# Opérateurs arithmétiques matériels
# Systèmes logarithmiques

Florent de Dinechin

7 décembre 2005

# Représentation flottante et représentation logarithmique

# Virgule flottante

$$X = (-1)^s \cdot m \cdot \beta^e$$

- $s$ est un bit de signe
- $m$ est une mantisse en virgule fixe, composée d'un certain nombre de chiffres (dans une certaine base)
- $e$ est un exposant, pas forcément de la même base
  - Pour l'addition, faudra tout de même qu'un décalage de la mantisse puisse se traduire par une addition sur l'exposant
  - donc la base d'exposant doit être une puissance de la base de représentation.

# Virgule flottante utile

- format non redondant $\Longleftarrow$ premier chiffre de $m$ non nul.
- tout nombre a un opposé $\Longleftarrow$ $(m, s)$ ou complément à 2
- La notation signe-valeur absolue est actuellement préférée, c'est pour des raisons d'implémentation.
- Il y a deux zéros mais c'est utile (il y a deux infinis)

Hossam Fahmy and Michael Flynn, *The Case For a Redundant Format in Floating Point Arithmetic*, Arith 16. Ils suggèrent que les nombres soient codés avec une mantisse redondante à l'intérieur des FPU.

# Choix de la base 10

Arguments en faveur de la base 10 :

- c'est celle utilisé en en entrées/sorties
- peu de langages spécifient les conversions précisément, et elles sont coûteuses
  *La norme IEEE-754 les spécifie précisément*
- les gens écrivent "un taux de 10%" et gagnent/perdent plein d'argent sur l'erreur d'arrondi du taux
- Michael F. Cowlishaw, Eric M. Schwarz, Ronald M. Smith, Charles F. Webb, *A Decimal Floating-Point Specification*, Arith 2001
  - base 10 pour la mantisse, exposant binaire
  - codage BCD en interne
  - codage de 3 chiffres BCD par 10 bits en externe
  - le codage de l'exposant n'est pas important, mais 3 ou 4 chiffres décimaux

# Choix de la base 10

Tous ces gens travaillent chez IBM, donc en principe pas des charlots. Je ne résiste pas à vous montrer leur tableau de motivation :

| Language | Platform | Support | Precision | Comments |
|---|---|---|---|---|
| C & C++ | S/390 | Fixed | 31 | C only |
| | AS/400 | Fixed | 31 | C only |
| | Others | Fixed | 31-38 | In various libraries |
| COBOL | All | Fixed | 31 | 32 digit Floating proposed |
| C# | All | Float | 28 | |
| Java | All | Floating | infinite | Using IBM BigDecimal class; java.math is fixed point |
| OS/400 CL | AS/4000 | Fixed | 15 | Scale <= 9 |
| PL/I | S/390 | Fixed | 15 | The FLOAT Decimal data type is actually binary |
| | AS/400 | Fixed | 15 | |
| | Others | Fixed | 31 | |
| PSM | All | Fixed | 31 | Implemented via translation to C |
| Rexx | All | Floating | infinite | |
| RPG | AS/400 | Fixed | 30 | Scale <= 9 |

**Table 1. Software and Platforms Implementing Decimal Arithmetic**

# Choix de la base (2 et ses puissances)

Arguments en faveur de la base 2 :

- la plus compacte :
  - toutes les chaines de bits ont une signification
  - premier chiffre non nul $\longrightarrow$ 1 implicite
- transcription triviale en d'autres bases (4, 8, 16) pour les implémentations
- quelques propriétés utiles que nous allons voir tout de suite

Arguments en faveur de bases 4,8,16...

- tout aussi compactes
- moins de bits d'exposants pour la même dynamique

Une citation du Ercegovac et Lang :
*Many studies have been made to quantify these tradeoffs, and the present conclusion is that best is* $\beta = 2$.
Remarque : il existe une norme IEEE-854 qui étend la norme IEEE-754 pour d'autres bases. Je ne l'ai pas lue...

# Codage de l'exposant : biais

- on veut des exposants positifs et négatifs
- on pourrait coder l'exposant en complément à 2
- mais on préfère une représentation *biaisée* :

| S | E ($w_E$ bits) | F ($w_F$ bits) |
|---|---|---|

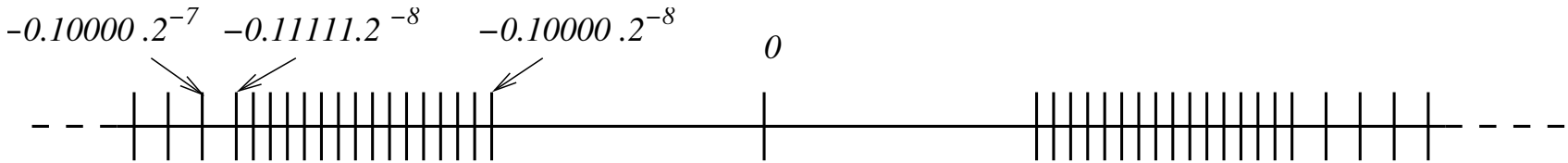
code $X = (-1)^S \cdot 1,F \cdot 2^{E-E_0}$

avec typiquement $E_0 = 2^{w_E-1} - 1$.

