cours 3

étude de l'arithmétique des ordinateurs

cours 3

étude de l'arithmétique des ordinateurs

tous les traitements s'appuient sur des opérations arithmétiques

plan

- arithmétique
  - arithmétique binaire
  - arithmétique flottante
- circuits arithmétiques
  - addition
  - multiplication
  - division

note

symbole	signification
$+$	addition
$\times$	multiplication
$\vee$	ou logique
$\wedge$	et logique

arithmétique binaire

les opérations en binaire s'effectuent comme en base 10

arithmétique binaire

les opérations en binaire s'effectuent comme en base 10

- addition et soustraction
  - chiffre par chiffre
  - des poids faibes aux poids forts
  - en propageant la retenue

arithmétique binaire

les opérations en binaire s'effectuent comme en base 10

- addition et soustraction
  - chiffre par chiffre
  - des poids faibes aux poids forts
  - en propageant la retenue
- multiplication et division
  - par série d'additions ou de soustractions

addition

a	b	somme	retenue
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$\text{somme}(a,b) = a \oplus b$

$\text{retenue}(a,b) = ab$

soustraction

a	b	différence	retenue
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

différence(a,b) $=$ a $\oplus$ b

retenue(a,b) $= \overline{a}$b

exemple

	27	11011
+	22	10110
=	49	110001

exemple

	27	11011
+	22	10110
=	49	110001

	27	11011
-	22	10110
=	5	000101

débordement (overflow)

le résultat de l'opération n'est pas représentable

dans le système utilisé

débordement (overflow)

le résultat de l'opération n'est pas représentable

dans le système utilisé

$$\begin{array}{rr} & 11011 \\ + & 10110 \\ \hline = & 10001 \end{array}$$

débordement (overflow)

le résultat de l'opération n'est pas représentable

dans le système utilisé

$$\begin{array}{rr} & 11011 \\ + & 10110 \\ \hline = & 10001 \end{array}$$

addition en binaire naturel :

le débordement correspond à une retenue sortante à 1

addition avec le complément à deux

cas d'une addition *sans débordement*

5	00101	-5	11011	-5	11011	5	00101
9	01001	-9	10111	9	01001	-9	10111
14	01110	-14	10010	4	00100	-4	11100

comment est-ce possible ?

cas de $Z = X + Y$ avec $X < 0$ et $Y < 0$

en complément à 2 : $-\mid X \mid = 2^n - \mid X \mid$

$$\begin{aligned} Z &= (2^n - \mid X \mid) + (2^n - \mid Y \mid) \\ &= 2^{n+1} - (\mid X \mid + \mid Y \mid) \\ &= 2^{n+1} - \mid Z \mid \end{aligned}$$

la lecture sur n bits donne bien 2^n - $\mid Z \mid$

débordement

il y a débordement si

les opérandes X et Y sont de même signe, et
$| X | + | Y | \geq 2^{n-1}$

	13	01101	-13	10011
	9	01001	-9	10111
sur n bits	-10	10110	10	01010

débordement

il y a débordement si

les opérandes X et Y sont de même signe, et
$| X | + | Y | \geq 2^{n-1}$

	13	01101	-13	10011
	9	01001	-9	10111
sur n bits	-10	10110	10	01010
sur n+1 bits	22	010110	-22	101010

détection du débordement

- comparer le signe des opérandes et le signe du résultat
  - s'ils sont différents, il y a débordement
- comparer la retenue entrante dans le bit de poids fort avec la retenue sortante
  - si elles sont différentes, il y a débordement

multiplication

additions successives du multiplicande

avec lui même décalé

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

résultat codé sur $2n$ bits pour 2 opérandes sur n bits

multiplication en complément à 2

	11	01011	-5	1...11011	11	01011
×	13	01101	-3	1...11101	-3	1...11101
	143	010001111	15	000001111	-33	111011111

attention à la propagation du bit de signe !

