Introduction aux Opérateurs Arithmétiques

Alain GUYOT

Concurrent Integrated Systems
TIMA

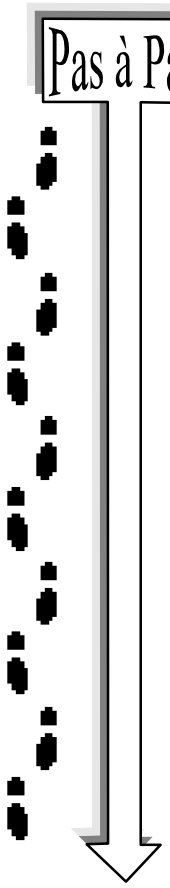
(33) 04 76 57 46 16

Alain.Guyot@imag.fr

http://tima-cmp.imag.fr/Homepages/guyot

Techniques de l'Informatique et de la Microélectronique pour l'Architecture. Unité associée au C.N.R.S. n° B0706

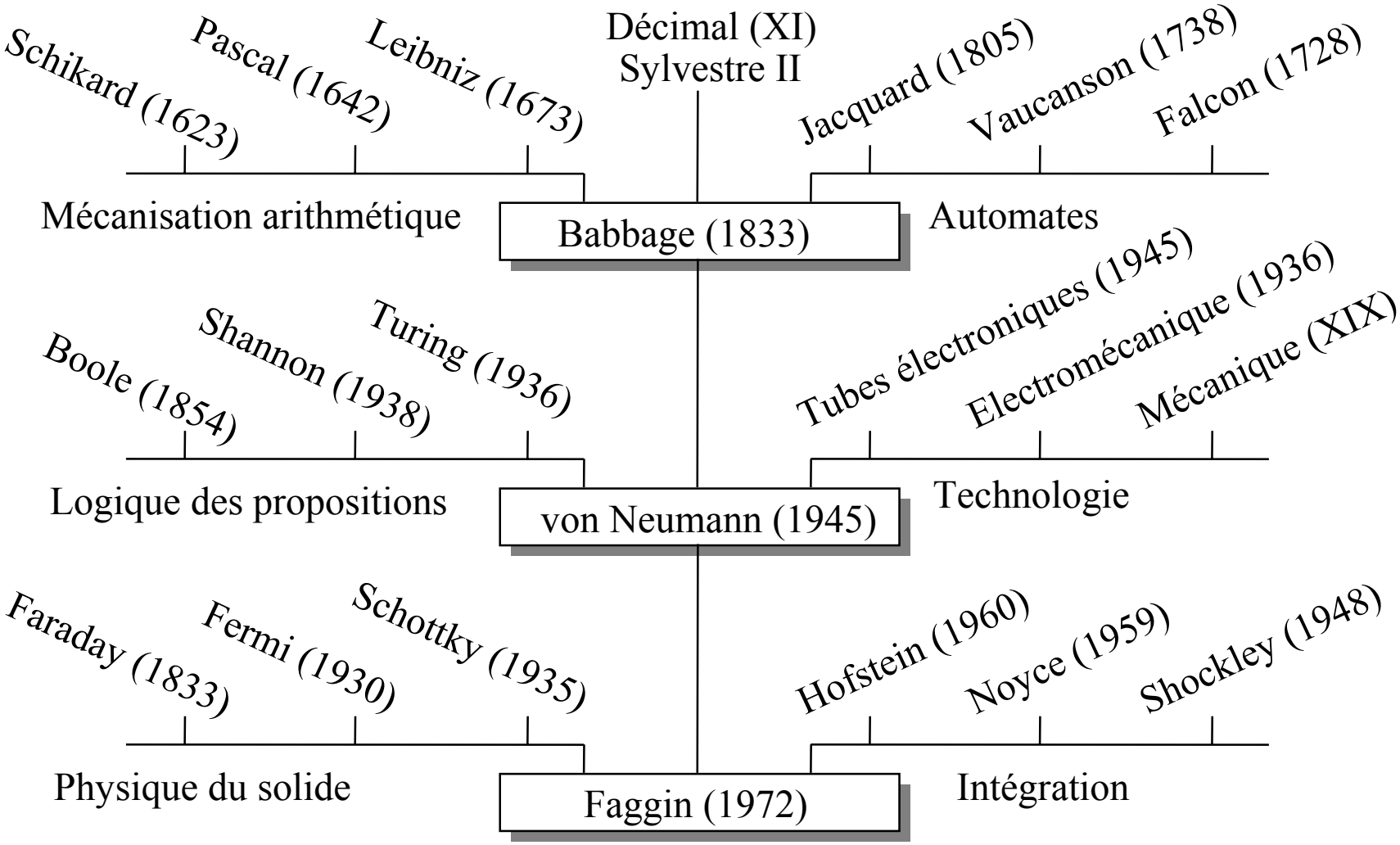
Issues de ce cours



1- Introduction
2- Addition matérielle
3- Multiplication matérielle
4- Résidus
5- Division
6- Racine carrée
7- Virgule Flottante
8- Fonctions élémentaires

Remarques:
- *L'addition et la multiplication sont assez exhaustives.*
- *La division est la racine carrée sont simplifiées.*
- *La virgule flottante et les fonctions sont survolées.*

La mécanisation du calcul



NonBut de ce cours

Le but de ce cours n'est pas d'enseigner comment on fait un opérateur arithmétique avec des portes logiques ou des transistors.

Les opérations arithmétiques des ordinateurs ne sont qu'une transposition à la base 2 des algorithmes humainement utilisés en base 10 depuis plusieurs siècles.

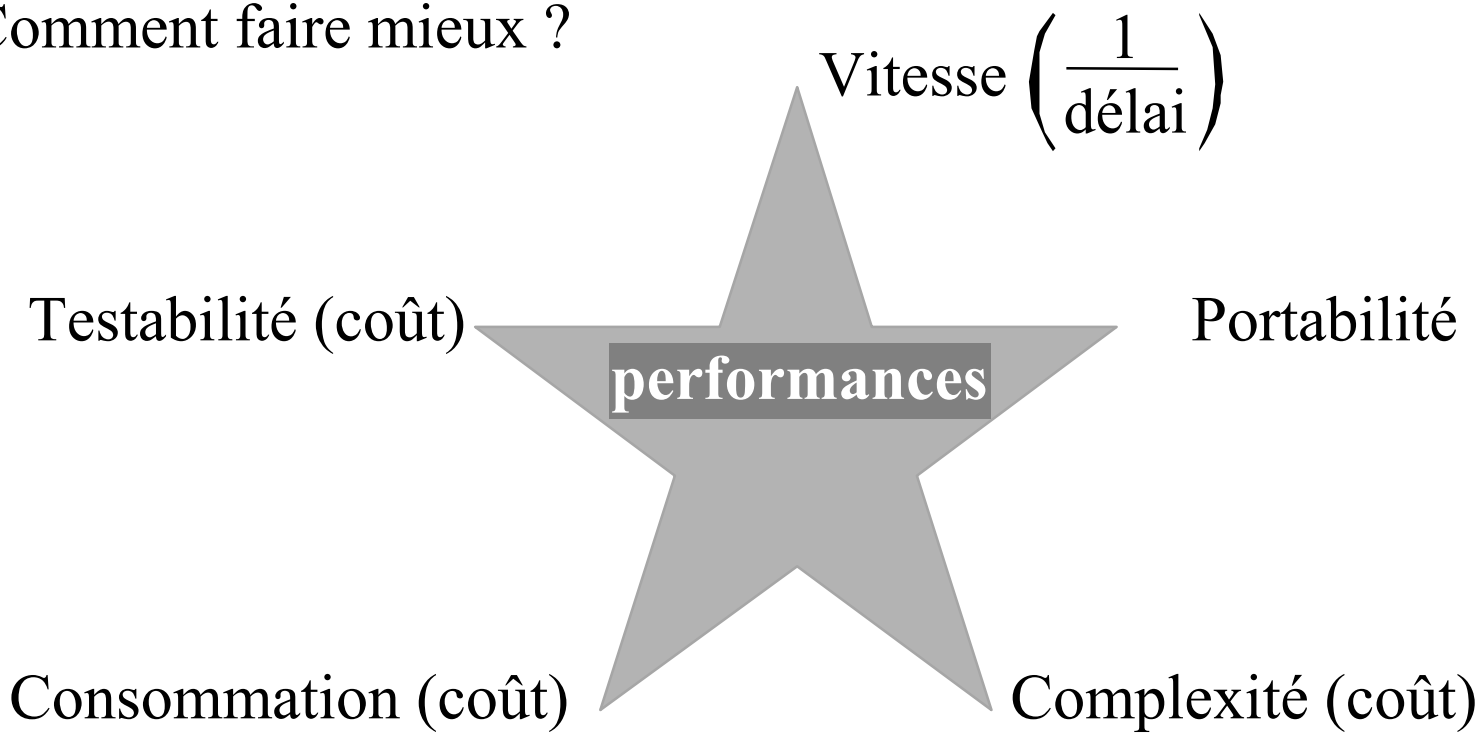
Un élève qui sait effectuer une addition, une multiplication, une division, une extraction de racine carrée ou le calcul d'une fonction élémentaire peut concevoir un opérateur logique effectuant mécaniquement cette opération.

On utilise la base 2 uniquement parce que c'est technologiquement plus efficace. Cependant certaines opérations marchent "mieux" dans d'autres systèmes.

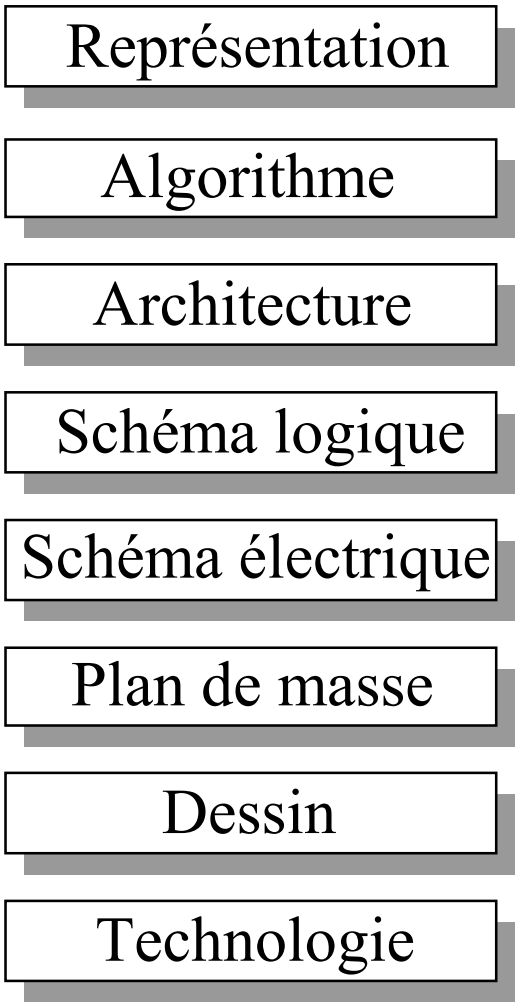
But de ce cours

Faire un opérateur naïf étant trivial , le but de ce cours est d'enseigner comment faire des opérateurs arithmétiques mieux.

Qu'est-ce que mieux ?
Pourquoi faire mieux ?
Comment faire mieux ?



Domaines et Modèles



•Ce cours établit des modèles simples de coût et délai pour comparer des architectures d'opérateurs combinatoires.

•le coût est évalué en nombre de portes ou de cellules

•le délai est en général le nombre de cellules sur le chemin critique

•la "sortante" est prise en compte si elle est importante pour

- dimensionner
- introduire des "buffers"

Choix de la représentation la plus appropriée

Représentation	Base	valeurs chiffre	valeur nombre	bit(s) / chiffre	nom usuel
standard	2	0,1	$\sum_{i=0}^{n} a_i 2^i$	1	conventionnelle
complément à 2^{n+1}	2	0,1	$-a_n 2^n + \sum_{i=0}^{n-1} a_i 2^i$	1	complément à 2
multiplicative	2	0,1	$\prod_{i=1}^{n} (1+2^{-i})^{a_i}$	1	
redondante	2	0,1,2	$\sum_{i=0}^{n} a_i 2^i$	2	carry save
chiffre signé	2	-1,0,+1	$\sum_{i=0}^{n} a_i 2^i$	2	borrow save
chiffre signé	4	-2,-1,0,1,2	$\sum_{i=0}^{n} a_i 4^i$	3	code de Booth
additive	$\log(1+2^{-i})$	0,1	$\sum_{i=0}^{n} a_i \log(1+2^{-i})$	1	
additive	$\mathrm{arctg}(2^{-i})$	-1,1	$\sum_{i=0}^{n} a_i \mathrm{arctg}(2^{-i})$	1	

Ouvrages & URL recommandés

Arithmétique des Ordinateurs, opérateurs et fonctions élémentaires
J.-M. Muller, Masson 1989 ISBN: 2225816891

Computer Arithmetic Algorithms
I. Koren, Prentice Hall 1993 ISBN: 0131519522

Computer Arithmetic Systems, algorithms, architecture & implementation
A. M. Omondi, Prentice Hall 1994 ISBN: 0133343014

Elementary functions: Algorithms & implementation
J.-M. Muller, Birkauser 1997 ISBN: 081763990

http://liinwww.ira.uka.de/bibliography/Theory/arith.html
http://www.eecs.lehigh.edu/~mschulte/hsca/START.html
http://http.cs.berkeley.edu/~wkahan/ieee754status

Addition Matérielle

Alain GUYOT

Concurrent Integrated Systems
TIMA

(33) 04 76 57 46 16

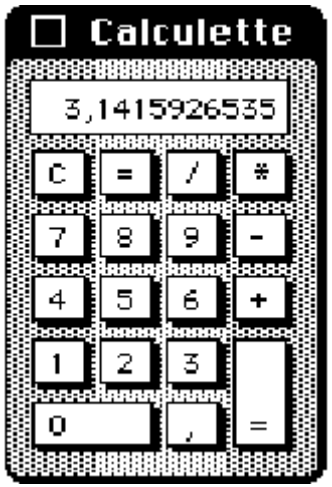
Alain.Guyot@imag.fr

http://tima-cmp.imag.fr/Homepages/guyot

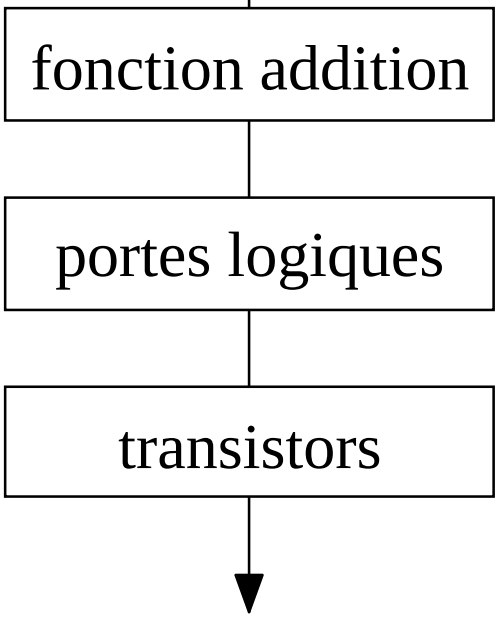
Techniques de l'Informatique et de la Microélectronique
pour l'Architecture. Unité associée au C.N.R.S. n° B0706

But
Réaliser des additionneurs combinatoires.
Optimiser le coût et/ou la vitesse
et/ou la consommation.

Problème
- Propagation de
  la retenue



Contraintes:
délai max
consommation max
surface max.



Remarque:
l'addition est l'opération
arithmétique la plus commune.

Rappels sur l'écriture des entiers en base 2

Digital

Entiers positifs

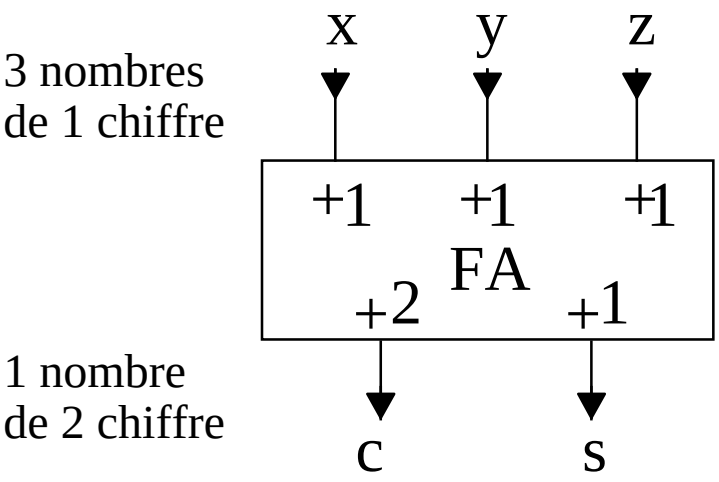
$$A = \sum_{i=0}^{n-1} a_i \, 2^i \qquad a_i \in \{ 0 , 1 \} \qquad A \in [0, 2^n - 1]$$

Notation de position imitée de la notation décimale adoptée en Europe au XI^{ème} siècle. La valeur d'un nombre est la somme pondérée de ses chiffres.

Entiers relatifs

$$B = -b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i \, 2^i \qquad b_i \in \{ 0 , 1 \} \qquad B \in [-2^{n-1}, +2^{n-1} - 1]$$

Fonction "Full Adder" (FA)



x	y	z	Σ	c	s
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

$x, y, z, c, s \in \{0 , 1\}$

$x + y + z \equiv 2*c + s$

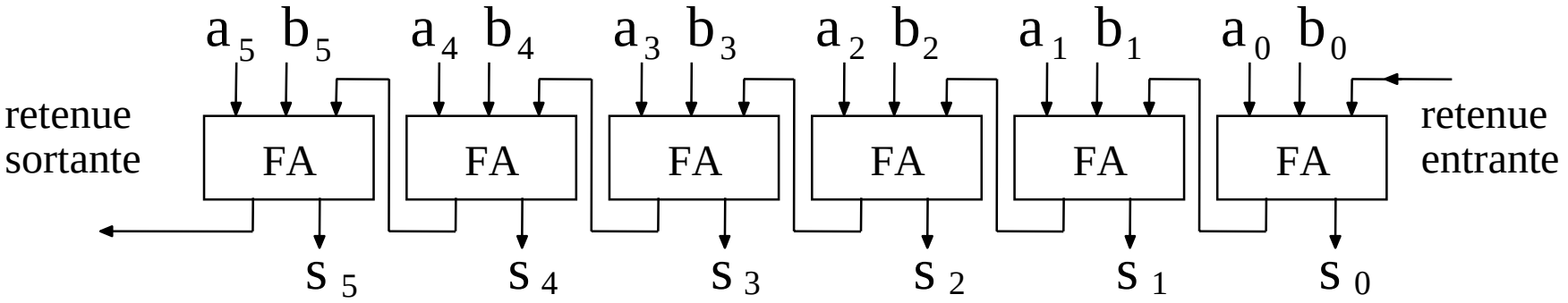
$s = x \oplus y \oplus z$ *somme modulo 2*

$c = \text{majorité}(x,y,z) = x \wedge y \vee x \wedge z \vee y \wedge z$

La somme pondérée de ce qui sort du “FA”est égale à la somme pondérée de ce qui entre dans le “FA”

Addition de Nombres ≥ 0

$$A = \sum_{i=0}^{5} a_i 2^i \qquad B = \sum_{i=0}^{5} b_i 2^i \qquad S = \sum_{i=0}^{5} s_i 2^i \qquad a_i, b_i, s_i \in \{0 , 1\}$$



Tout assemblage cohérent de “FA” conserve la propriété: La somme pondérée de ce qui sort est égale à la somme pondérée de ce qui entre

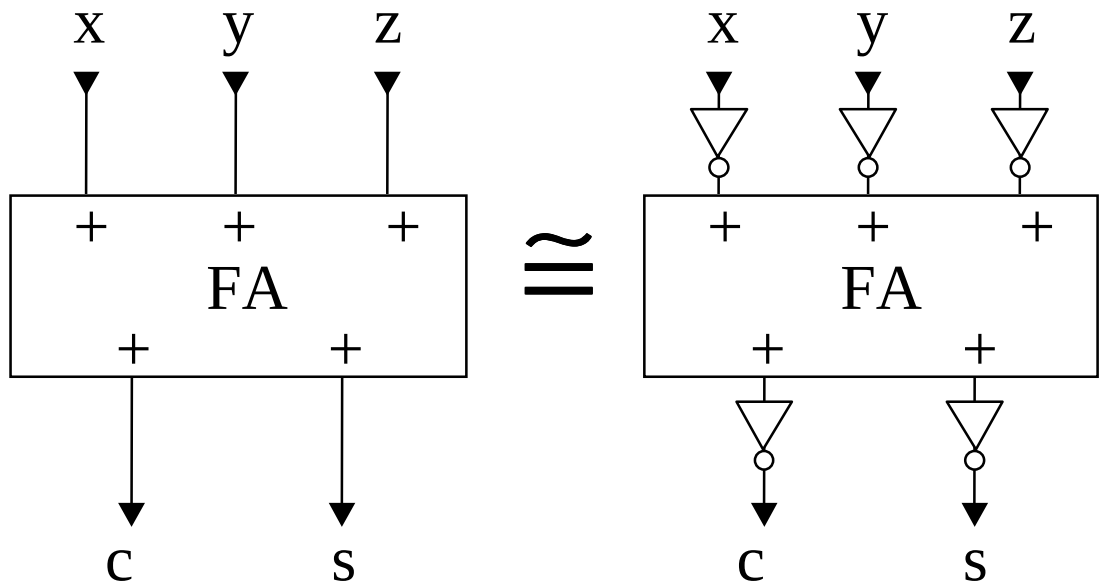
Si on ignore la retenue sortante:

$$S = \left(A + B + \text{retenue entrante} \right)_{\text{modulo } 2^6}$$

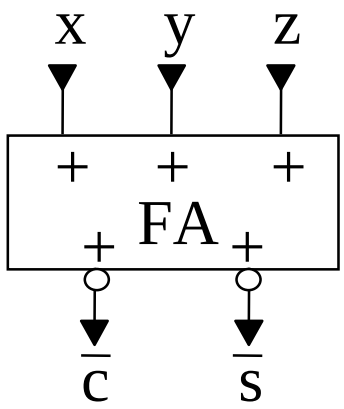
Dans ce cas l’opérateur accepte 2 conventions.

Le "Full Adder" est Autodual
(propriété générale des additionneurs)

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



"Full Adder" (FA) symétrique

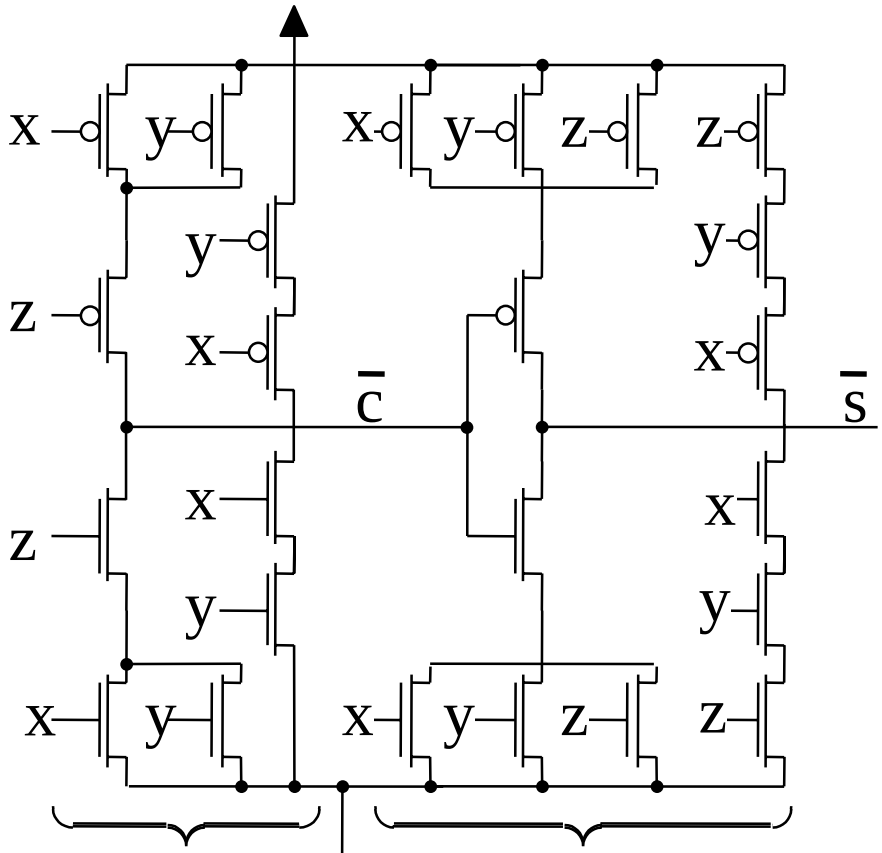


c = majorité (x,y,z)

s = **si** c **alors** (x∧y∧z) **sinon** (x∨y∨z)

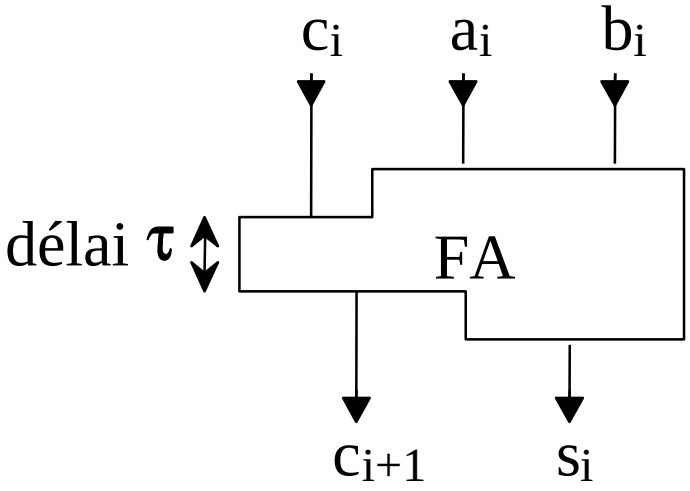
s = (x∧y∧z) ∨ $\overline{c}$∧(x∨y∨z)

	∧	M	∨	c	s
000	0	0	0	0	0
001	0	0	1	0	1
011	0	1	1	1	0
111	1	1	1	1	1



Circuit « miroir »

"Full Adder" (FA) dissymétrique



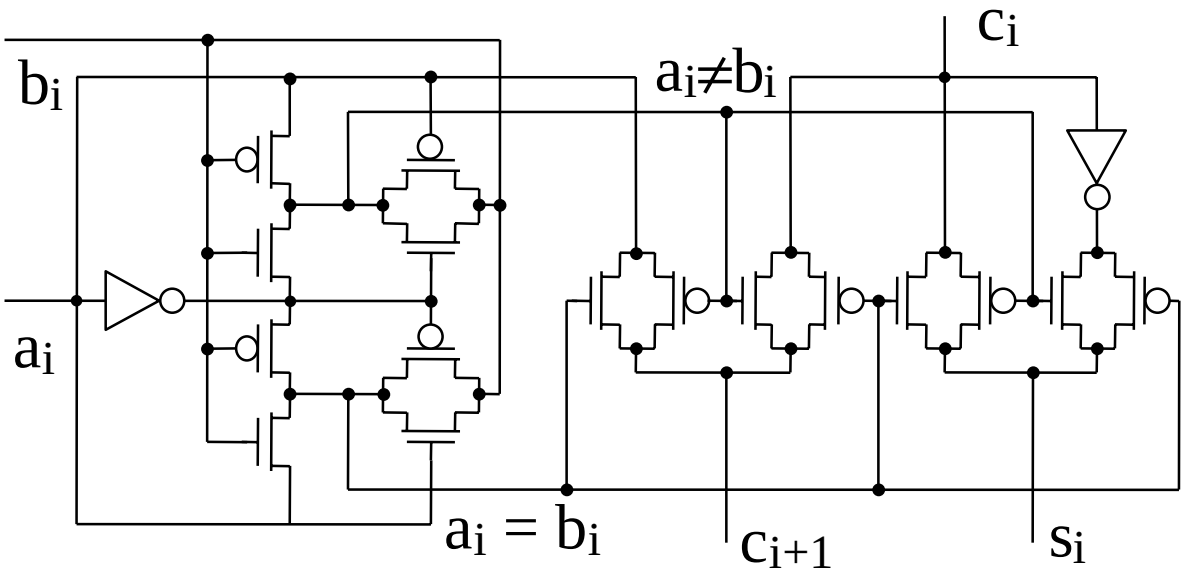
minimiser le délai τ entre
c_i et c_{i+1} (au dépens des autres)

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

a_i	b_i	c_{i+1}	s_i
0	0	0	c_i
0	1	c_i	$\overline{c_i}$
1	0	c_i	$\overline{c_i}$
1	1	1	c_i

a_i	b_i	c_{i+1}	s_i
$a_i = b_i$		a_i	c_i
$a_i \neq b_i$		c_i	$\overline{c_i}$

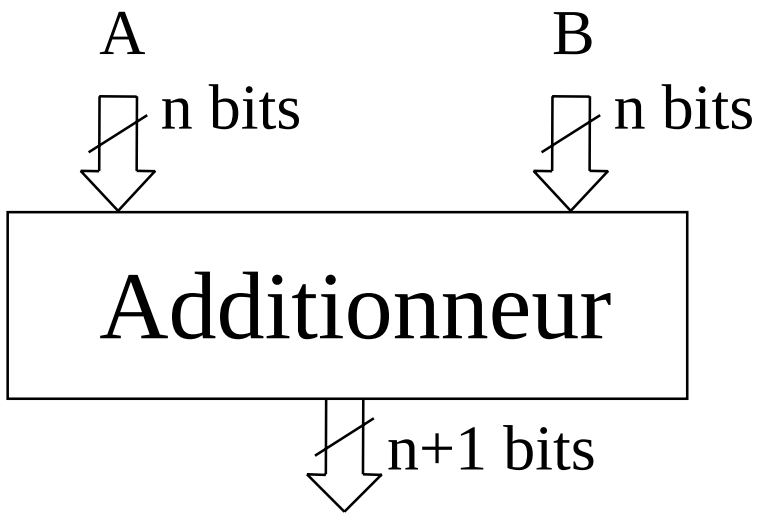
"Full Adder" (FA) à porte de transmission



si $a_i = b_i$ **alors** $c_{i+1} := a_i$, $s_i := \overline{c_i}$

si $a_i \neq b_i$ **alors** $c_{i+1} := c_i$, $s_i := \overline{c_i}$

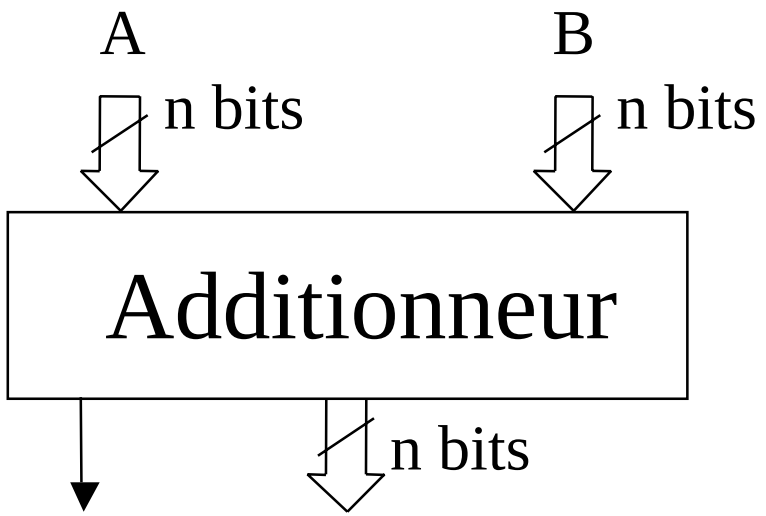
Débordement d'entiers positifs



$S = A + B$

La somme pondérée de ce qui sort est égale à la somme pondérée de ce qui entre

Solution 1



débordement
$S \geq 2^n$, ne tient pas sur n bits

$S \in [0, 2^n - 1]$

$S = (A + B) \text{ modulo } 2^n$

La somme pondérée de ce qui sort n'est pas égale à la somme pondérée de ce qui entre

Solution 2

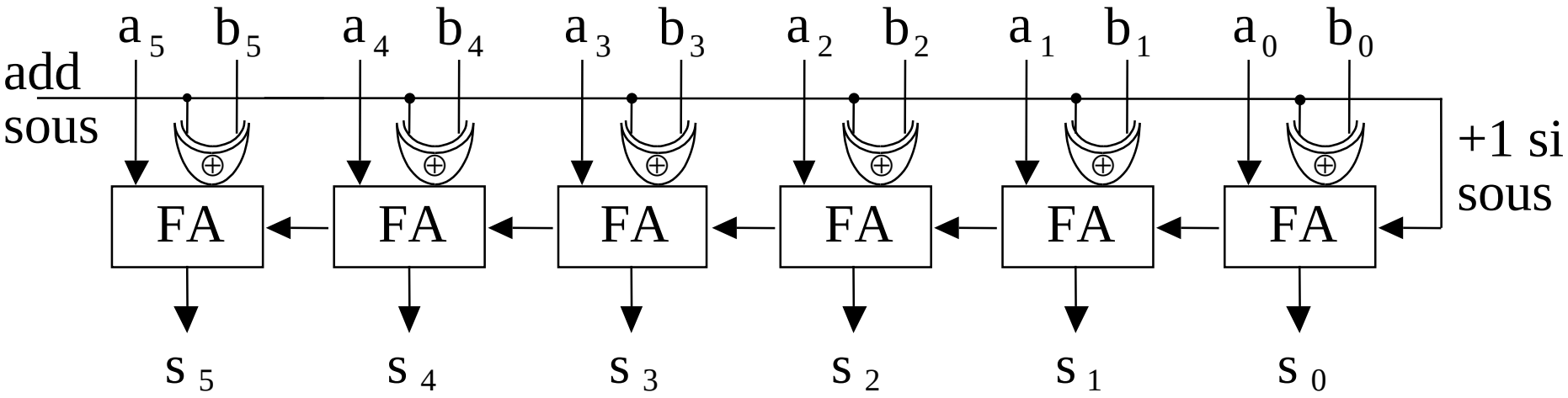
Entiers relatifs

Définition de l'opposé à partir de l'additionneur modulo 2^n

$$B \cong -A \leftrightarrow (A + B) = 0 \ (2^n_{\bmod 2^n})$$

$$A + \overline{A} = \sum_{i=0}^{n-1} 2^i = 2^n - 1 \qquad A + (\overline{A} + 1) = 2^n = 0 \qquad -A = (\overline{A} + 1)$$

Additionneur/soustracteur



$$S = A \pm B$$

Notation des entiers relatifs

a_3	a_2	a_1	a_0	A		
0	0	0	0	0	0	-0+0
0	0	0	1	+1	+1	-0+1
0	0	1	0	+2	+2	-0+2
0	0	1	1	+3	+3	-0+3
0	1	0	0	+4	+4	-0+4
0	1	0	1	+5	+5	-0+5
0	1	1	0	+6	+6	-0+6
0	1	1	1	+7	+7	-0+7
1	0	0	0	-8	-8	-8+0
1	0	0	1	-7	-7	-8+1
1	0	1	0	-6	-6	-8+2
1	0	1	1	-5	-5	-8+3
1	1	0	0	-4	-4	-8+4
1	1	0	1	-3	-3	-8+5
1	1	1	0	-2	-2	-8+6
1	1	1	1	-1	-1	-8+7

est son propre opposé

n'a pas d'opposé

(débordement détectable)

$-A = (\overline{A} + 1)$

$$A = -a_3 2^3 + \sum_{i=0}^{2} a_i 2^i$$

Notation en complément à 2

Dite en complément à 2, en fait à 2^n

Le bit poids fort est négatif, les autres positifs

La valeur d'un nombre est la somme pondérée de ses bits

Le bit poids fort indique le signe du nombre ($0 \Leftrightarrow \geq 0$, $1 \Leftrightarrow < 0$)

Alors 0 est positif

Le plus grand nombre négatif n'a pas d'opposé

Le changement de signe provoque une propagation de retenue

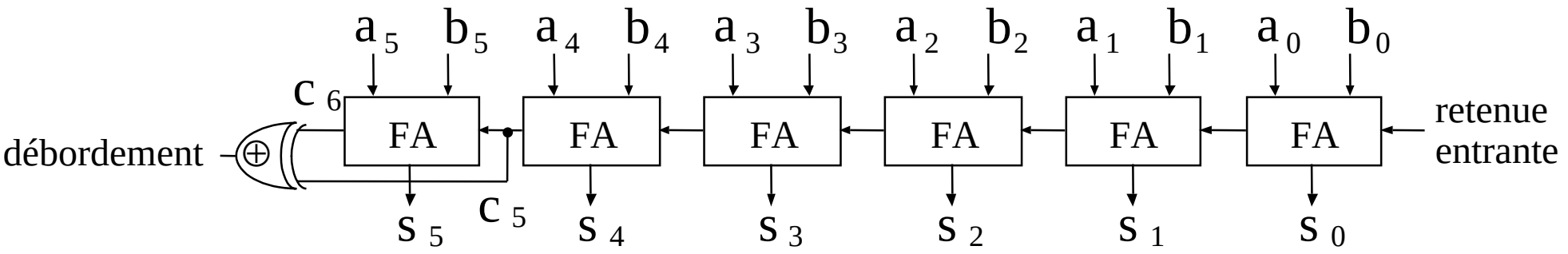
Il y a d'autres systèmes (peu usités)

Débordement de l'addition d'entiers relatifs

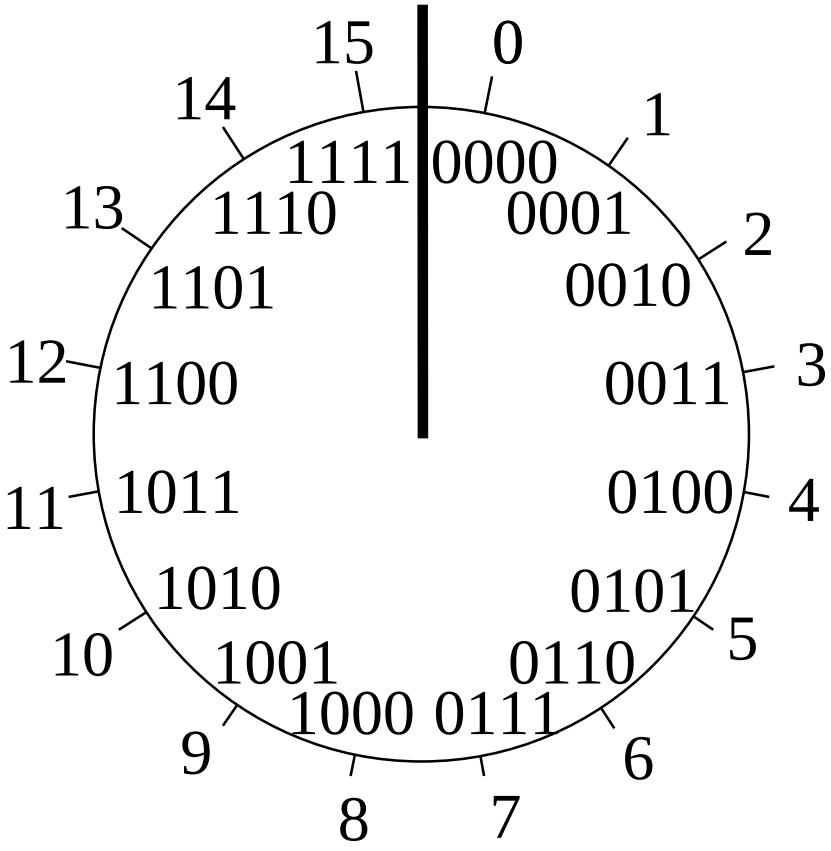
A	B	S=A+B
+	+	+
+	+	-
+	-	+
+	-	-
-	+	+
-	+	-
-	-	+
-	-	-

cas de débordement

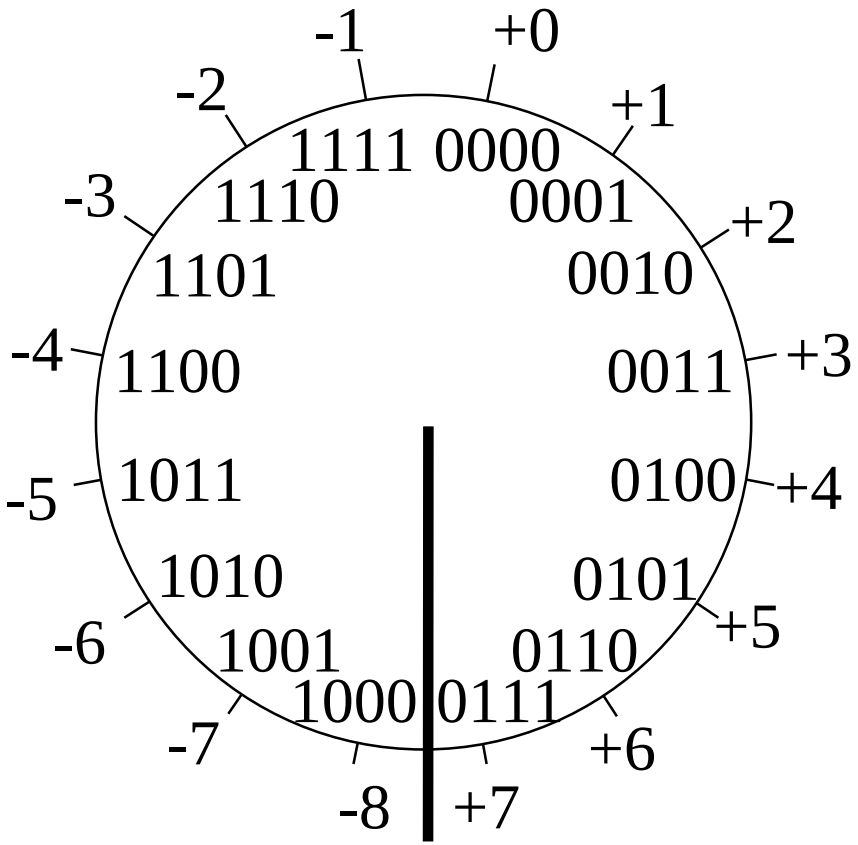
a_{n-1}	b_{n-1}	c_{n-1}		c_n	s_{n-1}
+	+	0	=	0	+
+	+	1	≠	0	-
+	-	0	=	0	-
+	-	1	=	1	+
-	+	0	=	0	-
-	+	1	=	1	+
-	-	0	≠	1	+
-	-	1	=	1	-



Débordement (exemple sur 4 bits)



Entiers positifs



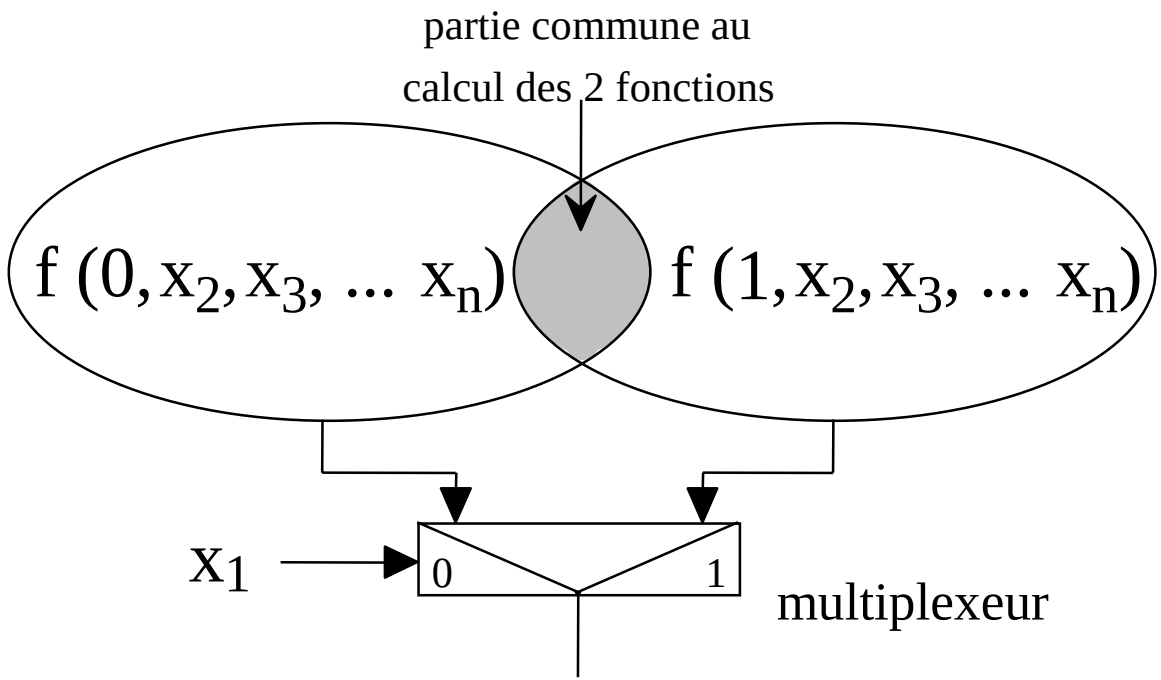
Entiers relatifs

Écriture de Shannon

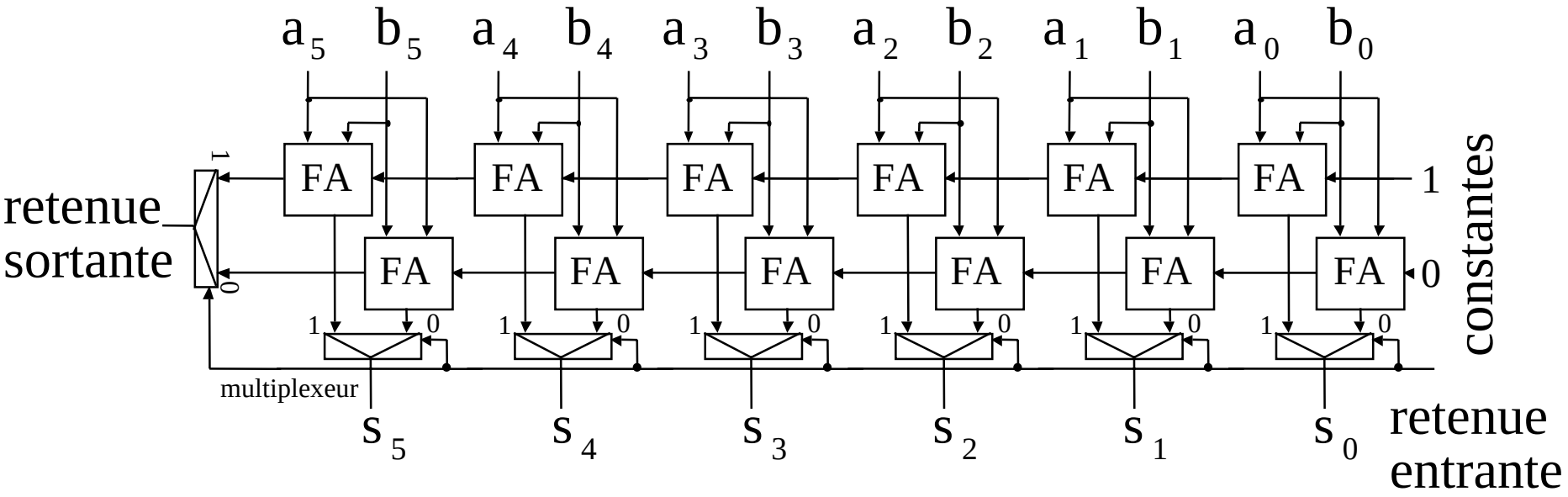
On a à réaliser $f(x_1, x_2, x_3, \ldots x_n)$

on veut disposer de temps pour calculer x_1

$\Rightarrow$ on précalcule $f(0, x_2, x_3, \ldots x_n)$ et $f(1, x_2, x_3, \ldots x_n)$



“Carry select adder”

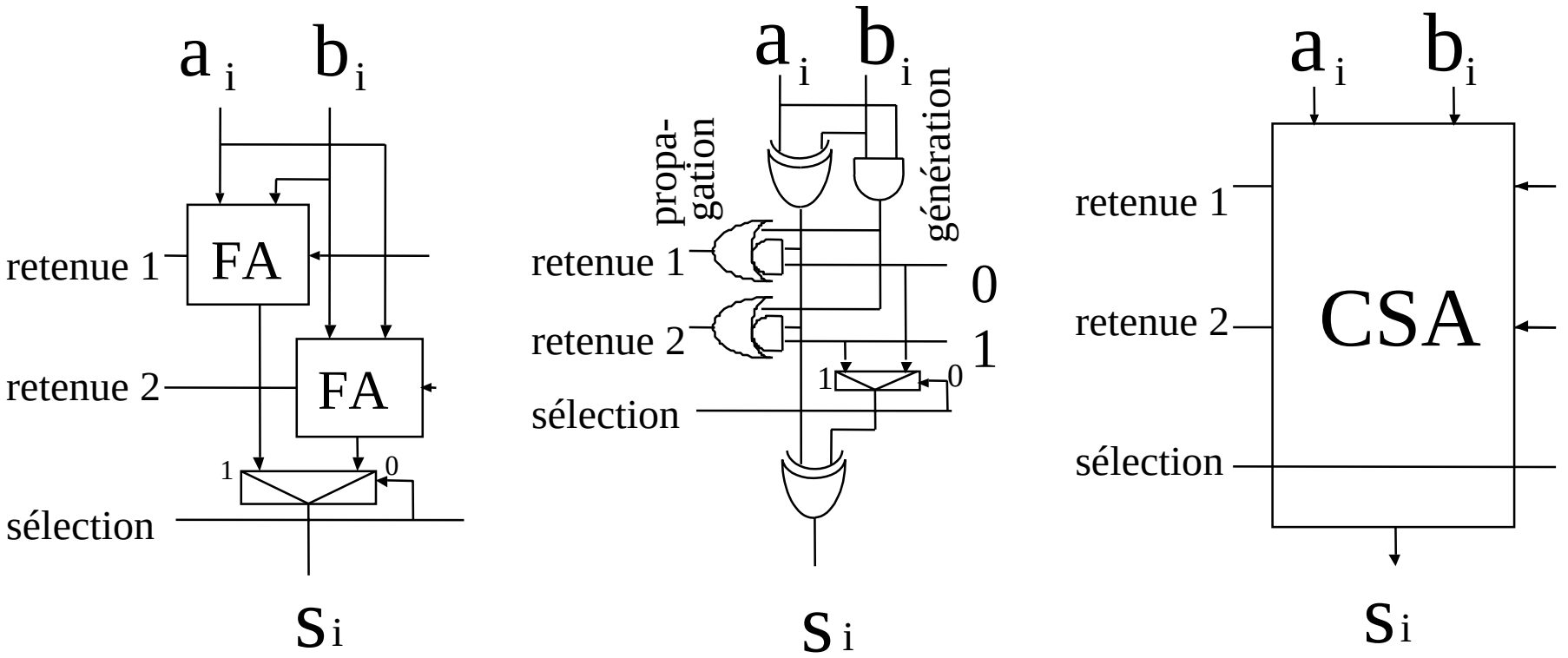


La retenue entrante ne se propage pas à travers les FA

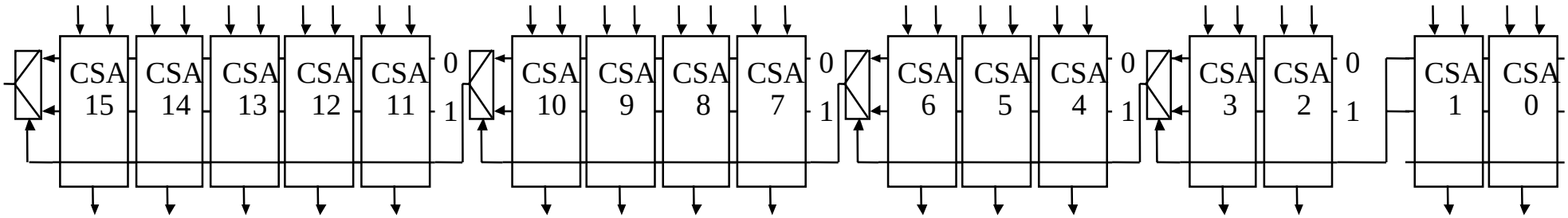
⇒ on dispose de temps pour la calculer

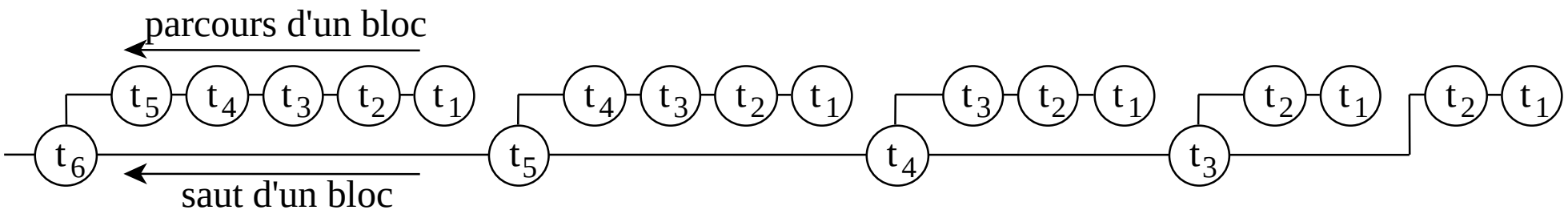
⇒ à mettre en poids forts la où la retenue est en retard

Cellule de carry select adder



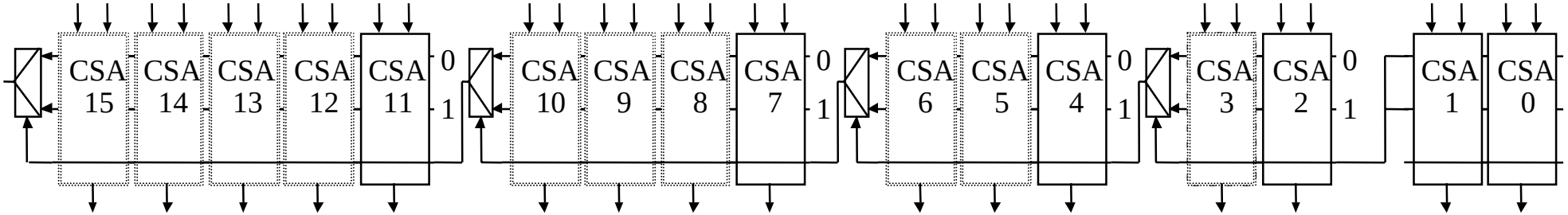
Additionneur en temps $\sqrt{\ }\overline{n}$

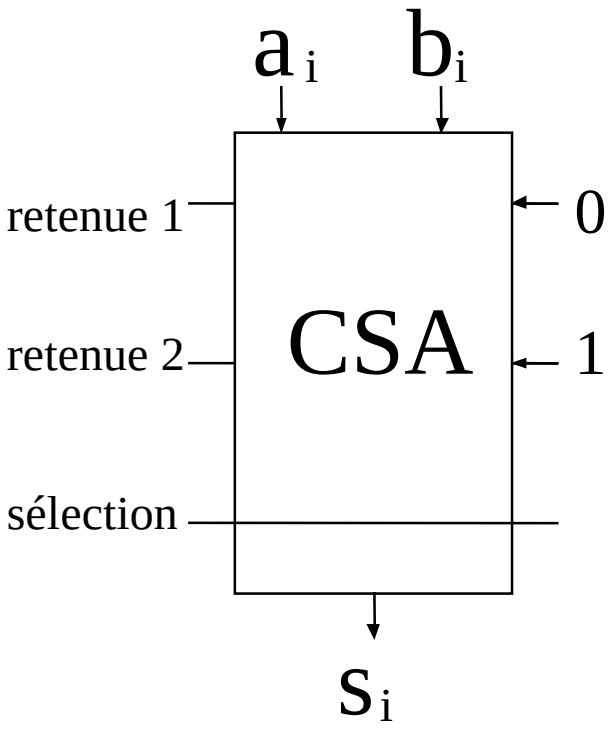


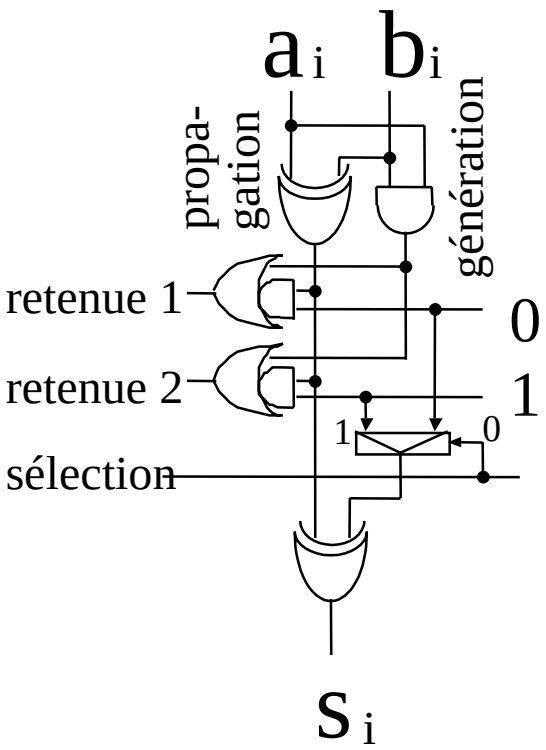


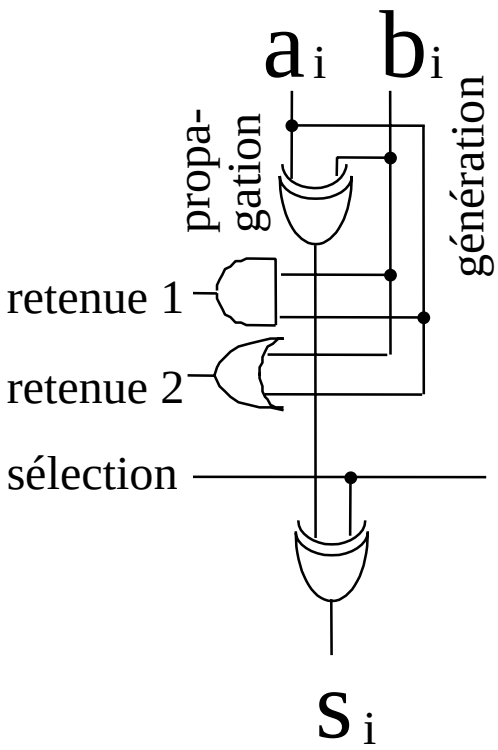
$$n = \sum_{i=0}^{\tau-2} i = \frac{\tau(\tau+1)}{2} + 1 \qquad 2n = \tau^2 + \tau + 2 \Rightarrow \tau = \frac{\sqrt{8n-7}-1}{2} \approx \sqrt{2n} \approx 1{,}4\sqrt{n}$$

Première cellule d'un bloc de CSA









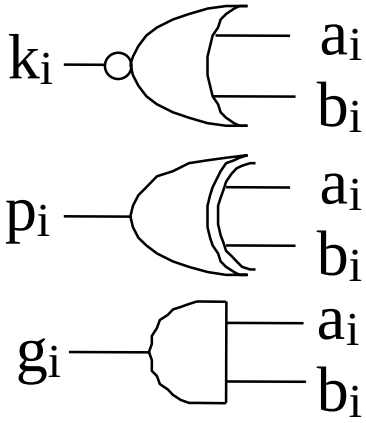
Génération et propagation de la retenue

a_i	b_i	c_{i+1}
0	0	0
0	1	c_i
1	0	c_i
1	1	1

k_i (Absorption) $k_i = \overline{a_i \vee b_i}$

p_i (Propagation) $p_i = a_i \oplus b_i$

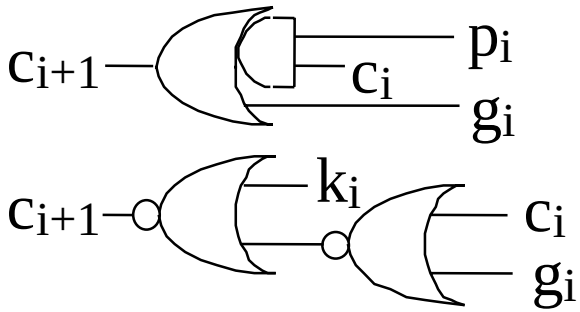
g_i (Génération) $g_i = a_i \wedge b_i$



A	1	0	1	0	0	0	1	0	1	0	1	0	1	1	0	0
+B	0	1	1	0	0	1	0	1	1	0	1	0	0	0	1	0
	p	p	g	k	k	p	p	p	g	k	g	k	p	p	p	k
C	1	1	1	0	0	1	1	1	1	0	1	0	0	0	0	

$c_{i+1} = (p_i \wedge c_i) \vee g_i = (\overline{k_i} \wedge c_i) \vee g_i$

$\overline{c_{i+1}} = (p_i \wedge \overline{c_i}) \vee k_i = (g_i \wedge \overline{c_i}) \vee k_i$



Génération et propagation de la retenue (2)

Définitions

$p_i = a_i \oplus b_i$ propagation: $c_{i+1} = c_i$

$g_i = a_i \wedge b_i$ génération: $c_{i+1} = 1$

$k_i = \overline{a_i} \wedge \overline{b_i}$ absorption: $c_{i+1} = 0$

$\mathbf{P}_{i,j} = \prod_{n=i}^{j} p_n$ propagation: $c_{i+1} = c_j \quad i \geq j$

$\mathbf{G}_{i,j} = g_i \vee \sum_{n=i+1}^{j} (\mathbf{P}_{i,n} \wedge g_n)$ $c_{i+1} = 1$

$\mathbf{K}_{i,j} = k_i \vee \sum_{n=i+1}^{j} (\mathbf{P}_{i,n} \wedge k_n)$ $c_{i+1} = 0$

$\mathbf{P}_{i,j}$ = la retenue a été propagée du rang j au rang i+1

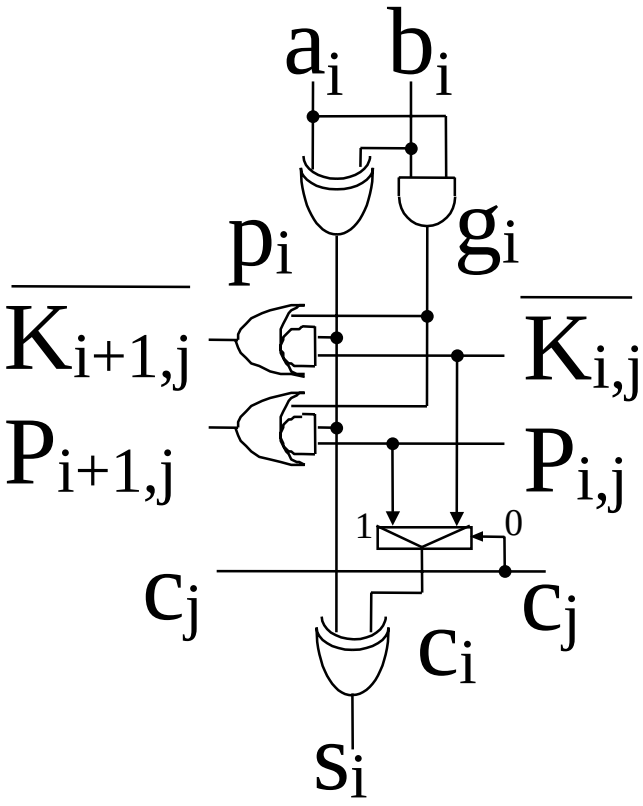
$\mathbf{G}_{i,j}$ = la retenue a été générée entre i et j et propagée jusqu'à i+1

$\mathbf{K}_{i,j}$ = la retenue a été absorbée entre i et j et propagée jusqu'à i+1

Exercice

Simplification du CSA

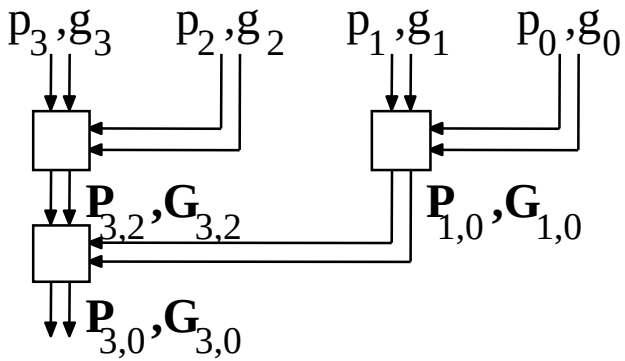
1- Simplifier les équations logiques de la cellule de “Carry-Select Adder” en exprimant la relation entre $P_{i,j}$ et $K_{i,j}$

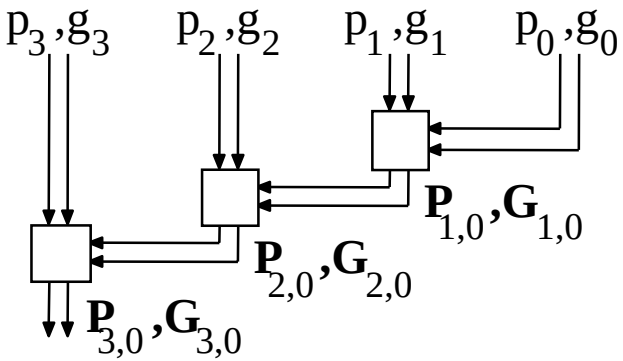


Génération et propagation de groupe

Définissons g_i, p_i, $\mathbf{G}_{i,j}$ et $\mathbf{P}_{i,j}$ de la façon récursive suivante :

$\mathbf{G}_{i,i} = g_i = a_i \wedge b_i$	*génération de retenue au rang i*
$\mathbf{P}_{i,i} = p_i = a_i \oplus b_i$	*propagation de retenue au rang i*
$\mathbf{G}_{i,k} = \mathbf{G}_{i,j} \vee \mathbf{P}_{i,j} \wedge \mathbf{G}_{j-1,k}$	*génération de retenue entre le rang i et le rang k* $(n \geq i \geq j > k \geq 0)$
$\mathbf{P}_{i,k} = \mathbf{P}_{i,j} \wedge \mathbf{P}_{j-1,k}$	*propagation de la retenue du rang k au rang i*
$c_{i+1} = \mathbf{G}_{i,0} \vee \mathbf{P}_{i,0} \wedge c_0$	*retenue au rang i+1, ce que l'on cherche à obtenir*



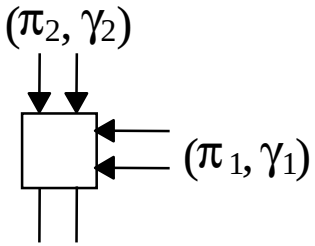


Propriété: La cellule est associative.

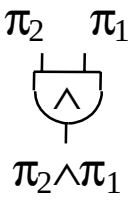
Conséquence: De nombreux assemblages sont possibles

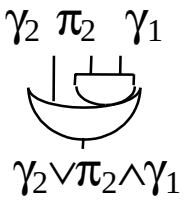
Règle d'assemblage: Toute sortie de rang i dépends des entrées de rang 0 à i.

Cellule de Brent et Kung pour calculer les "propagation de groupe" et "génération de groupe"

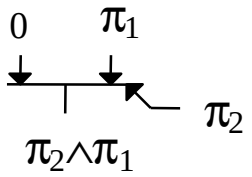


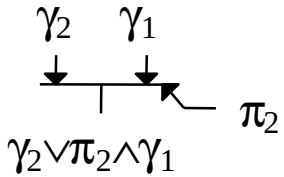
$(\pi_2 \wedge \pi_1 , \gamma_2 \vee \pi_2 \wedge \gamma_1)$





Réalisation à portes logiques

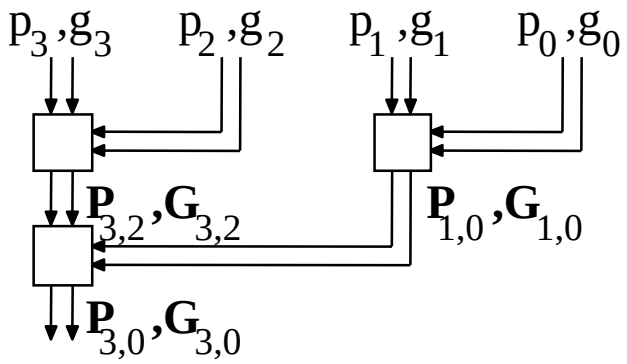


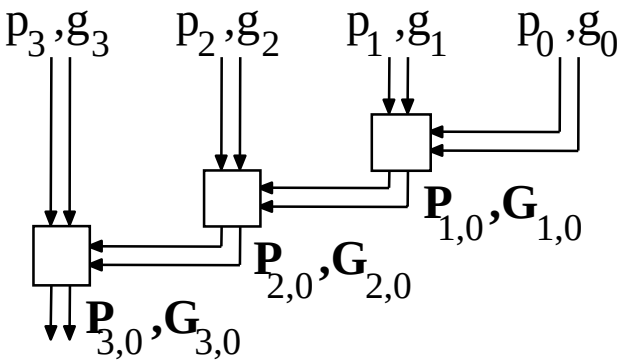


Réalisation à multiplexeurs

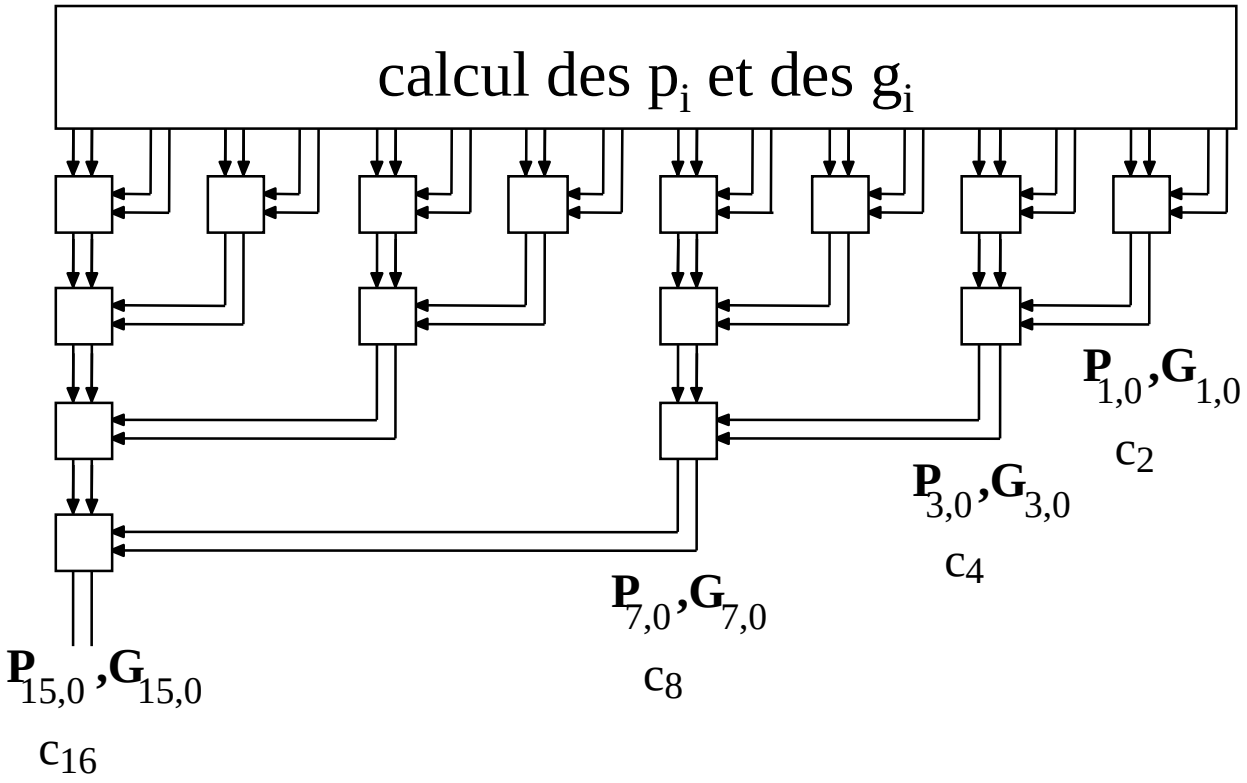
- <u>Associative</u>
- Non commutative
- Idempotente
- Non décroissante (inverseurs)

$$\mathbf{G}_{i,k} = \mathbf{G}_{i,j} \vee \mathbf{P}_{i,j} \wedge \mathbf{G}_{j-1,k}$$
$$\mathbf{P}_{i,k} = \mathbf{P}_{i,j} \wedge \mathbf{P}_{j-1,k}$$





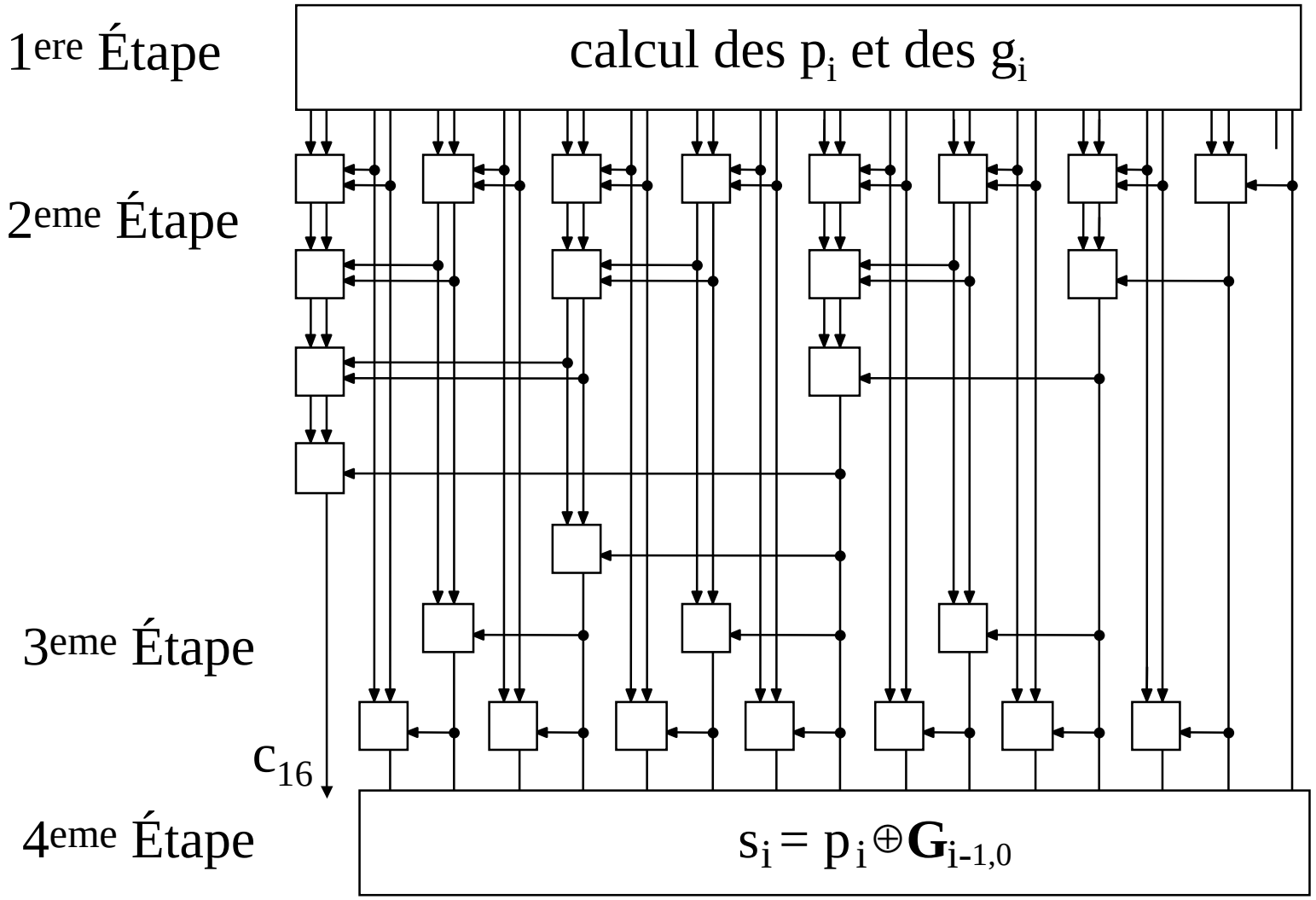
Tous les additionneurs en temps $\log_2(n)$ commence par un calcul arborescent



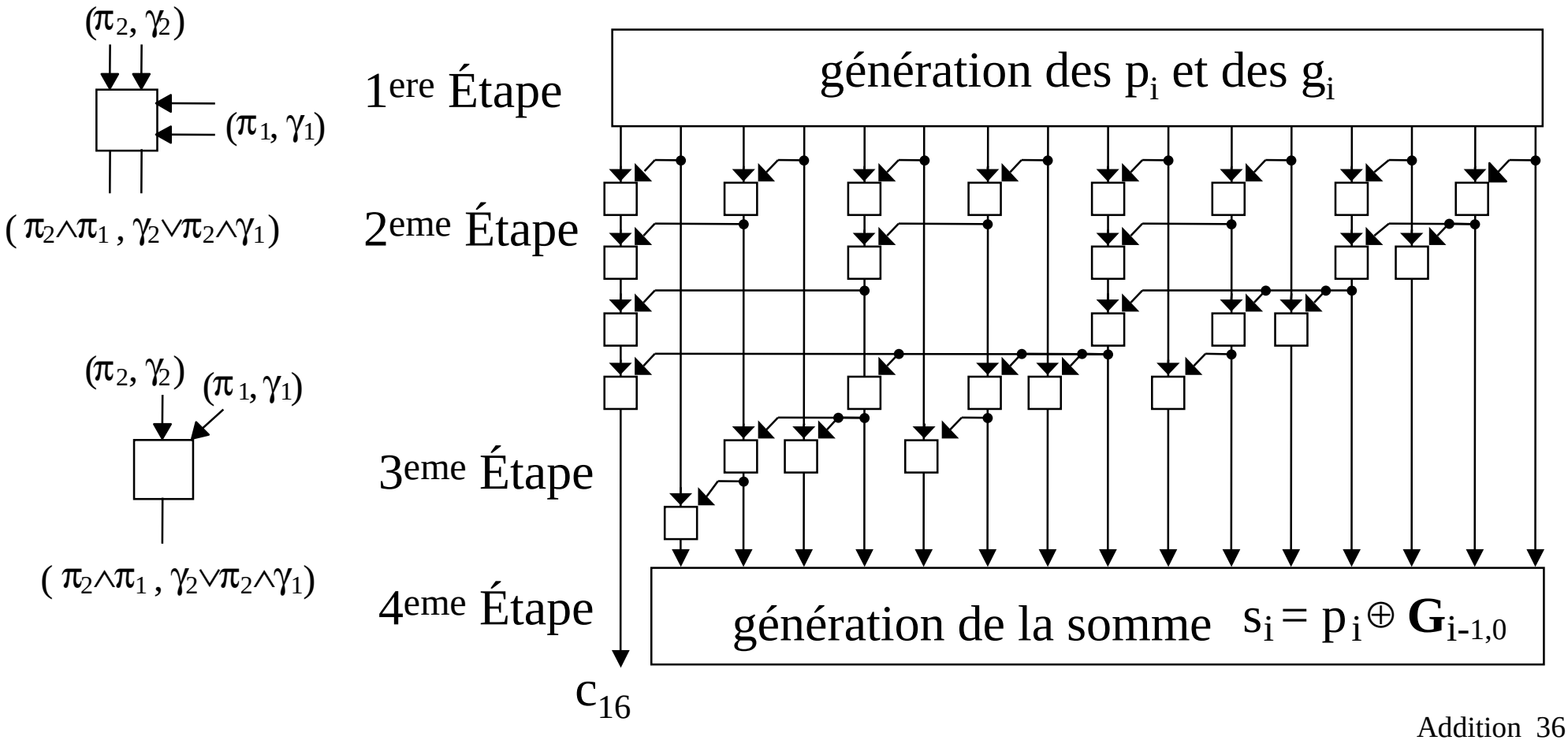
Le calcul des autres retenues est:

- 1 coût minimum
- 2 rapide à “fan out” variable
- 3 rapide à fan-out fixe

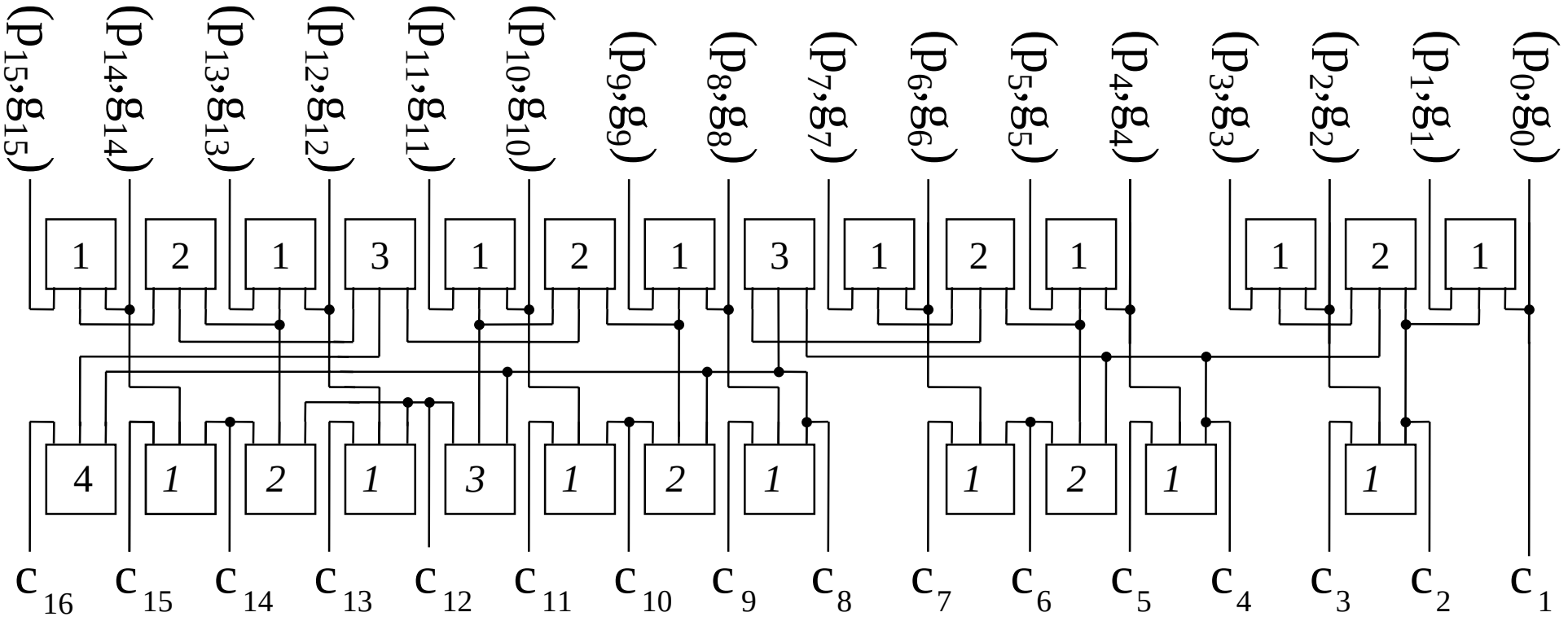
Additionneur de Brent et Kung en temps $\log_2(n)$



