Nombres, opérateurs, algorithmes et circuits

Systèmes de numération pour les quatre opérations

Florent de Dinechin

19 janvier 2007

Numération simple de position (addition et multiplication)

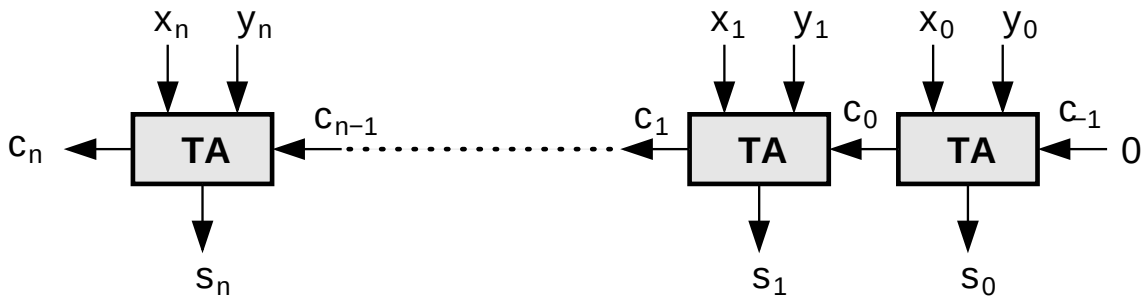
Rappels

- base $\beta \geq 2$
- chiffres compris entre 0 et $\beta - 1$.
- l'écriture $x_{n-1}...x_0$ représente l'entier $X = \sum_{i=0}^{n-1} \beta^i x_i$

Addition

- Soit la fonction *table d'addition* TA.
- algo :

  1: $c_{-1} = 0$
  2: **for** $i = 0$ to n **do**
  3: $\quad (c_i, s_i) = \mathrm{TA}(c_{i-1}, x_i, y_i)$
  4: **end for**



- Exercice : écrire $X = \sum_{i=0}^{n-1} \beta^i x_i$ etc, et prouver que cet algorithme réalise bien l'addition.
- Remarque supplémentaire : une retenue sortante signifie un dépassement de capacité.

Multiplication comme à l'école

$$X = \sum_{i=0}^{n-1} \beta^i x_i \quad \text{donc} \quad XY = \sum_{i=0}^{n-1} \beta^i x_i Y$$

Multiplication comme à l'école

$$X = \sum_{i=0}^{n-1} \beta^i x_i \quad \text{donc} \quad XY = \sum_{i=0}^{n-1} \beta^i x_i Y$$

- Les multiplications par β^i sont des décalages avec ajout de zéros (exemple : multiplication par 10000 en base 10)

Multiplication comme à l'école

$$X = \sum_{i=0}^{n-1} \beta^i x_i \quad \text{donc} \quad XY = \sum_{i=0}^{n-1} \beta^i x_i Y$$

- Les multiplications par β^i sont des décalages avec ajout de zéros (exemple : multiplication par 10000 en base 10)
- Vous reconnaissez donc l'algo du primaire

Multiplication comme à l'école

$$X = \sum_{i=0}^{n-1} \beta^i x_i \quad \text{donc} \quad XY = \sum_{i=0}^{n-1} \beta^i x_i Y$$

- Les multiplications par β^i sont des décalages avec ajout de zéros (exemple : multiplication par 10000 en base 10)
- Vous reconnaissez donc l'algo du primaire
- On va le généraliser en démontant chaque multiplication $x_i Y$:

$$Y = \sum_{j=0}^{n-1} \beta^j y_j \quad \text{donc} \quad x_i Y = \sum_{j=0}^{n-1} \beta^j x_i y_i$$

Multiplication comme à l'école

$$X = \sum_{i=0}^{n-1} \beta^i x_i \quad \text{donc} \quad XY = \sum_{i=0}^{n-1} \beta^i x_i Y$$