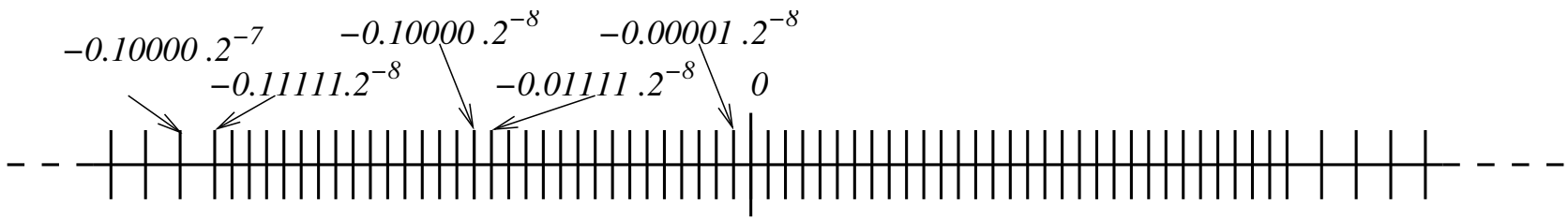
- Avantages
  - les nombres flottants positifs sont ordonnés comme les entiers
  - y compris pour la relation de succession par ajout de 1 au LSB
  - bref, comparaisons rapides
  - et par ailleurs $+0.0$ est codé par le mot composé uniquement de zéros.
- Implementation :
  - pour $\times$ et $/$ il faudra ajouter/retrancher le biais, mais cela se fait en // du calcul des mantisses
  - pour $+$ et $-$ le calcul des exposants est dans le chemin critique mais c'est une soustraction

# Nombres dénormalisés

- Inconvénient autour de zéro :




- Solution : pour l'exposant minimal (codé par zéro), on a un zéro implicite au lieu du 1 implicite.




- Vérifiez que la propriété sur l'ordre marche toujours...
- Dur à implémenter en matériel mais on y arrive

# On en est là : la norme IEEE-754

- elle réserve aussi l'exposant maximal pour coder les infinis et les *Not a number*
- ya des avantages et des inconvénients au $+/-0$
- c'est à peu près tout pour la représentation
- reste à parler d'implémentation des opérateurs
- La norme définit pour cela 4 modes d'arrondi déterministes
  - vers $+\infty$
  - vers $-\infty$
  - vers 0
  - *round to nearest, tie to even* (pourquoi ?)
- et les impose pour les 4 opérations et la racine carrée
  - parce que cela ne coûte pas cher, sauf à Cray (Kahan)

# Faut-il compter le 1 implicite dans la taille de la mantisse ?

- Oui si l'on parle d'opérateurs : il va être explicité et va passer dans du matériel
- Non si l'on parle de stockage en mémoire

# Représentation : systèmes logarithmiques

- Un nombre réel $x$ est représenté en LNS par
  - un bit qui donne le signe de $x$,
  - un bit ou un code spécial qui indique que $x = 0$ (pas normalisé encore !),
  - et le logarithme de sa valeur absolue en base $\beta$ :

$$l_x = \log_\beta(|x|)$$

- La gestion des signes et des zéros est triviale et on n'en parle plus.
- Le logarithme discret d'un réel positif $x$ sera noté $L_x$. Il s'agit d'un nombre fractionnaire signé codé en binaire (complément à 2) sur $w_E + w_F$ bits :
  - $w_E$ bits pour la partie entière, incluant le signe ;
  - $w_F$ bits pour la partie fractionnaire.
- Attention, il y a deux couches de représentation (comme en flottant)

# Choix de la base du logarithme

- Tout le monde prend 2 pour s'aligner sur la dynamique des flottants :
  - L'exposant d'un nombre flottant est une représentation logarithmique
  - mais pas sa mantisse
  - donc une représentation LNS avec $w_E$ bits de partie entière a la même dynamique qu'une représentation flottante avec $w_E$ bits d'exposant (à un log(2) près).
- il y a un papier de Vuillemin montrant que $e$ est la meilleure base possible (mais donc que 2 est pas mal).
  - mais je ne me souviens plus des arguments
- avantages architecturaux respectifs de 2 et $e$ :
  - calcul de $2^n$ par shift, utile car le logarithme est lui-même codé en base 2.
  - calcul de $e^x$ par $1 + x$ si $x$ petit, utile pour la conversion LNS $\longrightarrow$ binaire.

# Précision et dynamique : comparaison avec la virgule flottante

- Format flottant (je ne dis pas si on a un 1 implicite) :

| S | E ($w_E$ bits) | M ($w_F$ bits) |
|---|---|---|

- Format LNS

| S | I ($w_E$ bits) | F ($w_F$ bits) |
|---|---|---|

- Sans 1 implicite à la représentation flottante, le LNS a une plage un peu plus grande que le flottant
- Avec le 1 implicite, la plage est un peu plus petite
- cf “faut-il compter le 1 implicite”
- Remplacer plage par “erreur relative maximale à $w_F$ fixé”
- Le LNS est naturellement dénormalisé
- Le LNS a une erreur relative de représentation constante

Bref les deux systèmes sont bien comparables.

C’est là qu’il faut que je sorte le transparent 37 de Jérémie

# Opérateurs pour le flottant

# Calcul de l'arrondi

Soit à arrondir un nombre $x$ dont on suppose qu'on a une mantisse sur $w_F + N$ bits.

