

Arithmétique du processeur

Christophe Moulleron



À disposition = Unité arithmétique et logique (ALU) :

- opérations logiques bit à bit
- addition et soustraction d'entiers
- drapeaux

Besoin = Autres opérations courantes :

- comparaisons d'entiers
- multiplication d'entiers
- calcul flottant

- 1 Comparaison d'entiers
- 2 Multiplication d'entiers
- 3 Arithmétique flottante

Test d'égalité $a = b$

En non-signé : $a, b \in \llbracket 0, 2^n - 1 \rrbracket$

Donc $b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$

Et $b - a \equiv 0 \pmod{2^n}$ ssi $b - a = 0$.

Test d'égalité $a = b$

En non-signé : $a, b \in \llbracket 0, 2^n - 1 \rrbracket$

Donc $b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$

Et $b - a \equiv 0 \pmod{2^n}$ ssi $b - a = 0$.

En signé : $a, b \in \llbracket -2^{n-1}, 2^{n-1} - 1 \rrbracket$

Donc $b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$

Et $b - a \equiv 0 \pmod{2^n}$ ssi $b - a = 0$.

Test d'égalité $a = b$

En non-signé : $a, b \in \llbracket 0, 2^n - 1 \rrbracket$

Donc $b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$

Et $b - a \equiv 0 \pmod{2^n}$ ssi $b - a = 0$.

En signé : $a, b \in \llbracket -2^{n-1}, 2^{n-1} - 1 \rrbracket$

Donc $b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$

Et $b - a \equiv 0 \pmod{2^n}$ ssi $b - a = 0$.

Propriété

Pour tester si $a = b$, il suffit d'utiliser l'ALU pour calculer $b - a$. On a alors :

$$a = b \Leftrightarrow Z = 1.$$

Test $a \geq b$ en non-signé

Propriété

Pour tester si $a \geq b$ en arithmétique non-signée, il suffit d'utiliser l'ALU pour calculer $b - a$. On a alors :

$$a \geq b \Leftrightarrow Z = 1 \text{ ou } C = 1.$$

Preuve : Soient $a, b \in \llbracket 0, 2^n - 1 \rrbracket$

$$b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$$

Test $a \geq b$ en non-signé

Propriété

Pour tester si $a \geq b$ en arithmétique non-signée, il suffit d'utiliser l'ALU pour calculer $b - a$. On a alors :

$$a \geq b \Leftrightarrow Z = 1 \text{ ou } C = 1.$$

Preuve : Soient $a, b \in \llbracket 0, 2^n - 1 \rrbracket$

$$b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$$

- $b - a = 0$

résultat de l'ALU = 0 \rightsquigarrow condition vérifiée et $Z = 1$

$$C = 0$$

Test $a \geq b$ en non-signé

Propriété

Pour tester si $a \geq b$ en arithmétique non-signée, il suffit d'utiliser l'ALU pour calculer $b - a$. On a alors :

$$a \geq b \Leftrightarrow Z = 1 \text{ ou } C = 1.$$

Preuve : Soient $a, b \in \llbracket 0, 2^n - 1 \rrbracket$

$$b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$$

- $b - a = 0$

résultat de l'ALU = 0 \rightsquigarrow condition vérifiée et $Z = 1$ $C = 0$

- $b - a \in \llbracket 1, 2^n - 1 \rrbracket$

résultat de l'ALU = $b - a$ \rightsquigarrow condition non vérifiée et $Z = C = 0$

Test $a \geq b$ en non-signé

Propriété

Pour tester si $a \geq b$ en arithmétique non-signée, il suffit d'utiliser l'ALU pour calculer $b - a$. On a alors :

$$a \geq b \Leftrightarrow Z = 1 \text{ ou } C = 1.$$

Preuve : Soient $a, b \in \llbracket 0, 2^n - 1 \rrbracket$

$$b - a \in \llbracket 1 - 2^n, 2^n - 1 \rrbracket$$

- $b - a = 0$

résultat de l'ALU = 0 \rightsquigarrow condition vérifiée et $Z = 1$ $C = 0$

- $b - a \in \llbracket 1, 2^n - 1 \rrbracket$

résultat de l'ALU = $b - a$ \rightsquigarrow condition non vérifiée et $Z = C = 0$

