Les nombres et les ordinateurs

Robert Rolland

rolland@iml.univ-mrs.fr

C.N.R.S., Institut de Mathématiques de Luminy

F13288 Marseille cedex 9, France

Introduction

Nous nous intéressons à la façon de définir les objets que nous allons utiliser dans les calculs sur machine pour représenter les nombres. Ceci passe par au moins deux stades :

- La **définition de la nature mathématique** de ces objets
- Leur **représentation interne**

Nous regarderons les cas des **réels** et des **entiers**.

Les nombres réels

Les nombres réels sont approchés par les **nombres à virgule flottante** (en simple précision, double précision, précisions étendues).

Les normes **IEEE-754** et **IEEE-854** (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers) définissent ces nombres ainsi que certaines façons d'opérer dessus ($+, -, \div, \times, \sqrt{}$).

Première approche

On fixe une **base** de numération b, un nombre fixe p de digits, un signe $S = (-1)^s$ (avec $s = 0$ ou $s = 1$) un exposant e compris entre deux valeurs fixées $e_{min} \leq e \leq e_{max}$. On considère alors les nombres de la forme :

$$x = (-1)^s x_0.x_1 x_2 \cdots x_{p-1} b^e,$$

où $0 \leq x_i < b$.

la partie S est le **signe**, $x_0 x_1 \cdots x_{p-1}$ la **mantisse** et e l'**exposant**.

Les flottants normalisés

Un **flottant normalisé** est un réel non nul qui peut s'écrire sous la forme précédente avec $x_0 \neq 0$. Les nombres réels qui seront exactement représentables en machine seront 0 et les flottants normalisés. Un réel pourra ne pas être exactement représenté pour les raisons suivantes : mantisse trop longue (infinie pour les nombres non b-adiques), exposant trop grand (**overflow**) ou trop petit (**underflow**).

Cas concrets

Les bases utilisées sont 2 et 10. Chacune a ses avantages.
La base 2 sera choisie quand surtout des pas de calcul et peu d'affichages (langages de programmation généralistes par exemple).
La base 10 sera choisie si on a des affichages à chaque pas (calculette par exemple) et donc beaucoup de traductions en décimal à faire.
La norme **IEEE-754 régit le cas de la base 2.**

La base 2, simple précision

- **La définition :** $b = 2$, $p = 24$, $e_{max} = 127$, $e_{min} = -126 = -e_{max} + 1$. On introduit aussi l'exposant **biaisé** $E = e + e_{max}$.
- **Le stockage en machine :** 32 bits,

$$sE_7 \cdots E_0 x_1 \cdots x_{23}.$$

Remarquons que x_0 qui vaut 1 n'est pas écrit.
exemple : $(-1.00110001000111111110011)_2 \times 2^{(11)_2}$ se représente par les 32 bits suivants :

$$1 \mid 10000010 \mid 00110001000111111110011.$$

Bits non utilisés

Remarquons que $1 \leq E \leq 2e_{max} = 254$. Il reste donc la valeur $E = 0$ et $E = 255$ qui sont non encore utilisées.
On utilise $E = 0$ avec une mantisse nulle pour 0.
On utilise $E = 255$ avec une mantisse nulle pour $\pm\infty$ (overflow).
On utilise $E = 255$ avec une mantisse non nulle pour "Not a Number".

On utilise $E = 0$ avec une mantisse non nulle pour l'underflow.

La base 2, double précision

$b = 2$, $p = 53$, $e_{max} = 1023$, $e_{min} = -1022 = -e_{max} + 1$, $E = e + e_{max}$. Le stockage se fait sur 64 bits suivant le même principe que pour la simple précision, avec 1 bits de signe, puis 11 bits d'exposant, puis 52 bits de mantisse (x_0 n'est pas écrit).

Il existe aussi une notion de **double précision étendue** avec un stockage sur ≥ 80 bits.