- Arrondi vers 0
  - troncature
- Arrondi vers $\infty$
  - troncature à $w_F$ bits
  - si tous les bits après $w_f$ sont nuls, c'est tout
  - sinon ajout de 1 en position $w_F$
- Arrondi au plus près pair-si-pile-au-milieu (nearest, tie to even)
  - Si $x$ est pile entre deux flottants
    - i.e., si tous les bits à droite du $(w_F + 1)$-ième sont nuls
    - "bit collant" *(sticky bit)*
    - Dans ce cas, regarder le bit en position $w_F$ : si 1, ajouter 1 en position $(w_F + 1)$
  - Sinon, ajout de 1 en position $w_F + 1$, puis troncature à $w_F$ bits
  - Rem : On n'ajoute qu'une fois le 1.

# Une anti-loi de Murphy

L'ajout de 1 peut faire changer d'exposant

- dans ce cas on a une propagation de retenue sur les exposants
- mais dans ce cas on peut économiser celle sur les mantisses !
- C'est une LCE bénéfique. On va en voir une autre.

# Addition / soustraction

On considère que $E_Y \leq E_X$

$$X + Y = (-1)^{S_X} \times (1, F_X \ \pm \ 2^{E_Y - E_X} \times (1, F_Y)) \times 2^{E_X - E_0}$$

- Algo :
  - décalage de la mantisse de $Y$
  - addition / soustraction des mantisses alignées
  - normalisation et arrondi du résultat (décalage evt)
- La seule difficulté c'est la dernière étape. Deux cas :
  - vraie addition
    - risque de dépassement de capacité
    - dans ce cas, décalage à gauche de 1
  - vraie soustraction
    - pas de risque de dépassement de capacité
    - mais risque de cancellation : décalage à droite arbitraire
  - Les deux décalages arbitraires (de la mantisse de y, et du résultat d'une vraie soustraction) sont exclusifs !
    - Preuve : cancellation que si $E_X - E_Y \leq 1$
    - Conséquence : architecture à deux chemins

# Architecture à deux chemins

Dessin ©J.Detrey (p.45)

# Normalisation et arrondi pour l'addition

... Montrons que trois bits supplémentaires suffisent toujours (dont un *sticky*)

- Si cancellation, alors l'opération était exacte (pas d'arrondi)

# Multiplication

$$X \times Y = (-1)^{S_X+S_Y} \times (1, F_X \times 1, F_Y) \times 2^{E_X+E_Y-2E_0}$$

- produit des mantisses
- somme des exposants
- Seules difficultés : normalisation et arrondi

# Normalisation et arrondi pour la multiplication

... Montrons que deux bits supplémentaires suffisent toujours (dont un *sticky*)
... On intégrera les bits ajoutés à l'addition finale de notre multiplieur.

# Division

$$X \ / \ Y = (-1)^{S_X - S_Y} \times (1, F_X \ / \ 1, F_Y) \times 2^{E_X - E_Y}$$

- quotient des mantisses
- différence des exposants

# Normalisation et arrondi pour la division

... Montrons que deux bits supplémentaires suffisent toujours (dont un *sticky*)
... Le *sticky* ici c'est "le reste est non nul"
... Ne montrons pas que le cas *even* du *round to nearest even* n'arrive jamais.
... le reste, c'est du bricolage de bits pour faire la correction du dernier $R_j$ (vous vous souvenez ?), la normalisation et l'arrondi en une seule étape.

# Racine carrée

Racine carrée :

$$\sqrt{X} = \sqrt{1, F_X} \times 2^{(E_X - E_0)/2} \qquad \text{si } S_X = 0$$

- extraction de racine carrée de la mantisse
- division par 2 de l'exposant
- Seules difficultés : normalisation et arrondi

Mais tout le monde en a déjà marre. Voir le Ercegovac et Lang.

# Opérateurs pour le LNS

Représentation flottante et représentation logarithmique

Opérateurs pour le flottant

Opérateurs pour le LNS

Quel système pour quelle application ?

Le système semi-logarithmique

Annexe : (rares) applications connues du LNS

# Opérations faciles

Rappels : Les signes de $x$ et $y$ sont gérés à part, dans la suite $x$ et $y$ sont positifs, et leurs logarithmes $l_x$ et $l_y$ sont signés et rationnels.

- $l_{x\times y} = l_x + l_y$ : opération exacte, sauf overflow
- $l_{x/y} = l_x - l_y$ : opération exacte, sauf overflow
- $l_{\sqrt{x}} = l_x/2$ :
  - opération exacte une fois sur deux,
  - arrondie toujours dans le même sens sinon.

# LCDE (Opérations difficiles)

- Addition :

$$l_{x+y} = \log_\beta(x+y) = \log_\beta(\beta^{l_x}+\beta^{l_y}) = l_x+\log_\beta(1+\beta^{l_y-l_x})$$

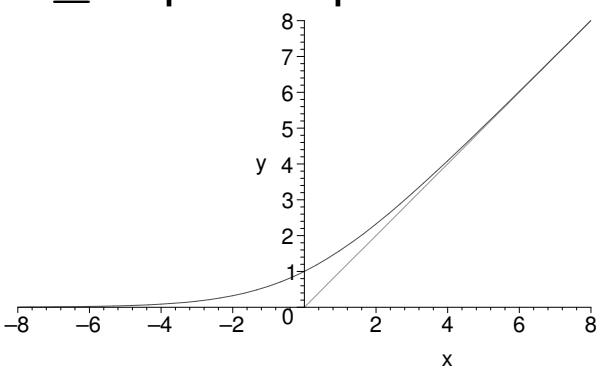
- Soustraction :

$$l_{x-y} = \log_\beta(\beta^{l_x}-\beta^{l_y}) = l_x+\log_\beta(1-\beta^{l_y-l_x})$$