- Les multiplications par β^i sont des décalages avec ajout de zéros (exemple : multiplication par 10000 en base 10)
- Vous reconnaissez donc l'algo du primaire
- On va le généraliser en démontant chaque multiplication $x_i Y$:

$$Y = \sum_{j=0}^{n-1} \beta^j y_j \quad \text{donc} \quad x_i Y = \sum_{j=0}^{n-1} \beta^j x_i y_i$$

donc

$$XY = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \beta^{i+j} x_i y_j$$

Multiplication

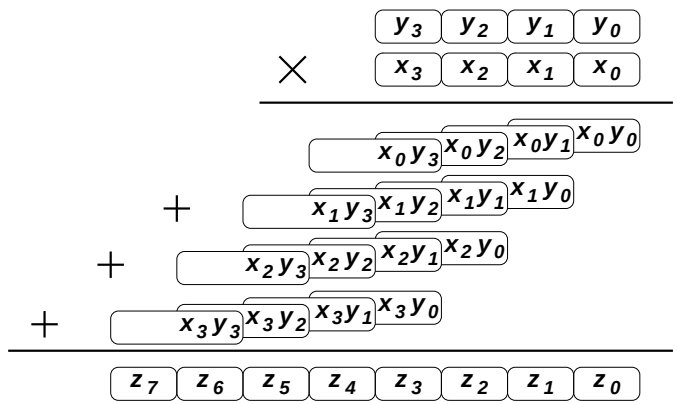
$$XY = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \beta^{i+j} x_i y_j$$

Multiplication

$$XY = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \beta^{i+j} x_i y_j$$

L'algo du primaire est donc une version *séquentielle* d'un algo plus général que voici :

1: **for all** $i, j \in \{0..n-1\}$ **do**
2: $p_{ij} = x_i \times y_j$
(calculer des produits partiels)
3: **end for**
4: Sommer tous ces p_{ij}
(alignés comme il faut)

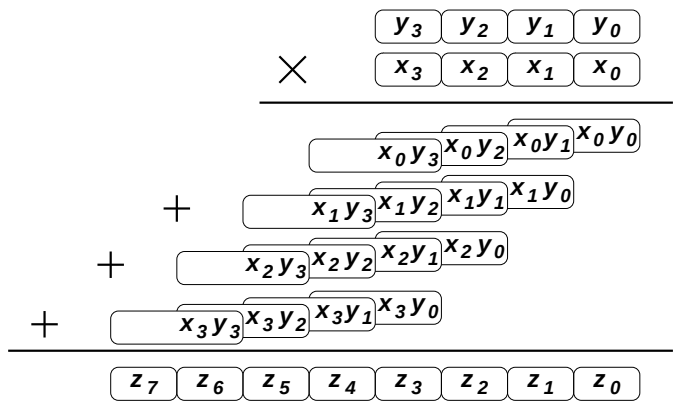


Multiplication

$$XY = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \beta^{i+j} x_i y_j$$

L'algo du primaire est donc une version *séquentielle* d'un algo plus général que voici :

1: **for all** $i, j \in \{0..n-1\}$ **do**
2: $p_{ij} = x_i \times y_j$
(calculer des produits partiels)
3: **end for**
4: Sommer tous ces p_{ij}
(alignés comme il faut)



Le gros de la recherche concernant les multiplieurs, c'est pour trouver de bonnes manières de faire cette sommation...

Un théorème compliqué

Theorem

Quelque soit β, chaque produit partiel tient sur deux chiffres.

Démonstration.

$x_i < \beta, y_j < \beta \implies x_i y_j < \beta^2$ □

Multiplication binaire

- La table de multiplication est plus facile que celle que vous avez mis tout votre CE1 à apprendre.

Multiplication binaire

- La table de multiplication est plus facile que celle que vous avez mis tout votre CE1 à apprendre.
- Différence avec le cas général : les produits partiels tiennent sur un chiffre !

Multiplication binaire

- La table de multiplication est plus facile que celle que vous avez mis tout votre CE1 à apprendre.
- Différence avec le cas général : les produits partiels tiennent sur un chiffre !

Avec des équations :

$$X = \sum_{i=0}^{n-1} 2^i x_i$$

$$XY = \sum_{i=0}^{n-1} 2^i x_i Y$$

Codage des entiers relatifs (soustraction)

Représentation signe + valeur absolue

- La valeur absolue est codée en numération simple de position
- ⊕ Conceptuellement simple
- ⊕ Calcul de l'opposé trivial
- ⊖ Deux représentations du zéro : $+0$ et -0
- ⊖ Nécessite un algorithme (du matériel) différent pour addition et soustraction

Exercice : écrivez l'algorithme de soustraction de deux nombres codés en signe / valeur absolue en numération simple de position.

