Chapitre 1

Arithmétique des Ordinateurs

1.1 Introduction

La mise en oeuvre d'une méthode numérique sur une machine amène un certain nombre de difficultés supplémentaires, liées à une nécessaire représentation approchée des nombres. Dans ce chapitre nous expliquons comment les nombres réels sont représentés par l'ordinateur sous le format "flottant", ainsi que les problème de précision et stabilité liés à cette représentation.

1.2 Notation Scientifique - Rappel

Avant d'entamer la représentation des nombres flottants, nous commençons d'abord par un petit rappel sur la notation scientifique et son intérêt mathématique.

On dit qu'une valeur x est exprimée en notation scientifique lorsqu'elle s'écrit sous la forme :

$$x = \pm a \times 10^b \tag{1.1}$$

tel que a (appelé mantisse) est un nombre décimal dont la valeur est supérieure ou égale à 1 et strictement inférieure à 10 ($1 \leq a < 10$) et b (appelé exposant) un nombre entier.

Exemple :

2.9×10^1 **est en notation scientifique**, par contre 29×10^0 **ne l'est pas** car $29 > 10$.

5×10^0 **est en notation scientifique**, par contre 5 **ne l'est pas** parce qu'il ne montre pas de multiplication par une puissance de 10.

$2.25 \times 10^{3.5}$ **n'est pas en notation scientifique** car l'exposant 3.5 n'est pas un nombre entier.

1.2.1 À Quoi ça Sert ?

La notation scientifique permet d'exprimer plus facilement de **très grades** ou **très petite** valeurs.

Exemple :

Pour la masse d'un électron, au lieu d'écrire $0.00000000000000000000000000016726kg$ il est possible d'écrire simplement $1.6726 \times 10^{-27}kg$.

Pour l'âge de l'univers, il est possible d'écrire 1.37×10^{10} au lieu de 13700000000.

1.2.2 Écrire un Nombre en Notation Scientifique

Pour écrire un nombre x sous la notation scientifique il faut considérer les **trois** possibilités suivantes :

1. $1 \leq x < 10$: dans ce cas la notation scientifique est simplement $x \times 10^0$.
2. $x < 1$: dans ce cas, l'exposant de la notation scientifique est un entier strictement inférieur à 0. Pour calculer la mantisse il faut décaler la virgule sur x **vers la droite** jusqu'à avoir un seul chiffre supérieur à 0 avant la virgule. L'exposant (commençant à 0) est **décrémenté de 1** pour chaque décalage à droite de la virgule.
3. $x \geq 10$: dans ce cas, l'exposant de la notation scientifique est un entier strictement supérieur à 0. Pour calculer la mantisse il faut décaler la virgule sur x **vers la gauche** jusqu'à avoir un seul chiffre supérieur à 0 avant la virgule. L'exposant (commençant à 0) est **incrémenté de 1** pour chaque décalage à gauche de la virgule.

Exemple :

Soit $x = 2019$, pour l'écrire sous la notation scientifique il faut décaler la virgule de **3** positions vers la gauche. Nous obtenons : $x = 2.019 \times 10^3$.

Soit $y = -0.057$, dans ce cas il faut décaler la virgule de **2** positions vers la droite. Nous obtenons $y = -5.7 \times 10^{-2}$.

1.2.3 Généralisation à d'autres Bases

La notion de la notation scientifique peut être généralisée à d'autres bases (différentes de la base 10). Dans une base donnée n (avec n un nombre entier strictement supérieur à 1), la notation scientifique d'un nombre s'écrit sous la forme $a_n \times n^{b_n}$, avec a_n un nombre réel

dans l'intervalle $[1, n[$ écrit en base n, et b_n un exposant (souvent écrit en base 10 mais il est aussi possible de l'écrire en base n).

Exemple :

Soit $x = 2019$, la notation scientifique de x en base 10 est 2.019×10^3.

Pour écrire la notation scientifique de x en base 5, nous allons d'abord l'écrire dans la même base. Nous avons $x = 2019 = 3 \times 5^4 + 1 \times 5^3 + 0 \times 5^2 + 3 \times 5^1 + 4 \times 5^0$, il s'écrit donc $x = (31034)_5$. La notation scientifique de x en base 5 est la suivante 3.1034×5^4.