- Dans toute la suite on prend $l_x \geq l_y$ et on pose

$$\begin{aligned} r &= l_y - l_x \\ f_\oplus(r) &= \log_\beta(1+\beta^r) \\ f_\ominus(r) &= \log_\beta(1-\beta^r) \end{aligned}$$

$r \leq 0$ parce que cela nous arrange pour la fonction $f_\oplus$ :



# Précision des opérations LNS

En LNS,

- $\times, /$ sont exactes aux dépassements de capacité près
- $+$ et $-$ ont le problème du fabricant de table :
  - calcul de $r = l_y - l_x$ facile, addition finale facile
  - mais fonctions $f_\oplus(r) = \log_\beta(1 + \beta^r)$ et $f_\ominus(r) = \log_\beta(1 - \beta^r)$ difficiles à arrondir au plus près
- donc pas d'espoir d'arrondi correct sur les logs.
- Heureusement, notion de "better than floating point" :
  - en virgule flottante $a +_{\mathrm{FP}} b = a + b(1 + \epsilon)$, où l'erreur relative $\epsilon$ varie du simple au double suivant la valeur de la mantisse.
  - on va se permettre $a +_{\mathrm{LNS}} b = a + b(1 + \epsilon)$ en prenant comme $\epsilon$ la pire erreur flottante, qui est de $1/\log(2) \approx 1.44$ demi-bit de $f_\oplus$
  - ce qui est plus strict que l'arrondi fidèle de $f_\oplus$, mais faisable

# Précision des opérations LNS

Rappel : en virgule flottante tout allait bien

- dans le cas général, arrondi correct pour $+, -, *, /, \sqrt{}$
- quelques théorèmes qui donnent des cas dans lesquels l'opération est exacte (lemme de Sterbenz)
- peu d'espoir de théorèmes correspondant en LNS

Cela dit, ce qui compte c'est l'erreur moyenne pour un calcul complet. Rappel : on a une multiplication et une division exactes, et on utilise LNS lorsqu'il y a beaucoup de ces opérations.

# Kahan est au 3ème étage alors je vais chuchoter

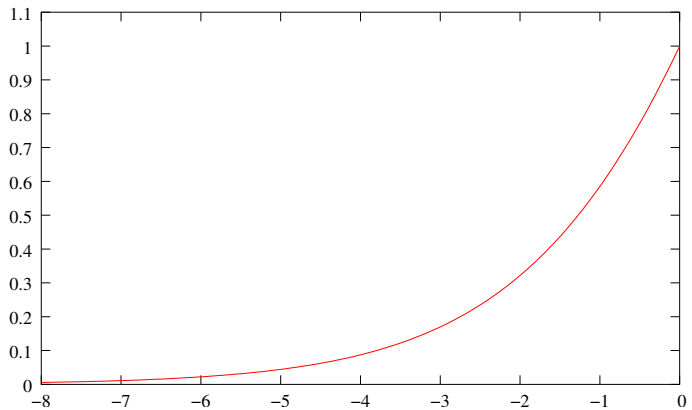
- D'aucuns disent d'ailleurs que vu qu'on a une multiplication tellement précise, on peut se permettre de faire une addition cochonne (arrondi fidèle classique), et qu'en moyenne sur une application on s'y retrouvera :

  Mark G. Arnold, Colin Walter, *Unrestricted Faithful Rounding is Good Enough for Some LNS Applications* :

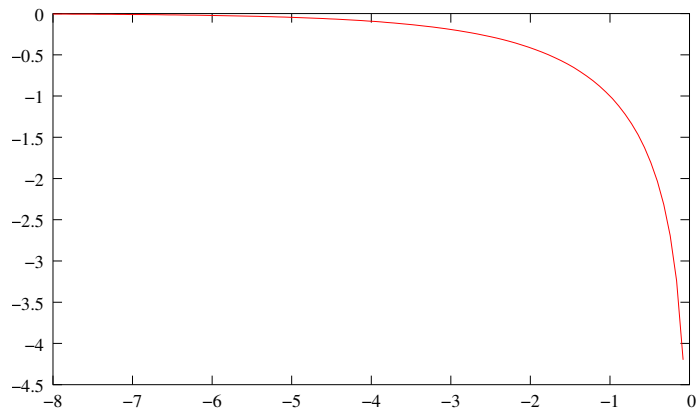
  We propose relaxing the restricted form of faithful rounding used in prior 32-bit Logarithmic Number System (LNS) implementations. Unrestricted faithful rounding yields a three- to six-fold savings in VLSI ROM size (or four- to six- fold savings in F GPA table size) with only modest increase in error. This can be acceptable for the DSP and multimedia applications in which the non-standard LNS is a candidate for adoption. Such applications are cost sensitive, and the tremendous cost reduction of the LNS model proposed here should argue very favourably for its adoption.

- Par ailleurs il y a des tas de papiers montrant que pour telle ou telle application, du LNS sur 12 bits donne un aussi bon résultat que du FP sur 16 bits...