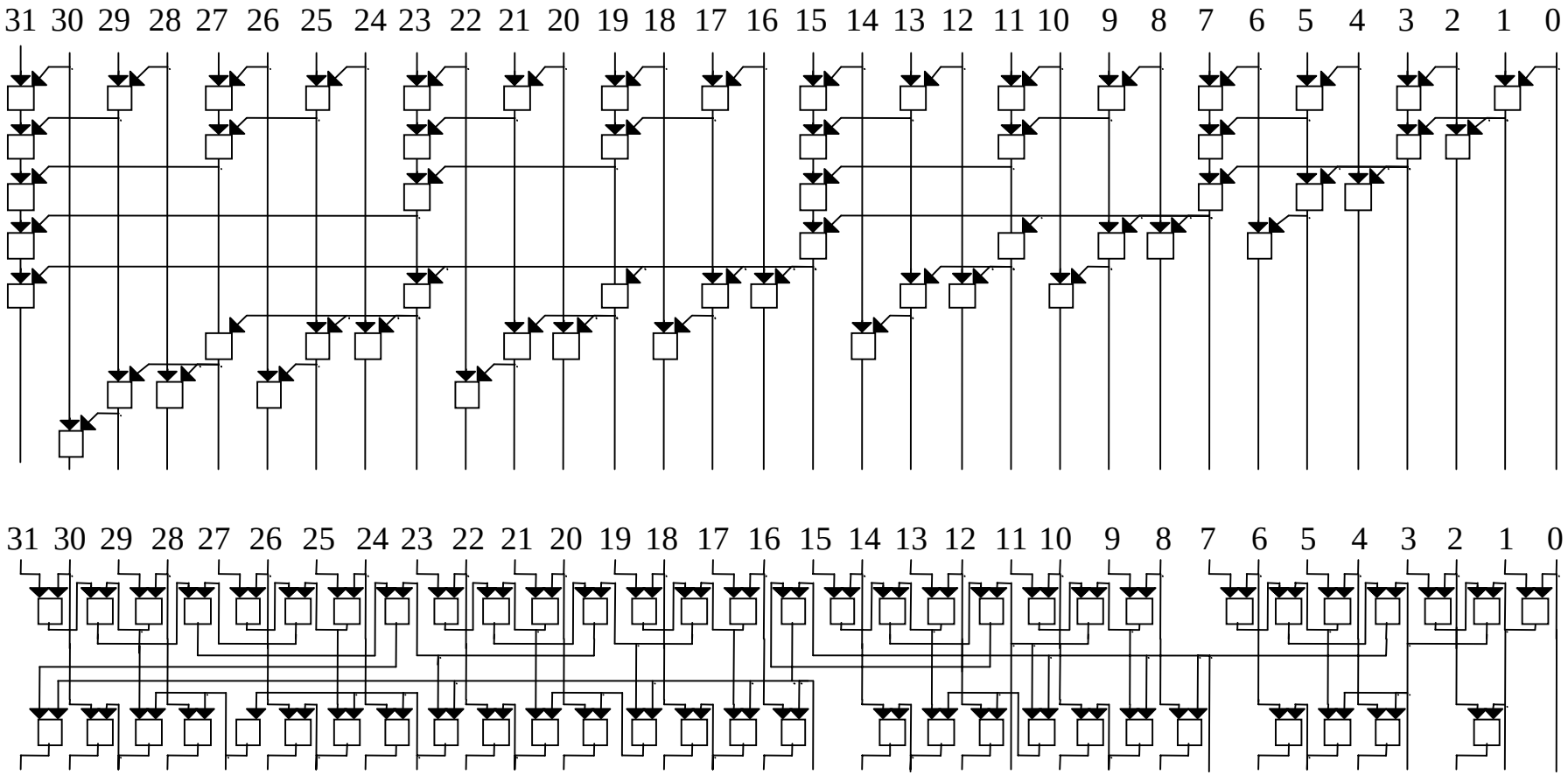
Additionneur de Brent et Kung en temps $\log_2(n)$ (2)



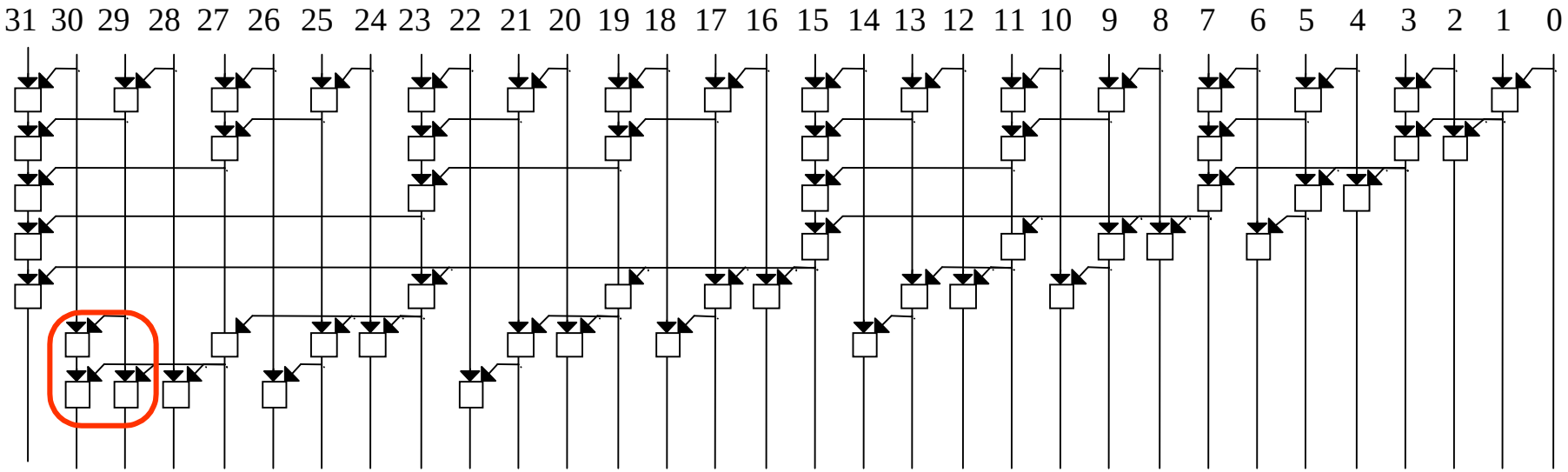
Mise à plat des deux arbres binaires (16 bits)



Additionneur de Brent et Kung



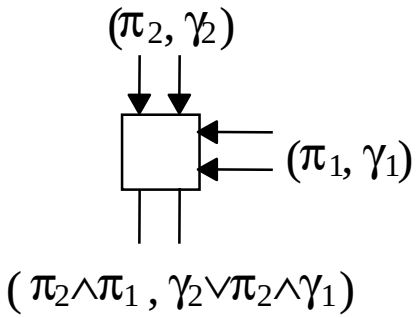
Additionneur de Brent et Kung modifié

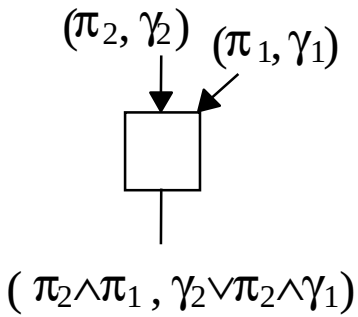


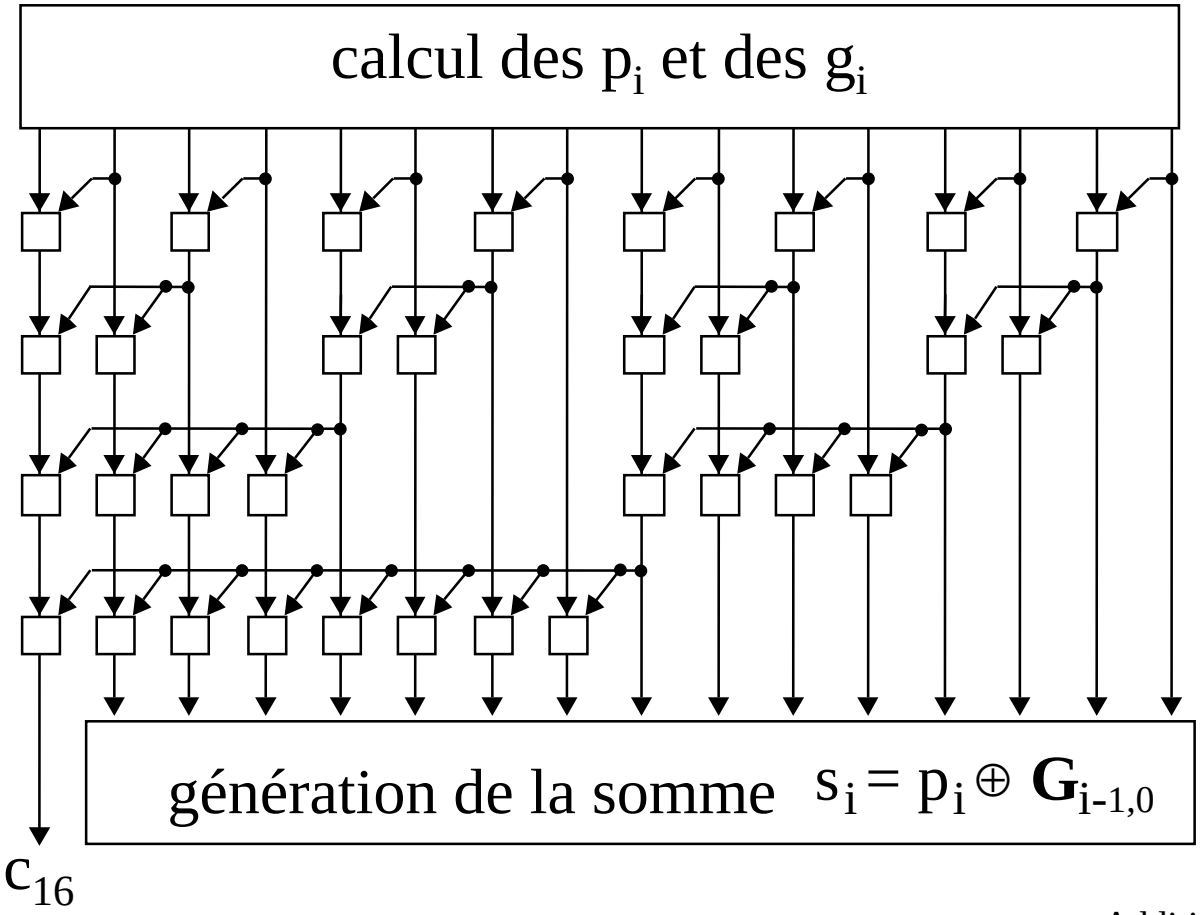
Modification de J.P. Fishburn (1990)

Une cellule de plus (+ 2%) décroît le chemin critique de 8 à 7 cellules

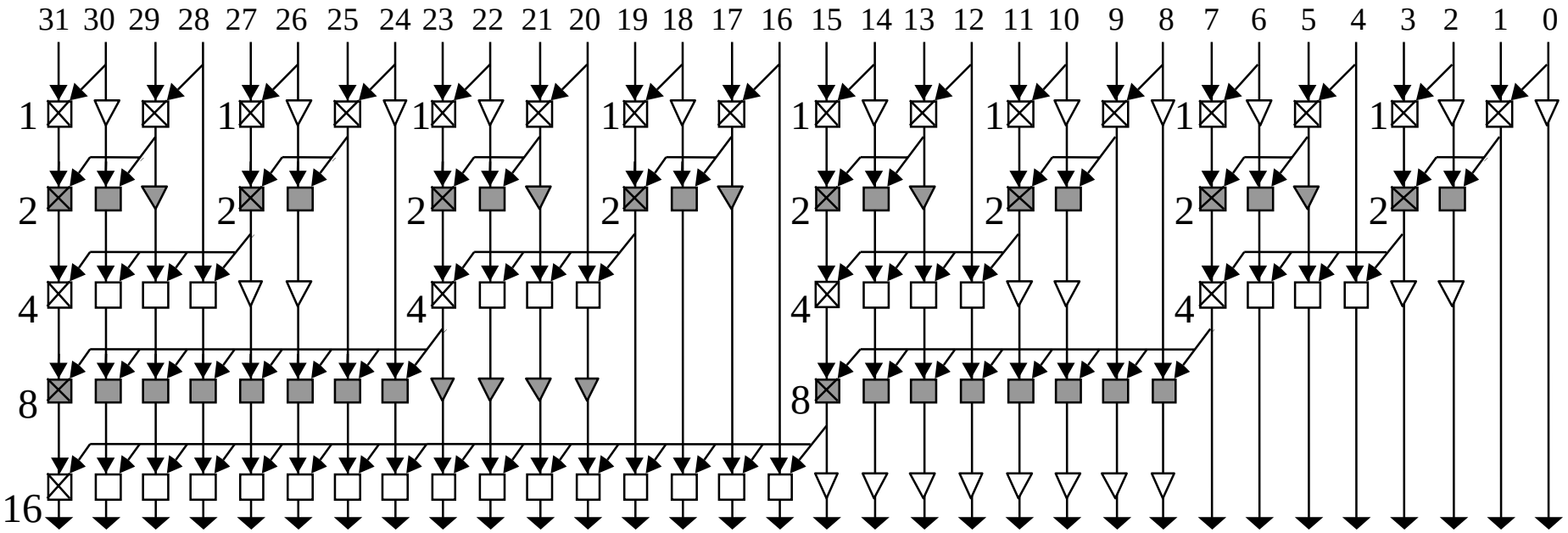
Additionneur de Sklansky en temps $\log_2(n)$

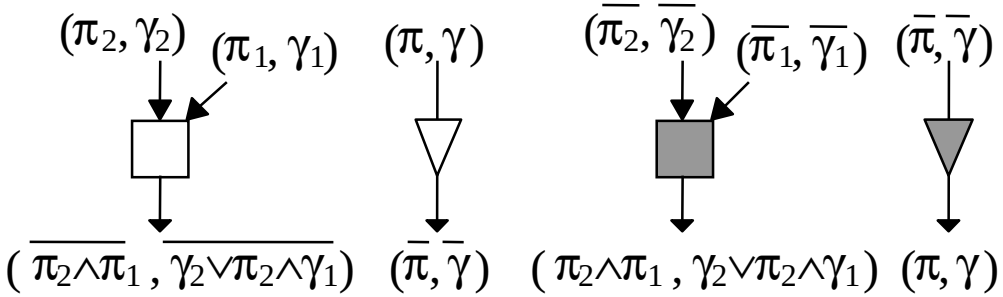




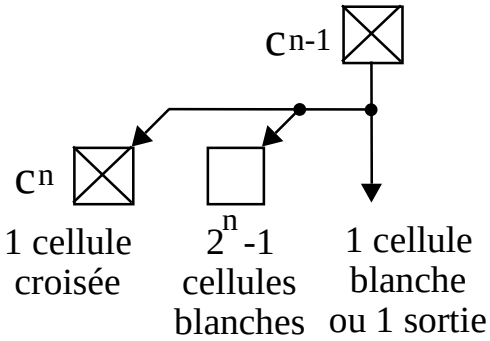


Prise en compte des aspects électriques





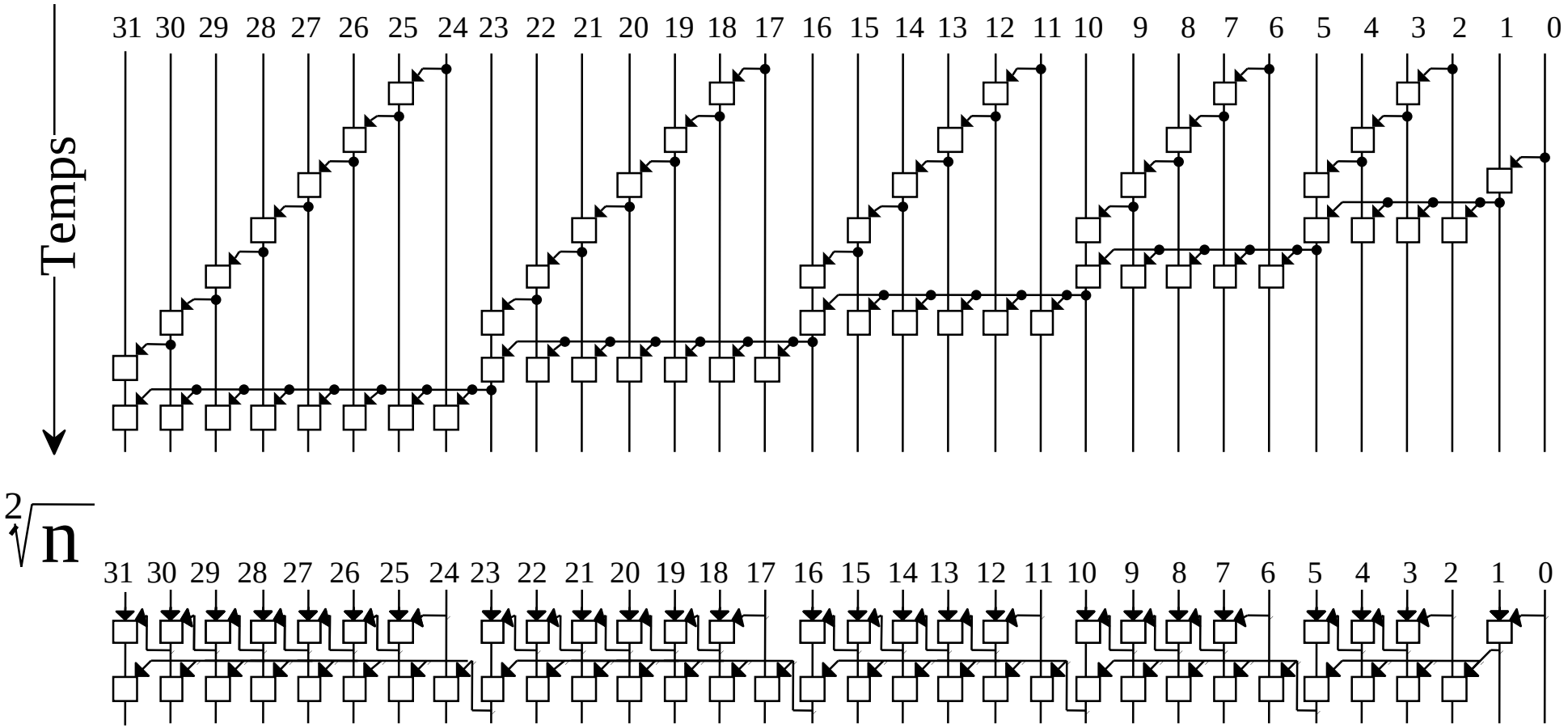
Prise en compte des inversions



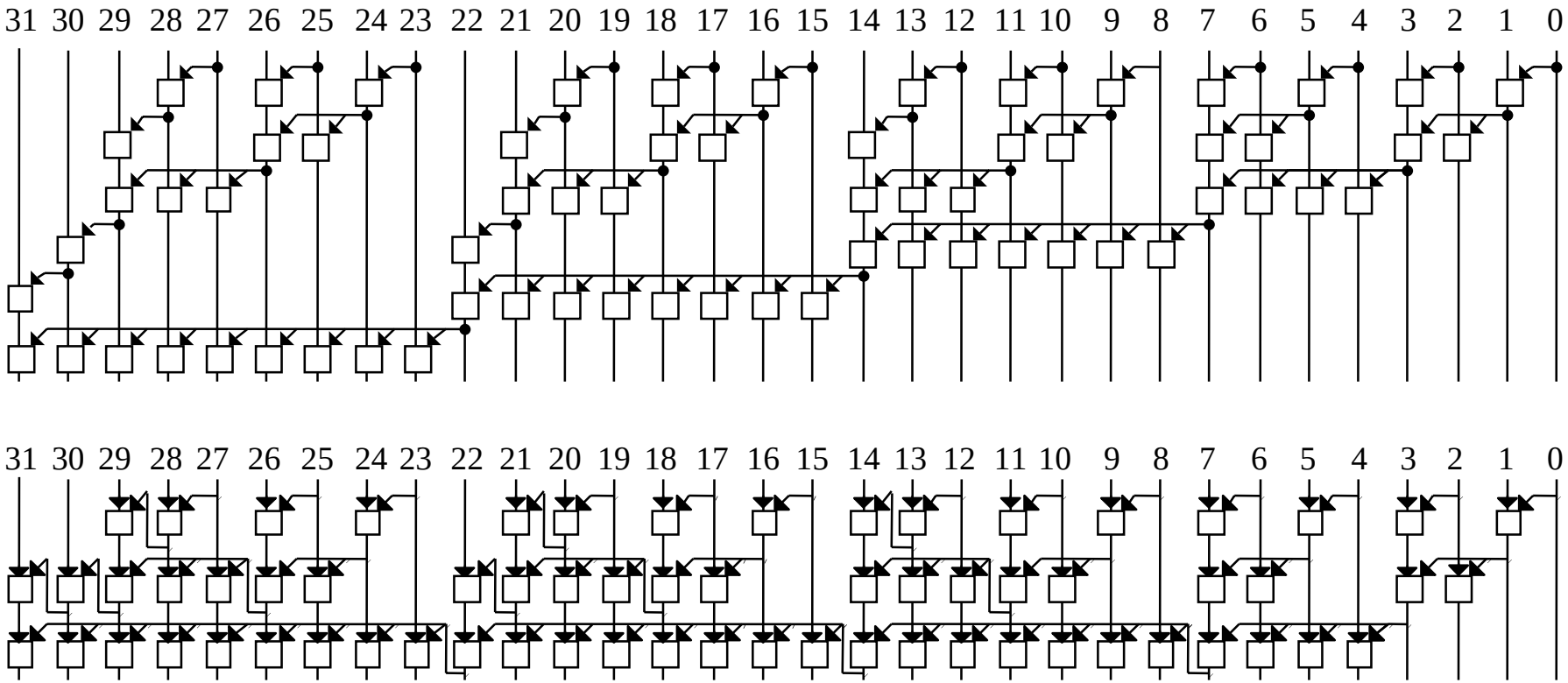
$C_{n-1} = k\,(C_n + 2^n - 1 + 1)$

dimensionnement

Additionneur en temps $\sqrt{n}$

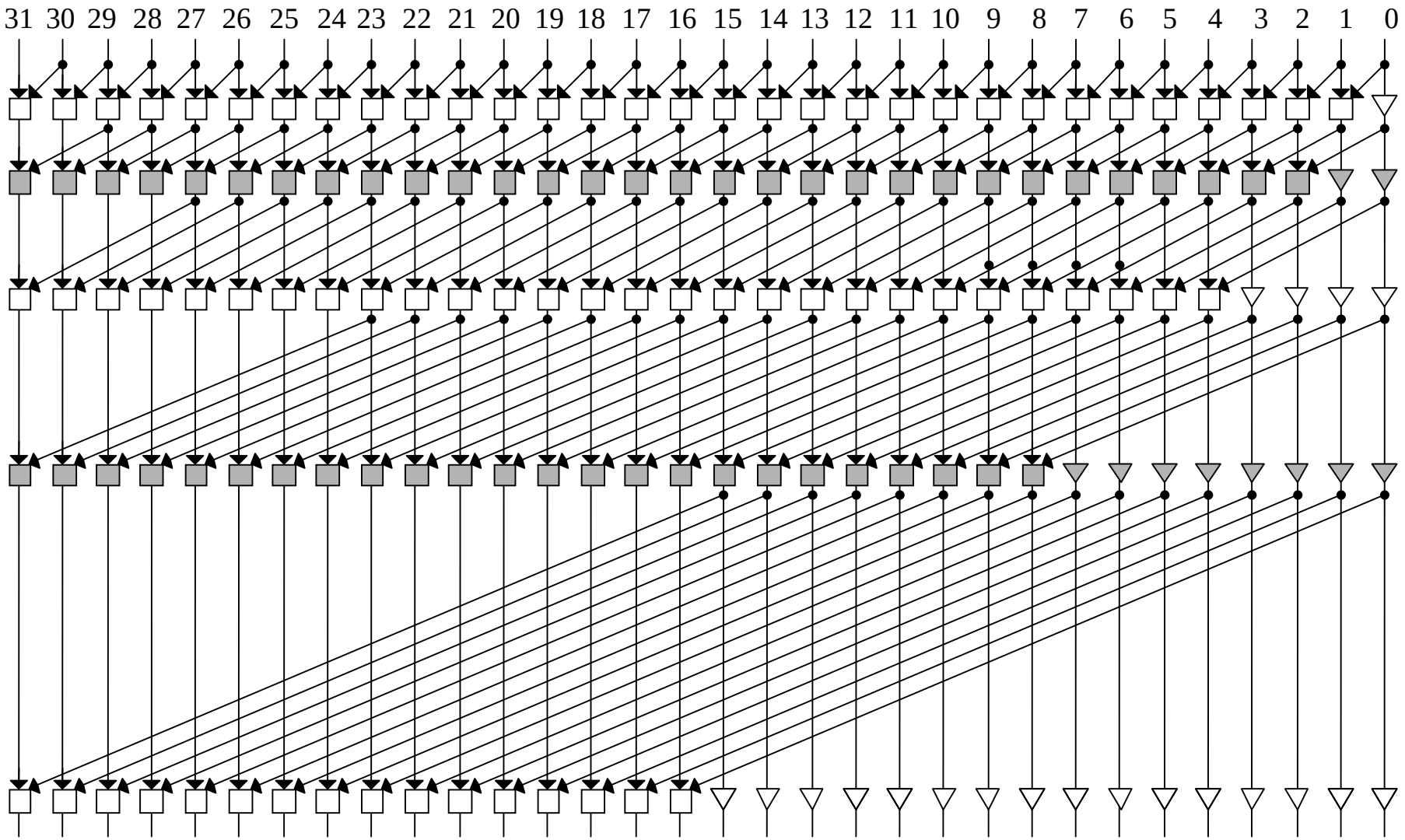


Additionneur en temps $\sqrt[3]{n}$



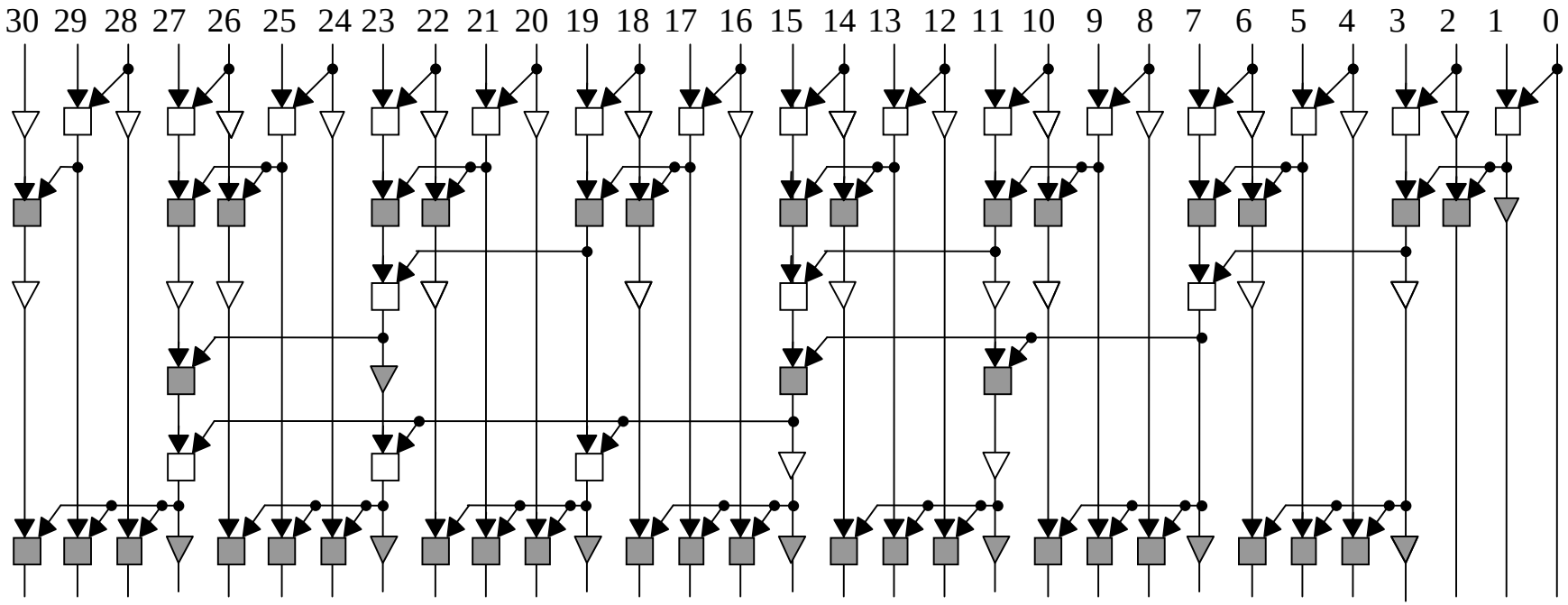
Pour un délai τ le nombre n de bits est : $\sum_{i=1}^{\tau} \sum_{j=1}^{i} j \Rightarrow \quad \tau = \sqrt[3]{n}$

Additionneur de Kogge et Stone



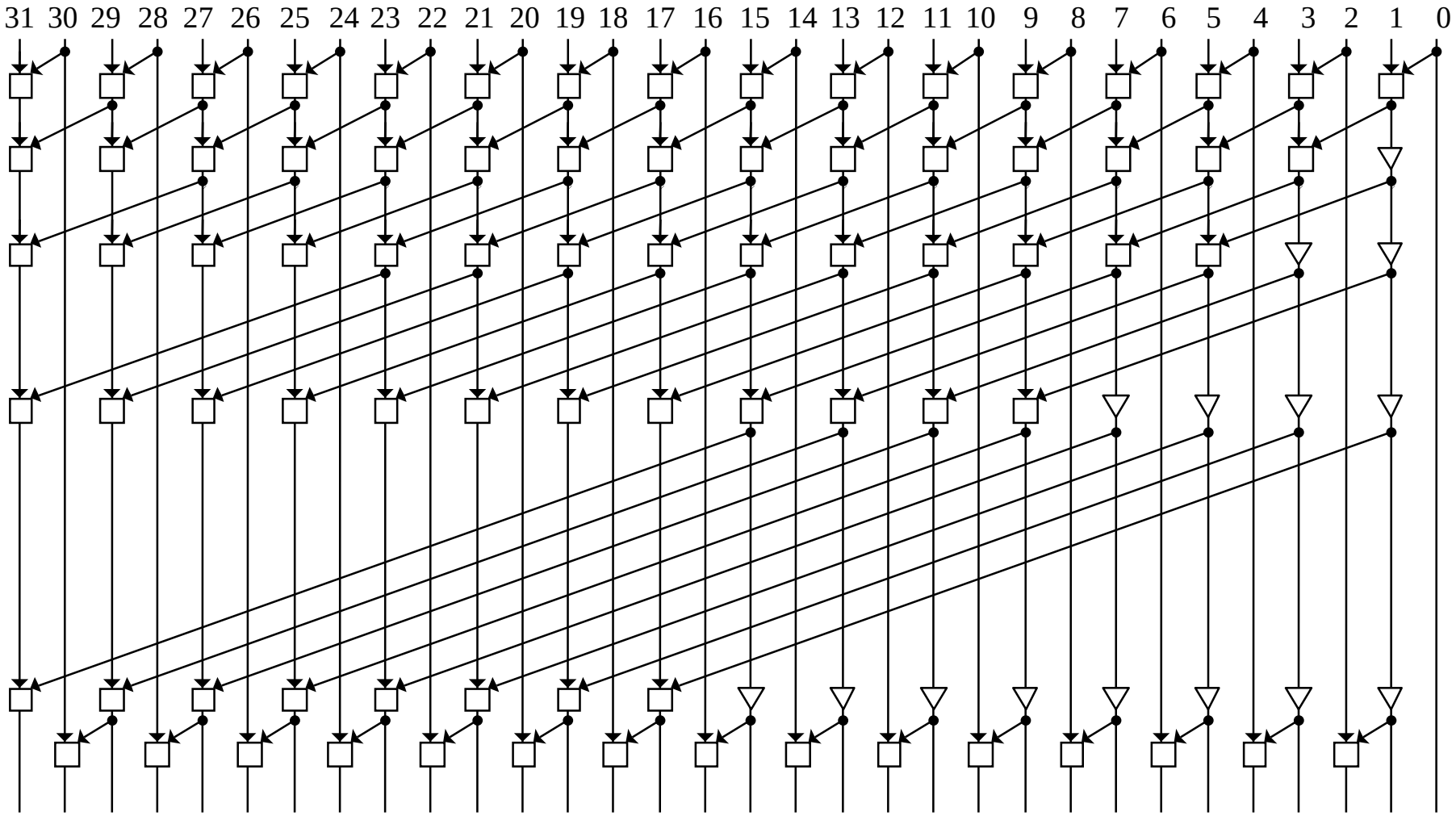
Le "fan-out" est toujours 2

Additionneur hybride
(arbre de Brent & Kung / Carry Select)



Pendant que l'arbre calcule 1 retenue sur 4,
les CSA propagent la retenue sur 4 positions.

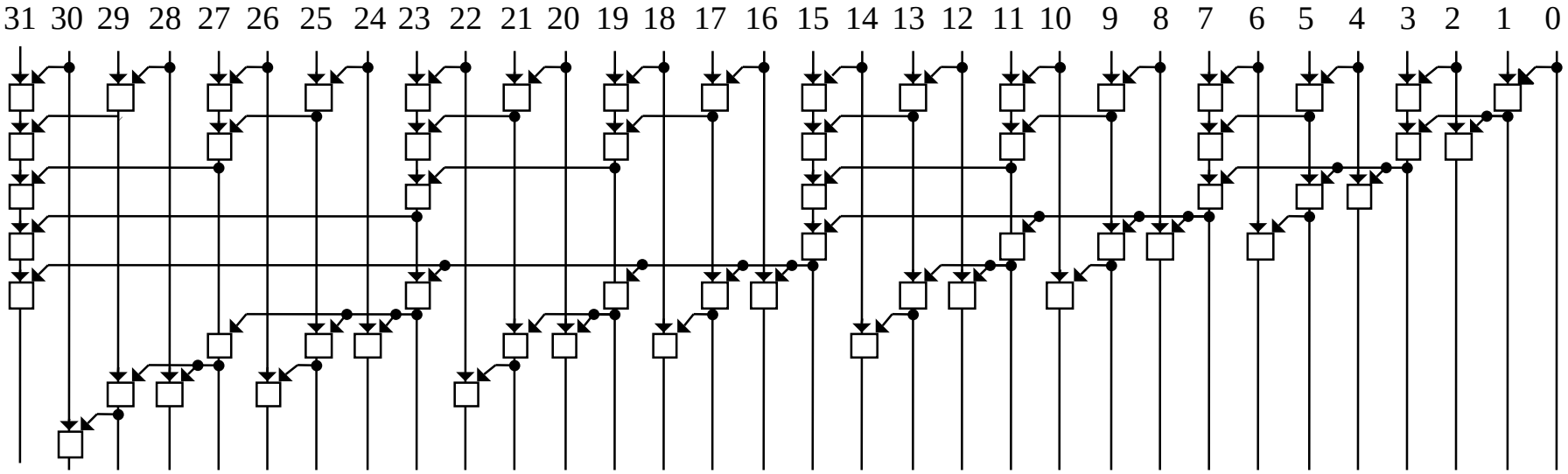
Additionneur de Han et Carlson



Résumé sur les additionneurs à cellule de Brent et Kung (Δ -cell)

Type d'addition	# de Δ-cells	Délai (Δ-cell)	Max. fan-out	Exemple n = 32 bits		
Propagation	$n-1$	$n-1$	2	31	31	2
2-level carry select	$\lceil 2n-\sqrt{2n} \rceil$	$\lceil \sqrt{2n} \rceil$	$\lceil \sqrt{2n} \rceil$	54	8	6
3-level carry select	$3n$??	$\lceil \sqrt[3]{6n} \rceil$	$\lceil \sqrt[3]{6n} \rceil$	66	6	9
Brent-Kung	$\lceil 2n- \log_2(n) \rceil$	$\lceil 2 \log_2(n) -2 \rceil$	$\lceil 2 \log_2(n) -2 \rceil$	57	8	5
Variante du BK	ci-dessus +1	ci-dessus -1	$\lceil 2 \log_2(n) -2 \rceil$	58	7	5
Sklanski	$\lceil n/2 \log_2(n) \rceil$	$\lceil \log_2(n) \rceil$	$n/2$	80	5	16
Kogge and Stone	$\lceil n (\log_2(n)-1) \rceil$	$\lceil \log_2(n) \rceil$	2	129	5	2
Han and Carlson	$\lceil n/2 \log_2(n) \rceil$	$\lceil \log_2(n) \rceil +1$	2	80	6	2
Hybrid CS-VN	$\lceil 2.5n -\sqrt{2n} \rceil$	$\lceil 1+\sqrt{n} \rceil$	$n/2$	65	6	16

Exercice



1- De quel type est cet additionneur ?

2- Quel est le délai (en nombre de cellules sur le chemin critique) de cet additionneur ?

3- Proposer une modification (ajout de une cellule) qui fasse décroître la longueur du chemin critique (de une cellule)

Additionneur de Ling

Un des additionneurs les plus rapides a été décrit par H. Ling, d'IBM, en 1981. La contribution de Ling est la "pseudo retenue". Prenons par exemple le calcul du 6^{eme} bit de la somme.

$s_6 = p_6 \oplus \mathbf{G}_{5,0}$ ($\mathbf{G}_{i,0}$ est la retenue au rang i+1)

$\mathbf{G}_{5,0} = \mathbf{G}_{5,3} \vee \mathbf{P}_{5,3}\, \mathbf{G}_{2,0}$

$\mathbf{G}_{5,3} = g_5 \vee \overline{k_5}\, g_4 \vee \overline{k_5}\ \overline{k_4}\, g_3$

$\mathbf{G}_{2,0} = g_2 \vee \overline{k_2}\, g_1 \vee \overline{k_2}\ \overline{k_1}\, g_0$

$\mathbf{P}_{5,3} = \overline{k_5}\ \overline{k_4}\ \overline{k_3}$

Le délai de s_6 est déterminé par $\mathbf{G}_{5,0}$ Tous les termes de $\mathbf{G}_{5,0}$ contiennent $\overline{k_5}$ sauf le premier qui est g_5. $\mathbf{G}_{5,0}$ peut être simplifié en notant que $g_i = \overline{k_i} \wedge g_i$. Si on remplace g_5 par $\overline{k_5} \wedge g_5$ on peut mettre $\overline{k_5}$ en facteur : $s_6 = p_6 \oplus (\overline{k_5}\, \boldsymbol{G}_{5,0})$. On note $\boldsymbol{G}_{i,0}$ la pseudo retenue au rang i+1.

$\boldsymbol{G}_{5,0} = \boldsymbol{G}_{5,3} \vee \boldsymbol{P}_{5,3}\, \boldsymbol{G}_{2,0}$. (se vérifie à partir de $\mathbf{G}_{5,0} = \mathbf{G}_{5,3} \vee \mathbf{P}_{5,3}\, \mathbf{G}_{2,0}$)

$\boldsymbol{G}_{5,3} = g_5 \vee g_4 \vee \overline{k_4}\, g_3$

$\boldsymbol{G}_{2,0} = g_2 \vee g_1 \vee \overline{k_1}\, g_0$

$\boldsymbol{P}_{5,3} = \overline{k_4}\ \overline{k_3}\ \overline{k_2}$

Pour s_6 on précalcule p_6 et $p_6 \oplus \overline{k_5}$.

On vérifie que le calcul de $\boldsymbol{G}_{5,3}$ est plus rapide que le calcul de $\mathbf{G}_{5,3}$.

$\mathbf{G}_{5,3} = a_5\, b_5 \vee (a_5 \vee b_5)\, a_4\, b_4 \vee (a_5 \vee b_5)\, (a_4 \vee b_4)\, a_3\, b_3$

$\boldsymbol{G}_{5,3} = a_5\, b_5 \vee a_4\, b_4 \vee (a_4 \vee b_4)\, a_3\, b_3$

Additionneur de Ling (2)

La performance repose sur un arbre ternaire (maximum raisonnable d'entrées par porte), sur un calcul direct à partir des a_i et b_i (sans passer explicitement par g_i, p_i, $\overline{k_i}$), su un arbre plus simple et sur le précalcul de la somme modulo 2 pour le résultat S.
On vérifie facilement que:

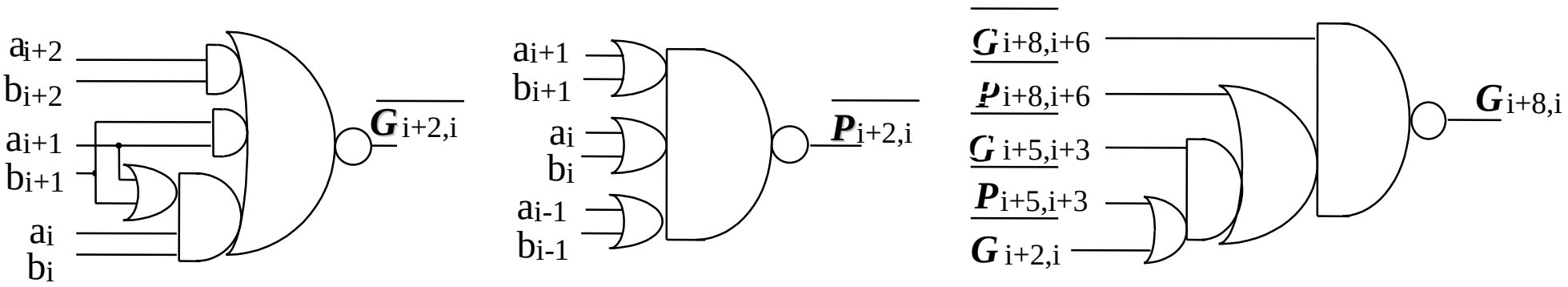
$\mathbf{G}_{i+2,i} = \overline{k}_{i+2}\, \boldsymbol{G}_{i+2,i}$

$\mathbf{P}_{i+2,i}\ \overline{k}_{i-1} = \overline{k}_{i+2}\, \boldsymbol{P}_{i+2,i}$

ce qui permet de vérifier que: $\boldsymbol{G}_{5,0} = \boldsymbol{G}_{5,3} \vee \boldsymbol{P}_{5,3}\, \boldsymbol{G}_{2,0}$. (voir transparent précédent)
A partir de cette propriété on calcule

$\boldsymbol{G}_{i+8,i} = \boldsymbol{G}_{i+8,i+6} \vee \boldsymbol{P}_{i+8,i+6}\boldsymbol{G}_{i+5,i+3} \vee \boldsymbol{P}_{i+8,i+6}\, \boldsymbol{P}_{i+5,i+3}\, \boldsymbol{G}_{i+2,i}$.

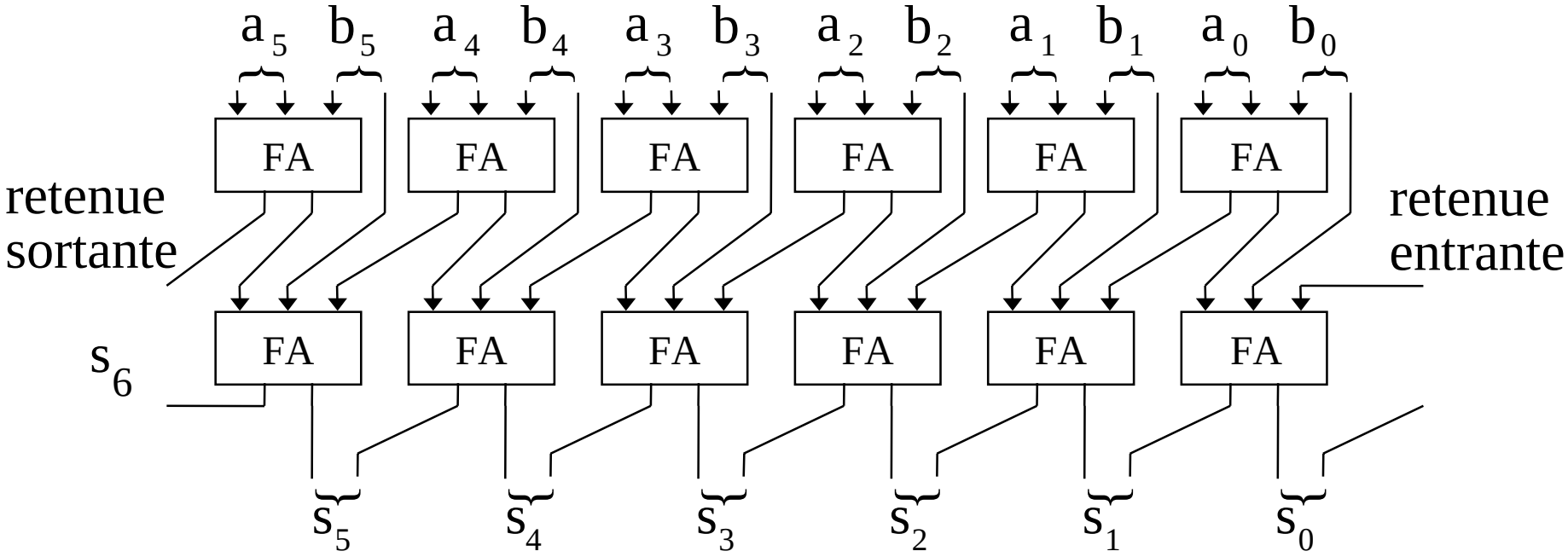
$\boldsymbol{P}_{+8,i} = \boldsymbol{P}_{i+8,i+6}\, \boldsymbol{P}_{i+5,i+3}\, \boldsymbol{P}_{i+2,i}$.



Aucune de ces portes complexes n'a plus de 3 transistors série entre sortie et alimentations

Addition parallèle sans propagation de la retenue CS

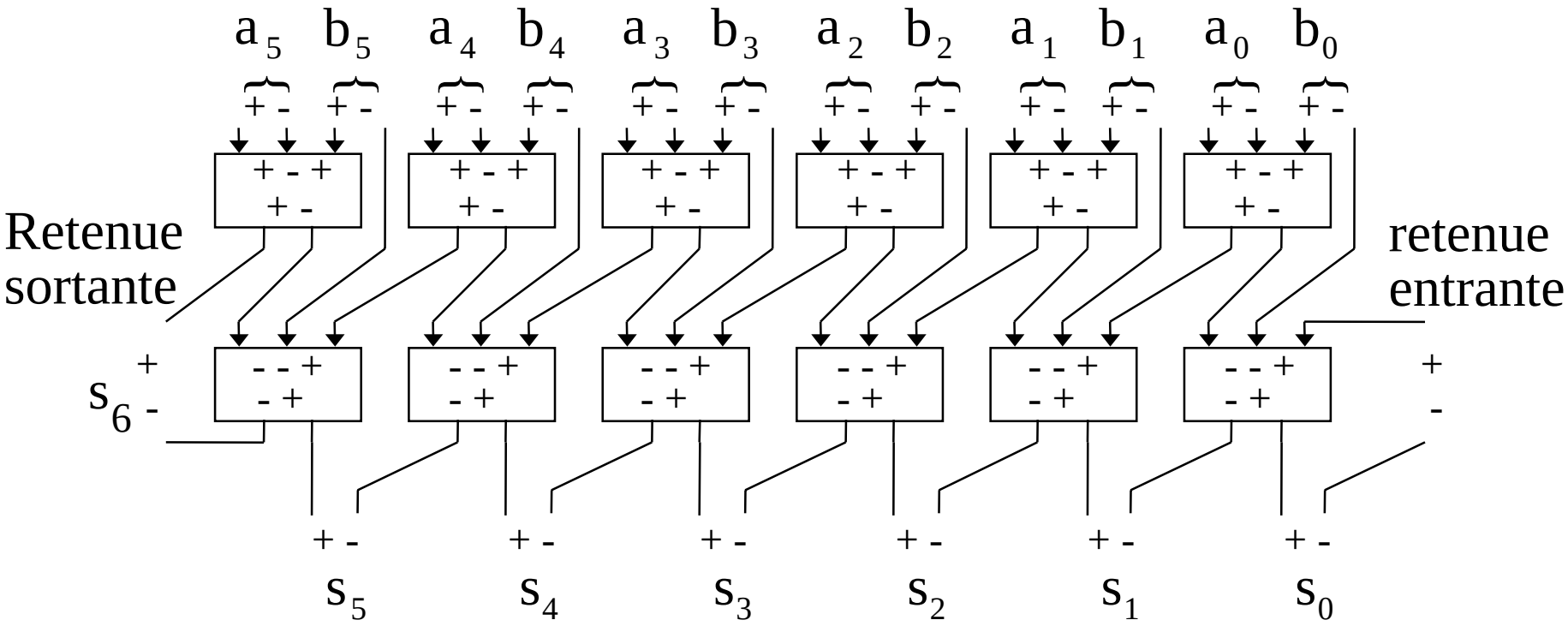
$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad B = \sum_{i=0}^{n-1} b_i\, 2^i \qquad S = \sum_{i=0}^{n} s_i\, 2^i \qquad a_i, b_i, s_i \in \{0,1,2\}$$



La somme pondérée des bits qui entrent est égale à la somme pondérée des bits qui sortent !

Addition parallèle sans propagation de la retenue BS

$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad B = \sum_{i=0}^{n-1} b_i\, 2^i \qquad S = \sum_{i=0}^{n} a_i\, 2^i \qquad a_i, b_i, s_i \in \{-1, 0, 1\}$$



Nombre, Chiffre et Bit

Un nombre est un ensemble ordonné de chiffres (en général).
La valeur d'un nombre est (en général) la somme pondérée de ses chiffres.
Les poids sont (en général) des puissances de la base de numération.
La base de numération est (souvent) une puissance de 2 (2,4,8,16)

$$A=\sum_{i=0}^{n-1} a_i\, 2^i \qquad A=\sum_{i=1}^{n} (1+2^{-i})a^i \qquad A=\sum_{i=0}^{n-1} a_i \log (1+2^{-i})$$

Si un chiffre a 2 valeurs, il est codé sur 1 bit. Ces deux valeurs sont généralement 0 et 1
il y a alors confusion entre chiffre et bit
Si un chiffre a plus de 2 valeurs,
sa valeur est (commodément) la somme pondérée des bits qui le codent.

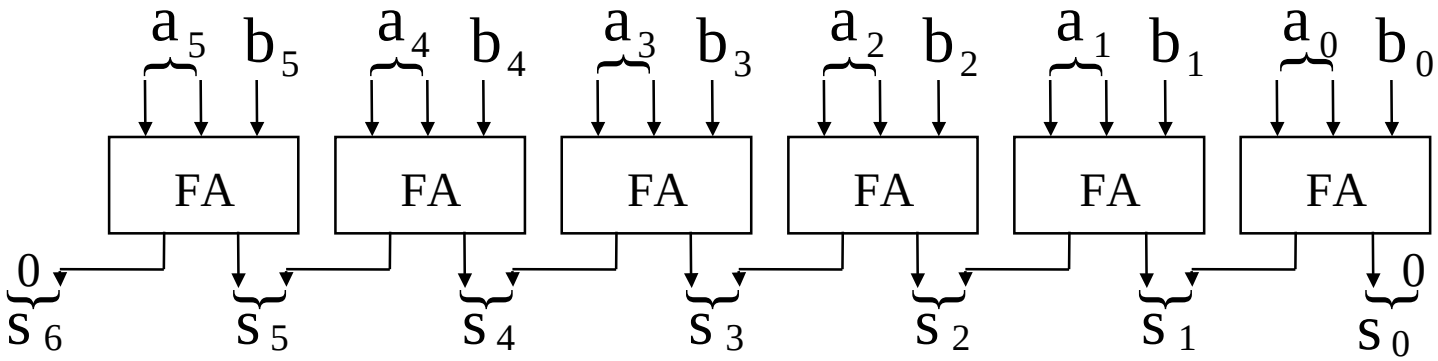
$a_i \in \{0,1\}$ $a_i \in \{-1,1\}$ $a_i \in \{0,1,2\}$ $a_i \in \{0,1,2,3\}$ $a_i \in \{-1,0,1\}$

Des contre-exemples sont donnés.

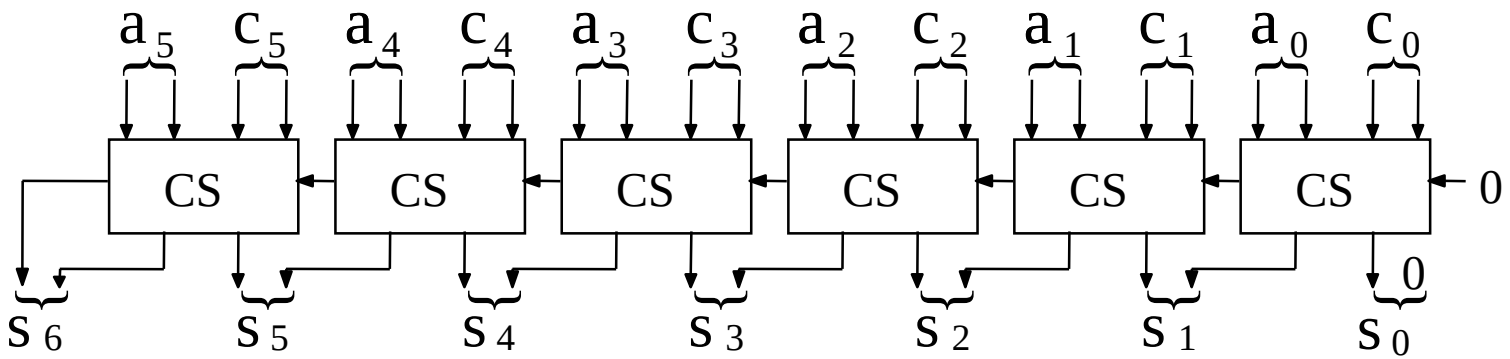
Additions sans propagation

$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad C = \sum_{i=0}^{n-1} c_i\, 2^i \qquad S = \sum_{i=0}^{n} s_i\, 2^i \qquad a_i, c_i, s_i \in \{0,1,2\}$$

1- Hybride: Carry-save + Conventionnel → Carry-save

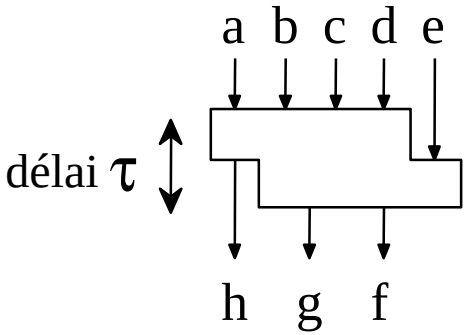


2- Binaire: Carry-save + Carry-save → Carry-save

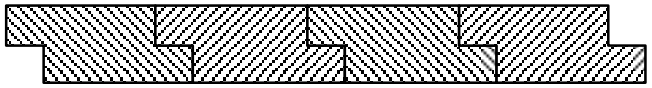


Variantes de cellules de CS

Modèle de délai



h sort avant
que e rentre



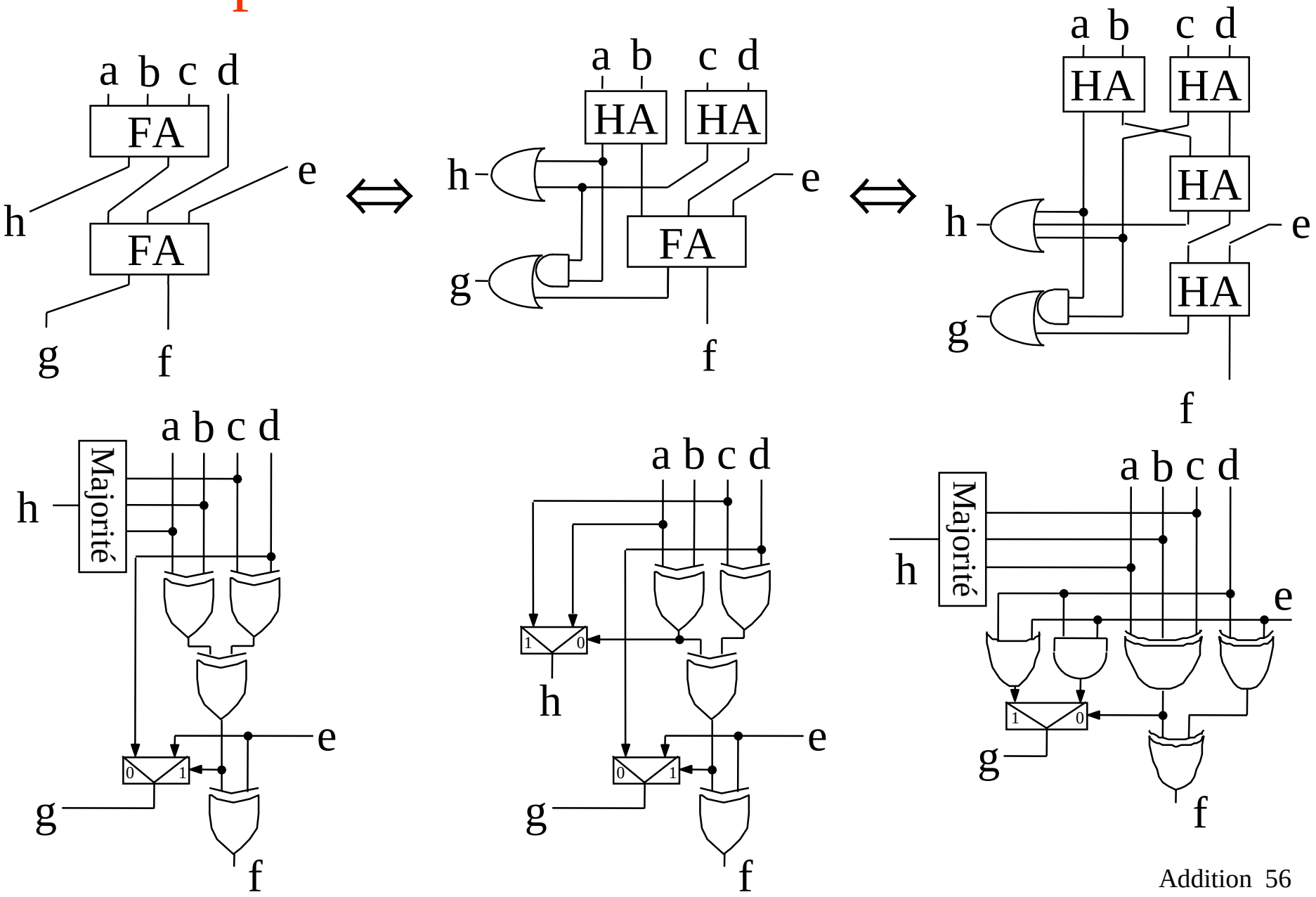
Cette cellule est également appelée “4 donne 2”

$a + b + c + d + e = f + 2*g + 2*h$

h ne dépend pas de e

a	b	c	d	Σ	f	g	h	
0	0	0	0	0	e	0	0	
0	0	0	1	1	$\bar{e}$	e	0	
0	0	1	0	1	$\bar{e}$	e	0	
0	0	1	1	2	e	0/1	1/0	← 2
0	1	0	0	1	$\bar{e}$	e	0	
0	1	0	1	2	e	0/1	1/0	← 4
0	1	1	0	2	e	0/1	1/0	← 8
0	1	1	1	3	$\bar{e}$	e	1	
1	0	0	0	1	$\bar{e}$	e	0	
1	0	0	1	2	e	0/1	1/0	← 16
1	0	1	0	2	e	0/1	1/0	← 32
1	0	1	1	3	$\bar{e}$	e	1	
1	1	0	0	2	e	0/1	1/0	← 64
1	1	0	1	3	$\bar{e}$	e	1	
1	1	1	0	3	$\bar{e}$	e	1	
1	1	1	1	4	e	1	1	

Optimisation de cellules de CS



Multiplication Matérielle

Alain GUYOT

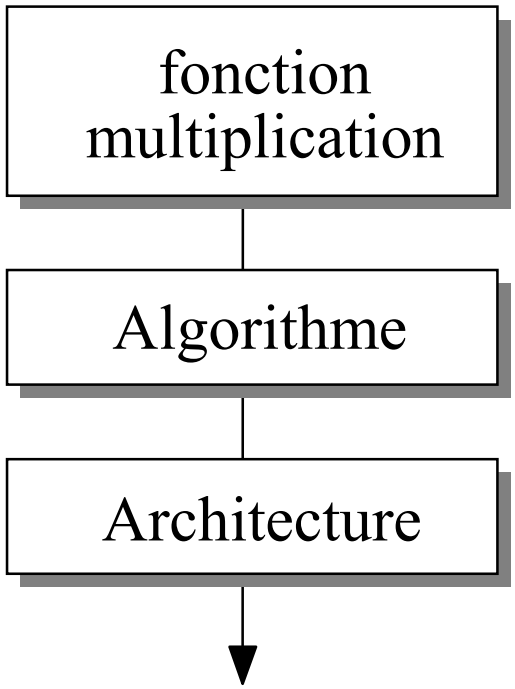
Concurrent Integrated Systems
TIMA

(33) 04 76 57 46 16
Alain.Guyot@imag.fr
http://tima-cmp.imag.fr/Homepages/guyot
Techniques de l'Informatique et de la Microélectronique pour l'Architecture. Unité associée au C.N.R.S. n° B0706

But
Réaliser des multiplieurs

Optimiser la surface et/ou la vitesse

Problèmes
- Propagation de la retenue



Plan

- Multiplieur "naïf" d'entiers ≥ 0 — 1,5 n
- Multiplieur "carry save" d'entiers ≥ 0 — n
- Multiplieur "naïf" d'entiers relatifs — 1,5 n
- Multiplieur "carry save" d'entiers relatifs — n
- Multiplieur performant — $\log_{3/2} n$, $\log_2 n$

Multiplication d'entiers

Soient $A = \sum_{i=0}^{n-1} a_i * 2^i$; $B = \sum_{j=0}^{n-1} b_j * 2^j$ avec $a_i, b_i \in \{0,1\}$.

Le produit $P = A * B$ s'écrit $\left(\sum_{i=0}^{n-1} a_i * 2^i\right) * \left(\sum_{j=0}^{n-1} b_j * 2^j\right)$

ou encore en appliquant la distributivité somme / produit

$$P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(a_i * b_j * 2^{i+j} \right).$$

Or $A < 2^n$ et $B < 2^n$ impliquent que $P < 2^{2n}$,

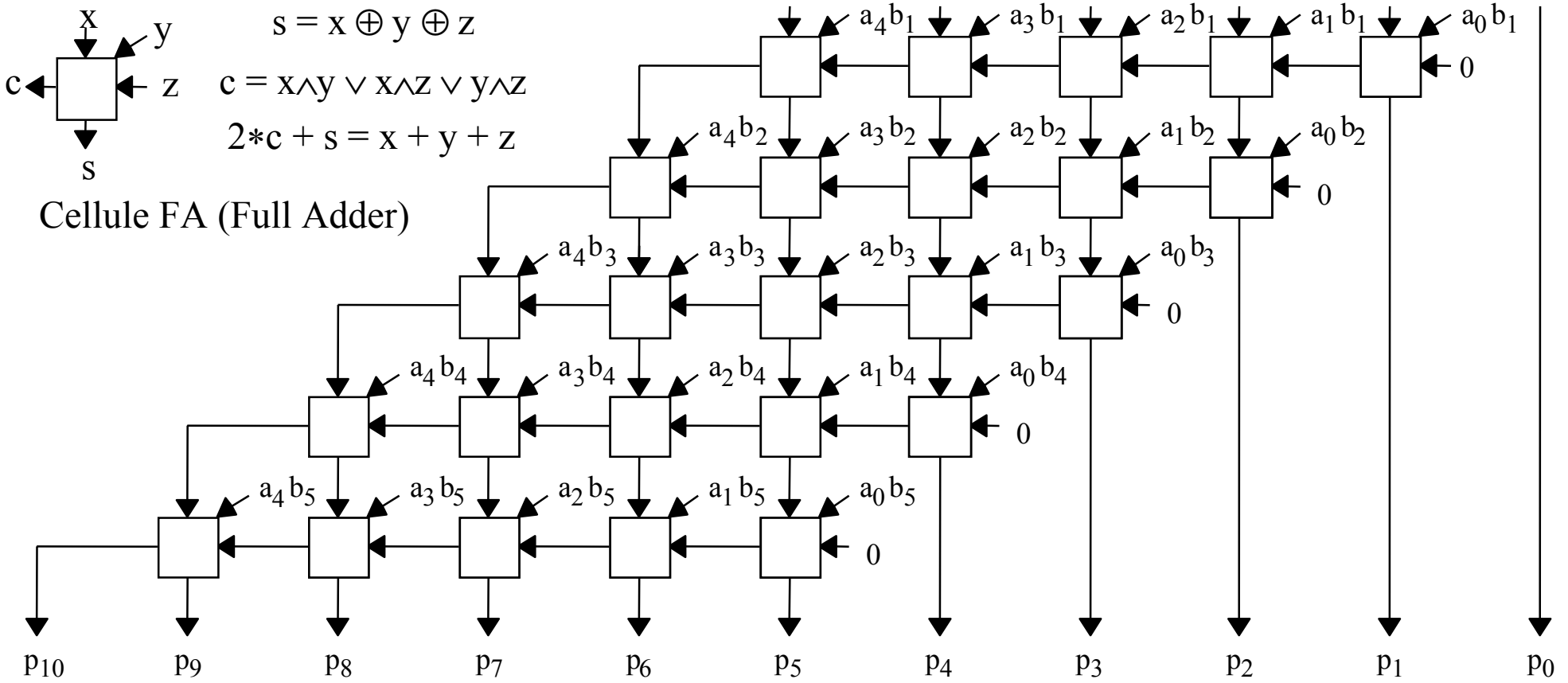
Par conséquent P peut être écrit avec 2n bits.

Le problème est alors de réduire la somme pondérée de n^2 bits en la somme pondérée de 2n bits.

Cette réduction peut être faite en temps $O(n)$, $O(\sqrt[2]{n})$, $O(\log_{3/2} n)$, $O(\log_2 n)$

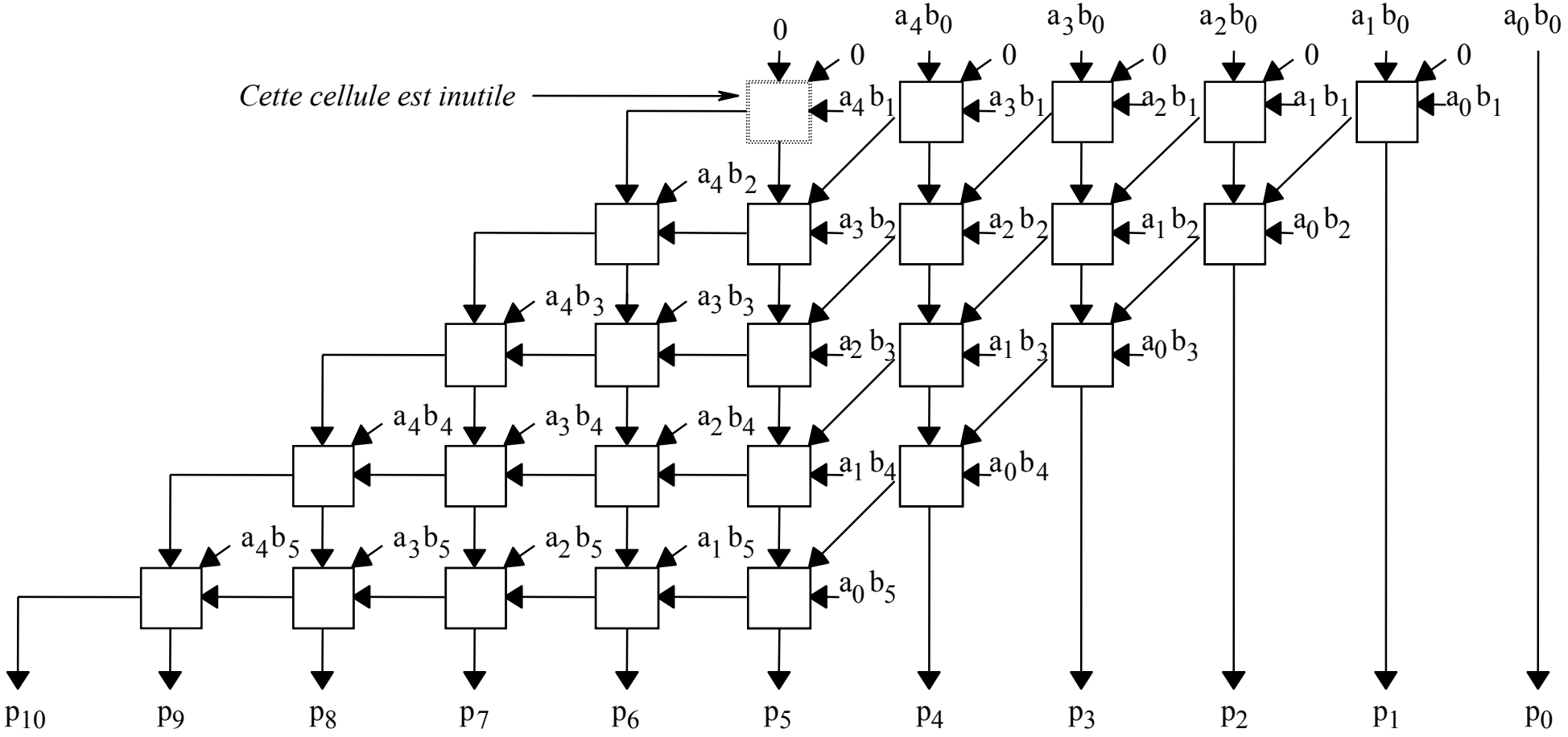
Multiplieur naïf

$$A = \sum_{i=0}^{4} a_i 2^i \qquad B = \sum_{i=0}^{5} b_i 2^i \qquad P = A*B = \sum_{i=0}^{10} p_i 2^i$$

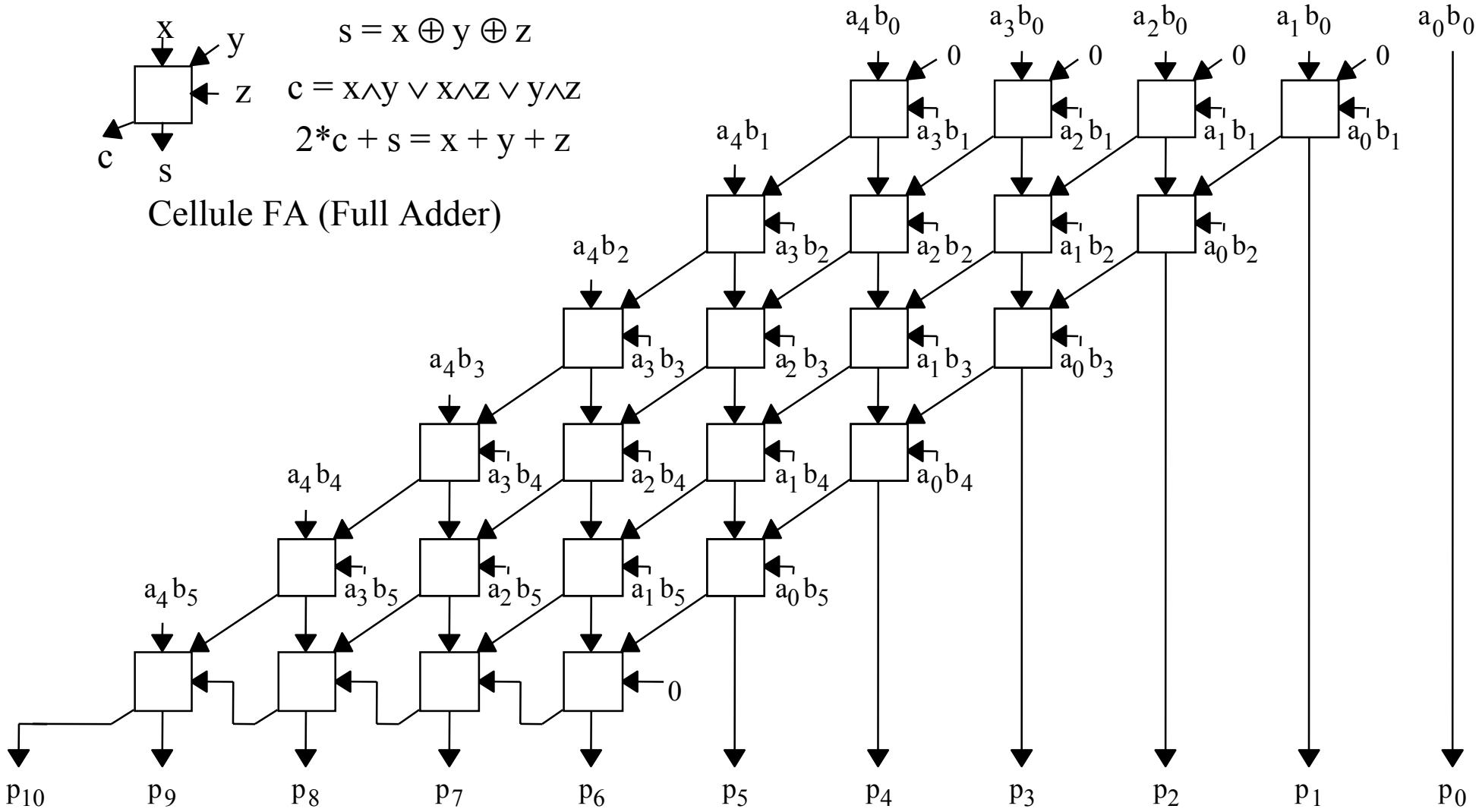


Multiplieur amélioré

Tous les chiffres d'une même colonne sont de même poids.
En jouant sur l'associativité de l'addition, on peut réduire le chemin critique de 13 à 9.



Multiplieur de Braun



Multiplieur signé

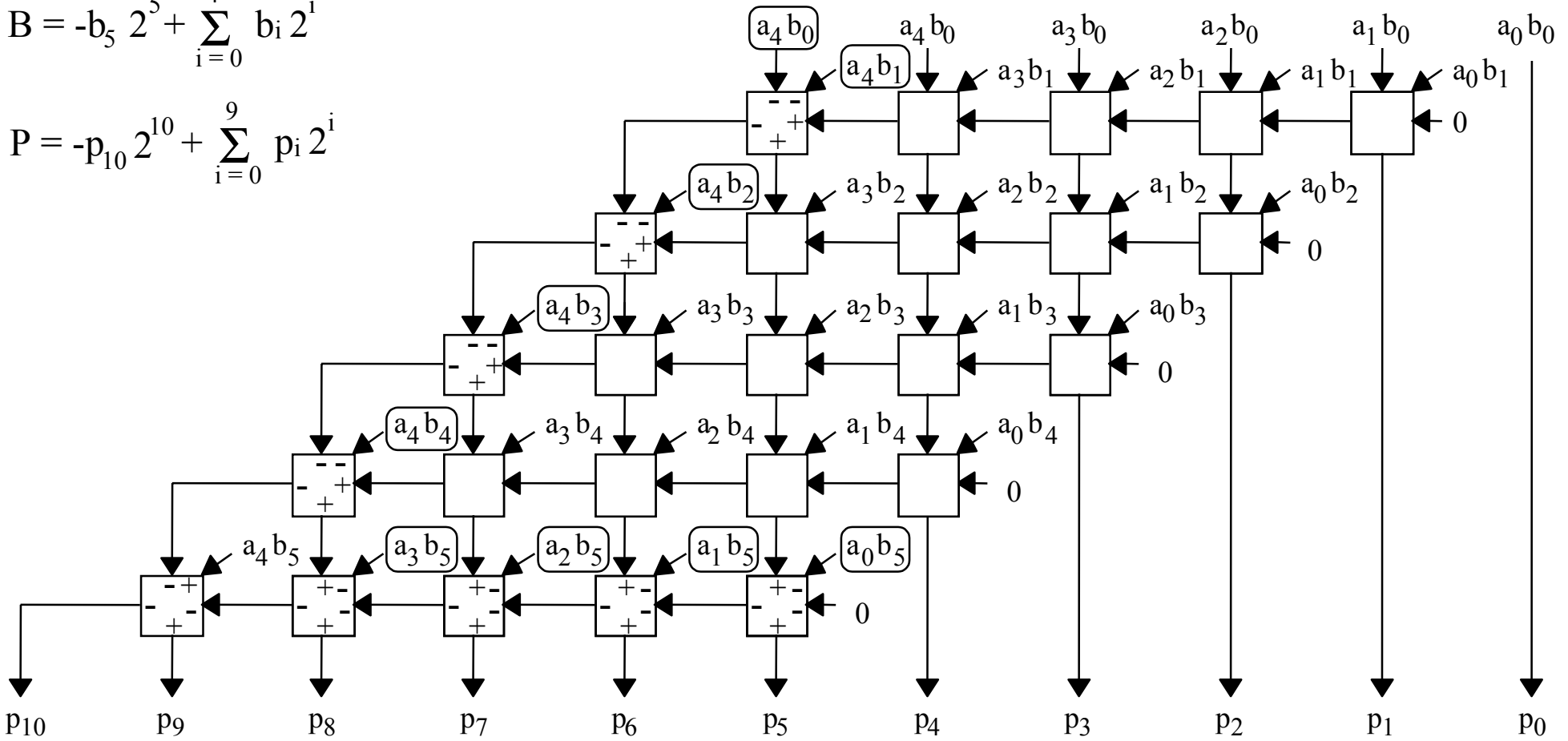
$$A = -a_4\,2^4 + \sum_{i=0}^{3} a_i\,2^i$$

$$B = -b_5\,2^5 + \sum_{i=0}^{4} b_i\,2^i$$

$$P = -p_{10}\,2^{10} + \sum_{i=0}^{9} p_i\,2^i$$

Multiplieur de Pezaris

Pezaris, S.D. "A 40 ns 17 bit by 17 bit Array Multiplier"
IEEE Transactions on Computers (1971) pp 442-447.