division

soustractions successives

exemple $15_{10}/3_{10}$

1111	11
01	101
11	
0	

```
  1111
- 11      1
------
  1100
- 11      10
------
  1001
- 11      11
------
  0110
- 11      100
------
  0011
- 11      101
------
     0
```

division en complément à 2

exemple : $-7_{10} / -3_{10} = 2$, reste -1

$$\begin{array}{rcl} & 1001 & \\ + & 0011 & 1 \\ \hline & 1100 & \\ + & 0011 & 10 \\ \hline & 1111 & \end{array}$$

arithmétique flottante

utilisation de la représentation IEEE754

simple précision

– la mantisse $m \in [1.0_2, 10.0_2[$
– le 1 à gauche de la virgule n'est pas représenté
– mantisse sur 23 bits
– exposant codé avec un excès de 127

multiplication

multiplication de

$$x_1 = (-1)^{s_1} \times m_1 \times 2^{e_1}$$

par

$$x_2 = (-1)^{s_2} \times m_2 \times 2^{e_2}$$

$$x_1 \times x_2 = (-1)^{s_1} \times (-1)^{s_2} \times m_1 \times m_2 \times 2^{e_1+e_2}$$

principe général

1. calcul du signe $(s_1 \oplus s_2)$
2. multiplication entière des mantisses : $m_1 \times m_2$
3. mise de la mantisse à p bits
4. calcul de l'exposant : $e_1 + e_2$

   enlever une fois la valeur de l'excès

arrondi

soit p le nombre de bits utilisé pour chaque mantisse

arrondi

soit p le nombre de bits utilisé pour chaque mantisse

multiplier des mantisses donne un résultat sur $2p$ bits

arrondi

soit p le nombre de bits utilisé pour chaque mantisse

multiplier des mantisses donne un résultat sur $2p$ bits

on doit conserver une mantisse de p bits

arrondi

soit p le nombre de bits utilisé pour chaque mantisse

multiplier des mantisses donne un résultat sur $2p$ bits

on doit conserver une mantisse de p bits

il faut donc

– arrondir

arrondi

soit p le nombre de bits utilisé pour chaque mantisse

multiplier des mantisses donne un résultat sur $2p$ bits

on doit conserver une mantisse de p bits

il faut donc

– arrondir
– éventuellement ajuster l'exposant

arrondi

pour arrondir, on s'appuie sur :

– *le bit de garde* : $p + 1^{ieme}$ bit
– *le bit d'arrondi* : $p + 2^{ieme}$ bit
– *le bit persistant* : un ou logique des bits éliminés

arrondi

les p bits gardés sont fonction du bit de poids fort du résultat

- si le bit de poids fort est 0
  - le résultat obtenu est de la forme 01,...
  - il faut conserver le bit de garde dans le résultat

arrondi

- si le bit de poids fort est 1
  - le résultat est de la forme 10,... il faut ajuster l'exposant
  - le bit de garde devient le bit d'arrondi
  - l'ancien bit d'arrondi entre dans le calcul du bit persistant

il y a plusieurs manières d'arrondir...

les arrondis IEEE

le standard IEEE propose 4 approches d'arrondi

on appelle

– r le bit d'arrondi
– s le bit persistant
– p_0 bit de poid faible du résultat

arrondis IEEE

mode d'arrondi	résultat positif ou nul	résultat négatif
au plus près	résultat $+$ (r $\wedge p_0$) $\vee$ (r $\wedge$ s)	résultat $+$ (r $\wedge p_0$) $\vee$ (r $\wedge$ s)
vers - ∞	inchangé	résultat $+$ (r $\vee$ s)
vers $+\infty$	résultat $+$ (r $\vee$ s)	inchangé
0	inchangé	inchangé

par défaut : arrondi au plus près

consiste à arrondir à la plus proche valeur représentable

débordement

lorsque l'exposant du résultat est

– supérieur à la valeur maximum ou
– inférieur à la valeur minimum.

