# Arithmétique du processeur

Christophe Mouilleron

ensiie
école nationale supérieure d'informatique
pour l'industrie et l'entreprise

# Introduction

**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

# Plan

# Test d'égalité $a = b$

**En non-signé :** $a, b \in [\![0,\ 2^n - 1]\!]$

Donc $b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

Et $b - a \equiv 0 \mod 2^n$ ssi $b - a = 0$.

# Test d'égalité $a = b$

**En non-signé :** $a, b \in [\![0,\ 2^n - 1]\!]$

Donc $b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

Et $b - a \equiv 0 \mod 2^n$ ssi $b - a = 0$.

**En signé :** $a, b \in [\![-2^{n-1},\ 2^{n-1} - 1]\!]$

Donc $b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

Et $b - a \equiv 0 \mod 2^n$ ssi $b - a = 0$.

# Test d'égalité $a = b$

**En non-signé :** $a, b \in [\![0,\ 2^n - 1]\!]$

Donc $b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

Et $b - a \equiv 0 \mod 2^n$ ssi $b - a = 0$.

**En signé :** $a, b \in [\![-2^{n-1},\ 2^{n-1} - 1]\!]$

Donc $b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

Et $b - a \equiv 0 \mod 2^n$ 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 $a, b \in [\![0, 2^n - 1]\!]$ $\quad b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

# 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 $a, b \in [\![0, 2^n - 1]\!]$ $\quad b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

- $b - a = 0$

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

# 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 $a, b \in [\![0, 2^n - 1]\!]$ $\quad b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

- $b - a = 0$

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

- $b - a \in [\![1,\ 2^n - 1]\!]$

  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 $a, b \in [\![0, 2^n - 1]\!]$ $\quad b - a \in [\![1 - 2^n,\ 2^n - 1]\!]$

- $b - a = 0$

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

- $b - a \in [\![1,\ 2^n - 1]\!]$

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

- $b - a \in [\![1 - 2^n,\ -1]\!]$

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

# 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 $b > a$

# 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

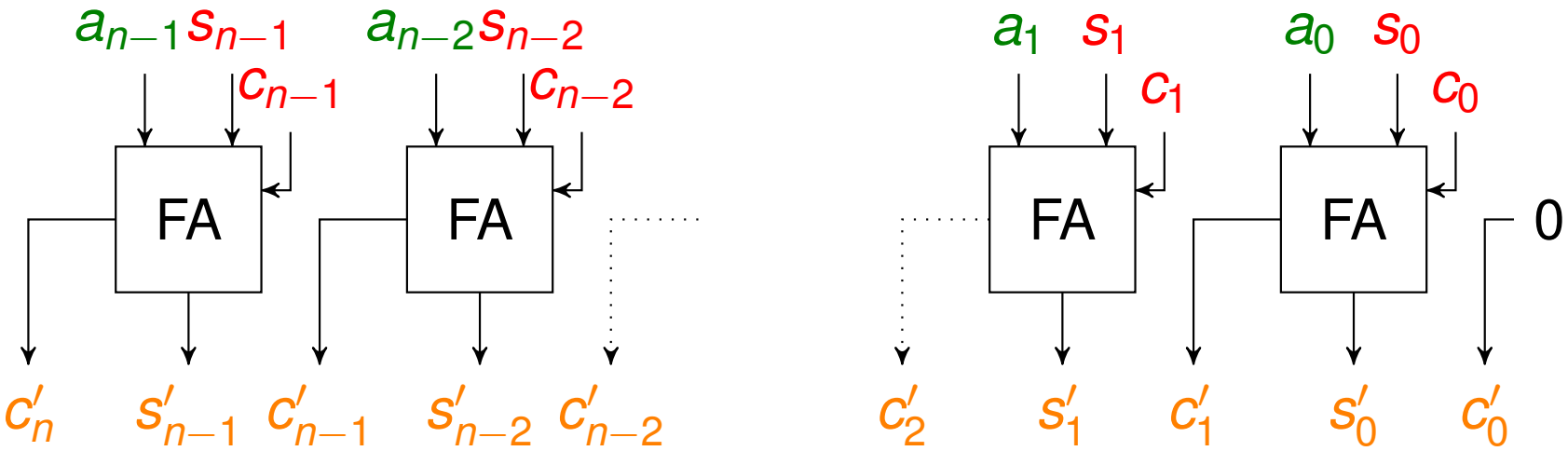
# Multiplication d'entiers non-signés

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
| $\times$ | | | | | $b_3$ | $b_2$ | $b_1$ | $b_0$ |
| $+$ | | | | | $b_0\ a_3$ | $b_0\ a_2$ | $b_0\ a_1$ | $b_0\ a_0$ |
| $+$ | | | | $b_1\ a_3$ | $b_1\ a_2$ | $b_1\ a_1$ | $b_1\ a_0$ | |
| $+$ | | | $b_2\ a_3$ | $b_2\ a_2$ | $b_2\ a_1$ | $b_2\ a_0$ | | |
| $+$ | | $b_3\ a_3$ | $b_3\ a_2$ | $b_3\ a_1$ | $b_3\ a_0$ | | | |
| | $c_7$ | $c_6$ | $c_5$ | $c_4$ | $c_3$ | $c_2$ | $c_1$ | $c_0$ |