Représentation en complément à la base

Exemple en base 10 avec 3 chiffres :

- ignorer la retenues sortante, dans notre algorithme d'addition, équivaut à travailler modulo 1000.

Représentation en complément à la base

Exemple en base 10 avec 3 chiffres :

- ignorer la retenues sortante, dans notre algorithme d'addition, équivaut à travailler modulo 1000.
- On peut donc décider que l'intervalle entier représenté n'est pas $\{0..999\}$ mais (par exemple) $\{-500..499\}$

Représentation en complément à la base

Exemple en base 10 avec 3 chiffres :

- ignorer la retenues sortante, dans notre algorithme d'addition, équivaut à travailler modulo 1000.
- On peut donc décider que l'intervalle entier représenté n'est pas $\{0..999\}$ mais (par exemple) $\{-500..499\}$
- Dans ce cas on code un nombres négatif $-500 \leq X < 0$ par $1000 - X$.

Représentation en complément à la base

Exemple en base 10 avec 3 chiffres :

- ignorer la retenues sortante, dans notre algorithme d'addition, équivaut à travailler modulo 1000.
- On peut donc décider que l'intervalle entier représenté n'est pas $\{0..999\}$ mais (par exemple) $\{-500..499\}$
- Dans ce cas on code un nombres négatif $-500 \leq X < 0$ par $1000 - X$.
- Le signe est alors déterminé par le chiffre de poids fort.

Représentation en complément à la base

Exemple en base 10 avec 3 chiffres :

- ignorer la retenues sortante, dans notre algorithme d'addition, équivaut à travailler modulo 1000.
- On peut donc décider que l'intervalle entier représenté n'est pas $\{0..999\}$ mais (par exemple) $\{-500..499\}$
- Dans ce cas on code un nombres négatif $-500 \leq X < 0$ par $1000 - X$.
- Le signe est alors déterminé par le chiffre de poids fort.
- Et notre algorithme d'addition fonctionne à présent sur des entiers relatifs.

Représentation en complément à la base

Exemple en base 10 avec 3 chiffres :

- ignorer la retenues sortante, dans notre algorithme d'addition, équivaut à travailler modulo 1000.
- On peut donc décider que l'intervalle entier représenté n'est pas $\{0..999\}$ mais (par exemple) $\{-500..499\}$
- Dans ce cas on code un nombres négatif $-500 \leq X < 0$ par $1000 - X$.
- Le signe est alors déterminé par le chiffre de poids fort.
- Et notre algorithme d'addition fonctionne à présent sur des entiers relatifs.

(je n'ai jamais vraiment compris pourquoi cela s'appelle le complément à la base)

En prime, la soustraction

En prime, la soustraction

- On remarque que $0 \equiv 1000 = 1 + 999$

En prime, la soustraction

- On remarque que $0 \equiv 1000 = 1 + 999$
- Par conséquent,
  $123 - 45 \equiv 123 + 1000 - 45$

En prime, la soustraction

- On remarque que $0 \equiv 1000 = 1 + 999$
- Par conséquent,
  $123 - 45 \equiv 123 + 1000 - 45 \equiv 123 + (999 - 45) + 1$