addition

nécessité de

– tenir compte des signes
– aligner les virgules

addition

$$-1{,}001 \times 2^{-2} + -1{,}111 \times 2^{0}$$

addition

$$-1{,}001 \times 2^{-2} + -1{,}111 \times 2^{0}$$

$$= -0{,}01001 \times 2^{0} + -1{,}111 \times 2^{0}$$

addition

$$-1{,}001 \times 2^{-2} + -1{,}111 \times 2^{0}$$

$$= -0{,}01001 \times 2^{0} + -1{,}111 \times 2^{0}$$

$$= 10{,}00101 \times 2^{0}$$

addition

$$-1{,}001 \times 2^{-2} + -1{,}111 \times 2^{0}$$

$$= -0{,}01001 \times 2^{0} + -1{,}111 \times 2^{0}$$

$$= 10{,}00101 \times 2^{0}$$

$$= 1{,}000101 \times 2^{1}$$

addition

$$-1{,}001 \times 2^{-2} + -1{,}111 \times 2^{0}$$

$$= -0{,}01001 \times 2^{0} + -1{,}111 \times 2^{0}$$

$$= 10{,}00101 \times 2^{0}$$

$$= 1{,}000101 \times 2^{1}$$

$= 1{,}000 \times 2^{1}$ sur 4 bits

addition

$$-1{,}001 \times 2^{-2} + -1{,}111 \times 2^{0}$$

$$= -0{,}01001 \times 2^{0} + -1{,}111 \times 2^{0}$$

$$= 10{,}00101 \times 2^{0}$$

$$= 1{,}000101 \times 2^{1}$$

$= 1{,}000 \times 2^{1}$ sur 4 bits

$= 1{,}001 \times 2^{1}$ sur 4 bits après arrondi

division

$x_1 = s_1 \times 2^{e_1}$ et $x_2 = s_2 \times 2^{e_2}$

$x_1/x_2 = s_1/s_2 \times 2^{e_1 - e_2}$

diviser deux mantisses sur p bits :

- calcul du résultat sur p bits
- calcul de deux bits de résultat supplémentaires
  - le bit de garde
  - le bit d'arrondi
- le reste donne la valeur du bit persistant

division

$1{,}000 \times 2^3 / 1{,}100 \times 2^1$

```
1,  0  0  0                    |  1,  1  0  0
1,  0  0  0  0                 |  0,  1  0  1  0  1
-   1  1  0  0                 |              g  a
    1  0  0  0  0              |
    -  1  1  0  0              |
       1  0  0  0  0           |
       -  1  1  0  0           |
          1  0  0              |
          p  p  p              |
```

division

résulat $0{,}101 \times 2^2$

– bit de garde g= 0,
– bit d'arrondi a=1
– bit persistant p=100

résulat après normalisation : $1{,}010 \times 2^1$

résultat après arrondi : $1{,}011 \times 2^1$

circuits arithmétiques

quelques circuits typiques de mise en oeuvre de opérations arithmétiques

addition

demi additionneur

– somme $= a \oplus b$

– retenue $= ab$

additionneur complet (full adder)

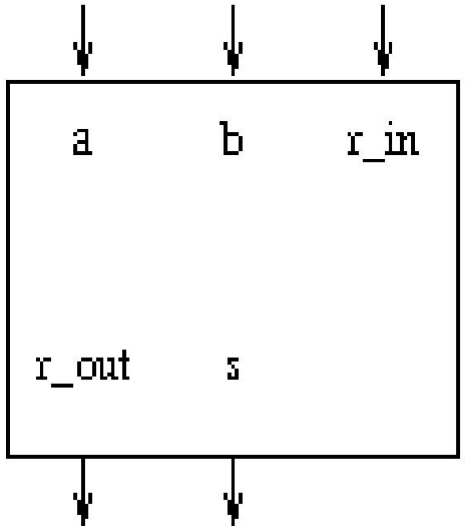
– prise en compte de la retenue entrante

– génération de la retenue sortante

addition

r_{in}	a	b	somme	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

addition



additionneurs n bits

l'amélioration d'un additionneur porte sur la propagation de la retenue

– additionneur à propagation de retenue
– additionneur à anticipation de retenue
– additionneur à saut de retenue
– additionneur à sélection de retenue

