d'addition d'un 1 à l'ulp, vérifier si le résultat n'atteint pas 2.0 et le renormaliser dans le cas contraire.

Addition flottante (cont.)

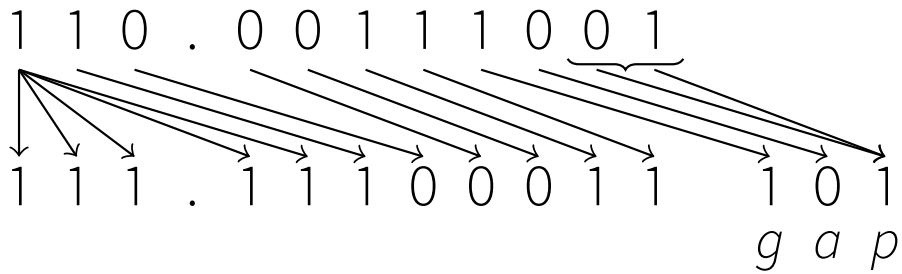
Exemple (mantisse 8 bits) :

$A = 1.10110011 \times 2^1 \quad B = -1.11000111 \times 2^{-3} \quad e_a - e_b = 4$

Passage en C_2 sur $n_b + 3$ bits.

$A = 001.10110011 \times 2^1 \quad B = 110.00111001 \times 2^{-3}$

Passage à $n_b + 6$ bits avec décalage (arithmétique) de B de 4 bits vers la droite



Calcul de la somme

$$\begin{array}{rcl} m_a & & 0\,0\,1\,.\,1\,0\,1\,1\,0\,0\,1\,1 \quad 0\,0\,0 \\ m_b & + & 1\,1\,1\,.\,1\,1\,1\,0\,0\,0\,1\,1 \quad 1\,0\,1 \\ \hline m & = & 0\,0\,1\,.\,1\,0\,0\,1\,0\,1\,1\,0 \quad 1\,0\,1 \end{array}$$

Arrondi : $gap = 101 \implies$ ajouter 1 à l'ulp

$001.10010110 + 000.00000001 = 001.10010111$

Résultat final après passage en signe-valeur absolue

$A + B = +1.10010111 \times 2^1$

Plan

Multiplication flottante

Soit à $A \times B$ avec A et B flottants normalisés $A = (s_a, e_a, m_a)$ et $B = (s_b, e_b, m_b)$ et le résultat $A \times B = (s, e, m)$
Soit n_b le nombre de bits des mantisses.

1. Faire apparaitre les bits cachés des mantisses (pour nombres normalisés)
2. Multiplier les mantisses m_a et m_b $(m = m_a \times m_b)$.
   Additionner les exposants e_a et e_b $(e = e_a + e_b)$.
   Calculer le signe du résultat $(s = s_a \oplus s_b)$.
3. si $2.0 \leq m < 4.0$,
   décaler m d'un cran vers la droite, ajouter 1 à e.

   si un des nombres est dénormalisé, on peut avoir $m < 1.0$
   Il faut alors renormaliser m et e comme pour l'addition.
4. Tous les calculs ont été effectués sur $2n_b$ bits.
   Arrondir le résultat suivant les méthodes générales, et ajouter $+1$ à l'ulp si nécessaire.
5. Détecter les débordements éventuels (résultat à 2.0 après ulp+1).