En base 2, on peut faire la même chose pour trouver que $x = (11111100011)_2$, et par conséquent la notation scientifique en base 2 est la suivante $x = 1.1111100011 \times 2^{10}$.

Remarque : La notation scientifique en base 2 est très proche de la représentation des nombres flottants que nous allons détailler dans la prochaine section.

1.3 Arithmétique des Ordinateurs

Nous pouvons résumer les objectifs de cette section en deux grands points :

— Comprendre le codage des nombres réels sur les ordinateurs en format flottant.
— Connaître les limites et les conséquences de cette représentation.

1.3.1 Nombres Flottants

Les nombres flottants sont utilisés par les machines actuelles comme une approximation des nombres réels. Nous allons expliquer dans cette section comment les nombres flottants sont codés, et introduire les différents formats flottants spécifiés dans le standard IEEE-754. Ce dernier est venu pour résoudre plusieurs problèmes que rencontraient les machines de l'époque, parmi les-quelles le manque de la fiabilité et celui de la probabilité. Aujourd'hui ce standard représente la base sur laquelle appuient les machines modernes pour représenter les nombres réels. Le standard fixe trois exigences principales :

— Une représentation consistante des nombres flottants par toutes les machines adoptant le standard.
— Des opérations flottantes correctement arrondies (à détailler plus tard).
— Un traitement consistant des exceptions, tel que la division par zéro.

1.3.2 Codage des Nombres Flottants

Le standard IEEE-754 considère les nombres flottants comme un regroupement de trois parties :

— Signe : qui peut évidement être positif ou bien négatif.

— Mantisse : une chaîne binaire.

— Exposant : un entier permettant de définir le décalage de la virgule sur la mantisse.

Il est naturel d'observer que le codage des nombres flottants n'est pas trop différent de la notation scientifique vu dans la section 1.2. Sur ordinateur, il faut utiliser stocker les trois parties du nombre flottant en binaire. Pour le signe il y a deux possibilités, donc un seul bit est suffisant pour indiquer si le nombre est positif ou bien négatif (0 pour positif et 1 pour négatif). D'un autre côté, les tailles (le nombre de bits) de la mantisse est l'exposant dépendent du format utilisé. Plusieurs formats standards existent mais dans ce cours nous allons détailler les deux formats binaires les plus utilisés qui sont : *Binary32* et *Binary64* qui permettent respectivement de coder un nombre flottant sur 32 ou 64 bits.

Le format *Binary32*

Ce format code un nombre flottant sur 32 bits en utilisant :

— 1 bit de signe.

— 8 bits d'exposant.

— 23 bits de mantisse.

Le format *Binary64*

Appelé souvent format à double précision, permet de coder un nombre flottant sur 64 bits en utilisant :

— 1 bit de signe.

— 11 bits d'exposant.

— 52 bits de mantisse.

le format *Binary64* grâce à sa mantisse plus large offre plus de précision, ainsi qu'un range d'exposant supérieur permettant de représenter de plus petits ou plus grands nombres. Mais en pratique les opérations en double précision sont deux fois plus lentes (lorsqu'elles sont vectorisées).

1.3.3 Coder un Nombre au Format Flottant

Pour coder un nombre réel au format flottant, il faut suivre les étapes suivantes :

1. Convertir le nombre en question au format binaire.
2. Écrire le résultat sous la notation scientifique (toujours en binaire).

3. Coder séparément le signe, la partie fractionnaire de la mantisse, et l'exposant.

Nous allons par la suite expliquer comment coder chaque partie d'un nombre flottant. Supposant que nous voulons coder un nombre flottant X qui s'écrit comme suit

$$X = (-1)^s \times 1.F \times 2^e \tag{1.2}$$

tel que $s = 1$ pour les nombres négatifs et $s = 0$ pour les nombres positifs. F étant la partie fractionnaire de la mantisse et e l'exposant.

Codage du Signe

Pour le signe un seul bit est utilisé c'est li premier bit de la chaîne codant le nombre flottant. pour coder un signe positif la valeur 0 est utilisée, pour un signe négatif on utilise la valeur 1.