# Les fonctions $f_{\oplus}$ et $f_{\ominus}$




La fonction $f_{\oplus}$



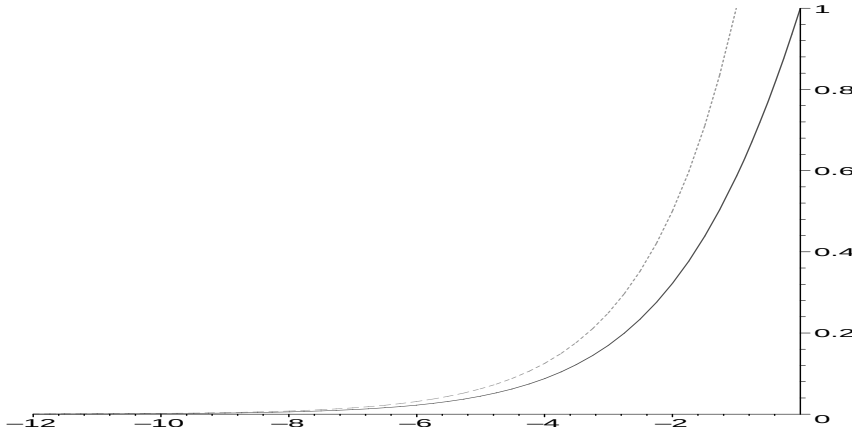

La fonction $f_{\ominus}$

- $f_{\oplus}$ a une meilleure tête que $f_{\ominus}$
- Quels sont leurs domaines images ?

La suite pourrait s'appeler "un exemple concret d'application du cours précédent sur les fonctions élémentaires"

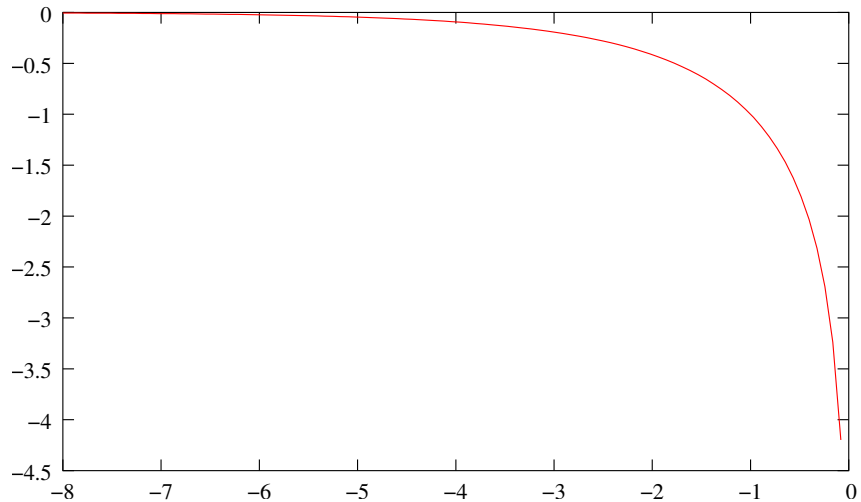
# Pour $f_\oplus$ tout va bien

- Notion de zéro essentiel :
  - lorsque $\beta = 2$, pour tout $r < 0$ on a $f_\oplus(r) < 2^{r+1}$

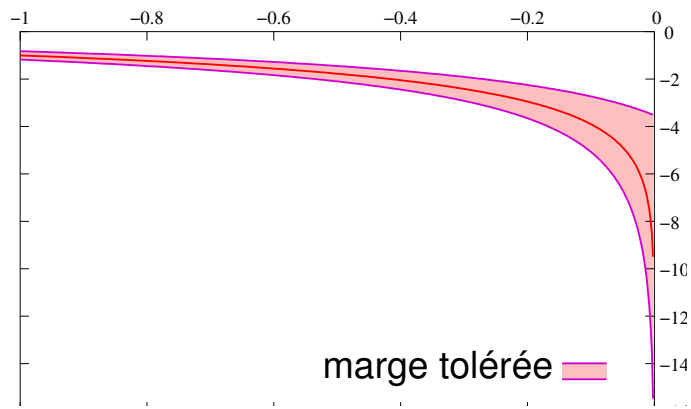



  - autrement dit $f_\oplus(r) < 2^{-w_F-1}$ pour tout $r < -w_F - 2$,
  - autrement dit $f_\oplus(r)$ s'arrondit en zéro pour tout $r < -w_F - 2$.
  - Il y a une propriété similaire pour tout $\beta$.
  - Le nombre de bits à considérer pour $r$ ne dépend pas de $w_I$ !
- Par ailleurs cette fonction a une dérivée bien bornée, donc s'approximera bien par des méthodes d'ordre 1 ou 2
- On coupera tout de même son intervalle en 2 ou 4
  - pour économiser sur les bits de sortie
  - parce que la dérivée tend aussi franchement vers 0