- $b - a \in \llbracket 1 - 2^n, -1 \rrbracket$

résultat de l'ALU = $b - a + 2^n$ \rightsquigarrow condition vérifiée et $C = 1$ $Z = 0$

Autres cas en arith. non-signée :

par déduction

- $a > b \Leftrightarrow C = 1$ et $Z = 0$ sur $b - a$
- $a \leq b$ ssi $b \geq a$, $a < b$ ssi $b > a$

Inégalités, cas général

Autres cas en arith. non-signée :

par déduction

- $a > b \Leftrightarrow C = 1$ et $Z = 0$ sur $b - a$
- $a \leq b$ ssi $b \geq a$, $a < b$ ssi $b > a$

En arith. signée :

- utiliser les drapeaux S et O
- raisonnement similaire

Propriété

Pour tester si $a > b$ en arithmétique signée, il suffit d'utiliser l'ALU pour calculer $b - a$. On a alors :

$$a > b \Leftrightarrow$$

Inégalités, cas général

Autres cas en arith. non-signée :

par déduction

- $a > b \Leftrightarrow C = 1$ et $Z = 0$ sur $b - a$
- $a \leq b$ ssi $b \geq a$, $a < b$ ssi $b > a$

En arith. signée :

- utiliser les drapeaux S et O
- raisonnement similaire

Propriété

Pour tester si $a > b$ en arithmétique signée, il suffit d'utiliser l'ALU pour calculer $b - a$. On a alors :

$$a > b \Leftrightarrow S \neq O.$$

Plan

- 1 Comparaison d'entiers
- 2 Multiplication d'entiers**
- 3 Arithmétique flottante

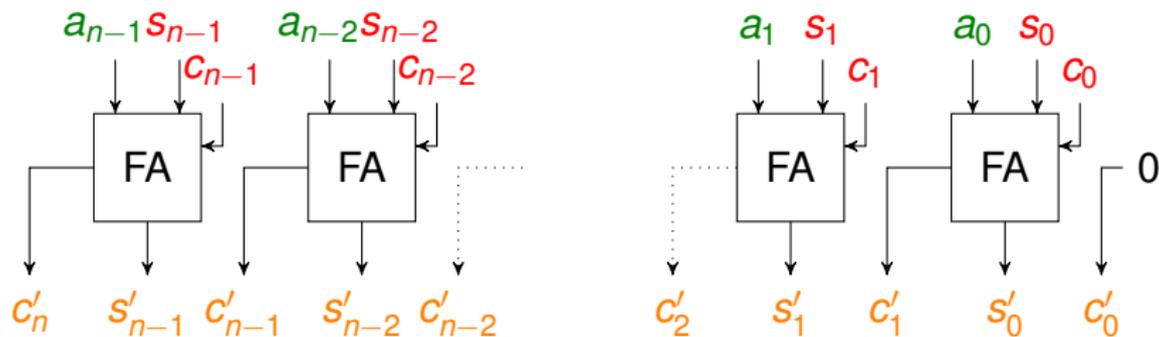
Multiplication d'entiers non-signés

$$\begin{array}{r} \times \\ \\ \hline + \\ + \\ + \\ + \\ \hline c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0 \end{array}$$

- opérandes sur n chiffres \Rightarrow résultat sur $2n$ chiffres
- offre valable quelque soit la base
- en base 2, $b_i \in \{0, 1\} \Rightarrow$ mult. par b_i triviale
- vrai problème = **addition efficace de n entiers**

Addition *carry-save*, le retour

Rappel : *binaire* + *carry-save* = *carry-save*



Bilan sur l'*additionneur carry-save* :

- ✓ addition toujours *en parallèle* latence $\Theta(1)$
- ✓ processus itérable pour sommer > 2 entiers
- ✗ conversion *carry-save* \rightarrow *binaire* = vraie addition $C' + S'$ coûteux

Coût de la multiplication en non-signé

Multiplication $a \times b$ en arithmétique non-signée :