Codage de la Mantisse

Pour coder une mantisse $S = 1.F$, il suffit d'écrire la chaîne binaire F représentant la partie fractionnaire F. La partie entière de S n'a pas besoin d'être codée car nous savons qu'elle vaut toujours 1, ce qui permet de gagner un bit de plus pendant le codage d'un nombre flottant. Par exemple, pour le format *Binary32* la mantisse est codée sur 23 bits pour la partie fractionnaire plus 1 bit implicite pour la partie entière (aussi appelé bit de normalisation). La valeur de la partie entière doit être prise en compte plus tard pendant la phase de décodage. Cette étape est appelée *la normalisation.*

Codage de l'exposant

Pour coder l'exposant, e nous nous allons stocker le code binaire de $E = e + \beta$, tel que pour une taille d'exposant X nous avons :

$$\beta = 2^{X-1} - 1 \tag{1.3}$$

1.3.4 Décodage d'un nombre flottant

Pour décoder un nombre flottant et avoir le nombre réel correspondant, il faut suivre les étapes suivantes :

1. Séparer le code en *signe, mantisse et exposant* et les décoder séparément pour avoir la notation scientifique en binaire.
2. Convertir le résultat du binaire au décimal.

Décodage du Signe

Évidement, si le code du signe est **0** le nombre est positif, sinon le nombre est négatif.

Décodage de la Mantisse

Comme nous l'avons expliquer avant, seulement la partie fractionnaire de la mantisse est enregistrée, donc pour le décodage, il faut ajouter le bit de normalisation avant la virgule qui vaut toujours **1**.

Décodage de l'exposant

Finalement pour décoder l'exposant, vous devez d'abord convertir son code du binaire au décimal, puis le soustraire à β avec β défini dans l'équation 1.3.

Plus simplement, si le code du signe est s, le code de l'exposant est E et le code de la mantisse est F. La notation scientifique binaire est :

$$(-1)^S \times 1.F \times 2^{E-\beta} \tag{1.4}$$

1.3.5 Le Format *Binary8*

Le format *Binary8* n'est pas un format standard, il nous servira dans ce cours à expliquer le codage et le décodage des nombres flottants à un minimum d'effort sur une taille réduite. Nous utiliserons dans ce cas :

— 1 bit de signe.

— 4 bits d'exposant (avec $\beta = 2^{4-1} - 1 = 7$).

— 3 bits de mantisse (+ 1 bit de normalisation).

Exemple de codage sur *Binary8*

Nous allons coder les trois nombres suivants au format *Binary8* :
A = 5, B = 0.625, C = -52.

Codage de A :

1. Écrire A sous la notation scientifique : $A = (5)_{10} = (101)_2 = 1.01 \times 2^2$.
2. Codage du signe, exposant et mantisse :
   — Le bit de signe vaut **0** pour un nombre positif.
   — Le code de l'exposant est le code binaire de $2 + \beta = 2 + 7 = 9 = (1001)_2$

— Le code de la mantisse est la partie fractionnaire de la notation scientifique : **010**. Si la taille de la partie fractionnaire est inférieure à la taille de la mantisse dans le format utilisé, il faudra ajouter des zéros à droite de la mantisse.

Nous obtenons comme résultat : 01001010
Les détails du résultat sont expliqués dans le tableau suivant :

signe	exposant	mantisse
0	1001	010

Codage de B :

1. Écrire B sous la notation scientifique : $B = (0.625)_{10} = (0.101)_2 = 1.01 \times 2^{-1}$.
2. Codage du signe, exposant et mantisse :
   — Le bit de signe vaut **0** pour un nombre positif.
   — Le code de l'exposant est le code binaire de $-1 + \beta = -1 + 7 = 6 = (0110)_2$
   — Le code de la mantisse est la partie fractionnaire de la notation scientifique : **010**.