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

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

# Et en complément à 2 alors ?

$$
\begin{array}{cccccccccc}
 & & & & & 1 & 0 & 0 & 1 \\
\times & & & & & 1 & 1 & 1 & 0 \\
\hline
+ & & & & & 0 & 0 & 0 & 0 \\
+ & & & & 1 & 0 & 0 & 1 & \\
+ & & & 1 & 0 & 0 & 1 & & \\
+ & & 1 & 0 & 0 & 1 & & & \\
\hline
 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

$\overline{1001}^{(2)} \times \overline{1110}^{(2)} = (-7) \times (-2) = 14$ $\qquad$ $\overline{01111110}^{(2)} = 126$

**FAUX ? !**

# Et en complément à 2 alors ?

$$
\begin{array}{cccccccccc}
 & & & & & 1 & 0 & 0 & 1 \\
\times & & & & & 1 & 1 & 1 & 0 \\
\hline
+ & & & & & 0 & 0 & 0 & 0 \\
+ & & & & 1 & 0 & 0 & 1 & \\
+ & & & 1 & 0 & 0 & 1 & & \\
+ & & 1 & 0 & 0 & 1 & & & \\
\hline
 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

$\overline{1001}^{(2)} \times \overline{1110}^{(2)} = (-7) \times (-2) = 14$

$\overline{01111110}^{(2)} = 126$

Résultat garanti modulo $2^n = 16$

$126 \equiv 14 \equiv -2 \mod 16$

# Et en complément à 2 alors ?

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | 1 | 0 | 0 | 1 |
| $\times$ | | | | | 1 | 1 | 1 | 0 |
| $+$ | | | | | 0 | 0 | 0 | 0 |
| $+$ | 1 | 1 | 1 | 1 | 0 | 0 | 1 | |
| $+$ | 1 | 1 | 1 | 0 | 0 | 1 | | |
| $-$ | 1 | 1 | 0 | 0 | 1 | | | |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

$\overline{1001}^{(2)} \times \overline{1110}^{(2)} = (-7) \times (-2) = 14$ $\qquad$ $\overline{00001110}^{(2)} = 14$

Résultat correct si calculs modulo $2^{2n}$ :

- extensions de signes $\qquad n \to 2n$ bits
- soustraction finale $\qquad b = -2^{n-1}\, b_{n-1} + \ldots$

# Améliorations possibles

Améliorer la somme des $n$ lignes :

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

# Améliorations possibles

Améliorer la somme des $n$ lignes :

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

Diminuer la profondeur :

- multiplieur pipeliné
  $\rightarrow$ somme partielle stockée dans un registre / calcul en $n$ cycles

# Plan

# Représentation des nombres flottants

**Flottants** : Norme IEEE 754 ici, flottants 64 bits normalisés


| 1 bit | 11 bits | 52 bits |
|---|---|---|
| signe | exposant | mantisse |
| | entier non-signé $E$ | 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 $\pm 0$ | |
| $E = 0$ 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 $\pm 0$ | |
| $E = 0$ et $M \neq 0$ | nombre dénormalisé | valeurs autour de 0 |
| $E = 2047$ | infinis et NaN | |

Comparaison des flottants normalisés :

1. comparer les signes — poids le + fort
2. comparer les exposants — poids forts
3. comparer les mantisses — poids faibles

Intérêt du biais sur l'exposant ? $E - 1023$

# Représentation des nombres flottants (2)

Cas particuliers :

| | | |
|---|---|---|
| $E = 0$ et $M = 0$ | valeur $\pm 0$ | |
| $E = 0$ et $M \neq 0$ | nombre dénormalisé | valeurs autour de 0 |
| $E = 2047$ | infinis et NaN | |

Comparaison des flottants normalisés :

1. comparer les signes — poids le + fort
2. comparer les exposants — poids forts
3. comparer les mantisses — 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
   - $\rightsquigarrow$ entrées non normalisées $\pm\infty$, $\pm 0$, NAN, ...
2. extraire les signes/exposants/mantisses
   - $\rightsquigarrow$ opérande $A \rightarrow s_A$, $M_A$, $E_A$
   - $\rightsquigarrow$ opérande $B \rightarrow s_B$, $M_B$, $E_B$
3. calcul le signe/exposant/mantisse du résultat
   - $\rightsquigarrow$ $s$, $M$ et $E$ non normalisés
4. produire la sortie
   - $\rightsquigarrow$ gérer à part les dépassements de capacité
   - $\rightsquigarrow$ normalisation + arrondi correct pour $M$