Multiplieur de Baugh-Wooley (1)

$$P = A * B = \left(-a_4 2^4 + \sum_{i=0}^{3} a_i 2^i \right) * \left(-b_4 2^4 + \sum_{j=0}^{3} b_j 2^j \right) = a_4.b_4 2^8 + \sum_{i=0}^{3}\sum_{j=0}^{3} a_i.b_j 2^{i+j} - a_4 2^4 \sum_{j=0}^{3} b_j 2^j - b_4 2^4 \sum_{i=0}^{3} a_i 2^i$$

$$= a_4.b_4 2^8 + \sum_{i=0}^{3}\sum_{j=0}^{3} a_i.b_j 2^{i+j} + a_4 2^4 \left(-2^4 + \sum_{j=0}^{3} \overline{b_j} 2^j + 1 \right) + b_4 2^4 \left(-2^4 + \sum_{i=0}^{3} \overline{a_i} 2^i + 1 \right)$$

$$= a_4.b_4 2^8 + \sum_{i=0}^{3}\sum_{j=0}^{3} a_i.b_j 2^{i+j} + \left(\overline{a_4} - 1 + \overline{b_4} - 1 \right) 2^8 + (a_4 + b_4) 2^4 + \sum_{i=0}^{3} \left(a_4.\overline{b_i} + \overline{a_i}.b_4 \right) 2^{i+4}$$

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
					$a_4.\overline{b_0}$	$a_3.b_0$	$a_2.b_0$	$a_1.b_0$	$a_0.b_0$
				$a_4.\overline{b_1}$	$a_3.b_1$	$a_2.b_1$	$a_1.b_1$	$a_0.b_1$	
			$a_4.\overline{b_2}$	$a_3.b_2$	$a_2.b_2$	$a_1.b_2$	$a_0.b_2$		
		$a_4.\overline{b_3}$	$a_3.b_3$	$a_2.b_3$	$a_1.b_3$	$a_0.b_3$			
	$a_4.b_4$	$\overline{a_3}.b_4$	$\overline{a_2}.b_4$	$\overline{a_1}.b_4$	$\overline{a_0}.b_4$				
	$\overline{a_4}$				a_4				
1	$\overline{b_4}$				b_4				

2^9	2^8
	$a_4.b_4$
	$\overline{a_4}$
1	$\overline{b_4}$
2^9	2^8
$a_4 \vee b_4$	$a_4 \vee b_4$

Amélioration de Blankenship

Multiplieur de Baugh-Wooley (2)

$$A = -a_4 2^4 + \sum_{i=0}^{3} a_i 2^i$$

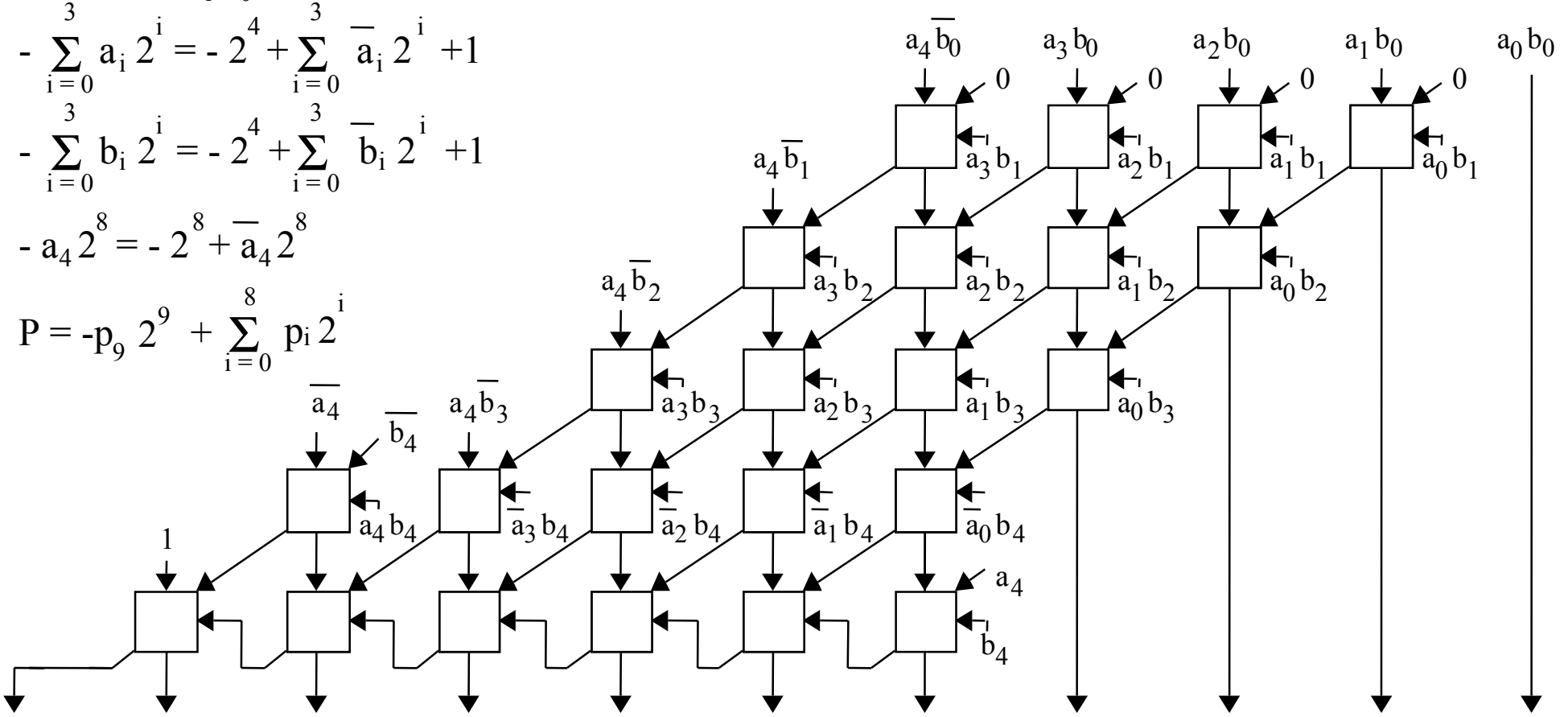
$$B = -b_4 2^4 + \sum_{i=0}^{3} b_i 2^i$$

$$-\sum_{i=0}^{3} a_i 2^i = -2^4 + \sum_{i=0}^{3} \overline{a}_i 2^i + 1$$

$$-\sum_{i=0}^{3} b_i 2^i = -2^4 + \sum_{i=0}^{3} \overline{b}_i 2^i + 1$$

$$-a_4 2^8 = -2^8 + \overline{a}_4 2^8$$

$$P = -p_9 2^9 + \sum_{i=0}^{8} p_i 2^i$$



Multiplieur rapide

Stratégie

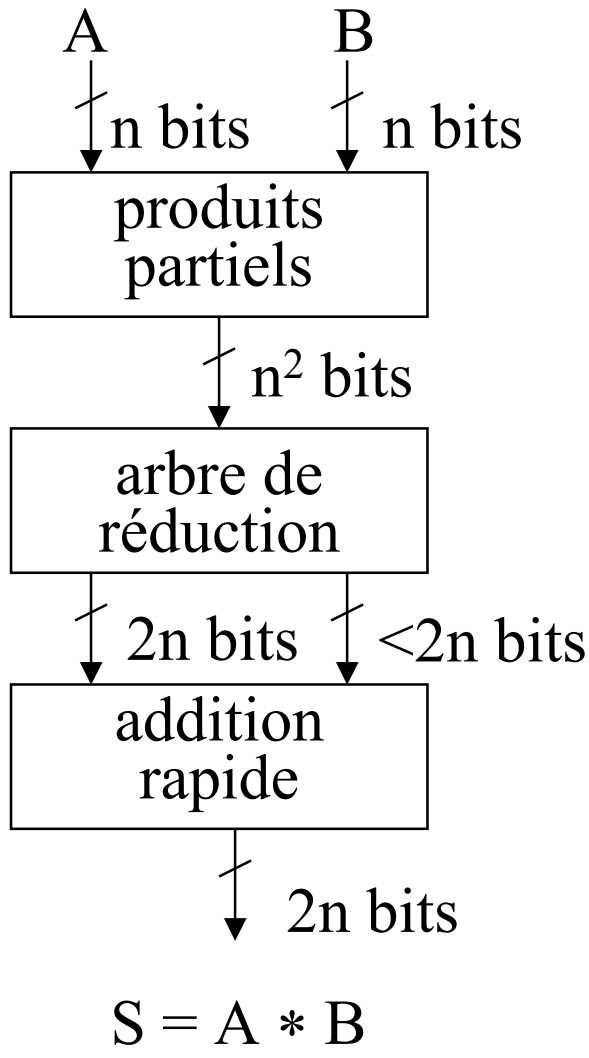
1- Au début calculer simultanément tous les produits partiels $a_i * b_j$

(il y en a n^2 pour le produit naïf, $n^2 + 5$ pour Baugh-Wooley et $n/2(n+4)$ pour Booth modifié)

2- Puis réduire ces produits partiels à 1 ou 2 bit(s) pour chaque poids sans propager de retenue

3- Enfin additionner les deux nombres de l'étape précédente le plus vite possible (ça, on sait faire)

Pour l'étape 2, on sait déjà combien de cellule "FA" et "HA" il y aura dans chaque colonne, cependant on va perdre la régularité des connections.



Exercice

On veut multiplier 2 nombres entiers A et B de n bits chacun.

1- Combien y a t'il de produits partiels ($a_j * b_k$) au total ?

2- Combien y a t'il de produits partiels ($a_j * b_k$) de poids 2^i ($0 \leq i \leq 2n-2$) ?

3- Combien faut-il de Cellules FA pour réduire tous ces produits partiels à (2n - 1) bits ?

4- Combien faut-il de Cellules HA ?

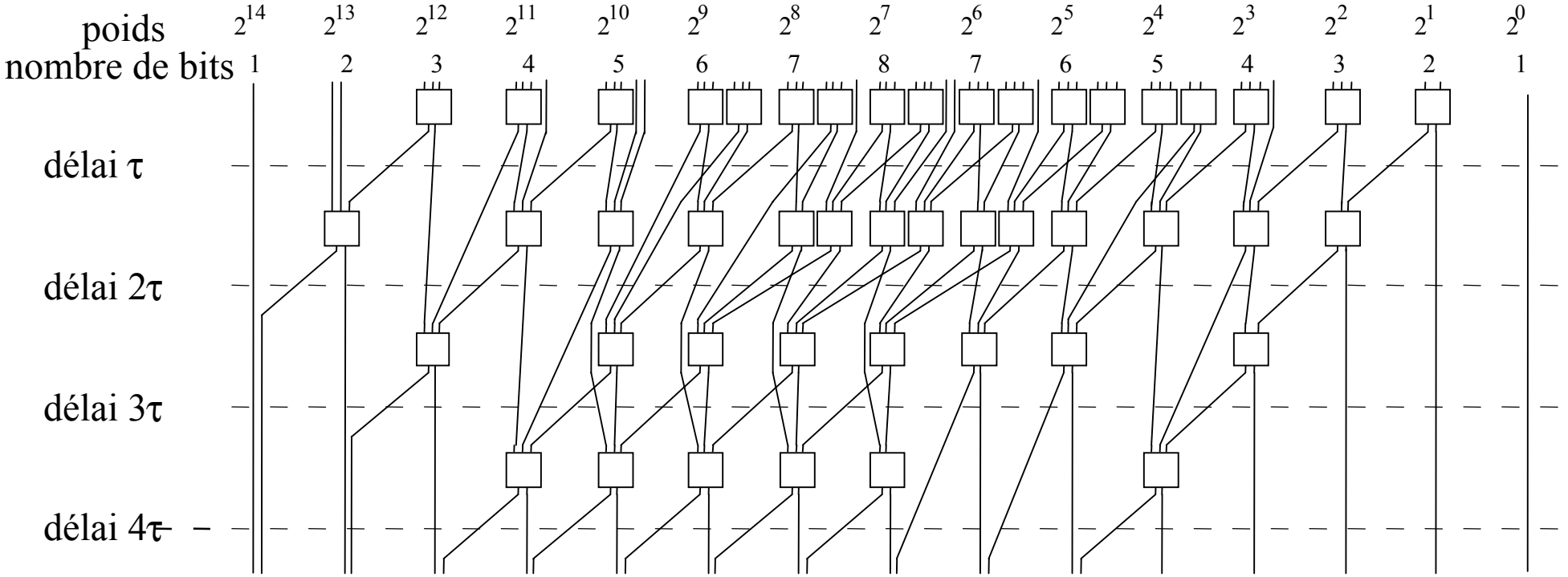
5- En comptant les retenues, combien y a t'il de bits de poids 2^i à réduire ?

6- Combien de cellules FA faut il traverser au minimum pour réduire k bits de poids 2^i en

un bit de poids 2^i et $1+\left\lceil \frac{k-3}{2} \right\rceil$ bits de poids 2^{i+1} ?

Arbre de Wallace

Calcul des n^2 produits partiels $a_i * b_j$
(exemple de 64 produits bit à bit)

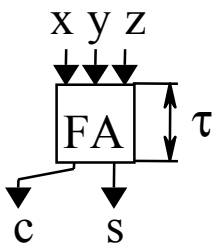


Vers addition rapide (10 bits)

La somme pondérée des bits qui entrent est égale à la somme pondérée des bits qui sortent !

Algorithme de Dadda

Equations logiques et modèle de délai

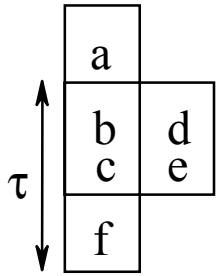


$s = x \oplus y \oplus z$

$c = x \wedge y \vee x \wedge z \vee y \wedge z$

$2*c + s = x + y + z$

a, f : # bits
b, d : # FA
c, e : # HA



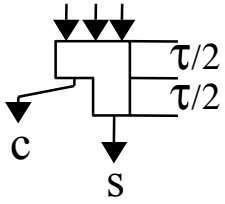
$3*b + 2*c \leq a$

$f = a + d + e - 2*b - c$

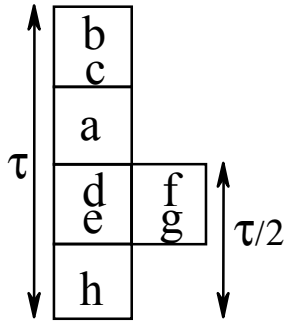
1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
		FA	FA	FA	FA FA	FA FA	FA FA	FA FA	FA FA	FA HA	FA	FA	HA	
1	3	2	3	5	4	5	6	5	4	3	3	2	1	1
	FA		FA	FA	FA	FA HA	FA FA	FA HA	FA	FA	FA	HA		
2	1	3	2	4	4	4	4	3	3	2	2	1	1	1
		FA		FA	FA	FA	FA	FA	FA		HA			
2	2	1	3	3	3	3	3	2	1	3	1	1	1	1
			FA	FA	FA	FA	HA			FA				
2	2	2	2	2	2	2	2	2	2	1	1	1	1	1

Algorithme de Dadda

Modèle de délai plus précis



La retenue c se calcule 2 fois plus vite que la somme s



a, h : # bits
b, d, f : # FA
c, e, g : # HA

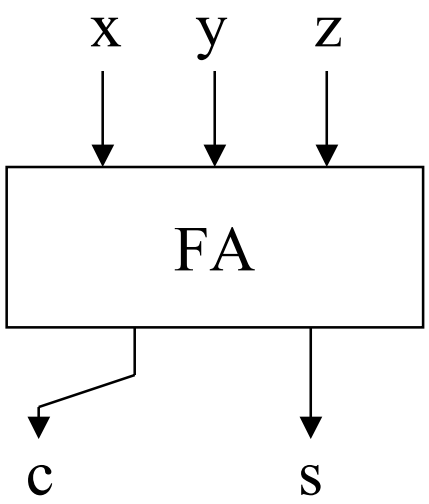
$3*d + 2*e \leq a$

$h = a + b + c + f + g - 3*d - 2*e$

1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
		FA	FA	FA	FA FA	FA FA	FA FA	FA FA	FA FA	FA	FA	FA	HA	
1	3	1	2	4	2	3	4	3	1	3	2	1	0	1
	FA			FA		FA	FA	FA		FA				
1	0	2	4	2	5	3	4	2	4	1	3	2	1	1
			FA		FA	FA	FA	HA	FA		FA	HA		
2	1	3	1	4	3	2	3	2	1	3	1	0	1	1
		FA		FA	FA	HA	FA			FA				
2	2	0	3	2	2	2	1	3	3	0	2	1	1	1
								FA	FA		HA			
2	2	1	3	3	3	3	3	1	0	2	0	1	1	1
			FA	FA	FA	FA	FA			HA				
2	2	2	1	1	1	1	0	2	2	0	1	1	1	1
2	2	2	2	2	2	2	1	2	2	1	1	1	1	1

τ/2

Modèles de délai de « FA »



$x \leq y \leq z$

(x arrive avant ou en même temps que y)

(y arrive avant ou en même temps que z)

$s = \max.\ (\ y + \delta_1,\ \ z + \delta_2)$

$c = z + \delta_3$

Valeurs courantes :

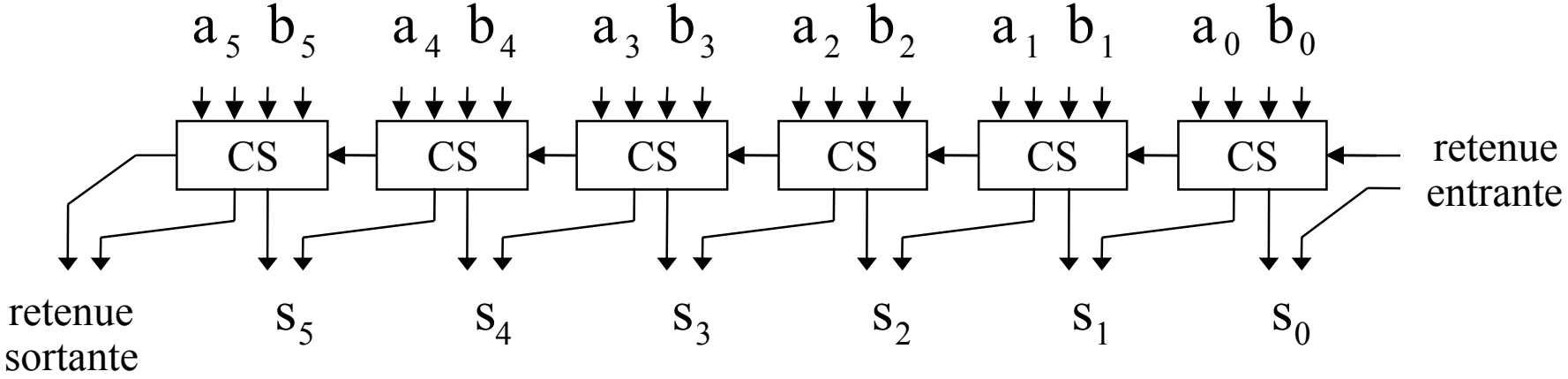
$\delta_1 = 1\ ,\ \delta_2 = 1\ ,\ \delta_3 = 1$

$\delta_1 = 1\ ,\ \delta_2 = 1\ ,\ \delta_3 = 1/2$

$\delta_1 = 1\ ,\ \delta_2 = 1/2\ ,\ \delta_3 = 1/2$

Rappel sur l'additionneur CS (ou réducteur 4 donne 2)

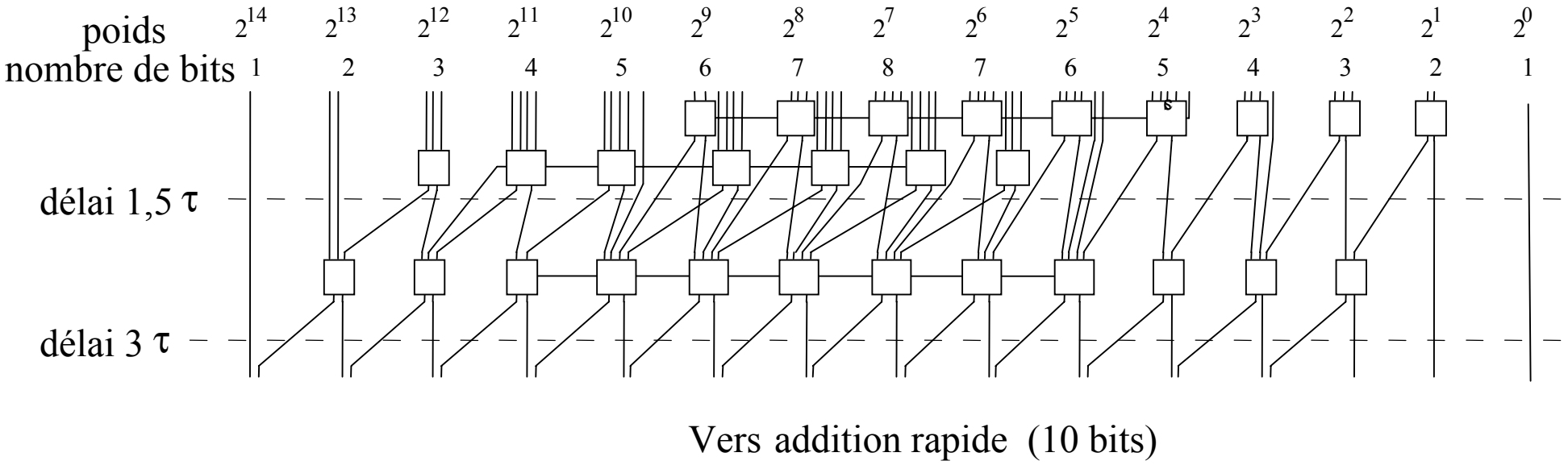
$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad B = \sum_{i=0}^{n-1} b_i\, 2^i \qquad S = \sum_{i=0}^{n} s_i\, 2^i \qquad a_i, b_i, s_i \in \{0,1,2\}$$



La somme pondérée des bits qui entrent est égale à la somme pondérée des bits qui sortent !

Multiplieur arborescent

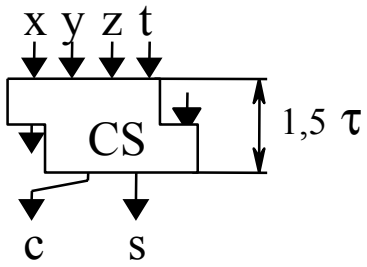
Calcul des n^2 produits partiels $a_i * b_j$
(exemple de 64 bits)



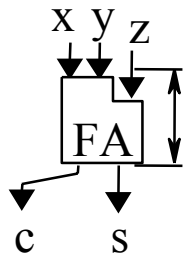
La somme pondérée des bits qui entrent est égale à la somme pondérée des bits qui sortent !

Multiplieur arborescent

1	2	3	4	5	6	7	8	7	6	5	4	3	2	1	# bits à réduire
0	0	1	1	1	2	2	1	1	1	0	0	0	0	0	# retenue horizontale
		FA	CS	CS	FA CS	CS CS	CS CS	CS FA	CS	CS	FA	FA	HA		1,5 τ
1	3	3	2	4	4	4	4	3	4	2	3	2	1	1	
0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	
	FA	FA	FA	CS	CS	CS	CS	CS	CS	HA	FA	HA			
2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	



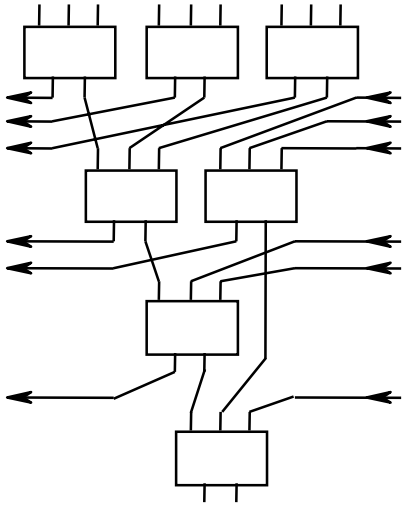
	-4		-5
+1	-1	+1	0
	CS	ou	CS
+1	+1	+1	+1



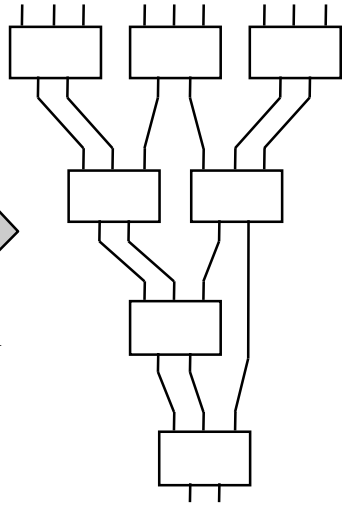
	-2		-3
0	-1	0	0
	FA	ou	FA
+1	+1	+1	+1

Modèle de réducteur

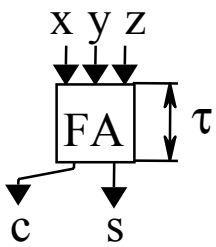
Le nombre et la position des retenues qui sortent sont les mêmes pour les retenues qui entrent

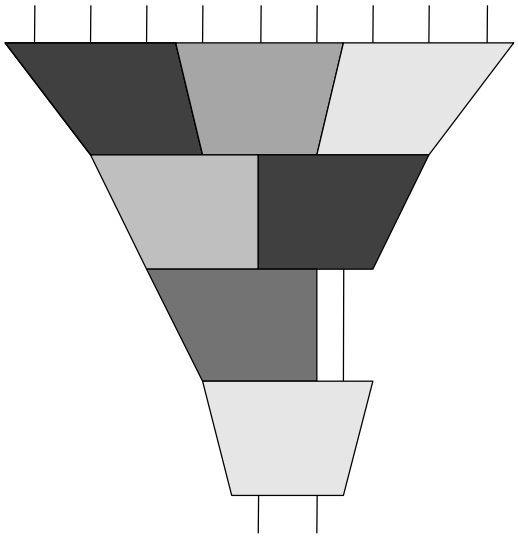


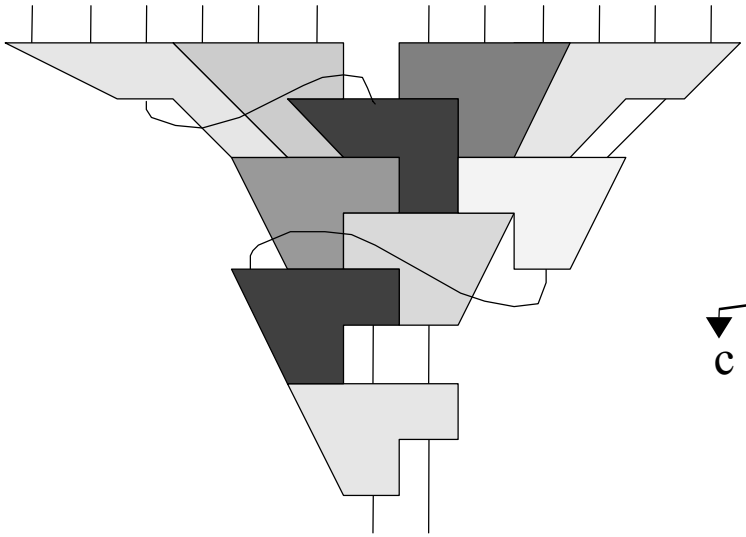
même délai

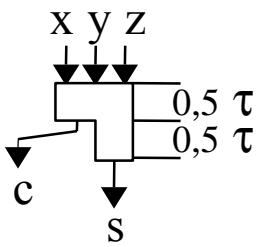


Ce modèle de réducteur est lui même réducteur: les retenues ne sont pas uniformément réparties

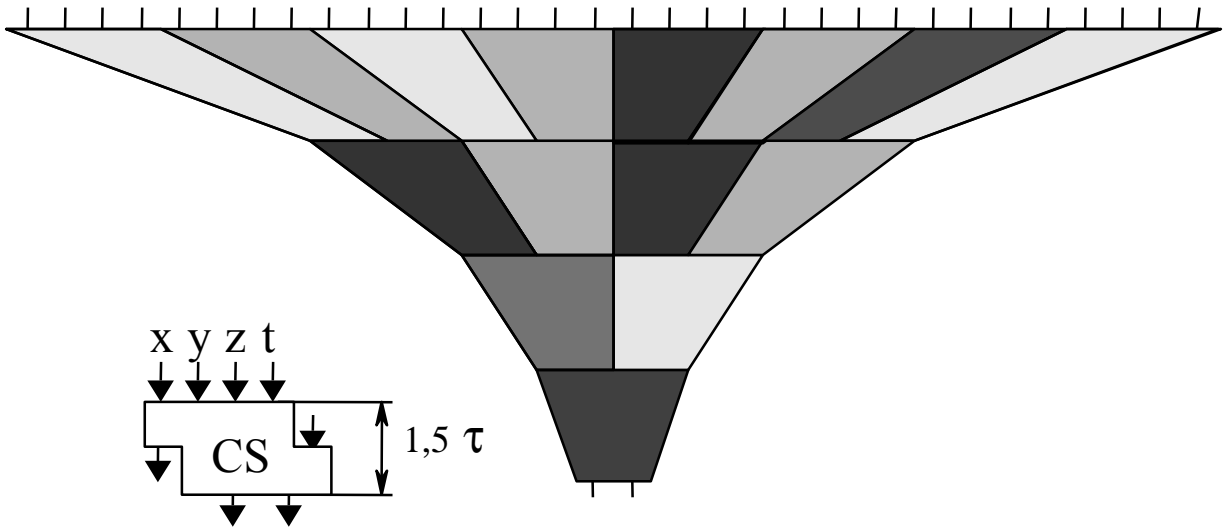




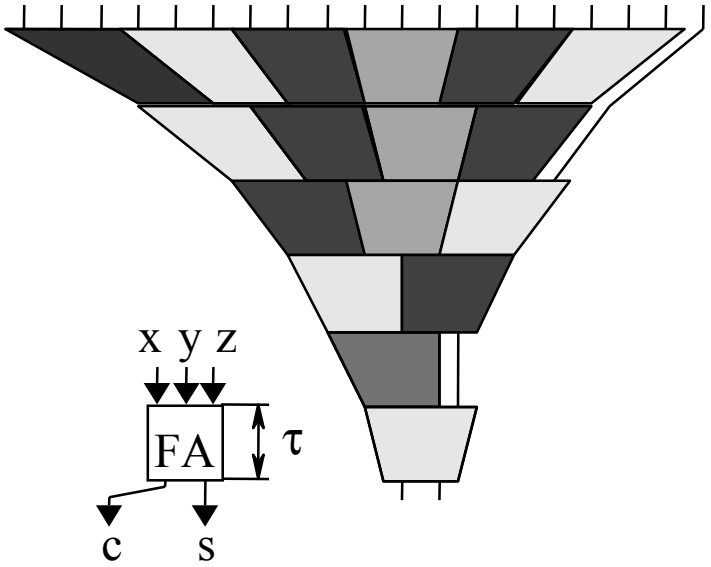




Comparaison de réducteur

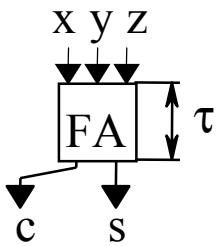


La retenue horizontale est cachée

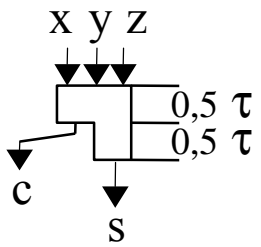


Exemple de multiplication 8 bits par 8 bits:

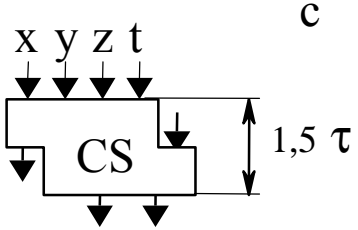
Arbre de Wallace, modèle 1 délai = 4 τ



Arbre de Wallace, modèle 2 délai = 3,5 τ



Réducteur 4 donne 2 (CS) délai 3 τ



Code de Booth

code de B	$B = \sum_{i=0}^{n-1} b_i * 2^i$	$B = \sum_{i=0}^{(n-2)/2} c_i * 4^i$
produit A * B	$A*B = \sum_{i=0}^{n-1} A*b_i*2^i$	$A*B = \sum_{i=0}^{(n-2)/2} A*c_i*4^i$
valeurs des chiffres	$b_i \in \{0,1\}$	$c_i \in \{-2, -1, 0, 1, 2\}$
nombre d'opérations	n-1	$(n-2)/2$
type d'opération	addition	addition/ soustraction

Conversion conventionnel en code de Booth

$$B = \sum_{i=0}^{n-1} b_i * 2^i \quad \Rightarrow \quad B = \sum_{i=0}^{\frac{n-2}{2}} c_i * 4^i$$

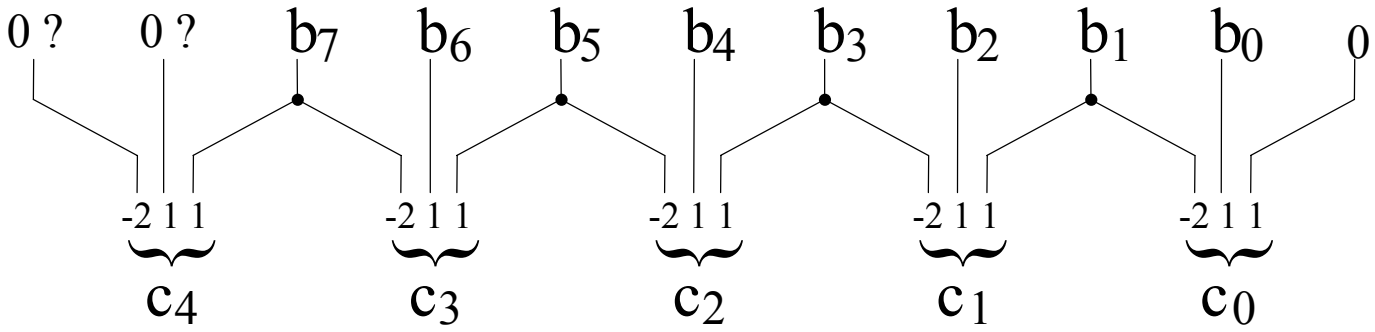
Soient $B'' = \sum_{i=0}^{\frac{n-2}{2}} b_{2i} * 2^{2i}$ termes de rang pair

$B' = \sum_{i=0}^{\frac{n-2}{2}} b_{2i+1} * 2^{2i+1}$ termes de rang impair

$B = B'' + B' = B'' + (2*B') - 2*(1/2*B')$

$(2*B')$: décalé à gauche

$(1/2*B')$: décalé à droite



la valeur d'un chiffre c_i est la somme pondérée de ses bits

$$c_i = -2\, b_{2i+1} + b_{2i} + b_{2i-1} \qquad c_i \in \{-2, -1, 0, 1, 2\}$$

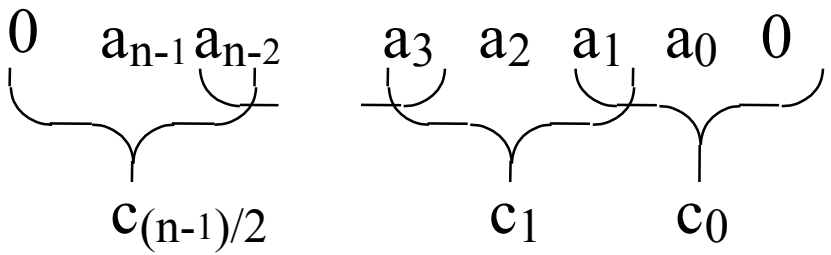
La somme pondérée des chiffres qui entrent est égale à la somme pondérée des chiffres qui sortent

Remarques sur le code de Booth

Entiers positifs

$$A = \sum_{i=0}^{n-1} a_i \, 2^i$$

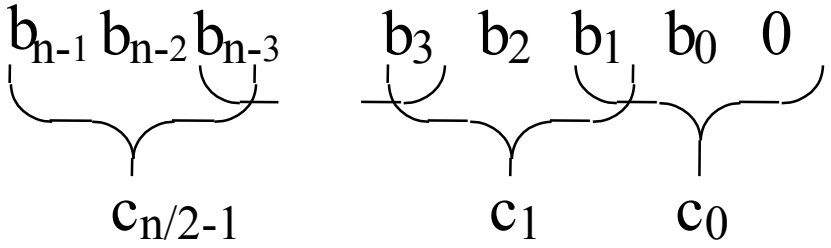
$\Leftrightarrow$



Entiers signés

$$B = -b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i \, 2^i$$

$\Leftrightarrow$

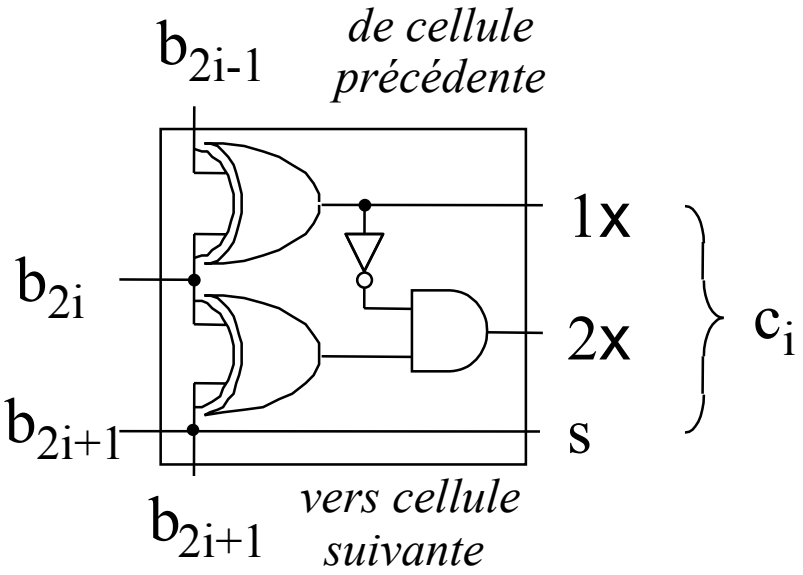


La notation de Booth est redondante: $(0\ 2)_4 = (1\ \overline{2})_4$

(c'est la notation symétrique de redondance minimale en base 4)

Réécriture des chiffres en signe/valeur absolue

b_{2i+1}	b_{2i}	b_{2i-1}	c_i	1x	2x	s
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	0	1	1	0	0
0	1	1	2	0	1	0
1	0	0	-2	0	1	1
1	0	1	-1	1	0	1
1	1	0	-1	1	0	1
1	1	1	-0	0	0	1



$$c_i * A = \left(s \oplus \left(A \wedge 1x \vee \left(2 * A\right) \wedge 2x\right)\right) + s$$

La multiplication d'un nombre par un chiffre est plus aisée mais la valeur d'un chiffre c_i n'est plus la somme pondérée de ses bits (code signe + valeur absolue).

Réduction de chiffres signés

1- Multiples du Multiplicande A

$2*A = -a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0\ 0$

$1*A = -a_6\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$

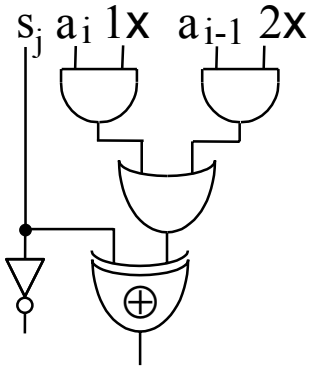
$0*A = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

$-0*A = -1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ +1$

$-1*A = \overline{-a_6}\ \overline{a_6}\ \overline{a_5}\ \overline{a_4}\ \overline{a_3}\ \overline{a_2}\ \overline{a_1}\ \overline{a_0}\ +1$

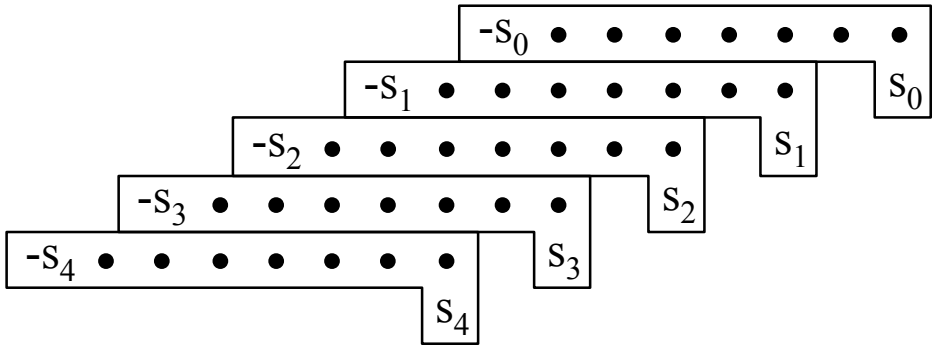
$-2*A = \overline{-a_6}\ \overline{a_5}\ \overline{a_4}\ \overline{a_3}\ \overline{a_2}\ \overline{a_1}\ \overline{a_0}\ 1\ +1$

$c_i*A = \boxed{-d_7\ d_6\ d_5\ d_4\ d_3\ d_2\ d_1\ d_0 \atop s}$



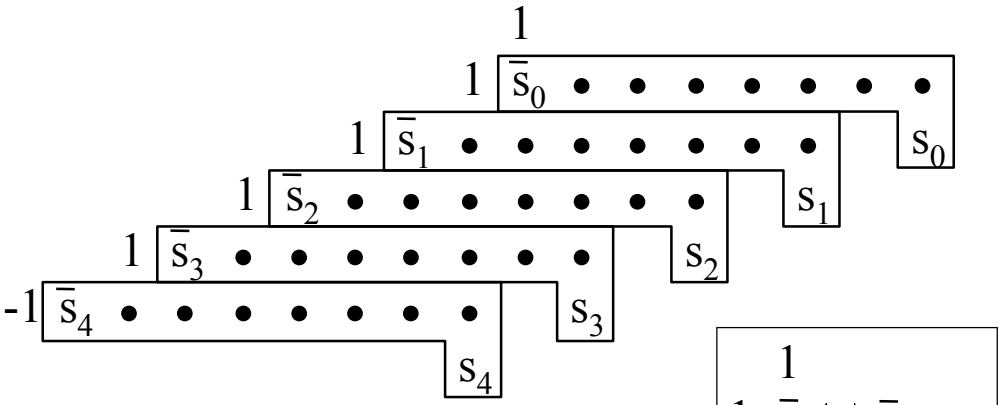
$d_i = s_j \oplus ((a_i \wedge 1x) \vee (a_{i-1} \wedge 2x))$

2- Addition des multiples



3- Propagation des signes pour l'addition

$-s_4\ 0\ -s_3\ 0\ -s_2\ 0\ -s_1\ 0\ -s_0 = -1\ \overline{s}_4\ 1\ \overline{s}_3\ 1\ \overline{s}_2\ 1\ \overline{s}_1\ 1\ \overline{s}_0\ +1$

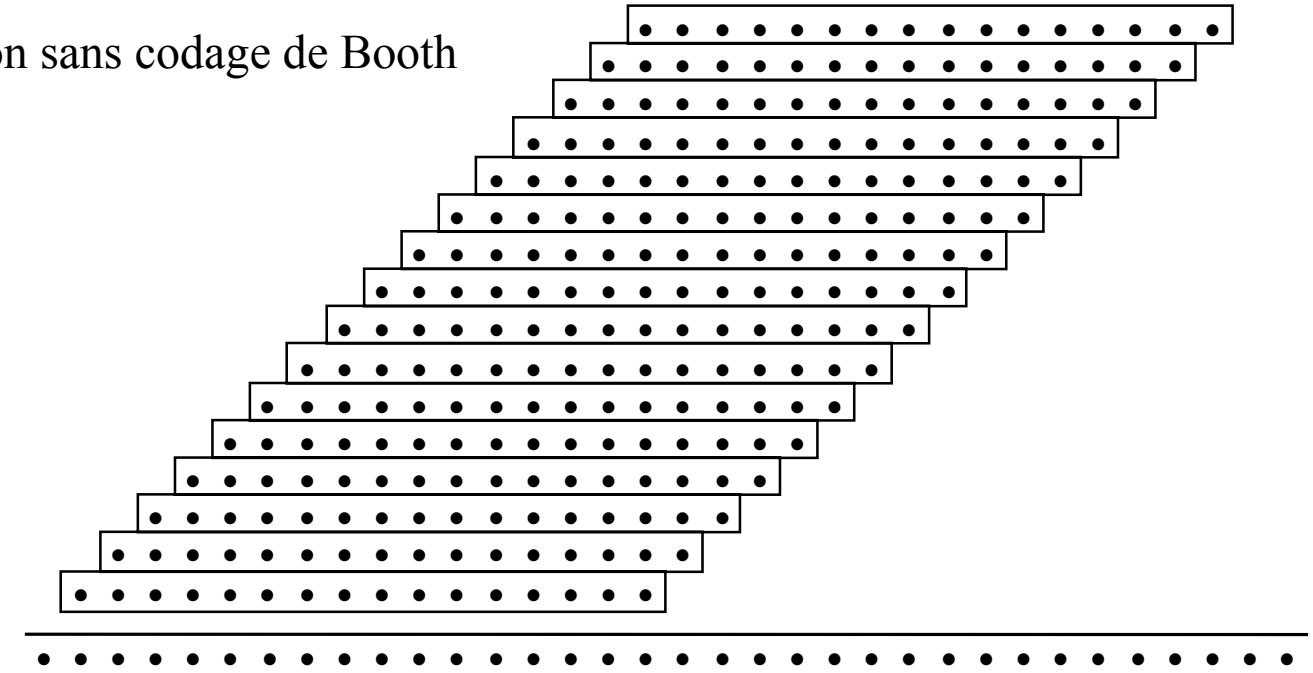


$$\begin{array}{l} 1 \\ 1\ \overline{s}_0 \end{array} \Leftrightarrow \overline{s}_0 s_0 s_0$$

Comparaison Booth-conventionnelle

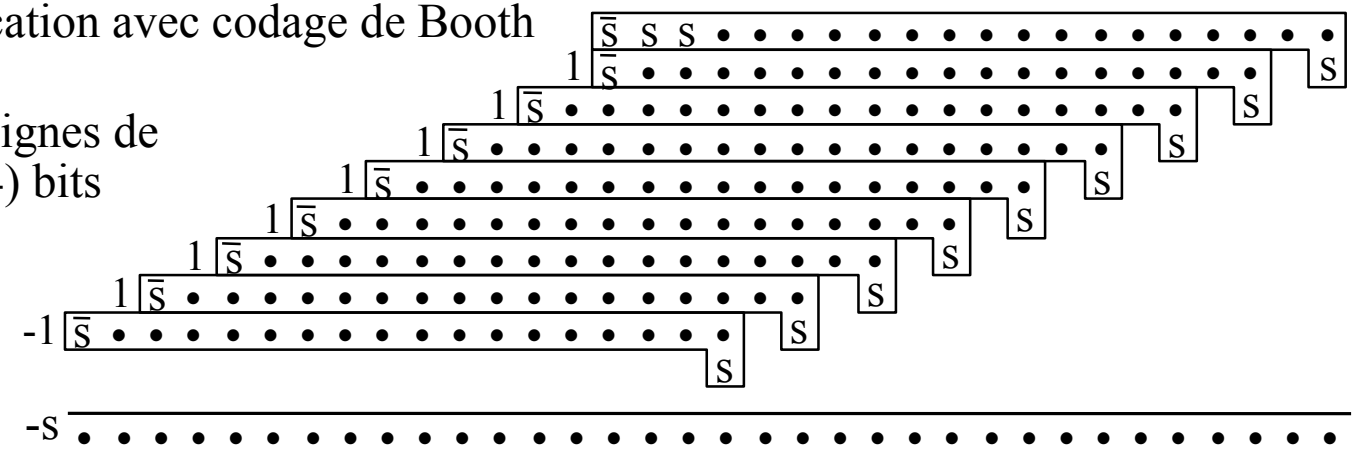
Multiplication sans codage de Booth

n lignes de
n bits



Multiplication avec codage de Booth

n/2 lignes de
(n+4) bits



Multiplieur rapide

multiplicande $A = \sum_{i=0}^{n-1} a_i * 2^i$

multiplieur $B = \sum_{i=0}^{n-1} b_i * 2^i$

Conversion de standard à "code de Booth"

$\sum_{i=0}^{\frac{n-2}{2}} c_i * 4^i$

Arbre binaire d'additionneurs sans propagation

$a_i, b_i, p_i \in \{0,1\}$

$c_i \in \{-2,-1,0,1,2\}$

$s_i \in \{0,1,2\}$

$P = \sum_{i=0}^{2n-1} s_i * 2^i$

conversion de "carry save" à standard

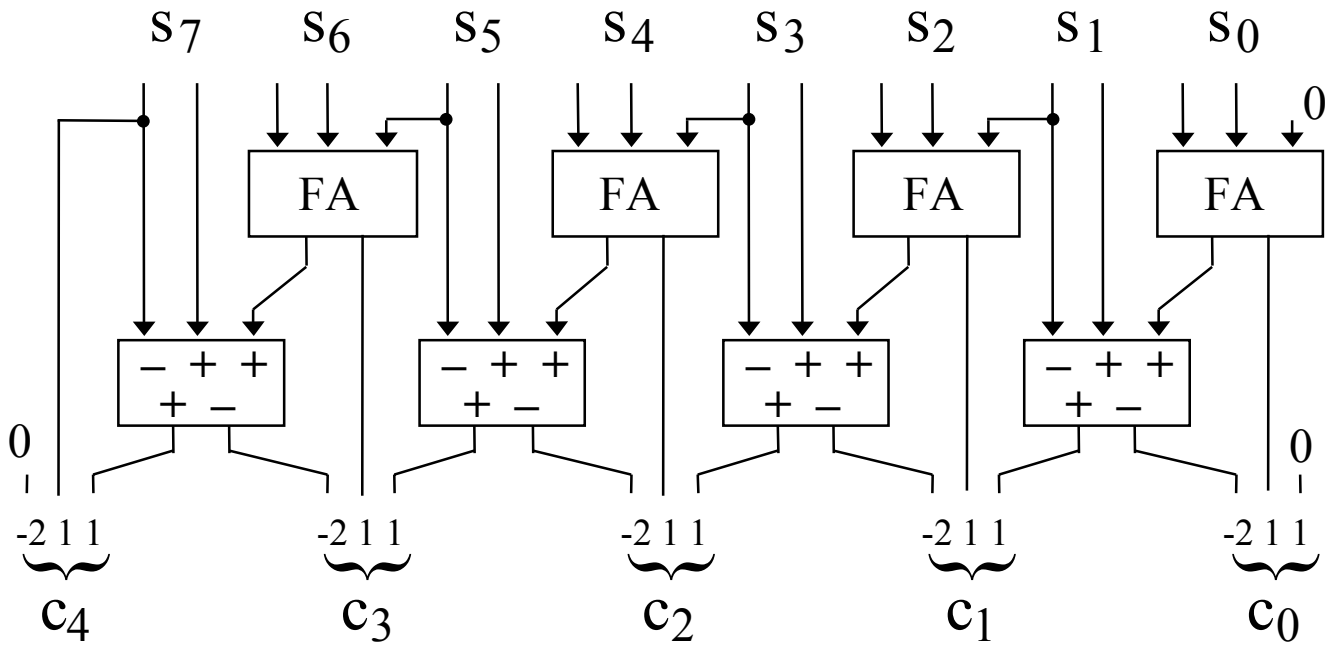
produit $P = A * B = \sum_{i=0}^{2n-1} p_i * 2^i$

Conversion de CS à code de Booth

Si le produit est un addende, un multiplieur ou un dividende, il n'est pas nécessaire de le convertir le « carry save » en notation conventionnelle.

$$P = \sum_{i=0}^{n-1} s_i * 2^i \quad \Rightarrow \quad P = \sum_{i=0}^{\frac{n-2}{2}} c_i * 4^i$$

$$s_i \in \{0,1,2\}$$
$$c_i \in \{-2,-1,0,1,2\}$$



La somme pondérée des chiffres qui entrent est égale à la somme pondérée des chiffres qui sortent

Arithmétique des Résidus

Alain GUYOT

Concurrent Integrated Systems
TIMA

(33) 04 76 57 46 16
Alain.Guyot@imag.fr
http://tima-cmp.imag.fr/Homepages/guyot
Techniques de l'Informatique et de la Microélectronique
pour l'Architecture. Unité associée au C.N.R.S. n° B0706

But

Opérateurs détectant les fautes

codes avec résidu modulo P

$\Rightarrow$ valeurs de P pratiquement utilisées $\in \{3, 7, 15, 31\}$

Opérateurs corrigeant les fautes

codes avec bi-résidu

$\Rightarrow$ pas d'applications pratiques (?)

Opérateurs rapides

processeurs de traitement de signal utilisant le code à résidu (RNS)

$\Rightarrow$ valeurs de P pratiquement utilisées telles que $\lceil \log_2 P \rceil \leq 12$

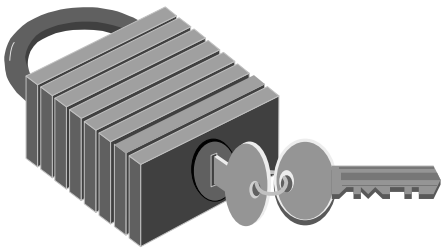
Opérateurs de Cryptographie

multiplication et exponentiation modulo P très grand

$\Rightarrow$ valeurs de P pratiquement utilisées: centaines à milliers de bits

Remarque:

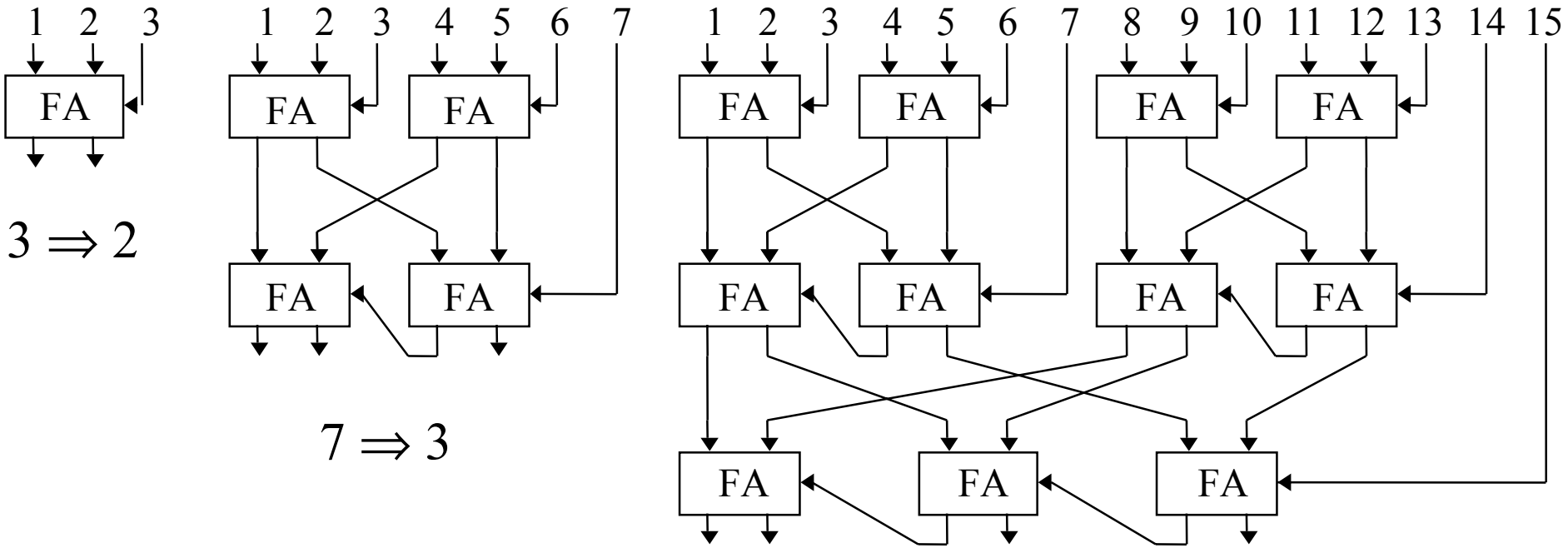
Dans cette partie nous nous limiterons à P petit



Arbre de Wallace

Compter les bits à 1 dans une chaîne

Tout assemblage cohérent de “FA” conserve la propriété:
La somme pondérée de ce qui sort est égale à la somme pondérée de ce qui entre.



Ce qui compte, c’est ce qu’on peut compter
Harpagon

Construction d'un arbre de Wallace

2^4	2^3	2^2	2^1	2^0	*poids des bits*
				22	*22 bits à réduire*
				7 FA	
			7	8	*15 bits restant à la 1ère étape*
			2 FA	2 FA	
		2	5	4	*11 bits restant à la 2eme étape*
			1 FA	1 FA	
		3	4	2	*9 bits restant à la 3eme étape*
		1 FA	1 FA	1 HA	
	1	2	3	1	*7 bits restant à la 4eme étape*
			1 FA		
	1	3	1	1	*6 bits restant à la 5eme étape*
		1 FA			
	2	1	1	1	*5 bits restant à la 6eme étape*
	1 HA				
1	1	1	1	1	*résultat de 5 bits* (22 $\Rightarrow$ 5)

Codes arithmétiques décimaux

Ces opérations sont elles correctes ?

391 * 972 = 390052

365481 + 44154 = 409535

365481 – 44154 = 331327

Soit $A = \sum_{i=0}^{n-1} a_i * 2^{10}$

A est multiple de 9 si ($a_{n-1} + a_{n-2} + \ldots + a_1 + a_0$) est multiple de 9.

A est multiple de 11
si $(-1)^{n-1}a_{n-1} + (-1)^{n-2}a_{n-2} + \ldots - a_1 + a_0$) est multiple de 11.

891 * 969 = 854379

Codes arithmétiques

On note $|A|_P$ le reste de la division entière de A par P.

$0 \leq |A|_P \leq P-1.$

Le nombre codé est $(A, |A|_P)$

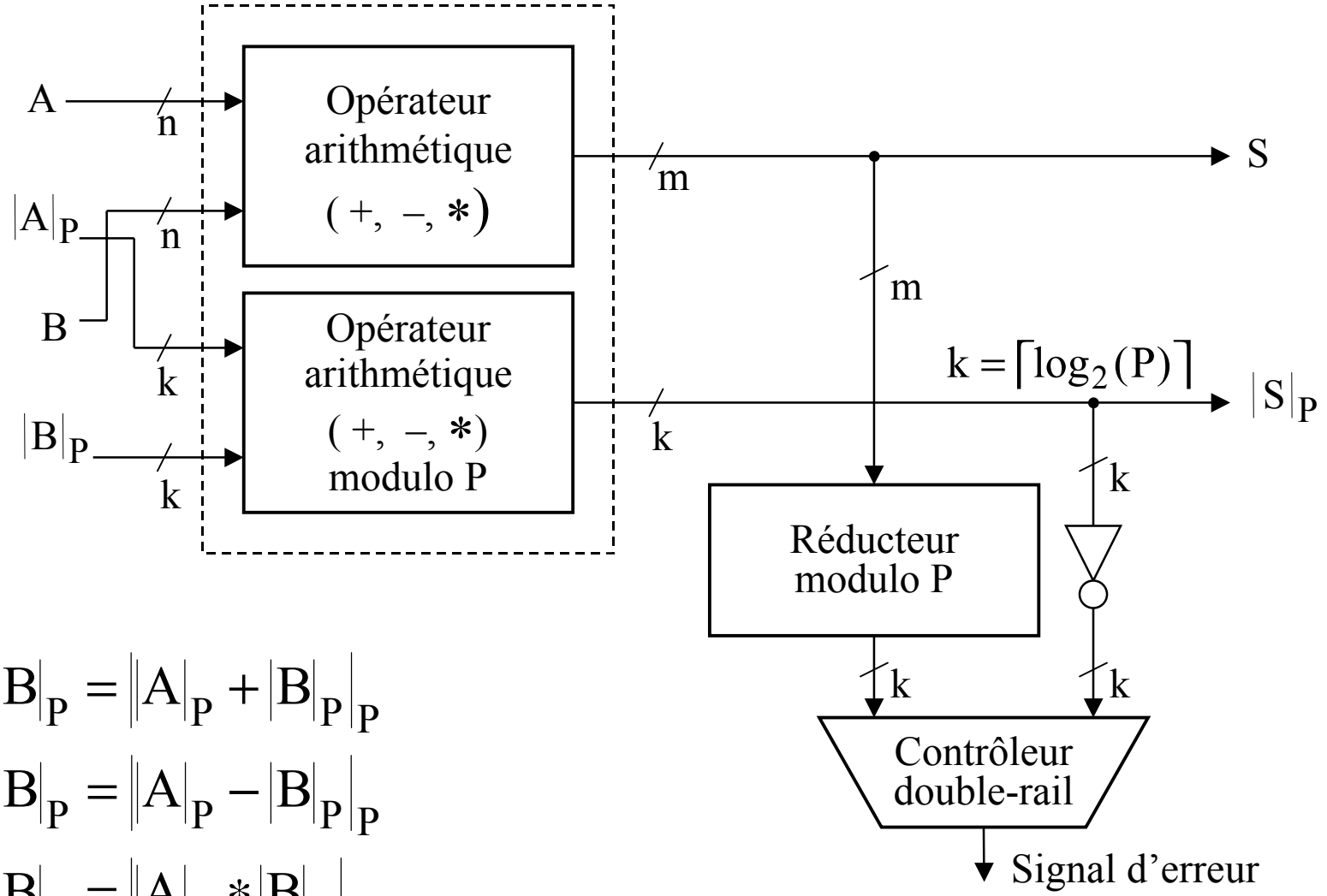
Pour les opérations d'addition, soustraction, multiplication, on a:

$$|A+B|_P = \left||A|_P + |B|_P\right|_P$$

$$|A-B|_P = \left||A|_P - |B|_P\right|_P$$

$$|A*B|_P = \left||A|_P * |B|_P\right|_P$$

Opérateurs arithmétiques auto-testables



$$|A+B|_P = ||A|_P + |B|_P|_P$$

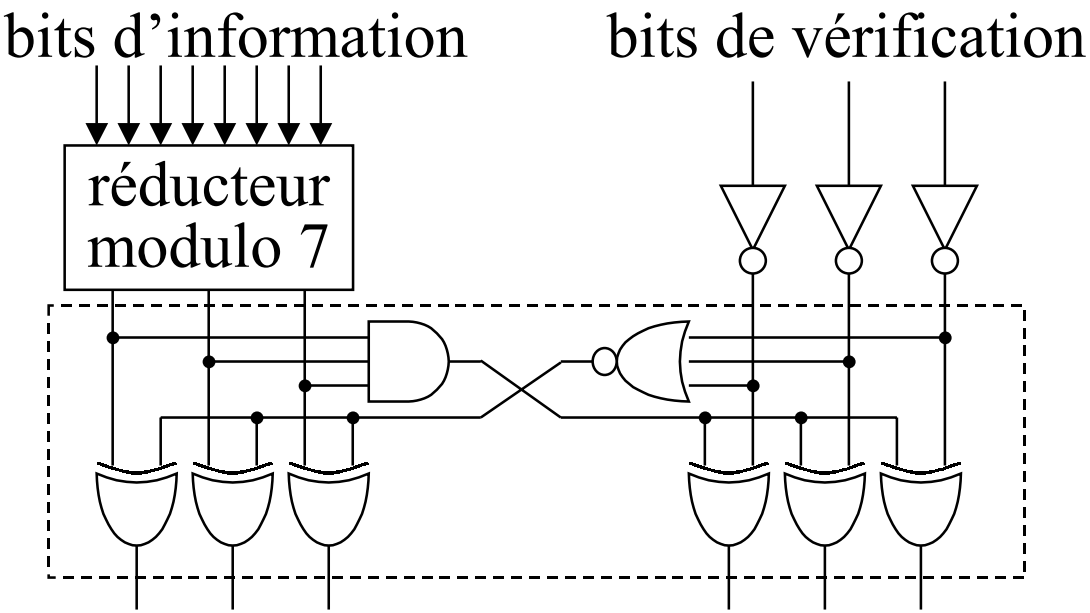
$$|A-B|_P = ||A|_P - |B|_P|_P$$

$$|A*B|_P = ||A|_P * |B|_P|_P$$

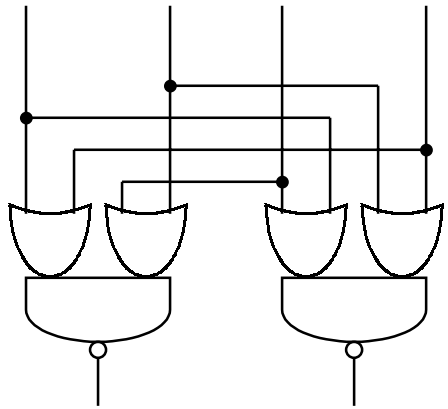
Contrôleur auto-testable

Un contrôleur auto-testable a au moins 2 sorties
En pratique on utilise le code "double rail" :

- 01 10 : opération correcte
- 00 11 : erreur



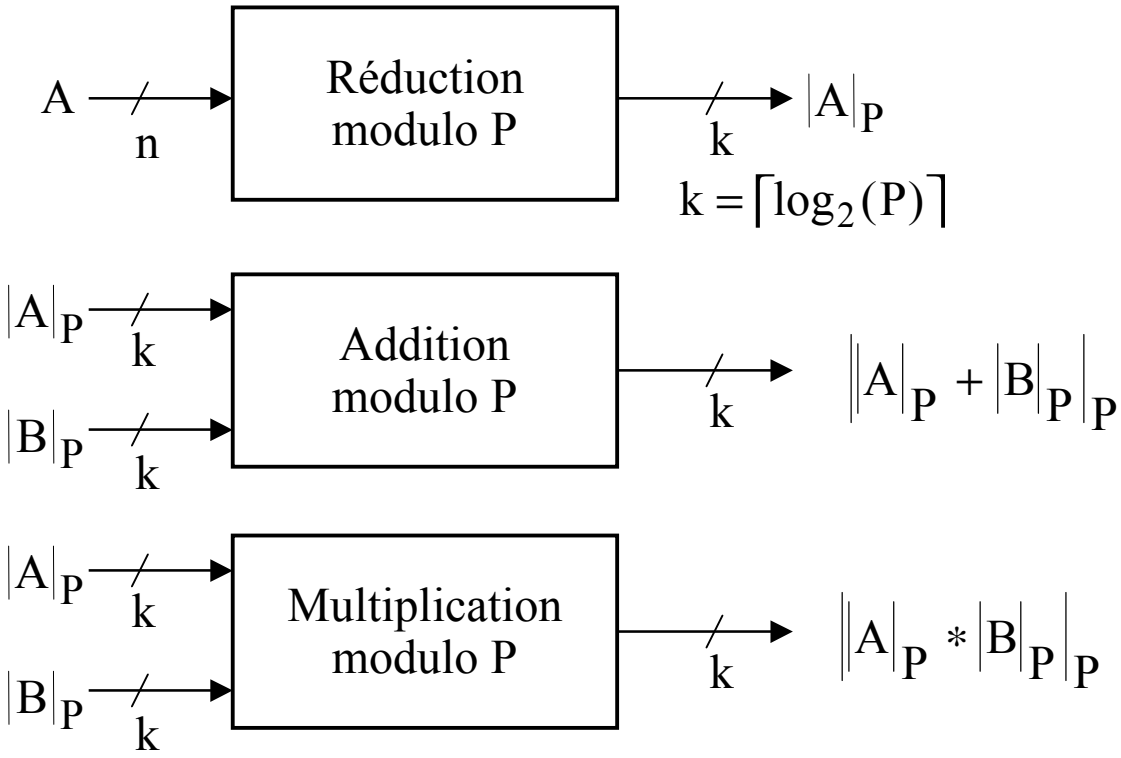
Convertisseur en code "double rail"



Porte "double rail"

Opérateurs modulo P

Réaliser les opérateurs suivants
en minimisant coût et délai



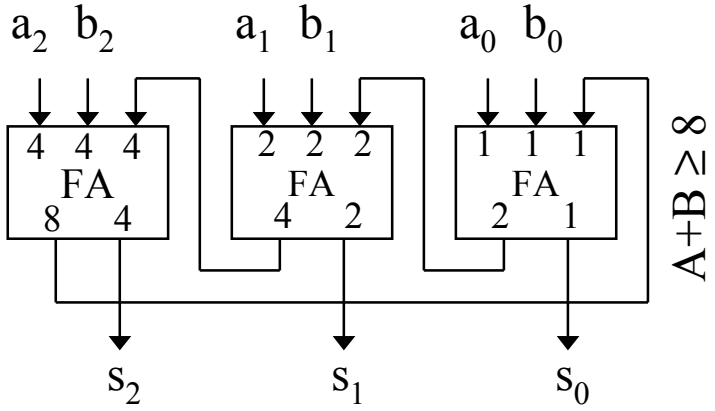
Poids des bits modulo P

$$\text{Soit } A = \sum_{i=0}^{11} a_i * 2^i$$

$$\text{Alors } |A|_P = \left| \sum_{i=0}^{11} a_i * |2^i|_P \right|_P$$

Poids des a_i	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
	2048	1024	512	256	128	64	32	16	8	4	2	1
$\|2^i\|_8$	0	0	0	0	0	0	0	0	0	4	2	1
$\|2^i\|_7$	4	2	1	4	2	1	4	2	1	4	2	1
$\|2^i\|_9$	-4	-2	-1	4	2	1	-4	-2	-1	4	2	1

Carry End around Adder



$$S = |A + B|_7$$

La valeur 0 de S a deux représentations: 000 ou 111

$$S = |A + B + c_0|_8$$

$$c_0 = c_3 = (A + B + c_0) \geq 8$$

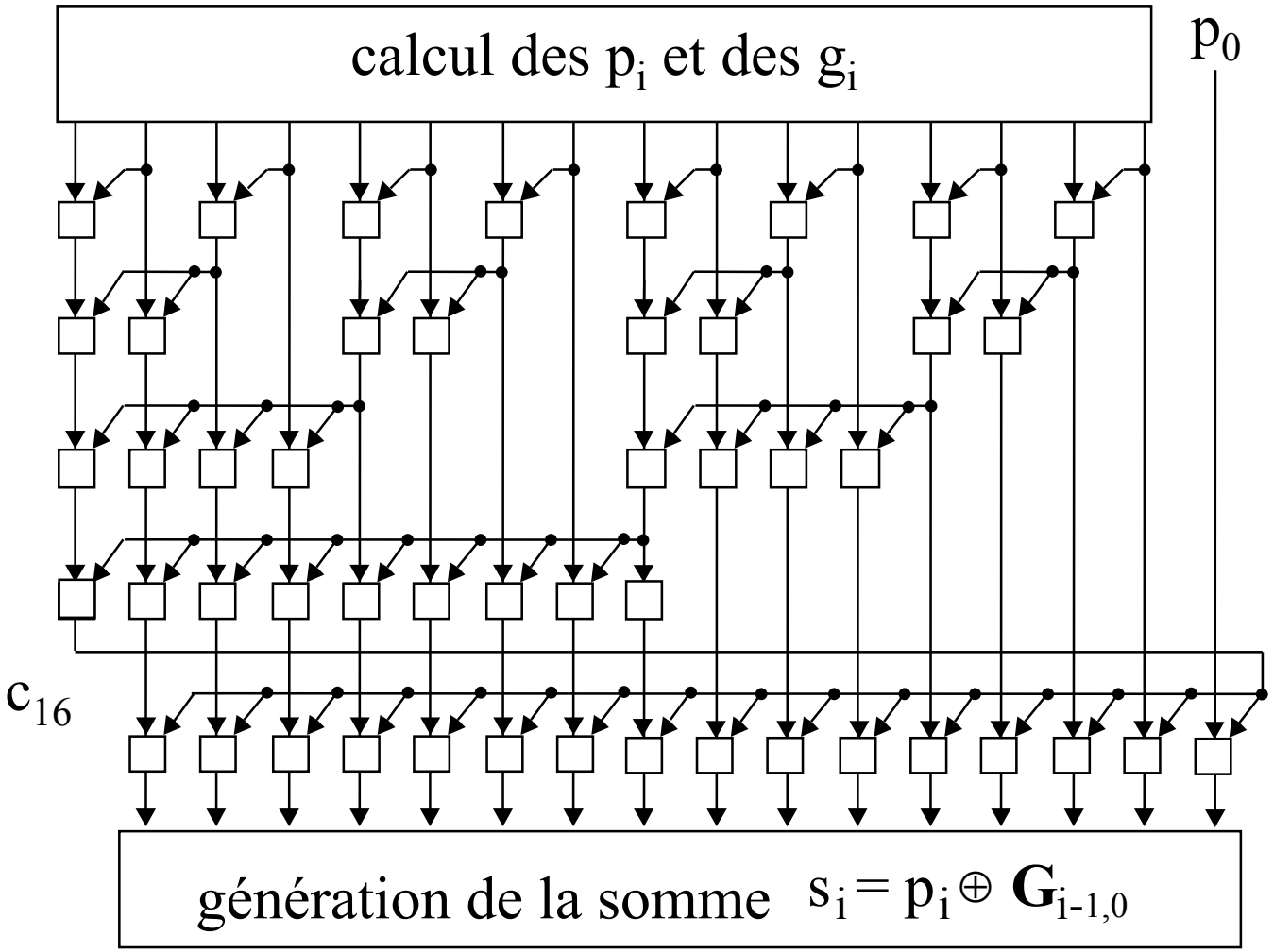
A+B	$\lvert A+B \rvert_7$
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	0 ou 7
8	1
9	2
10	3
11	4
12	5
13	6
14	7

Lorsque A + B = 7, S = 000 ou S = 111 suivant la valeur initiale de c_0

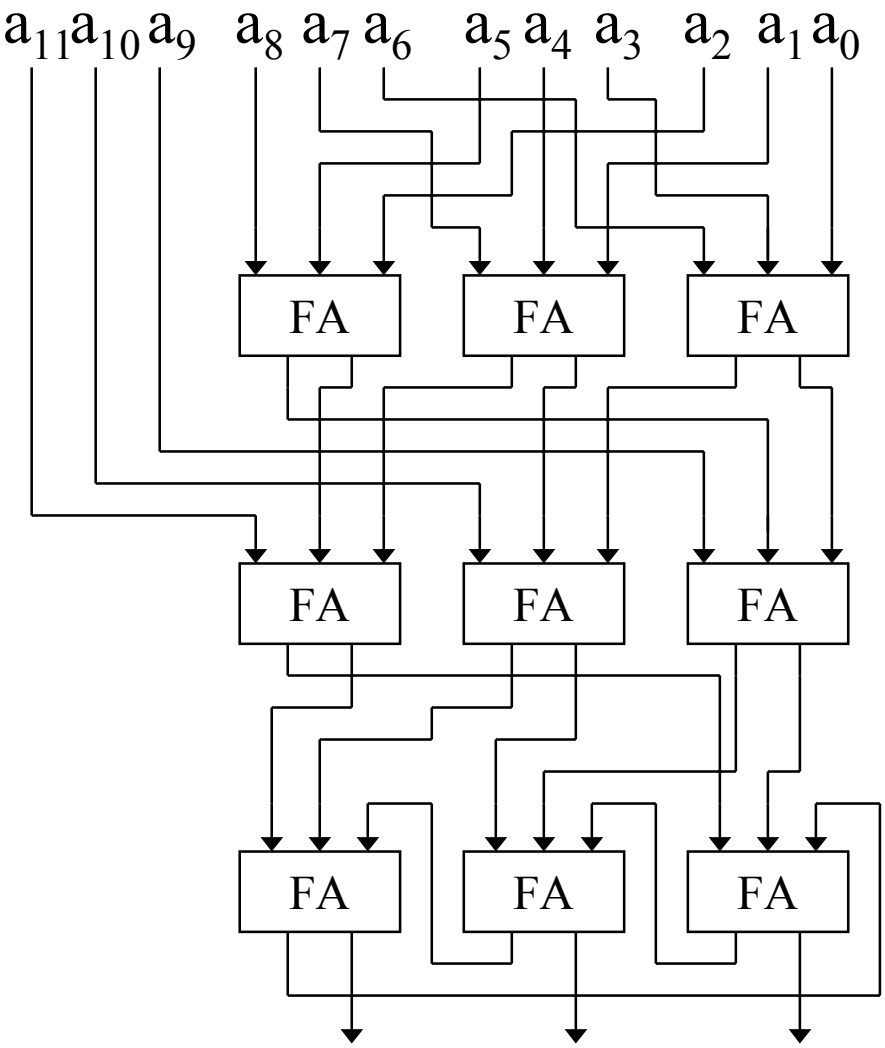
⇒ On admet une double représentation de la valeur 0,

⇒ il faut en tenir compte lors de la comparaison des résidus.

Carry End around Adder rapide



Réducteur modulo 7



Soit $A = \sum_{i=0}^{11} a_i * 2^i$

2^2	2^1	2^0	
4	4	4	*Bits à réduire*
FA	FA	FA	
3	3	3	*Bits 1ere étape*
FA	FA	FA	
2	2	2	*Bits 2eme étape*
FA ←	FA ←	FA ←	
1	1	1	*Bits 3eme étape*

Addition, soustraction, multiplication modulo 7

$$|A|_7 = \sum_{i=0}^{2} a_i * 2^i \qquad |B|_7 = \sum_{i=0}^{2} b_i * 2^i \qquad S = \sum_{i=0}^{2} s_i * 2^i$$

Bits à réduire

2^2	2^1	2^0
a_2 b_2	a_1 b_1	a_0 b_0

$$S = \left| |A|_7 + |B|_7 \right|_7$$

2^2	2^1	2^0
a_2 $\overline{b_2}$	a_1 $\overline{b_1}$	a_0 $\overline{b_0}$

$$S = \left| |A|_7 - |B|_7 \right|_7$$

$$-B = \overline{B} - 7$$

2^2	2^1	2^0
$a_0\ b_2$ $a_1\ b_1$ $a_2\ b_0$	$a_2\ b_2$ $a_0\ b_1$ $a_1\ b_0$	$a_1\ b_2$ $a_2\ b_1$ $a_0\ b_0$

$$S = \left| |A|_7 * |B|_7 \right|_7$$

Addition, soustraction, multiplication modulo 7

$$|A|_7 = \sum_{i=0}^{2} a_i * 2^i \qquad |B|_7 = \sum_{i=0}^{2} b_i * 2^i \qquad S = \sum_{i=0}^{2} s_i * 2^i$$

Tables de Pithagore

$$S = \left| |A|_7 + |B|_7 \right|_7 \qquad S = \left| |A|_7 - |B|_7 \right|_7 \qquad S = \left| |A|_7 * |B|_7 \right|_7$$

	0	1	2	3	4	5	6	7
0	Z	1	2	3	4	5	6	Z
1	1	2	3	4	5	6	Z	1
2	2	3	4	5	6	Z	1	2
3	3	4	5	6	Z	1	2	3
4	4	5	6	Z	1	2	3	4
5	5	6	Z	1	2	3	4	5
6	6	Z	1	2	3	4	5	6
7	Z	1	2	3	4	5	6	Z

	0	1	2	3	4	5	6	7
0	Z	6	5	4	3	2	1	Z
1	1	Z	6	5	4	3	2	1
2	2	1	Z	6	5	4	3	2
3	3	2	1	Z	6	5	4	3
4	4	3	2	1	Z	6	5	4
5	5	4	3	2	1	Z	6	5
6	6	5	4	3	2	1	Z	6
7	Z	6	5	4	3	2	1	Z

	0	1	2	3	4	5	6	7
0	Z	Z	Z	Z	Z	Z	Z	Z
1	Z	1	2	3	4	5	6	Z
2	Z	2	4	6	1	3	5	Z
3	Z	3	6	2	5	1	4	Z
4	Z	4	1	5	2	6	3	Z
5	Z	5	3	1	6	4	2	Z
6	Z	6	5	4	3	2	1	Z
7	Z	Z	Z	Z	Z	Z	Z	Z

Z = 0 ou 7 (double représentation)

Exercices

1- Combien faut il de de bits pour exprimer le nombre de bits à 1 dans une chaîne de n bits ?

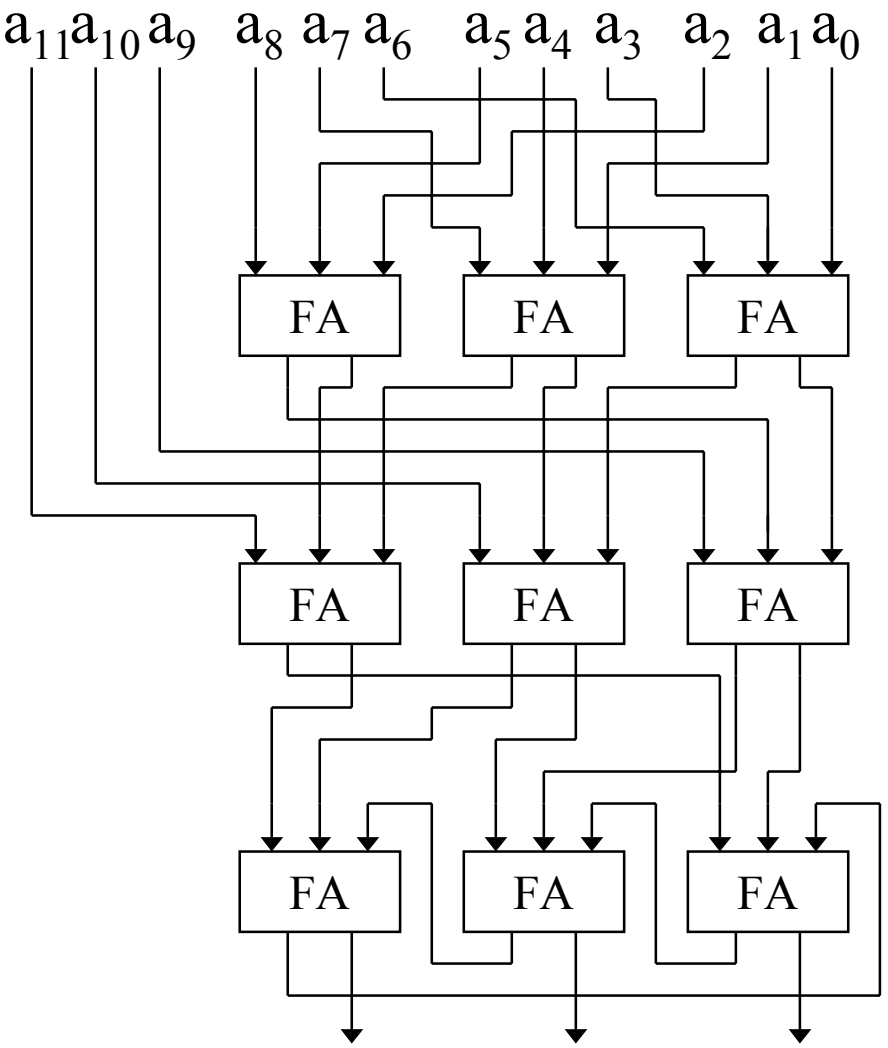
2- Combien faut il de cellule FA (avec toutes les entrées utilisées) pour compter les bits à 1 dans une chaîne de n bits ?

3- Donnez le circuit qui calcule le reste modulo 7 d'un entier signé représenté sur 13 bits:

$$A = -a_{12}\, 2^{12} + \sum_{i=0}^{11} a_i\, 2^i$$

4- Donner le circuit calculant la somme de deux nombres en "signe, valeur absolue"

Réducteur modulo 9



Soit $A = \sum_{i=0}^{11} a_i * 2^i$

2^2	2^1	2^0	
4	4	4	*Bits à réduire*
FA	FA	FA	
3	3	3	*Bits 1ere étape*
FA	FA	FA	
2	2	2	*Bits 2eme étape*
FA ←	FA ←	FA ←	
1	1	1	*Bits 3eme étape*

Périodicité (1)

La réduction modulo 7 est simple car $\left|2^{3k}\right|_7 = 1, \left|2^{3k+1}\right|_7 = 2, \left|2^{3k+2}\right|_7 = 4$

Définition: la période de P est le plus petit entier j tel que $\left|2^{j}\right|_P = 1$

Définition: la demi-période de P est le plus petit entier j tel que $\left|2^{j}\right|_P = -1$

P	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33
demi période	1	2	-	3	5	6	-	4	9	-	-	10	9	14	-	5
période	2	4	3	6	10	12	4	8	18	6	11	20	18	28	5	10

P	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63	65
demi période	-	18	-	10	7	-	-	-	-	26	-	9	29	30		6
période	12	36	12	20	14	12	23	21	8	52	20	18	58	60	6	12

Modulo P (P impair quelconque)

La demi-période n'existe pas pour tous les P, mais si elle existe elles vaut la moitié de la période, d'où son nom.

Le calcul du reste modulo P commence par une réduction sur un nombre de bits égal à la demi-période de P ou à défaut la période de P.