Nous obtenons comme résultat : 00110010
Les détails du résultat sont expliqués dans le tableau suivant :

signe	exposant	mantisse
0	0110	010

Codage de C :

1. Écrire C sous la notation scientifique : $C = (-52)_{10} = (-110100)_2 = -1.101 \times 2^5$.
2. Codage du signe, exposant et mantisse :
   — Le bit de signe vaut **1** pour un nombre négatif.
   — Le code de l'exposant est le code binaire de $5 + \beta = 5 + 7 = 12 = (1100)_2$
   — Le code de la mantisse est la partie fractionnaire de la notation scientifique : **101**.

Nous obtenons comme résultat : 11100101
Les détails du résultat sont expliqués dans le tableau suivant :

signe	exposant	mantisse
1	1100	101

Exemple de décodage sur *Binary8*

Nous allons décoder les trois chaînes binaires obtenus dans la section précédente. Les chaînes binaires à décoder sont : 01001010, 00110010 et 11100101. Nous devons obtenir les mêmes nombres A, B et C que nous avons codés.

Décodage de 01001010 : Dans ce cas, nous avons

— Le bit de signe $S = 0$.

— Le code de l'exposant est $E = (1001)_2 = 9$.

— La partie fractionnaire de mantisse est $F = 010$.

Pour le décodage nous allons utiliser la formule suivante :

$$\begin{aligned}(-1)^S \times 1.F \times 2^{E-\beta} &= (-1)^0 \times 1.010 \times 2^{9-7} \\ &= 1.010 \times 2^2 \\ &= (101)_2 = 5.\end{aligned}$$

Décodage de 00110010 : Dans ce cas, nous avons

— Le bit de signe $S = 0$.

— Le code de l'exposant est $E = (0110)_2 = 6$.

— La partie fractionnaire de mantisse est $F = 010$.

Pour le décodage nous allons utiliser la formule suivante :

$$\begin{aligned}(-1)^S \times 1.F \times 2^{E-\beta} &= (-1)^0 \times 1.010 \times 2^{6-7} \\ &= 1.010 \times 2^{-1} \\ &= (0.101)_2 = 0.625.\end{aligned}$$

Décodage de 11100101 : Dans ce cas, nous avons

— Le bit de signe $S = 1$.

— Le code de l'exposant est $E = (1100)_2 = 12$.

— La partie fractionnaire de mantisse est $F = 101$.

Pour le décodage nous allons utiliser la formule suivante :

$$\begin{aligned}(-1)^S \times 1.F \times 2^{E-\beta} &= (-1)^1 \times 1.101 \times 2^{12-7} \\ &= -1.101 \times 2^5 \\ &= (-110100)_2 = -52.\end{aligned}$$

1.3.6 L'arrondi

Lorsqu'on doit coder un nombre réel sous le format flottant, on pourrais tomber sur des cas ou la partie fractionnaire de la mantisse est infini, comme c'est le cas pour les nombres irrationnels, ou même pour certains nombres qui ont une représentation avec partie fractionnaire finie en base 10, mais pas en base 2. Il est aussi possible que le nombre soit simplement représentable en base 2 sous un nombre fini de bits, mais avec une partie fractionnaire de taille supérieure à la taille de la mantisse dans le format utilisé. Dans ces plusieurs différents cas une opération d'arrondi est nécessaire pour pouvoir que l'opération de codage soit accomplie.

Le standard IEEE-754 définit plusieurs modes d'arrondi, mais dans la suite de ce cours nous n'utilisons que le mode d'arrondi au plus proche, qui consiste simplement à choisir le nombre flottant (réel exactement représentable dans le format utilisé) le plus proche au nombre qu'on a envie de coder. Dans le cas d'égalité lorsqu'on code un nombre qui tombe exactement au milieu entre deux nombres flottants, on choisit celui avec une mantisse paire (ayant 0 à la fin).

Le principe de l'opération d'arrondi des nombres flottants n'est pas très différent de l'arrondi à l'unité permettant d'arrondir un nombre réel à l'entier le plus proche. Donc pour un nombre X qui n'est pas représentable, on représente $\widehat{X}$ qui est le nombre représentable le plus proche de X. On dit dans ce cas que le nombre flottant correspondant à X est $\widehat{X}$, et on note :

$$fl(X) = \widehat{X} \tag{1.5}$$

Si le nombre X est exactement représentable, alors simplement $fl(X) = X$.