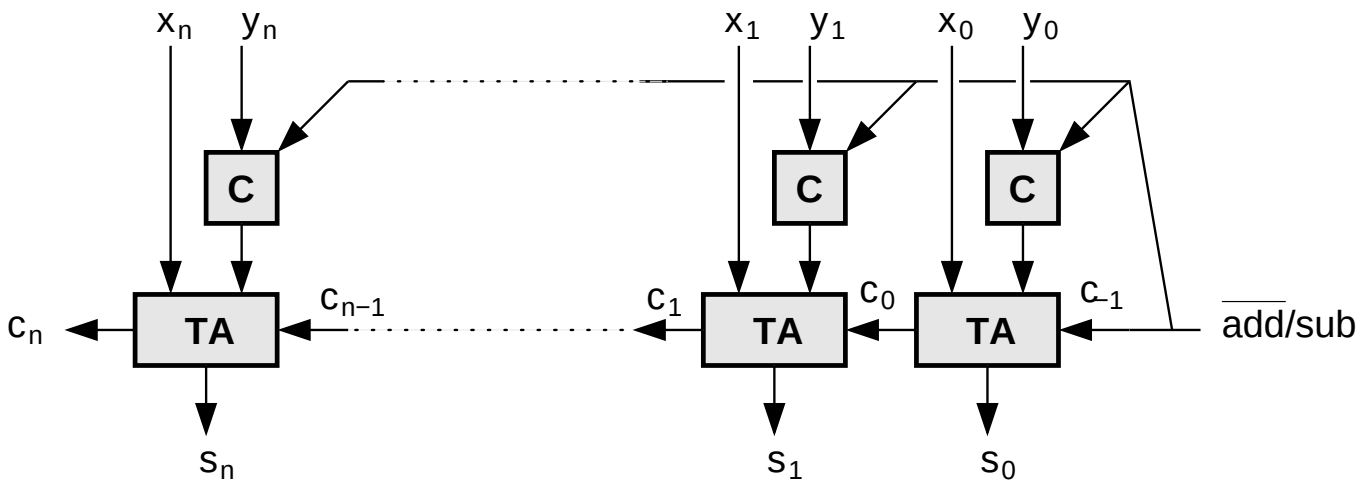
En prime, la soustraction

- On remarque que $0 \equiv 1000 = 1 + 999$
- Par conséquent,
  $123 - 45 \equiv 123 + 1000 - 45 \equiv 123 + (999 - 45) + 1$
- Or l'opération $999 - 45$ est facile (chiffre à chiffre)

En prime, la soustraction

- On remarque que $0 \equiv 1000 = 1 + 999$
- Par conséquent,
  $123 - 45 \equiv 123 + 1000 - 45 \equiv 123 + (999 - 45) + 1$
- Or l'opération $999 - 45$ est facile (chiffre à chiffre)
- et le $+ 1$ est gratuit aussi : retenue entrante

Additionneur/soustracteur en complément à la base



Remarques

- La propagation de l'information de signe est une diffusion : elle ne traverse pas de boîte.
  - en VLSI il y a une limite au nombre de boîtes que l'on peut alimenter avec un fil...
- Pour le moment on se garde de parler de performances :
  - quels sont les temps de traversée comparés de la boîte C, de la boîte TA, d'un fil ?
  - ...

Inconvénients du complément à la base

- Détection des dépassements de capacité compliquée :
  Pour une addition,
  - Si les deux nombre sont de signes différents, alors pas de dépassement possible.
  - Si les deux nombres sont de même signe, alors dépassement ssi le résultat est de signe différent.

  Pour une soustraction c'est le contraire.
- calculer l'opposé d'un nombre demande une propagation de retenue.

En binaire : complément à 2

- l'écriture $x_{n-1}...x_0$ représente l'entier

$$X = -2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i \ .$$

- le signe du nombre est donné par son bit de poids fort.
- La boîte $\boxed{\text{C}}$ est un XOR.
- Dépassement de capacité ssi $c_{n-1} \neq c_n$ (exercice)

Multiplication en complément à 2

Multiplication en complément à 2

On reprend l'approche avec des équations :

$$X = -2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i$$

$$XY = -2^{n-1}x_{n-1}Y + \sum_{i=0}^{n-2} 2^i x_i Y$$

Multiplication en complément à 2

On reprend l'approche avec des équations :

$$X = -2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i$$

$$XY = -2^{n-1}x_{n-1}Y + \sum_{i=0}^{n-2} 2^i x_i Y$$

- Faudra remplacer une addition par une soustraction, et puis c'est tout
- Vous avez remarqué, on se fiche du codage de Y
- En fait il faut quand même faire attention au risque d'overflow intermédiaire
- (on verra çà : il suffit d'ajouter un bit au multiplicande et à la somme partielle)

Codages redondants et addition parallèle

Motivation : accélérer l'addition

- Propagation de retenue $\longrightarrow$ temps d'addition en $o(n)$
- En bricolant, on arrivera à $o(\log n)$
- On veut une addition en $o(1)$

Une petite escroquerie pour commencer

Voici un nouveau système de représentation des nombres :

- Un nombre entier X peut être représenté par une séquence de couples (c_i, s_i)
  tels que

$$X = \sum_{i=0}^{n-1} 2^i (c_i + s_i)$$

Une petite escroquerie pour commencer

Voici un nouveau système de représentation des nombres :

- Un nombre entier X peut être représenté par une séquence de couples (c_i, s_i)
  tels que
$$X = \sum_{i=0}^{n-1} 2^i (c_i + s_i)$$

- Cette représentation n'est plus unique. Et alors ? Cela s'appelle un codage redondant

Une petite escroquerie pour commencer

Voici un nouveau système de représentation des nombres :

- Un nombre entier X peut être représenté par une séquence de couples (c_i, s_i)
  tels que
$$X = \sum_{i=0}^{n-1} 2^i (c_i + s_i)$$
- Cette représentation n'est plus unique. Et alors ? Cela s'appelle un codage redondant
- On sait passer de cette représentation à la représentation standard. Comment ?