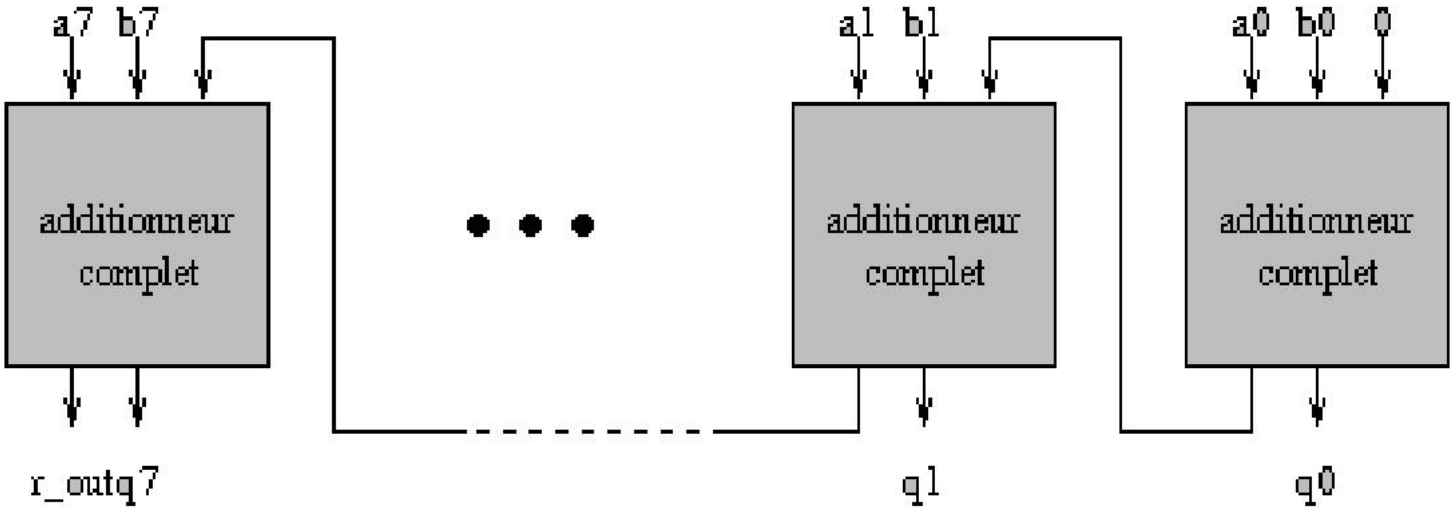
Additionneur à propagation simple de retenue (CPA : carry propagate adder)

utilisation de n additionneurs complets

la retenue sortant de l'étage d'addition i est la retenue entrante dans l'étage $i+1$

– la retenue entrante au premier étage est à 0
– la retenue sortante du dernier étage sert d'indicateur de débordement

CPA


a7 b7
a1 b1
a0 b0 0
additionneur complet
additionneur complet
additionneur complet
r_outq7
q1
q0

CPA

inconvénient majeur : le temps nécessaire à la propagation de la retenue

ce temps est proportionnel au nombre n d'étage

on dira que la complexité en temps d'un additionneur n bits est en $O(n)$

additionneur à retenue anticipée (CLA : carry look-ahead adder)

chaque étage d'addition anticipe la retenue avant de calculer la somme

une retenue sortante à un étage d'addition est due à

– une *génération* de retenue à cet étage, ou
– une *propagation* de retenue par cet étage

CLA

à l'étage i :

$$r_{i+1} = (a_i \wedge b_i) \vee (a_i \wedge r_i) \vee (b_i \wedge r_i)$$

$$= (a_i \wedge b_i) \vee (a_i \vee b_i) \wedge r_i$$

on appelle

– $g_i = (a_i \wedge b_i)$ *générateur* de retenue

– $p_i = (a_i \vee b_i)$ *propagateur* de retenue

CLA

Donc

$$r_{i+1} = g_i \vee (p_i \wedge r_i)$$

en développant les r_i :

$$r_{i+1} = g_i \vee (p_i \wedge (g_{i-1} \vee (p_{i-1} \wedge (\ldots g_0 \vee (p_0 \wedge r_0) \ldots)$$

CLA

$$
\begin{aligned}
r_{i+1} = g_i \quad & \vee \quad (p_i \wedge g_{i-1}) \\
& \vee \quad (p_i \wedge p_{i-1} \wedge g_{i-2}) \\
& \vee \quad \ldots \\
& \vee \quad (p_i \wedge p_{i-1} \wedge \ldots \wedge p_1 \wedge g_0) \\
& \vee \quad (p_i \wedge p_{i-1} \wedge \ldots \wedge p_1 \wedge p_0 \wedge r_0)
\end{aligned}
$$

chaque retenue peut être calculée en fonction de r_0 et des a_i,b_i des étages précédents

mise en oeuvre du CLA

calcul de

– la propagation
– la génération

à l'étage i en fonction de

– la propagation
– la génération

aux étages précédents

propagation

propagation à l'étage j si dans les étages i à j il y a toujours eu propagation

$$P_{j,i} = p_j \wedge p_{j-1} \cdots p_{i+1} \wedge p_i$$

génération

génération à l'étage j si

– c'est l'étage j lui même qui génère, ou

– si il y a eu une génération propagée dans un des étages précédents

$$G_{j,i} = g_j \vee (p_j \wedge g_{j-1}) \vee (p_j \wedge p_{j-1} \wedge g_{j-2}) \vee \ldots$$