Division Matérielle

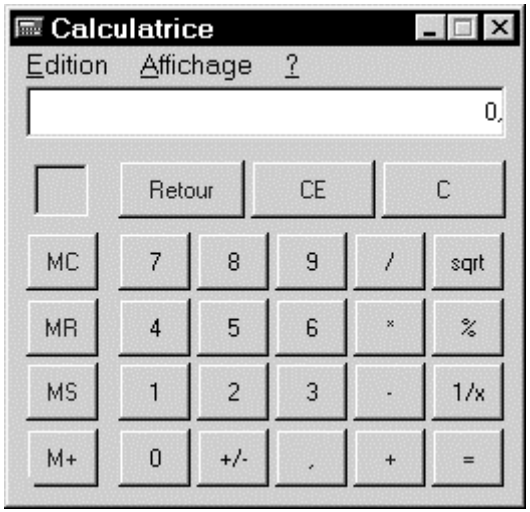
Alain GUYOT

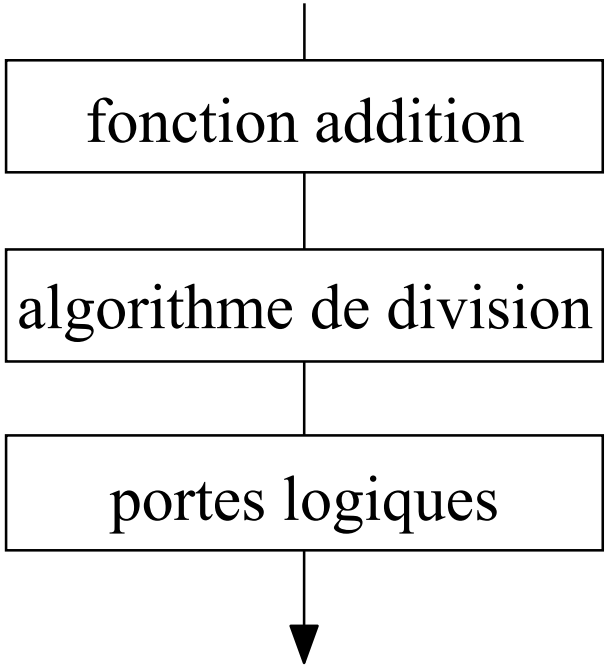
Concurrent Integrated Systems
TIMA

(33) 04 76 57 46 16
Alain.Guyot@imag.fr
http://tima-cmp.imag.fr/Homepages/guyot
Techniques de l'Informatique et de la Microélectronique pour l'Architecture. Unité associée au C.N.R.S. n° B0706

But
Réaliser des diviseurs combinatoires rapides

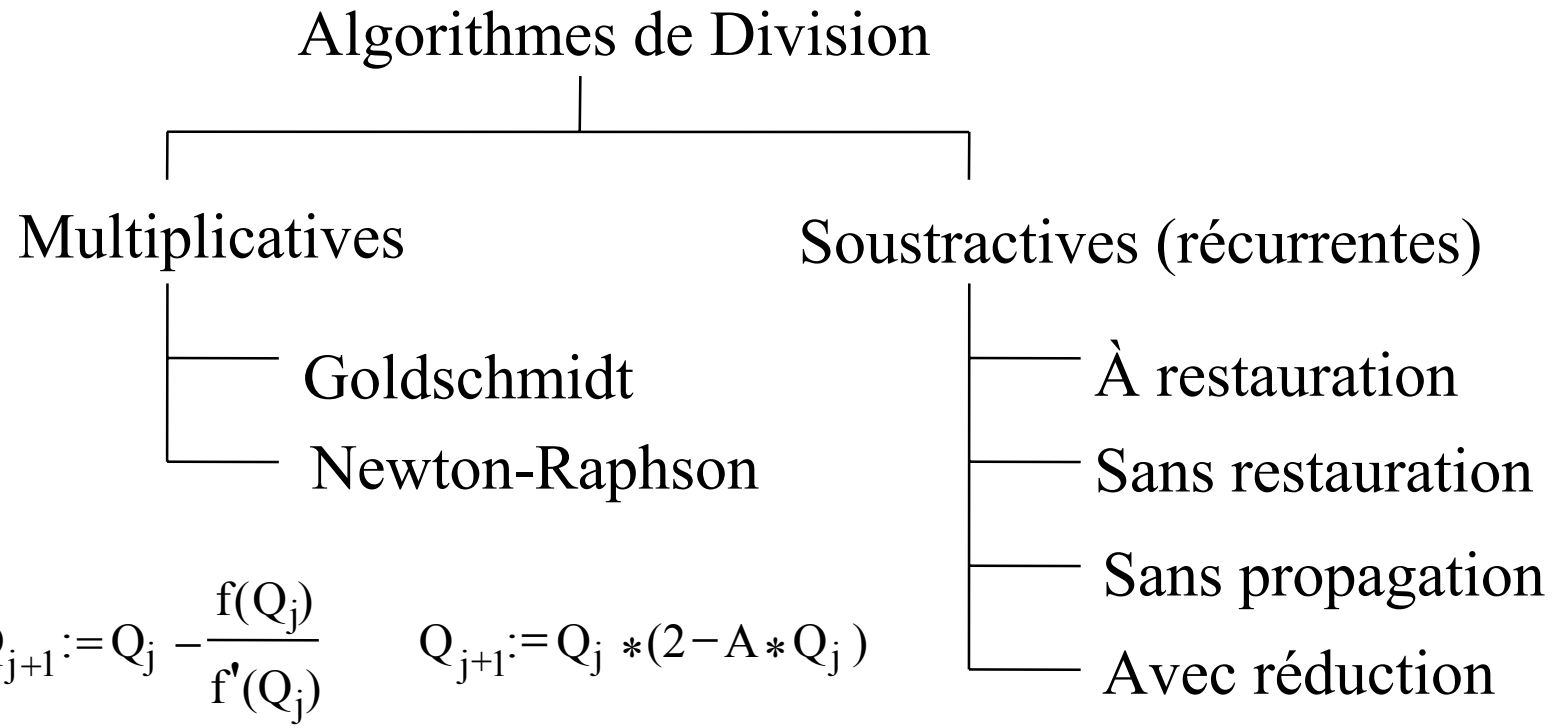
Problème
- Propagation de la retenue





Moyen
Utiliser des additionneurs sans propagation de retenue

Généralités / Plan



La division de deux entier n'est en général pas un entier,
en conséquence on introduit les rationnels (virgule fixe)

$$A = \sum_{i=0}^{n-1} a_i 2^{-i} = a_0, a_1 a_2 a_3 a_4 .... a_{n-2} a_{n-1} \qquad -A = \overline{A} + 2^{-n+1}$$

Division récurrente: principes

On veut calculer $Q = \frac{A}{D}$.

On va construire une suite $Q_0, Q_1, Q_2, \ldots Q_n$ et une suite $R_0, R_1, R_2, \ldots R_n$ telles que l'invariant $A = Q_j * D + R_j$ soit respecté $\forall$ j.

La récurrence est : $Q_{j+1} = Q_j + q_{j+1} * 2^{-j-1}$

$R_{j+1} = R_j - q_{j+1} * D * 2^{-j-1}$

avec comme état initial: $Q_0 = 0$

$R_0 = A$

Quand on s'arrête, on a $Q_n = \sum_{i=0}^{n} q_i * 2^{-i}$

On impose que le choix des q_j soit tel que $R_j \rightarrow 0$ quand $j \rightarrow \infty$.

On aura donc une approximation Q_n de Q telle que $Q_n = \frac{A - R_n}{D}$ avec R_n petit.

Comme la valeur de Q est bornée par l'implémentation ($|Q| < 2$),
il faut que $-2 * D < A < 2 * D$ (D *doit être suffisamment grand*)

Exemple de division récurrente en décimal

On veut calculer $Q = \frac{22}{7}$.

Calcul des restes	Valeurs	Calcul des quotients	Valeurs
$R_0 := 22$	22	$Q_0 := 0$	
$R_1 := R_0 - 3 * D$	01,0	$Q_1 := Q_0 + 3$	3
$R_2 := R_1 - 0,1 * D$	00,30	$Q_2 := Q_1 + 0,1$	3,1
$R_3 := R_2 - 0,04 * D$	00,020	$Q_3 := Q_2 + 0,04$	3,14
$R_4 := R_3 - 0,002 * D$	00,0060	$Q_4 := Q_3 + 0,002$	3,142
$R_5 := R_4 - 0,0008 * D$	00,00040	$Q_5 := Q_4 + 0,0008$	3,1428
$R_6 := R_5 - 0,00005 * D$	00,000050	$Q_6 := Q_5 + 0,00005$	3,14285

On vérifie que

22 - 0,3 = 21,7 = 7 * 3,1
22 - 0,02 = 21,98 = 7 * 3,14
22 - 0,006 = 21,994 = 7 * 3,142
22 - 0,0004 = 21,9996 = 7 * 3,1428
22 - 0,00005 = 21,99995 = 7 * 3,14285

Exemple de division récurrente en binaire

On veut calculer $Q = \frac{10110}{111}$.

L'algorithme de division en base 2 est une transposition de l'algorithme en base 10.

Calcul des restes	Valeurs reste	Calcul des quotients	Valeurs du quotient	Valeurs du quotient
$R_0 := 10110$	10110	$Q_0 := 0$		
$R_1 := R_0 - 10 * D$	01000	$Q_1 := Q_0+10$	10	*2*
$R_2 := R_1 - 1 * D$	00001	$Q_2 := Q_1 + 1$	11	3
$R_3 := R_2 - 0{,}0 * D$	00001,0	$Q_3 := Q_2 + 0{,}0$	11,0	3
$R_4 := R_3 - 0{,}00 * D$	00001,00	$Q_4 := Q_3 + 0{,}00$	11,00	3
$R_5 := R_4 - 0{,}001 * D$	00000,001	$Q_5 := Q_4 + 0{,}001$	11,001	3,1*25*
$R_6 := R_5 - 0{,}0000 * D$	00000,0010	$Q_6 := Q_5 + 0{,}0000$	11,0010	3,1*25*
$R_7 := R_6 - 0{,}00000 * D$	00000,00100	$Q_7 := Q_6 + 0{,}00000$	11,00100	3,1*25*
$R_8 := R_7 - 0{,}000001 * D$	00000,000001	$Q_8 := Q_7 + 0{,}000001$	11,001001	3,14*0625*

3,142 *578125*

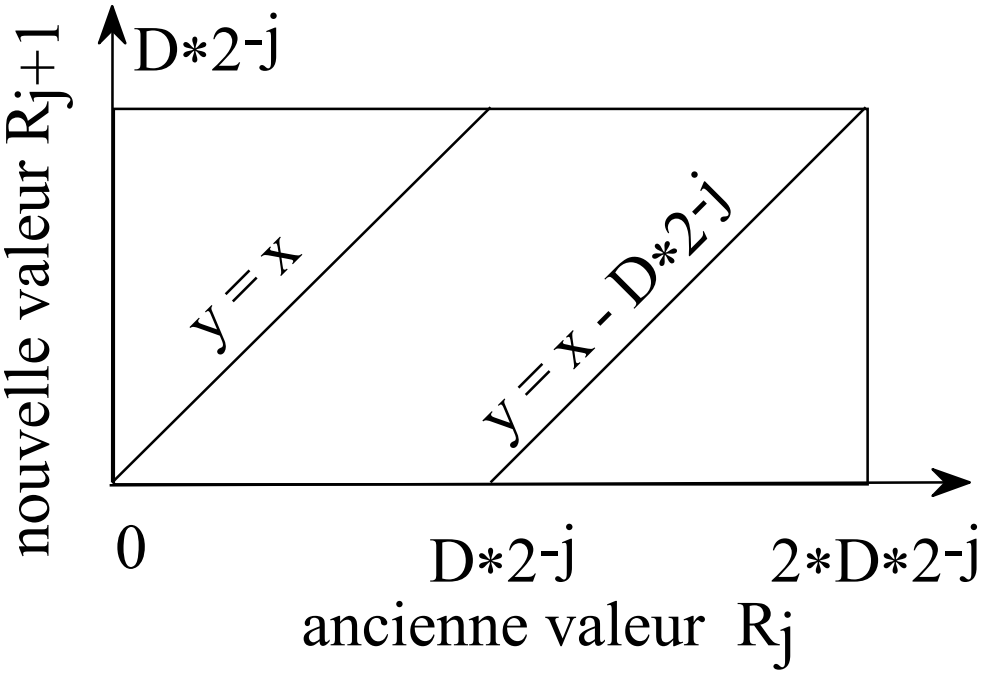
3,1428 *22265*

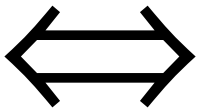
3,14285*2783*

Diagramme de « Robertson »

(diviseur naïf ou à restauration)

Invariant: $0 \leq R_j \leq 2*D*2^{-j}$





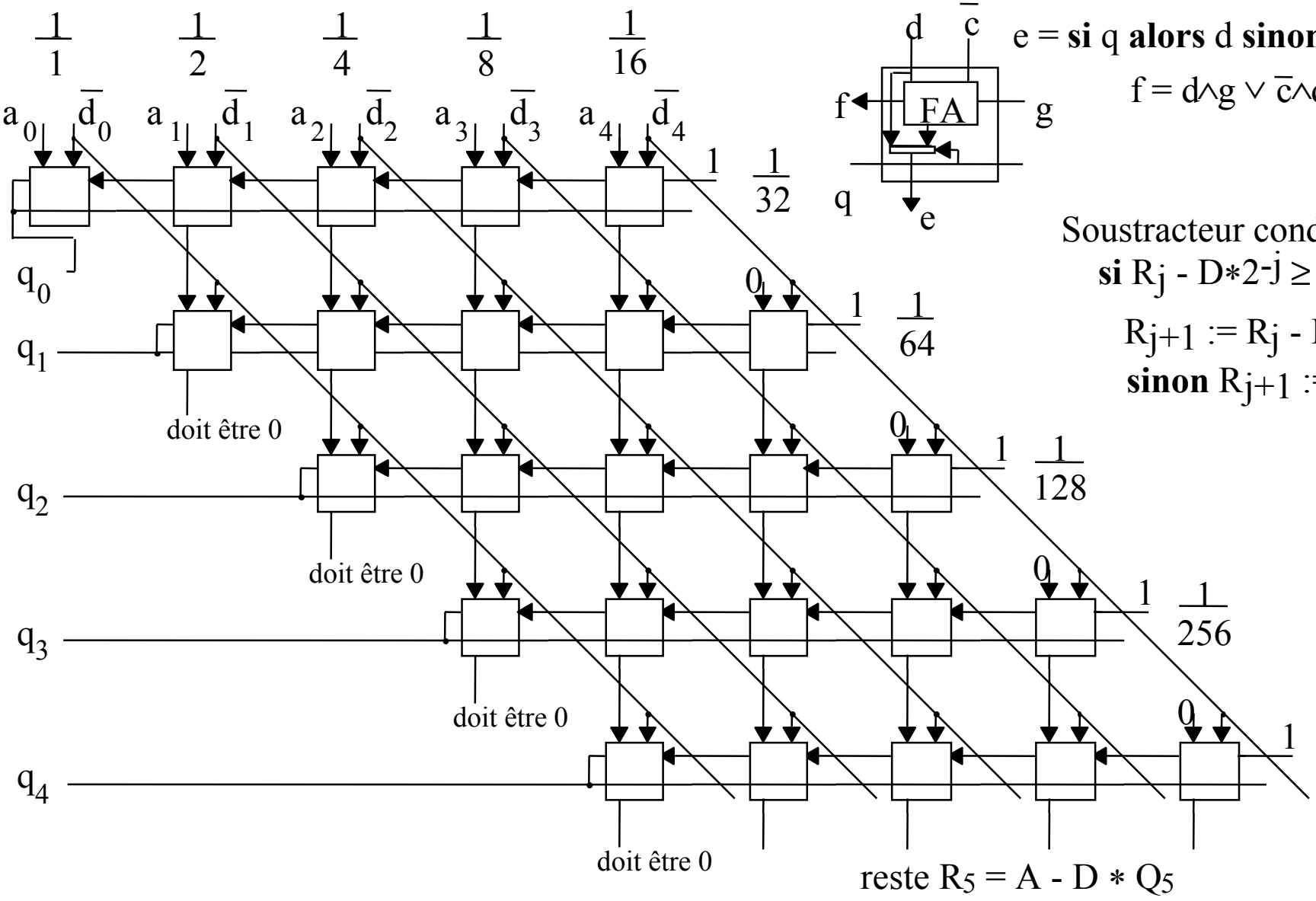
si $R_j \geq D*2^{-j}$ **alors**

$R_{j+1} := R_j - D*2^{-j}$

sinon $R_{j+1} := R_j$;

Récurrence

Diviseur naïf (à restauration)

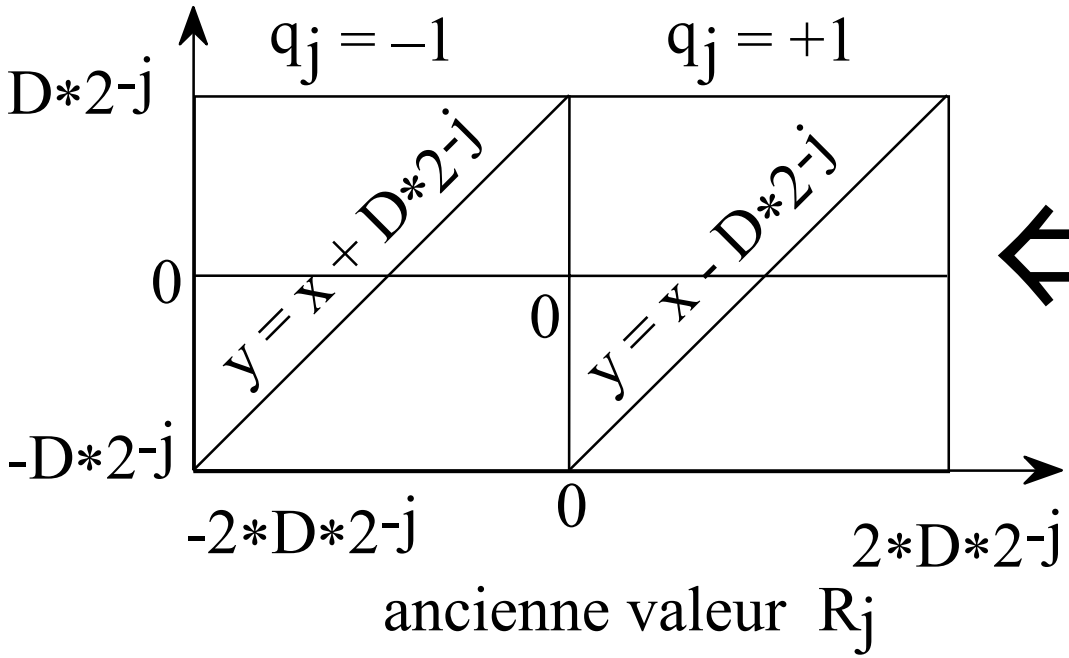


Exemple de division récurrente sans restauration

Calcul des restes	Valeurs du reste (signé)		Calcul des quotients	Valeurs
$R_0 := 10110$	10110	22	$Q_0 := 0$	
$R_1 := R_0 - 10 * D$	$\bar{0}1000$	8	$Q_1 := Q_0 + 10$	10
$R_2 := R_1 - 1 * D$	$0\bar{0}001$	1	$Q_2 := Q_1 + 1$	11
$R_3 := R_2 - 0,1 * D$	$00\bar{1}01,1$	-2,5	$Q_3 := Q_2 + 0,1$	11,1
$R_4 := R_3 + 0,01 * D$	$000\bar{1}1,01$	-0,75	$Q_4 := Q_3 - 0,01$	11,01
$R_5 := R_4 + 0,001 * D$	$0000\bar{0},001$	0,125	$Q_5 := Q_4 - 0,001$	11,001
$R_6 := R_5 - 0,0001 * D$	$00000,\bar{1}011$	-0,3125	$Q_6 := Q_5 + 0,0001$	11,0011
$R_7 := R_6 + 0,00001 * D$	$00000,0\bar{1}101$	-0,09375	$Q_7 := Q_6 - 0,00001$	11,00101
$R_8 := R_7 + 0,000001 * D$	$00000,00\bar{0}001$	0,015625	$Q_8 := Q_7 - 0,000001$	11,001001

Division sans restauration

Invariant: $-2*D*2^{-j} \leq R_j \leq 2*D*2^{-j}$



$\Leftrightarrow$

si $R_j \geq 0$ **alors**

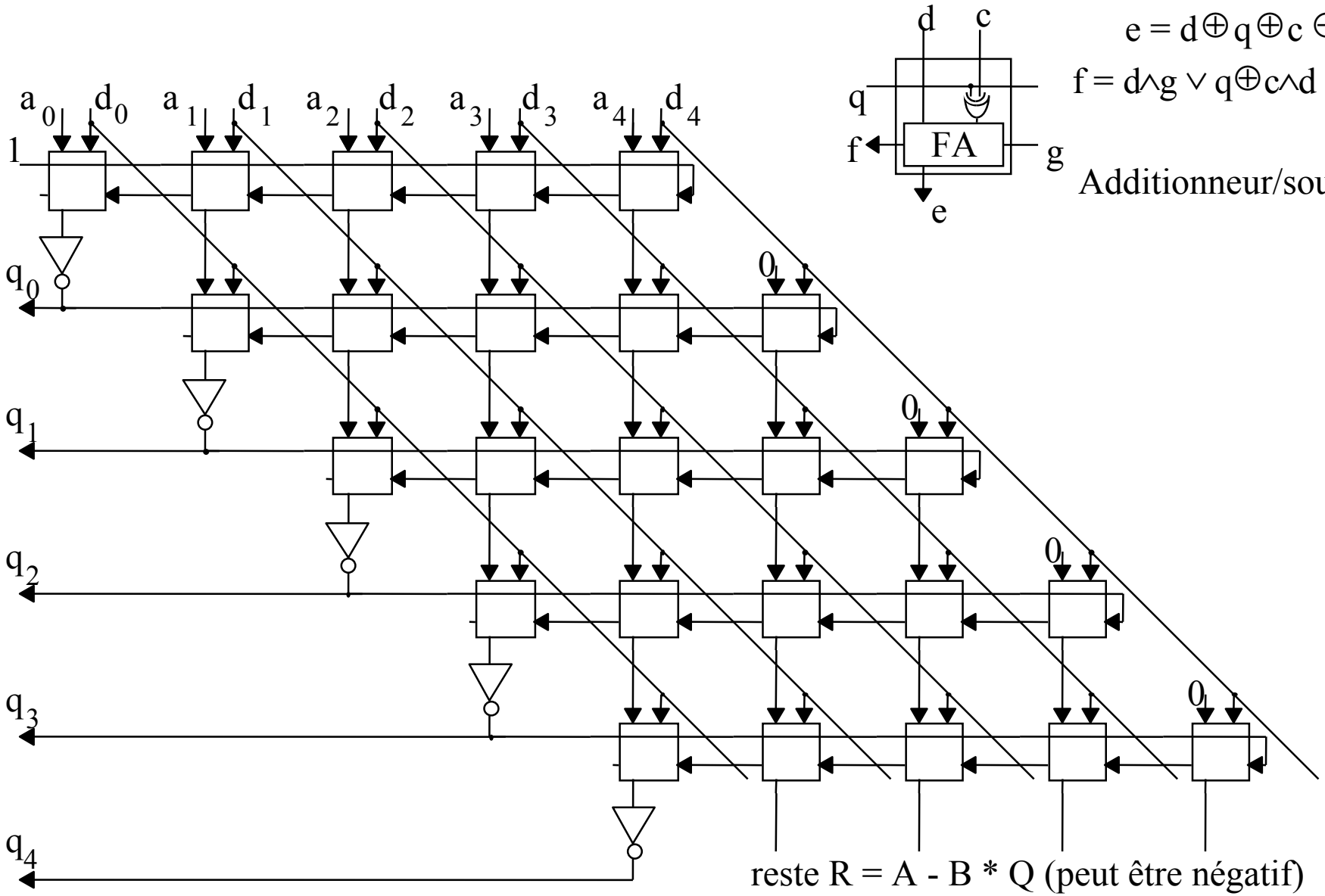
$R_{j+1} := R_j - D*2^{-j}$

sinon

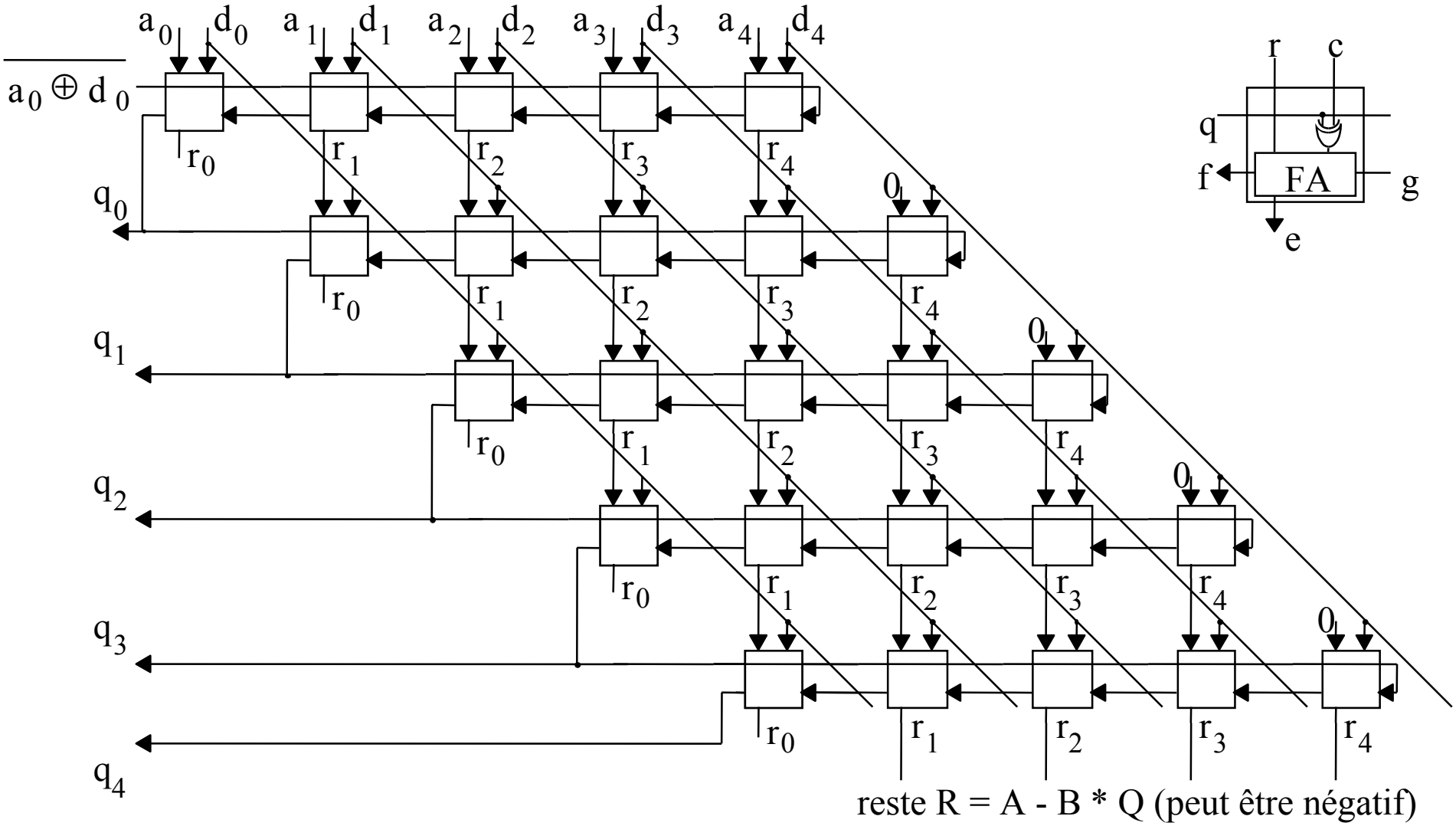
$R_{j+1} := R_j + D*2^{-j}$;

Récurence

Diviseur sans restauration (2)



Division signée



Conversion du quotient

Le quotient Q s'écrit $Q = \sum_{i=0}^{n-1} q_i * 2^{-i}$ avec $q_i \in \{-1,+1\}$. Pour le convertir on le réécrit:

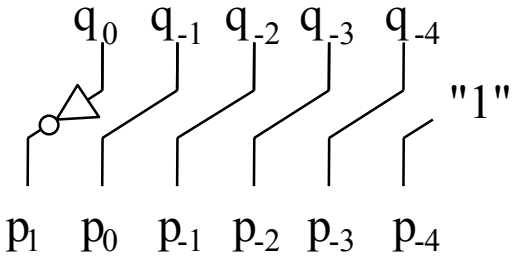
$$Q = \sum_{i=0}^{n-1} q_i * 2^{-i} = \sum_{i=0}^{n-1} q_i * 2^{-i} + \sum_{i=0}^{n-1} 2^{-i} - \sum_{i=0}^{n-1} 2^{-i}$$

$$" \quad = \sum_{i=0}^{n-1} (q_i + 1) * 2^{-i} - \sum_{i=0}^{n-1} 2^{-i}$$

$$" \quad = 2 * \sum_{i=0}^{n-1} p_i * 2^{-i} - 2 + 2^{-n}$$

$$" \quad = 2 * \left(p_0 - 1 + \sum_{i=1}^{n-1} p_i * 2^{-i} + 2^{-n-1} \right)$$

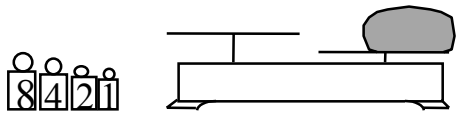
$$" \quad = 2 * \left(\underbrace{-\overline{p_0} * 2^0 + \sum_{i=1}^{n-1} p_i * 2^{-i}}_{\text{Ceci est la notation en complément à 2} \qquad p_i \in \{0,1\}} + 2^{-n-1} \right)$$

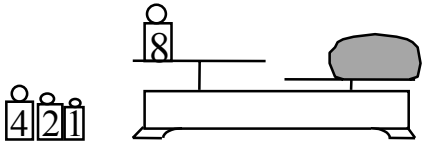


La conversion ne change pas la valeur de Q mais seulement sa représentation sous forme de chaîne de bits

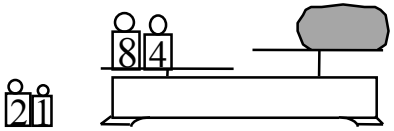
Remarque: si on connaît a priori le signe du résultat, l'inverseur n'est même pas nécessaire

Pesée à restauration

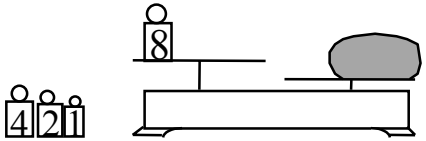




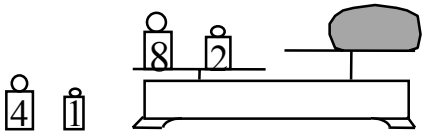
On ajoute 8,
ça va



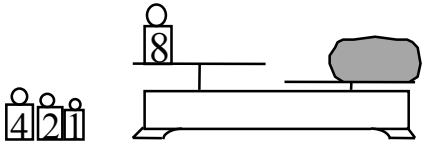
On ajoute 4,
c'est trop



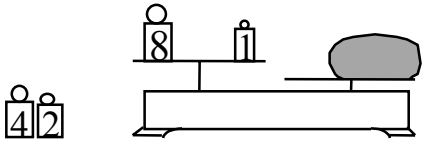
On enlève 4
(restauration)



On ajoute 2,
c'est trop

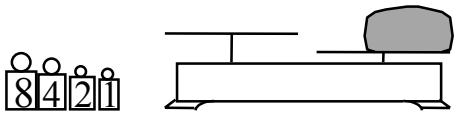


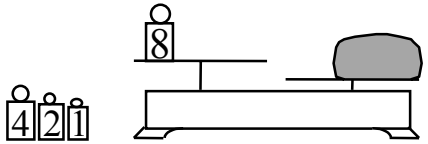
On enlève 2
(restauration)



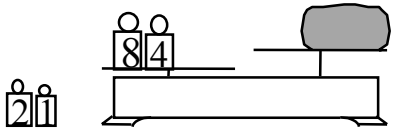
On essaye 1,
ça va

Pesée sans restauration

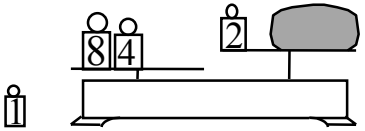




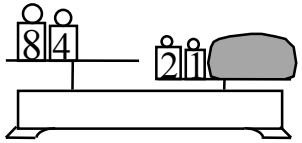
On ajoute 8
sur le plus haut



On ajoute 4
sur le plus haut



On ajoute 2
sur le plus haut



On ajoute 1
sur le plus haut

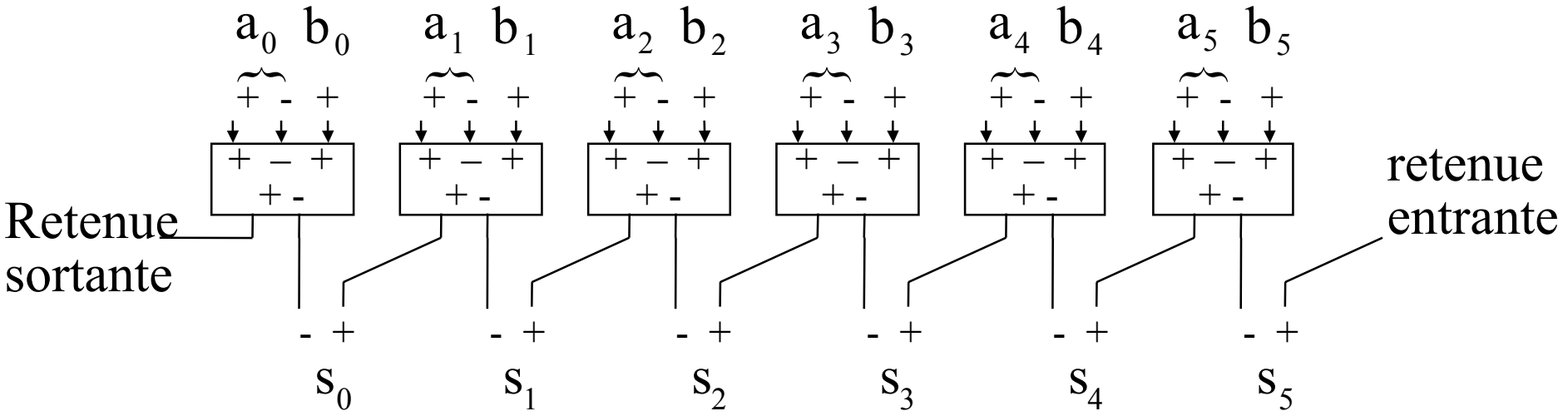
On se sert de tous les poids
On pourrait peser un poids "négatif"

Le "BS", mais c'est très simple

$$A = A^{+} - A^{-} = \sum_{i=0}^{n-1} a_i^{+} 2^{-i} - \sum_{i=0}^{n-1} a_i^{-} 2^{-i} = \sum_{i=0}^{n-1} (a_i^{+} - a_i^{-}) 2^{-i} = \sum_{i=0}^{n-1} a_i 2^{-i}$$

$$a_i^{+}, a_i^{-} \in \{0, 1\} \qquad a_i \in \{-1, 0, 1\}$$

Addition hybride $S = A + B$ $\qquad a_i, s_i \in \{-1, 0, 1\} \quad b_i \in \{0, 1\}$



Avec ou Sans propagation

Type d'opération — Propagation de Retenue

Additionner de deux nombres en BS ----------
(A + B) ⇒ S

Avec ❑ Sans ❑

Déterminer le signe d'un nombre en BS --------
(0 0 0 $\bar{1}$ ≥ 0 ?)

Avec ❑ Sans ❑

Forcer à 0 les poids forts d'un nombre petit -----
1 $\bar{1}$ $\bar{1}$ 0 ⇒ 0 0 1 0

Avec ❑ Sans ❑

Convertir de notation BS à Standard ----------
0 1 0 $\bar{1}$ ⇒ 0 0 1 1

Avec ❑ Sans ❑

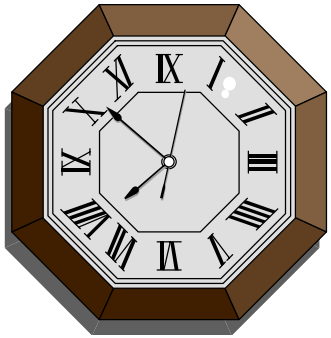
¿¿ Utilité du BS ??

¿¿¿ Comment utiliser cette [illegible] de notation ???

- Sans comparaison
- Sans élimination de chiffres non significatifs
- Avec un coût de conversion exorbitant

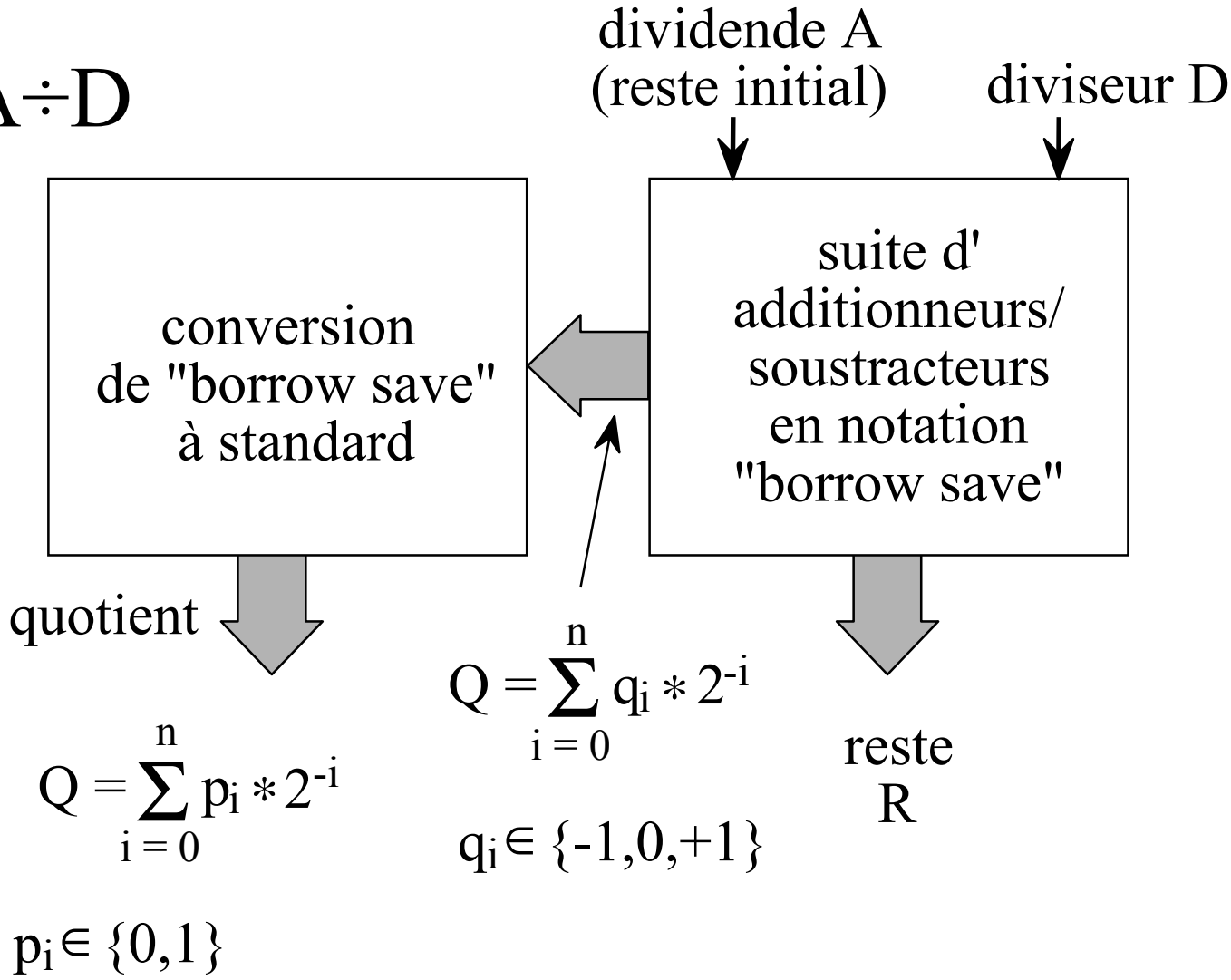
Dans un algorithme comme la DIVISION RAPIDE

Divide ut regnes Machiavel

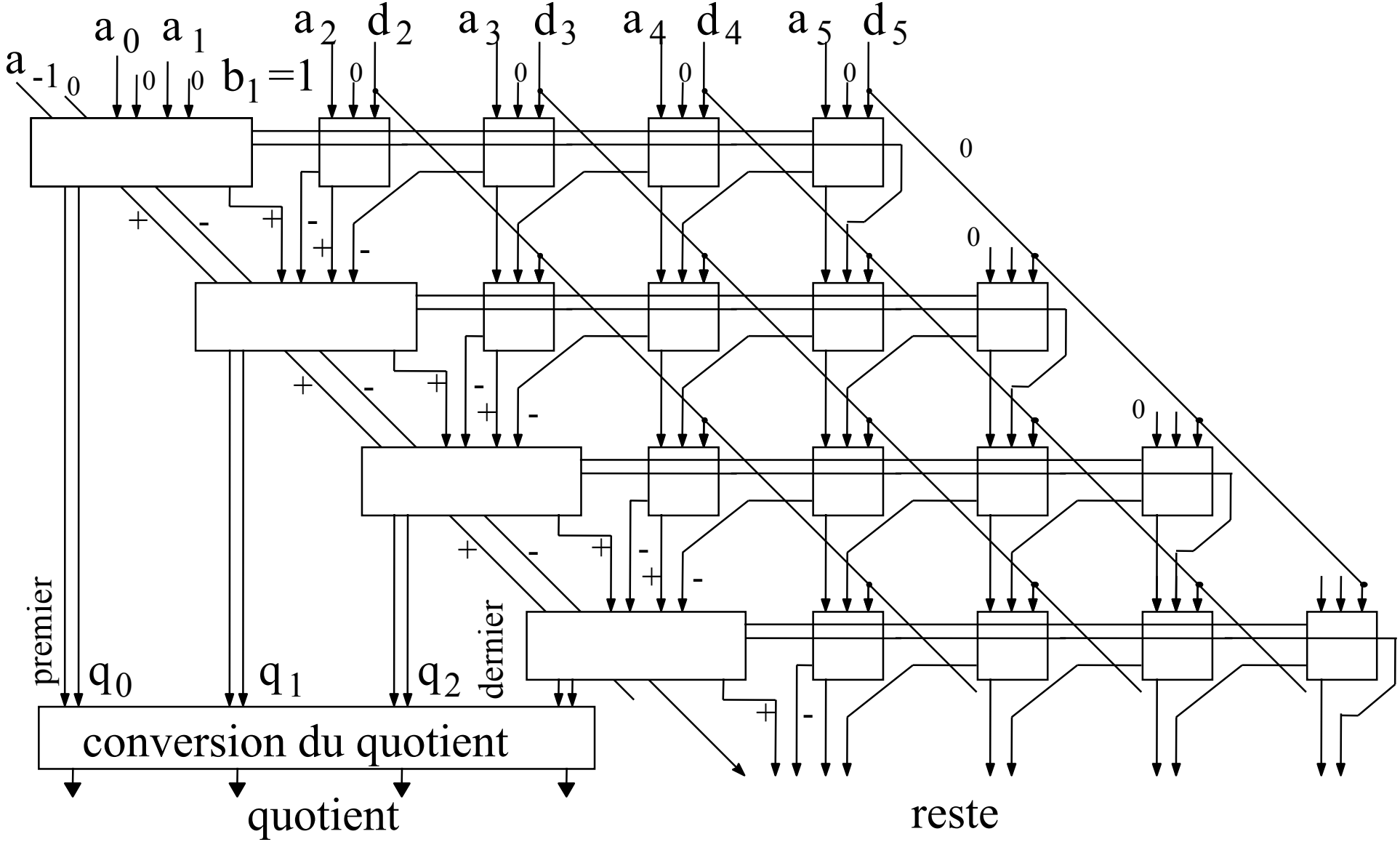


Diviseur en notation "BS"

$Q := A \div D$



Diviseur régulier en notation "borrow save"



Format du reste partiel R_j

On suppose D normalisé $1 \le D < 2$: $D = \sum_{i=0}^{n} d_i 2^{-i}$ $d_0 = 1$

Donc $-4 < -2D \le R_j * 2^j \le +2D < 4$.

Pour être additionné ou soustrait rapidement, R_j est écrit en "BS"
Pour avoir une écriture bornée de R_j, les 2 premiers chiffres de R_j ne peuvent pas être non nul de signe différent

$$R_j * 2^j = \sum_{i=-2}^{n} r_i 2^{-i} = r_{-2}\, r_{-1}\, r_0 \,,\, r_1\, r_2\, r_3 \ldots\ldots r_n \qquad r_{-2} * r_{-1} \ge 0$$

Choix de l'opération à exécuter

Itération à l'étape j: $R_{j+1} := R_j - q_j * D * 2^{-j}$ avec $q_j \in \{-1, 0, +1\}$

Pour déterminer l'opération q_j à exécuter à l'étape j,
il suffit d'examiner les 3 premiers chiffres de R_j seulement.

Condition de convergence: $|R_j| \leq 2 * D * 2^{-j} \Longrightarrow |R_{j+1}| \leq 2 * D * 2^{-j-1}$

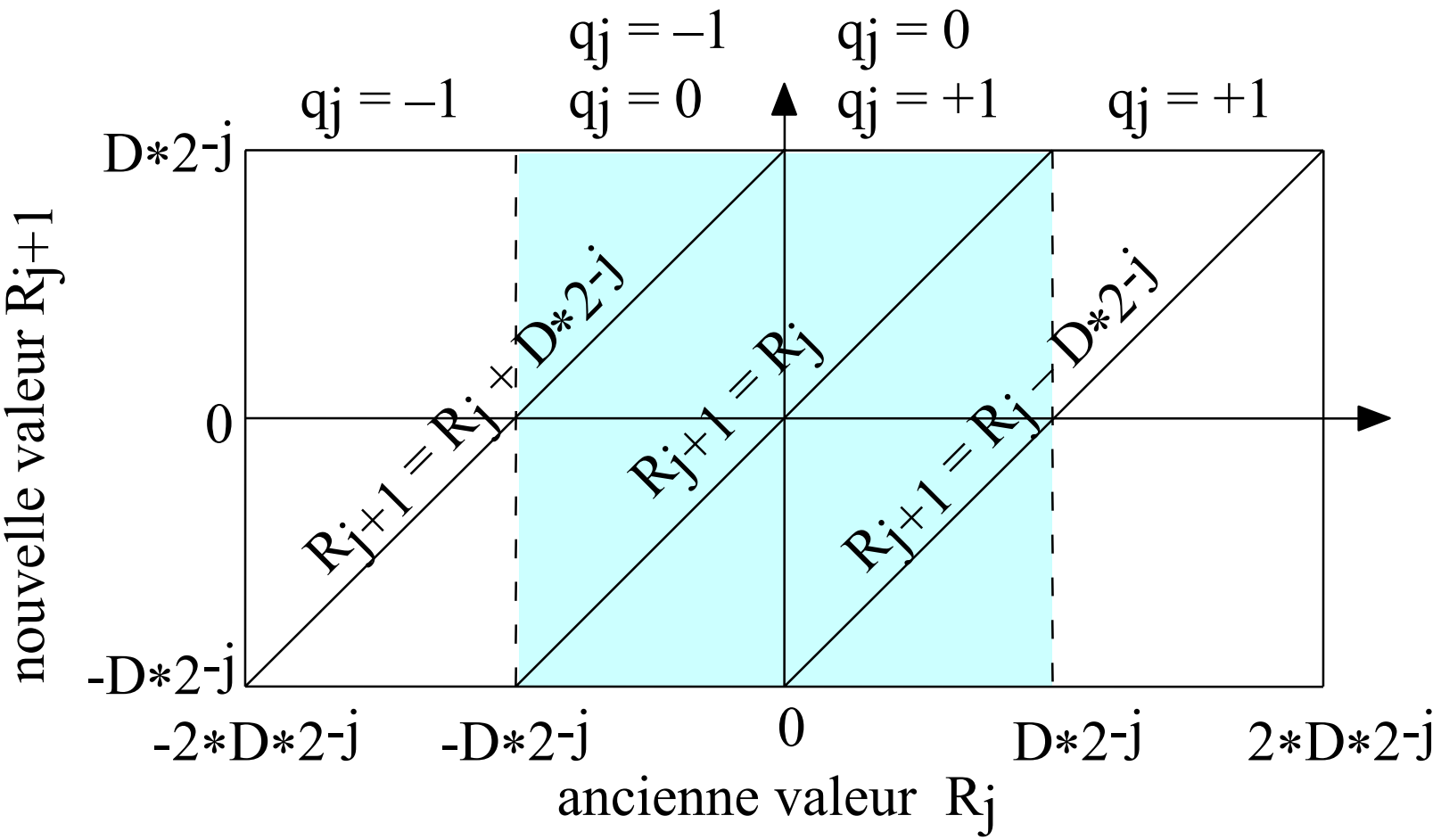
Examen de r_{-2} r_{-1} et r_0

r_{-2}	r_{-1}	r_0	
-1	x	x	$-2*D*2^{-j} \leq R_j < 0 \Longrightarrow R_{j+1} = R_j + D * 2^{-j}$
0	-1	x	
0	0	-1	
0	0	0	$-2^{-j} < R_j < +2^{-j} \Longrightarrow R_{j+1} = R_j$
0	0	+1	$0 < R_j \leq 2*D*2^{-j} \Longrightarrow R_{j+1} = R_j - D * 2^{-j}$
0	+1	x	
+1	x	x	

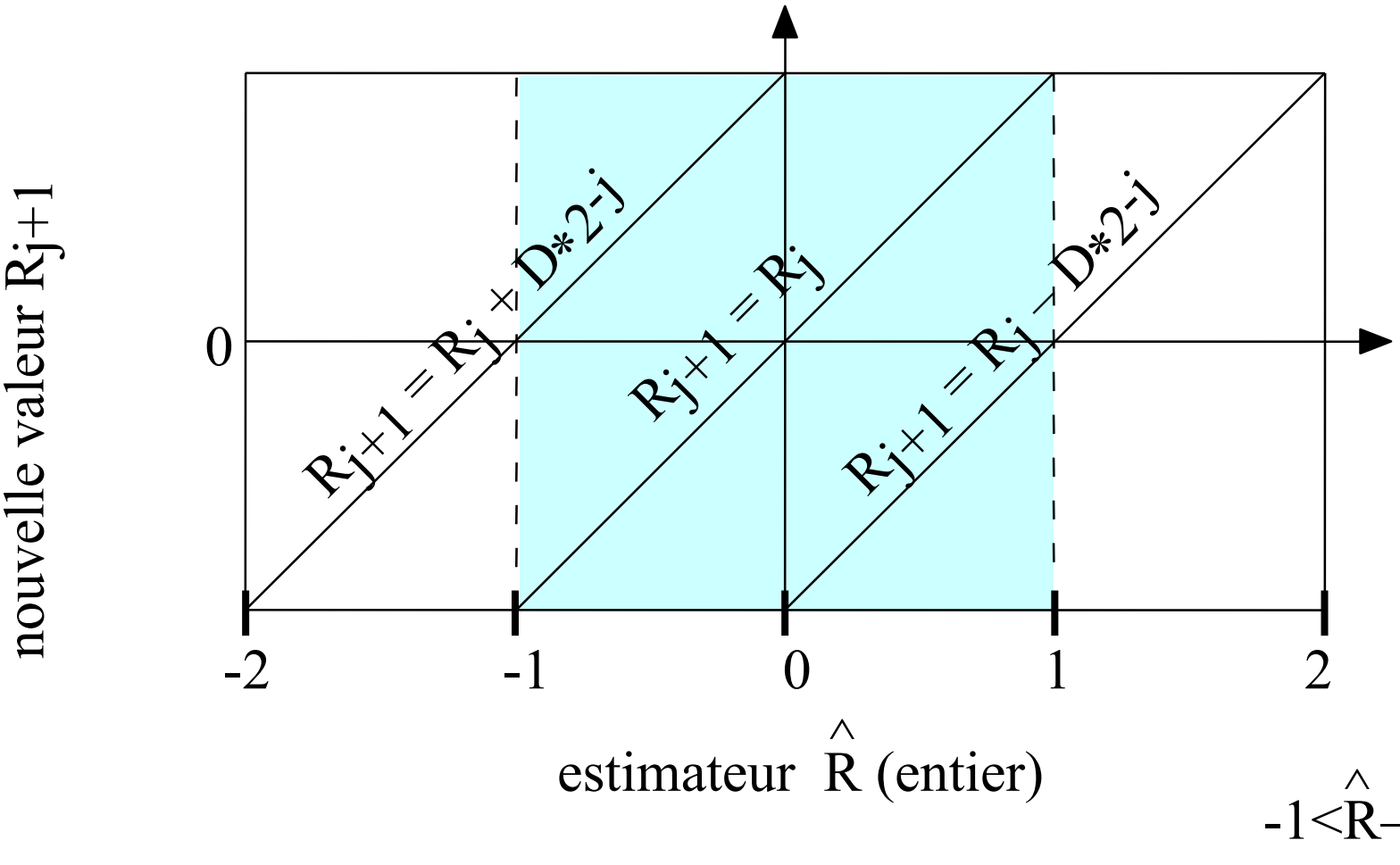
On introduit $\hat{R} = 4*r_{-2} + 2*r_{-1} + r_0$, "estimation" de R_j*2^j $\quad -1 < \hat{R} - R_j*2^j < 1$

Division SRT

Invariant: $-2*D*2^{-j} \le R_j \le 2*D*2^{-j}$



Division SRT (2)

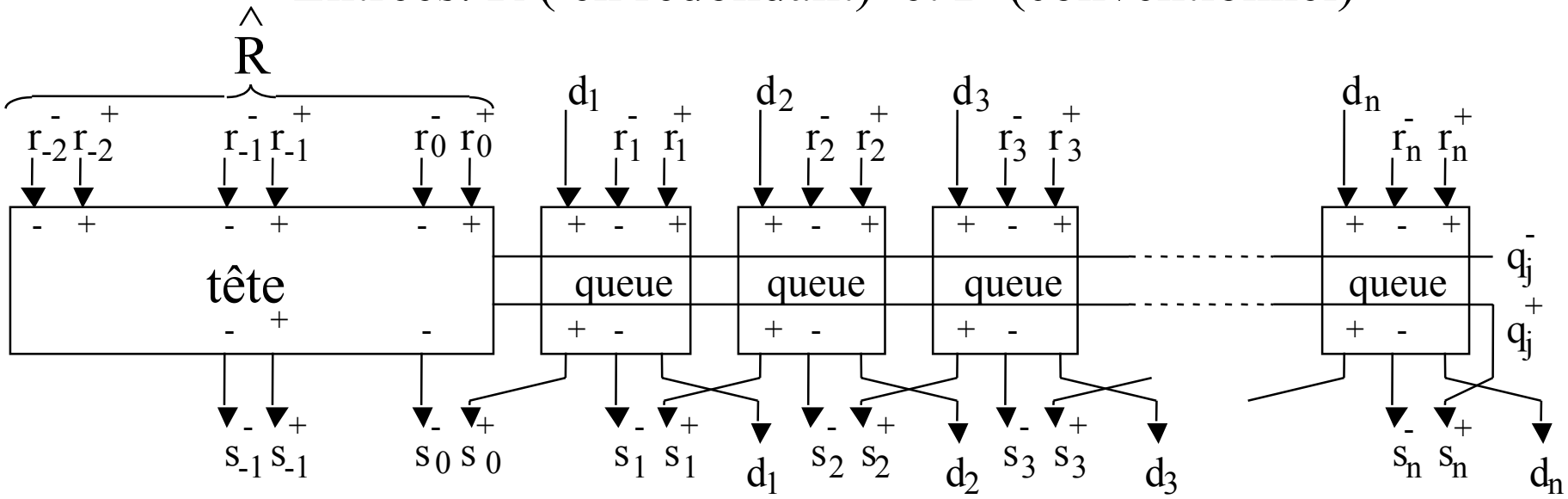


$$-1 < \hat{R} - R_j * 2^j < 1$$

valeur minimum de D = 1

Étage de diviseur

Entrées: R (en redondant) et D (conventionnel)



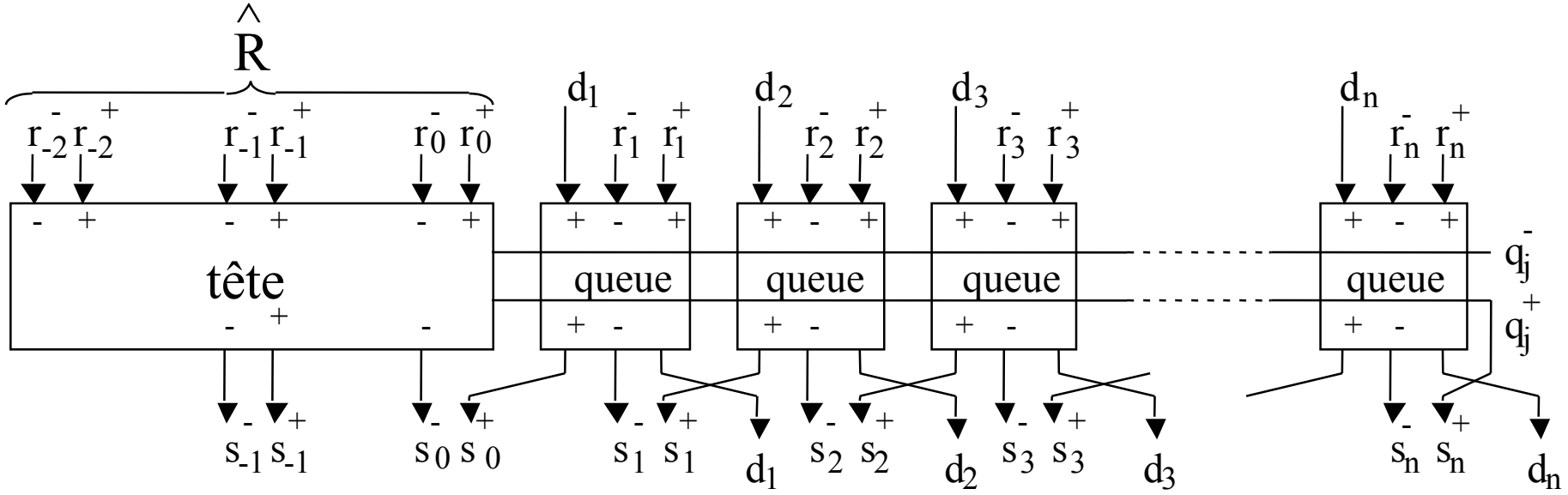
Sorties: S (en redondant) et D (conventionnel)

si $\hat{R} < 0$ **alors** S = R + D { add et sous sans propagation}

sinon si $\hat{R} > 0$ **alors** S = R - D

sinon S = R

Rôles de la tête et de la queue



Tête:

1- Déterminer l'opération à exécuter (add, sous ou rien)

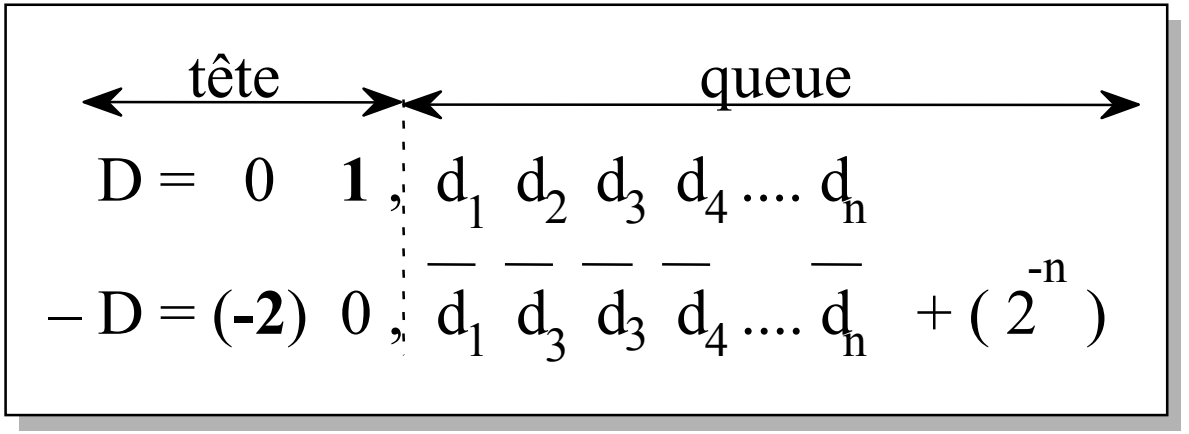
2- Exécuter cette opération sur les chiffres de tête

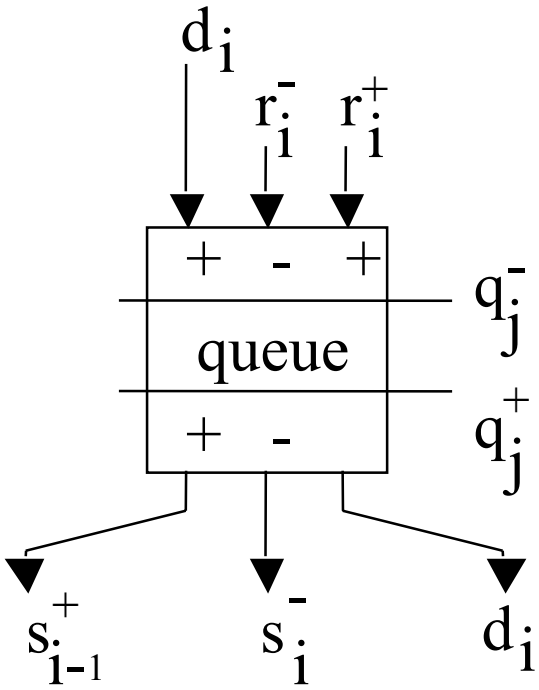
3- Recoder le résultat pour éliminer le chiffre poids fort s_{-2}

Queue:

- Exécuter l'opération (add, sous ou rien) sur les chiffres de queue sans retenue propagé
- Transmettre D décalé vers les poids faibles

Equations d'une cellule de queue





Equation arithmétique

$$2 * s^{+}_{i-1} - s^{-}_{i} = r^{+}_{i} - r^{-}_{i} + (q^{+}_{j}\,\overline{d_i} + q^{-}_{j}\,d_i)$$

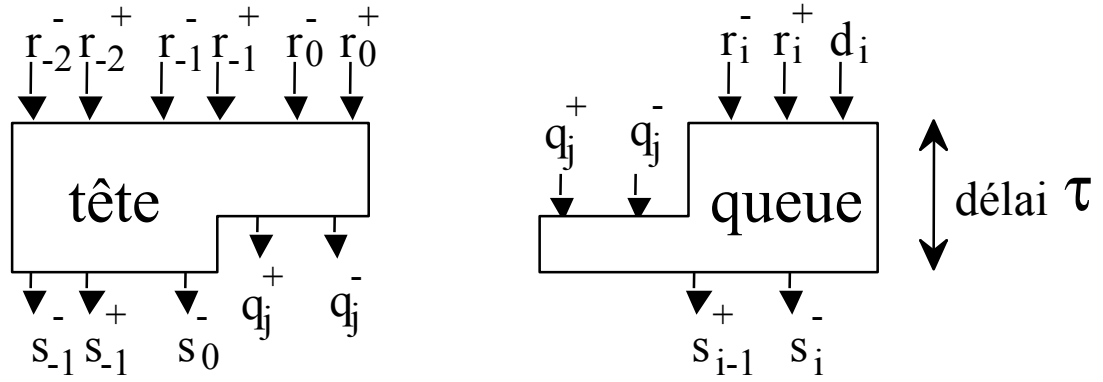
$$2 * s^{+}_{i-1} + \overline{s^{-}_{i}} = r^{+}_{i} + \overline{r^{-}_{i}} + (q^{+}_{j}\,\overline{d_i} + q^{-}_{j}\,d_i)$$

Les contrôles q^{-}_{j} et q^{+}_{j} arrivent après les autres

Optimisation de la cellule de queue

Les contrôles q_j^- et q_j^+ arrivent après les autres dans la queue

Modèles de délai recherchés



queue	s_{i-1}^+	s_i^-
q_j^+	$(r_i^+ \wedge \overline{r_i^-}) \vee (r_i^+ \vee \overline{r_i^-}) \wedge \overline{d_i}$	$r_i^+ \oplus \overline{r_i^-} \oplus \overline{d_i}$
q_j^-	$(r_i^+ \wedge \overline{r_i^-}) \vee (r_i^+ \vee \overline{r_i^-}) \wedge d_i$	$r_i^+ \oplus \overline{r_i^-} \oplus d_i$
$\overline{q_j^+ \vee q_j^-}$	$(r_i^+ \wedge \overline{r_i^-})$	$r_i^+ \oplus \overline{r_i^-}$

Equations du bloc de tête

$\hat{R} := 4 *(r_{-2}^{+} - r_{-2}^{-}) + 2 *(r_{-1}^{+} - r_{-1}^{-}) + (r_{0}^{+} - r_{0}^{-})$

$q_j^+ := (\hat{R} > 0)\ ;\ q_j^- := (\hat{R} < 0)\ ;$

si q_j^+ **alors** $\Sigma_{out} := \hat{R} - 2$

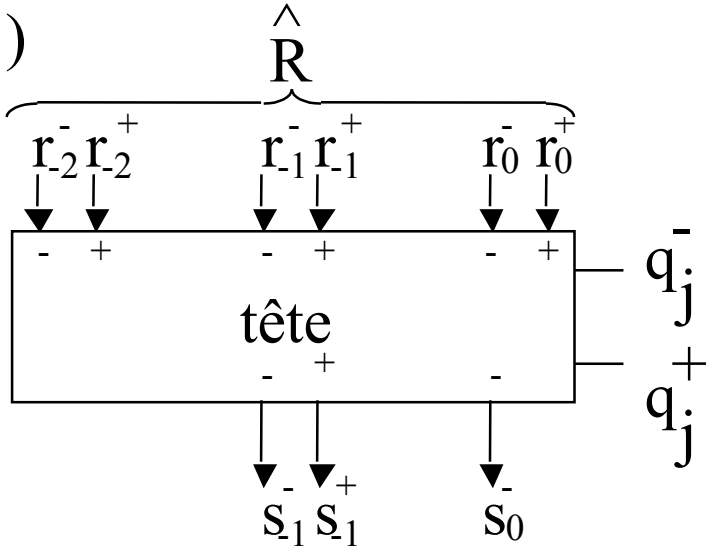
sinon si q_j^- **alors** $\Sigma_{out} := \hat{R} + 1$

sinon $\Sigma_{out} := 0\ ;$

$s_{-1}^+ := (\Sigma_{out} > 0)\ ;$

$s_{-1}^- := (\Sigma_{out} \le -2)\ ;$

$s_0^- := \Sigma_{out} - 2 *(s_{-1}^+ - s_{-1}^-)\ ;$



$\Rightarrow$

$q_j^+ := r_{-2}^+ + \overline{r_{-2}^-}\,(r_{-1}^+ \overline{r_{-1}^-} + \overline{r_{-1}^-}\, r_0^+ \overline{r_0^-} + r_{-1}^+ r_0^+ \overline{r_0^-})\ ;$

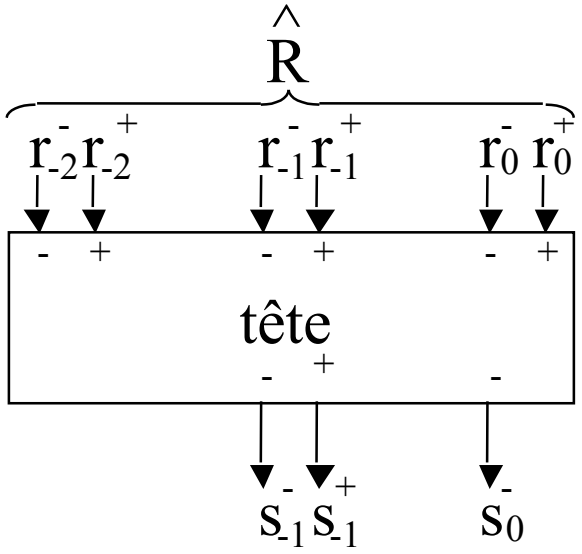
$q_j^- := r_{-2}^- + \overline{r_{-2}^+}\,(\overline{r_{-1}^+}\, r_{-1}^- + \overline{r_{-1}^+}\,\overline{r_0^+}\, r_0^- + r_{-1}^-\,\overline{r_0^+}\, r_0^-)\ ;$

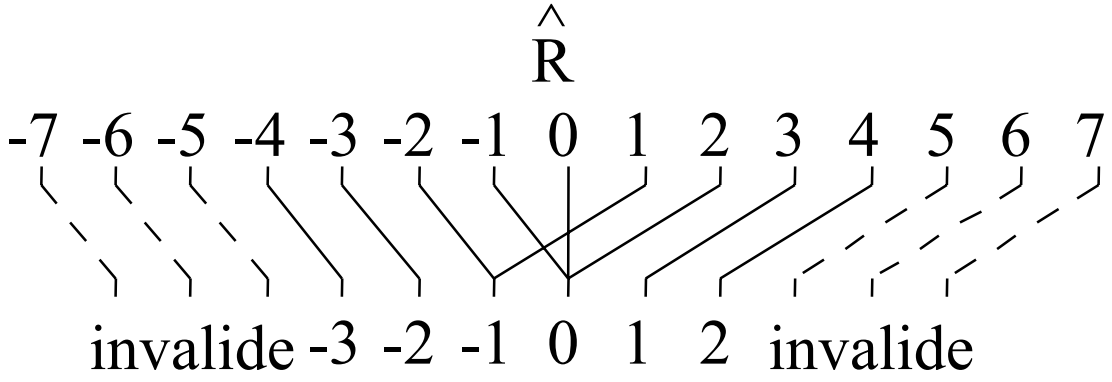
$s_{-1}^+ := r_{-2}^+(r_{-1}^+ + \overline{r_{-1}^-} + r_0^+ \overline{r_0^-}) + \overline{r_{-2}^-}\, r_{-1}^+ \overline{r_{-1}^-}\, r_0^+ \overline{r_0^-}\ ;$

$s_{-1}^- := r_{-2}^- (\overline{r_{-1}^+} + r_1^- + \overline{r_0^+}\, r_0^-) + \overline{r_{-2}^+}\,\overline{r_{-1}^+}\, r_{-1}^-\,\overline{r_0^+}\, r_0^-\ ;$

$s_0^- := q_j^- \oplus r_0^+ \oplus r_0^-$ { seuls chiffres de poids 2^0 };

Condition arithmétique





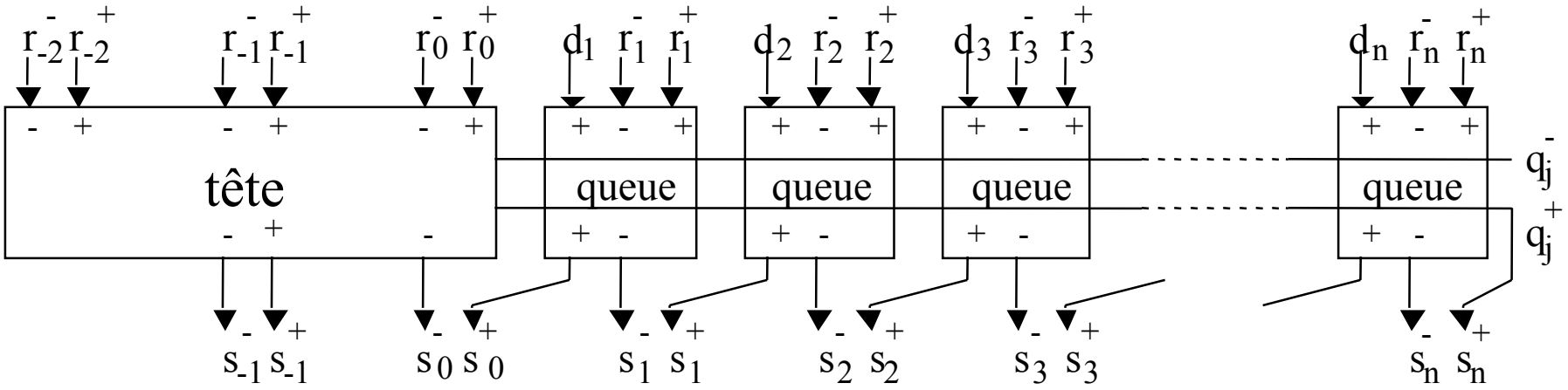
Il faut que $-4 \leq \hat{R} \leq +4$

$$-4 < -2*D \leq R_j*2^j \leq 2*D < 4$$

$$-1 < \hat{R} - R_j*2^j < 1$$

$$\Rightarrow -5 < \hat{R} < 5$$

Le Quotient est en notation BS
$\Rightarrow$ il faut le convertir

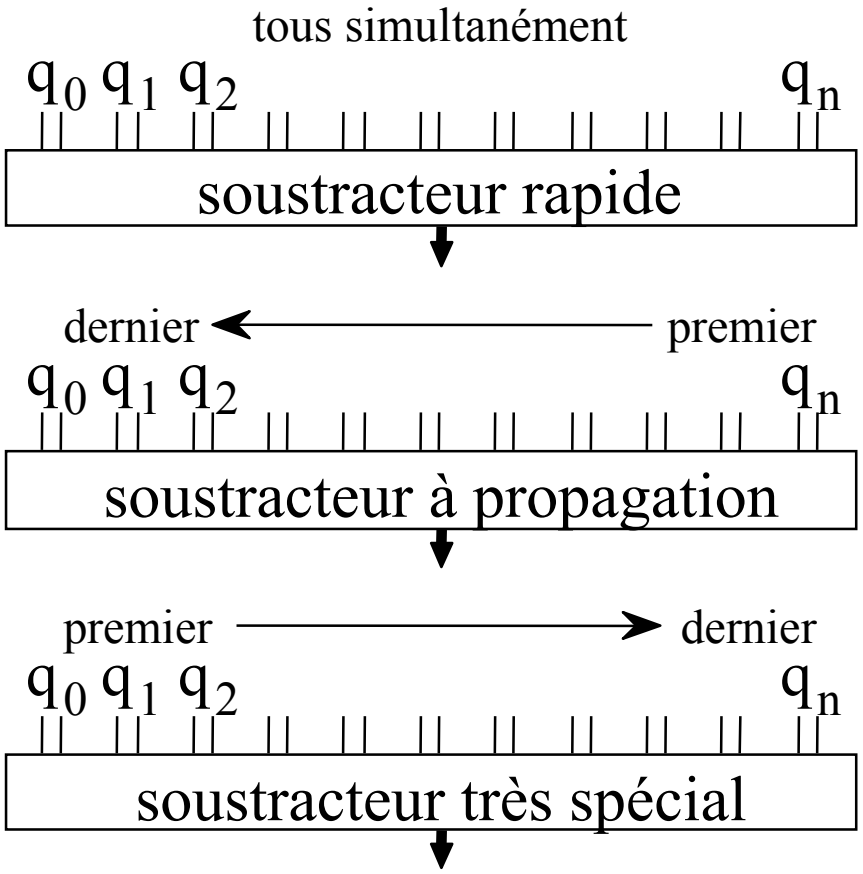


La conversion ne change pas la valeur de Q mais seulement sa représentation sous forme de chaîne de bits.
La somme pondérée des bits qui entrent dans le convertisseur est égale à la somme pondérée des bits qui en sortent.

$$q_j = q_j^+ - q_j^-$$
$$q_j \in \{-1, 0, +1\}$$

Conversion de "borrow save" à standard

$$Q_n = \sum_{i=0}^{n} q_i * 2^{-i} \quad q_i \in \{-1,0,+1\} \Longrightarrow Q_n = \sum_{i=0}^{n} \max(q_i, 0) * 2^{-i} \; - \sum_{i=0}^{n} \max(-q_i, 0) * 2^{-i}$$

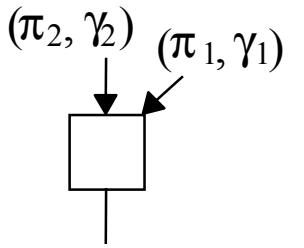


L'additionneur doit être très rapide

Le délai de propagation de la retenue sur une position doit être inférieur au retard entre 2 q

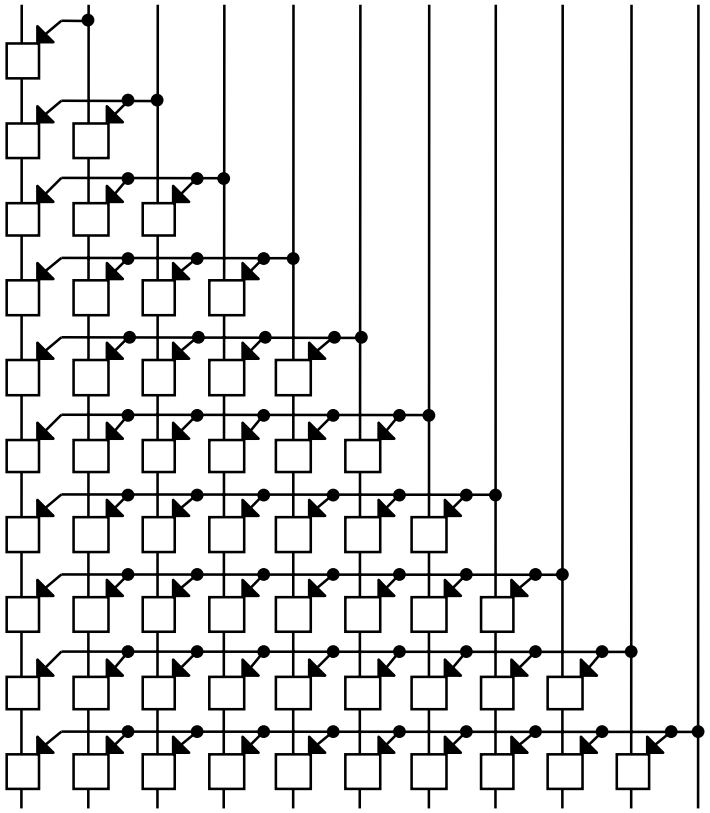
On veut le résultat dès que q_n est prêt

Conversion de "borrow save" à standard (2)



- Associatif
- Non commutatif
- Idempotent
- Croissant (inverseurs)

génération des p_i et des g_i



$q_i \in \{-1,0,+1\}$

q_i	p_i	g_i
-1	0	1
0	1	0
+1	0	0

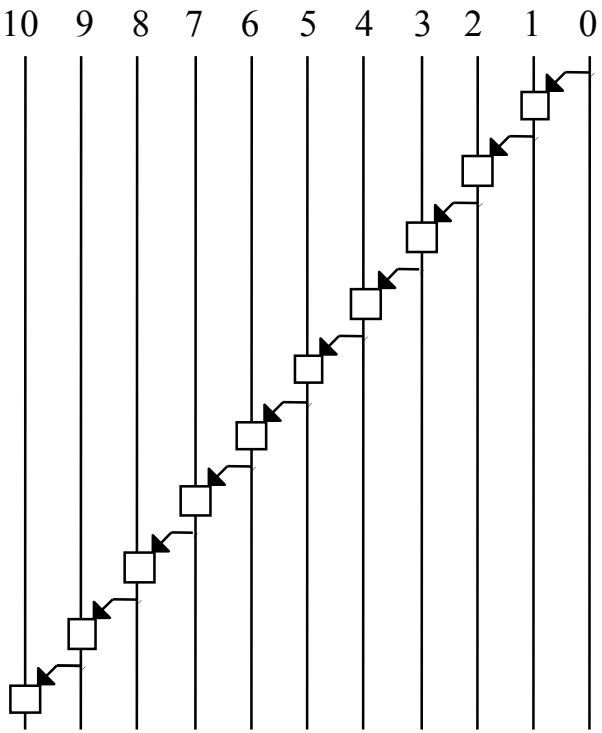
génération de la somme $s_i = p_i \oplus \mathbf{G}_{i-1,0}$

$s_i \in \{0,1\}$

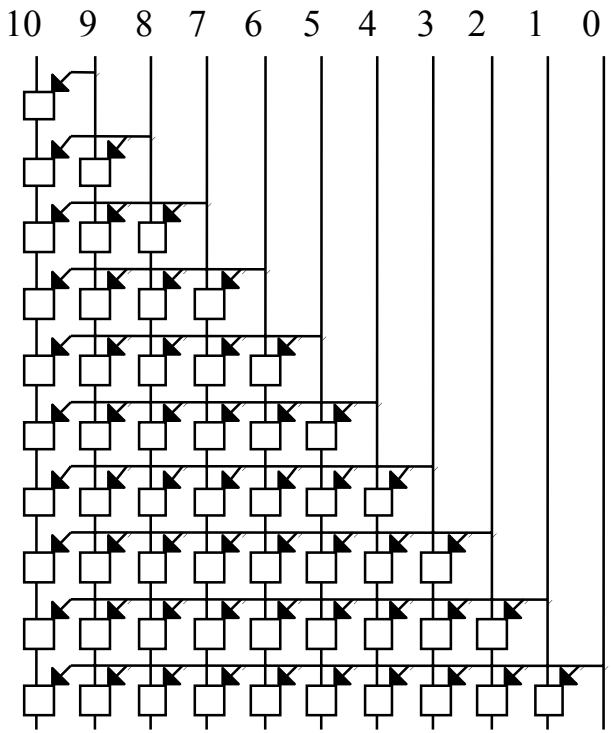
Conversion de "borrow save" à standard (3)

$q_i \in \{-1,0,+1\}$ $p_i \in \{0,1\}$

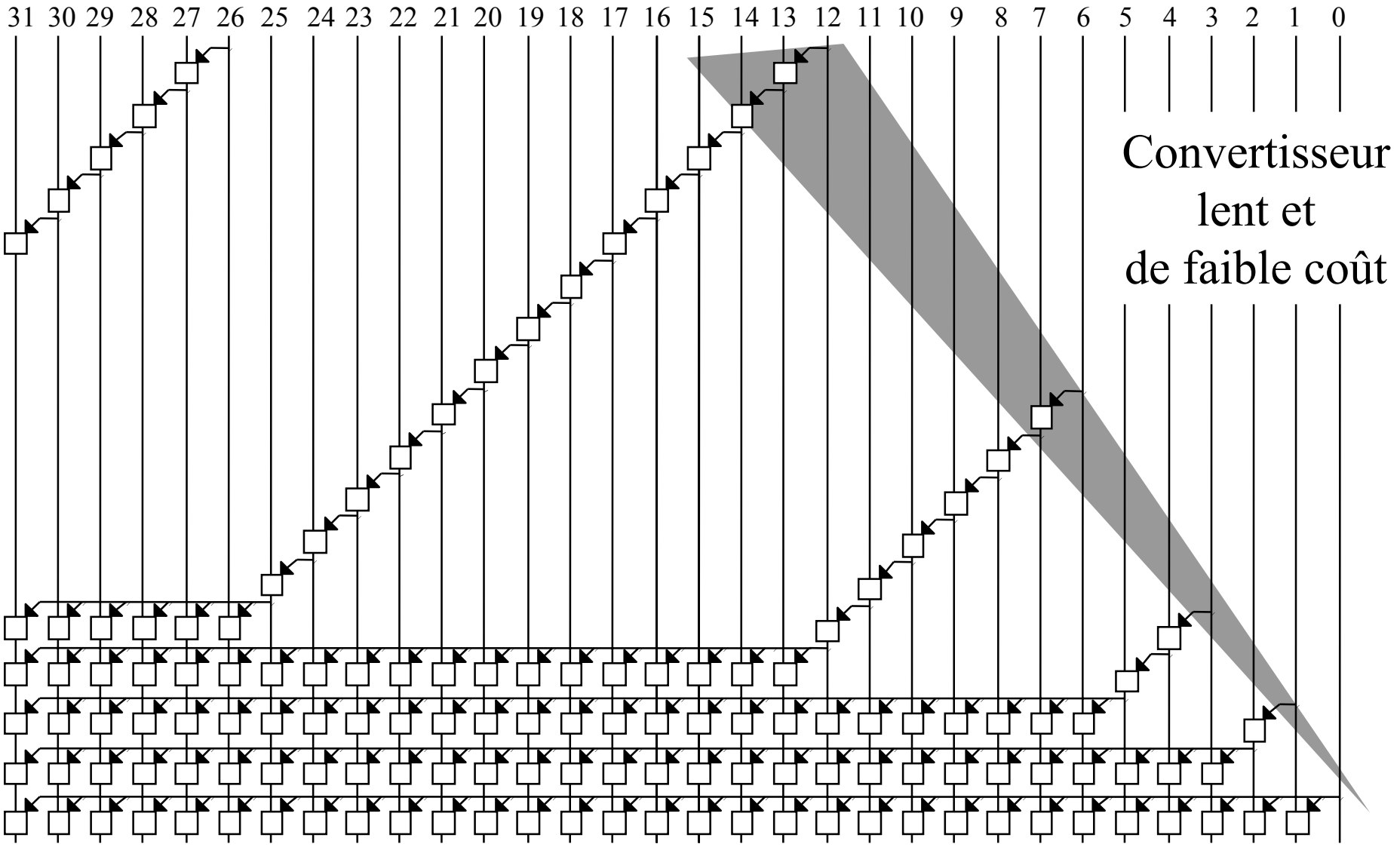
séquentiellement
poids faibles d'abord



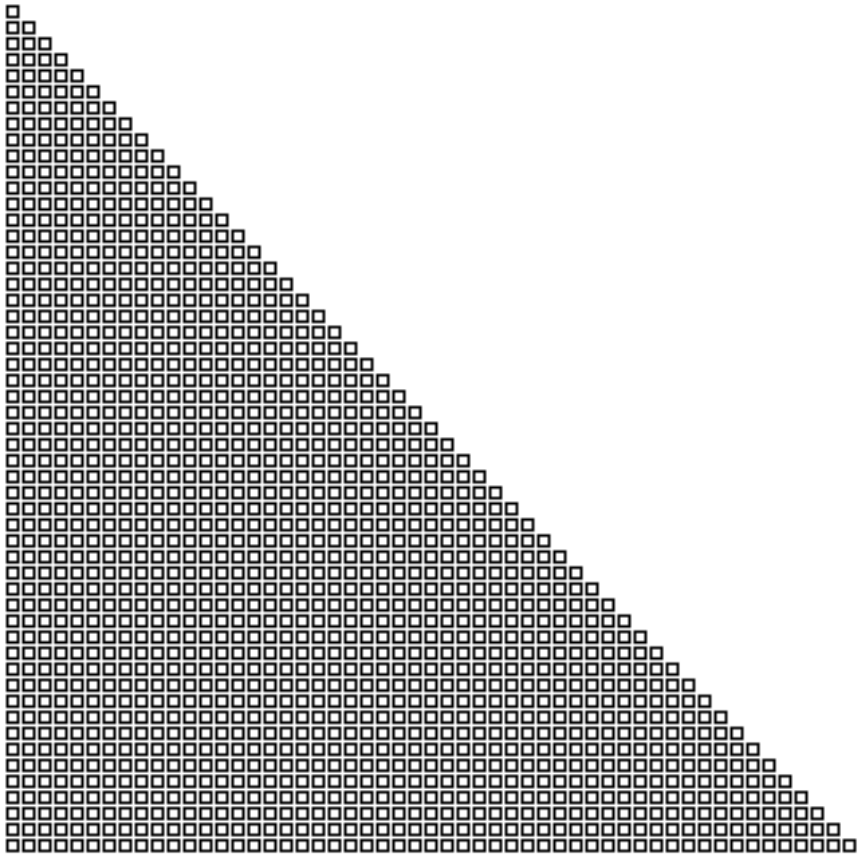
séquentiellement
poids forts d'abord



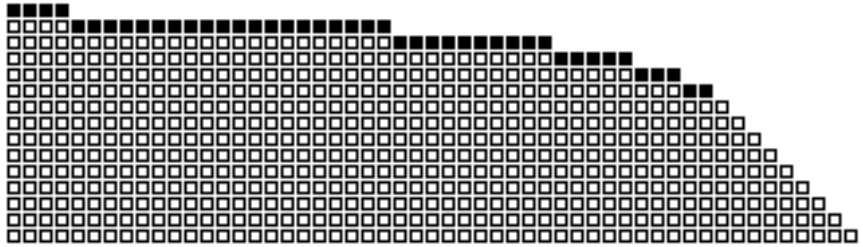
Conversion de "borrow save" à standard (4)



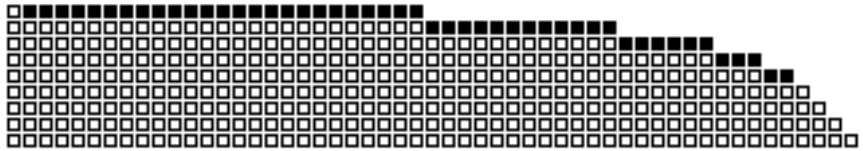
Formes du convertisseur



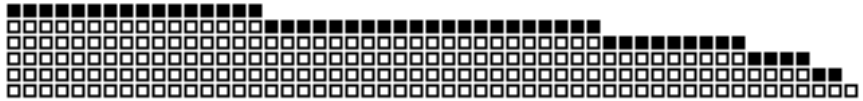
≤ 1



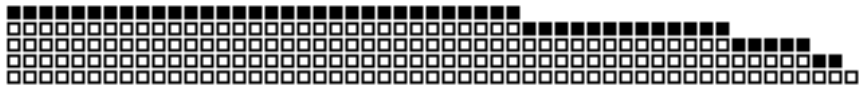
1,1



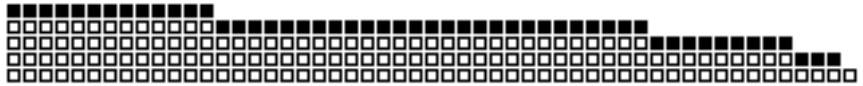
1,2



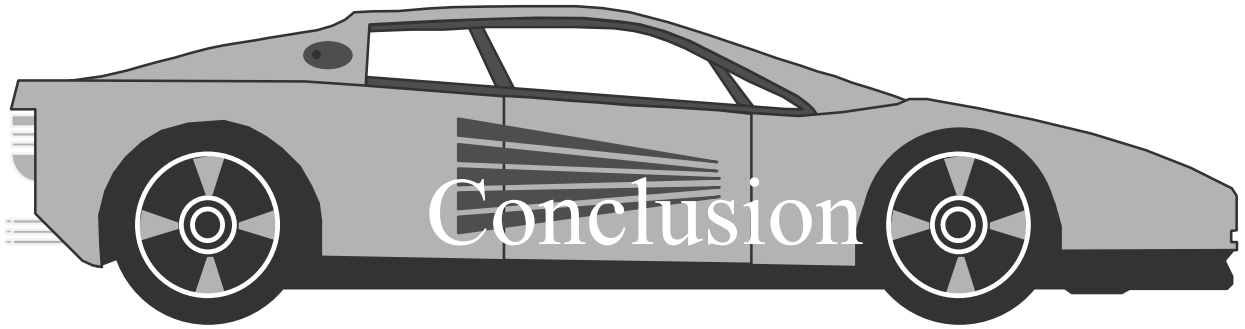
1,5



1,8



2,2



Un algorithme de division rapide a été proposé:

- Les Additions/Soustraction ne propagent pas la retenue

- Le calcul, le recodage et le test du reste partiel sont effectués simultanément (et non séquentiellement)

- Les lignes longues et leurs amplis sont éliminées du chemin critique

- Un nouvel algorithme de conversion du quotient, rapide et peu coûteux, a été introduit.