Une petite escroquerie pour commencer

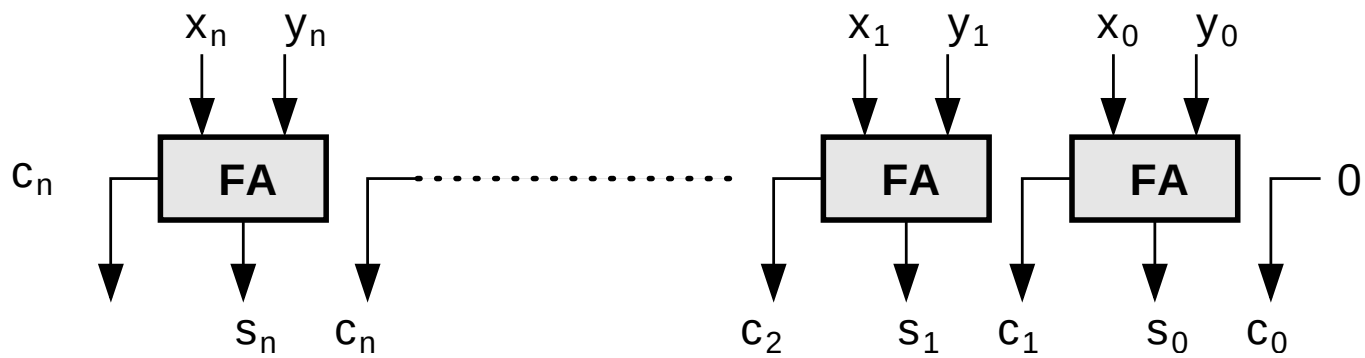
Voici un nouveau système de représentation des nombres :

- Un nombre entier X peut être représenté par une séquence de couples (c_i, s_i)
  tels que

$$X = \sum_{i=0}^{n-1} 2^i (c_i + s_i)$$

- Cette représentation n'est plus unique. Et alors ? Cela s'appelle un codage redondant
- On sait passer de cette représentation à la représentation standard. Comment ?
- Construisons un additionneur qui prend deux nombres en binaire normal, et renvoie sont résultat dans ce système de numération...

Mon premier additionneur parallèle



Cette représentation s'appelle *carry-save*, ou représentation à retenue conservée.

Et pour le même prix

Le même additionneur sait additionner un nombre en *carry-save* et un nombre en binaire normal, et produire son résultat en *carry-save*

Amoralité

Ce n'était pas une escroquerie :

- Si vous avez plein de nombres en binaires à additionner, conservez votre somme partielle en *carry-save*.
- Toutes vos additions intermédiares seront parallèles.
- (mais bon, le résultat sera en *carry-save*...)
- Si vous le voulez en binaire normal, il faut faire une addition à propagation de retenue tout de même.

Utilité : les multiplieurs, les diviseurs, plein de filtres... On verra tout cela.

Merci la redondance

C'est la redondance de notre représentation *carry-save* qui lui permet de manger les retenues sans les propager.
Généralisons donc.

- On était en base 2 $$X = \sum_{i=0}^{n-1} 2^i (c_i + s_i)$$
- On avait des chiffres entre 0 et 2 $$X = \sum_{i=0}^{n-1} 2^i (c_i + s_i)$$

On va se placer en base β quelconque et prendre nos chiffres entre ω et $\omega + q$...

Systèmes d'Avizienis

Le Muller Rose définit le système $S_{\beta,\omega,q,n}$ ainsi :

- Système de position en base β : $X = \sum_{i=0}^{n-1} \beta^i x_i$
- Un nombre est donc représenté par n chiffres
- Les chiffres x_i sont des entiers entre ω et $\omega + q$
- $\omega \in Z$, $q \in N+$

Les systèmes de numération classiques en base β sont des cas particuliers :
$\omega = 0$, $q = \beta - 1$.