- 1 calcul des produits $a \times b_i$
→ n séries de n portes ET en parallèle $O(1) / O(n^2)$
- 2 passage de $a \times b_0$ en *carry-save*
→ fils gratuit / \approx gratuit
- 3 ajouts des $a \times b_i \times 2^i$ pour $i > 0$ en *carry-save*
→ $n - 1$ rangées de $O(n)$ FAs en parallèle $O(n) / O(n^2)$
- 4 conversion *carry-save* → *binaire* finale
→ 1 addition sur $2n$ bits (propag. retenue) $O(n) / O(n)$

Bilan : profondeur en $O(n)$ et surface en $O(n^2)$

Améliorations possibles

Améliorer la somme des n lignes :

- somme selon un arbre binaire équilibré
 - nécessite une addition *carry-save* + *carry-save* parallèle
- généralisation du FA :
 - arbres de Wallace $W_n : (a_0, \dots, a_{n-1}) \mapsto \sum_i a_i$
- compression futée du parallélogramme à l'aide de HA/FA
 - idée = maximiser la rentabilité des *full adders*

Améliorations possibles

Améliorer la somme des n lignes :

- somme selon un arbre binaire équilibré
 - nécessite une addition *carry-save* + *carry-save* parallèle
- généralisation du FA :
 - arbres de Wallace $W_n : (a_0, \dots, a_{n-1}) \mapsto \sum_i a_i$
- compression futée du parallélogramme à l'aide de HA/FA
 - idée = maximiser la rentabilité des *full adders*

Diminuer la profondeur :

- multiplieur **pipeliné**
 - somme partielle stockée dans un registre / calcul en n cycles

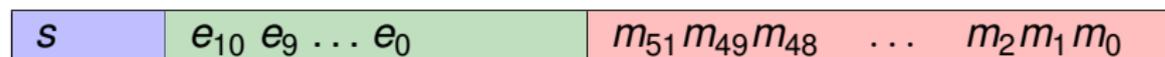
Plan

- 1 Comparaison d'entiers
- 2 Multiplication d'entiers
- 3 Arithmétique flottante

Représentation des nombres flottants

Flottants : Norme IEEE 754

ici, flottants 64 bits normalisés



1 bit

signe

11 bits

exposant

entier non-signé E

52 bits

mantisse

entier non-signée M

Valeur représentée :

notation scientifique

$$F = (-1)^s \times \left(1 + \frac{M}{2^{52}} \right) \times 2^{(E-1023)}$$

Représentation des nombres flottants (2)

Cas particuliers :

$E = 0$ et $M = 0$ valeur ± 0

$E = 0$ et $M \neq 0$ nombre dénormalisé

valeurs autour de 0

$E = 2047$ infinis et NaN

Représentation des nombres flottants (2)

Cas particuliers :

$E = 0$ et $M = 0$ valeur ± 0

$E = 0$ et $M \neq 0$ nombre dénormalisé

$E = 2047$ infinis et NaN

valeurs autour de 0

Comparaison des flottants normalisés :

- 1 comparer les signes
- 2 comparer les exposants
- 3 comparer les mantisses

poids le + fort

poids forts

poids faibles

Intérêt du **biais sur l'exposant** ?

$E - 1023$

Représentation des nombres flottants (2)

Cas particuliers :

$E = 0$ et $M = 0$ valeur ± 0

$E = 0$ et $M \neq 0$ nombre dénormalisé

$E = 2047$ infinis et NaN

valeurs autour de 0

Comparaison des flottants normalisés :

- 1 comparer les signes
- 2 comparer les exposants
- 3 comparer les mantisses

poids le + fort

poids forts

poids faibles

Intérêt du **biais sur l'exposant** ?

$E - 1023$

- comparaison de flottants **via une ALU 64 bits**

Circuit de calcul avec des flottants

Approche générale :

① détecter/traiter les cas particuliers

↪ entrées non normalisées

$\pm\infty$, ± 0 , NAN, ...

② extraire les signes/exposants/mantisses

↪ opérande $A \rightarrow s_A, M_A, E_A$

↪ opérande $B \rightarrow s_B, M_B, E_B$

③ calcul le signe/exposant/mantisse du résultat

↪ s, M et E non normalisés

④ produire la sortie

↪ gérer à part les dépassements de capacité

↪ normalisation + arrondi correct pour M