Exemples avec le format *Binary8*

Supposons que nous voulons coder les nombres suivants au format *Binary8* : 45, 110 et 19. Comme dans les exemples précédents nous commençons par l'écriture de ces nombres sous la forme scientifique en binaire, ce qui donne :

$$(45)_{10} = 1.01101 \times 2^5$$
$$(110)_{10} = 1.10111 \times 2^6$$
$$(19)_{10} = 1.0011 \times 2^4$$

Nous allons ignorer les détails de codage du signe et de l'exposant car nous nous intéressons seulement à l'arrondi de la mantisse. Il est évident d'observer que la partie fractionnaire des trois nombres est de taille supérieure au nombre de bits utilisés pour coder la mantisse au format *Binary8*, ici une opération d'arrondi est nécessaire pour coder le nombre représentable le plus proche.

pour la mantisse du premier nombre, les deux nombres représentables inférieur et supérieur sont :

$$1.011 < 1.01101 < 1.100$$

Le nombre inférieur (1.011) est choisi ici car il est plus proche, donc

$$fl(45) = 1.011 \times 2^5 = (101100)_2 = (44)_{10}$$

Pour le codage de "110" nous avons :

$$1.101 < 1.10111 < 1.110$$

Dans ce cas nous prenons le nombre supérieur car il est plus proche, et on obtient

$$fl(110) = 1.110 \times 2^6 = (1110000)_2 = (112)_{10}$$

Finalement, pour le codage du dernier nombre, nous observons que sa mantisse tombe exactement au point milieu entre deux nombres représentables qui sont :

$$1.001 < 1.0011 < 1.010$$

Dans ce cas nous choisissons 1.010 parce que c'est la mantisse qui finit avec 0. Le résultat de l'arrondi est donc

$$fl(19) = 1.010 \times 2^4 = (10100)_2 = (20)_{10}$$

Erreurs d'arrondi

Avant de clôturer la discussion sur l'opération d'arrondi, il faut signaler qu'on pratique les erreurs d'arrondi en pratique sont largement moins significatives par rapport à ce qu'on montre dans les exemples précédents, et cela grâce à l'utilisation de formats qui

réserve significativement plus d'espace à la mantisse pour avoir une meilleure précision de calcul. Par exemple, si on utilise le format standard *Binary64*, la partie fractionnaire de la mantisse peut contenir jusqu'à 52 bits. En d'autres termes, le résultat arrondi est précis jusqu'à 52 bits après la virgule en binaire (presque 16 chiffres après la virgule en décimal).

Formellement, avec un format qui utilise p bits pour coder la mantisse, l'erreur absolue commise par une opération d'arrondi sur un nombre X donnant comme résultat un nombre flottant $\widehat{X}$ est majorée par l'équation suivante :

$$\left|\widehat{X} - X\right| \leq |X| \times 2^{p+1} \tag{1.6}$$

En pratique, il est plus important de regarder l'erreur relative donnée par l'équation suivante :

$$\left|\frac{\widehat{X} - X}{X}\right| \leq u \tag{1.7}$$

avec u étant la précision machine donnée par

$$u = 2^{p+1} \tag{1.8}$$

1.4 Opérations Flottantes

La différence entre les opérations flottantes et les opérations classiques est que les opérations flottantes exigent que le résultat soit représentable au format flottant utilisé, alors qu'une opération classique, peut accepter des résultats irrationnels. Le standard IEEE-754 exige que le résultat des opérations base (+, -, ×, /) soit correctement arrondi (selon le mode d'arrondi utilisé), ce qui signifie dans le contexte de notre cours que le résultat de l'opération flottante doit être le nombre représentable le plus proche au résultat exacte. Nous notons $\oplus$, $\ominus$, $\otimes$ et $\oslash$ les opérations flottante correspondant respectivement aux opérations réelles +, -, × et /, tel que :

$$\begin{aligned} a \oplus b &= fl(a + b) \\ a \ominus b &= fl(a - b) \\ a \otimes b &= fl(a \times b) \\ a \oslash b &= fl(a/b) \end{aligned}$$