Entiers représentables ?

Theorem

- *Si $q < \beta - 1$, il y a plein de trous dans l'ensemble des entiers représentables. Le système est peu intéressant...*
  Exemple : en base 10, quels sont les nombres représentables en n'utilisant que les chiffres 1 et 2 ?

Entiers représentables ?

Theorem

- *Si $q < \beta - 1$, il y a plein de trous dans l'ensemble des entiers représentables. Le système est peu intéressant...*
  Exemple : en base 10, quels sont les nombres représentables en n'utilisant que les chiffres 1 et 2 ?
- *Si $q = \beta - 1$, le système permet de représenter un intervalle connexe de $\mathbb{N}$. Chaque nombre admet une unique représentation.*
  Exemples :
  - *Notre bonne vieille base 10*
  - *La même, avec des chiffres entre -5 et 4. Quel est l'ensemble des nombres représentables ?*

Entiers représentables ?

Theorem

- *Si $q < \beta - 1$, il y a plein de trous dans l'ensemble des entiers représentables. Le système est peu intéressant...*
  Exemple : en base 10, quels sont les nombres représentables en n'utilisant que les chiffres 1 et 2 ?
- *Si $q = \beta - 1$, le système permet de représenter un intervalle connexe de N. Chaque nombre admet une unique représentation.*
  Exemples :
  - *Notre bonne vieille base 10*
  - *La même, avec des chiffres entre -5 et 4. Quel est l'ensemble des nombres représentables ?*
- *Si $q > \beta - 1$, le système permet de représenter un intervalle connexe de N, mais l'écriture d'un entier n'est plus nécessairement unique.*
  Exemple en base 10 avec les chiffres entre -6 et 7 :
  - *(1)(-3) = 07*

Pour un système utile

- Quitte à s'embêter, autant pouvoir représenter des entiers négatifs.
- On va prendre un ensemble de chiffres symétriques pour que tout nombre représentable admette un opposé représentable. Exit ω et q, on prend nos chiffres entre $-\alpha$ et α (compris)

Pour un système utile

- Quitte à s'embêter, autant pouvoir représenter des entiers négatifs.
- On va prendre un ensemble de chiffres symétriques pour que tout nombre représentable admette un opposé représentable. Exit ω et q, on prend nos chiffres entre $-\alpha$ et α (compris)
- On va faire en sorte que le signe soit donné par le chiffre de poids fort. Une condition suffisante est que $\alpha < \beta - 1$.

L'addition d'Avizienis

- Algo pour additionner X et Y :

$$\begin{cases} \forall i \quad t_{i+1} = \begin{array}{lll} -1 & \text{si} & x_i + y_i < -\alpha + 1 \\ 1 & \text{si} & x_i + y_i > \alpha - 1 \\ 0 & \text{si} & -\alpha + 1 \leq x_i + y_i \leq \alpha - 1 \end{array} \\ \forall i \quad w_i = x_i + y_i - \beta t_{i+1} \\ w_n = t_0 = 0 \\ \forall i \quad s_i = w_i + t_i \end{cases}$$

- Petit dessin pour vérifier que c'est bien parallèle

L'addition d'Avizienis

- Algo pour additionner X et Y :

$$\begin{cases} \forall i \quad t_{i+1} = \begin{cases} -1 & \text{si } x_i + y_i < -\alpha + 1 \\ 1 & \text{si } x_i + y_i > \alpha - 1 \\ 0 & \text{si } -\alpha + 1 \leq x_i + y_i \leq \alpha - 1 \end{cases} \\ \forall i \quad w_i = x_i + y_i - \beta t_{i+1} \\ w_n = t_0 = 0 \\ \forall i \quad s_i = w_i + t_i \end{cases}$$

- Petit dessin pour vérifier que c'est bien parallèle
- Intuition : considérant $x_i + y_i$, on sort une retenue telle qu'on sera capable de manger n'importe quelle retenue entrante