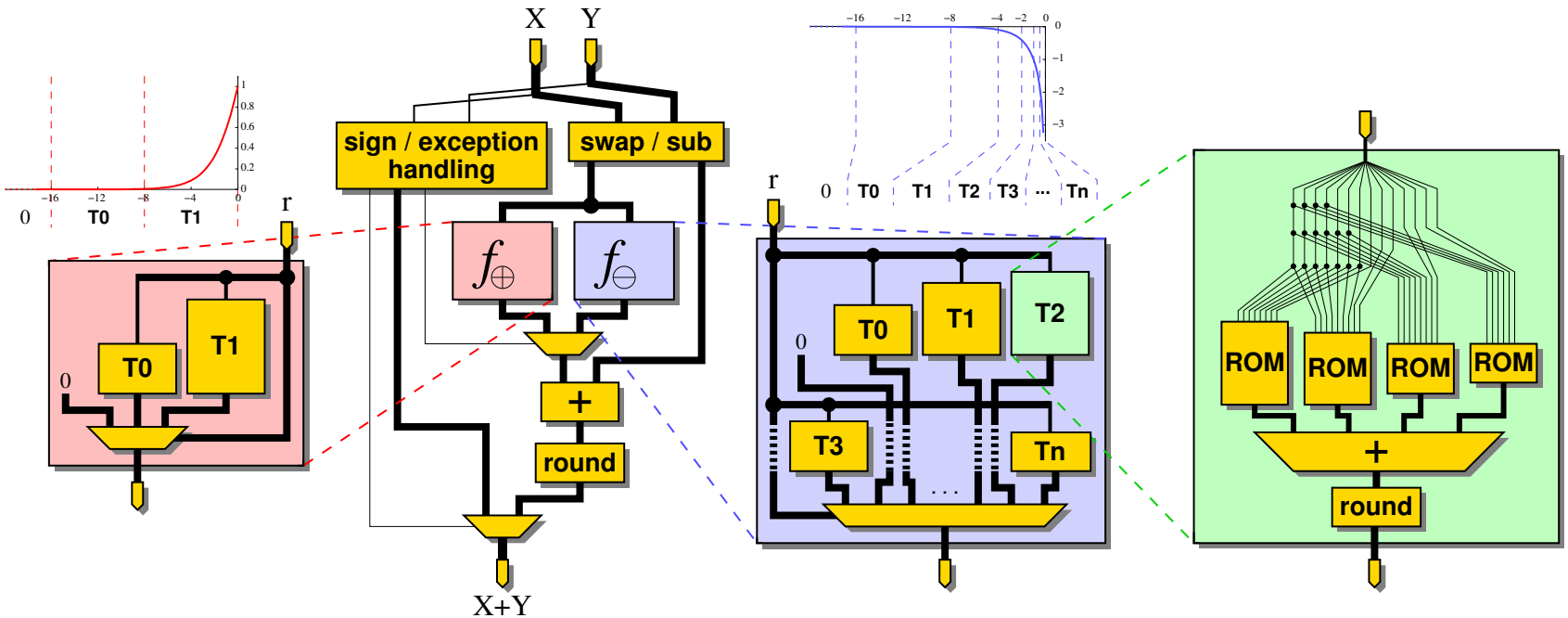
# Pour $f_\ominus$ tout va mal




- Zéro essentiel pareil
- Vilaine dérivée pas bornée
- Découpage de l'intervalle pareil, mais les morceaux seront plus petits
- On renégocie la précision :



# Un exemple de découpage




©J.Detrey

# Un florilège de méthodes d'approximations

Je sais que vous en aviez déjà marre la semaine dernière, mais...
Sortons les transparents d'Arnold (résumé d'une vingtaine de papiers en trente ans)
Il manque un papier de Lee de 2003 qui fait une réduction d'argument encore plus intelligente

# Quel système pour quelle application ?

Encore J.Detrey ? Vous me dites quand vous en avez marre que je vous parle de lui.

# Le système semi-logarithmique

# Présentation

- Famille de systèmes de représentation des réels
- Un paramètre, $k$, dont les valeurs extrêmes définissent FP et LNS
- Du point de vue archi, permet des compromis intermédiaires entre FP et LNS
- Jamais implémenté (même Jérémie voudrait pas)

Muller, Scherbyna, Tisserand, *Semi-Logarithmic Number Systems*, IEEE Transactions on Computers, 47-2, 1998

# Allons-y

Let $k$ be an integer, let $x$ be a real number different from 0, and define $e_{k,x}$ as the multiple of $2^{-k}$ satisfying

$$2^{e_{k,x}} \leq |x| < 2^{e_{k,x}+2^{-k}}. \tag{1}$$

We immediately find

$$e_{k,x} = \frac{\lfloor 2^k \log_2 |x| \rfloor}{2^k} \tag{2}$$

Define $m_{k,x}$ as :

$$m_{k,x} = \frac{|x|}{2^{e_{k,x}}}$$

If $s_x$ is the sign of $x$, we obviously have

$$x = s_x \times m_{k,x} \times 2^{e_{k,x}}.$$

where $e_{k,x}$ is a $k$-bit approximation of $\log_2 |x|$ and $m_{k,x}$ a multiplicative correction factor.

From (1) we deduce :

$$1 \leq m_{k,x} = \frac{|x|}{2^{e_{k,x}}} < 2^{\frac{1}{2^k}}$$

Now let us bound the value $2^{\frac{1}{2^k}}$. This value is equal to $e^{\frac{\ln 2}{2^k}}$. One can easily show that for $\alpha \in (0, 1)$,

$$1 + \alpha > 2^{\alpha}.$$

Using this result with $\alpha = 1/2^k$, we get

$$2^{\frac{1}{2^k}} \leq 1 + \frac{1}{2^k} \tag{3}$$

for $k \geq 0$. As a consequence, $1 \leq m_{k,x} < 1 + \frac{1}{2^k}$.
This leads to the two following definitions. The difference between both is the "normalization" of $m_{k,x}$ : the bound on $m_{k,x}$ required by the "general form" is easier to check.