1.4.1 Addition et Soustraction

Pour effectuer l'addition de deux nombres flottants, il est recommandé de suivre les étapes suivantes :

1. Aligner les mantisses : il n'est pas possible d'additionner proprement deux nombres en notation scientifique ayant deux exposants différents. Donc il faut d'abord modifier l'exposant de l'un des deux nombre, par exemple ajouter la différence entre les deux exposants à l'exposant minimal et décaler la virgule sur la mantisse du nombre flottant correspondant relativement.
2. Additionner les deux mantisses : une opération binaire simple lorsque les deux nombres à additionner ont le même exposant.
3. Normalisation : le résultat lui même doit être écrit sous la notation scientifique une fois calculé, donc avec un seul chiffre avant la virgule.
4. Arrondir : finalement, si la mantisse ne peut pas être exactement représentable dans le format utilisé, une opération d'arrondi est nécessaire.

Il est parfois nécessaire aussi de faire une deuxième normalisation après l'opération l'arrondi si cette dernière change (par propagation) la valeur du chiffre avant la virgule, mais ce cas arrive très rarement en pratique.

Pour la soustraction, les mêmes étapes de l'addition doivent être suivies, il faut seulement soustraire les mantisses au lieu de les additionner.

1.4.2 Multiplication et Division

Soit A et B deux nombres flottants qui peuvent s'écrire sous la forme suivante :

$$A = M_A \times 2^{E_A}$$
$$B = M_B \times 2^{E_B}$$

Nous avons donc

$$\begin{aligned} A \times B &= M_A \times 2^{E_A} \times M_B \times 2^{E_B} \\ &= (M_A \times M_B) \times 2^{E_A + E_B} \end{aligned}$$

de même pour la division nous avons

$$A/B = (M_A \times 2^{E_A})/(M_B \times 2^{E_B})$$
$$= (M_A/M_B) \times 2^{E_A - E_B}$$

Donc pour multiplier ou diviser deux nombres flottants A et B, il faut :

1. Calculer l'exposant du résultat en Additionnant les deux exposants de A et B pour la multiplication, alors que pour la division il faut soustraire l'exposant de B à A.
2. Multiplier ou diviser les deux mantisse.
3. Normaliser et arrondir le résultat.

1.4.3 Exemples Avec le Format *Binary8*

Soit A et B deux nombres flottants exactement représentables au format *Binary8* avec :

$$A = 1.001 \times 2^3$$
$$B = 1.110 \times 2^2$$

Nous allons calculer $A \oplus B$, $A \ominus B$, $A \otimes B$ et $A \oslash B$.

Calcul de l'addition :

1. La première étape est l'alignement des deux exposants, dans ce cas nous allons décaler la virgule sur la mantisse de B d'une position à gauche est augmenter de 1 son exposant pour obtenir :

$$1.001 \times 2^3 + 1.110 \times 2^2 = 1.001 \times 2^3 + 0.1110 \times 2^3$$

2. Maintenant il est possible de prendre l'exposant comme facteur commun, et calculer la somme des deux mantisses :

$$1.001 \times 2^3 + 0.1110 \times 2^3 = (1.001 + 0.1110) \times 2^3$$
$$= 10 \times 2^3$$

3. Maintenant il faut normaliser le résultat (garder un seul bit avant la virgule) :

$$10 \times 2^3 = 1.0 \times 2^4$$

4. Dans ce cas, aucune opération d'arrondi n'est nécessaire car la partie fractionnaire de la mantisse est exactement représentable sur le format *Binary8*.