Extraction de Racine carrée matérielle

Alain GUYOT

Concurrent Integrated Systems
TIMA

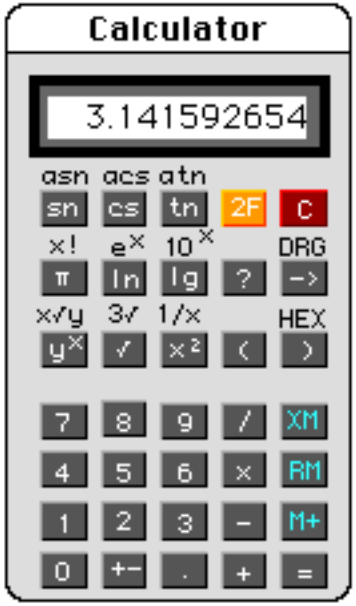
(33) 04 76 57 46 16
Alain.Guyot@imag.fr
http://tima-cmp.imag.fr/Homepages/guyot
Techniques de l'Informatique et de la Microélectronique pour l'Architecture. Unité associée au C.N.R.S. n° B0706

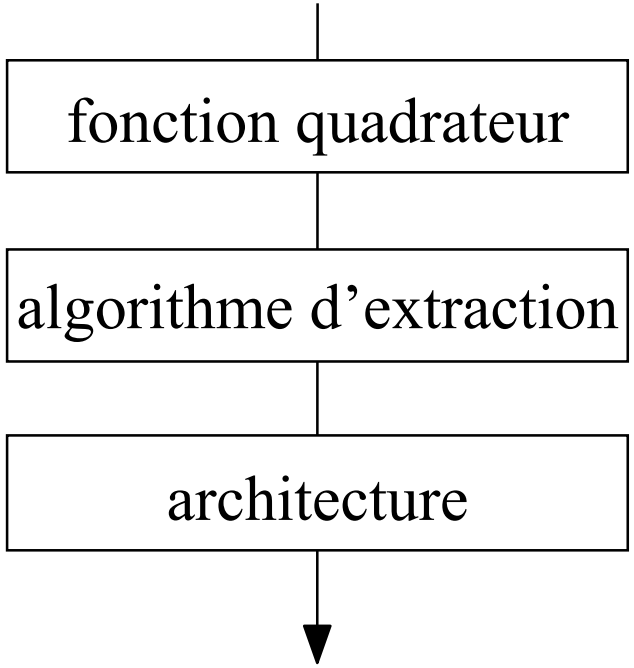
But: Réaliser des extracteurs de racine carrée combinatoires rapides

Optimiser la surface et/ou la vitesse

Problèmes
- Propagation de la retenue

Moyen
Utiliser des additionneurs sans propagation de retenue





Généralités sur la racine carrée

Nous calculerons la racine d'un nombre normalisé A tel que $1 \le A < 4$. $A = 2*a_{-1} + \sum_{i=0}^{n-1} a_i * 2^{-i}$

Alors $Q = \sqrt[2]{A}$ est également normalisé: $1 \le Q < 2$.

De même que l'architecture d'un diviseur (naïf) se déduit de l'architecture d'un multiplieur (naïf) l'architecture d'un extracteur de racine carrée se déduit de celle d'un quadrateur.

Les algorithmes d'extraction de racine carrée se déduisent également des algorithmes de division car:

$$\text{si } Q = \frac{A}{Q} \text{ alors } Q = \sqrt[2]{A}$$

Algorithme naïf:

$Q_0 := 1$;
si $(Q_j + 2^{-j-1})^2 \le A$ **alors** $Q_{j+1} := Q_j + 2^{-j-1}$
sinon $Q_{j+1} := Q_j$;

Iteration de Héron:

$$Q_{j+1} = \frac{1}{2}(Q_j + \frac{A}{Q_j})$$

Quadrateur

On veut maintenir l'invariant $Q_{2j} = R_j^2 \quad \forall\ j \quad R_j = \sum_{i=0}^{j} r_i 2^{-i}$

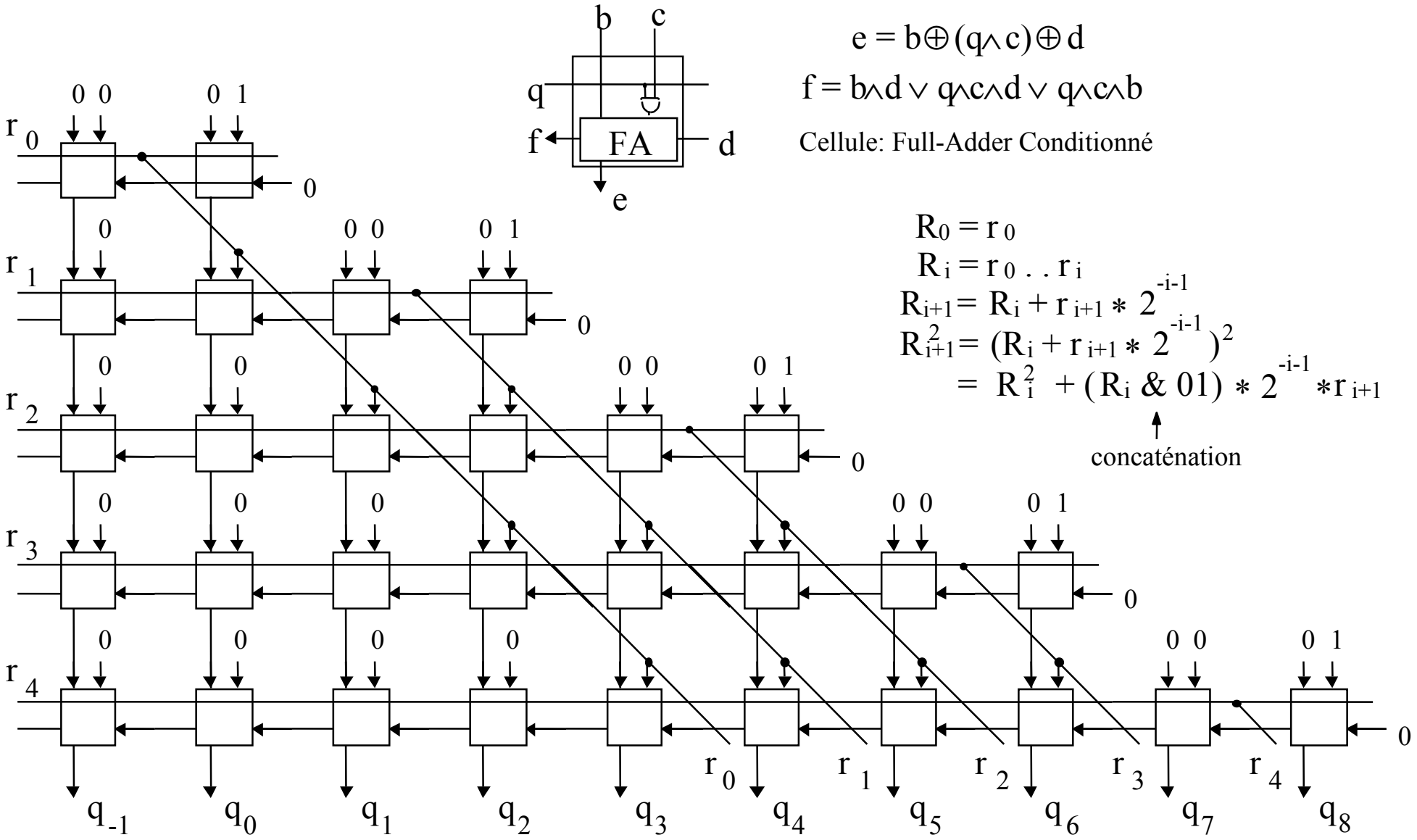
Donc quand $R_{j+1} := R_j + r_{j+1} * 2^{-j-1}$ alors

$$(R_{j+1})^2 := (R_j)^2 + r_{j+1} * (2 * R_j + 2^{-j-1}) * 2^{-j-1}$$

$R_0 := r_0$; $Q_0 := R_0$; $j := 0$;
Tantque $j \leq$ précision_requise **faire**
    $j := j+1$;
    $Q_{2j+2} := Q_{2j} + r_{j+1} * (2 * R_j + 2^{-j-1}) * 2^{-j-1}$;
    $R_{j+1} := R_j + r_{j+1} * 2^{-j-1}$
Fintantque

concaténations

Quadrateur à simplifier



$$e = b \oplus (q \wedge c) \oplus d$$

$$f = b \wedge d \vee q \wedge c \wedge d \vee q \wedge c \wedge b$$

Cellule: Full-Adder Conditionné

$$R_0 = r_0$$

$$R_i = r_0 \,..\, r_i$$

$$R_{i+1} = R_i + r_{i+1} * 2^{-i-1}$$

$$R_{i+1}^2 = (R_i + r_{i+1} * 2^{-i-1})^2$$

$$= R_i^2 + (R_i \,\&\, 01) * 2^{-i-1} * r_{i+1}$$

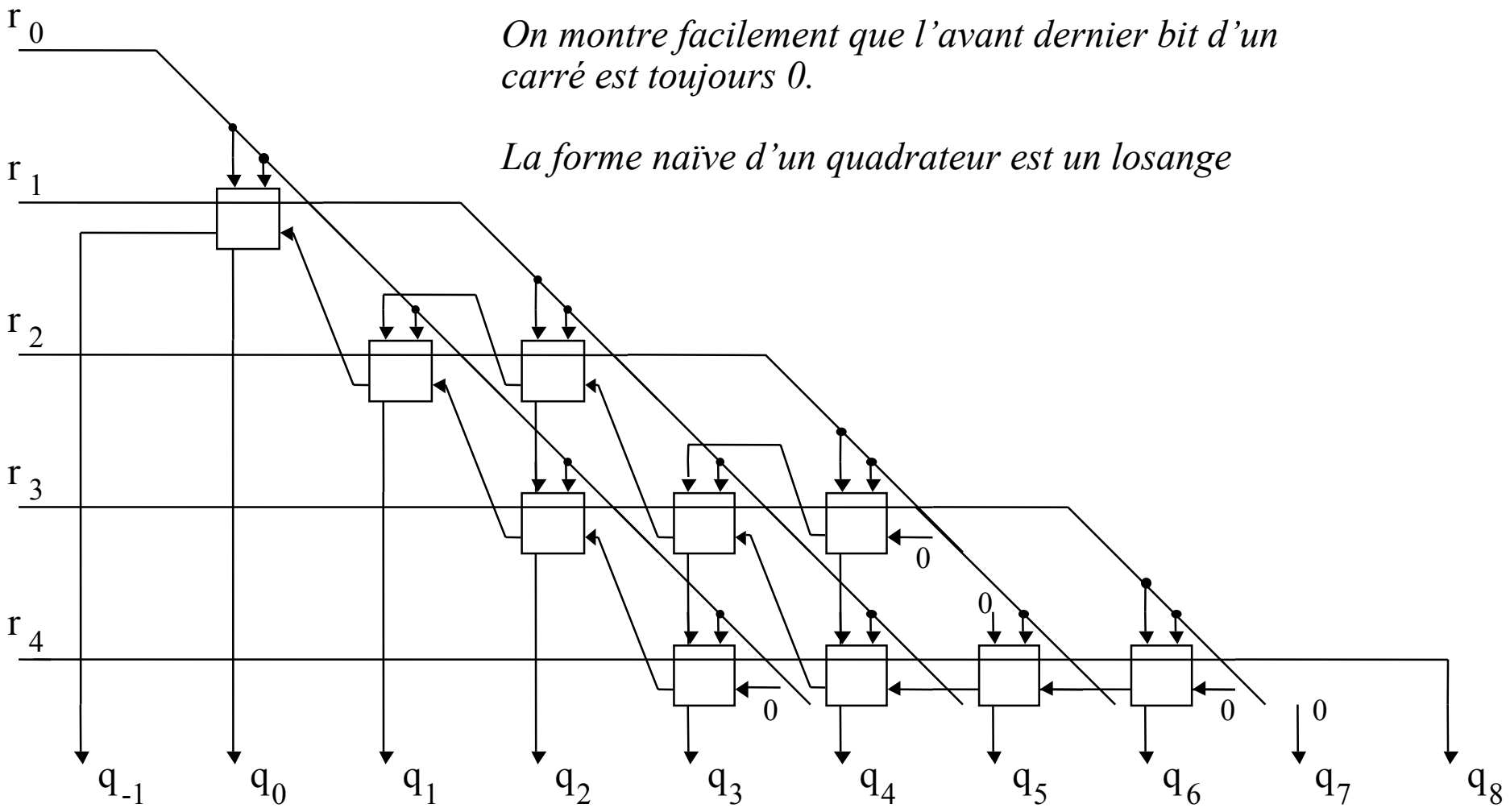
concaténation

Quadrateur

Elimination des FA dont la sortie est égale à l'entrée

On montre facilement que l'avant dernier bit d'un carré est toujours 0.

La forme naïve d'un quadrateur est un losange



Extracteur de racine carrée récurrent (sans restauration)

On va construire une suite $Q_0, Q_1, Q_2, \ldots Q_n$ et une suite $R_0, R_2, R_4, \ldots R_{2n}$

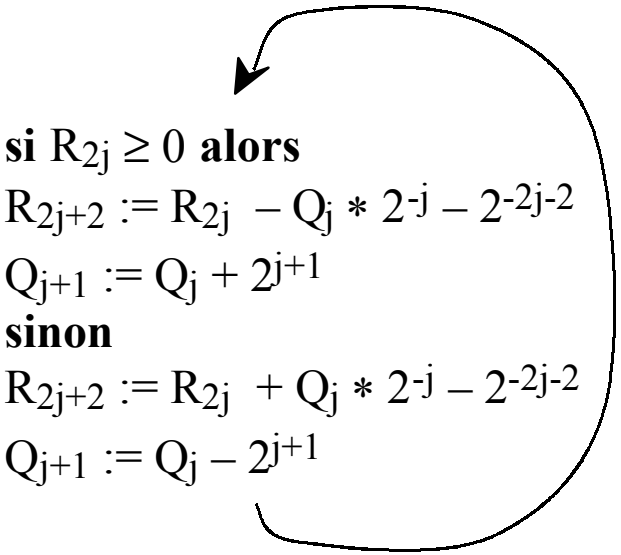
On veut maintenir l'invariant $(Q_j)^2 = A - R_{2j}$ $\forall$ j ($R_{2j} \rightarrow 0$) { $A \in [1,4[$ }

$R_{2j+2} := R_{2j} - Q_{j+1}^2 + Q_j^2 = R_{2j} - (Q_j + q_{j+1} * 2^{-j-1})^2 + Q_j^2$

$Q_1 := 1,1$; $R_2 := A - 1\ 0,0\ 1$; $j := 1$;
Tantque $j \leq$ précision_requise **faire**
    $j := j + 1$;
    si $R_{2j} \geq 0$ **alors** { $q_{j+1} = +1$ }
        $Q_{j+1} := Q_j + 2^{-j-1}$;
        $R_{2j+2} := R_{2j} - (Q_j * 2^{-j} + 2^{-2j-2})$;
    sinon { $q_{j+1} = -1$ }
        $Q_{j+1} := Q_j - 2^{-j-1}$;
        $R_{2j+2} := R_{2j} + (Q_j * 2^{-j} - 2^{-2j-2})$;
    finsinon ;
fintanque ;

concaténations

Exemple d'extraction de racine carrée sans restauration



Calcul des restes		Valeurs du reste (signé)		Calcul des racines	Valeurs
$R_0 := 1,111001 - 1,0$		01,111001	1,890625	$Q_0 := 1$	1
$R_2 := R_0 - 1 - 0,01$	$:= R_0 - 1,01$	00,111001	0,890625	$Q_1 := Q_0 + 0,1$	1,1
$R_4 := R_2 + 0,11 - 0,0001$	$:= R_2 + 0,1011$	11,101001	- 0,359375	$Q_2 := Q_1 - 0,01$	1,01
$R_6 := R_4 - 0,0101 - 0,000001$	$:= R_4 - 0,010101$	00,010101	0,328125	$Q_3 := Q_2 + 0,001$	1,011
$R_8 := R_6 -$		00,000000	0,000000	$Q_4 := Q_3 + 0,0001$	1,0111

Extracteur de racine carrée

$Q_1 := 1 * 2^{-1}$; $R_2 := (a_1 * 2^{-1}\ a_2 * 2^{-2})$ - 0,0 1 ; $j := 1$;

Tantque $j \leq$ précision_requise **faire**

$j := j + 1$;

si $R_{2j} \geq 0$ **alors**

$q_{j+1} := +1$

sinon

$q_{j+1} := -1$

finsinon ;

$Q_{j+1} := Q_j + q_{j+1} * 2^{-j-1}$;

$R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) - q_{j+1} * (Q_j + q_{j+1} * 2^{-j-2}) * 2^{-j}$;

fintanque ;

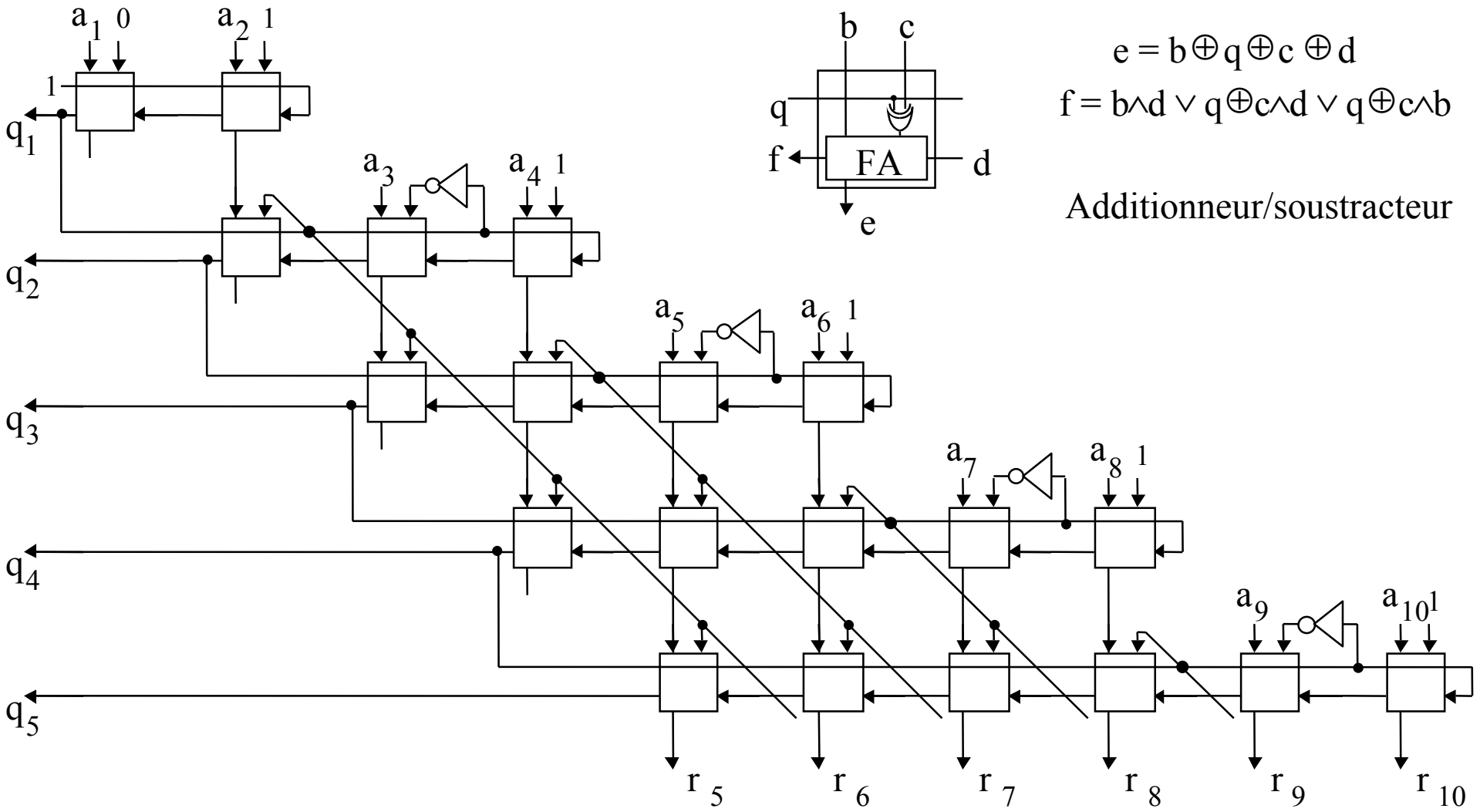
On code en binaire $p_{i+1} = 0$ si $q_{i+1} = +1$ et $p_{i+1} = 1$ si $q_{i+1} = -1$ (signe de q_{i+1}) .

Le cas général est p_{i+1}= retenue sortante de l'opération add/sous (signe)

$$R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) - (-1)^{p_{j+1}} (Q_{j-1} \,\&\, p_{j+1}\ \overline{p_{j+1}}\ 1) * 2^{-j}$$

Extracteur de racine carrée (2)

sans restauration



$e = b \oplus q \oplus c \oplus d$

$f = b \wedge d \vee q \oplus c \wedge d \vee q \oplus c \wedge b$

Additionneur/soustracteur

Extracteur de racine carrée(3)

(examen des 3 premiers chiffres significatifs de R^{2j})

$Q_0 := 1 * 2^{-0}$; $R_0 := (a_{-1} * 2^1 \; a_{-2} * 2^2) - 1$; $j := 0$;

Tantque $j \leq$ précision_requise **faire**

$j := j + 1$;

si le signe de R_{2j} n'est pas distinguable par l'examen de ses 3 premier chiffres **alors**

$q_{j+1} := 0$

sinon si $R_{2j} \geq 0$ **alors**

$q_{j+1} := +1$

sinon

$q_{j+1} := -1$

finsinon ;

$Q_{j+1} := Q_j + q_{j+1} * 2^{-j-1}$;

$R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) - q_{j+1} * (Q_j - q_{j+1} * 2^{-j-2}) * 2^{-j}$;

fintanque ;

Calcul du reste partiel R_{2j+2}

Itération générale: $R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) - q_{j+1} * (Q_j - q_{j+1} * 2^{-j-2}) * 2^{-j}$;

Si $q_{j+1} = 0$ alors $R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2})$;

Si $q_{j+1} = +1$ alors $R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) - Q_j * 2^{-j} + 2^{-2j-2}$; {soustraction}

Si $q_{j+1} = -1$ alors $R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) + Q_j * 2^{-j} + 2^{-2j-2}$; {addition}

Par concaténation on a: $Q_j = \sum_{i=-1}^{j} q_i * 2^{-i}$ avec $q_i \in \{-1, 0, +1\}$

Pour effectuer l'addition/soustraction, on désire $Q_j = \sum_{j=-1}^{j} p_i * 2^{-i}$ avec $p_i \in \{0, +1\}$

Solution: convertir Q_j "à la volée" (changer la représentation en conservant la valeur).

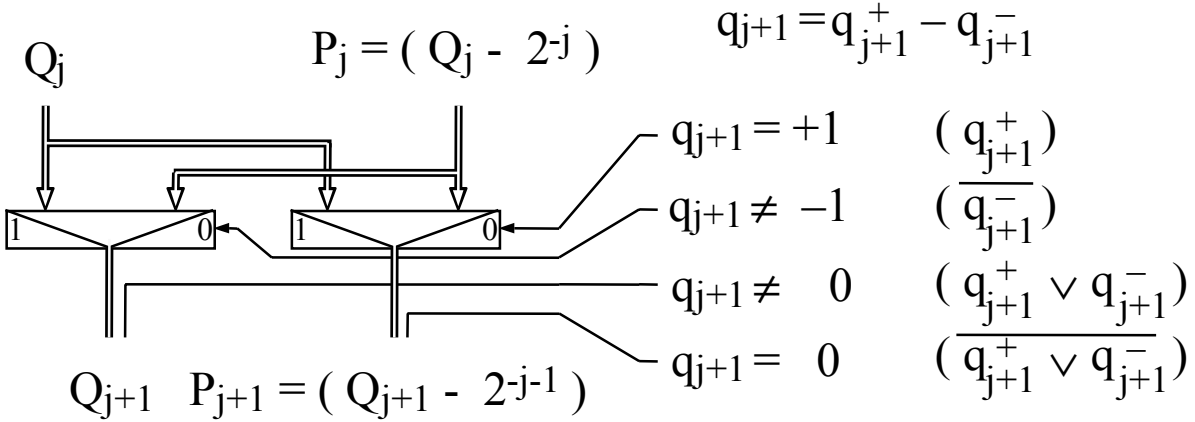
Conversion de la racine carrée

Par concaténation on a: $Q_j = \sum_{i=-1}^{j} q_i * 2^{-i}$ avec $q_i \in \{ -1, 0, +1\}$

Pour effectuer l'addition/soustraction, on désire $Q_j = \sum_{i=-1}^{j} p_i * 2^{-i}$ avec $p_i \in \{0, +1\}$

Solution: calculer simultanément les valeurs Q_i et $P_j = (Q_j - 2^{-j}) \forall\ j$.
On a Q_j et $(Q_j - 2^{-j})$ et on veut obtenir Q_{j+1} et $(Q_{j+1} - 2^{-j-1})$ sans propagation de retenue

Si $q_{j+1} = -1$ alors $Q_{j+1} := (Q_j - 2^{-j}) + 2^{-j-1}$; $(Q_{j+1} - 2^{-j-1}) := (Q_j - 2^{-j})$;
Si $q_{j+1} = 0$ alors $Q_{j+1} := Q_j$; $(Q_{j+1} - 2^{-j-1}) := (Q_j - 2^{-j}) + 2^{-j-1}$;
Si $q_{j+1} = +1$ alors $Q_{j+1} := Q_j + 2^{-j-1}$; $(Q_{j+1} - 2^{-j-1}) := Q_j$;



Extraction de racine carrée sans propagation

$Q_1 := 1 * 2^{-1}$; $P_1 := O$; $R_2 := (a_1 * 2^{-1}\ a_2 * 2^{-2})$ - 0,0 1 ; $j := 1$;

Tantque $j \leq$ précision_requise **faire**

$j := j + 1$;

si le signe de R_{2j} n'est pas distinguable par l'examen de ses 3 premier chiffres **alors**

$Q_{j+1} := Q_j$;

$P_{j+1} := P_j + 2^{-j-1}$;

$R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2})$;

sinon si $R_{2j} > 0$ **alors**

$Q_{j+1} := Q_j + 2^{-j-1}$;

$P_{j+1} := Q_j$;

$R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) - Q_j * 2^{-j} + 2^{-2j-2}$;

sinon

$Q_{j+1} := P_j + 2^{-j-1}$;

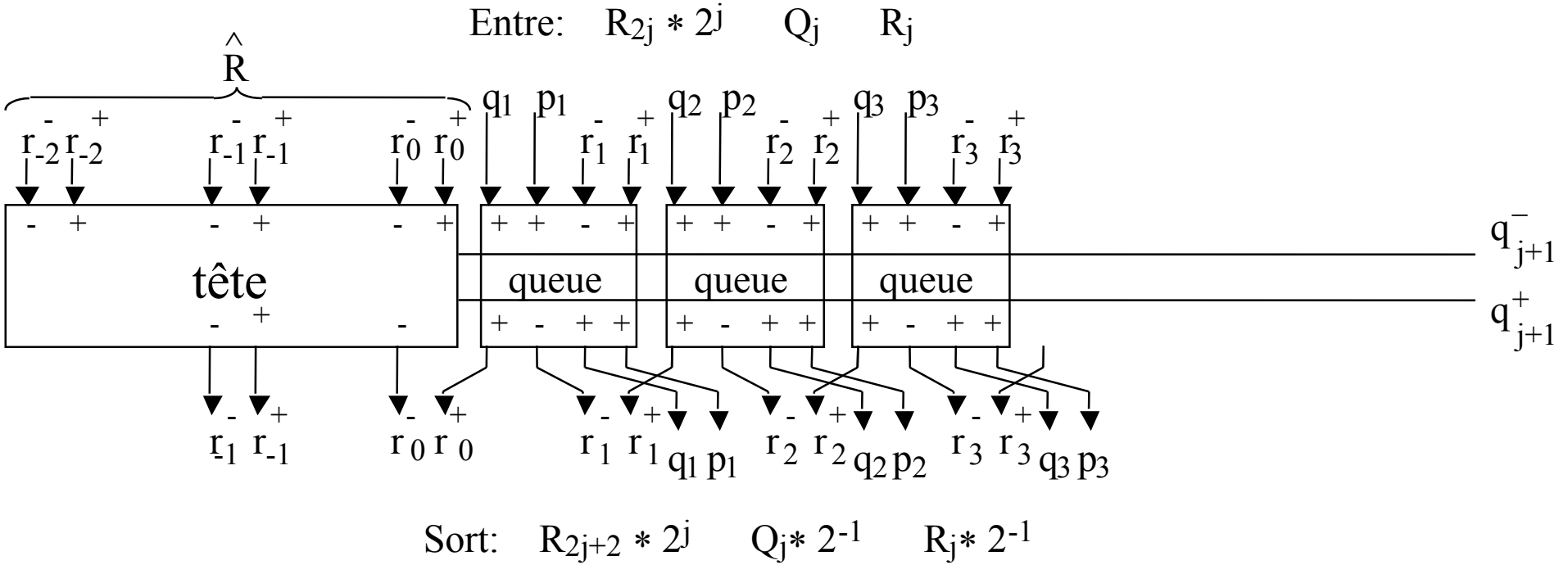
$P_{j+1} := P_j$;

$R_{2j+2} := (R_{2j} + a_{2j+1} * 2^{-2j-1} + a_{2j+2} * 2^{-2j-2}) + Q_j * 2^{-j} + 2^{-2j-2}$;

finsinon ;

fintanque ;

Rôles de la tête et de la queue



Tête:

1- Déterminer l'opération à exécuter
q_{j+1} = (add, sous ou rien)

2- Exécuter cette opération sur les chiffres de tête

3- Recoder le résultat pour éliminer le chiffre poids fort r_{-2} en l'annulant

Queue:

1- Exécuter l'opération q_{j+1} = (add, sous ou rien) sur les chiffres de queue sans retenue propagé

2- Transcoder Q_{j+1} de notation redondante BS à conventionnelle

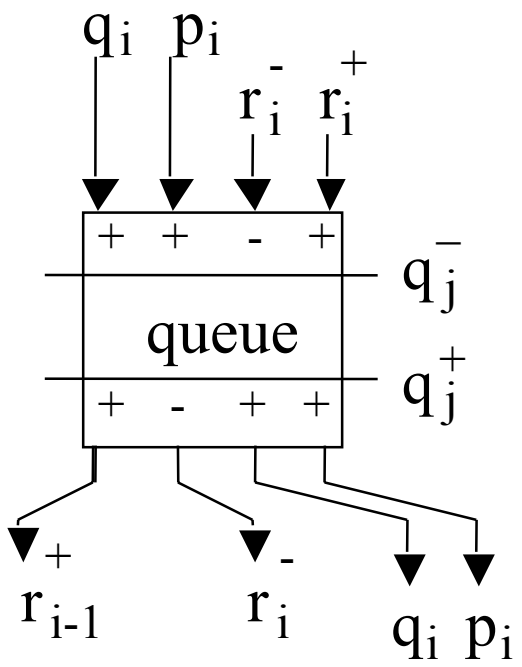
Équations de la queue

Remarques:

1- Pour chaque tranche, R_{2j+2} a 1 chiffre de moins à gauche et 2 chiffres de plus à droite que R_{2j}.

2- Pour chaque tranche Q_{j+1} et P_{j+1} ont un bit de plus à droite que Q_j et P_j.

3- $P_j = (Q_j - 2^{-j})$. Donc $-Q_j = -P_j - 2^{-j} = \overline{P_j} + 2^{-j} - 2^{-j} = \overline{P_j}$.



Equation arithmétique

$$2 * r^{+}_{i-1} + \overline{r^{-}_{i}} = r^{+}_{i} + \overline{r^{-}_{i}} + (q^{+}_{j} \wedge \overline{p_i} \vee q^{-}_{j} \wedge q_i)$$

Conversion

$$q_i = (q^{-}_{j} \wedge p_i \vee \overline{q^{-}_{j}} \wedge q_i)$$

$$p_i = (q^{+}_{j} \wedge p_i \vee \overline{q^{+}_{j}} \wedge q_i)$$

Un algorithme d'extraction de racine carrée rapide a été proposé:

- Les Additions/Soustraction ne propagent pas la retenue

- Le calcul, le recodage et le test du reste partiel sont effectués simultanément (et non séquentiellement)

- Les lignes longues et leurs amplis sont éliminées du chemin critique

- L'algorithme de conversion "a la volée" de la racine carrée, introduit dans la division (page ..), peut être utilisé.

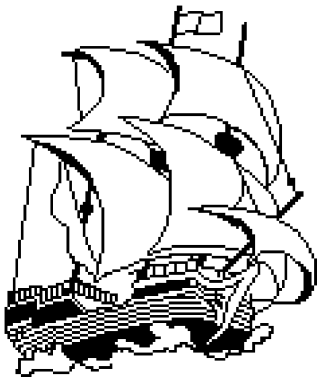
Virgule flottante

Dieu a créé les entiers naturels, tout le reste a été fait par l'homme L. Kronecker

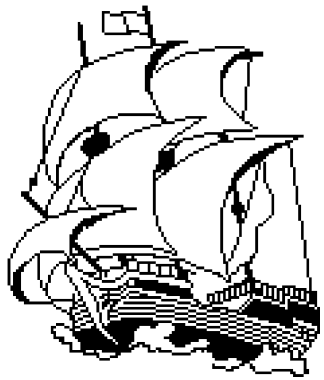
Alain GUYOT

Concurrent Integrated Systems
TIMA

(33) 04 76 57 46 16
Alain.Guyot@imag.fr
http://tima-cmp.imag.fr/Homepages/guyot
Techniques de l'Informatique et de la Microélectronique pour l'Architecture. Unité associée au C.N.R.S. n° B0706



Dieu a créé les entiers naturels, tout le reste a été fait par l'homme L. Kronecker



But de la virgule flottante: Représentation et calcul des nombres réels.

Approximés par des rationnels (avec une certaine erreur)

But du "standard": assurer la portabilité des logiciels de calcul.

Problèmes d'implémentation: les opérations sur les réels sont assez complexes et ont une grande influence sur les performances de la machine
La puissance de calcul se mesure en MFLOP (million de flottant par seconde). Actuellement de 5 à 200.

Solutions: 1- Anticipation
2- Prédiction
3- Spéculation

Standard ANSI/IEEE 754-1985 for Binary Floating-Point Arithmetic

Le standard spécifie:

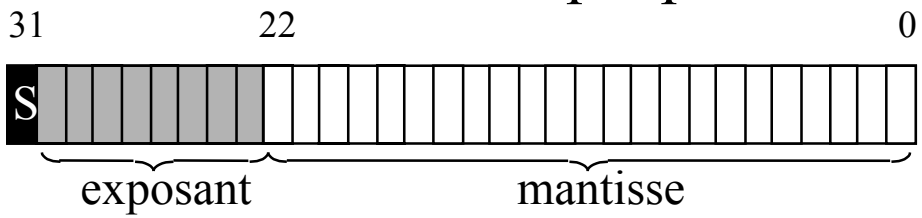
1-Les formats virgule flottante simple et double précision normalisés
2-Les échappement du format: ±0, ± ∞ , dénormalisé, nonnombres (NaN)
3-Les opérations addition, soustraction, multiplication, division, racine carrée, reste et comparaison (pas de fonction prévue)
4-Les conversions entre entiers et virgule flottante
5-Les conversions entre formats virgule flottante
6-Les conversions entre virgule flottante et chaîne décimales
7-Les modes d'arrondi (très important)
8-Les exceptions et leurs traitement

http://www.loria.fr/serveurs/CCH/documentation/IEEE754

Format IEEE 754-1985 Réels Normalisés

Mantisse normalisée
$1 \leq m < 2, \quad m = 1,f$

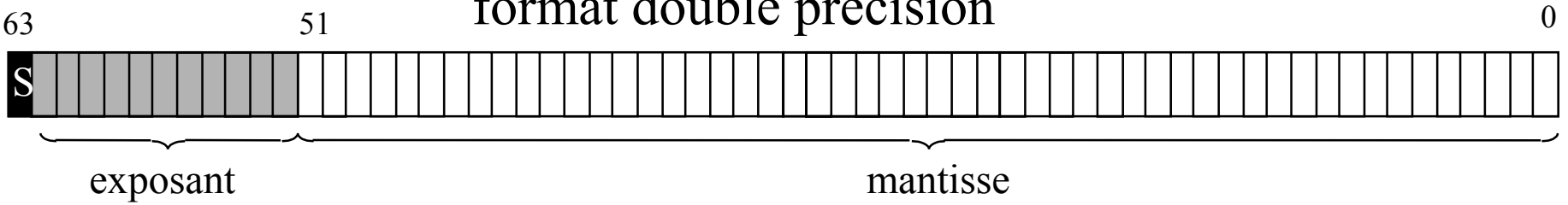
format simple précision



Calcul de la valeur de V

$$V = (-1)^{r_{31}} \times 2^{\left(\sum_{i=0}^{7} r_{i+23}\, 2^i - 127\right)} \times \left(\frac{2^{23} + \sum_{i=0}^{22} r_i \; 2^i}{2^{23}} \right)$$

format double précision



Champs et bits dans les champs rangés par importance décroissante

Normalisation de la mantisse (ou significande)

Avantages

1- Notation unique

$$11{,}00 * 2^{-1} = 3 * 1/2 \quad \textit{non valide}$$

$$1{,}10 * 2^{0} = 1{,}5 \quad \in [\,1, 2\,[$$

$$0{,}11 * 2^{1} = 0{,}75 * 2 \quad \textit{non valide}$$

2 - "1" avant la virgule implicite (peut être omis ou caché)

Inconvénients

1 - La valeur "0" ne s'exprime pas

2 - Les valeur "petites" ($< 2^{\text{min expo}}$) ne s'expriment pas

Représentation “biaisée” de l’exposant

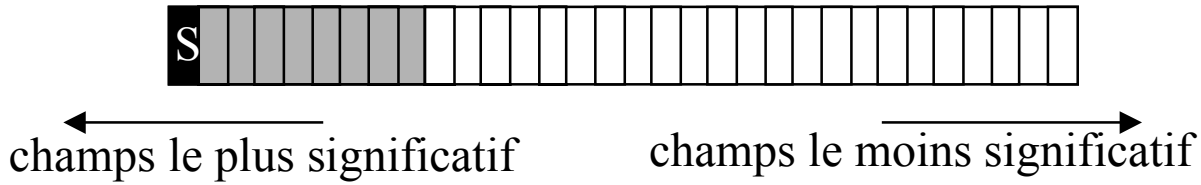
Avantages

Pas de "bit de signe".

1- Comparaison de nombres:

nombres en virgule flottante ≡ entiers

(les champs sont par ordre de signification)



2- Comparaison d’exposant

Inconvénients

Lorsqu’on ajoute deux exposants, il faut rajouter le biais

Lorsqu’on soustrait deux exposants, il faut retrancher le biais.

Remarque: la représentation biaisée (biased) s’appelle également la représentation par excès (excess)

Format IEEE 754-1985: Limites

Longueur totale	32 bits	64 bits
mantisse + bit implicite	23 + 1 bits	52 + 1 bits
exposant	8 bits	11 bits
biais, max, min	127, +127, -126	1023, +1023, -1022
domaine approximatif	$2^{128} \approx 3{,}8 \times 10^{38}$	$2^{1024} \approx 9 \times 10^{307}$
précision approximative	$2^{-23} \approx 10^{-7}$	$2^{-52} \approx 10^{-15}$
plus petit nombre normalisé	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
plus petit nombre ≠ 0	2^{-148}	2^{-1073}

Mantisse normalisée $\Rightarrow$

1- Notation spéciale du **0**

2- Notation spéciale de nombres "dénormalisés"

3- Notations spéciales pour $\infty, +\infty, -\infty$
et **NaN** (**N**ot **a** **N**umber)

Le rapport entre la masse de l'univers et celle du proton est d'environ 10^{78} (Paul Dirac)

Standard IEEE 754-1985
Échappements des formats

Le standard spécifie pour les simple précision:

1-Si e = 255 et m ≠ 0 alors v est **NaN**

2-Si e = 255 et m = 0 alors v est $(-1)^s \infty$

3-Si 0 < e < 255 alors v = $(-1)^s\ 2^{\,e-127}\ (1,m)$

4-Si e = 0 et m ≠ 0 alors v = $(-1)^s\ 2^{-126}\ (0,m)$ *(dénormalisé)*

5-Si e = 0 et m = 0 alors v = $(-1)^s\ 0$

Le standard spécifie pour les double précision:

1-Si e = 2047 et m ≠ 0 alors v est **NaN**

2-Si e = 2047 et m = 0 alors v est $(-1)^s\ \infty$

3-Si 0 < e < 2047 alors v = $(-1)^s\ 2^{\,e-1023}\ (1,m)$

4-Si e = 0 et m ≠ 0 alors v = $(-1)^s\ 2^{\,-1022}\ (0,m)$ *(dénormalisé)*

5-Si e = 0 et m = 0 alors v = $(-1)^s\ 0$

Standard IEEE 754-1985 Algèbre d'exceptions

	a	b	a + b	a * b	a ÷ b
	0	0	0	0	NaN
	0	y	z	0	0
	0	∞	∞	NaN	0
x > 0	x	0	z	0	∞
y > 0	x	y	0, z ou ∞	0, z ou ∞	0, z ou ∞
z > 0	x	∞	∞	∞	0
	∞	0	∞	NaN	∞
	∞	y	∞	∞	∞
	∞	∞	∞	∞	NaN
	∞	$-\infty$	NaN	$-\infty$	NaN
	NaN	0	NaN	NaN	NaN
	NaN	y	NaN	NaN	NaN
	NaN	∞	NaN	NaN	NaN

Standard IEEE 754-1985 Incohérence

		Exécution théorique	Exécution réelle
1:	a = Lnn (Lnn est le plus grand nombre représentable)		
2:	b = a + a	b = Lnn+Lnn = 2 Lnn	b = Lnn+Lnn = ∞
3:	c = b ÷ a	c = 2 (Lnn ÷ Lnn) = 2	c = ∞ ÷ Lnn = ∞
4:	d = 1 ÷ c	d = 1 ÷ 2 = 0,5	d = 1 ÷∞ = 0
5:	e = 1 ÷ (d - 0,5)	e = 1 ÷ (0,5 - 0,5) = ∞	e = 1 ÷ (0 - 0,5) = -2

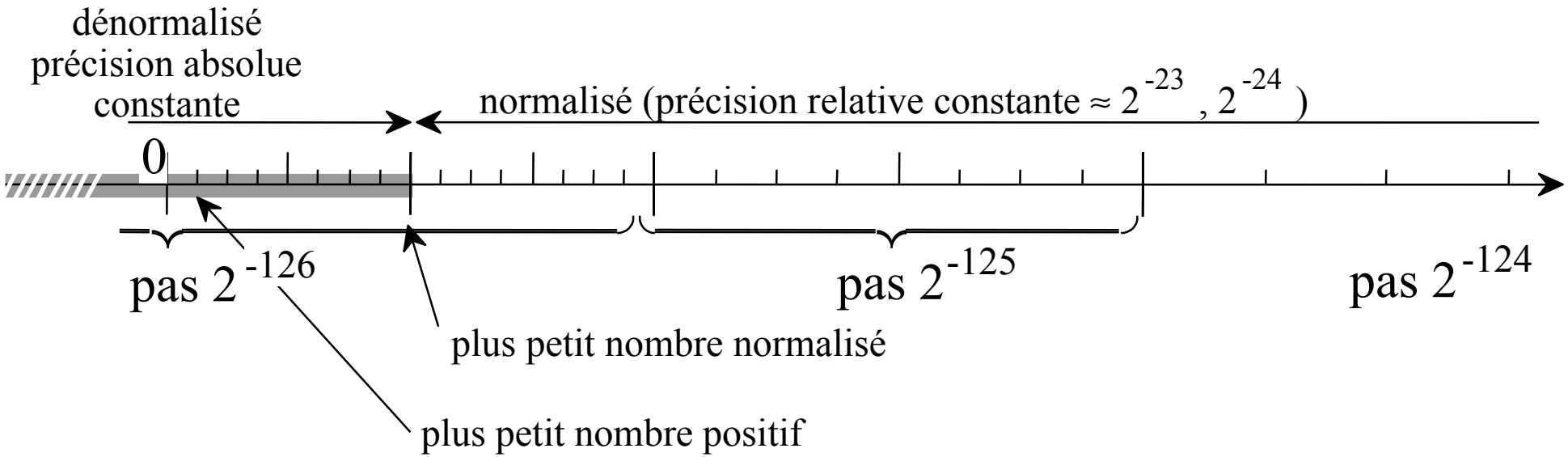
" *It makes me nervous to fly an airplane since I know they are designed using floating-point arithmetic* "

Anton Householder, un des pères de l'algorithmique numérique

Nombres dénormalisés (1)

optionnels

But: combler le trou entre le plus petit nombre normalisé et 0.
On conserve la précision absolue de ce nombre, avec une précision relative qui se dégrade.



Les nombre dénormalisés sont seulement recommandés par la norme.
Leur usage est coûteux en délai et en matériel;
Un mode de calcul permet d'éviter leur usage pour une exécution plus rapide.

Nombres dénormalisés (2)

Un résultat "très petit" ($< 2^{\text{min expo}}$) peut être produit par:

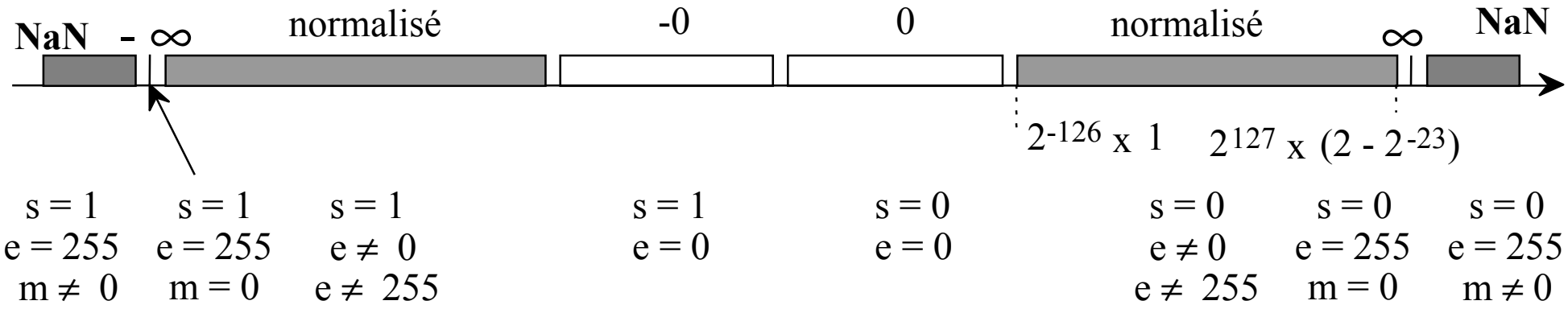
1- Une soustraction de deux nombres "petits"
La soustraction de deux nombres "grands" donne un nombre "grand" ou zéro.

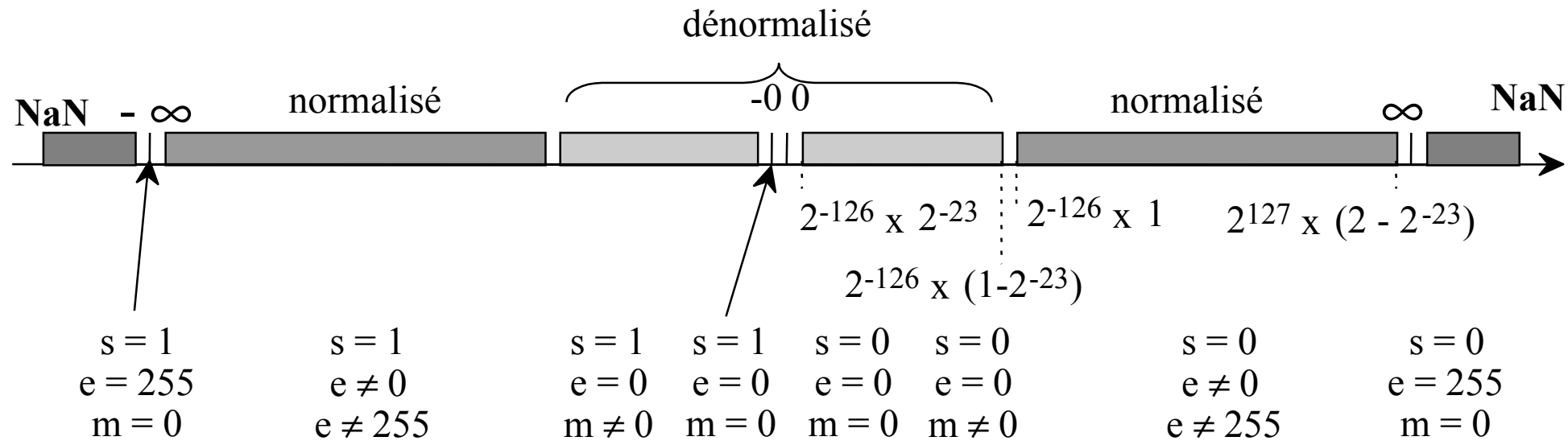
2 - Une multiplication de deux nombres "petits"

3- Une division d'un dividende "petit" par un diviseur "grand"

Le matériel pour traiter le cas 1 et les cas 2 et 3 est très différent.

Domaines de representation du 32 bits

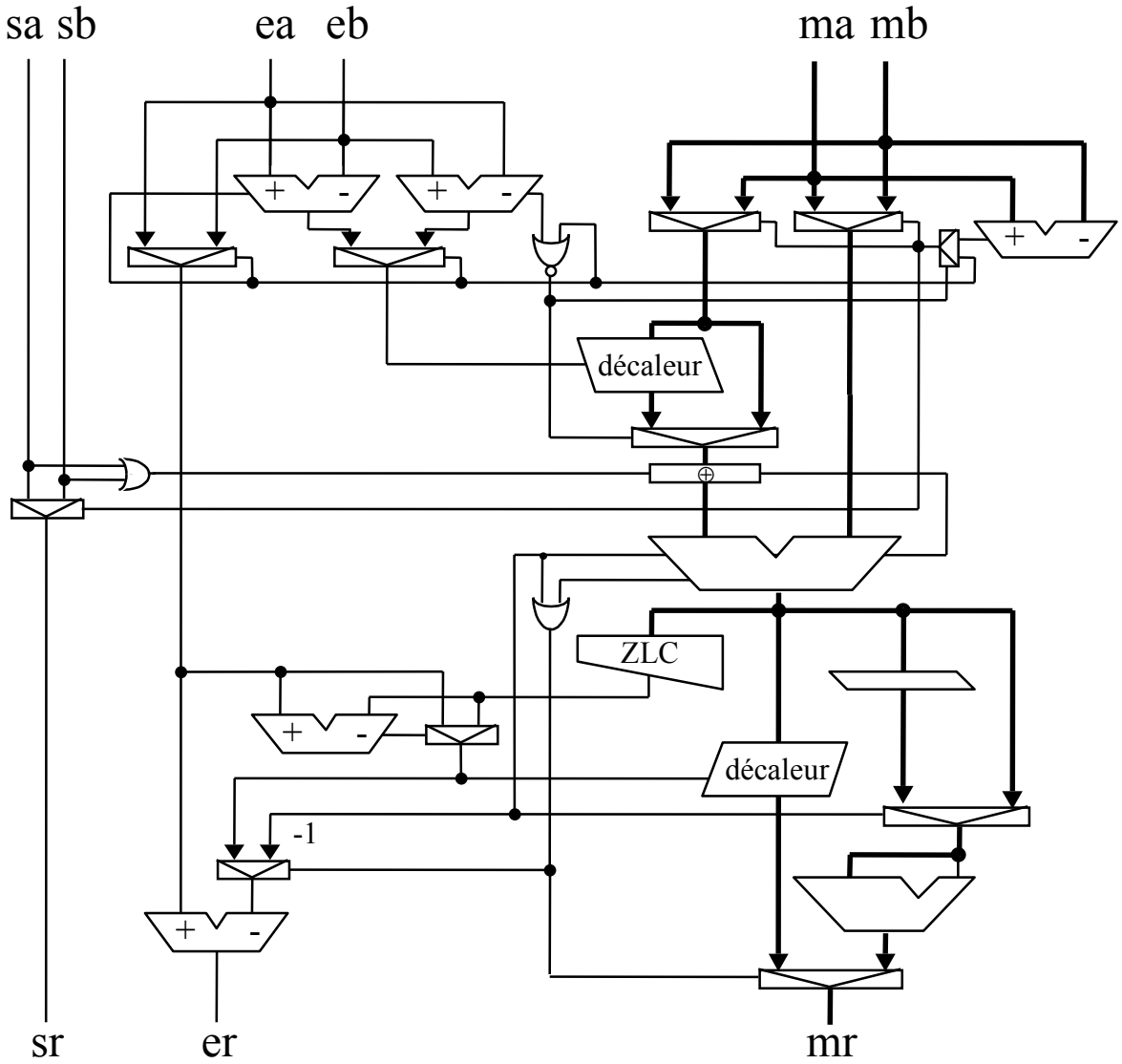




Exemples d'addition

Nombres Décimaux	Binaire en virgule flottante	Opérandes Alignés	Résultat normalisé
1 $+\ 0{,}75$	$1{,}0 * 2^{0}$ $1{,}1 * 2^{-1}$	$1{,}0 * 2^{0}$ $0{,}11 * 2^{0}$ $1{,}11 * 2^{0}$	$1{,}11 * 2^{0}$
1 $+\ 1{,}5$	$1{,}0 * 2^{0}$ $1{,}1 * 2^{0}$	$1{,}0 * 2^{0}$ $1{,}1 * 2^{0}$ $10{,}1 * 2^{0}$	$1{,}01 * 2^{1}$
1 $-\ 0{,}75$	$1{,}0 * 2^{0}$ $1{,}1 * 2^{-1}$	$1{,}0 * 2^{0}$ $-0{,}11 * 2^{0}$ $0{,}01 * 2^{0}$	$1{,}0 * 2^{-2}$

Addition flottante



tri du plus grand et du plus petit

alignement de la mantisse du plus petit

complémentation éventuelle du plus petit

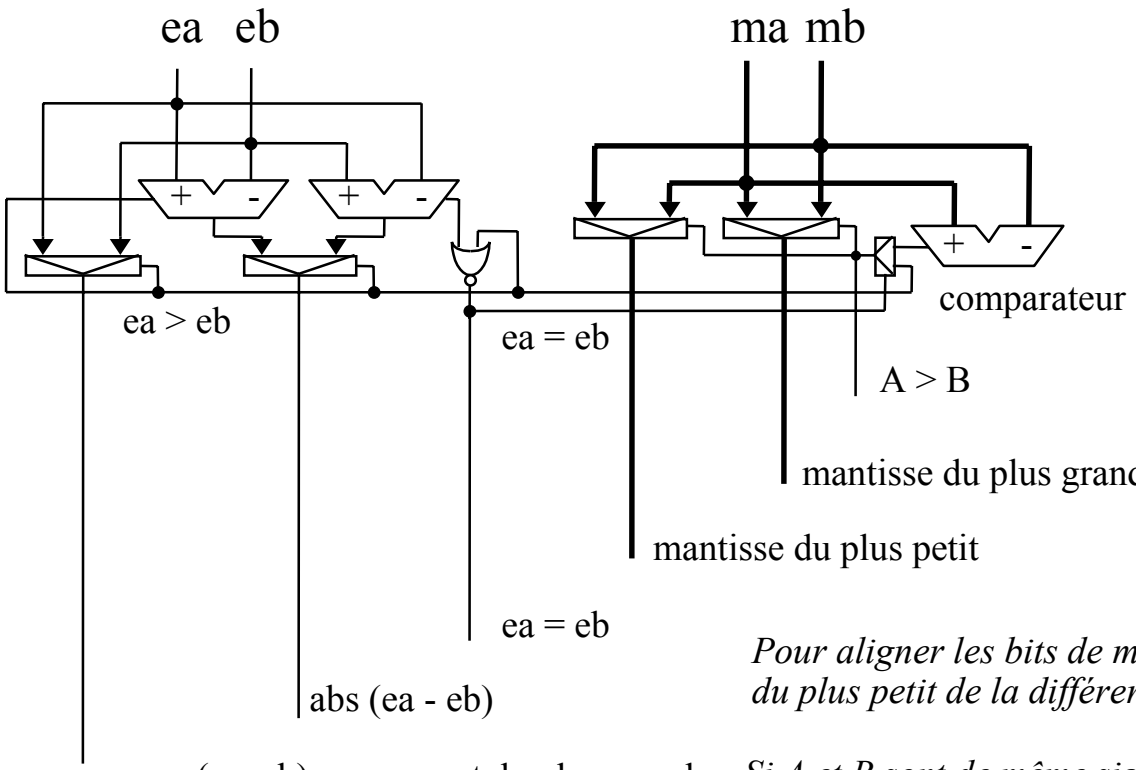
exécution de l'opération (addition ou soustraction)

normalisation (si possible)

arrondi du résultat

Addition (1)

Sélection du plus grand



Le signe du résultat est le signe du plus grand

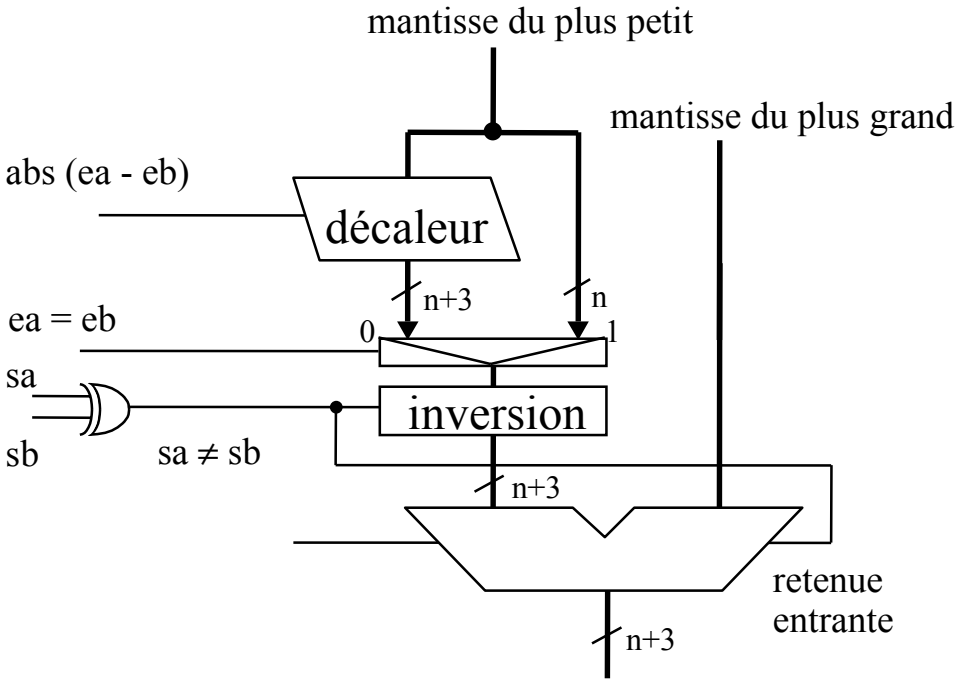
Les mantisses étant positive, il faut soustraire la plus petite de la plus grande

Pour aligner les bits de même poids, il faut décaler la mantisse du plus petit de la différence des exposants

Si A et B sont de même signe, l'exposant du résultat aura celle valeur ou cette valeur plus 1 (dépend de la retenue sortante).
Si A et B sont de signe différent, l'exposant du résultat sera compris entre cette valeur et cette valeur moins le nombre de bits de la mantisse

Addition (2)

Alignement des mantisses



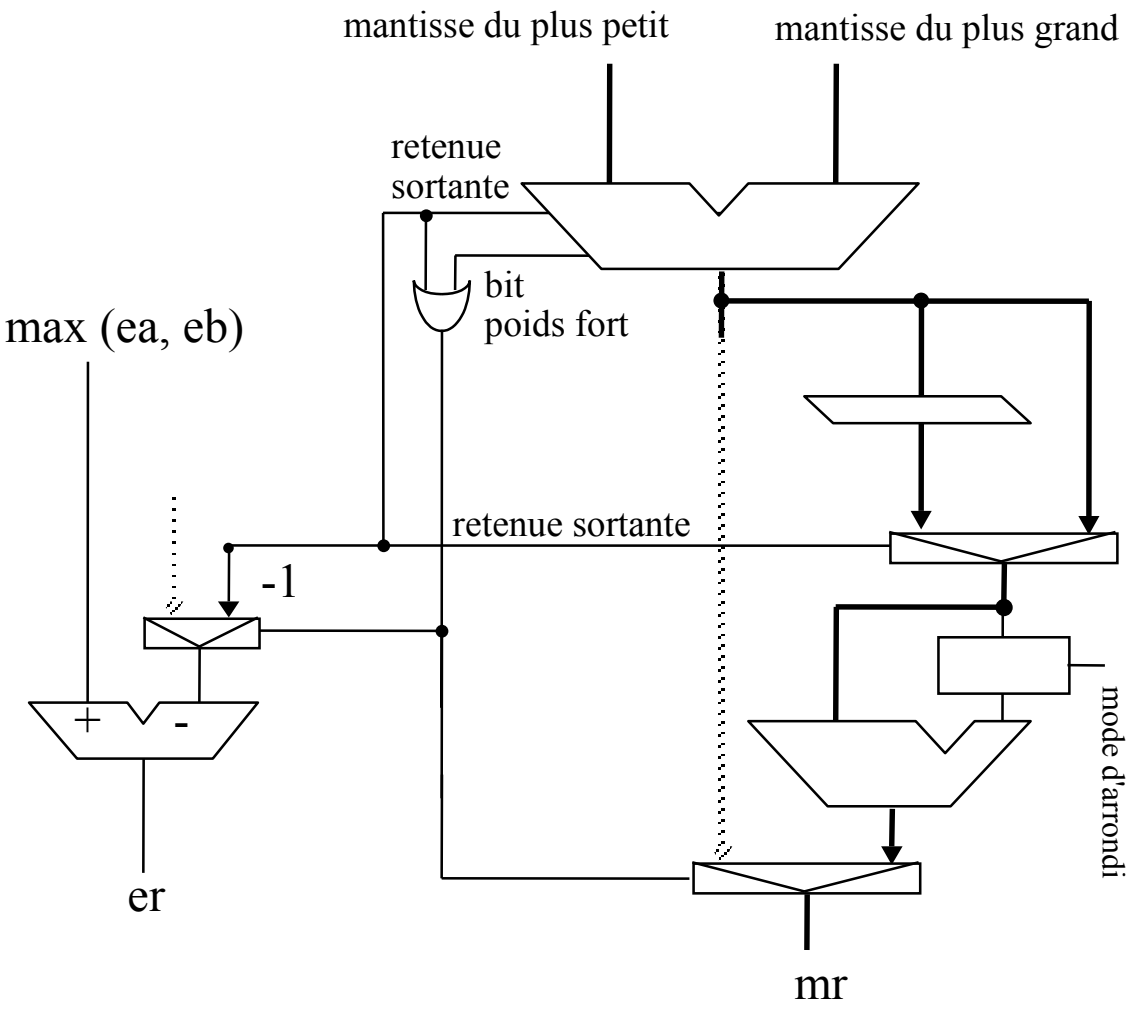
Pour aligner les bits de même poids, on décale vers la droite celui de plus petit exposant.

On garde 3 des bits sortant du décalage:
G: garde
R: arrondi
C: persistant, "ou logique" de tous les bits restant.

On soustrait toujours la mantisse du plus petit de la mantisse du plus grand

Pour aller plus vite, on peut ne pas attendre de savoir quelle est la mantisse du plus grand lorsque les exposants sont égaux en doublant l'additionneur et en choisissant le résultat positif après une soustraction.

Addition (3) - Normalisation du résultat

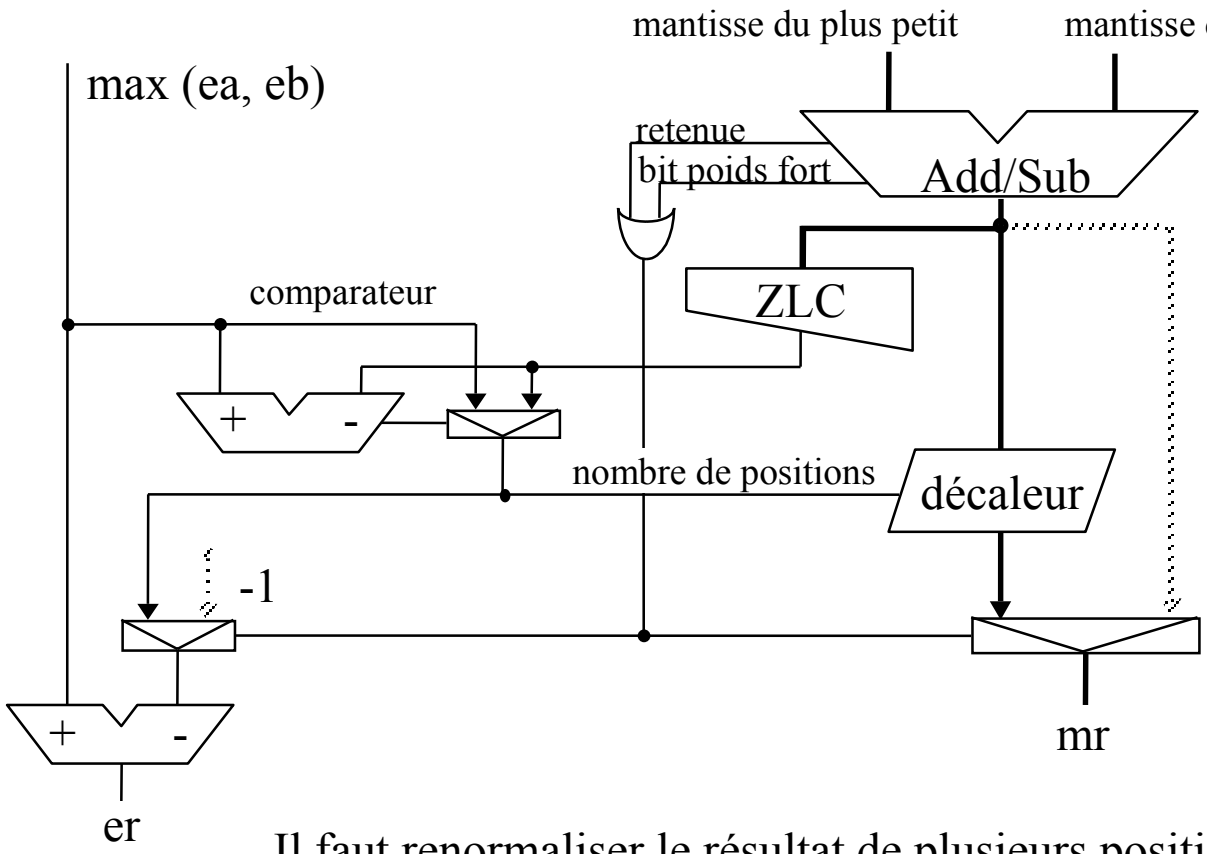


Si la retenue sortante vaut 1 alors décaler le résultat d'une position à droite.

Si la retenue vaut 0 et le bit poids fort vaut 1 alors le nombre est normalisé ($1 \leq n < 2$)

Le résultat doit être arrondi suivant le mode choisi

Addition (3') - Normalisation du résultat



ZLD calcule le nombre n de bits à 0 en poids forts.

On décale la mantisse vers la droite de min (n, max(ea, eb)+1) positions pour la renormaliser tout en évitant un exposant négatif ou nul.

On soustrait le nombre de positions de l'exposant

Il faut renormaliser le résultat de plusieurs positions uniquement en cas de soustraction de nombre de même exposant.
Dans ce cas il n'y a pas d'arrondi
Pour aller plus vite, on peut prédire le nombre de Zéros en poids forts à partir des opérandes de la soustraction et non à partir du résultat de la soustraction

Arrondi

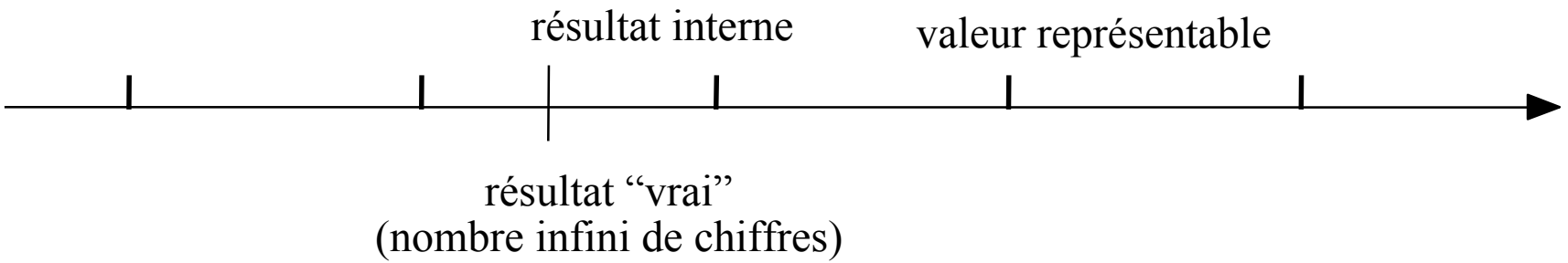
On effectue des opérations arithmétiques sur des réels représentables en format IEEE.

Le résultat d'une addition, d'une soustraction ou d'une multiplication peut être évalué sans erreur, mais le résultat peut ne pas être représentable en format IEEE.

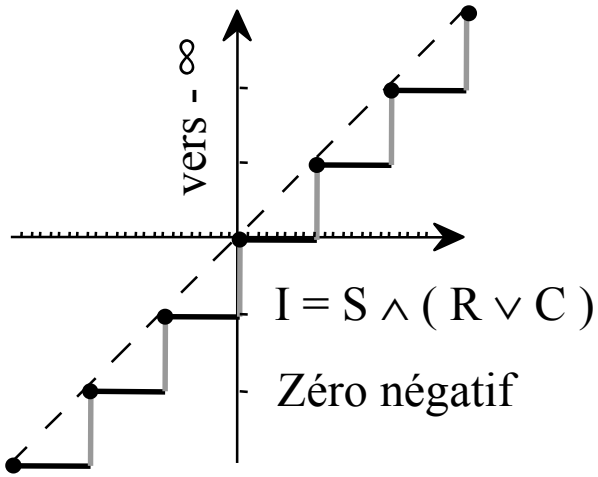
Le résultat d'une division ou d'une extraction de racine carrée peut ne pas être évaluable sans erreur sur un nombre fini de bits, (c'est pourquoi il y a un reste).

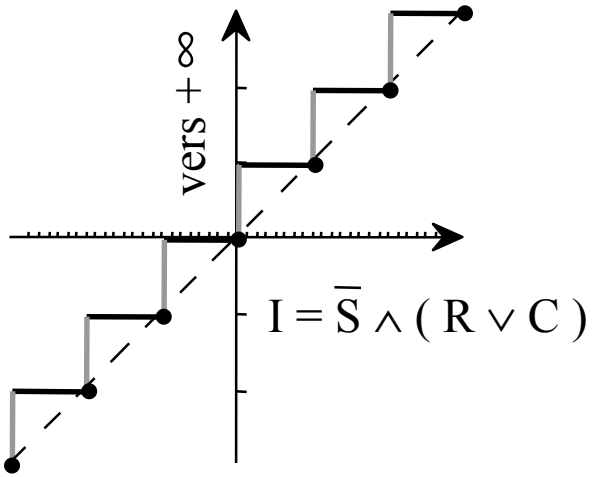
L'arrondi est destiné à trouver une représentation non exacte mais cependant acceptable.

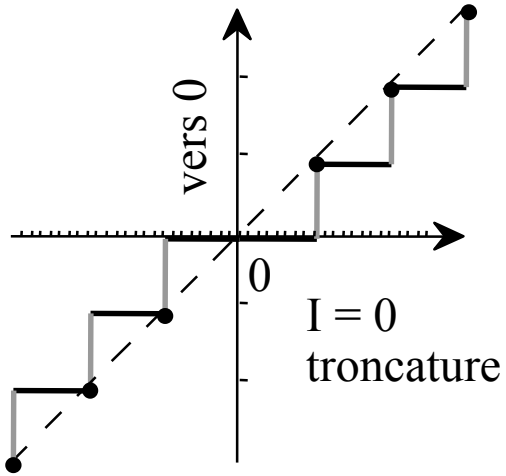
De nombreux travaux ont porté sur la génération et la propagation des erreurs d'arrondi.