# Canonical Form

Let $k$ be a positive integer. Every non zero real number $x$ will be represented in the *Canonical form* of the *Semi-Logarithmic Number System (SLNS for short) of Parameter k* by three values $s_x$, $m_{k,x}$ and $e_{k,x}$ satisfying :

- $s_x = \pm 1$
- $e_{k,x}$ is a multiple of $2^{-k}$
- $1 \leq m_{k,x} < 2^{\frac{1}{2^k}}$
- $x = s_x \times m_{k,x} \times 2^{e_{k,x}}$

# General Form

Let $k$ be a positive integer. Every non zero real number $x$ will be represented in the *General form* of the SLNS of Parameter $k$ by three values $s_x$, $m_{k,x}$ and $e_{k,x}$ satisfying :

- $s_x = \pm 1$
- $e_{k,x}$ is a multiple of $2^{-k}$
- $1 \leq m_{k,x} < 1 + 2^{-k}$
- $x = s_x \times m_{k,x} \times 2^{e_{k,x}}$

The canonical form is a kind of floating-point representation, with exponents that are multiples of $2^{-k}$, and a corresponding normalized mantissa.

The representation of $x$ with $n$ mantissa bits in the *semi-logarithmic number system of parameter $k$* will be constituted by $s_x$, $e_{k,x}$ and an $n$-fractional bit rounding of $m_{k,x}$. In practice, since $1 \leq m_{k,x} < 1 + 2^{-k}$, $m_{k,x}$ has a binary representation of the form :

$$1.\overbrace{\underbrace{0000\ldots000}_{k \text{ zeroes}}\texttt{xxxx}\ldots\texttt{xx}}^{n \text{ digits}}$$

- Since the first $k+1$ digits of $m_{k,x}$ are known in advance, there is no need to store them (comme le bit implicite de FP).
- A special representation must be chosen for zero.

# Semi log intermédiaire entre LNS et FP

- If $k = 0$, then the semi-logarithmic system of order $k$ is reduced to a $n$-mantissa digit floating-point system.
- if $k \geq n$ then the semi-logarithmic system of order $k$ is reduced to a logarithmic number system.
- the canonical form can be viewed as a *non-redundant* representation. In that form, comparisons are easily performed.
- the general form is a *redundant* representation. For instance, if $k = 1$, then $\sqrt{2}$ has two possible representations, namely $1.0000000\ldots \times 2^{0.1}$ — the exponent and mantissa are written in radix 2 — and $1.0110101000001001111\ldots \times 2^{0.0}$. Although the comparisons are slightly more difficult with the general form — this is due to the redundancy —, we will prefer that form, because the condition "$1 \leq m_{k,x} < 1 + 2^{-k}$" is easier to check than the condition "$1 \leq m_{k,x} < 2^{\frac{1}{2^k}}$," and because the general form leads to simpler arithmetic algorithms.

- Anyway, conversion from the general form to the canonical form : assume $s_x \times m_x \times 2^{e_x}$ is in general form. Compare $m_x$ with $\rho_k = 2^{2^{-k}}$. If $m_x < \rho_k$ then the number is already represented in canonical form. If $m_x \geq \rho_k$, then add $2^{-k}$ to $e_x$ and divide $m_x$ by $\rho_k$. The obtained result will be the representation of $x$ in canonical form.

So the parameter $k$ makes it possible to choose various compromises between the floating-point number system and the logarithmic number system.
Exactly as in floating-point arithmetic, there are various possible rounding modes. For instance, if we define $\mathcal{Z}(x)$ as the number obtained by rounding $m_x$ (in canonical form) to zero, then we get :

$$\mathcal{Z}(x) = s_x \times \frac{\left\lfloor 2^n \times \frac{|x|}{2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}} \right\rfloor}{2^n} \times 2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}$$

Similarly, we can define rounding towards $\pm\infty$ and rounding to the nearest.

# Le reste du papier :

Les opérations deviennent toutes horribles mais futées.

La précision de représentation (*average/mean relative representation error*) est un peu moins bonne que LNS et FP.

| | **Rounding to zero** | | **Rounding to nearest** | |
|---|---|---|---|---|
| | MRRE | ARRE | MRRE | ARRE |
| Floating Point | $2^{-n}$ | $0.36 \times 2^{-n}$ | $2^{-n-1}$ | $0.36 \times 2^{-n-1}$ |
| SLNS ($k \geq 2$) | $2^{-n}$ | $0.5 \times 2^{-n}$ | $2^{-n-1}$ | $0.5 \times 2^{-n-1}$ |
| Logarithmic | $0.69 \times 2^{-n}$ | $0.35 \times 2^{-n}$ | $0.69 \times 2^{-n-1}$ | $0.35 \times 2^{-n-1}$ |

# Sautons tout de suite à la conclusion

We have proposed a new class of number systems, called *semi-logarithmic number systems*. They constitute a compromise between the floating point and the logarithmic number systems : if the parameter $k$ is larger than $n/2+2$, multiplication and division are almost as easily performed as in the logarithmic number systems, whereas addition and subtraction require much smaller tables. The best value for $k$ must result from a compromise : if $k$ is large, the tables required for addition may become huge, and if $k$ is small, the algorithms become complicated. Values of $k$ slightly larger than $n/2$ are probably the best choice. Although the semi-logarithmic number systems is slightly less accurate than the floating-point and the logarithmic number systems, the difference is very small (roughly speaking, $1/2$ bit of accuracy). The domain of application of the semi-logarithmic number systems is the same as that of the logarithmic number systems : special purpose processors for solving problems where the ratio of multiplies to adds is relatively high.