Calcul de la Soustraction : La même approche que pour l'addition va être utilisée :

$$\begin{aligned} A \ominus B &= fl(A - B) \\ &= fl(1.001 \times 2^3 - 1.110 \times 2^2) \\ &= fl(1.001 \times 2^3 - 0.1110 \times 2^3) \\ &= fl(0.010 \times 2^3) \\ &= fl(1.0 \times 2^1) \\ &= 1.0 \times 2^1 \end{aligned}$$

Calcul de la Multiplication :

$$\begin{aligned} A \otimes B &= fl(A \times B) \\ &= fl(1.001 \times 2^3 \times 1.110 \times 2^2) \\ &= fl(1.001 \times 1.1102^{3+2}) \\ &= fl(1.001 \times 1.1102^{3+2}) \\ &= fl(1.11111 \times 2^5) \\ &= 10 \times 2^5 \\ &= 1.0 \times 2^6 \end{aligned}$$

Il y a deux point intéressants à signaler dans cet exemple, premièrement, la mantisse du résultat exacte de la multiplication "1.11111" devait être arrondi vers le haut car elle n'est par représentable sur le format *Binary8*. Deuxièmement, le résultat final a du être re-normalisé après l'arrondi puisqu'il n'avait pas un seul chiffre avant la virgule.

Calcul de la Division :

$$\begin{aligned} A \oslash B &= fl(A/B) \\ &= fl((1.001 \times 2^3)/(1.110 \times 2^2)) \\ &= fl((1.001/1.110) \times 2^{3-2}) \\ &= fl((1.001/1.110) \times 2^{3-2}) \\ &= fl(0.10100\underline{100} \cdots \times 2^1) \\ &= fl(1.0100\underline{100} \cdots \times 2^0) \\ &= 1.010 \times 2^0 \end{aligned}$$

1.5 Codage Spécial

Jusqu'ici nous n'avons parlé que des nombres flottants normalisés, mais en effet selon *le code* qu'on donne à l'exposant, il est aussi possible de coder des nombres flottants dénormalisés, ou même des valeurs spéciales comme : NaN, $\pm\infty$.

1.5.1 Les Dénormalisés

Pour les formats flottants standards qu'on évoqués, le code de l'exposant minimal (contenant que des zéros) est utilisé pour coder les nombres dénormalisés. Les nombres dénormalisés ont un exposant fixe (qui dépend du format) et n'ont pas de bit de normalisation (le bit implicite vaut zéro), donc ils ne sont pas codés à partir de leur notation scientifique. La valeur de l'exposant pour les dénormalisés prend toujours la valeur minimale de l'exposant pouvant être codée pour les nombres normaux.

1.5.2 Valeurs Spéciales

Pour les formats flottants standards qu'on évoqués, le code de l'exposant maximal (contenant que des uns) est utilisé pour coder les valeurs spéciales suivantes :

- NaN : acronyme de "Not a Number" qui signifie simplement que le résultat obtenu n'est pas un nombre ou que l'opération effectuée n'est pas valide. Ce résultat peut être généré des opérations comme : 0 / 0, $\sqrt{-1}\cdots$ etc.
- $\pm\infty$: ce résultat apparaît en cas de dépassement de le l'intervalle des nombres normaux représentables (exposant du résultat supérieur à l'exposant max qu'on peut coder), ou bien dans le cas d'une division par zéro.

Le code de la mantisse est utilisé pour faire la différence entre ces deux valeurs spéciales. Une mantisse contenant que des zéros signifie que la valeur codée est $\pm\infty$, le bit du signe

Code de l'exposant	Valeur de l'exposant	Valeur codée
$(00000000)_2 = 0$	-126	$0.F \times 2^{-126}$
$(00000001)_2 = 1$	1 - 127 = -126	$1.F \times 2^{-126}$
$(00000010)_2 = 2$	2 - 127 = -125	$1.F \times 2^{-125}$
$(00000011)_2 = 3$	3 - 127 = -124	$1.F \times 2^{-124}$
...	...	...
n	n - 127	$1.F \times 2^{n-127}$
...	...	...
$(11111110)_2 = 254$	254 - 127 = 127	$1.F \times 2^{127}$
$(11111111)_2 = 255$	Pas d'exposant	$\pm\infty$ si F ne contient que des zéros, *Nan* sinon

TABLE 1.1 – Caption

fait la différence entre l'infini positif et négatif. Sinon la valeur codée est *Nan*.

Pour récapituler, dans un format flottant utilisant X bits d'exposant, avec un biais d'exposant $\beta = 2^{X-1} - 1$, nous 2^X codes possibles. Le code ayant la valeur minimale est réservé pour les dénormalisés, alors que la valeur maximale est réservée pour les valeurs *Nan* et $\pm\infty$. Ce qui laisse $2^X - 2$ codes d'exposant possible pour les nombres normalisés.