$$\vee (p_j \wedge p_{j-1} \ldots p_{i+1} \wedge g_i)$$

expression de la retenue

la retenue à l'étage j peut s'exprimer en fonction de la retenue à l'étage i,$i < j$:

$$r_{j+1} = G_{j,i} \vee P_{j,i} \wedge r_i$$

circuit de propagation-génération

circuit générant $P_{1,0}$,$G_{1,0}$ et la retenue r_1 en fonction des couples (p_0,g_0), (p_1,g_1) et de la retenue entrante r_0

$$G_{1,0} = g_1 \vee p_1 \wedge g_0$$

$$P_{1,0} = p_1 \wedge p_0$$

$$r_1 = g_0 \vee p_0 \wedge r_0$$

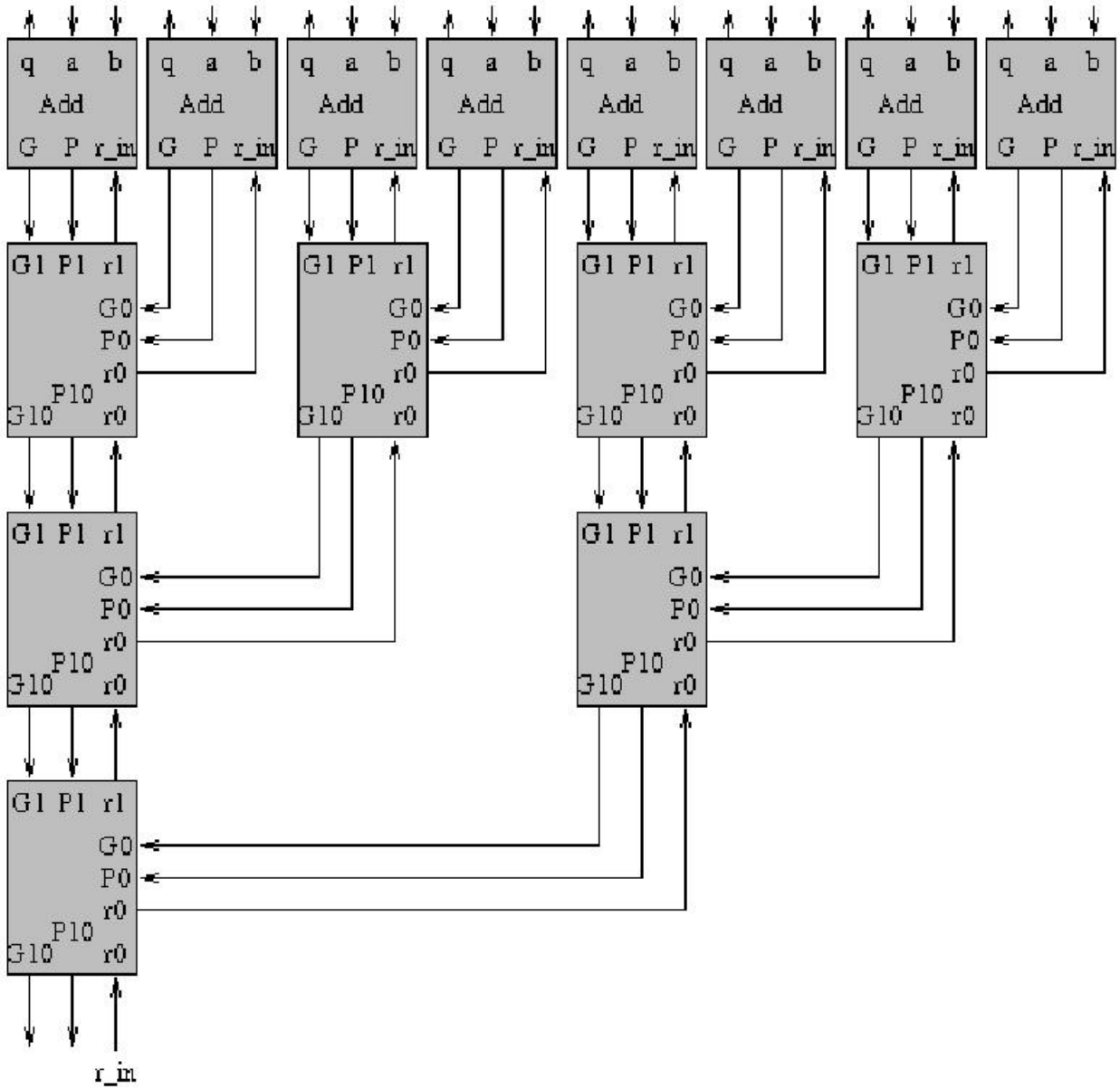
réalisation de l'additionneur

n additionneurs un bit pour calculer

- g_0,p_0,g_1,p_1,g_2,p_2,...
- les sommes

circuits de propagation-génération pour calculer

- $G_{1,0}$,$P_{1,0}$,$G_{3,2}$,$P_{3,2}$, ...