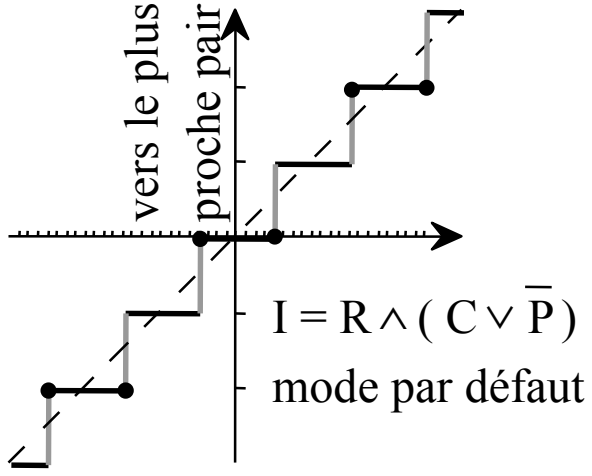


Modes d'arrondi (1)





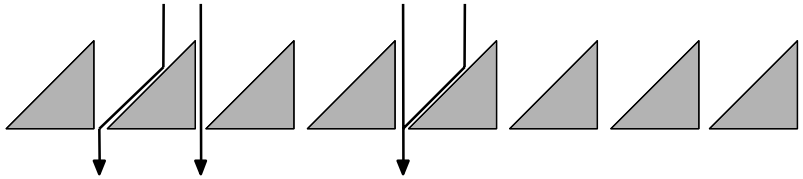




Modes d'arrondi (2)

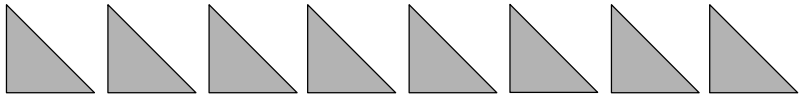
vers - ∞

Résultat de calcul

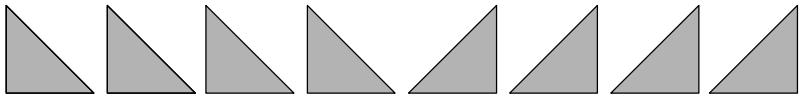


valeurs représentable en IEEE754

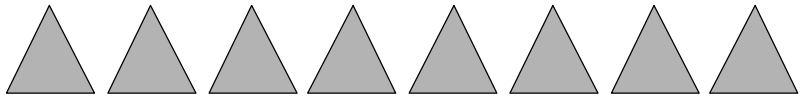
vers + ∞



vers 0



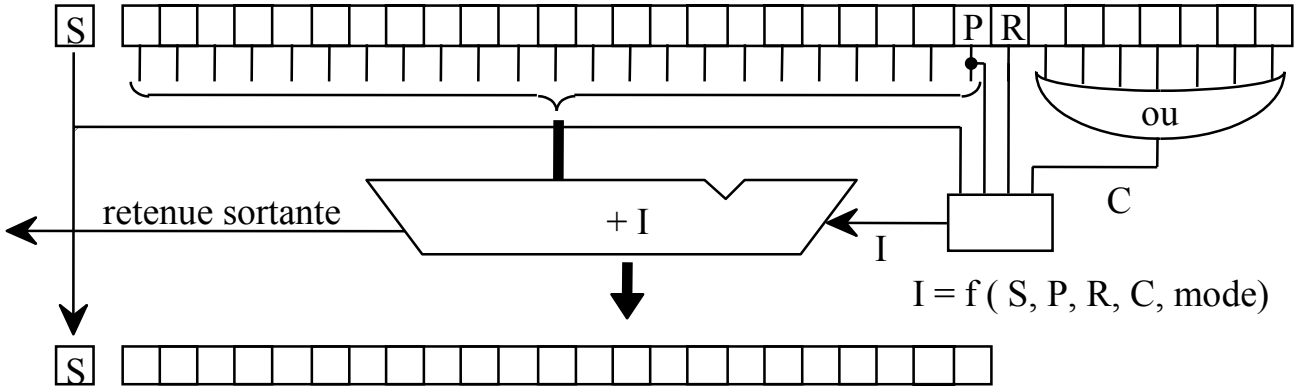
vers le plus proche



Matériel pour l'arrondi

Principe de l'arrondi

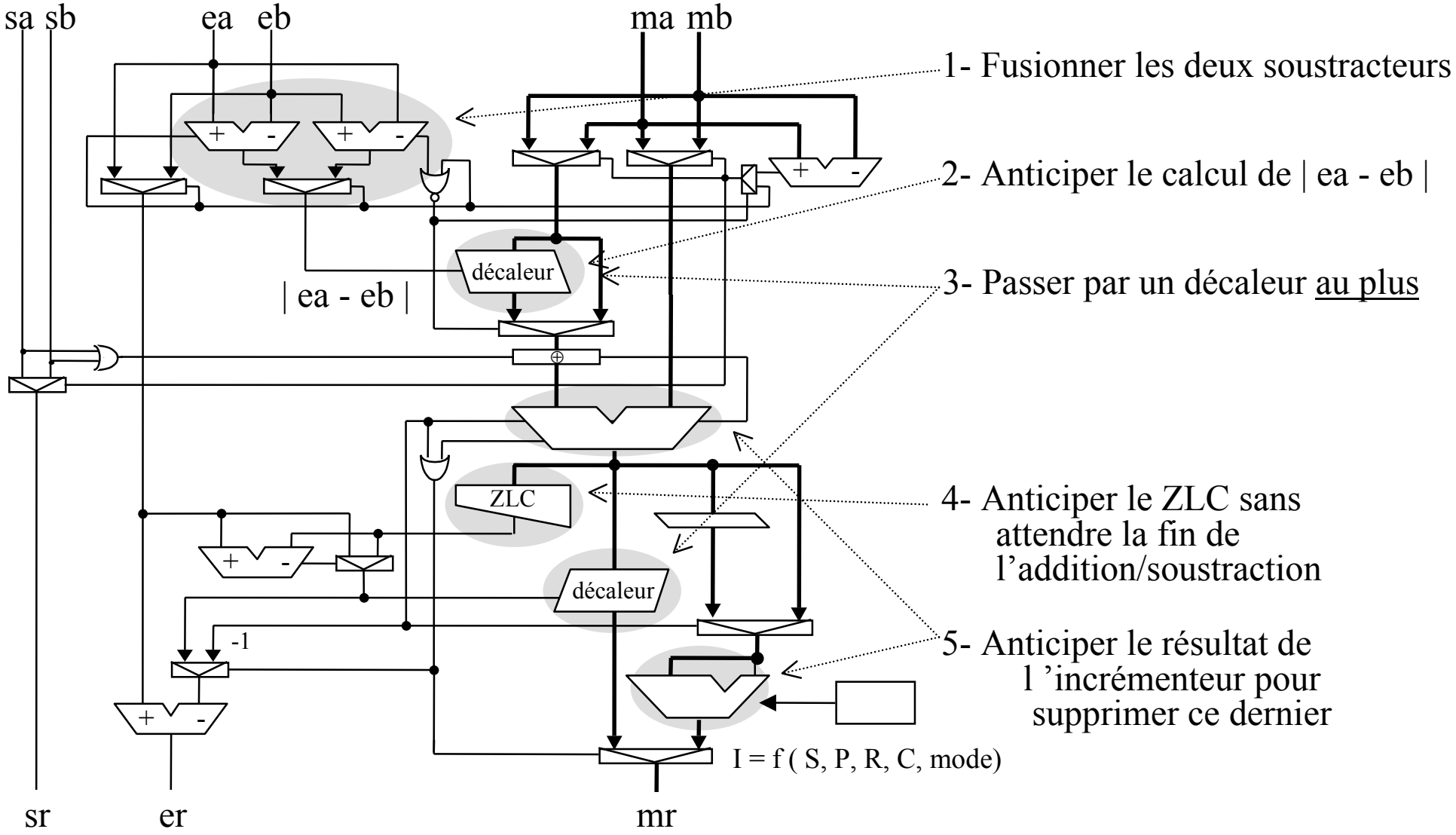
1- Le résultat est calculé en précision infinie (sans perte d'information)
2- Le résultat précédent est traité pour tenir dans un format virgule flottante.



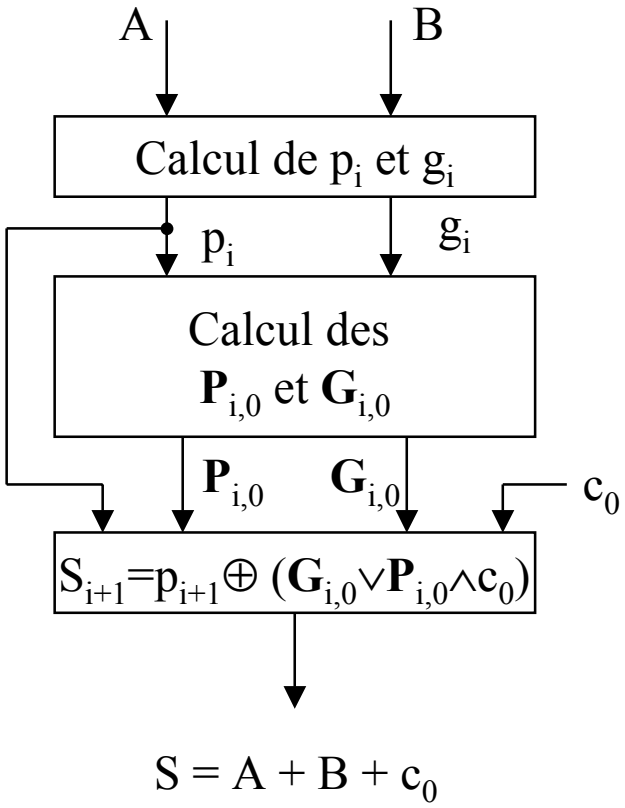
Si l'arrondi provoque une retenue sortante ≠ 0 alors le résultat est 2

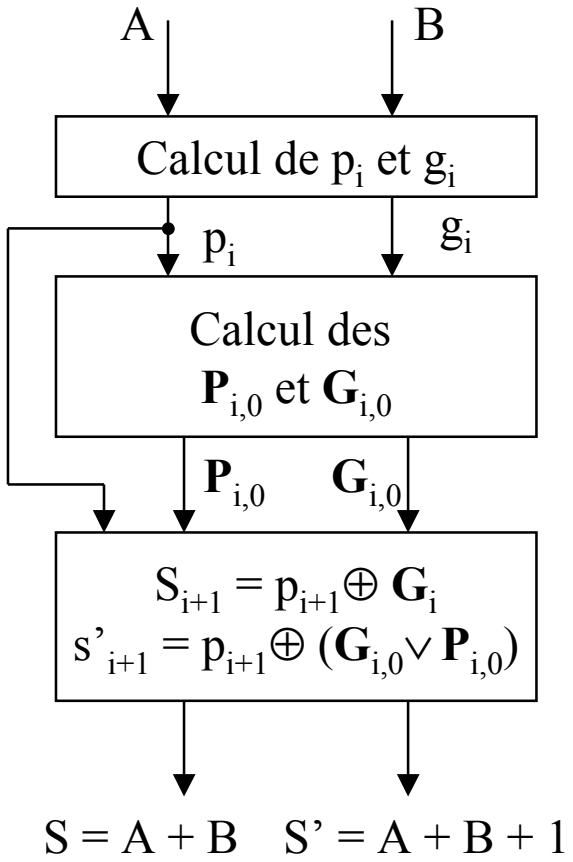
mode d'arrondi	valeur de I
vers + ∞	$I = \overline{S} \wedge (R \vee C)$
vers - ∞	$I = S \wedge (R \vee C)$
vers 0	$I = 0$
vers + proche	$I = R \wedge (C \vee \overline{P})$

Amélioration de l'Addition Flottante



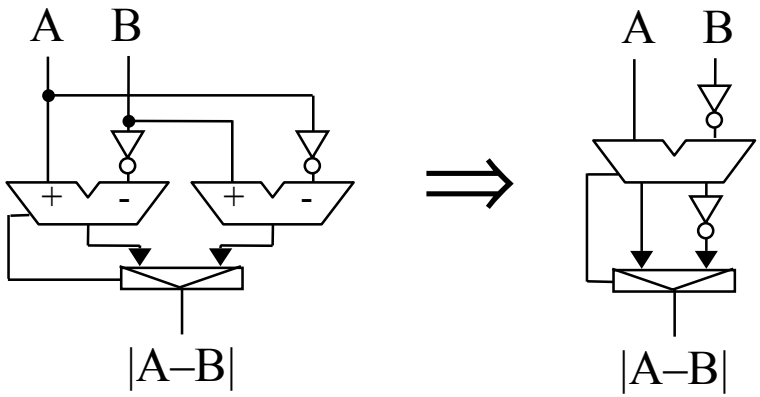
Addition à résultats multiples





On peut calculer <u>simultanément</u> S = A + B et S' = A + B + 1
On peut étendre à plus de 2 sorties

Valeur absolue de la différence



Pour calculer |A–B| on calcule simultanément A–B et B–A avec un additionneur à 2 sorties.

Soit $D = A + \overline{B} + 1 = A - B$.

Soit $S = \overline{A + \overline{B}} = \overline{A - B - 1} = -(A - B - 1) - 1 = B - A$.

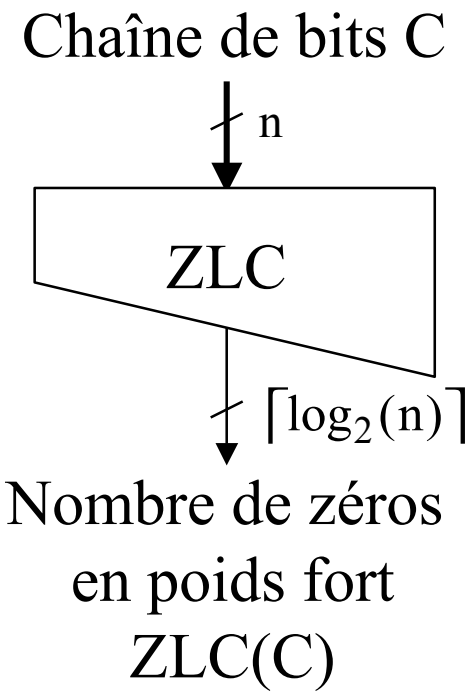
(on vérifie facilement que $D + S = 0$)

On calcule $p_i = a_i \oplus \overline{b_i}$ et $g_i = a_i \wedge \overline{b_i}$ $\forall i$

puis on calcule $G_{i,0}$ et $P_{i,0}$ avec un assemblage adéquat de cellules de Brent et Kung.

On a alors $d_i = p_i \oplus (G_{i,0} \vee P_{i,0})$ et $s_i = \overline{p_i \oplus G_{i,0}}$

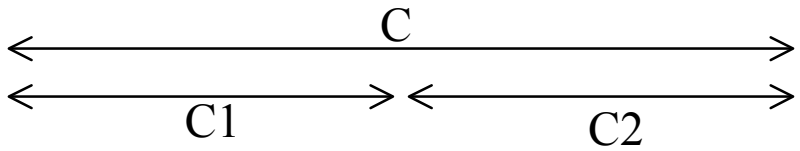
Comptage des zéros (1)



Soit une chaîne de bits C
Long(C) = longueur de la chaîne C (nombre de bits)
ZLC(C) = nombre de bits à zéro en poids forts de C

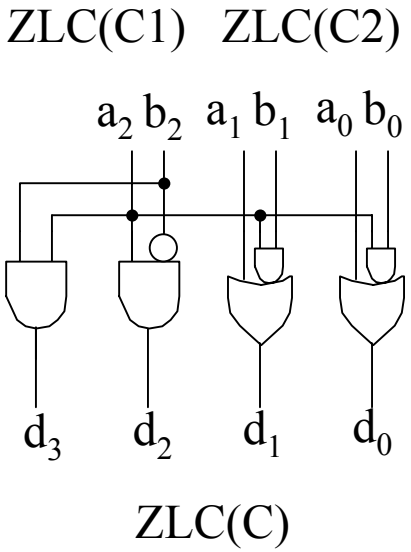
Comptage par dichotomie:
On coupe la chaîne C en deux chaînes C1 et C2.
Si ZLC(C1) < Long(C1) **alors** ZLC (C) = ZLC(C1)
sinon ZLC(C) = Long(C1) + ZLC(C2)

Pour remplacer la comparaison (<) par le test de 1 bit
et replacer l'addition (+) par une concaténation
les longueurs sont des puissances de 2.



Comptage des zéros (2)

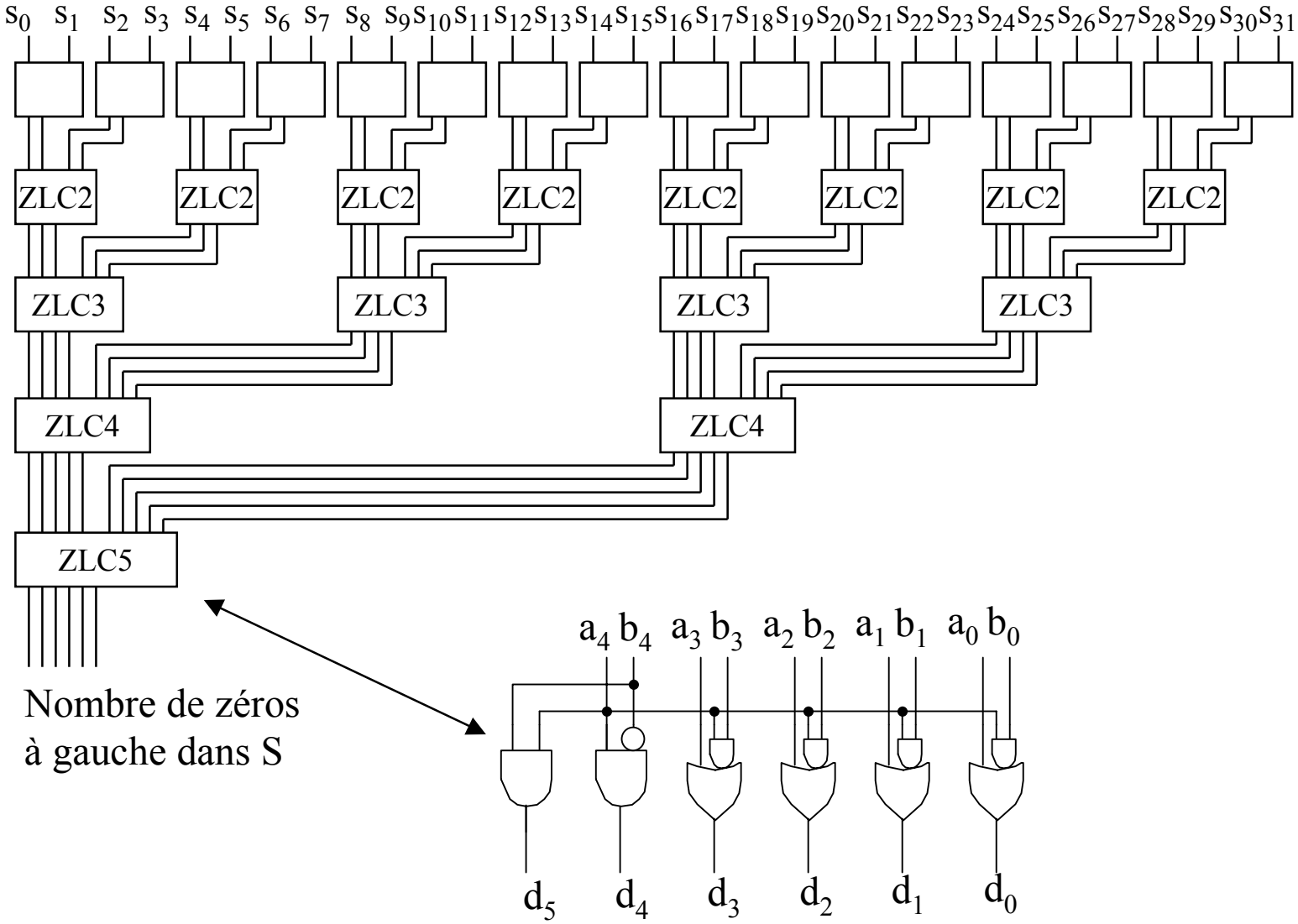
C		ZLC (C1)	ZLC (C2)	ZLC (C)
C1	C2	$a_2\,a_1\,a_0$	$b_2\,b_1\,b_0$	$d_3\,d_2\,d_1\,d_0$
1 x x x	x x x x	0 0 0	x x x	0 0 0 0
0 1 x x	x x x x	0 0 1	x x x	0 0 0 1
0 0 1 x	x x x x	0 1 0	x x x	0 0 1 0
0 0 0 1	x x x x	0 1 1	x x x	0 0 1 1
0 0 0 0	1 x x x	1 0 0	0 0 0	0 1 0 0
0 0 0 0	0 1 x x	1 0 0	0 0 1	0 1 0 1
0 0 0 0	0 0 1 x	1 0 0	0 1 0	0 1 1 0
0 0 0 0	0 0 0 1	1 0 0	0 1 1	0 1 1 1
0 0 0 0	0 0 0 0	1 0 0	1 0 0	1 0 0 0



Si ZLC(C1) < Long(C1) **alors** ZLC (C) = ZLC(C1)
sinon ZLC(C) = Long(C1) + ZLC(C2)

avec Long(C1) = Long(C2) = 2^i

Arbre de comptage des zéros



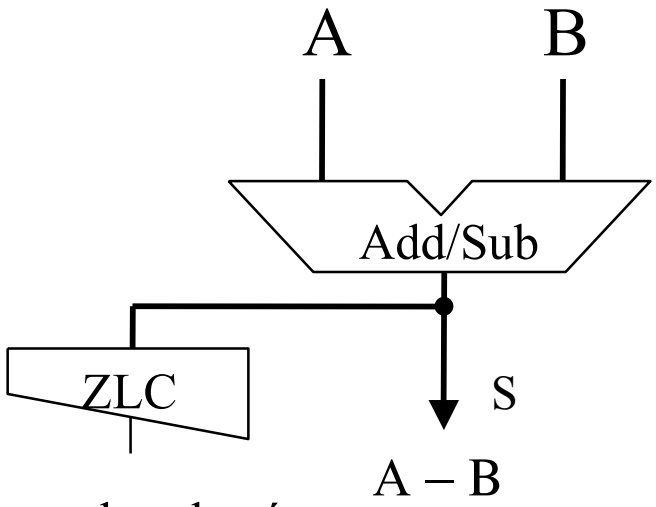
Anticipation du comptage des zéros (1)

Problème: prédire le nombre de zéros en poids forts de S = A – B sans calculer S.

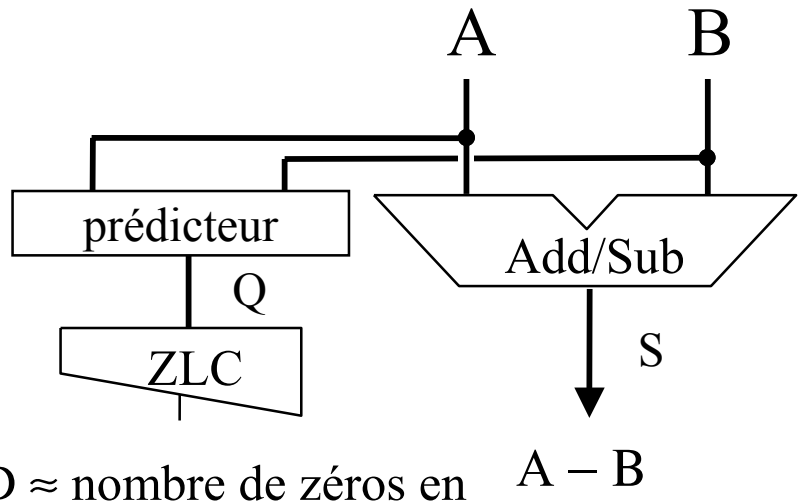
On veut une chaîne Q :

1- avec les mêmes zéros poids forts que S

2- avec un temps de calcul très court



D = nombre de zéros en poids forts du résultat



D ≈ nombre de zéros en poids forts du résultat

On n'a pas de prédicteur à la fois précis et rapide

Anticipation du comptage des zéros (2)

On peut prédire certains bits s_i de S à partir de p_i, g_i, k_i et p_{i+1}, g_{i+1}, k_{i+1} mais pas tous (p_{i+1} à droite)

k_i	k_{i+1}	$s_i = 0$
k_i	p_{i+1}	$s_i = x$
k_i	g_{i+1}	$s_i = 1$
p_i	k_{i+1}	$s_i = 1$
p_i	p_{i+1}	$s_i = x$
p_i	g_{i+1}	$s_i = 0$
g_i	k_{i+1}	$s_i = 0$
g_i	p_{i+1}	$s_i = x$
g_i	g_{i+1}	$s_i = 1$

On calcule la chaîne Q comme ci-dessous (on choisi des valeurs pour les x)

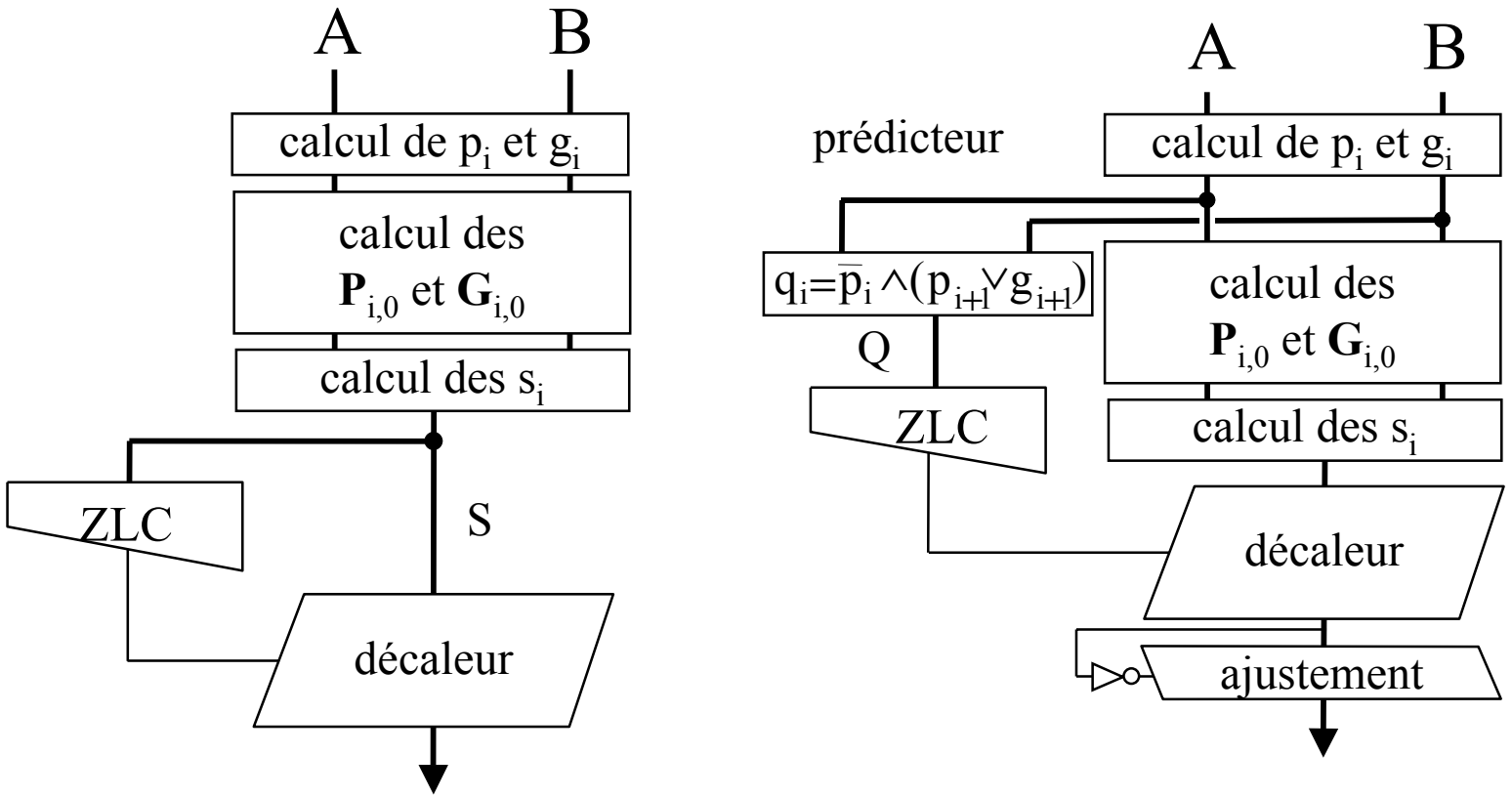
k_i	k_{i+1}	$q_i = 0$
k_i	p_{i+1}	$q_i = 1$
k_i	g_{i+1}	$q_i = 1$
p_i	k_{i+1}	$q_i = x$
p_i	p_{i+1}	$q_i = 0$
p_i	g_{i+1}	$q_i = 0$
g_i	k_{i+1}	$q_i = 0$
g_i	p_{i+1}	$q_i = 1$
g_i	g_{i+1}	$q_i = 1$

n'appartient pas à p* g k*
on choisi 0

$$q_i = (\overline{p_i} \wedge \overline{k_{i+1}})$$
$$= (\overline{a_i} \wedge \overline{b_i} \vee a_i \wedge b_i)$$
$$\wedge (a_{i+1} \vee b_{i+1})$$

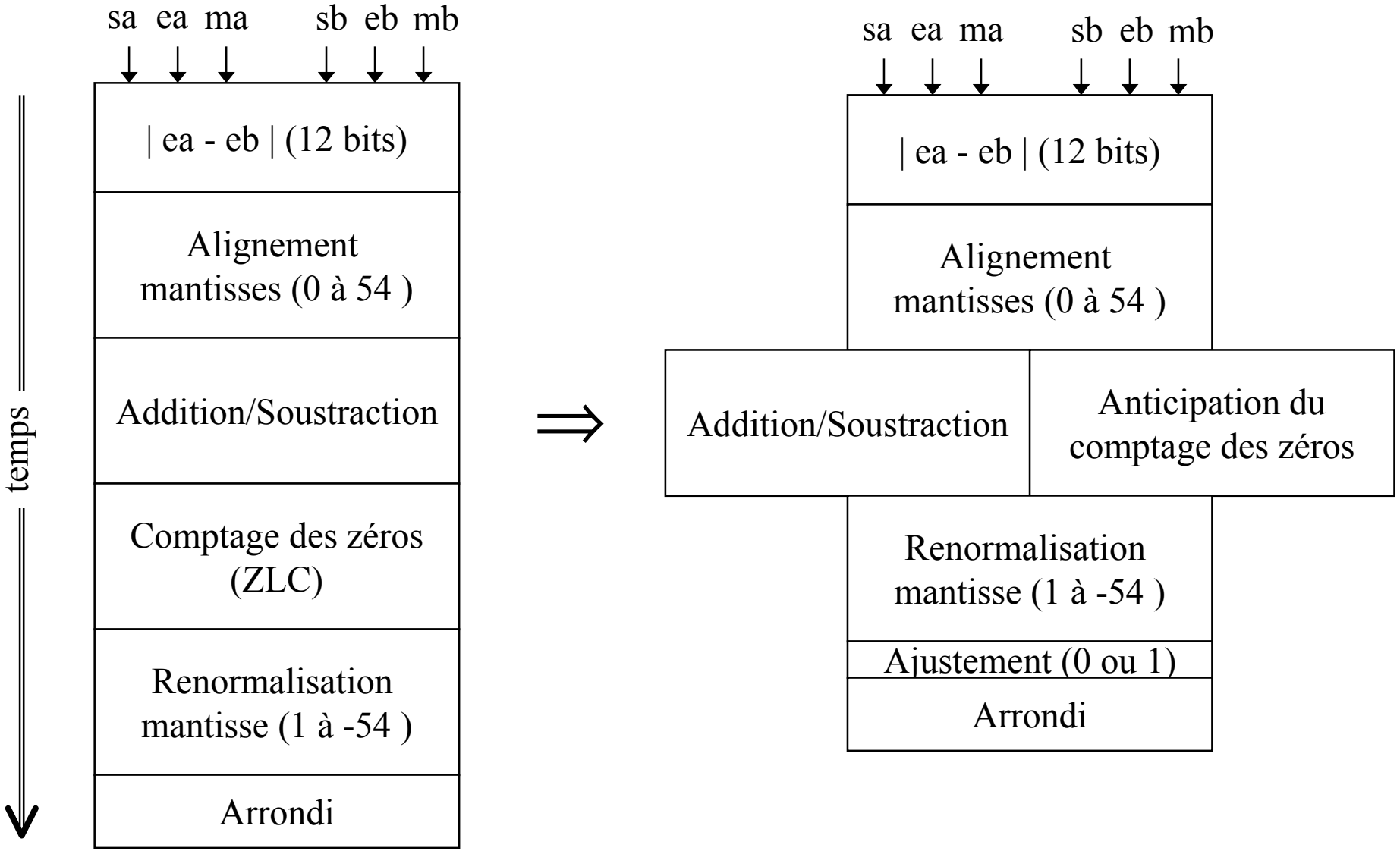
Alors la chaîne S = A - B , S ≥ 0 et la chaîne Q ont le même nombre de bits à zéro dans la chaîne p* g k*, c'est à dire le même nombre de zéros en poids forts à 1 près.

Anticipation du comptage des zéros (3)



L'imprécision du prédicteur demande un étage d'ajustement
$ZLD(S) \leq ZLD(S') \leq ZLD(S) + 1$

Anticipation du comptage des zéros (4)



Erreur de prédiction

Une chaîne de propagation de retenue (≠ p*) commence soit par p*k, soit par p*g.
Le début de chaîne p*k produit une retenue sortante à 0.
Le début de chaîne p*g produit une retenue sortante à 1.

Soit S = A - B avec A > B, S > 0.
Alors la chaîne de retenues commence par p*g.
Soit D' la longueur de la sous-chaîne p*g k*.

Soit D le nombre de zéros en tête de S.
Si la chaîne commence par p*g k*p* k alors D = D'
Si la chaîne commence par p*g k*p* g alors D = D'-1

En résumé:
On sait que la chaîne de propagation commence par p*g
On veut la longueur de la chaîne p*g k*
On veut savoir si la chaîne commence par p*g k*p*k.

Prédiction de l'ajustement (1)

Transcodage

$\overline{p}_i\,\overline{k}_{i+1} \Rightarrow q_i$

$p_{i-1}\,k_i \Rightarrow n_i$

autre $\Rightarrow z_i$

Chaîne à reconnaître

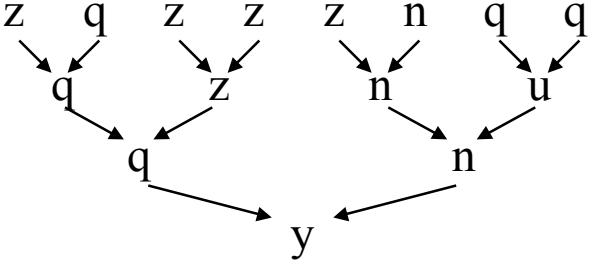
p* g k* p* k

transcodée en
z* q z* n

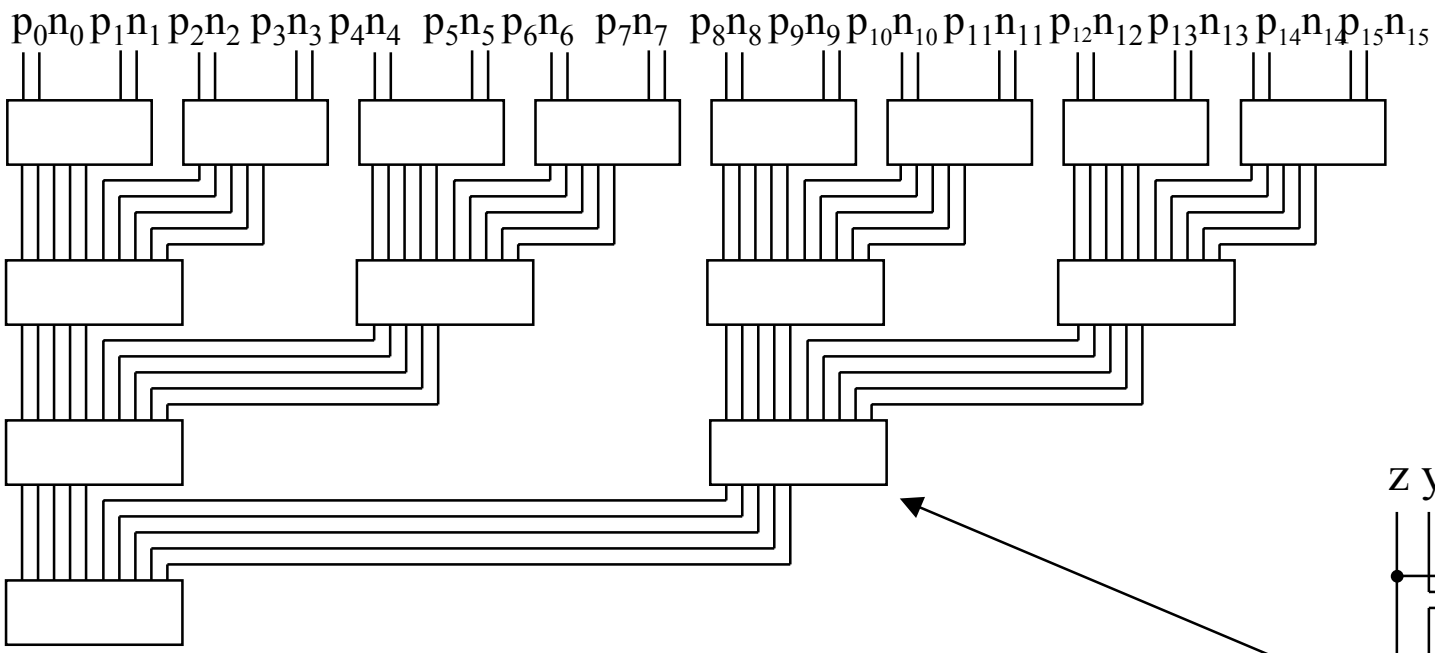
Concaténation de chaînes

chaîne de gauche \ chaîne de droite	z	q	n	y	u
z	z	q	n	y	u
q	q	u	y	u	u
n	n	n	n	n	n
y	y	y	y	y	y
u	u	u	u	u	u

A =	0	1	0	0	1	0	1	1
B =	0	0	0	0	1	1	0	0
A–B	0	1	0	0	0	$\overline{1}$	1	1
	z	q	z	z	z	n	q	q



Prédiction de l'ajustement (2)



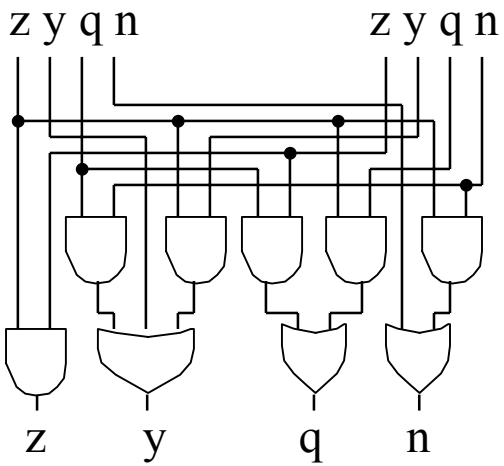
	z	q	n	y	u
z	z	q	n	y	u
q	q	u	y	u	u
n	n	n	n	n	n
y	y	y	y	y	y
u	u	u	u	u	u

$z = z\ z$

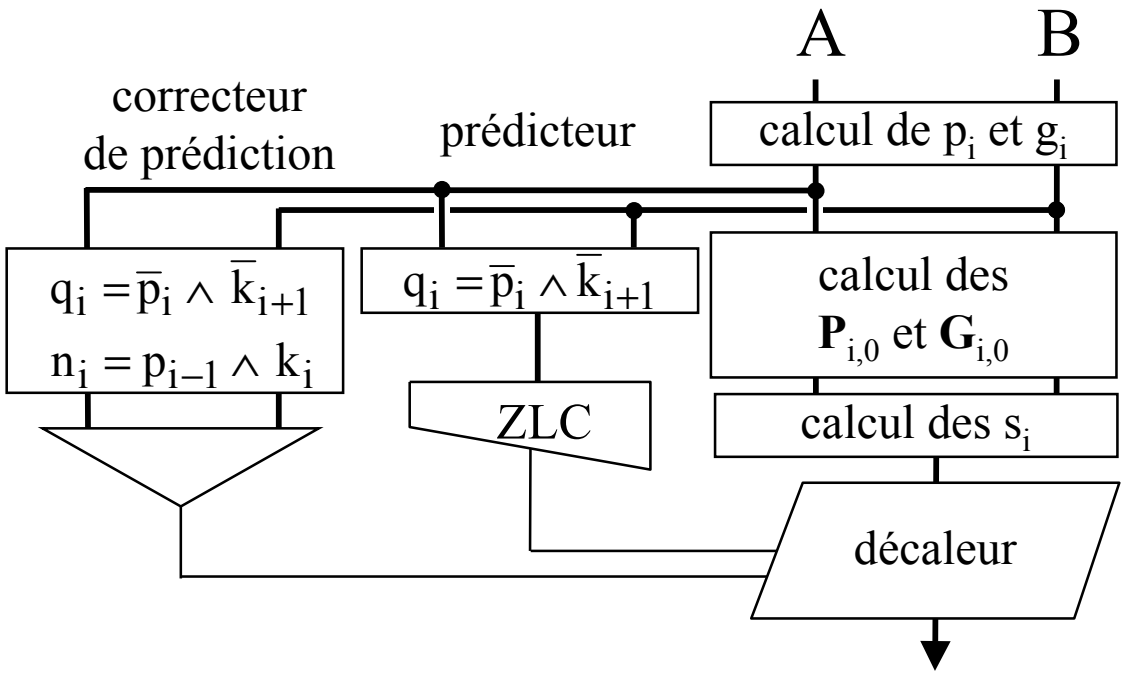
$y = q\ n \vee y\ \phi \vee z\ y$

$q = q\ z \vee z\ q$

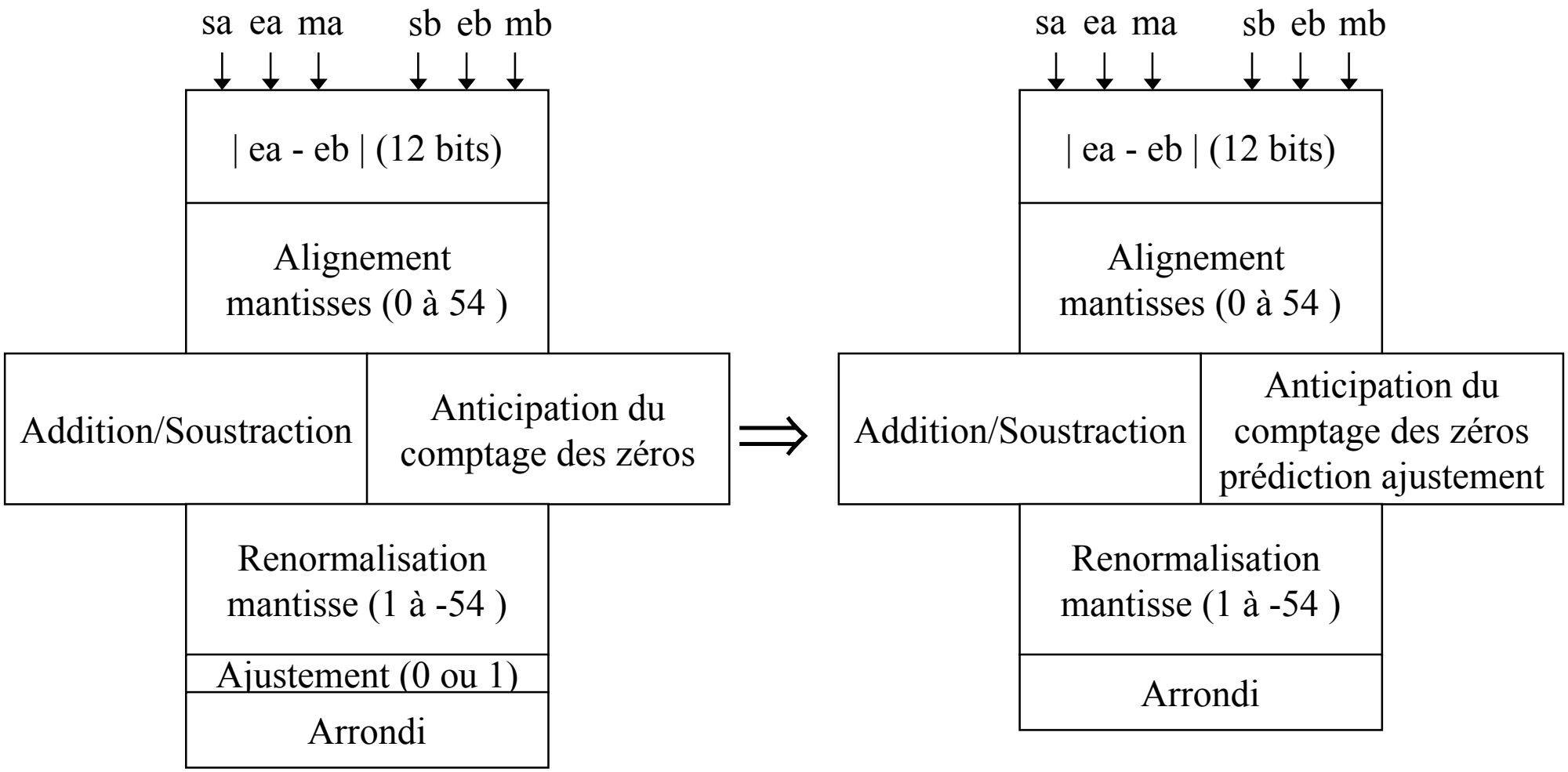
$n = n\ \phi \vee z\ n$



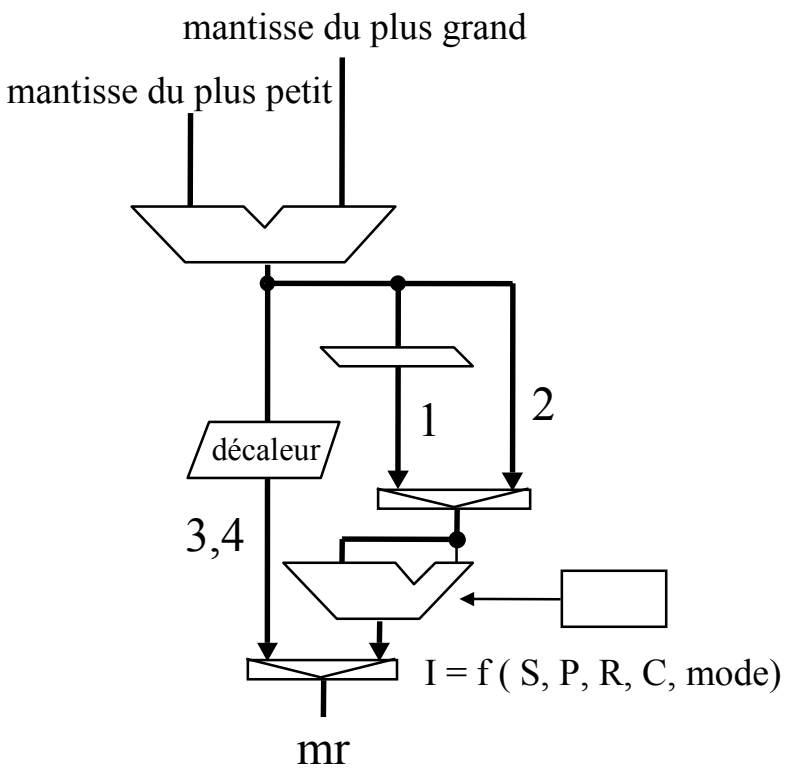
Prédiction de l'ajustement (3)



Prédiction de l'ajustement (4)



Renormalisation et arrondi de la mantisse



Pour être renormalisé, le résultat est soit
1- décalé à droite de 1 position
2- déjà normalisé
3- décalé à gauche de 1 position
4- décalé à gauche de 2 positions ou plus

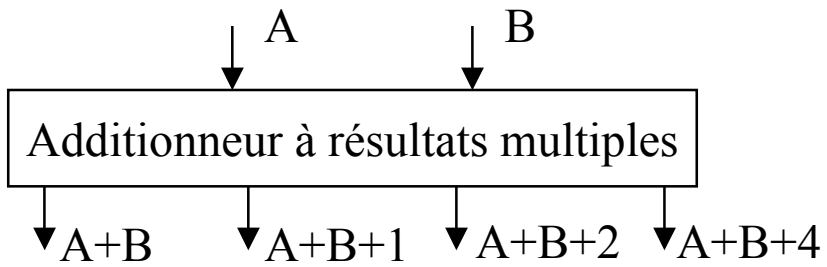
Dans le cas 4, $| ea - eb | \leq 1$, donc le décalage introduit à gauche 1 zéro au moins.

Seuls les cas 1, 2 et 3 peuvent provoquer une propagation de retenue pour l'arrondi.

On peut « précalculer » chacun de ces résultats avec un additionneur à résultats multiples

Arrondi spéculatif

On précalcule 1- la mantisse du résultat,
2 -la mantisse du résultat +2 lsb
3 -la mantisse du résultat +1 lsb
4 -la mantisse du résultat +1/2 lsb



Si l'arrondi du résultat est de
1- sans ajout (troncature)
2 -de une position à droite puis ajout
3 -de 0 position puis ajout
4 -de une position à gauche puis ajout

Alors le résultat est précalculé, sinon il n'y a pas besoin d'addition puisqu'il n'y a pas de propagation de retenue.

Addition/Soustraction spéculative (1)

On spécule que | ea - eb | > 1 Eloigné

Alignement

1 - calculer n = | ea - eb | sur 12 bits
(détermine si la spéculation est correcte)

2 - décaler la mantisse du plus petit de
n positions à droite

2' - Si n > 54 ou n ≤ 1 alors abandon.

Normalisation

3 - Le nombre de position est donné par les 3 premiers bits du résultat avant normalisation

4 - La mantisse du résultat est décalé:
soit 1 position à droite (addition)
soit 0 position (addition ou soustraction)
soit 1 position à gauche (soustraction)

On spécule que | ea - eb | ≤ 1 Proche

Alignement

1 - le nombre de positions est donné par les
2 derniers bits de ea et de eb

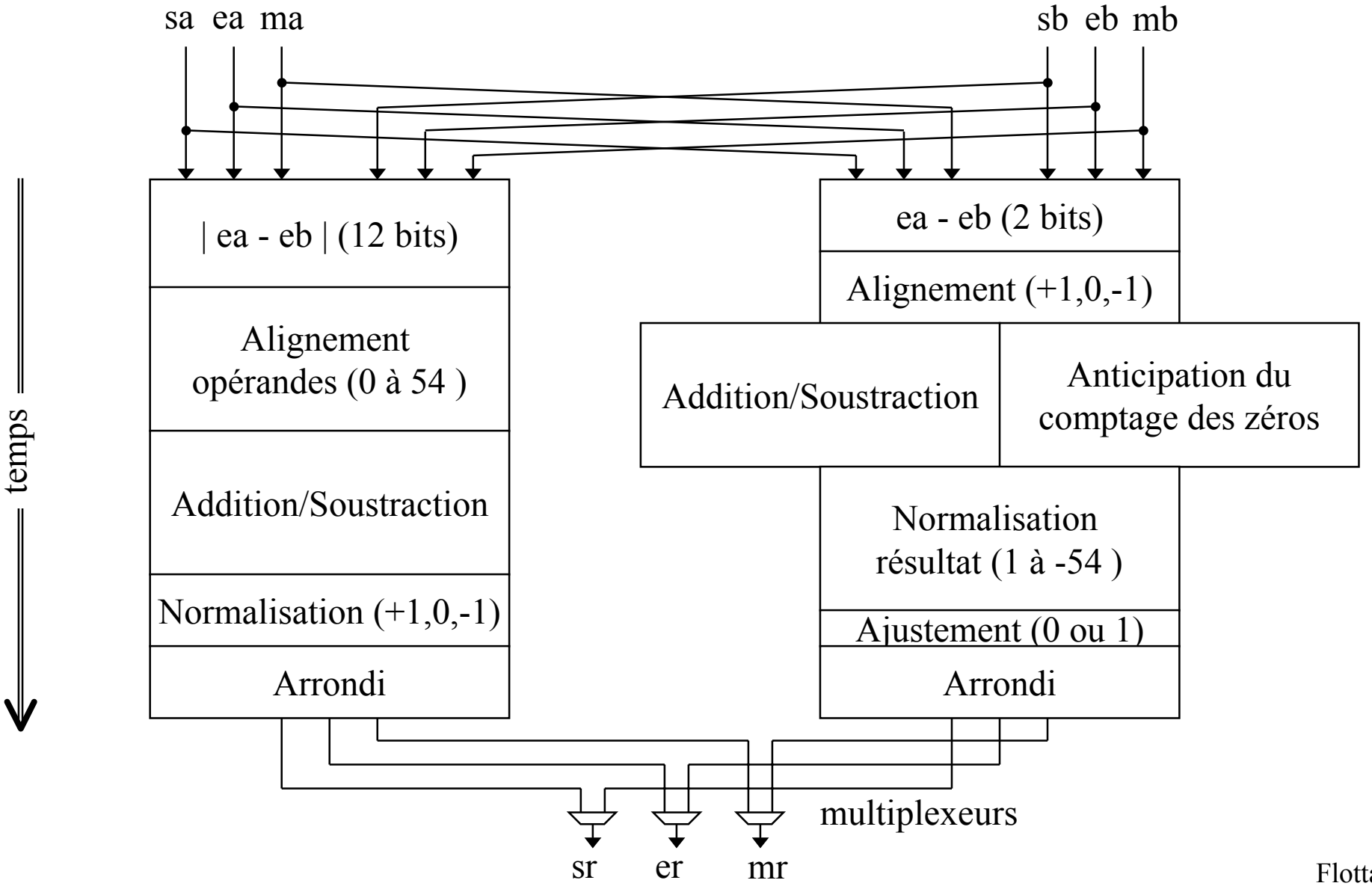
2 - décaler la mantisse du plus petit de
0 ou 1 position à droite

Normalisation

3 - pour une addition, le résultat est décalé
de 0 ou 1 position à droite

3' - pour une soustraction,
3'-1 calculer n = nombre de 0 en
poids forts
3'-2 décaler la mantisse du résultat de n
positions à gauche (introduire n zéros)

Addition/Soustraction spéculative (2)



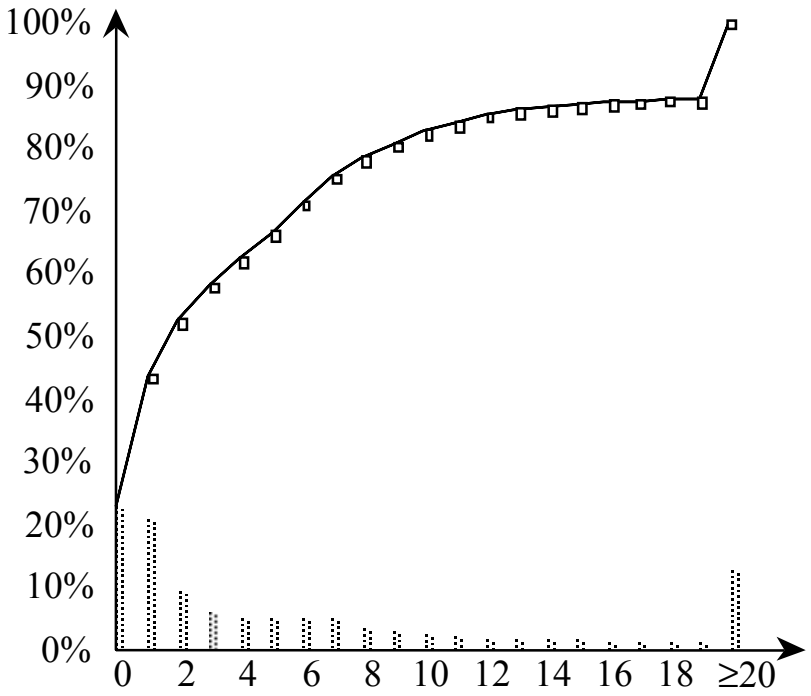
Calcul spéculatif de (ea - eb)

On spécule que | ea - eb | ≤ 1
On veut calculer (ea - eb) ∈ {-1, 0, 1}
On ne tient compte que des 2 derniers
bits de ea et de eb pour ce calcul
⇒ Le calcul spéculatif de (ea - eb) est rapide

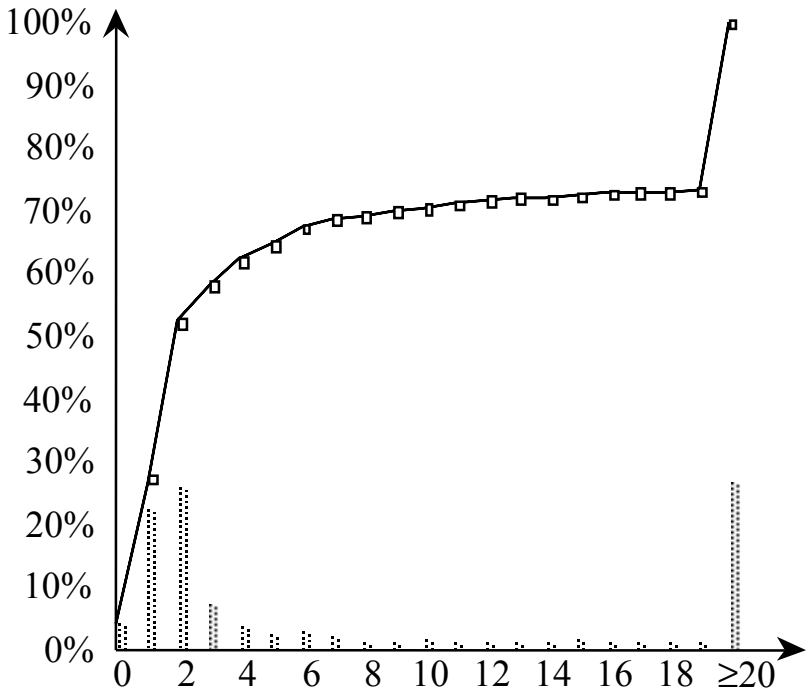
Remarque: dans les cases « ∉ {-1,0,1} », on peut mettre
ϕ puisque cette spéculation sera abandonnés.

ea	eb	ea - eb
00	00	0
00	01	-1
00	10	∉ {-1,0,1}
00	11	1
01	00	1
01	01	0
01	10	-1
01	11	∉ {-1,0,1}
10	00	∉ {-1,0,1}
10	01	1
10	10	0
10	11	-1
11	00	-1
11	01	∉ {-1,0,1}
11	10	1
11	11	0

Répartition des opérandes



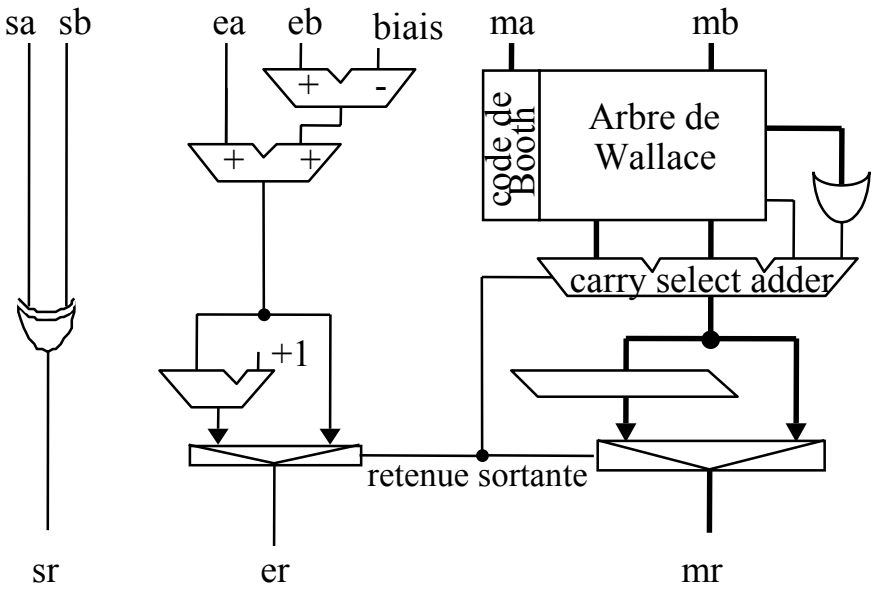
différence des exposants |ea-eb|
pour alignement



distance de décalage à gauche
pour renormalisation

Statistiquement, les deux branches de l'addition spéculative sont prises (≈ 44%, ≈ 56%)

Multiplication flottante



Multiplication des mantisses

Les bits poids faible servent pour l'arrondi

Ajout des exposants

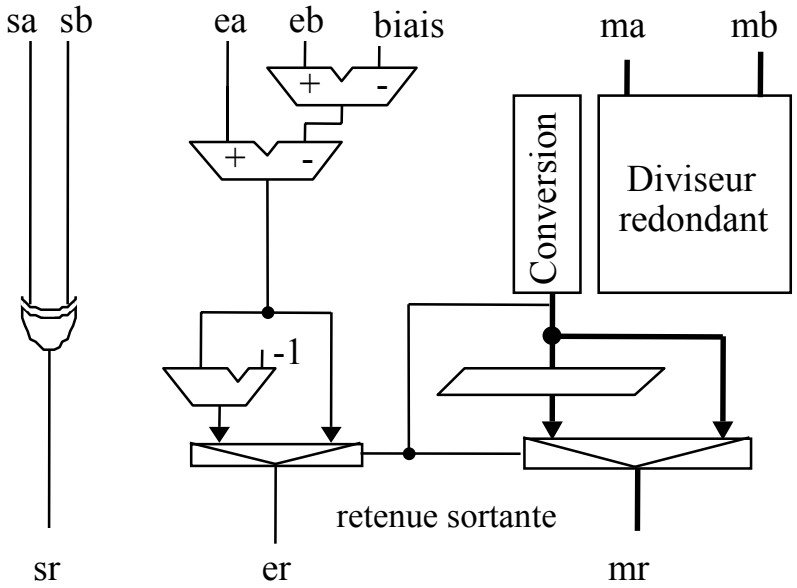
Si la retenue sortante vaut 1 alors décaler le résultat d'une position à droite

L'arbre de Wallace étant difficile à implémenter, on utilise des variantes
On utilise aussi les arbres binaires (dits CS)

Seuls les sorties poids fort du multiplieur, en "carry save" sont additionnés.
On fait le Ou des autres sans les additionner pour l'arrondi.

Le matériel nécessaire au calcul d'un résultat dénormalisé n'est pas inclus dans ce schéma.

Division flottante



Division des mantisses

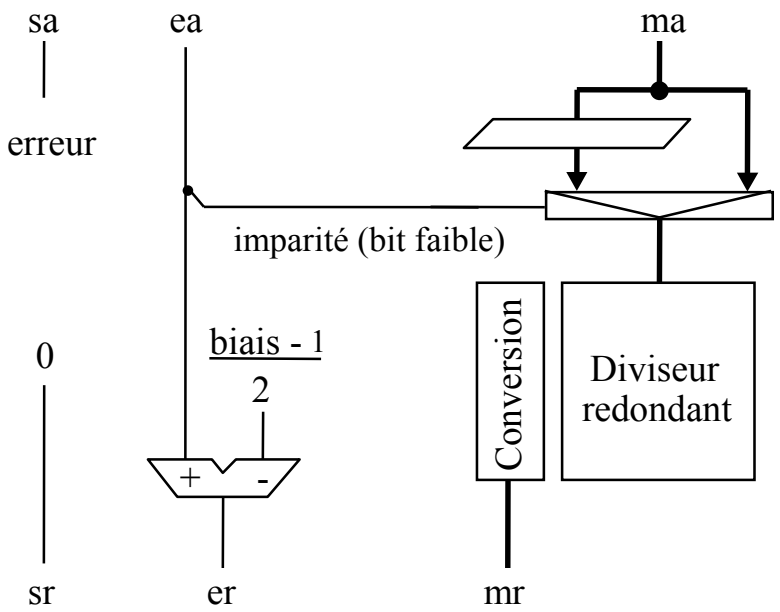
Différence des exposants

Si le quotient est < 1 alors le décaler de une position à gauche

La norme précise que le reste a le même signe que le dividende.

Le matériel nécessaire à la production d'un résultat dénormalisé

Racine Carrée flottante



Le biais est toujours impair

L'exposant d'un carré est pair

Avec des modifications mineures, un diviseur peut calculer la racine carrée

Les racines sont ≥ 0 sauf $\sqrt{-0} = -0$ (Standard IEEE)

Le matériel nécessaire à la production d'un résultat dénormalisé n'est pas inclus dans ce schéma.

Fonctions élémentaires flottante

Les fonctions élémentaires sont transcendantes

Il n'est donc pas possible de prédire le nombre de bits à calculer pour un arrondi au plus près pour une mantisse de taille fixée

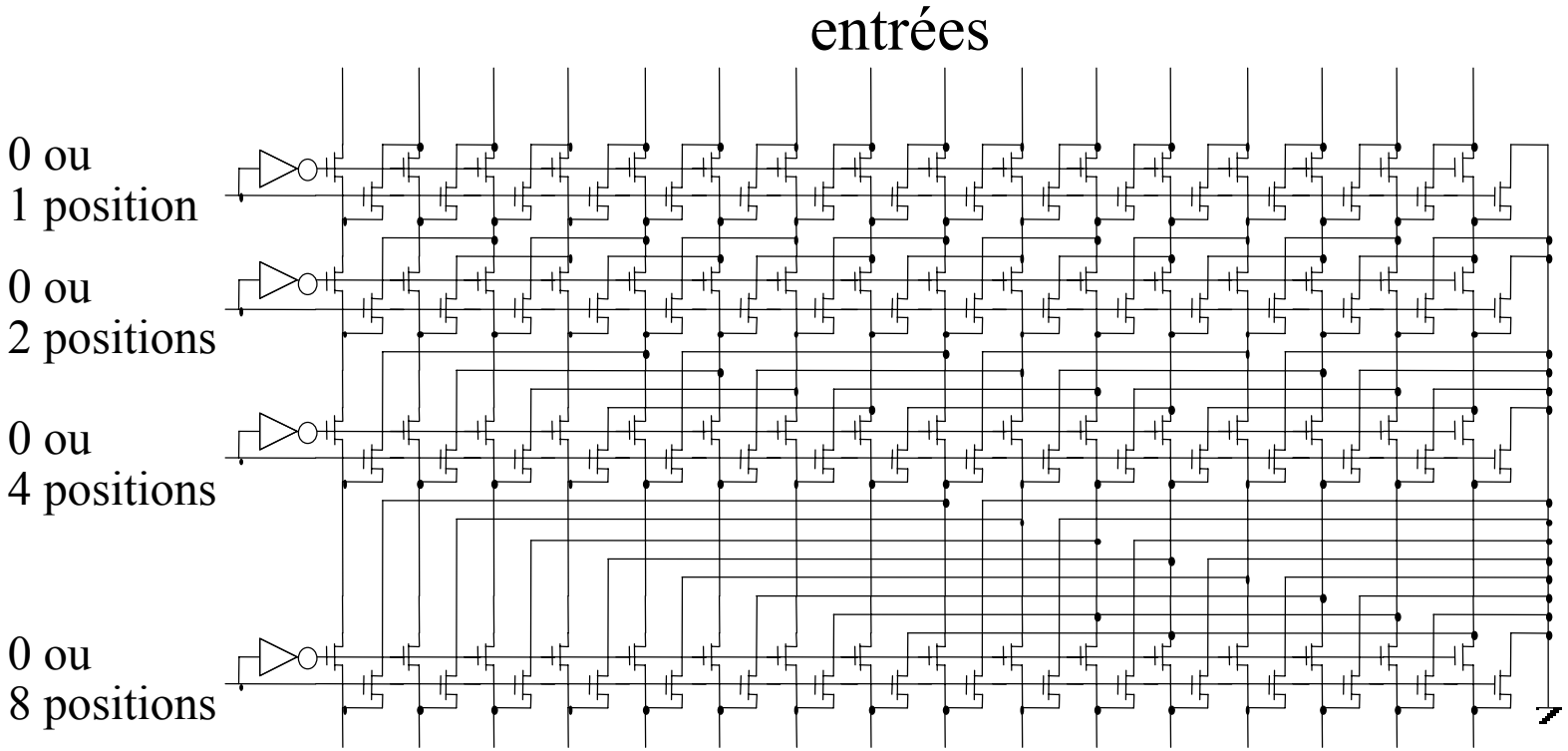
En conséquence la norme IEEE 754 ne prévoit rien pour le calcul des fonctions élémentaires

Conclusion

Les principales techniques d'accélération du calcul en virgule flottantes sont:

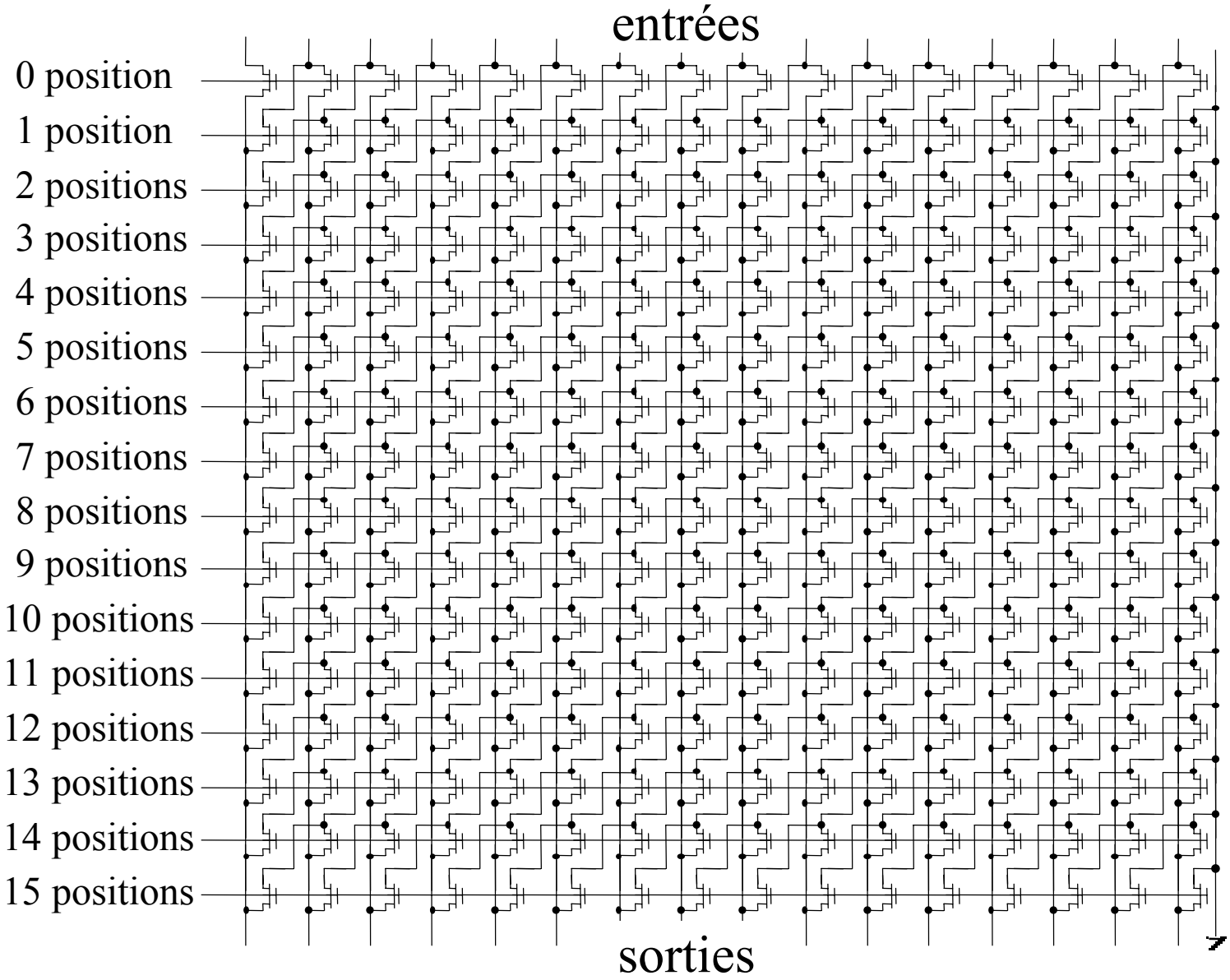
- Parallélisme (vu)
- Anticipation (vu)
- Exécution spéculative (spécule sur la différence des exposants) (vu)
- Pipe-line (non vu)
- Réinjection en redondant (retenue non propagée) (non vu)
- Autosynchrone (sans horloge) (non vu)

Décaleur logarithmique

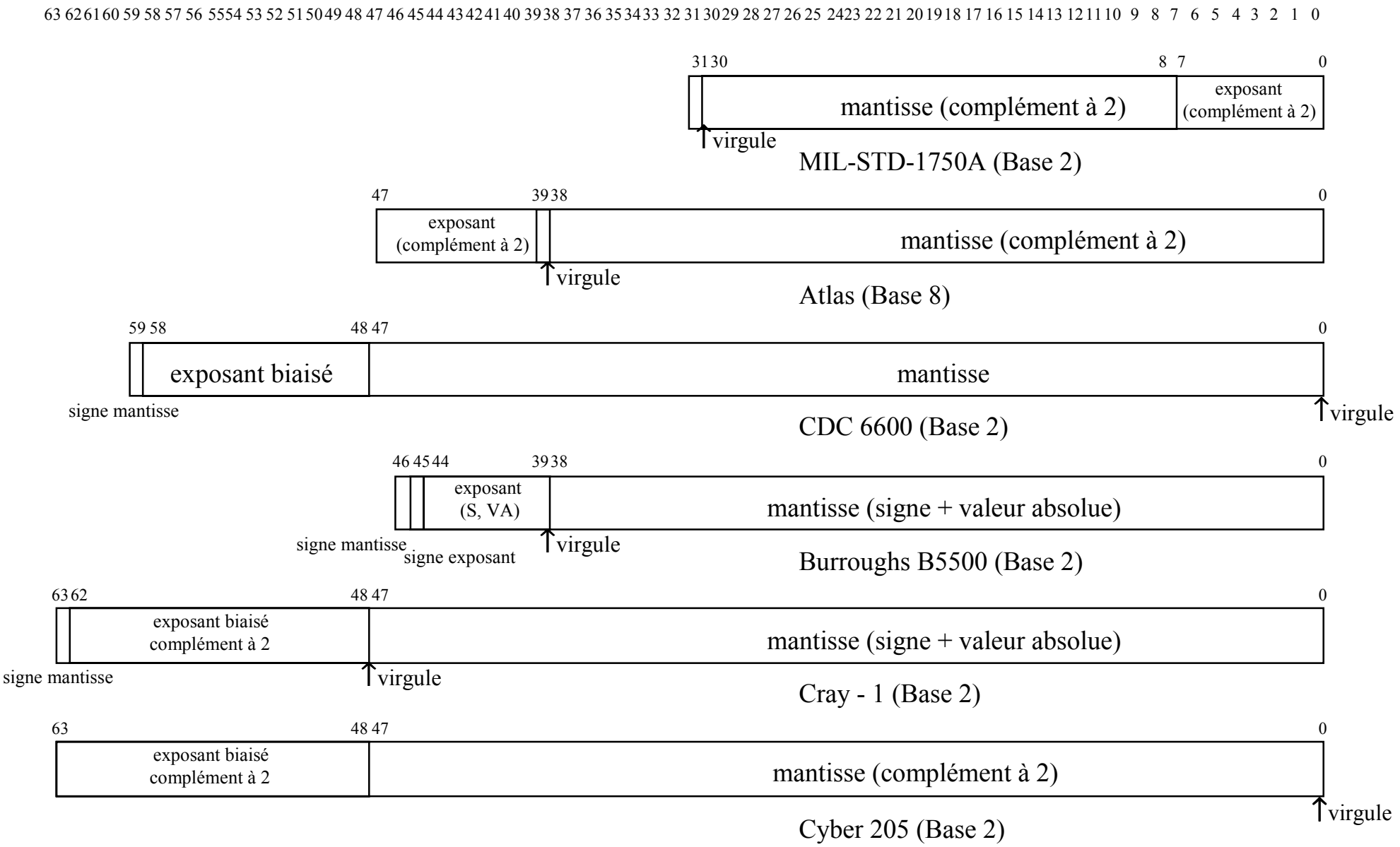


$s_i :=$ **si** $i+k \geq n$ **alors** e_{i+k} **sinon** 0

Décaleur en barillet



Autres formats virgule flottante



Fonctions élémentaires par matériel

Alain GUYOT

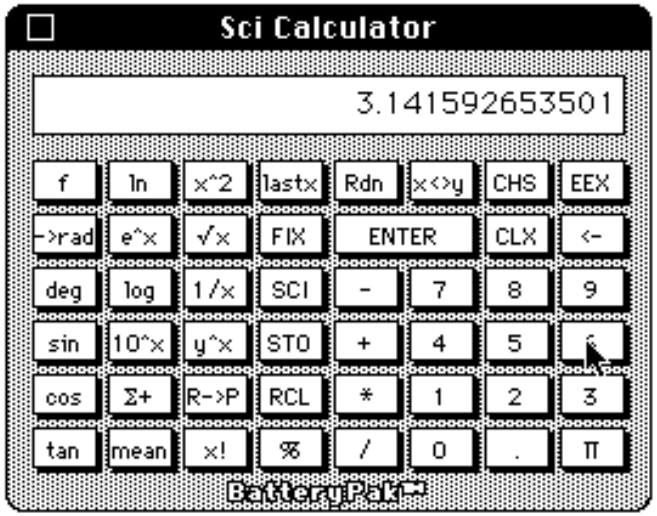
Concurrent Integrated Systems
TIMA

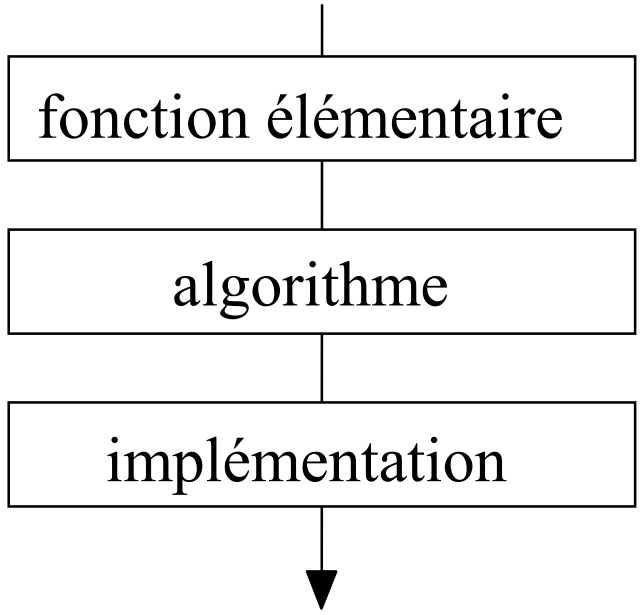
(33) 04 76 57 46 16
Alain. Guyot @imag .fr
http ://tima -cmp .imag .fr/~guyot
Techniques de l'Informatique et de la Microélectronique pour l'Architecture. Unité associée au C.N.R.S. n° B0706

But
Réaliser des fonctions
(log, exp, sin, cos, arctg..