La tableau 1.1 montre la valeur à décoder pour chaque valeur d'exposant dans le format standard *Binary32* qui utilise 8 bits d'exposant.

1.6 Exemples Intéressants

Les opérations flottantes étant différentes des opérations réelles, peuvent engendrer des phénomènes imprédictibles et même bizarre, et qui peuvent apparaître comme des bugs logiciels lorsqu'on ne sait pas vraiment comment les expliquer. Nous avons fait ici une petite sélection de quelques opération/traitement qui ont des résultats réels facile à prédire, mais qui donnent des résultats manifestement erronés à cause des erreurs engendrées par l'arithmétique des ordinateurs.

Pour tous les exemples qui suivent dans ce chapitre ou les chapitres suivants, nous allons utiliser le langage Matlab. Par défaut Matlab utilise le type *double* pour coder les nombres flottants, qui est basé sur le format standard *Binary64*. Pour que nos exemples soit compréhensibles, nous allons **forcer** Matlab à utiliser le type *single* qui est basé sur le format *Binary32* en utilisant simplement le mot clé single à l'initialisation des variables numériques.

***A* + *B* = *A*, pour $B \neq 0$:** On dirait que B se fait absorber par A. Ce phénomène peut arriver dans les cas ou la différence entre l'exposant de A et de B est supérieur à p (la taille de la mantisse). Dans le format *Binary32*, un exemple simple permettant d'expliquer ce phénomène est l'accumulation de la valeur "1" avec "2^{-24}" comme le montre la figure 1.1.

```
>> A = single(1);
>> B = single(2 ^ -24);
>> A + B

ans =

     1

>> A + B + B + B + B

ans =

     1

>> A + 2 * B

ans =

   1.0000001
```

FIGURE 1.1 – B se faire absorber par A

On peut dans cette exemple ajouter autant de B qu'on veut et ça ne changera jamais la valeur de A. Par contre il suffit d'ajouter à A le resultat de $2 \times B$ pour changer sa valeur.

$A \times (B+C) \neq A \times B + A \times C$: Celle la fait partie de plusieurs propriétés mathématiques des nombres réels qui ne sont pas forcement justes pour les nombres flottants. La distributivité de la multiplication sur l'addition n'est plus valide. Pour illustrer ce phénomène en exemple, nous allons utiliser l'identité remarquable :

$$a^2 - b^2 = (a + b) \cdot (a - b)$$

La figure 1.2 montre le résultat de ce test pour a et b initialisés avec des valeurs aléatoires.

```
>> a = single(rand());
>> b = single(rand());
>> c = a ^ 2 - b ^ 2;
>> d = (a - b) * (a + b);
>> c == d

ans =

     0
```

FIGURE 1.2 – Résultats différents pour des formules équivalentes

$A + (B + C) \neq (A + B) + C$: Même l'associativité de l'addition (et même la multiplication) n'est plus valide pour les nombres flottants. Pour illustrer ce phénomène nous allons initialiser un vecteur d'une grande taille contenant des valeur aléatoire, puis nous allons calculer sa somme dans les deux sens, du début à la fin, et de la fin au début. Et finalement nous allons comparer les deux résultats obtenus.

```
>> V = rand(1, 1000000);
>> S1 = sum(V)

S1 =

    4.998430854242155e+005

>> S2 = sum(fliplr(V))

S2 =

    4.998430854242007e+005
```

FIGURE 1.3 – L'addition n'est pas associative

La figure 1.3 montre le test de l'associativité de l'addition sur Matlab, et il est possible d'observer à l'oeil nu que les deux somme *S1*, et *S2* sont différentes sur les dernier chiffres malgré le fait que les deux variables travaillaient sur le même vecteur d'entrée.

Il y a plein d'autres exemple ou l'arithmétique flottante montre un comportement différent par rapport au calcul avec des nombres réels. Dans la suite de ce cours, nous allons concentrer seulement sur les problèmes liés à la résolution des systèmes linéaires.