L'addition d'Avizienis

- Algo pour additionner X et Y :

$$\begin{cases} \forall i \quad t_{i+1} = \begin{cases} -1 & \text{si } x_i + y_i < -\alpha + 1 \\ 1 & \text{si } x_i + y_i > \alpha - 1 \\ 0 & \text{si } -\alpha + 1 \leq x_i + y_i \leq \alpha - 1 \end{cases} \\ \forall i \quad w_i = x_i + y_i - \beta t_{i+1} \\ w_n = t_0 = 0 \\ \forall i \quad s_i = w_i + t_i \end{cases}$$

- Petit dessin pour vérifier que c'est bien parallèle
- Intuition : considérant $x_i + y_i$, on sort une retenue telle qu'on sera capable de manger n'importe quelle retenue entrante
- cf Muller Rose pour la démo : il suffit de vérifier que les t_i sont ajoutés et retranchés, et que les s_i sont bien des chiffres.

L'addition d'Avizienis

- Algo pour additionner X et Y :

$$\begin{cases} \forall i \quad t_{i+1} = \begin{array}{lll} -1 & \text{si} & x_i + y_i < -\alpha + 1 \\ 1 & \text{si} & x_i + y_i > \alpha - 1 \\ 0 & \text{si} & -\alpha + 1 \leq x_i + y_i \leq \alpha - 1 \end{array} \\ \forall i \quad w_i = x_i + y_i - \beta t_{i+1} \\ w_n = t_0 = 0 \\ \forall i \quad s_i = w_i + t_i \end{cases}$$

- Petit dessin pour vérifier que c'est bien parallèle
- Intuition : considérant $x_i + y_i$, on sort une retenue telle qu'on sera capable de manger n'importe quelle retenue entrante
- cf Muller Rose pour la démo : il suffit de vérifier que les t_i sont ajoutés et retranchés, et que les s_i sont bien des chiffres.
- Cela marche sous la condition $2\alpha \geq \beta + 1$.

L'addition d'Avizienis

- Algo pour additionner X et Y :

$$\begin{cases} \forall i \quad t_{i+1} = \begin{cases} -1 & \text{si } x_i + y_i < -\alpha + 1 \\ 1 & \text{si } x_i + y_i > \alpha - 1 \\ 0 & \text{si } -\alpha + 1 \leq x_i + y_i \leq \alpha - 1 \end{cases} \\ \forall i \quad w_i = x_i + y_i - \beta t_{i+1} \\ w_n = t_0 = 0 \\ \forall i \quad s_i = w_i + t_i \end{cases}$$

- Petit dessin pour vérifier que c'est bien parallèle
- Intuition : considérant $x_i + y_i$, on sort une retenue telle qu'on sera capable de manger n'importe quelle retenue entrante
- cf Muller Rose pour la démo : il suffit de vérifier que les t_i sont ajoutés et retranchés, et que les s_i sont bien des chiffres.
- Cela marche sous la condition $2\alpha \geq \beta + 1$.
- Donc cela ne parche pas pour le binaire signé, mais on peut bricoler...

Remarques sur les système d'Avizienis

Remarques sur les système d'Avizienis

- Choix du codage des chiffres ?

Remarques sur les système d'Avizienis

- Choix du codage des chiffres ?
- LCDE : La comparaison reste séquentielle :
  - comparer 0111 et $1000\overline{1}$
- La division aussi...

Codage modulaire

Ya qu'un écran

- Soit $\mu_1, \mu_2, ...\mu_n$ des entiers premiers entre eux deux à deux.
- Tout nombre entier X compris entre 0 et $\prod_i \mu_i - 1$ peut se représenter de manière unique par le vecteur $(m_1, m_2...m_n)$ où $m_i \equiv X[\mu_i]$
- L'addition, la soustraction, la multiplication se font en parallèle sur les m_i.
- LCDE : on a perdu
  - les ordres de grandeurs
  - donc la comparaison
  - et la détection des dépassements de capacité,
  - et la division
  - donc la conversion en un système de représentation usuel
- Applications en traitement du signal et cryptographie

Codages des réels

Codage en virgule fixe

- Rationnels positifs : l'écriture $x_{n-1}..x_0, x_{-1}..x_{-p}$ représente le rationnel

$$X = \sum_{i=-p}^{n-1} \beta^i x_i$$

Codage en virgule fixe

- Rationnels positifs : l'écriture $x_{n-1}..x_0, x_{-1}..x_{-p}$ représente le rationnel

$$X = \sum_{i=-p}^{n-1} \beta^i x_i$$

- Rationnels négatifs : le complément à la base marche toujours.