- $G_{3,0}$,$P_{3,0}$,$G_{7,4}$,$P_{7,4}$, ...
- ...


q a b
Add
G P r_in
G1 P1 r1
G0
P0
r0
P10
G10 r0
r_in

CLA

les générateurs de retenues sont organisés en arbre

chaque retenue est anticipée avant d'additionner

le CLA fonctionne en O(log(n))

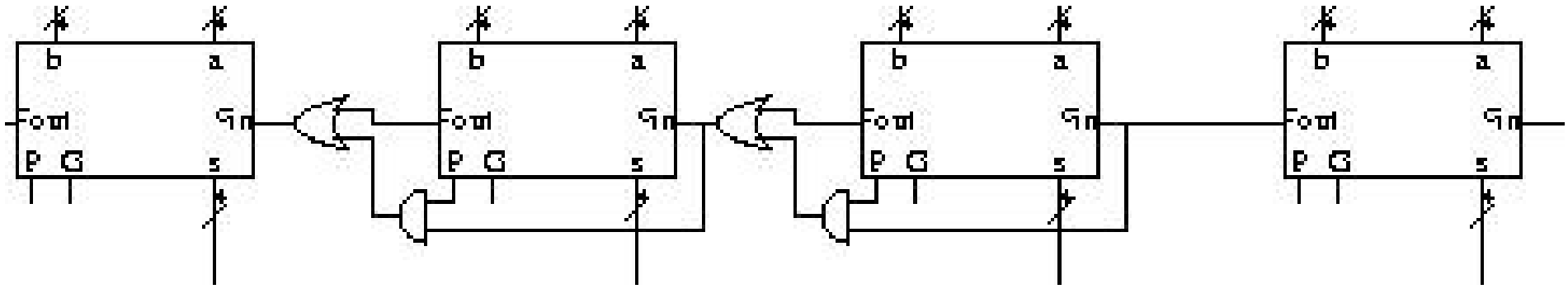
Additionneur à saut de retenue (Carry-Skip Adder)

étage d'addition i:

– s'il y a propagation de retenue, alors
  – la retenue sortante = la retenue entrante,
– sinon la retenue sortante = la retenue générée

le calcul du signal de propagation est plus rapide que l'addition

réalisation



CSA

avantages :

– plus rapide que le CPA
– plus simple que le CLA

Additionneur à sélection de retenue (Carry Select Adder)