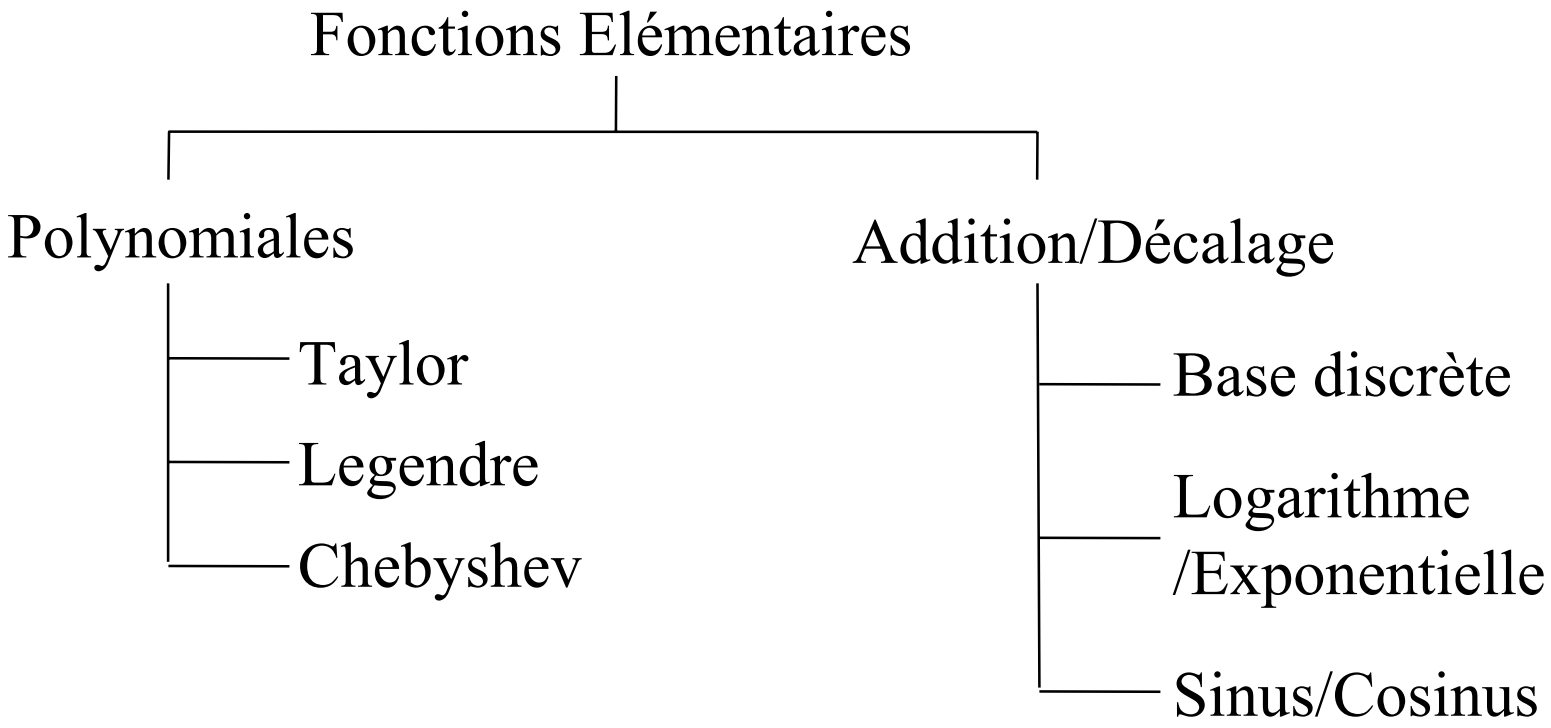
Optimiser la surface et la
vitesse par rapport au calcul
par développement limité

Moyens:
- Conversion de notation additive à multiplicative
- Conversion de notation multiplicative à additive
- Changement de base de numération





Généralités / Plan



Calcul du logarithme et de l'exponentielle

$$X = \prod_{i=0}^{n-1} (1 + 2^{-i})^{p_i} \iff \log(X) = \sum_{i=0}^{n-1} p_i \log(1 + 2^{-i})$$

$$\sum_{i=0}^{m-1} x_i \, 2^{-i} \qquad\qquad \sum_{i=0}^{m-1} y_i \, 2^{-i}$$

$$1 \downarrow \uparrow 2 \qquad\qquad\qquad 3 \downarrow \uparrow 4$$

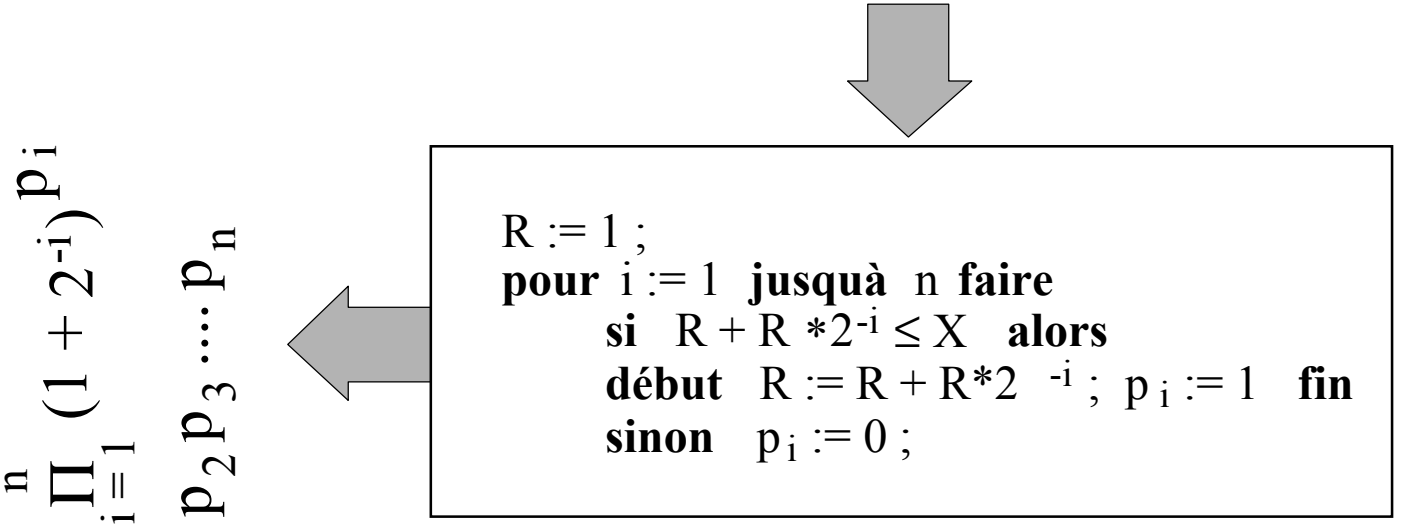
$$\prod_{i=0}^{n-1} (1 + 2^{-i})^{p_i} \quad \underset{\text{logarithme}}{\overset{\text{exponentielle}}{\leftrightarrows_{5}}} \quad \sum_{i=0}^{n-1} p_i \log(1 + 2^{-i})$$

1 - passage de notation additive à multiplicative	(pseudo division)
2 - passage de notation multiplicative à additive	(pseudo multiplication)
3 - passage de la base 2^{-i} à la base $\log(1 + 2^{-i})$	(pseudo division)
4 - passage de la base $\log(1 + 2^{-i})$ à la base 2^{-i}	(pseudo multiplication)
5- change la valeur mais pas la représentation	

Conversion de notation additive à multiplicative

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

$1 \quad x_1 \, x_2 \, x_3 \, ... \, x_n$



La conversion ne change pas la valeur de X mais seulement sa représentation sous forme de chaîne de bits

Valeurs numérique

n = 8 bits

$$\begin{aligned}
\log(2) &= 0{,}69314718 \\
\log(1{,}5) &= 0{,}40546511 \\
\log(1{,}25) &= 0{,}22314355 \\
\log(1{,}125) &= 0{,}11778304 \\
\log(1{,}0625) &= 0{,}06062462 \\
\log(1{,}03125) &= 0{,}03077166 \\
\log(1{,}015625) &= 0{,}01550419 \\
\log(1{,}0078125) &= 0{,}00778214 \\
\log(1+\varepsilon) &\approx \varepsilon
\end{aligned}$$

$$\begin{aligned}
X = {} & (1 + p_0 * 1) \\
& * (1 + p_1 * 0{,}5) \\
& * (1 + p_2 * 0{,}25) \\
& * (1 + p_3 * 0{,}125) \\
& * (1 + p_4 * 0{,}0625) \\
& * (1 + p_5 * 0{,}03125) \\
& * (1 + p_6 * 0{,}015625) \\
& * (1 + p_7 * 0{,}0078125) \\
& * (1 + R_8)
\end{aligned}
\quad \Leftrightarrow \quad
\begin{aligned}
\log(X) = {} & p_0 * 0{,}69314718 \\
& + p_1 * 0{,}40546511 \\
& + p_2 * 0{,}22314355 \\
& + p_3 * 0{,}11778304 \\
& + p_4 * 0{,}06062462 \\
& + p_5 * 0{,}03077166 \\
& + p_6 * 0{,}01550419 \\
& + p_7 * 0{,}00778214 \\
& + R_8
\end{aligned}$$

Exemple de calcul

$$X = (1 + p_0 * 1) * (1 + p_1 * 0{,}5) * (1 + p_2 * 0{,}25) * (1 + p_3 * 0{,}125) * (1 + p_4 * 0{,}0625) * (1 + p_5 * 0{,}03125) * (1 + p_6 * 0{,}015625) * (1 + p_7 * 0{,}0078125) * (1 + R_8)$$

$$\Leftrightarrow$$

$$Y = \log(X) = p_0 * 0{,}69314718 + p_1 * 0{,}40546511 + p_2 * 0{,}22314355 + p_3 * 0{,}11778304 + p_4 * 0{,}06062462 + p_5 * 0{,}03077166 + p_6 * 0{,}01550419 + p_7 * 0{,}00778214 + R_8$$

Exemple: calcul de X = exp(1,236)

$$1{,}236 = 1 * 0{,}69314718 + 1 * 0{,}40546511 + 0 * 0{,}22314355 + 1 * 0{,}11778304 + 0 * 0{,}06062462 + 0 * 0{,}03077166 + 1 * 0{,}01550419 + 0 * 0{,}00778214 + 0{,}00410048$$

$R_0 = 1{,}23600000$
$R_1 = 0{,}54285282$
$R_2 = 0{,}13738771$
$R_3 = 0{,}13738771$
$R_4 = 0{,}01960467$
$R_5 = 0{,}01960467$
$R_6 = 0{,}01960467$
$R_7 = 0{,}00410048$
$R_8 = 0{,}00410048$

$$\exp(1{,}236) = (1 + 1 * 1) * (1 + 1 * 0{,}5) * (1 + 0 * 0{,}25) * (1 + 1 * 0{,}125) * (1 + 0 * 0{,}0625) * (1 + 0 * 0{,}03125) * (1 + 1 * 0{,}015625) * (1 + 0 * 0{,}0078125) * (1 + 0{,}00410048)$$

$X_0 = 1{,}00000000$
$X_1 = 2{,}00000000$
$X_2 = 3{,}00000000$
$X_3 = 3{,}00000000$
$X_4 = 3{,}37500000$
$X_5 = 3{,}37500000$
$X_6 = 3{,}37500000$
$X_7 = 3{,}42773437$
$X_8 = 3{,}42773437$
$3{,}44178808$

Exemple de calcul

$$X = (1 + p_0 * 1) * (1 + p_1 * 0{,}5) * (1 + p_2 * 0{,}25) * (1 + p_3 * 0{,}125) * (1 + p_4 * 0{,}0625) * (1 + p_5 * 0{,}03125) * (1 + p_6 * 0{,}015625) * (1 + p_7 * 0{,}0078125) * (1 + R_8)$$

$$\Leftrightarrow$$

$$Y = \log(X) = p_0 * 0{,}69314718 + p_1 * 0{,}40546511 + p_2 * 0{,}22314355 + p_3 * 0{,}11778304 + p_4 * 0{,}06062462 + p_5 * 0{,}03077166 + p_6 * 0{,}01550419 + p_7 * 0{,}00778214 + R_8$$

Exemple: calcul de X = exp(Y)

$$Y = \square * 0{,}69314718 + \square * 0{,}40546511 + \square * 0{,}22314355 + \square * 0{,}11778304 + \square * 0{,}06062462 + \square * 0{,}03077166 + \square * 0{,}01550419 + \square * 0{,}00778214 + 0{,}00410048$$

$R_0 = Y$

$R_1 =$ $R_2 =$ $R_3 =$ $R_4 =$ $R_5 =$ $R_6 =$ $R_7 =$ $R_8 =$

$$\exp(Y) = (1 + \square * 1) * (1 + \square * 0{,}5) * (1 + \square * 0{,}25) * (1 + \square * 0{,}125) * (1 + \square * 0{,}0625) * (1 + \square * 0{,}03125) * (1 + \square * 0{,}015625) * (1 + \square * 0{,}0078125) * (1 + 0{,}00410048)$$

$X_0 = 1$

$X_1 =$ $X_2 =$ $X_3 =$ $X_4 =$ $X_5 =$ $X_6 =$ $X_7 =$ $X_8 =$

Calcul du sinus et du cosinus

$$\alpha = \sum_{i=0}^{n-1} a_i \, 2^{-i} \qquad a_i \in \{0,+1\}$$

1

$$\alpha = \sum_{i=0}^{n-1} \alpha_i \, \mathrm{arctg}\left(2^{-i}\right) \qquad \alpha_i \in \{-1,+1\}$$

2

$$\begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix} = \prod_{i=0}^{n-1} \begin{pmatrix} \cos\left(\alpha_i \, \mathrm{arctg}\left(2^{-i}\right)\right) & -\sin\left(\alpha_i \, \mathrm{arctg}\left(2^{-i}\right)\right) \\ \sin\left(\alpha_i \, \mathrm{arctg}\left(2^{-i}\right)\right) & \cos\left(\alpha_i \, \mathrm{arctg}\left(2^{-i}\right)\right) \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

1 - changement de base de l'angle α

2 - rotation de l'angle α

Réécriture de la rotation de l'angle α

$$\begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix} = \prod_{i=0}^{n} \begin{pmatrix} \cos(\alpha_i \operatorname{arctg}(2^{-i})) & -\sin(\alpha_i \operatorname{arctg}(2^{-i})) \\ \sin(\alpha_i \operatorname{arctg}(2^{-i})) & \cos(\alpha_i \operatorname{arctg}(2^{-i})) \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\prod_{i=0}^{n} \cos(\alpha_i \operatorname{arctg}(2^{-i})) \begin{pmatrix} 1 & -\operatorname{tang}(\alpha_i \operatorname{arctg}(2^{-i})) \\ \operatorname{tang}(\alpha_i \operatorname{arctg}(2^{-i})) & 1 \end{pmatrix}$$

$$\prod_{i=0}^{n} \cos(\operatorname{arctg}(2^{-i})) \begin{pmatrix} 1 & -\alpha_i \operatorname{tang}(\operatorname{arctg}(2^{-i})) \\ \alpha_i \operatorname{tang}(\operatorname{arctg}(2^{-i})) & 1 \end{pmatrix} \quad \alpha_i \in \{-1,+1\}$$

$$\begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix} = \prod_{i=0}^{n} \sqrt{1+2^{-2i}} \prod_{i=0}^{n} \begin{pmatrix} 1 & -\alpha_i 2^{-i} \\ \alpha_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Calcul du sinus et du cosinus (1)

$$\alpha = \sum_{i=0}^{n-1} a_i \, 2^{-i} \qquad a_i \in \{0,+1\}$$

1

Cette conversion change la représentation de α mais pas la valeur de α
(à l'erreur de conversion près)

$$\alpha = \sum_{i=0}^{n-1} \alpha_i \, \mathrm{arctg}\left(2^{-i}\right) \qquad \alpha_i \in \{-1,+1\}$$

2

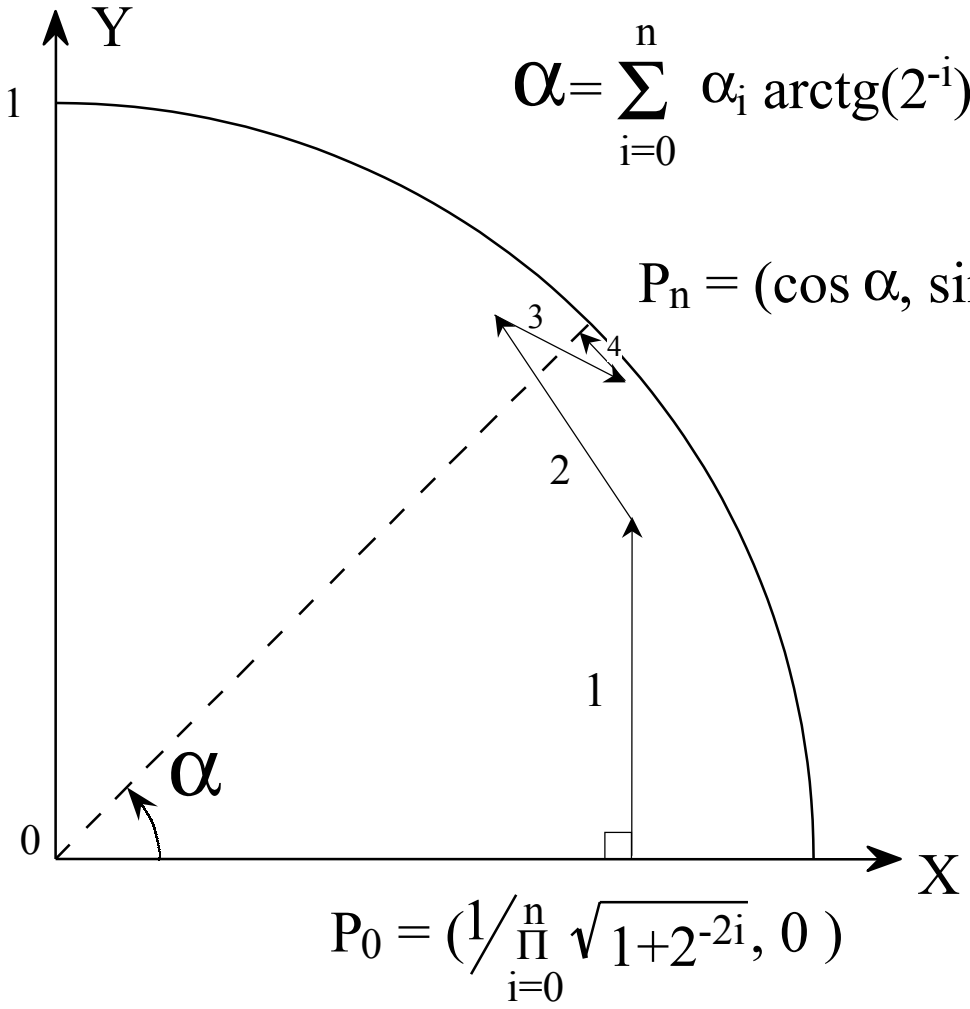
valeur initiale

$$\begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix} = \prod_{i=0}^{n} \begin{pmatrix} 1 & -\alpha_i \, 2^{-i} \\ \alpha_i \, 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \Big/ \prod_{i=0}^{n} \sqrt{1+2^{-2i}} \end{pmatrix}$$

1 - changement de base de l'angle α (pseudo division)

2 - conversion de multiplicative à additive (pseudo multiplications)

Calcul du sinus et du cosinus (2)



$$\alpha = \sum_{i=0}^{n} \alpha_i \operatorname{arctg}(2^{-i}) \qquad \alpha_i \in \{-1,1\}$$

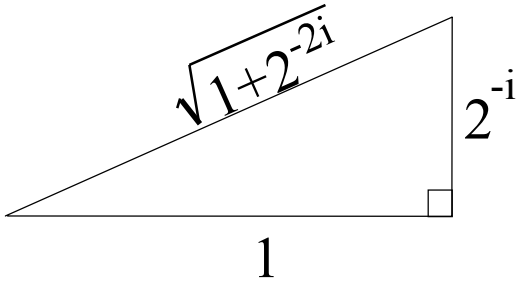
$$P_n = (\cos \alpha, \sin \alpha)$$

$$P_0 = \left(\frac{1}{\prod_{i=0}^{n} \sqrt{1+2^{-2i}}}, 0\right)$$

(environ 0,607 252 935…)

vecteurs perpendiculaires

$$x_{i+1} := x_i - \alpha_i\, y_i\, 2^{-i}$$
$$y_{i+1} := y_i + \alpha_i\, x_i\, 2^{-i}$$



Valeurs numériques

i	2^{-i}	$\arctan(2^{-i})$
0	1.00000 00000	0.110 001 000 000 000
1	0.10000 00000	0.011 010 100 100 001
2	0.01000 00000	0.001 110 000 010 010
3	0.00100 00000	0.000 111 001 000 000
4	0.00010 00000	0.000 011 100 100 000
5	0.00001 00000	0.000 001 110 101 100
6	0.00000 10000	0.000 000 111 001 010
7	0.00000 01000	0.000 000 011 100 101
8	0.00000 00100	0.000 000 001 110 010
9	0.00000 00010	0.000 000 000 111 001
10	0.00000 00001	0.000 000 000 011 100

La précision est significativement augmenté en normalisant les constantes $\arctan(2^{-i}) * 2^{i}$. Pour n bits de précision il faut calculer avec $n + \log_2(n) + 2$ bits

Bases discrètes

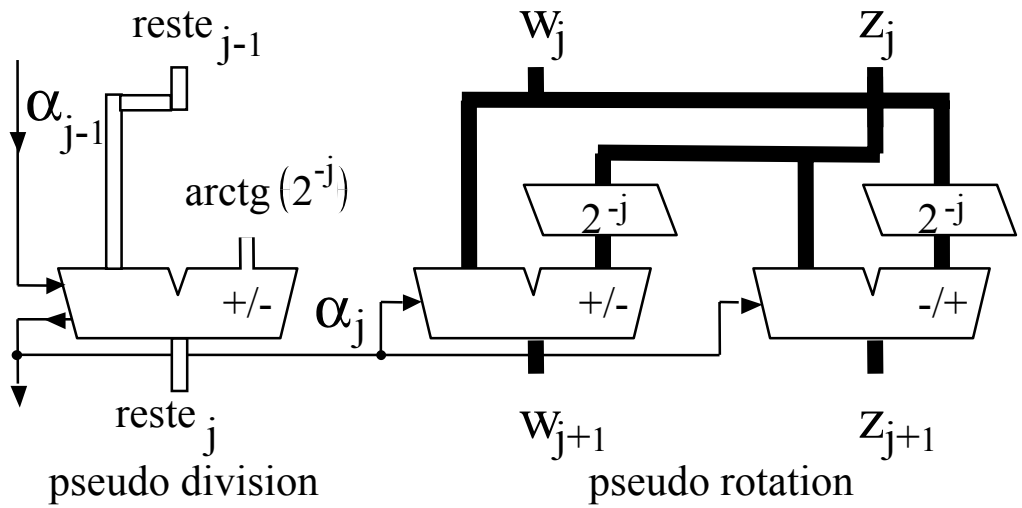
Quels nombres peut-on écrire en base:

$\log(1+2^i)$

$\mathrm{ArcTg}(2^i)$

Cette question détermine le domaine des algorithmes de calcul de logarithme et fonctions de trigonométrie.

Tranche pour le calcul du sinus et du cosinus



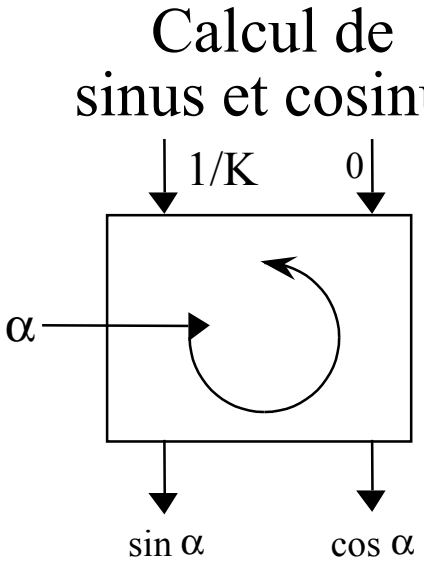
soit n = nombre de bits = nombre de pas
coût du j^{eme} décaleur: (n-j) fils $*$ j positions

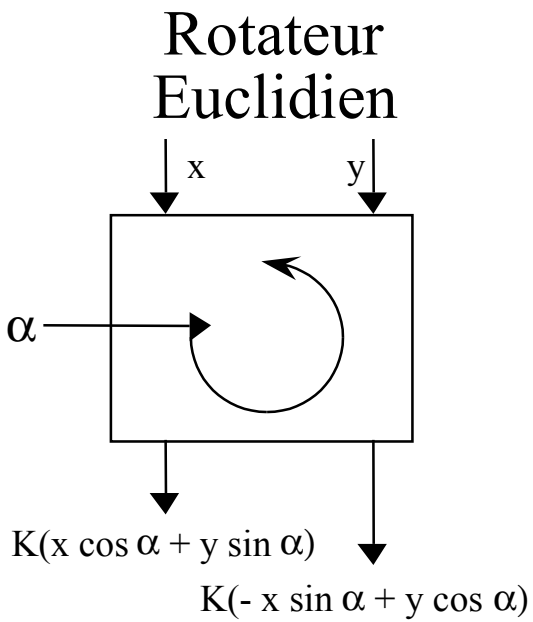
Remarque: les calculs peuvent également se faire sans propagation de retenue

Rotateur Euclidien (1)

(imagerie)

But: faire subir une rotation à un vecteur sans calcul explicite de sin et cos





Pour la rotation, il faut diviser le résultat par K

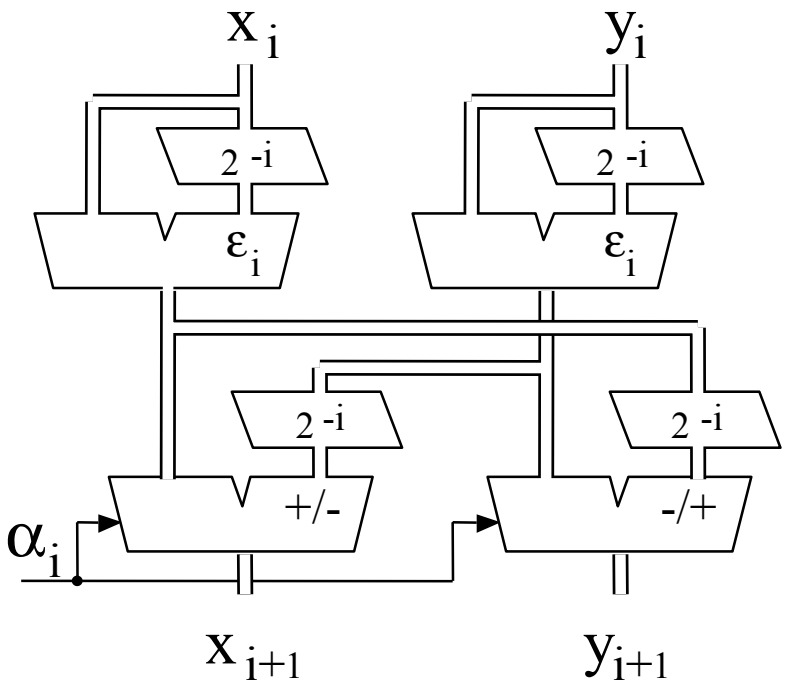
Solution 1: Ecrire $1/K = \prod_{i=0}^{n} (1 + \varepsilon_i 2^{-i})$ avec $\varepsilon_i \in \{-1,0,+1\}$ et le minimum de $\varepsilon_i \neq 0$

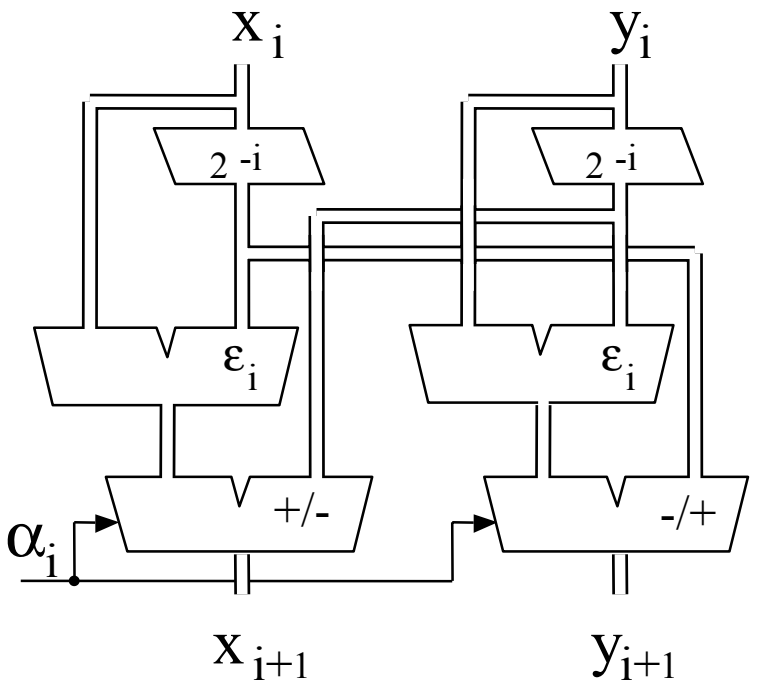
Solution 2: Décomposer α dans la base Arctg $(2^{-si}+2^{-s'i})$ tq. $K = \prod_{i=0}^{n} \sqrt{1 + (2^{-si}+2^{-s'i})^2}$
soit proche d'une puissance de 2

Rotateur Euclidien (2)

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = (1 + \varepsilon_i\, 2^{-i}) \begin{pmatrix} 1 & \alpha_i\, 2^{-i} \\ -\alpha_i\, 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

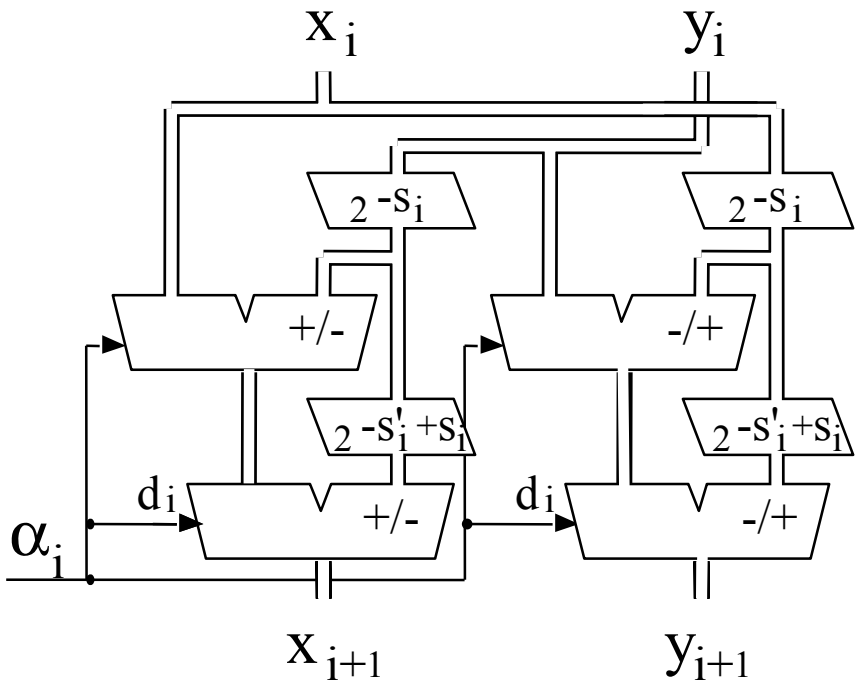
$$x_{i+1} = x_i + (\varepsilon_i\, x_i + \alpha_i\, y_i)\, 2^{-i} + \varepsilon_i\, \alpha_i\, y_i\, 2^{-2i}$$
$$y_{i+1} = y_i + (\varepsilon_i\, y_i - \alpha_i\, x_i)\, 2^{-i} + \varepsilon_i\, \alpha_i\, x_i\, 2^{-2i}$$





Rotateur Euclidien (3)

$$\alpha = \sum_{i=0}^{18} \text{arctg}\,(2^{-s_i} + d_i 2^{-s'_i})$$



$$K = \prod_{i=0}^{18} \sqrt{1 + (2^{-s_i} + d_i 2^{-s'_i})^2} = 0{,}5000096618$$

i	s_i	s'_i	d_i	arctg
0	0	3	+1	0,844154
1	1	8	+1	0,466768
2	1	6	+1	0,476069
3	2	14	+1	0,245036
4	2	4	-	0,185348
5	4	6	1+	0,077967
6	4	10	1-	0,061446
7	5		1	0,031240
8	6		0	0,015624
9	7		0	0,007812
10	8		0	0,003906
11	9		0	0,001953
12	10		0	0,000977
13	11		0	0,000488
14	12		0	0,000244
15	13		0	0,000122
16	14		0	0,000061
17	15		0	0,000031
18	16		0	0,000015
			0	

Opérateurs de Calcul en-ligne

Alain GUYOT

Concurrent Integrated Systems
TIMA

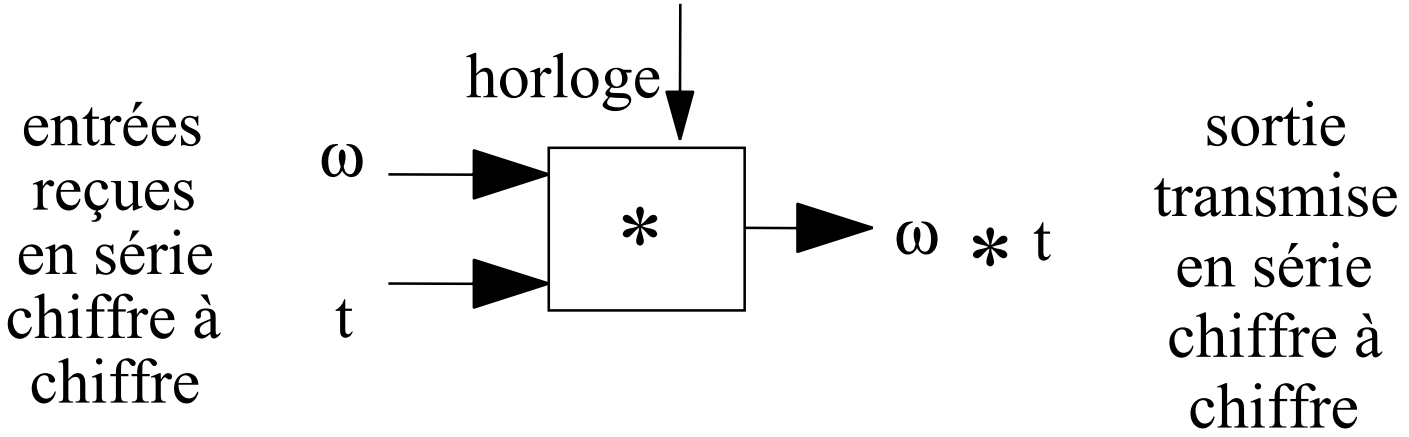
(33) 04 76 57 46 16

Alain.Guyot@imag.fr

http://tima-cmp.imag.fr/~guyot/Cours/Arithmetique

Techniques de l'Informatique et de la Microélectronique pour l'Architecture. Unité associée au C.N.R.S. n° B0706

Opérateurs en ligne



Exemples d'opérations en ligne

Addition

$$\begin{array}{r} 4\,3\,7\,2 \\ 5\,3\,9\,1 \\ +\ 9\,7\,6\,3 \\ \hline \end{array}$$

de droite à gauche

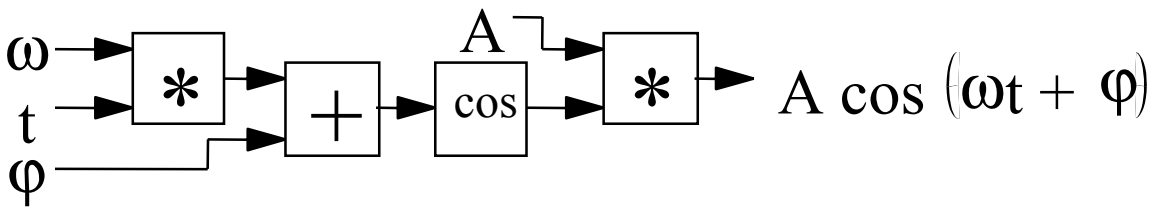
Maximum

$$\begin{array}{r} 1\,7\,8\,9 \\ 1\,7\,9\,0 \\ \geq\ 1\,7\,9\,0 \\ \hline \end{array}$$

de gauche à droite

Avantages des opérateurs en ligne

Avantage 1: parallélisme à grain fin (niveau du chiffre)

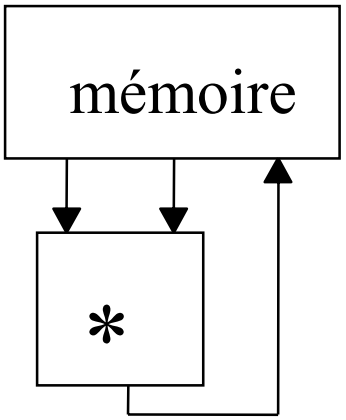


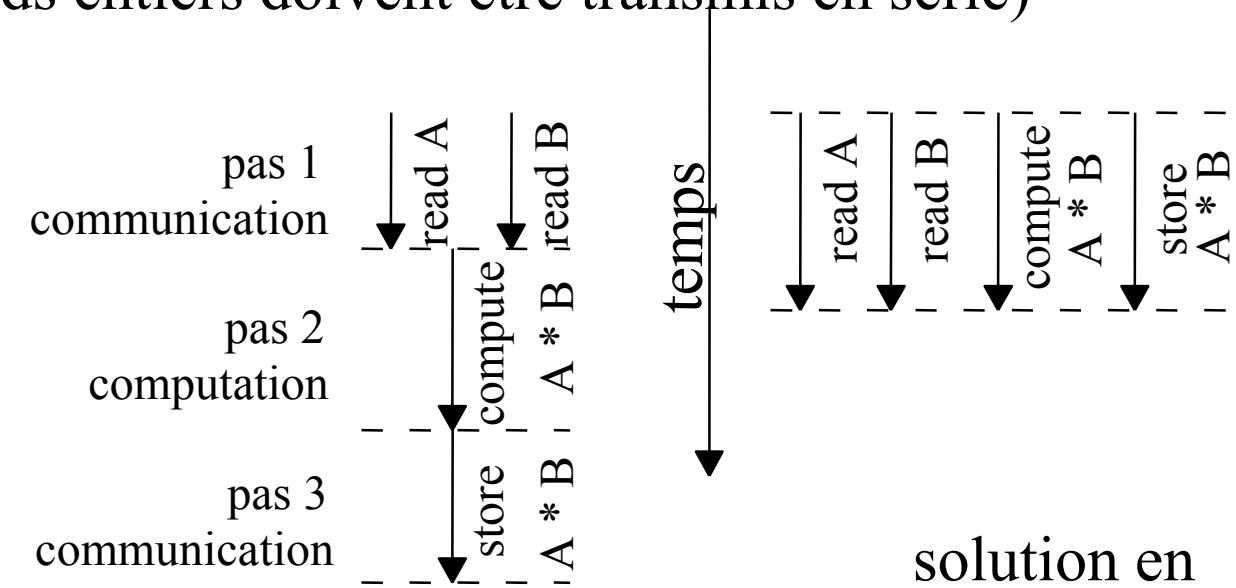
chaque opérateur travaille simultanément
(2 multiplieurs, 1 additionneur, 1 cosinus)

Avantages (2)

Avantage 2: la transmission recouvre l'exécution

exemple: multiplication de deux grands entiers A et B
(Les grands entiers doivent être transmis en série)



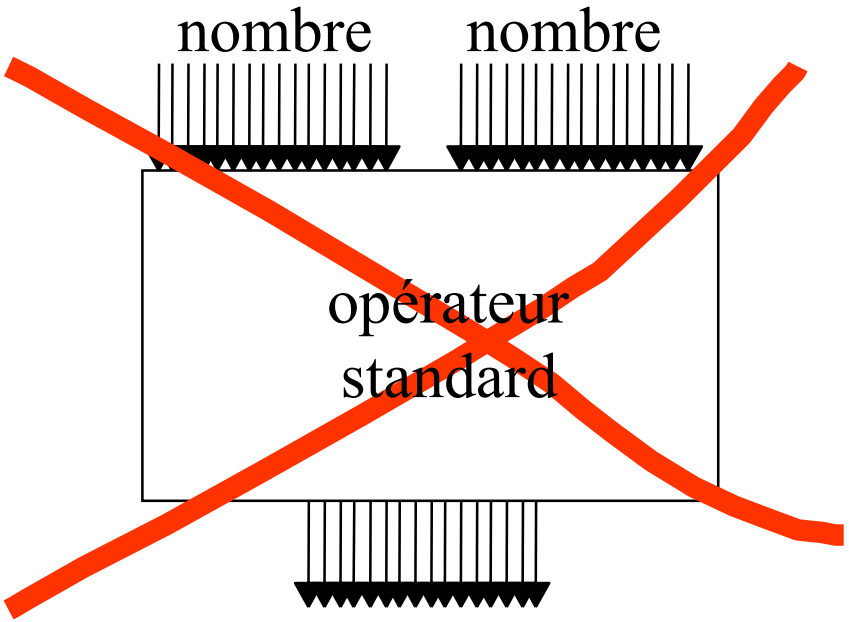


solution classique
séquentielle

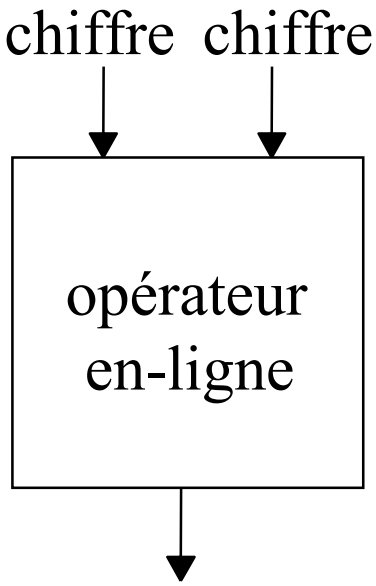
solution en
ligne
parallèle

Avantages (3)

Avantage 3: câblage



solution standard



solution en-ligne

Bonnes Nouvelles

Presque toutes les opérations habituelles peuvent être calculées en ligne

Addition	Maximum	Sinus/cosinus
Multiplication	Valeur Absolue	Tangente
Division	Saturation	Logarithme
Racine carrée	Tri	Exponentielle
Distance Euclidienne	Scaling	Polynômes

Opérations non calculables en ligne

Reste

Opérations modulaires

PGCD

Fonctions non continues

Arc sinus

Opérations en série

NOTATION En - tête	Standard Poids faibles	Standard Poids forts	Redondante Poids forts
addition	X		X (2)
multiplication	X (1)		X (2)
division			X (2)
maximum		X	X
racine carrée			X (2)

(1) Latence si on attend les poids forts

(2) Latence systématique

Comment mesurer les opérateurs en-ligne

période : inverse de la fréquence d'horloge

latence: nombre d'étages de registres entre entrée et sortie

(ou combinaison des poids des entrées moins le poids de la sortie)

La période peut être échangée contre de la latence

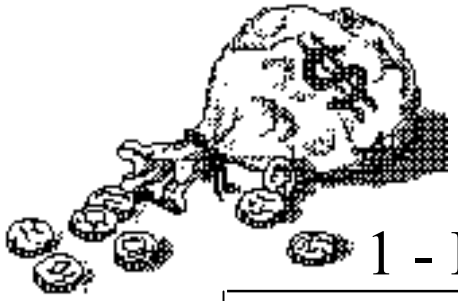
Quand des opérateurs sont mis en série:

période = max (périodes)

latence = Σ (latences) sur le chemin critique

(la latence peut devenir prédominante)

Question: Quel est le rapport coût/performance



1 - Délai

opération	construction habituelle			en-ligne
	naïve	usuelle	optimisée	
addition	n	$\log_2 n$	1*	n
multiplication	n^2	n**	$\log_2 n$**	n
division	n^2	$n \log_2 n$	n***	n
racine carrée	n^2	$n \log_2 n$	n***	n

* Additionneur à retenue sauvegardée

** Arbre de Wallace

*** Division SRT (non Newton)

Question: Quel est le rapport coût/performance (2)

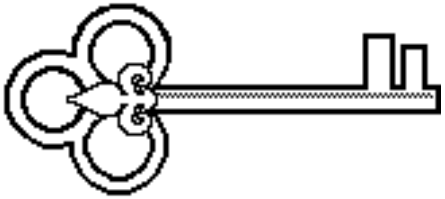
1 - Surface

opération	standard	en-ligne
addition	$n \log_2 n$	1
multiplication	n^2	n
division	n^2	n
racine carrée	n^2	n

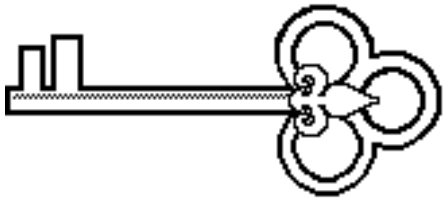
a) add,sub,max,abs,tri,gain — coût fixe — 1

b) mult,div,carré, racine — coût linéaire — $\theta(n)$

c) fonctions élémentaires — coût quadratique — $\alpha n + \beta n^2$



Clefs



Toutes les opérations en ligne reposent sur

1- l'addition sans propagation de retenus
(c'est à dire la notation redondante)

2- l'addition en série poids forts en tête dérivée de la précédente

3- en registre d'"erreur" interne

Quelque opérations en lignes reposent également sur

un estimation de faible précision de certaine variable
(reste partiel, angle, etc ...)

Notation binaire redondante BS

$$A = \sum_{i=0}^{n-1} a_i \, 2^i \qquad a_i \in \{-1, 0, 1\}$$

In the Borrow-Save (B.S.) notation, each digit is coded by 2 bits

a_i coded on 2 bits

	a_i^+	a_i^-
+1	1	0
0	0	0 or 1 1
-1	0	1

A number in redundant B.S. notation can be seen as the difference of two positive numbers

$$A = \sum_{i=0}^{n-1} a_i \, 2^i = \sum_{i=0}^{n-1} (a_i^+ - a_i^-) 2^i = \sum_{i=0}^{n-1} a_i^+ 2^i - \sum_{i=0}^{n-1} a_i^- \, 2^i = A^+ - A^-$$

Autres notations binaires redondantes

$$A = \sum_{i=0}^{n-1} a_i \, 2^i$$

Every digit is coded on 2 bits

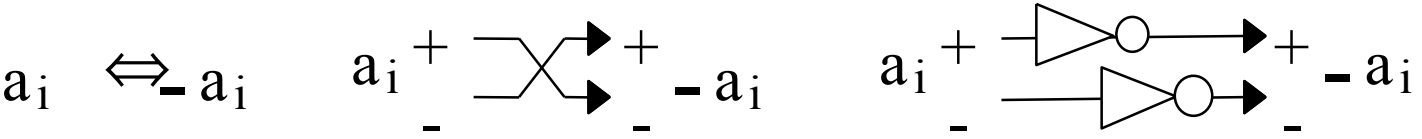
- unsigned — Carry save notation: $a_i \in \{0, 1, 2, 3\}$ coded as [1, 2]
- signed — sign+abs. value: $a_i \in \{-1, 0, 1\}$ coded as [sign, |value|]
- signed — Borrow save notation: $a_i \in \{-1, 0, 1\}$ coded as [+ , -]

Borrow Save is symmetrical

truncation = rounding

easy to change sign (no $-B = \bar{B} + 1$)

$a_i \Leftrightarrow -a_i$



Cas particulier du (Borrow Save)

$$A = \sum_{i=0}^{n-1} a_i \, 2^i \qquad a_i \in \{-1, 0, 1\}$$

1- $a_{n-1} \in \{-1, 0\}$ $a_{0..n-1} \in \{0, 1\}$ standard 2's complement

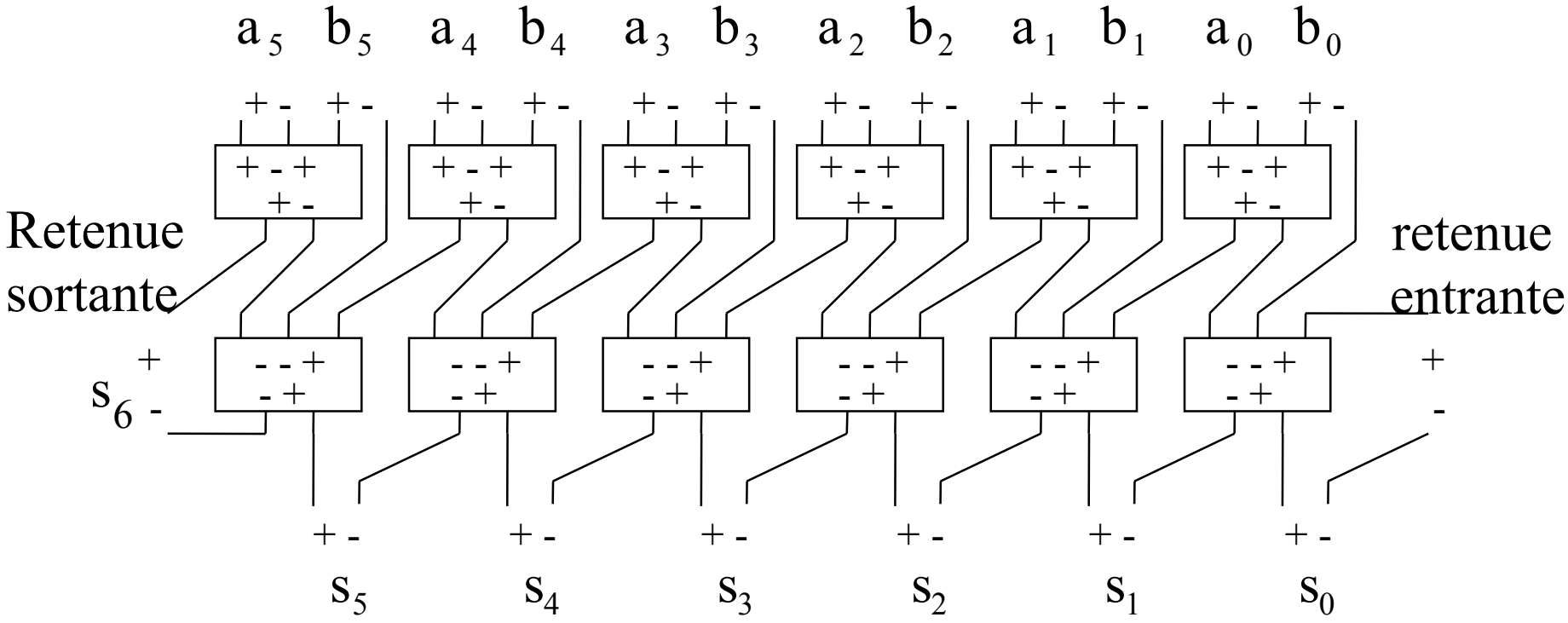
2- $a_i \in \{-1, 1\}$ on-line functions

3- $a_i \in \{-1, 0, 1\}$ with the maximum number of zero
(average 2/3 zero 1/6 one 1/6 minus one)

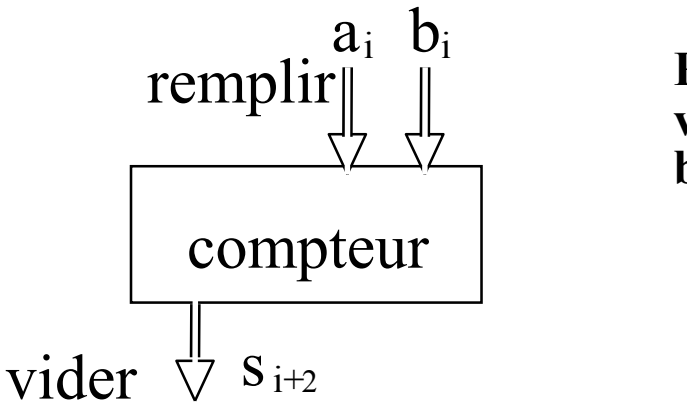
forms 1 and 3 may require an extra digit
only odd numbers representable in form 2
form 3 is the canonic signed binary digit form

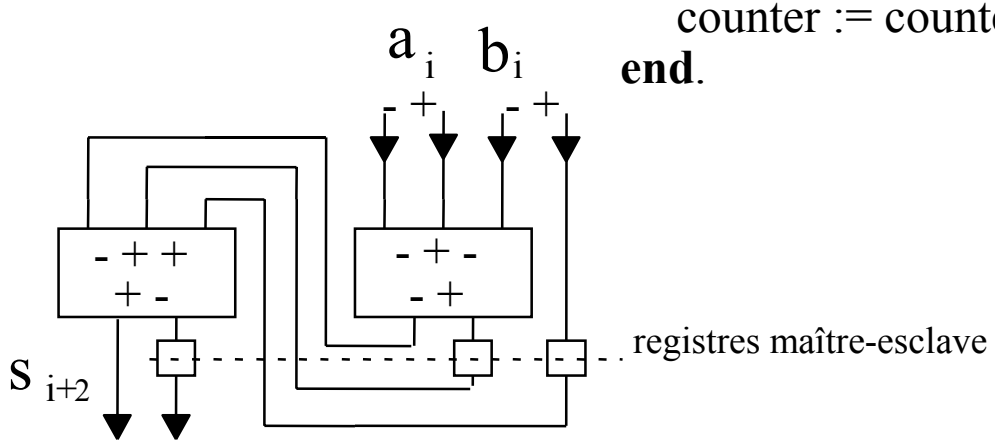
Addition parallèle sans propagation de la retenue

$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad B = \sum_{i=0}^{n-1} b_i\, 2^i \qquad S = \sum_{i=0}^{n} a_i\, 2^i \qquad a_i, b_i, s_i \in \{-1, 0, 1\}$$



Addition en-ligne (addition de chiffres en série)



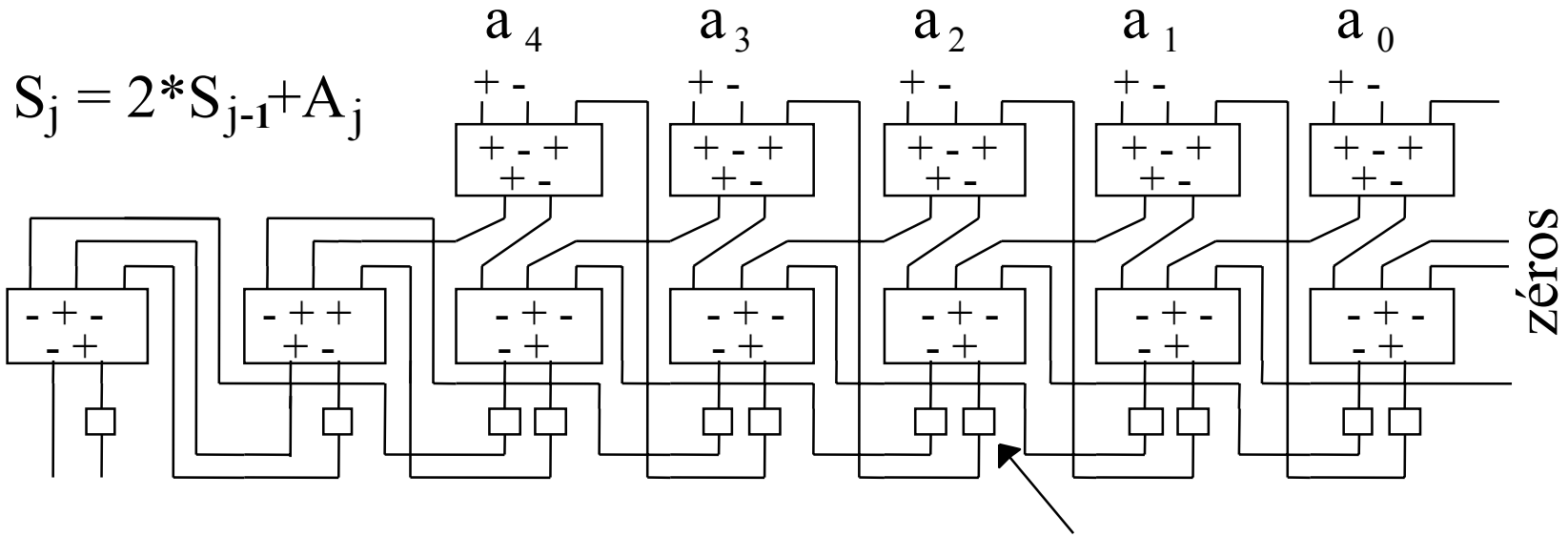



```
Procedure Addserie(ai, bi, si : SBD);
var counter: integer;
begin
    counter := 2*counter + ai + bi ;
    if counter > 2 then si := 1            {overflow}
    else if counter < -2 then si := -1   {underflow}
    else si := 0 ;
    counter := counter - 4*si  ;
end.
```

Addition d'une série de nombres avec résultat en-ligne

$$S = \sum_{j=m-1}^{0} A_j \, 2^j \qquad A_j = \sum_{i=0}^{n-1} a_{ij} \, 2^i \qquad a_{ij} \in \{-1, 0, 1\}$$

$$S_j = 2 * S_{j-1} + A_j$$

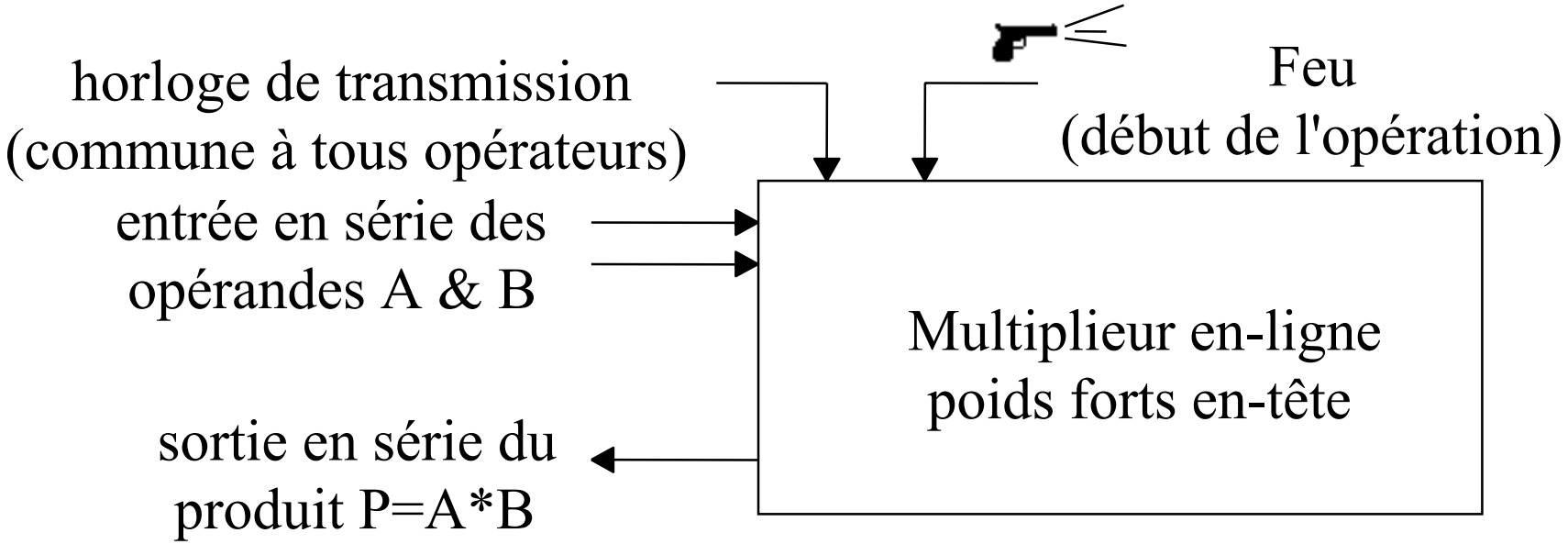


somme en-ligne
poids forte en-tête

registre d'erreur ou de récursion

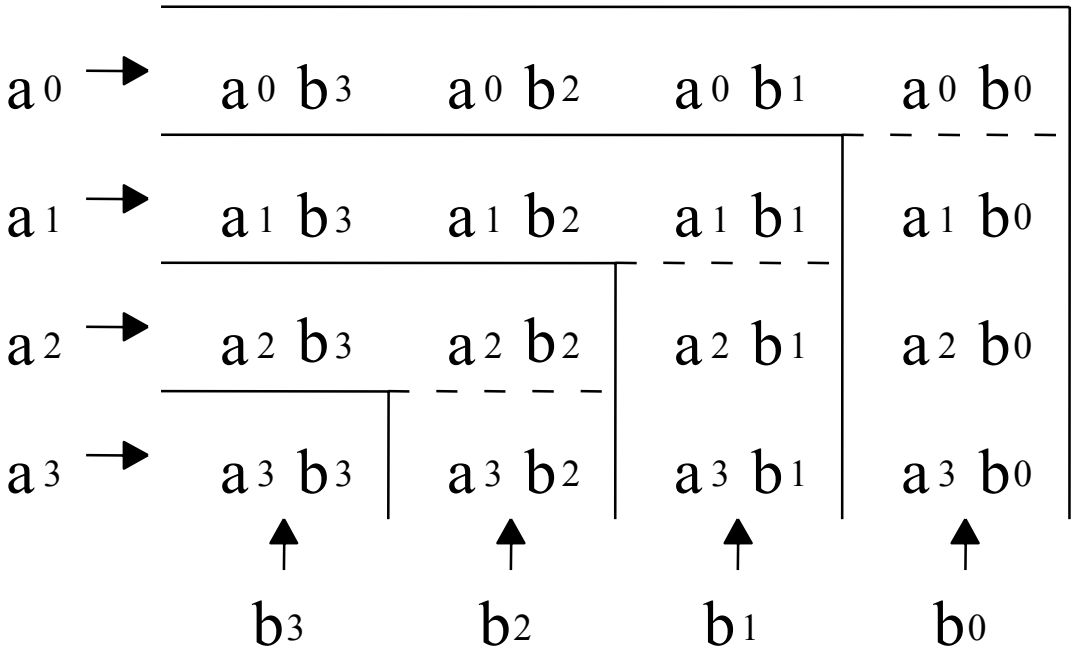
Multiplieur en ligne poids forts en-tête

$$A = \sum_{i=0}^{n-1} a_i \, 2^i \qquad B = \sum_{i=0}^{n-1} b_i \, 2^i \qquad P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i \, b_j \, 2^{i+j}$$



Multiplication en ligne poids forts en tête (2)

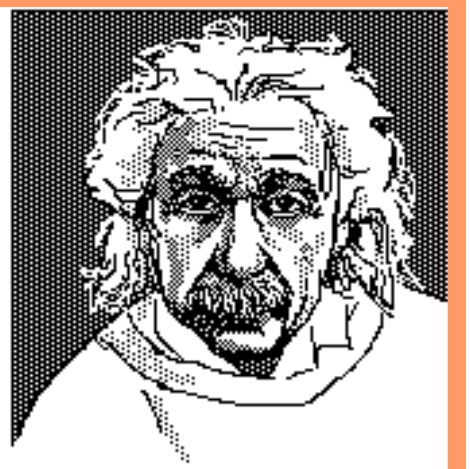
$$A = \sum_{i=0}^{n-1} a_i \, 2^i \qquad B = \sum_{i=0}^{n-1} b_i \, 2^i \qquad P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i \, b_j \, 2^{i+j}$$



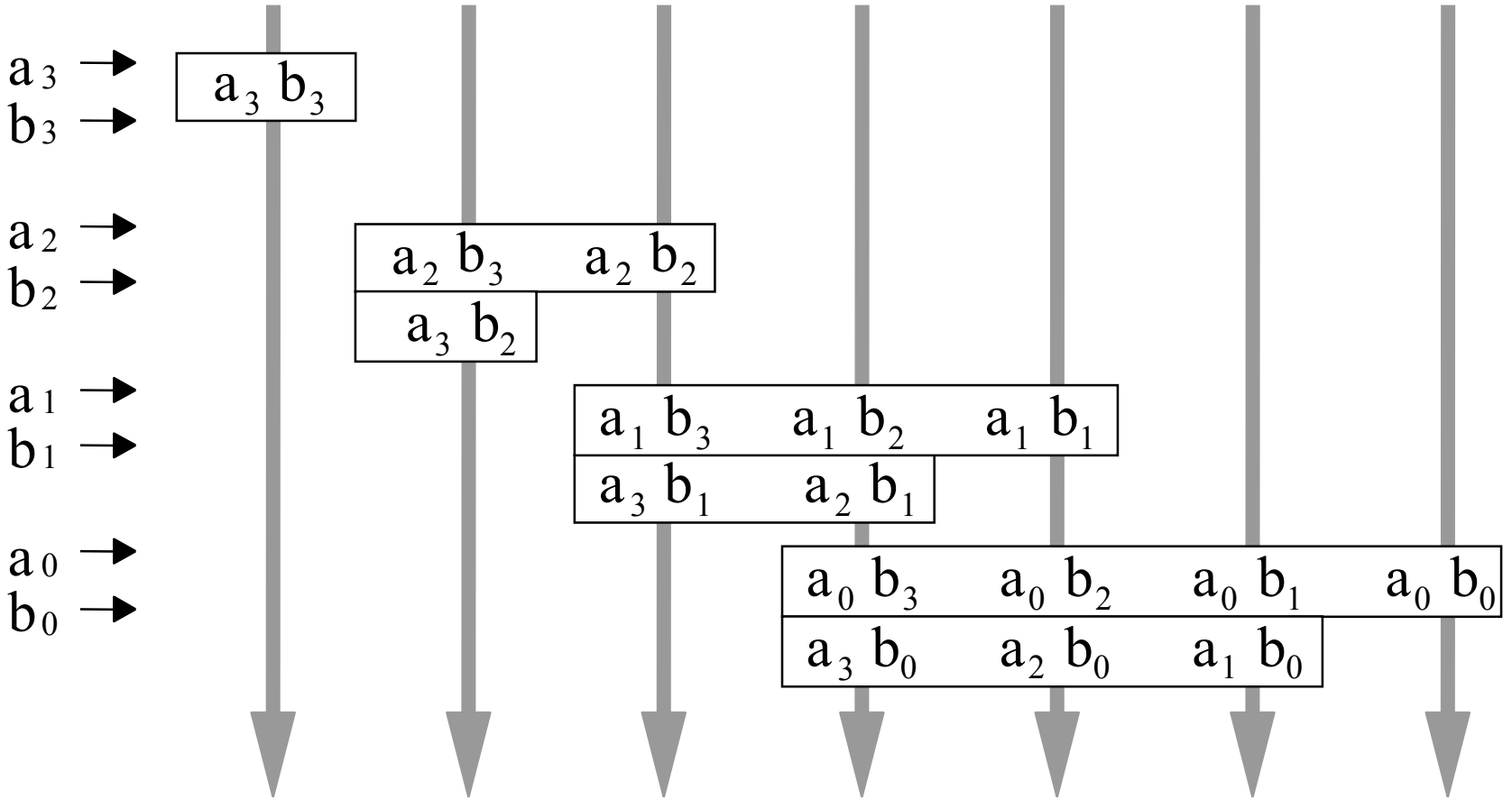
futur

présent

passé



Multiplication en ligne poids forts en tête (3)



sequence of numbers summed up to give a sequence of digits

Opérations à faire simultanément pour la multiplication en-ligne (4)

1- Ranger les chiffres reçus dans les registres A & B

$$A^j \Leftarrow A^{j-1} \,\&\, a_j \quad B^j \Leftarrow B^{j-1} \,\&\, b_j$$

2- Multiplier A et B par les chiffres reçus $\otimes$

$$A^j * b_j \;,\; B^{j-1} * a_j$$

3- Décaler et sortir le résultat partiel

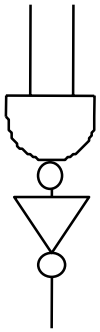
$$P^j \Leftarrow 2 * P^{j-1}$$

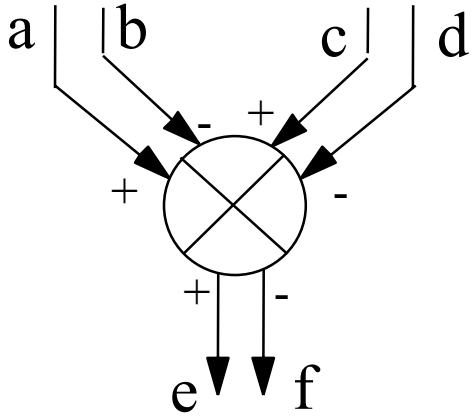
4- Ajouter les produits partiels au résultat partiel

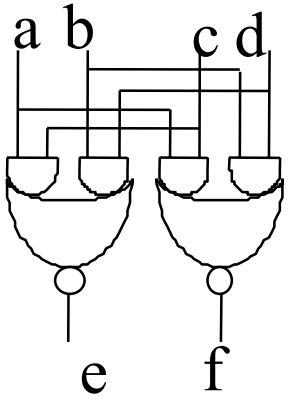
$$P^j \Leftarrow P^j + A^j * b_j + B^{j-1} * a_j$$

Multiplication de 2 chiffres

As for bits, the product of two signed digits is one digit.



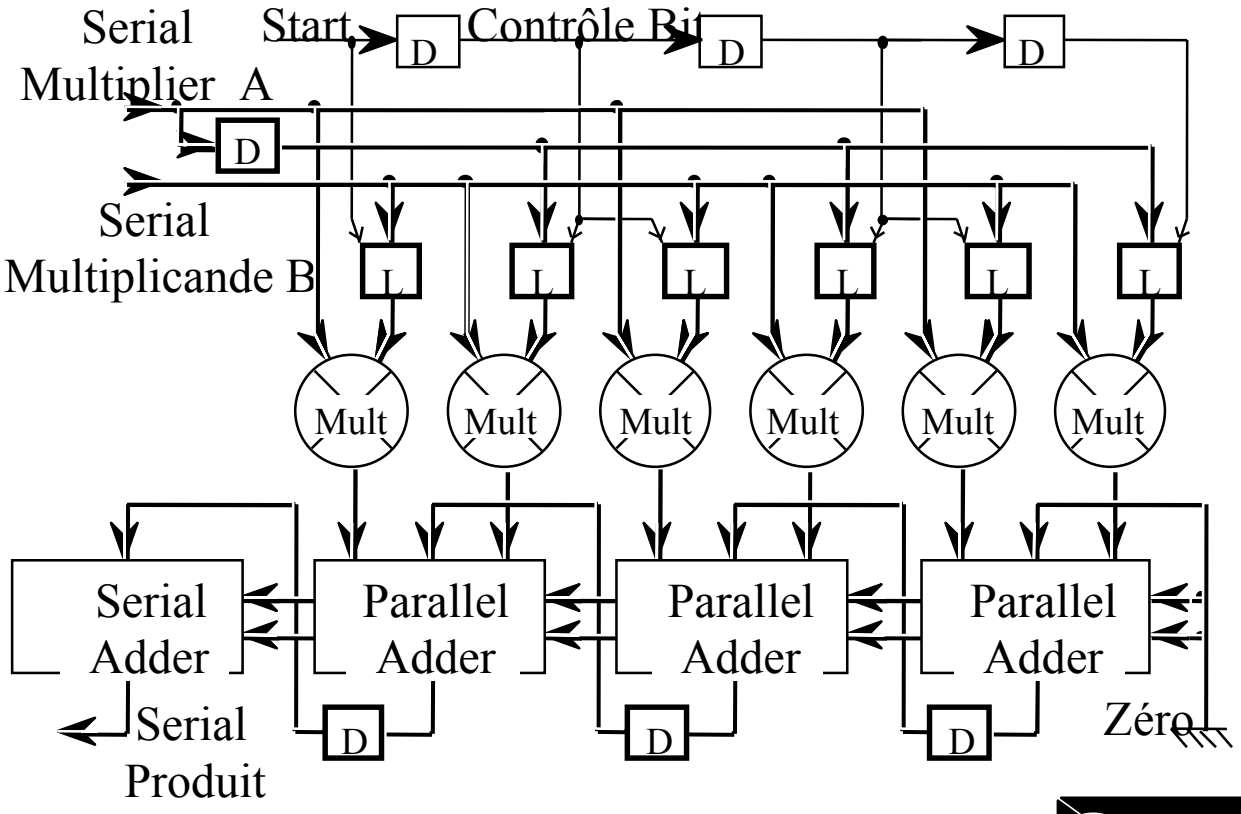


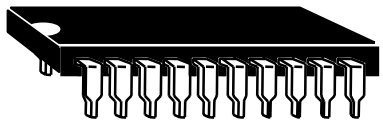


$$e = \overline{a\,c + b\,d}$$

$$f = a\,d + b\,c$$

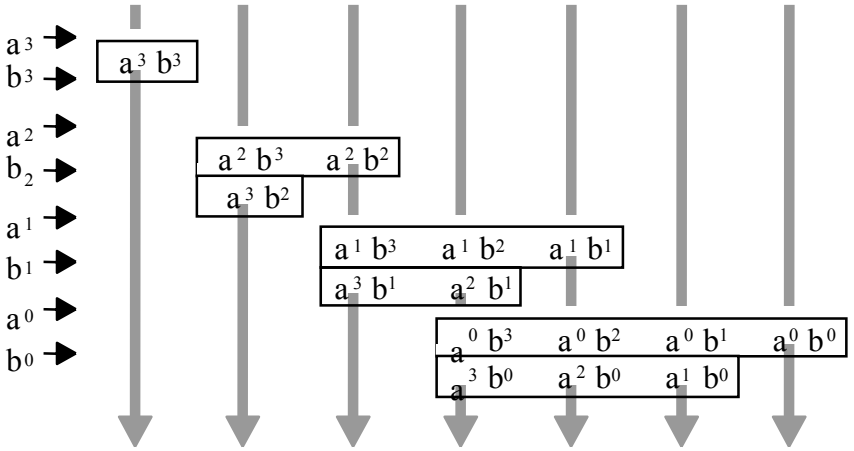
Multiplieur en-ligne poids forts en tête (5)



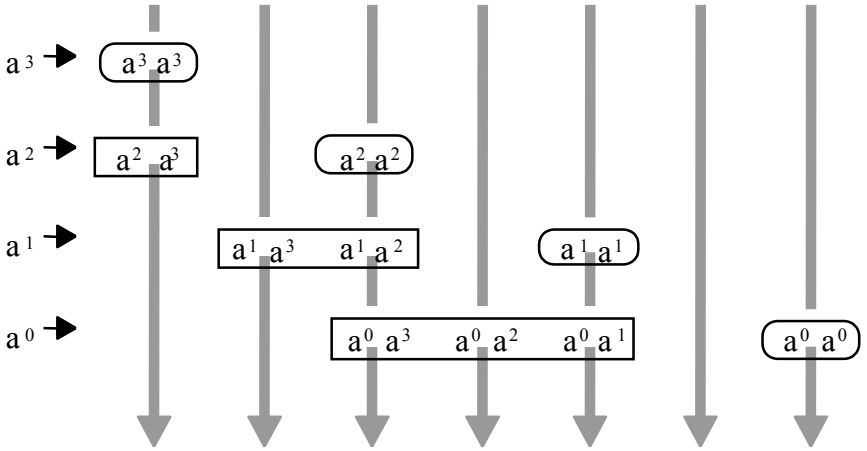


La latence vient du sommateur en-ligne

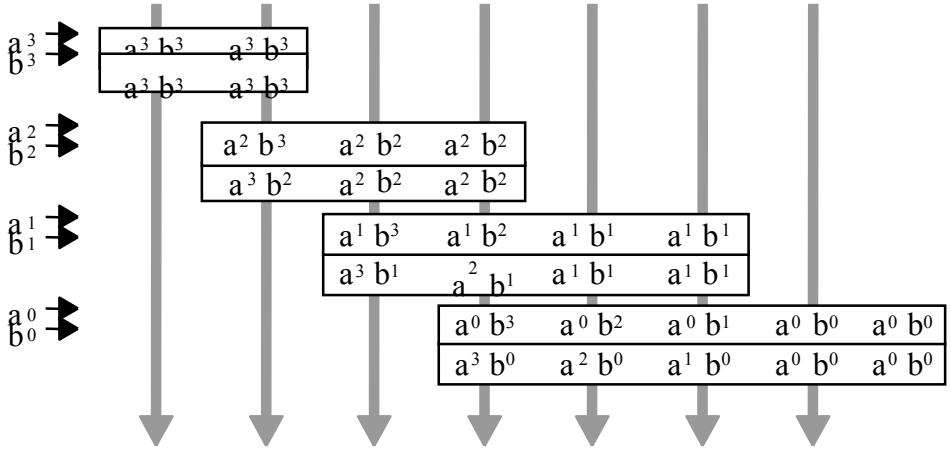
Multiplication et carré

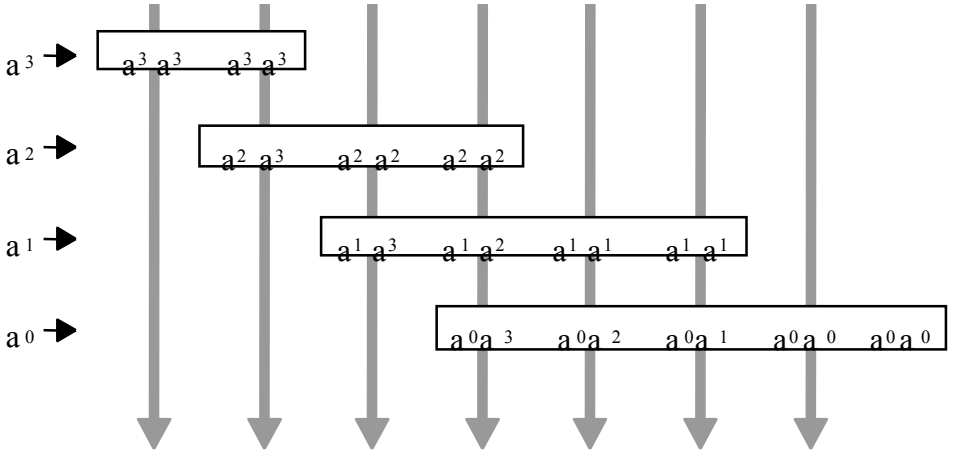


multiply

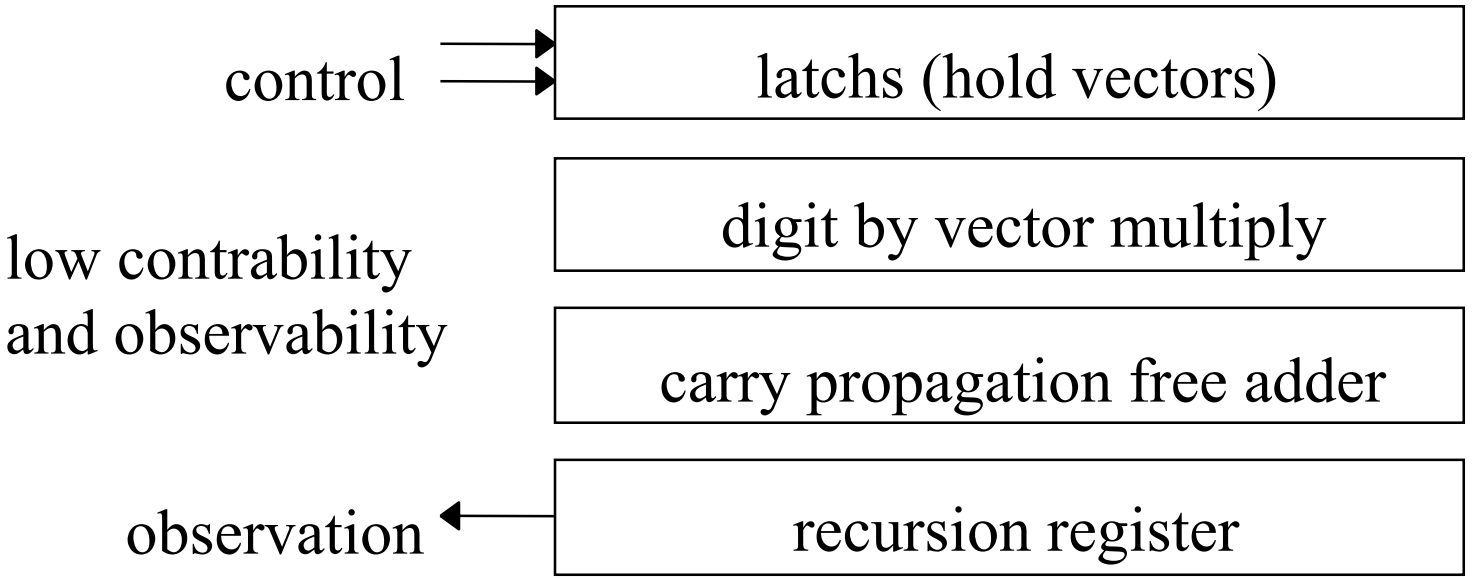


square



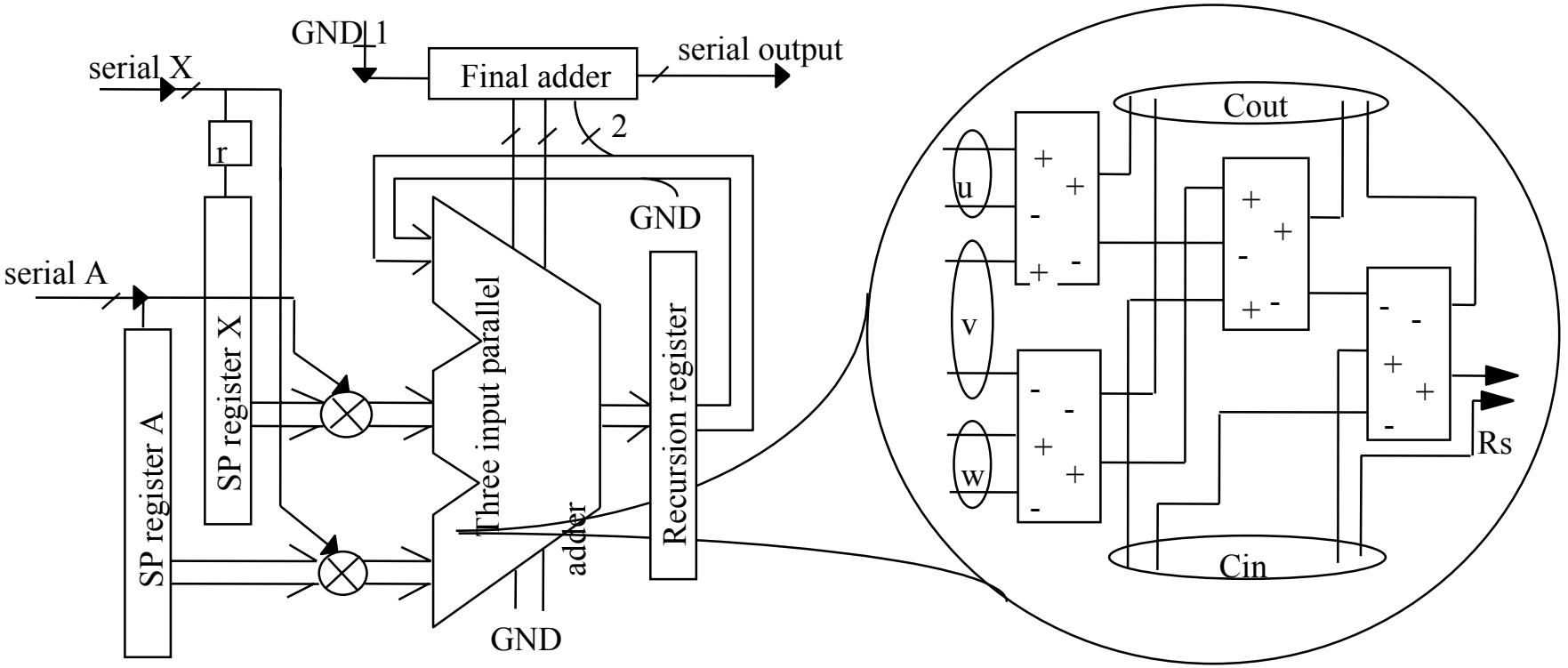


Test en ligne de multiplieur en ligne poids forts en tête



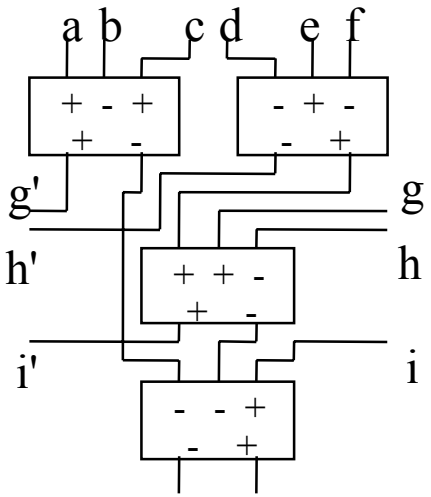
principle: exhaustive testing: no fault model
single fault : no masking
adder and recursion register to analyse the results

Test en ligne de multiplieur en ligne poids forts en tête (2)



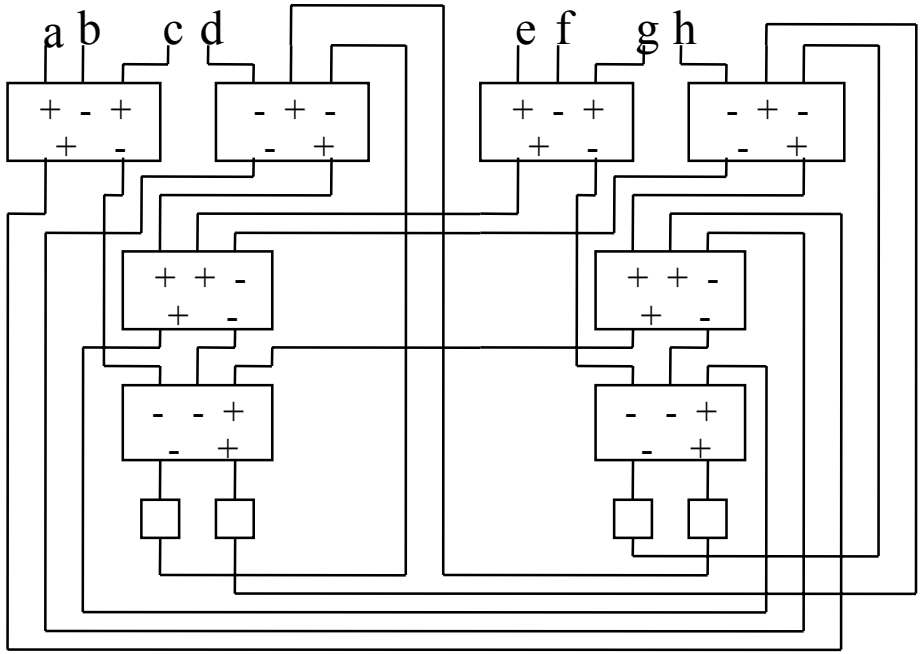
Test en ligne de multiplieur en ligne poids forts en tête (3)

The 3-input carry-free adder is C-testable



input vectors						carries		
a	b	c	d	e	f	g	h	i
0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	1	0
0	0	1	1	1	0	1	0	1
0	1	0	0	1	0	0	0	1
0	1	0	1	0	1	0	1	0
0	1	1	1	0	1	0	1	0
1	0	0	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1
1	0	1	0	1	1	1	0	1
1	1	0	0	0	1	0	1	0
1	1	0	1	1	1	0	1	0
1	1	1	1	0	0	1	1	1

Test en ligne de multiplieur en ligne poids forts en tête (4)

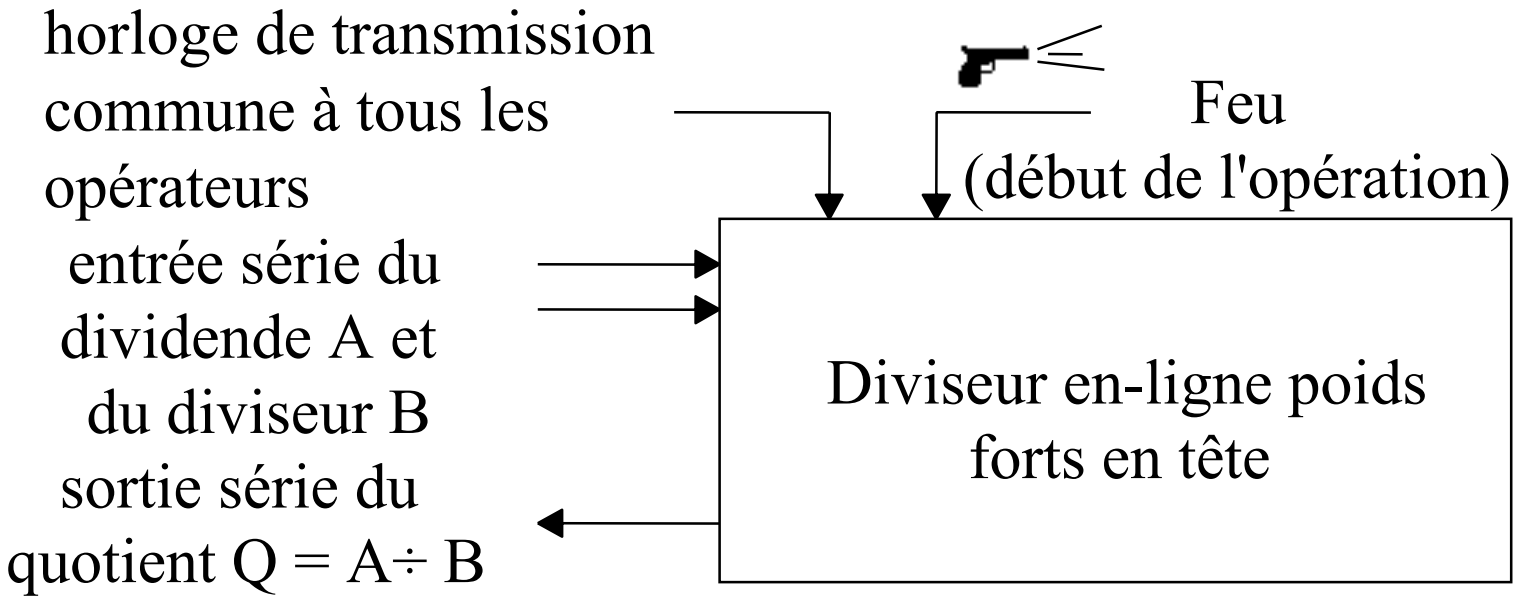


With a local feed-back loop,
more vectors are required

a	b	c	d	e	f	g	h
0	1	0	0	1	1	0	0
0	0	0	0	0	0	1	1
1	1	1	0	0	0	0	1
0	0	1	1	0	0	0	0
0	0	0	1	1	1	1	0
1	0	0	1	0	1	1	0
0	1	1	0	1	0	0	1
1	0	1	0	0	1	0	0
0	1	0	0	1	0	1	0
1	1	1	0	1	0	1	0
1	1	0	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	0
1	0	1	0	0	0	0	1
1	1	1	0	1	1	1	0