Codage en virgule fixe

- Rationnels positifs : l'écriture $x_{n-1}..x_0, x_{-1}..x_{-p}$ représente le rationnel

$$X = \sum_{i=-p}^{n-1} \beta^i x_i$$

- Rationnels négatifs : le complément à la base marche toujours.
- Les additions/soustractions marchent toujours.
  - Faut gérer la position de la virgule

Codage en virgule fixe

- Rationnels positifs : l'écriture $x_{n-1}..x_0, x_{-1}..x_{-p}$ représente le rationnel

$$X = \sum_{i=-p}^{n-1} \beta^i x_i$$

- Rationnels négatifs : le complément à la base marche toujours.
- Les additions/soustractions marchent toujours.
  - Faut gérer la position de la virgule

Quelques exemples utiles en pratique :

- Les entiers sont un cas particulier.

Codage en virgule fixe

- Rationnels positifs : l'écriture $x_{n-1}..x_0, x_{-1}..x_{-p}$ représente le rationnel

$$X = \sum_{i=-p}^{n-1} \beta^i x_i$$

- Rationnels négatifs : le complément à la base marche toujours.
- Les additions/soustractions marchent toujours.
  - Faut gérer la position de la virgule

Quelques exemples utiles en pratique :

- Les entiers sont un cas particulier.
- Nombres compris entre 0 et 1 : $n = 0$. Dans ce cas 0 a une représentation mais pas 1.

Codages en virgule flottante

- $x = s.m.\beta^e$

Codages en virgule flottante

- $x = s.m.\beta^e$
- Addition, soustraction, multiplication et division se ramènent à des opérations sur les mantisses et les exposants

Codages en virgule flottante

- $x = s.m.\beta^e$
- Addition, soustraction, multiplication et division se ramènent à des opérations sur les mantisses et les exposants
- Opérations parfois inexactes : le résultat doit être arrondi

Codages en virgule flottante

- $x = s.m.\beta^e$
- Addition, soustraction, multiplication et division se ramènent à des opérations sur les mantisses et les exposants
- Opérations parfois inexactes : le résultat doit être arrondi
- On sait réaliser avec un coût raisonnable “le meilleur arrondi possible”

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$
- l_x est représenté en binaire en complément à 2, par exemple

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$
- l_x est représenté en binaire en complément à 2, par exemple
- Précision et intervalle représentables similaires au système flottant

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$
- l_x est représenté en binaire en complément à 2, par exemple
- Précision et intervalle représentables similaires au système flottant
- Multiplication, division et racine carrée triviales

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$
- l_x est représenté en binaire en complément à 2, par exemple
- Précision et intervalle représentables similaires au système flottant
- Multiplication, division et racine carrée triviales
- LCDE : addition et soustraction (très) difficiles

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$
- l_x est représenté en binaire en complément à 2, par exemple
- Précision et intervalle représentables similaires au système flottant
- Multiplication, division et racine carrée triviales
- LCDE : addition et soustraction (très) difficiles
- Multiplication et division exacte, addition/soustraction inexacte

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$
- l_x est représenté en binaire en complément à 2, par exemple
- Précision et intervalle représentables similaires au système flottant
- Multiplication, division et racine carrée triviales
- LCDE : addition et soustraction (très) difficiles
- Multiplication et division exacte, addition/soustraction inexacte
- Il n'est pas possible de réaliser à un coût raisonnable le "meilleur arrondi possible" pour l'addition/soustraction.

Codages logarithmiques

- $x = s.\beta^{l_x}$ où $l_x = \log_\beta |x|$
- l_x est représenté en binaire en complément à 2, par exemple
- Précision et intervalle représentables similaires au système flottant
- Multiplication, division et racine carrée triviales
- LCDE : addition et soustraction (très) difficiles
- Multiplication et division exacte, addition/soustraction inexacte
- Il n'est pas possible de réaliser à un coût raisonnable le "meilleur arrondi possible" pour l'addition/soustraction.

Applications ? Problème ouvert à mon avis.

Conclusions

Yen a d'autres

- On n'a présenté que les codages utiles au calcul, c'est-à-dire pour lesquels on sait faire au moins une opération.
- Il y en a plein d'autres rigolos mais qui ne remplissent pas cette condition...