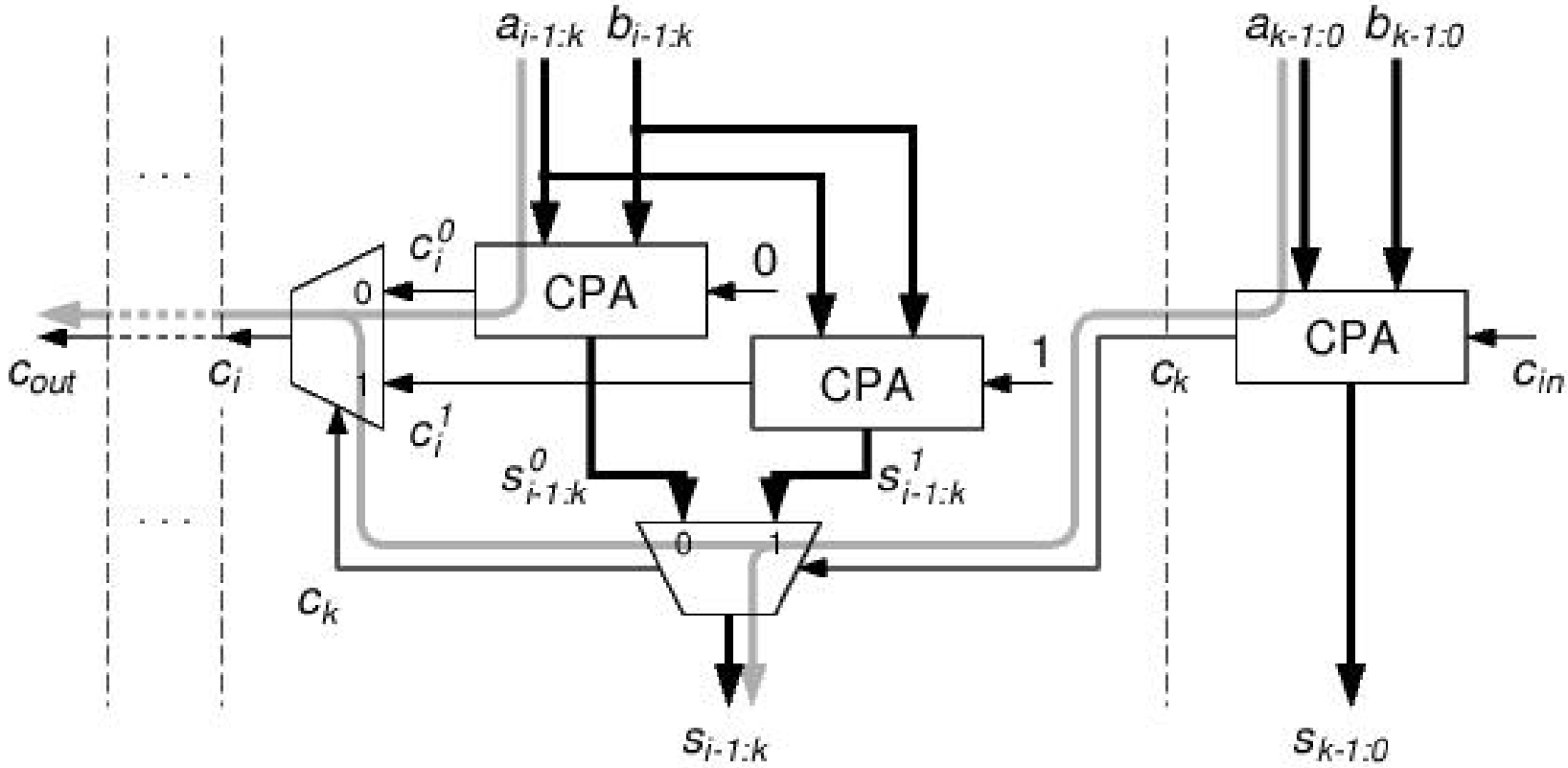
deux additionneurs fonctionnant en parallèle

– avec une retenue entrante à 0

– avec une retenue entrante à 1

lorsque la retenue est connue, une simple sélection permet d'obtenir le résultat

réalisation



multiplication

principe

– génération de produits partiels
– addition des produits partiels

l'amélioration d'un multiplieur passe par

– la réduction du nombre de produits partiels
– l'accélération de l'addition des produits partiels