La base 10

Les calculatrices utilisent en général la base 10 de manière à avoir un affichage facile. Les digits sont alors stockés en **D**écimal **C**odé **B**inaire. Chaque digit décimal est codé sur 4 bits par sa valeur en binaire. Ainsi 7 par exemple est codé 0111. On a de ce fait une petite perte (car sur 4 bits on pourrait stocker 16 symboles).

Taille du stockage

L'exposant e vérifie en général $-99 \leq e \leq 99$ et est stocké en DCB sur 1 octet. Les signes du nombre et de l'exposant sont stockés sur $1/2$ octet.
Si on dispose de n octets la mantisse aura une longueur $p = 2n - 3$ (par exemple si $n = 8$ on aurra 13 chiffres pour la mantisse). Attention il ne faut pas confondre p avec le nombre de chiffres affichés qui est souvent plus petit.

Exercice : Monter un calcul qui permette de déduire la véritable taille de la mantisse.

Arrondi

La norme IEEE définit quatre façons d'arrondir un résultat :

- Arrondi vers $+\infty$: x est arrondi au plus petit nombre représentable $\geq x$.
- Arrondi vers $-\infty$: x est arrondi au plus grand nombre représentable $\leq x$.
- Arrondi vers 0 : la valeur absolue de x est arrondie vers $-\infty$ (le signe est conservé).
- Arrondi au plus près : x est arrondi au nombre représentable le plus proche.

Arrondi correct

On choisit un mode d'arrondi. Soit f une fonction d'une ou de plusieurs variables réelles à valeurs réelles. On veut réaliser le calcul de f de telle manière que si les valeurs de $u = (u_1, \cdots, u_k)$ sont des nombres représentables alors la valeur approchée obtenue pour $f(u)$ soit l'arrondi de la vraie valeur de $f(u)$. (**Table Maker's dilemma**)

La norme IEEE impose l'arrondi correct pour les fonctions de base $+, -, \div, \times, \sqrt{}$ et les conversions. Ceci n'est pas imposé pour $\sin, \exp \cdots$.

Quelques difficultés

Supposons u, v, w des flottants. L'addition n'est pas associative : notons $F(x)$ le nombre flottant qui représente x.

$$F\left((u+v)+w\right) = F\left(F(u+v)+w\right),$$

$$F\left(u+(v+w)\right) = F\left(u+F(v+w)\right).$$

Or il se peut que $F(u+v) = u$ et $F(u+w) = u$ tandis que $F\left(u+F(v+w)\right) \neq u$. Par exemple si $b = 10$, $p = 11$ on prend $u = 1$, $v = w = 5 \times 10^{-11}$.

Les entiers

On veut représenter des nombres entiers en machine. Supposons par exemple que nous ayons **un octet** pour le faire. Avec un octet on dispose de $2^8 = 256$ écritures différentes, ces écritures pouvant être considérées comme les développements binaires des entiers de l'intervalle $\{0..255\}$.

Les entiers

Comme on veut répartir équitablement les entiers que l'on représente entre des entiers positifs et des entiers négatifs, on décide de s'intéresser aux 256 entiers de l'intervalle $\{-128..127\}$. Il convient donc d'établir une bijection qui permette des calculs commodes, entre l'intervalle $\{-128..127\}$ des entiers qu'on veut représenter, et l'intervalle $\{0..255\}$ des représentations.

Les entiers

En résumé, à tout entier x de l'intervalle $\{-128..127\}$ on va faire correspondre sa représentation $R(x)$ qui sera un entier de l'intervalle $\{0..255\}$. En outre comme $R(x)$ doit être stocké dans la mémoire d'une machine, on regardera plus spécialement les propriétés du développement binaire (sur un octet) de $R(x)$.

Représentation en "complément à 2"

Rappelons que si n est un entier, n mod 256 est le reste de la division de n par 256 ou encore l'unique entier m tel que $0 \leq m \leq 255$ et m congru à n modulo 256.