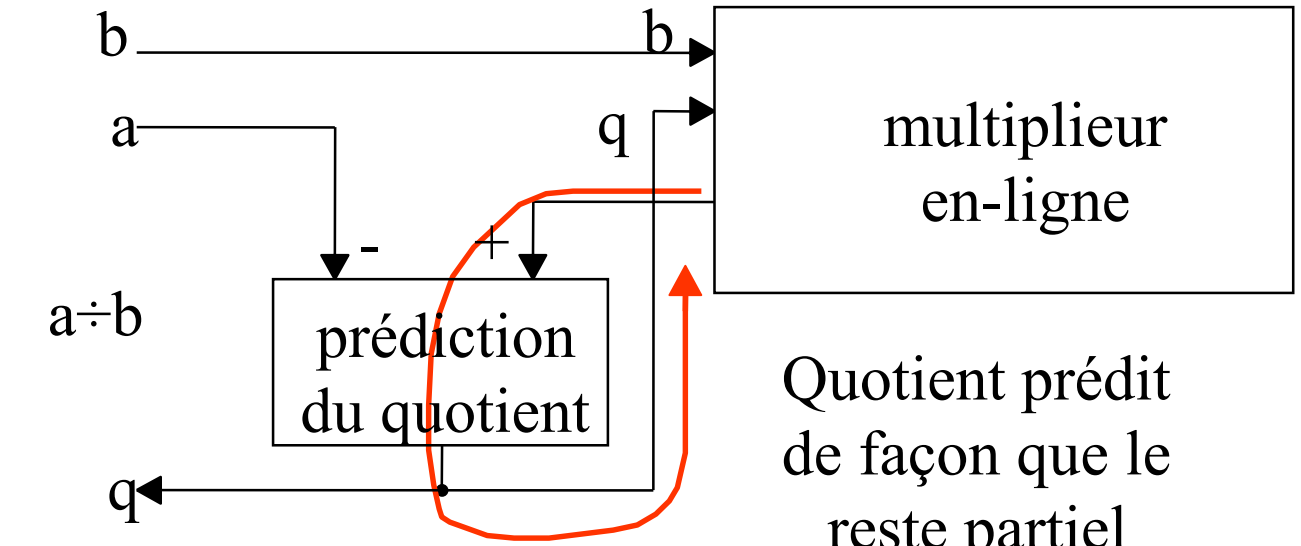
Diviseur en-ligne
(poids forts en tête)

La division s'appuie sur la multiplication en-ligne poids forts en tête



Division en-ligne (2)

Principe



Quotient prédit de façon que le reste partiel

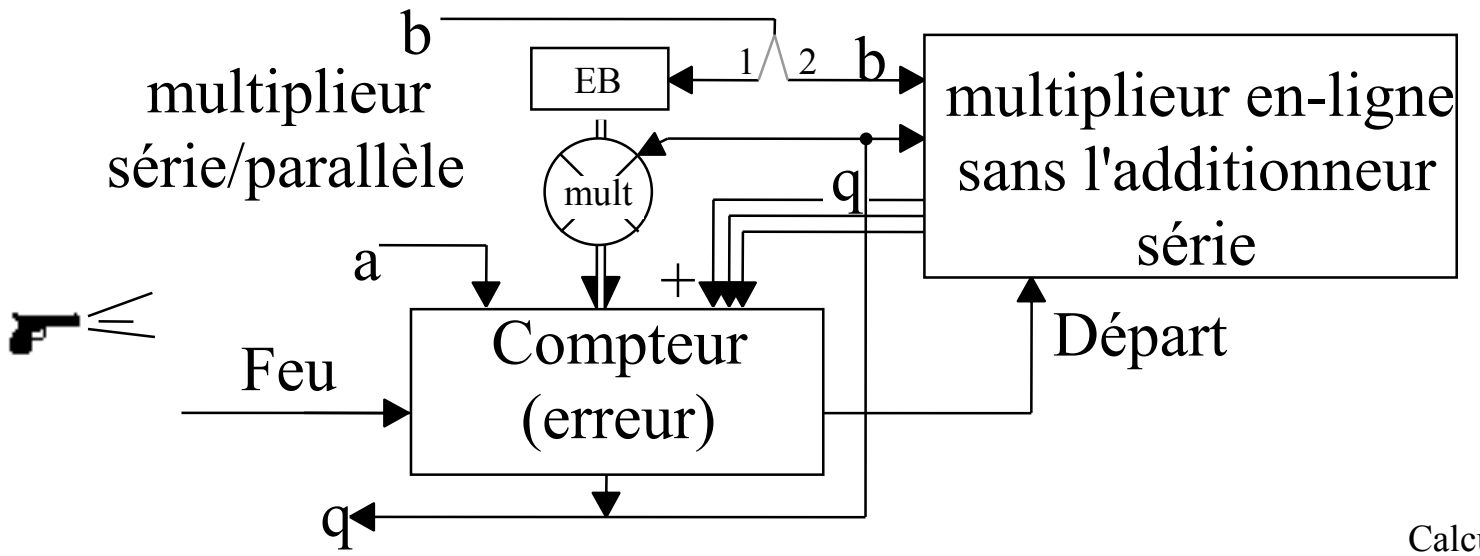
$Q^J * B^J - A^j \rightarrow 0$

boucle de rétroaction

Division en-ligne (3)

Pour assurer la stabilité (ou convergence) de l'algorithme, on doit
1- raccourcir la boucle de rétroaction
2- utiliser une partie du diviseur pour prédire le quotient

La multiplication est partiellement série/parallèle (4 chiffres)
partiellement série/série (en-ligne)



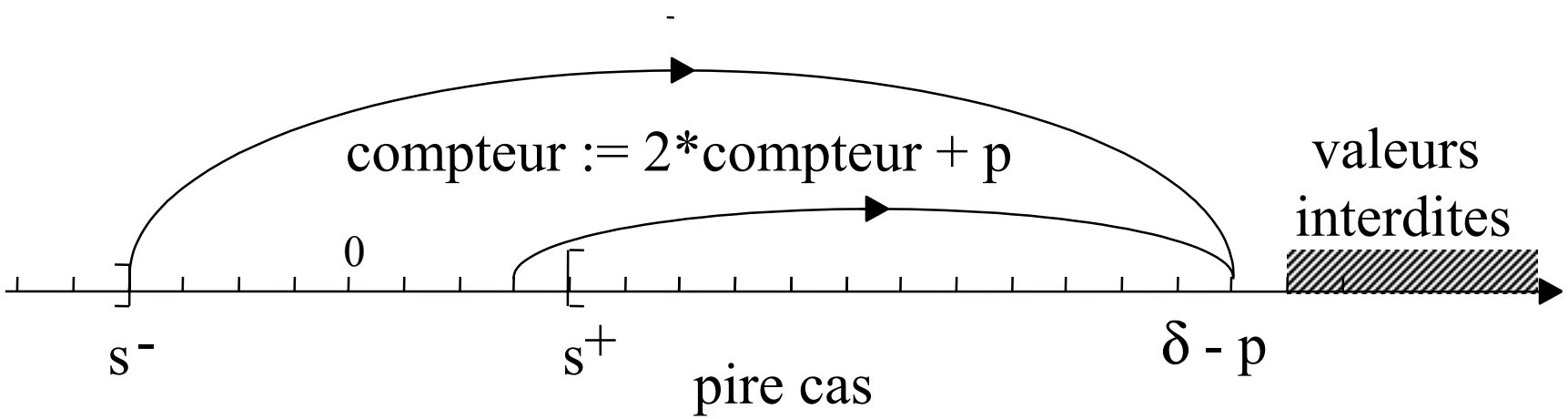
Division en-ligne (4)

A chaque cycle le compteur de prédiction du quotient reçoit
- deux fois son ancienne valeur (connue)
- plus une valeur inconnue p ($-4 \leq p \leq +4$)

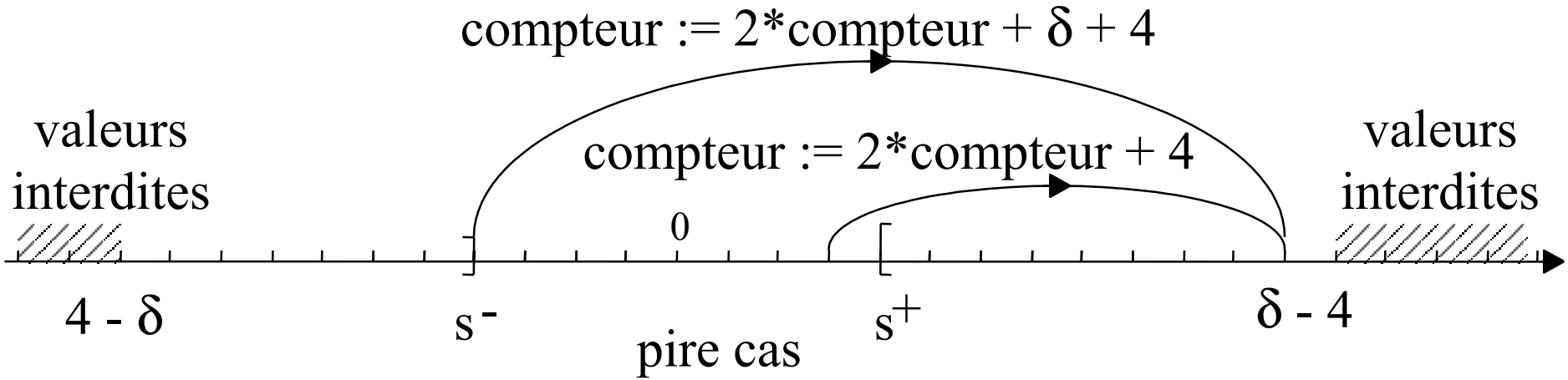
On rend le compteur petit en lui ajoutant ou soustrayant une valeur inconnue EB ($\delta \leq |EB| \leq 2*\delta - 1$) en comparant son ancienne valeur à 2 seuils s+ et s-

Les seuils s+ et s- et δ doivent être faciles à comparer

compteur := 2*compteur - δ + p

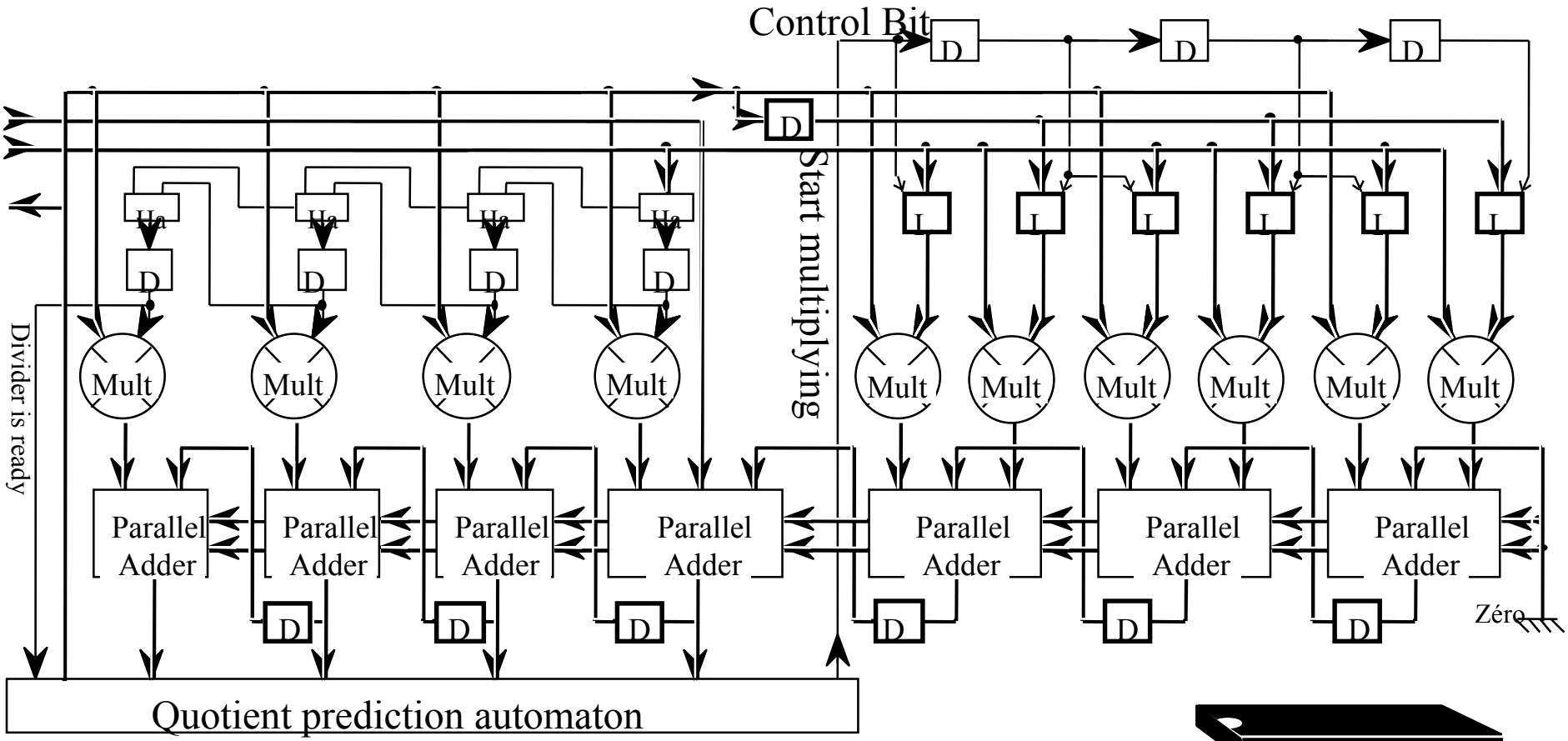


Division en-ligne (5)

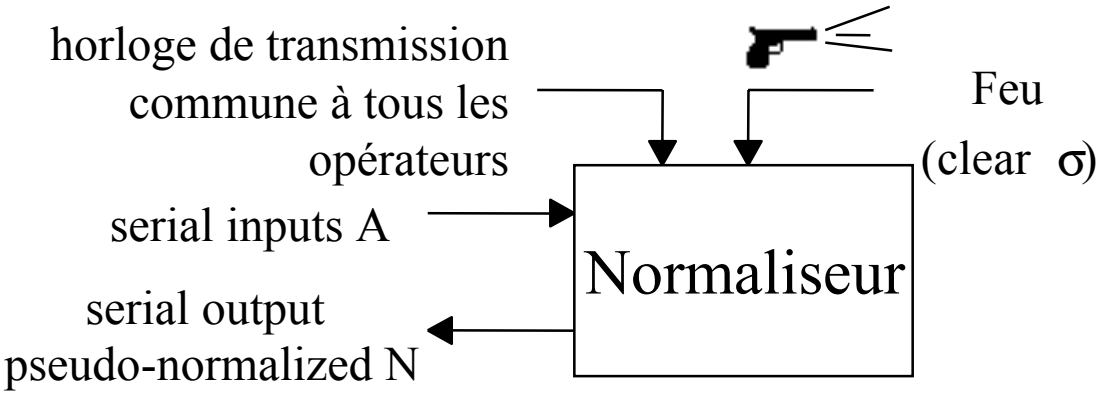


$$2*(s^{+}-1) + 4 \leq \delta - 4$$
$$2*s^{-} + \delta + 4 \leq \delta - 4$$
$$2*(s^{-}+1) - 4 \geq 4 - \delta$$
$$2*s^{+} - \delta - 4 \geq 4 - \delta$$

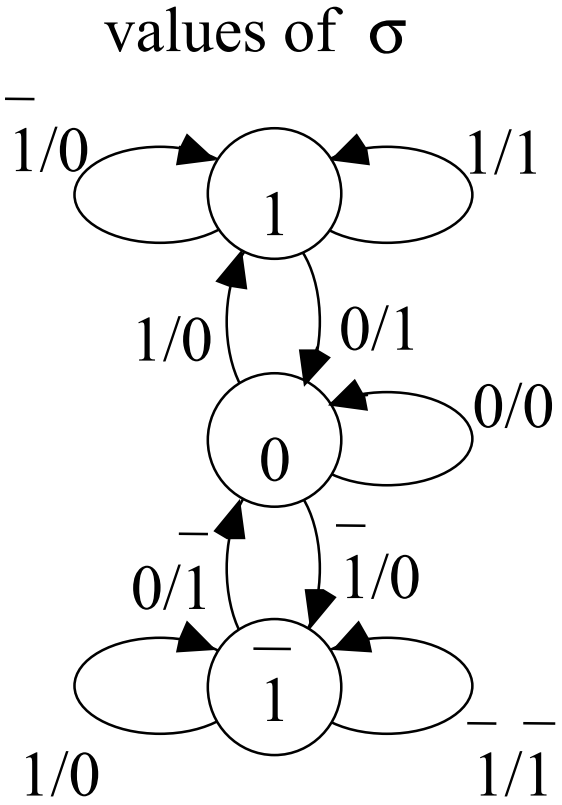
Diviseur en-ligne (6)



Normaliseur en ligne



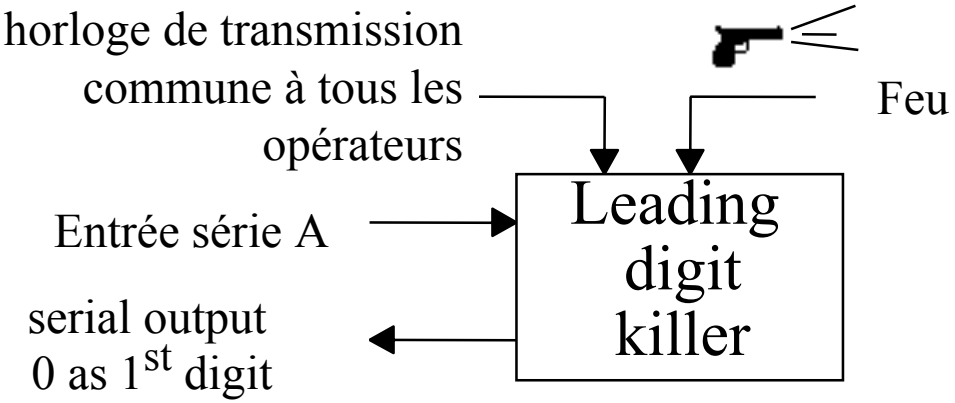

```
procedure normalize (a_i, n_i : SBD) ;
var σ : SBD ;
begin
      σ:= 2  * σ+a_i;
      if σ ≥ 2 then n_i := 1
      else if  σ ≤ 2 then n_i := -1
      else  n_i := 0 ;
      σ:= σ - 2 * n_i;
end .
```



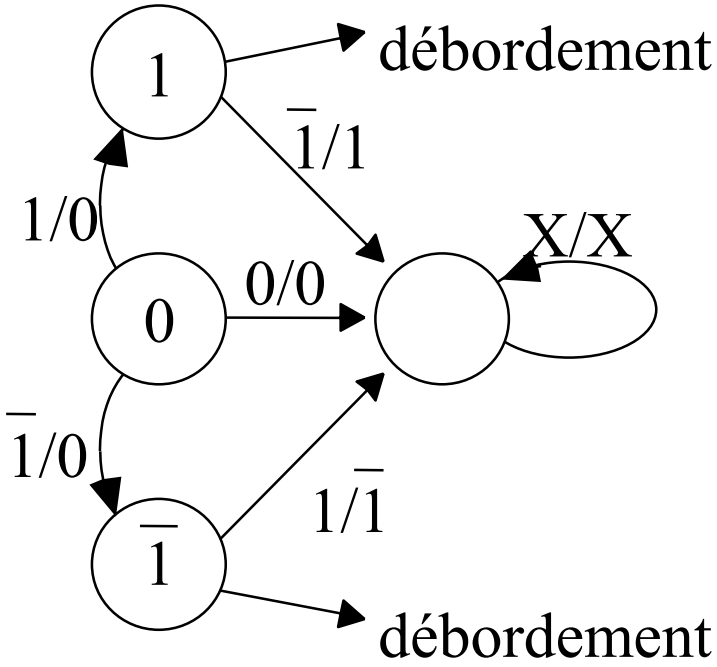
Détection de débordement

$$|A| < 2^n$$

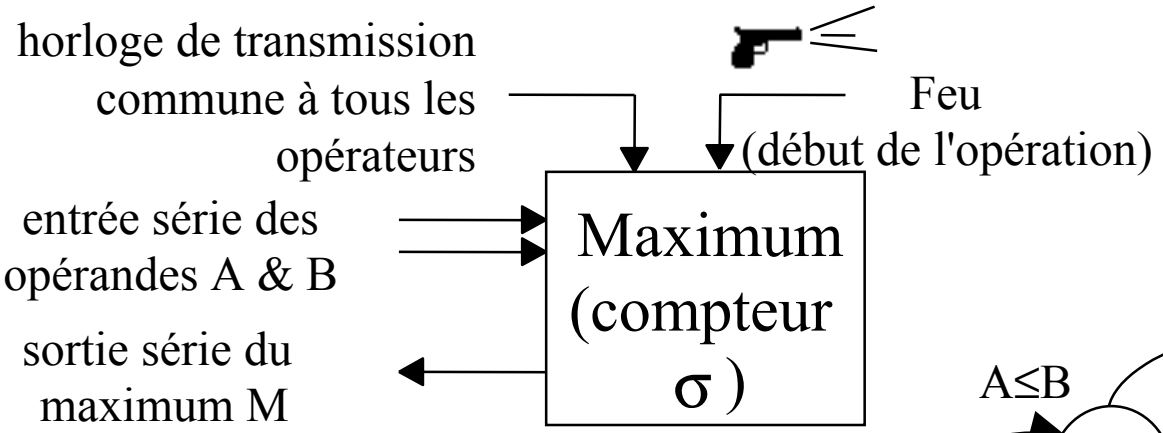
Force a_n à 0



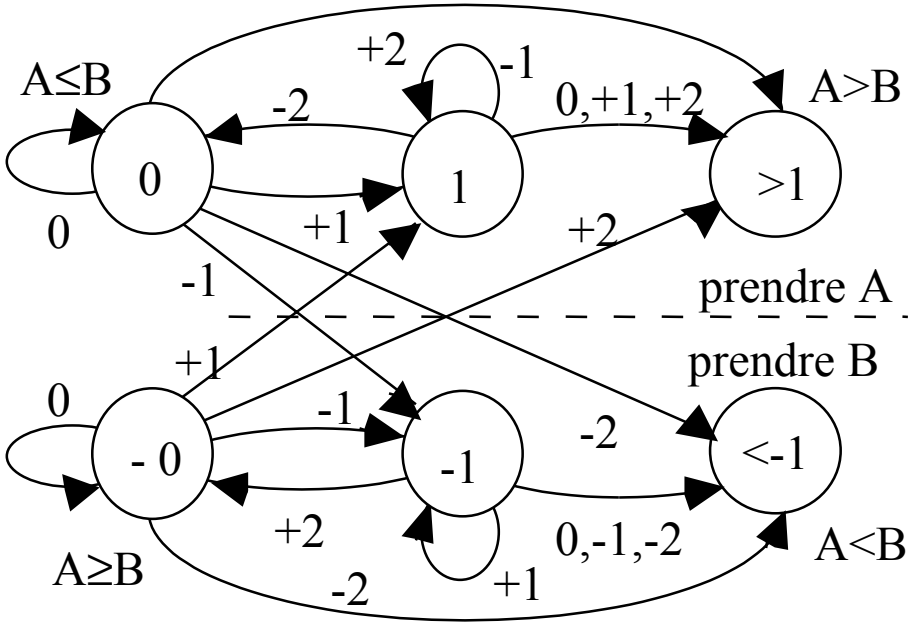
valeurs de σ



Maximum en-ligne de nombres redondants

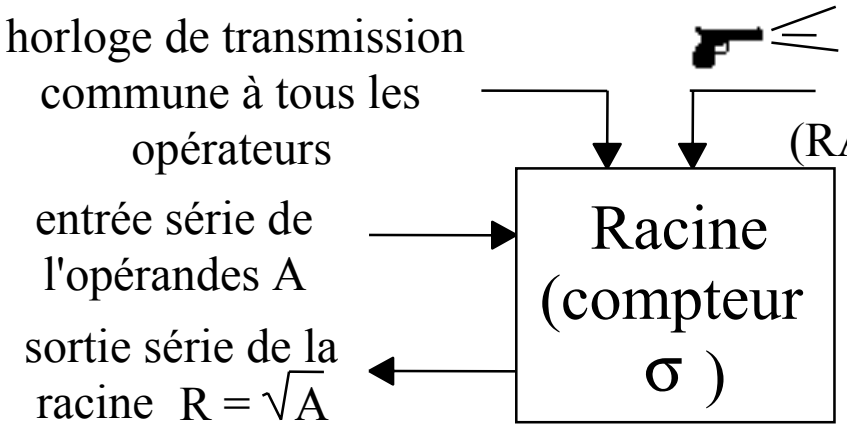


valeurs de σ

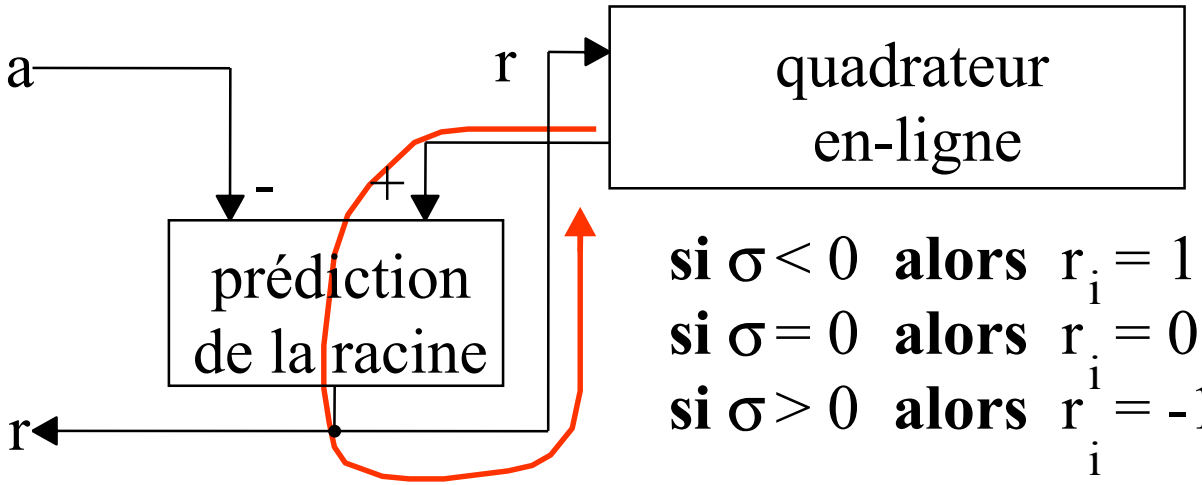


if σ_{j-1} **in** {-1,0,+1}
then $\sigma_j = 2*\sigma_{j-1}+(a_j - b_j)$ **else** $\sigma_j = \sigma_{j-1}$;
if $\sigma_j > 0$ **then** up = **true**;
if $\sigma_j < 0$ **then** up = **false** ;
if up **then** $m_j = a_j$ **else** $m_j = b_j$;

Racine carrée en-ligne de nombres redondants



La racine carrée est sensible à la parité du rang du chiffre le plus significatif du radicande

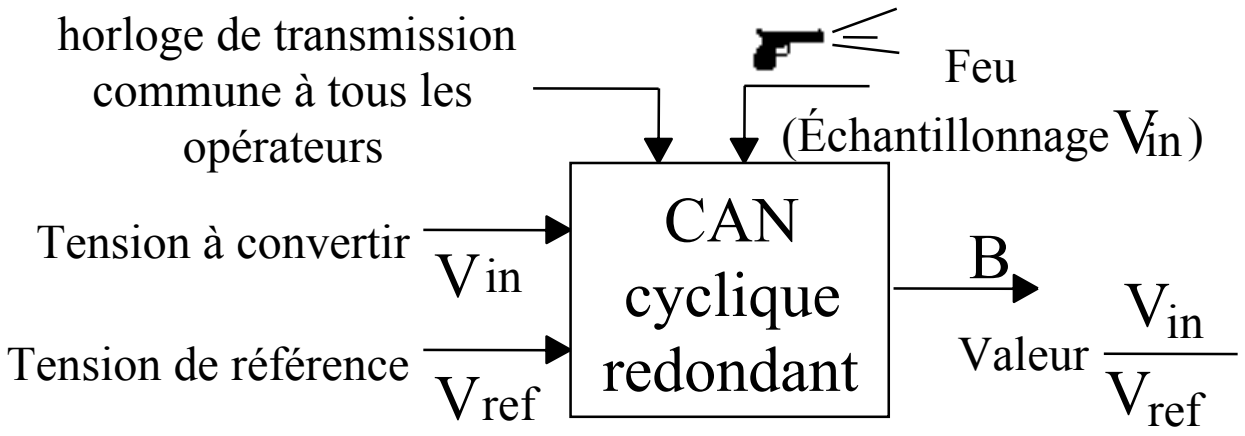


si $\sigma < 0$ **alors** $r_i = 1$

si $\sigma = 0$ **alors** $r_i = 0$

si $\sigma > 0$ **alors** $r_i = -1$

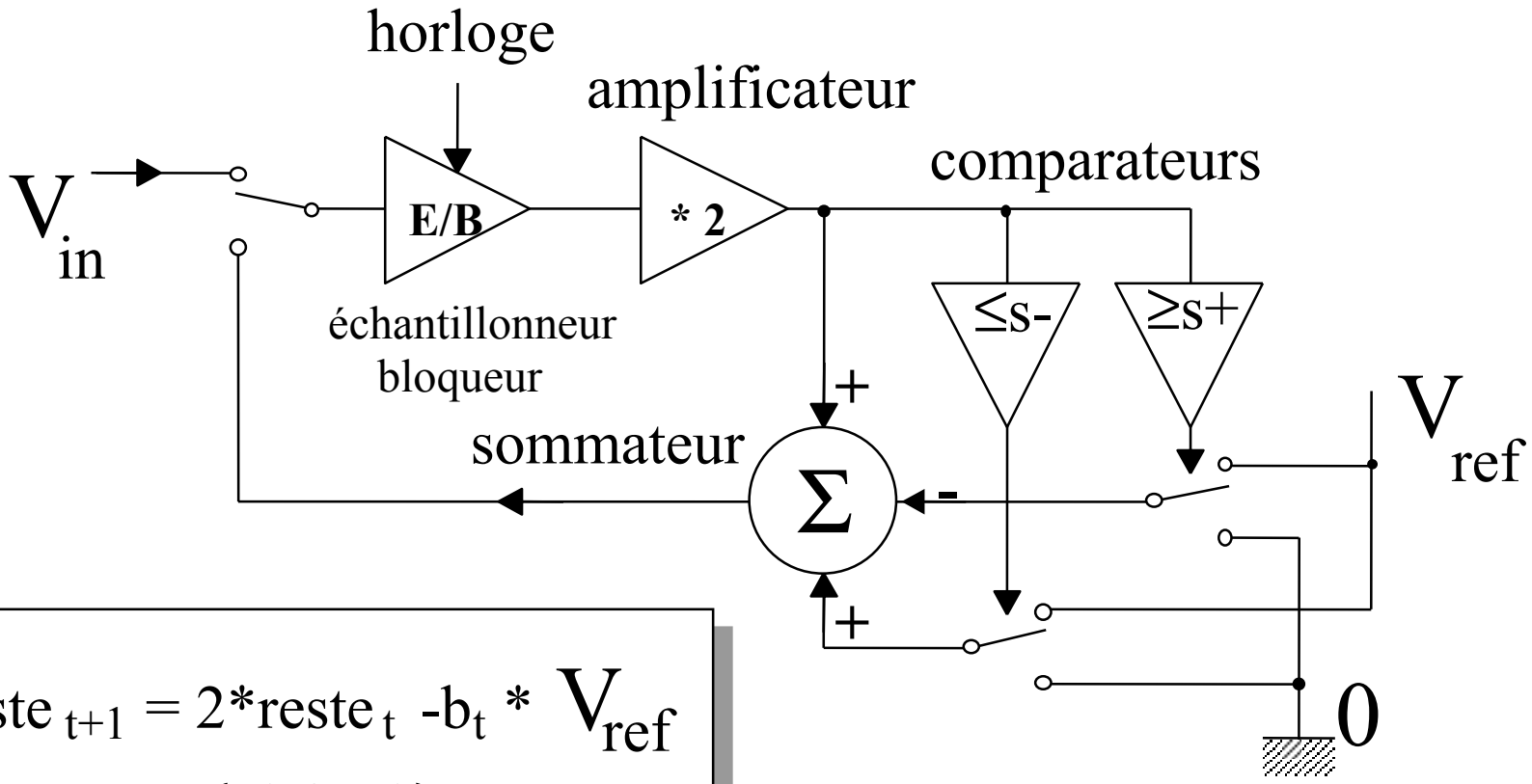
Convertisseur analogique-numérique redondant



$reste_0 := V_{in}$;
si $reste_t \geq (V_{ref}/2 \pm V_{ref}/2)$ **alors** $reste_{t+1} := 2*(reste_t - V_{ref})$

sinon si $reste_t \leq -(V_{ref}/2 \pm V_{ref}/2)$ **alors** $reste_{t+1} := 2*(reste_t + V_{ref})$
sinon $reste_{t+1} := 2*(reste_t)$

Convertisseur analogique-numérique redondant (2)

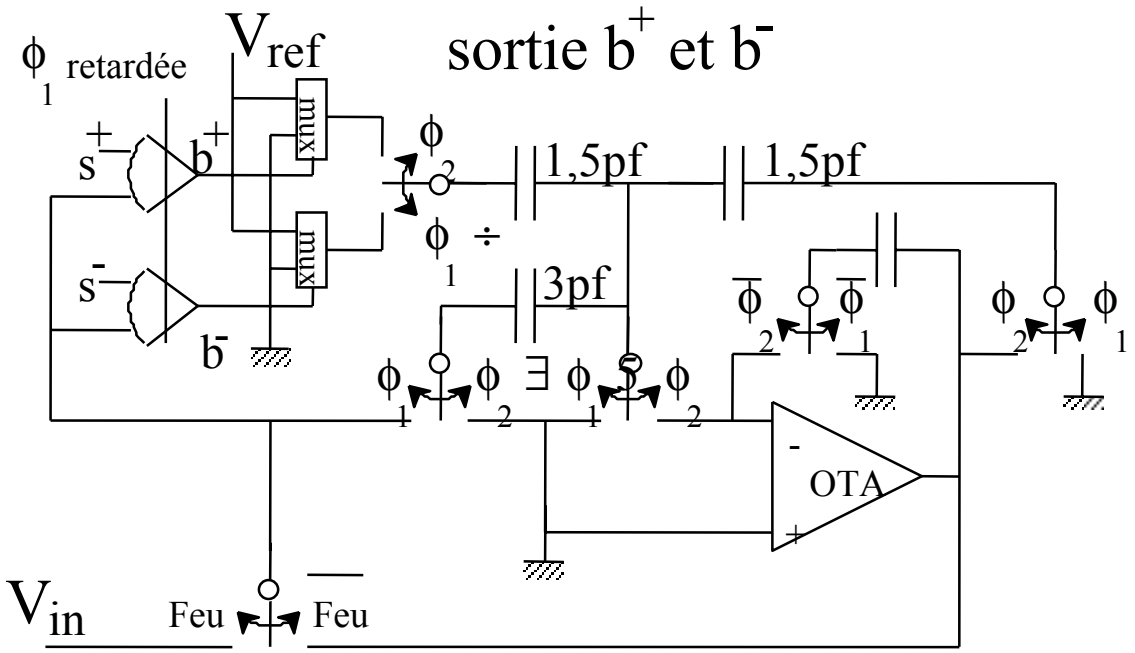


$$reste_{t+1} = 2*reste_t - b_t * V_{ref}$$

$$b_t \quad \{-1,0,+1\}$$

Convertisseur analogique-numérique redondant (3)

(4)



$$\phi_1 \wedge \phi_2 = 0$$

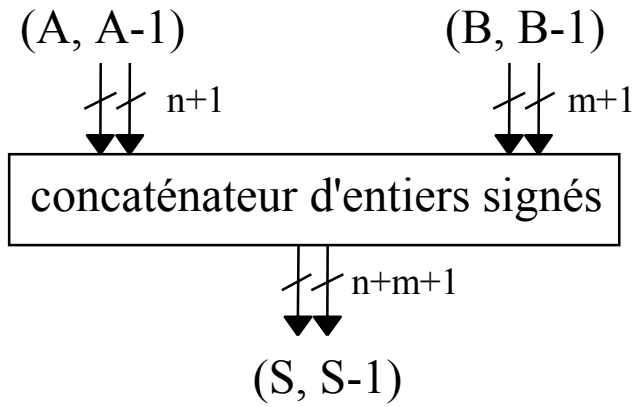
Les phases ϕ_1 et ϕ_2 ne se recouvrent pas

Conversion "à la volée" du redondant

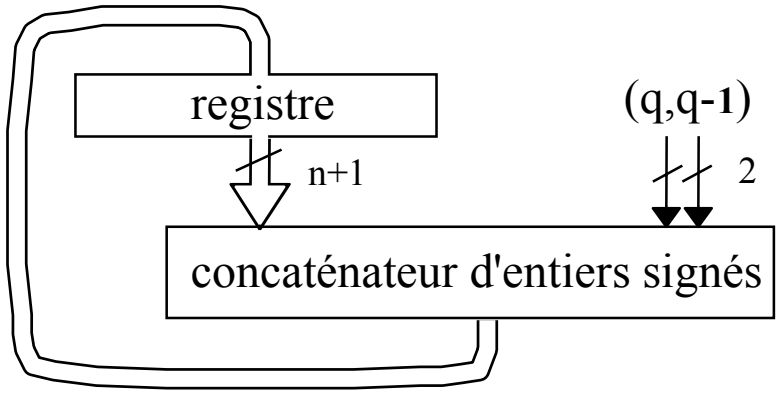
Concaténation de A et B
$S := A * 2^m + B$

si $B \geq 0$ **alors** $S := A \,\&\, B$
sinon $S := (A-1) \,\&\, B$;
si $(B-1) \geq 0$ **alors** $(S-1) := A \,\&\, (B-1)$
sinon $(S-1) := (A-1) \,\&\, (B-1)$;

(le bit de signe de B est omis dans la concaténation &)



q	q-1	bits
1		01
0	0	00
-1	-1	11
	-2	10

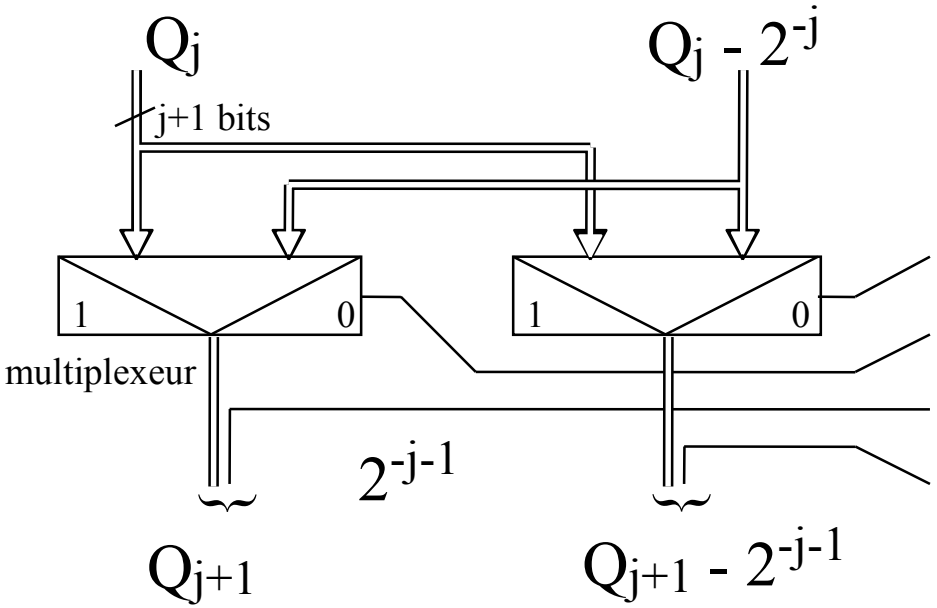


Conversion "à la volée" du redondant (2)

premier $\longrightarrow$ dernier

suite $q_0\, q_1\, q_2 \ldots q_n \quad q_i \in \{-1,0,+1\}$

$$Q_j = \sum_{i=0}^{j} p_i 2^{-i} \quad p_i \in \{0,1\} \qquad Q_0 = q_0 \,,\; Q_0 - 1 = \overline{q_0} \qquad q_0 \neq 0$$



$q_{j+1}\, 2^{-j-1}$

$$-2^{-j-1} = -2^{-j} + 2^{-j-1}$$

$q_{j+1} = 1$

$q_{j+1} \neq 0$

$q_{j+1} \neq 0$

$q_{j+1} = 0$

Combinaisons d'opérateurs en-ligne

Certains opérateurs en-ligne parallélisent implicitement leur(s) opérande(s) ou leur résultat.

Cette propriété peut être exploitée pour construire des opérateurs complexes avec

- un coût moindre
- une latence moindre (mais une période supérieure)

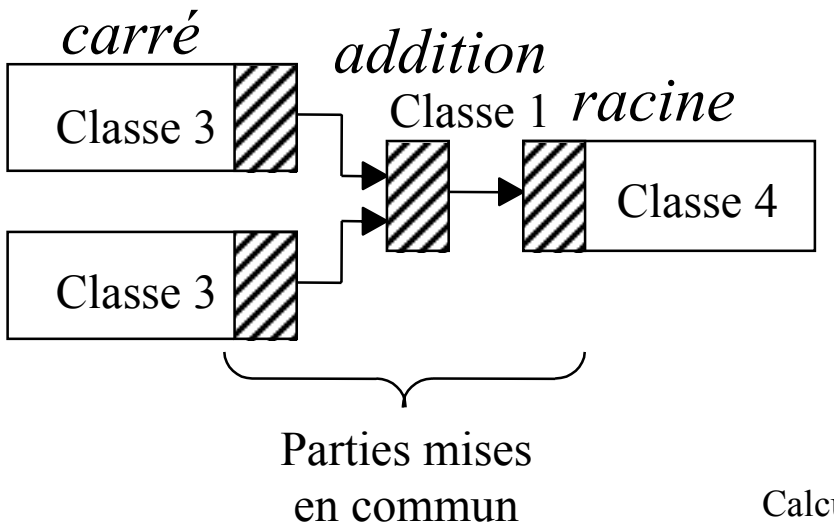
Par exemple :

Opérateur	coût par chiffre	latence
$C = A^2$	76	3
$S = A + B$	0	2
$R = \sqrt{A}$	76	1
$D = \sqrt{A^2 + B^2}$	126	**1**

Classes d'opérateurs en-ligne

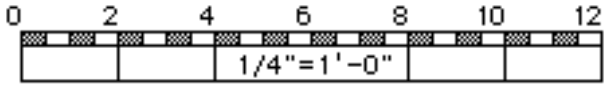
Opérateur	
Abs/max/min	1
Add/Sous $S = A \pm B$	2
Multiplication $P = A * B$	3
Carré $C = A^2$	
Division $Q = A \div B$	4
Racine carrée $R = \sqrt{A}$	

Classement des opérateurs en-ligne en fonction de leur affinité (possibilité de partage)



Opérateur	# classe	# variante	Opérandes A et B		résultat en série	résultat en parallèle	opérande en parallèle	coût matériel de l'opérateur en nombre de : SP register	Parallel inpuy register	vector/digit multiply⊗	2-input adder	Recursion register
Abs/max/min	1	/	ser	ser	M	na	na	0	0	0	0	0
Add/Sous $S = A \pm B$	2	1	par	par	na	S	A,B	0	0	0	1	0
		2	ser	ser	S	na	na	0	0	0	0	0
Multiplication $P = A * B$	3	1	par	ser	P	na	/	0	1	1	1	1
		2	ser	ser	P	na	A,B	2	0	2	2	1
Carré $C = A^2$		1	ser		C	na	A	1	0	1	1	1
		2	par/ser		C	na	/	0	1	1	1	1
Division $Q = A \div B$	4	1	par	par	Q	na	/	0	0	1	1	1
		2	par	ser	Q	Q	B	2	0	2	2	1
		3	ser	par	Q	na	/	0	0	1	1	1
		4	ser	ser	Q	Q	B	2	0	2	2	1
Racine carrée $R = \sqrt{A}$		1	par		R	na	/	1	0	1	1	1
		2	ser		R	R	na	0	1	1	1	1

Règles de combinaison d'opérateurs



Règle 1 : Deux opérateurs de classe 3 partageant la même entrée peuvent partager le même SP-registre.

Règle 2 : Si la sortie d'un opérateur de classe 4 est reliée à l'entrée d'un opérateur de classe 3 alors ils peuvent partager un SP-registre.

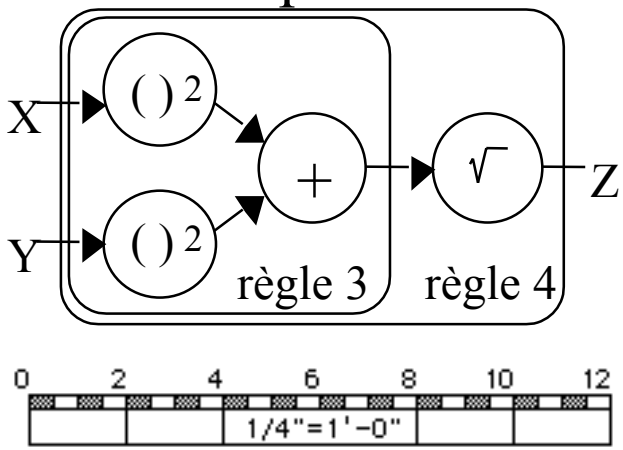
Règle 3 : Si les sorties d' opérateurs de classe 3 sont ajoutées (ou soustraites) alors ils peuvent partager leur registre de récursion. Le résultat de cette fusion est de classe 3.

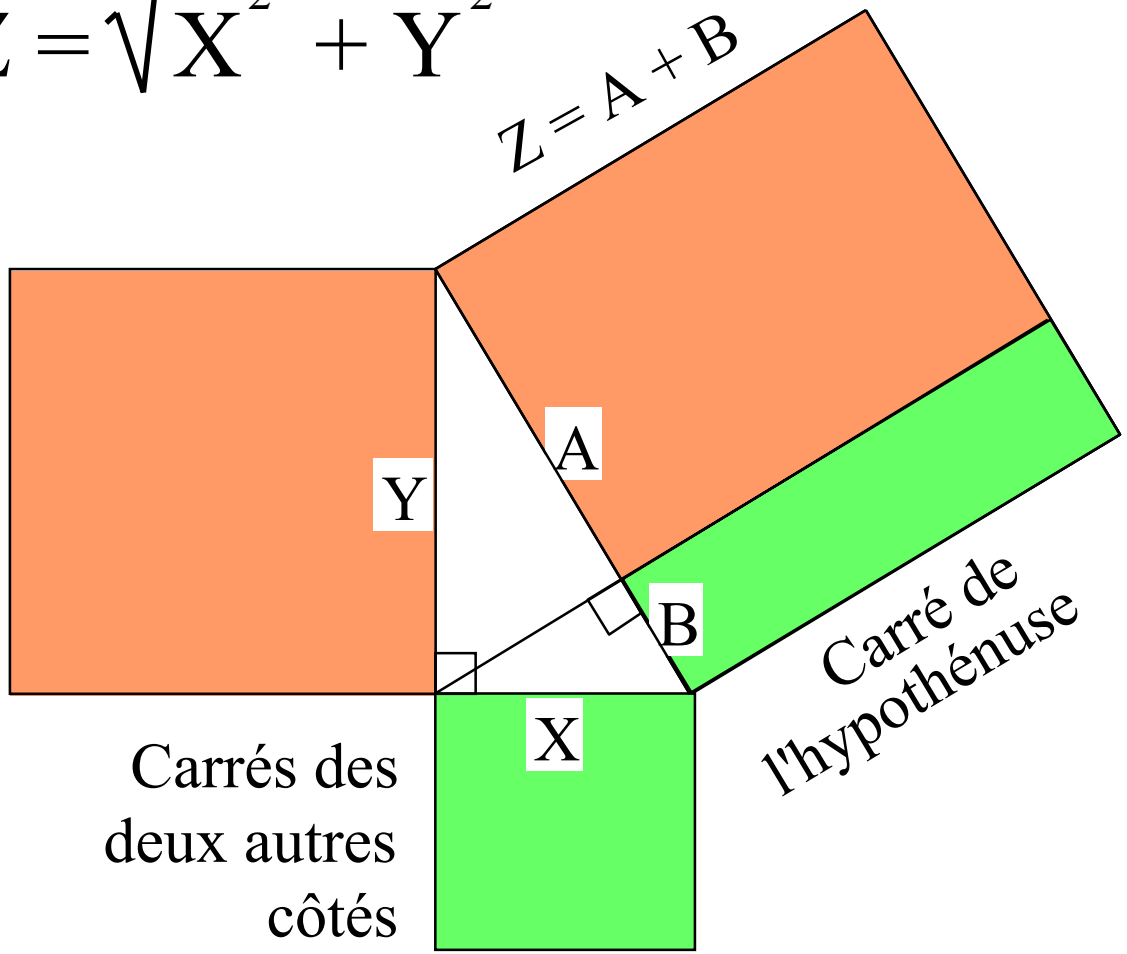
Règle 4 : Si la sortie d'un opérateur de classe 3 est reliée à l'entrée d'un opérateur de classe 4 ils peuvent partager leur registre de récursion.

Exemple 1: processeur Euclidien

$$Z = \sqrt{X^2 + Y^2}$$

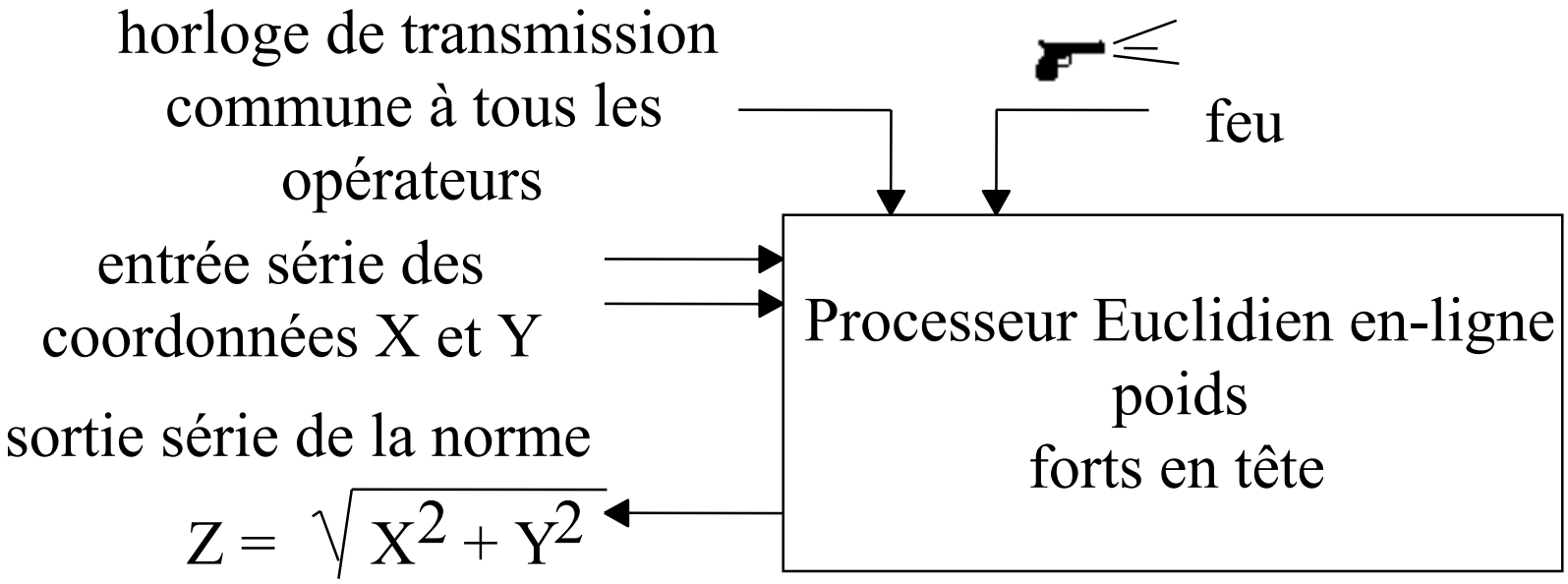
Règles d'assemblage
des opérateurs





Norme Euclidienne en-ligne

$$Z = \sqrt{X^2 + Y^2}$$

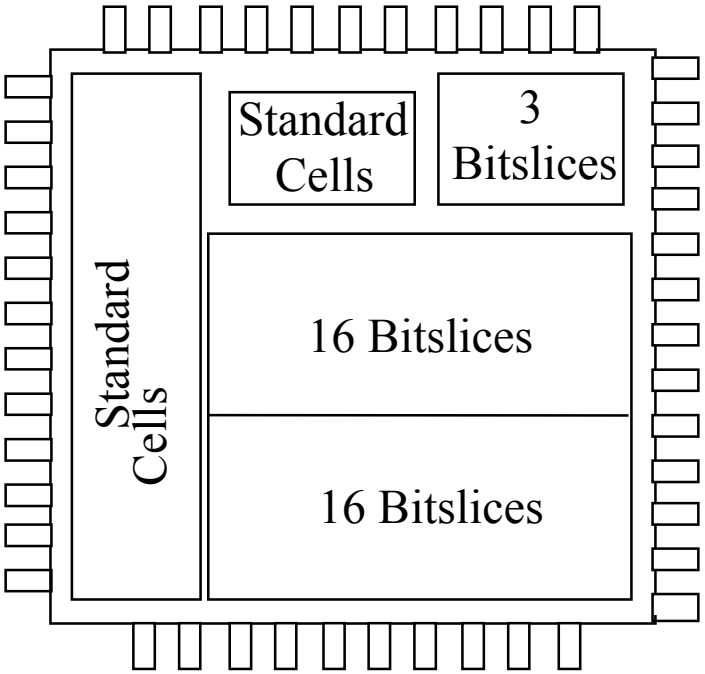


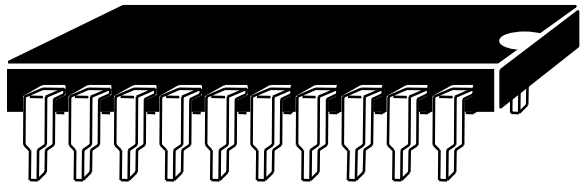
Norme Euclidienne en-ligne (2)

$$Z = \sqrt{X^2 + Y^2}$$ en-ligne

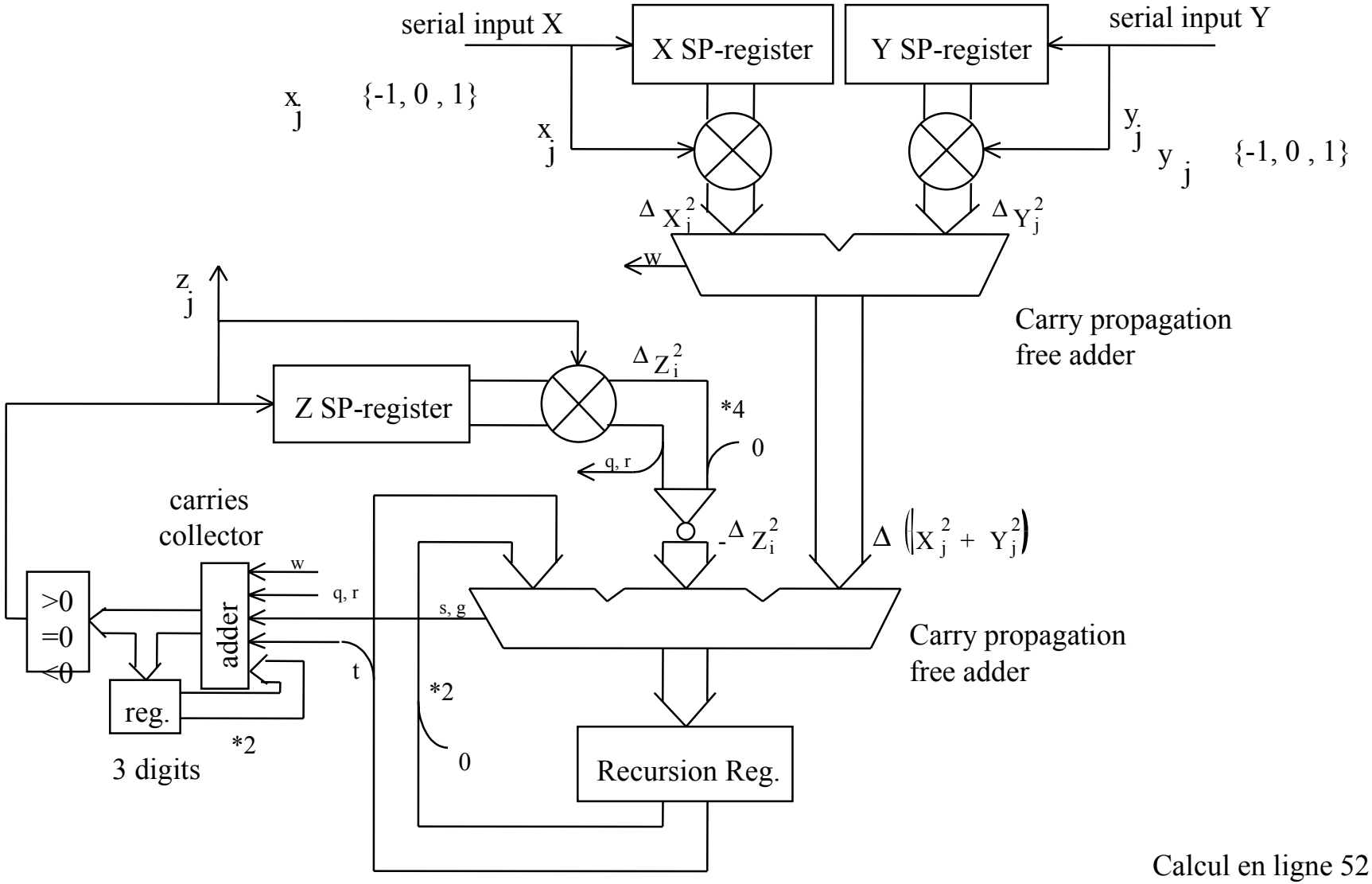
avec 32 chiffres de précision

Application des règles : gain 19%





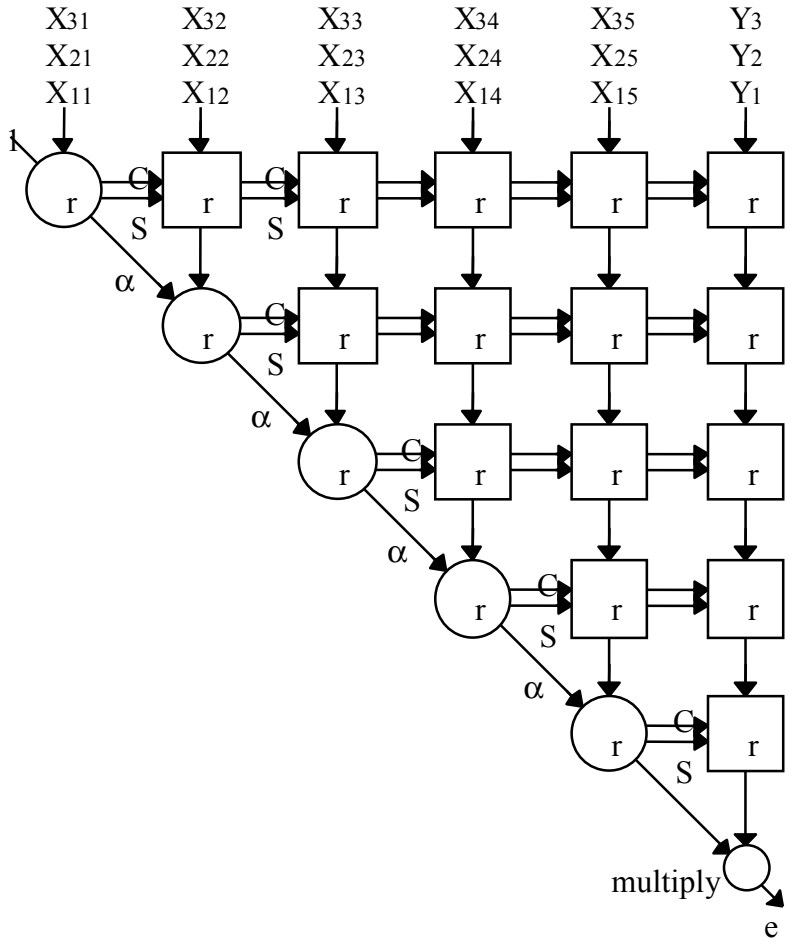
Norme Euclidienne en-ligne (3)



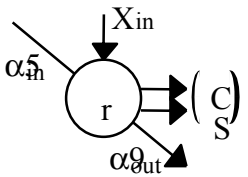
Exemple 2: filtre de Kalman

avec Jin Zian MOU ENST-Paris

β est le carré du facteur d'oubli.
Le circuit ci-contre n'est pas systolique, il n'y a pas de retard horizontal. Il est "pipeliné" au niveau du chiffre. Toutes les entrées se font en série.

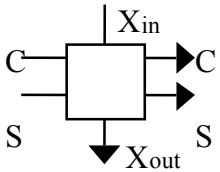


Description fonctionnelle de la cellule ronde



$r' = \sqrt{r^2 + X_{in}^2}$
$C = r \div r'$
$S = X_{in} \div r'$
$\alpha_{out} = C \cdot \alpha_{in}$
$r = \beta \cdot r'$

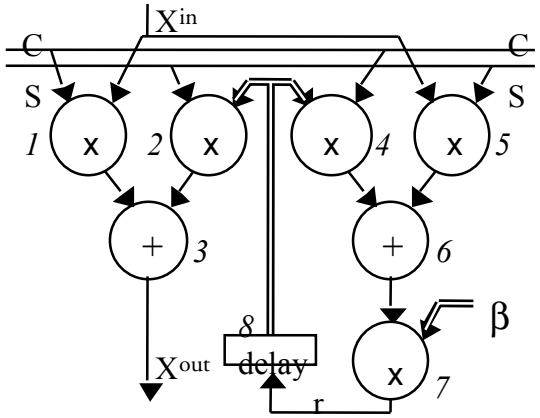
Description fonctionnelle de la cellule carrée



$X_{out} = (C*X_{in} - S*r)$
$r = \beta*(S*X_{in} - C*r)$

Filtre de Kalman (2)

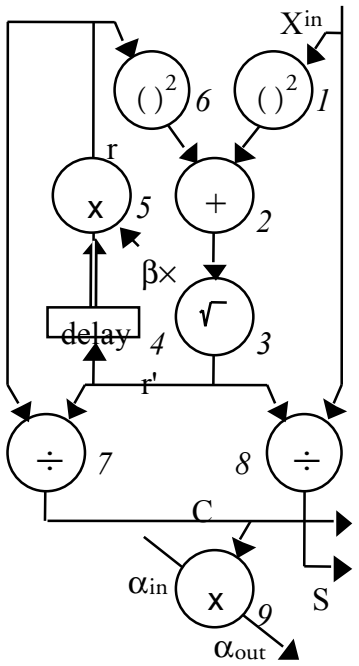
cellule carrée

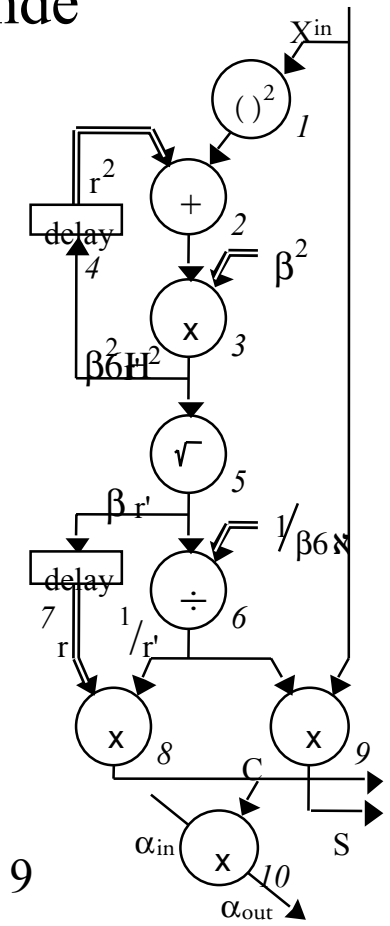


Règle 1 sur 1 & 4
règle 3 sur 1,2,3,4,5 & 6
gain de 19%

Règle 3 puis 4 sur 1,2,3 & 6
règle 2 sur 3,4,7 & 8
gain de 20%

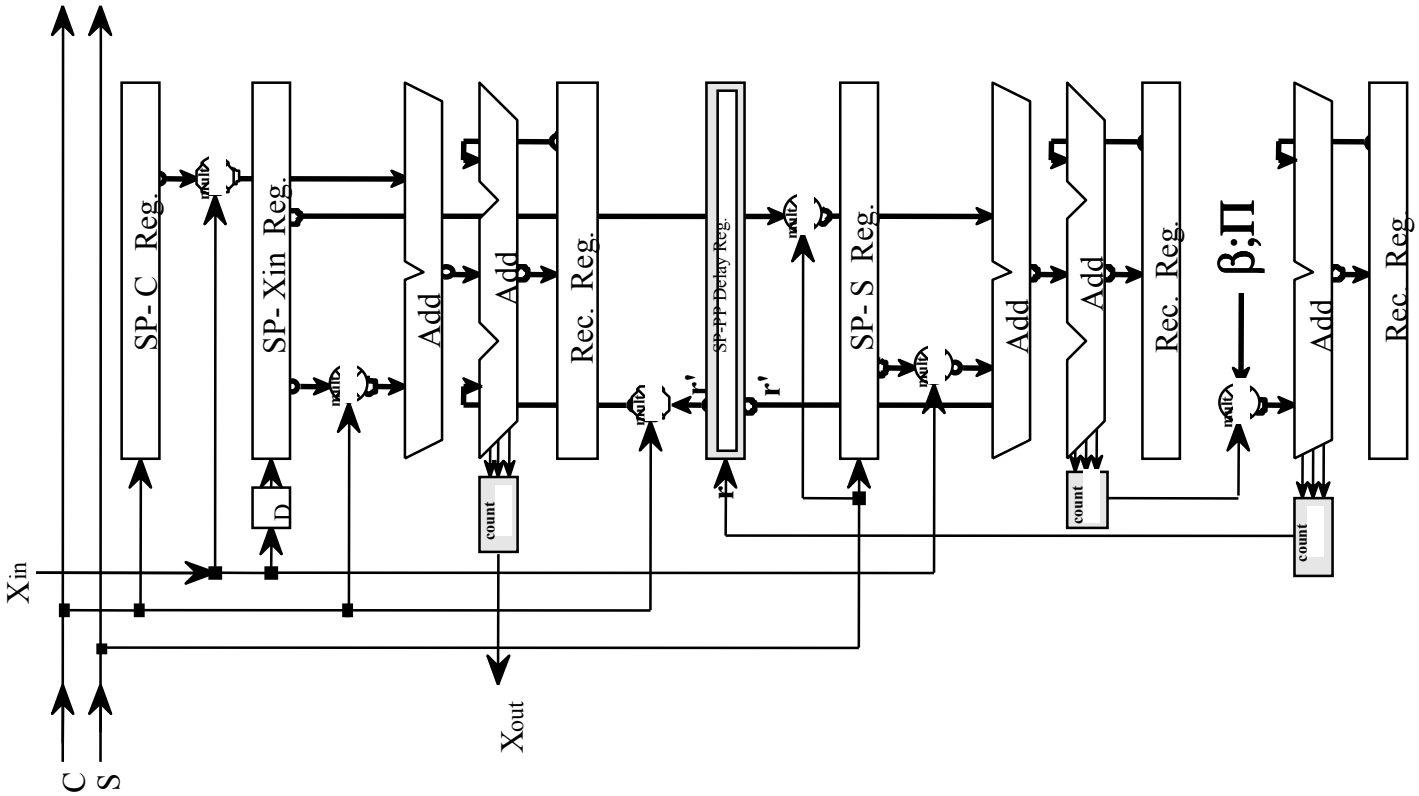
cellule ronde



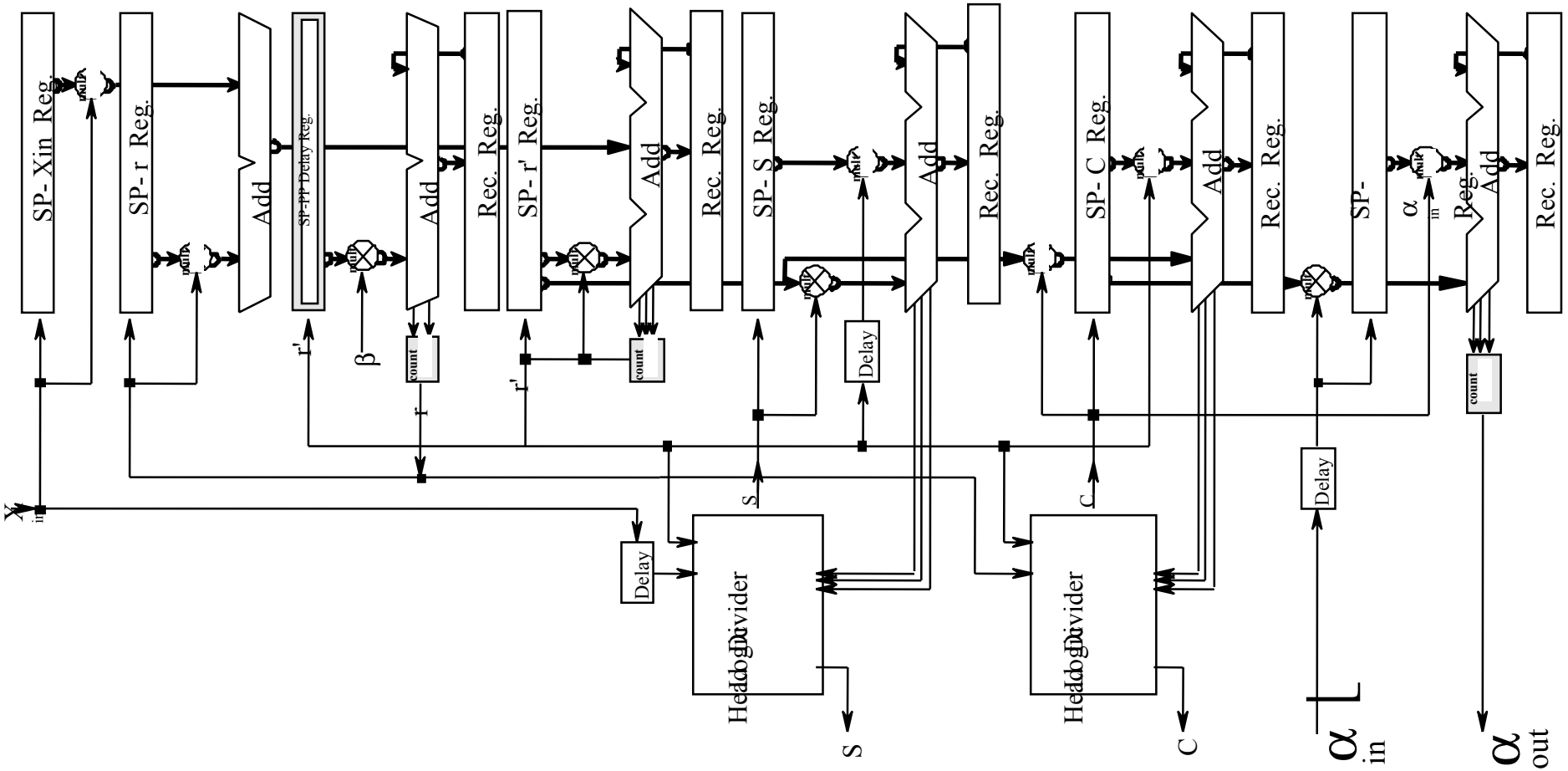


Règle 1 sur 1 & 2 puis 1 & 9
règle 2 sur 5,6 & 7 puis 6,8 & 9
gain de 21%

Cellule carrée



Cellule ronde

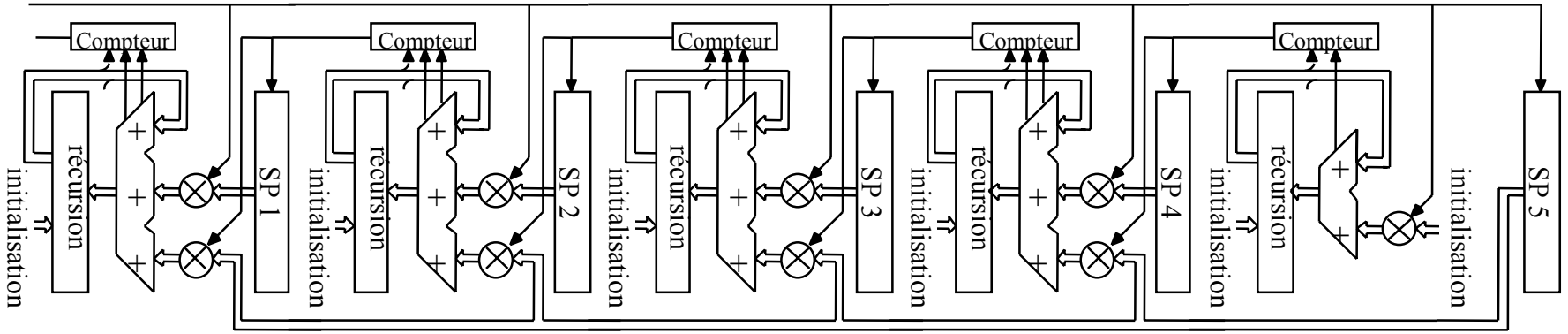


Calcul en ligne 56

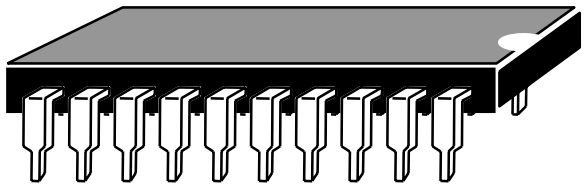
Polynômieur en ligne

$$e^x = \sum_{i=0}^{n} \frac{x^i}{i!} \quad \text{en-ligne}$$

application des règles: gain 19%



$$a_0 + x * (a_1 + x * (a_2 + x * (a_3 + x * (a_4 + x * a_5))))$$



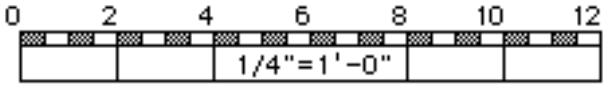
Polynômieur en ligne (2)

$$e^x = \sum_{i=0}^{n} \frac{x^i}{i!} \quad \text{en-ligne}$$

Règle : Si une constante est à ajouter à la sortie d'un opérateur de type 3 ou à l'entrée d'un opérateur de type 4 alors

son coût est nul (initialisation du registre de récursion)

sa latence d'addition est nulle

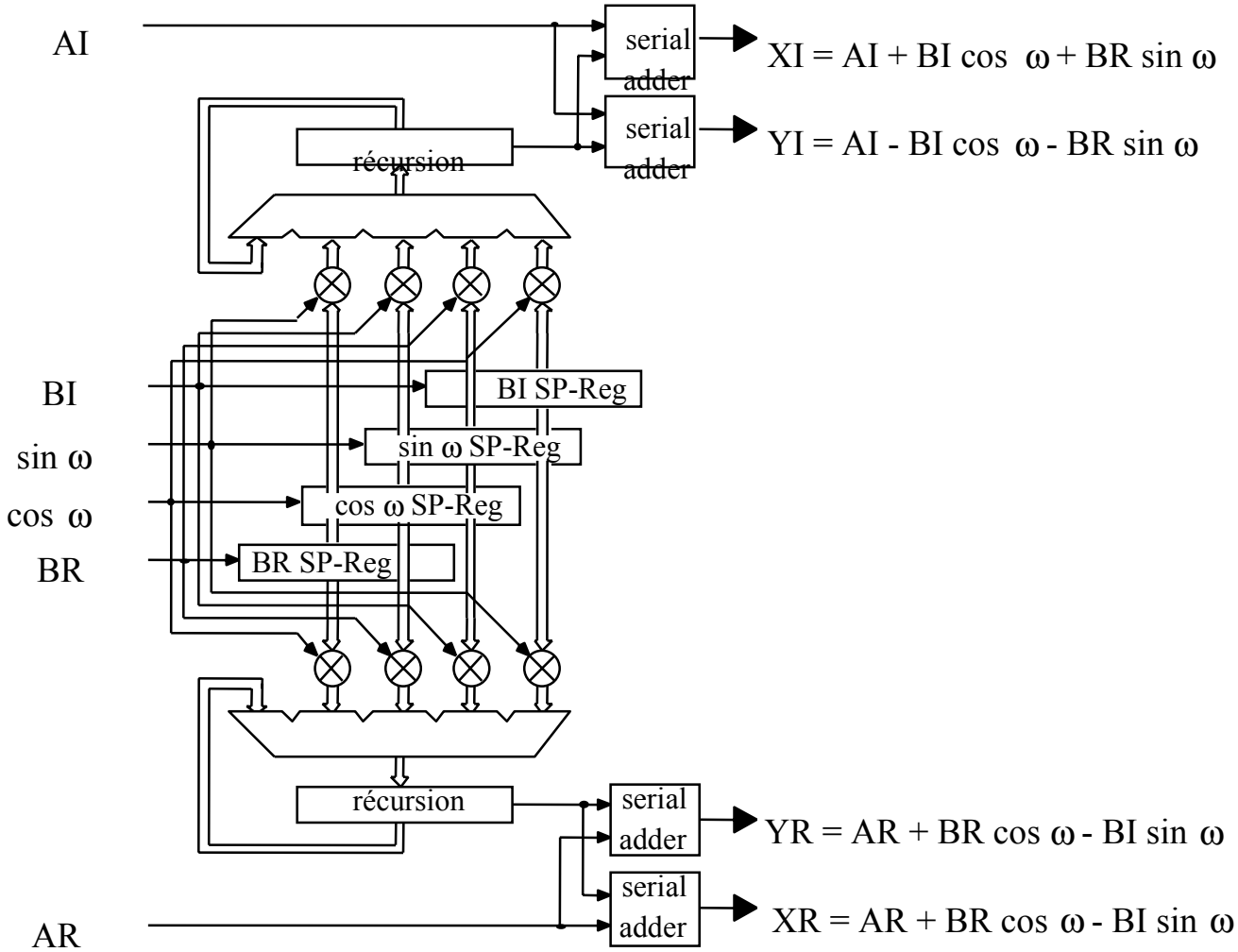


$$a_0 + x * (a_1 + x * (a_2 + x * (a_3 + x * (a_4 + x * a_5))))$$

latence 3 latence 3 latence 3 latence 3 latence 3

Latence indépendante du polynôme si $\exists\ k$ tq. $\forall\ n \geq k \quad \frac{a_{n+1}x}{a_n} \leq 2^{-3}$

Papillon FFT en ligne



Références

" *Design of an on-line Euclidean Processor* "
R. Bouraoui, A. Guyot and G. Walker
International Conference on Microelectronics 1992 (ICM'92), Monastir, Tunisia, Décembre 1992

" *On-line approximation of real functions using polynomials* "
A. Skaf, J.C. Bajard*, A. Guyot and J.M. Muller* (* LIP-ENS Lyon)
International Conference on Microelectronics 1992 (ICM'92), Monastir, Tunisia, Décembre 1992

" *Design of an on-line Euclidean processor* "
R. Bouraoui, A. Guyot and G. Walker
6 th International Conference on VLSI design, Bombay, India, Janvier 1993

" *On-line operator for Euclidean distance* "
R. Bouraoui, A. Guyot and G. Walker
EDAC-EUROASIC Conference, Paris, France, 22-25 Février 1993

" *A VLSI Circuit for on-line polynomial computing: application to exponential, trigonometric and hyperbolic functions* "
A. Skaf, J.C. Bajard*, A. Guyot and J.M. Muller* (* LIP-ENS Lyon)
VLSI 93, Grenoble, France, Septembre 1993

" *VLSI design of on-line add/multiply algorithms* "
A. Skaf and A. Guyot
ICCD 93, Cambridge, Massasusett, USA, Octobre 1993

" *A VLSI implementation of Parallel Fast Fourier Transform* "
A. Vacher, M. Benkhebbab, A. Guyot, T. Rousseau and A. Skaf
EDAC (European Design Automation Conference) , Paris, Février-Mars 1994

" *Error-Speed trade-off for FFT VLSI* "
A. Vacher and A. Guyot
26th IEEE Southeastern Symposium on System Theory, Athens, Ohio, Mars 1994

" *A VLSI implementation of Fast Fourier Transform for Large Sample Number* "
A. Vacher and A. Guyot
International Symposium on Signal Processing and Neural Network, Lille, Avril 1994

" *Design for testability of an on-line multiplier* "
H. Bederr*, M. Nicolaïdis* and A. Guyot (* Reliable Integrated System)
VLSI Test Symposium, Cherry Hill, New Jersey, Avril 1994

" *SAGA: The first general purpose on-line arithmetic co-processor* "
A. Skaf and A. Guyot
8 th International Conference on VLSI design (VLSI Design 95) New Delhi, India, Janvier 1995

" *On-line Hardware Implementation for Complex Exponential and Logarithm* "
A. Skaf, J.M. Muller* and A. Guyot (* LIP/ENSL Lyon)
20th European Solid-State CIRcuits Conference (ESSCIRC), Ulm, Germany, Septembre 1994