# Annexe : (rares) applications connues du LNS

Yamaha - DX-7 Music Synthesizer

In 1983, Yamaha introduced the DX-7 music synthesizer with a 13 bit LNS to implement the FM synthesis algorithm. Approximately 200,000 units were sold before the product line was discontinued in the late 80's. The DX-7 initially sold for $2,000 which means that this product probably made hundreds of millions for Yamaha. It is therefore reasonable to assume that the LNS component was worth a few million dollars to Yamaha at the time. (However it is unlikely that Yamaha still uses the LNS, especially considering the advances in synthesizers and integrated circuits that have been made since 1983.) For more information on this application, refer to the Uchiyama reference in the LNS Articles.

## GRAPE Project - Astrophysics

In 1989, researchers at the University of Tokyo built a special purpose computer that used LNS to speed up as astrophysical calculations involving the gravitational forces between stars. Because this hardware used a pipelined architecture, they called it the "gravity pipe" or "GRAPE". Over the years, several generations of GRAPE systems have been built. Although some models use floating point, the GRAPE-1, -1A, -3, -3A, and -5 models use LNS. The GRAPE-5 model won the Gordon Bell Prize (price/performance cateogory) in 1999. The GRAPE project is still going strong. For more information refer to the Kwai et al. and Makino et al. references in the LNS Articles or visit the GRAPE project at www.astrograpе.org.

## HTK - Speech Recognition

Many pattern recognition tasks, and in particular speech recognition, make use of a technique known as Hidden Markov Models (HMM). One aspect of these models is that they involve multiplying several probabilities together to come up with an overall probability that is used to determine if a spoken word matches a word in the system's vocabulary. Because these models involve a great deal of multiplication, a logarithmic format is often used in order to speed up calculations. An example of this is provided by the popular toolkit, HTK, used by hundreds of researchers worldwide. This software was originally developed at the Cambridge University Engineering Department in 1989, then spun off to a commercial firm which was acquired by Microsoft in 1999, and then licensed back to Cambridge in 2000 for distribution to researchers. The HTK software uses a 15-bit LNS. For more information refer to the Young et al. reference or visit the HTK web site at http ://htk.eng.cam.ac.uk.

## Aircraft Cabin Air Pressure Controller

The cabin air pressure controls for the Boeing 767 and several other aircraft apparently use logarithmic arithmetic. For more information, refer to Arnold et al. (1998, p. 785, last paragraph), read Pickett's summary (U.S. Patent 5,184,317, "Description of Prior Art" section, last three paragraphs) or contact Log Point Technologies at www.logpoint.com.

## Motorola - Communications Satellites

In 1995 and 1996, Motorola filed several related patents for a system that enhances wireless communication channels by using a digital signal processing technique known as "beam forming". For maximum throughput, the system uses a custom DSP chip that is based on several logarithmic processors working in parallel. According to the patents, a primary use for this technology is communication with satellites. Although Motorola has apparently fabricated the custom parallel logarithmic DSP chip, it is unclear whether this technology ever was used in commercial products. If the technology was commercialized, it was most likely as part of the Iridium satellite system that went into service in November 1998. Motorola was a major investor in Iridium and developed its communication satellites.

Interactive Machines, Inc. - Graphics

This firm built graphics workstations during the 1980's. One model, the IMI-500, apparently used LNS to speed up graphics calculations. This model appears to have been used by various TV and film studios for animation and special effects. It is now difficult to obtain information about the IMI-500. However when these details can be confirmed, a full reference will be provided.

## PC Software

Back in the days of the Intel 286 and 386, a floating point unit was not included in the microprocessor – but was instead sold as a separate (and expensive) math coprocessor. Because many PC users did not want to buy a coprocessor, software developers used a number of tricks to speed up computationally intensive software. At that time, Triakis – a very small firm in Bothell, WA – was selling a LNS software library called "FFP" to software developers. The FFP product was apparently introduced in March '91.

Computer Aided Tomography

During the late 1970's, there was a project involving the Mayo Clinic regarding development of a CAT scanner based on LNS. It is not clear whether this technology was ever used to examine patients. For more details, refer to the references for Swartzlander et al. in the LNS Articles.

## Wang Labs - Electronic Calculator

Around 1964, Wang Labs introduced a desktop engineering calculator – the LOCI (short for LOgarithmic Computing Instrument) – that sold for $6,500. Dr. An Wang chose to use logarithms because it eliminated the need for an expensive conventional multiplier and thus enabled the entire calculator to be implemented with about 300 discrete transistors and a few PC boards. The result was an affordable product that sold well and was a major source of revenue for Wang Labs during the mid-60's. Photographs and an excellent history of the LOCI can be found at the Old Calculator Web Museum. For more information on the LOCI's patent, refer to Wang, A. in the LNS Articles.

## Nuclear Reactor Monitor

In the mid 1960's, a computer with LNS was apparently used to monitor a French research reactor. For more information, refer to the Combet et al. and Furet et al. references in the LNS Articles.

Mechanical Calculators, Etc.

Prior to the invention of the electronic computer, there were commercial products based on the logarithmic addition and subtraction functions. Books of mathematical tables were undoubtedly the most popular LNS products – especially after Leonelli's work was popularized in 1812 by the great mathematician, Gauss. However, there were also a few mechanical calculators that used these functions. Two devices from 1893 are Brauer's compass and Torres' Algebraic Machine (see gear and diagram).