Notons I l'intervalle $\{-128..127\}$ et J l'intervalle $\{0..255\}$. Soit R l'application de I dans J définie par

$$R(x) = x \text{ mod } 256.$$

Premières propriétés

a) Montrer que R est une application bijective.

b) Calculer $R(0), R(100), R(127), R(-1), R(-100), R(-128)$. Donner les développements binaires des résultats obtenus.

c) Calculer $R(x)$ en fonction de x.

d) Déterminer l'image par R de l'ensemble des $x \geq 0$ de I ainsi que l'image par R de l'ensemble des $x < 0$ de I.

L'opposé

Comment reconnaître sur le développement binaire de $R(x)$ le signe de x?

e) On suppose que $x \in I \setminus \{-128, 0\}$. Calculer $R(-x)$ en fonction de $R(x)$.

On constatera que $R(-x) = (255 - R(x)) + 1$. En déduire un algorithme simple permettant de calculer le développement binaire de $R(-x)$ à partir de celui de $R(x)$ (algorithme dit de complémentation à 2).

Algorithme de représentation

f) Pour écrire en binaire la représentation $R(x)$ d'un entier x de l'intervalle I on applique la stratégie suivante:

- Si $x \geq 0$, on développe x en binaire.
- Si $x < 0$, on développe $-x$ en binaire et on applique l'algorithme de complémentation à 2 (cf. e)).

Appliquer cette méthode pour calculer l'écriture binaire de $R(18)$, $R(-20)$.

Addition des entiers et représentation

Soit T l'application de $\mathbb{N}$ dans $\{0..255\}$ qui à tout $n = \sum_{j=0}^{\infty} a_j 2^j$ fait correspondre $T(n) = \sum_{j=0}^{7} a_j 2^j$ (troncature limitée aux 8 premiers bits).

a) Quel est le lien entre n mod 256 et $T(n)$?

b) Montrer que si $x_1, x_2, x_1 + x_2$ sont des éléments de I alors

$$R(x_1 + x_2) = (R(x_1) + R(x_2)) \text{ mod } 256,$$

$$R(x_1 + x_2) = T(R(x_1) + R(x_2)).$$

Détection de débordement

c) Il est bien entendu que l'addition de deux éléments de I ne sera valide que si le résultat est aussi dans I. Comme la machine ne connaît que les représentants des nombres qu'on additionne, nous mettons en place ici une méthode (bien adaptée aux circuits électroniques) qui opère sur les représentations et permette à la fois de détecter si l'opération d'addition est valide et de calculer dans ce cas le résultat.

Algorithme de detection de débordeme

Soient x_1 et x_2 deux éléments de I. Soit C le carry, retenue du $8ieme$ bit vers l'exterieur, et α la retenue du $7ieme$ bit vers le $8ieme$, obtenus en faisant
l'addition $R(x_1) + R(x_2)$. On pose $V = C \oplus \alpha$.

Débordement (suite)

- Si $V = 0$ l'addition est valide et $R(x_1 + x_2)$ s'obtient en faisant en binaire l'addition de $R(x_1)$ avec $R(x_2)$ et en négligeant tout débordement au delà du huitième bit (cf. b)).
- Si $V = 1$ l'addition n'est pas valide.

Preuve

c1) Supposons $0 \leq x_1 \leq 127$ et $0 \leq x_2 \leq 127$. Examiner les deux cas $x_1 + x_2 \leq 127$ (opération valide) et $x_1 + x_2 > 127$ (opération invalide), et dans chaque cas calculer C, α, V.

c2) Supposons $0 \leq x_1 \leq 127$ et $-128 \leq x_2 < 0$ (opération toujours valide). On examinera suivant les valeurs possibles de $R(x_1) + R(x_2)$ quelles sont les valeurs possibles de C, α, V.

Preuve (suite)

c3) Supposons $-128 \leq x_1 < 0$ et $-128 \leq x_2 < 0$. Examiner les deux cas $x_1 + x_2 < -128$ (opération invalide) et $x_1 + x_2 \geq -128$ (opération valide), et dans chaque cas calculer C, α, V. (Indication: pour calculer α on pourra regarder si l'addition des deux nombres de 7 bits $(R(x_1) - 128)$ et $(R(x_2) - 128)$ a une retenue vers le huitième bit.)

c4) En conclure la validité de l'algorithme annoncé.