multiplication séquentielle

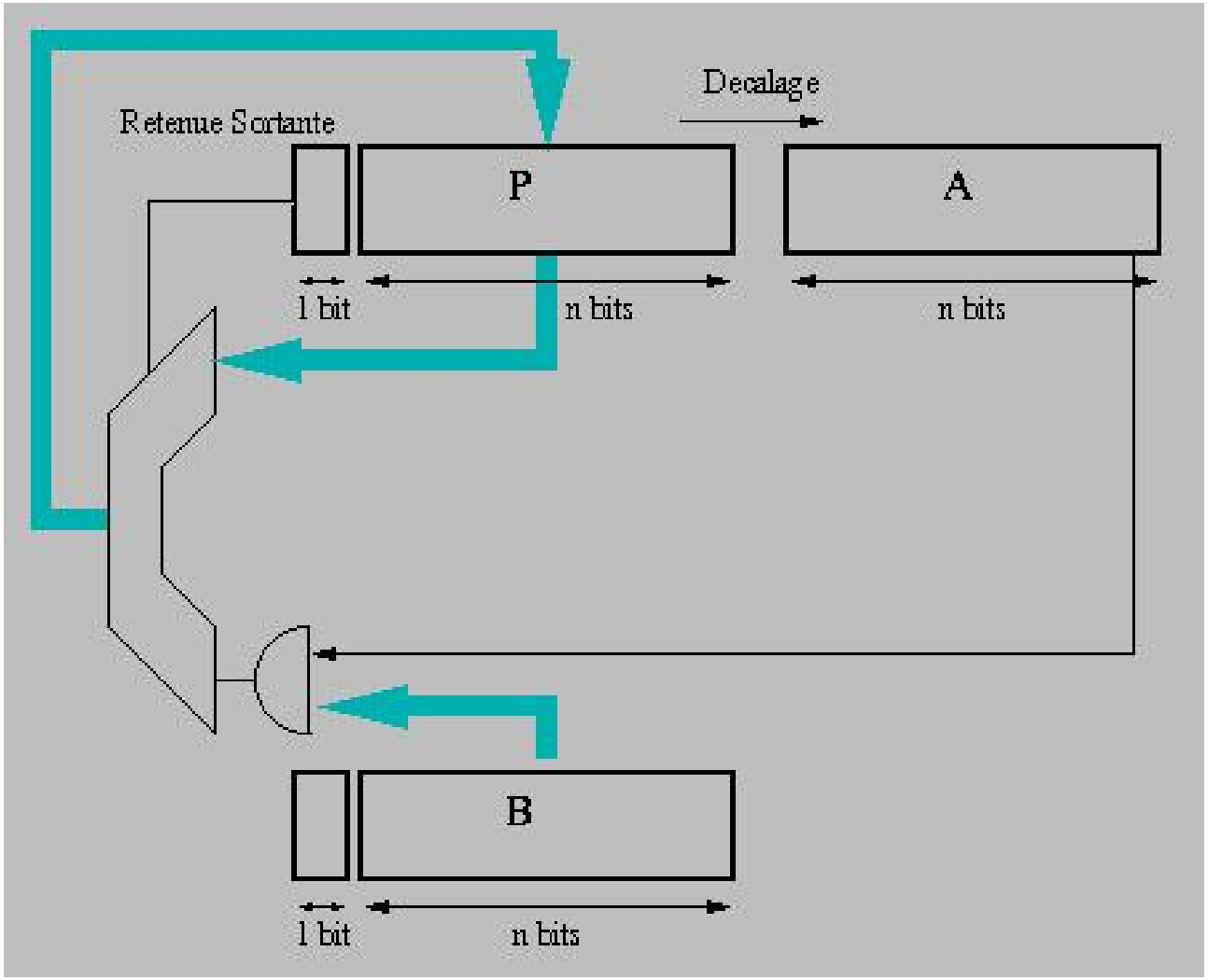
$A \times B$ sur n bits non signée

– un registre A contient $A = a_{n-1} \ldots a_1 a_0$
– un registre B contient $B = b_{n-1} \ldots b_1 b_0$
– un registre P contient les produits partiels

le résultat

– P contiendra les bits de poids fort
– A contiendra les bits de poids faibles

réalisation



algorithme

1. $P \leftarrow 0$
2. faire n fois
   (a) si le chiffre de poid faible de A est 1 alors
   $$P \leftarrow P + B$$
   (b) décalage des registres P et A vers la droite
   - le bit de poids fort de P reçoit la retenue sortante
   - le bit de poids faible de P est transféré dans le bit de poids fort de A
   - l'ancien bit de poids faible de A est perdu

multiplication à conservation de retenue

ne plus propager la retenue sur chaque étage d'une addition

la conserver pour l'addition suivante

revient à faire uniquement des demi-additions

exemple

				1	1	0	1
×				1	0	1	1
				1	1	0	1
			1	1	0	1	
			1	0	1	1	1
			1	0	0	0	
		1	0	0	1	1	1
	1	1	0	1			
	1	0	0	1	1	1	1
	1	0	0	0			
1	0	0	0	1	1	1	1

recodage de Booth

une séquence de j bits à $1 = j$ additions

comment réduire le nombre d'additions ?

recodage de Booth

une séquence de j bits à 1 $= j$ additions

comment réduire le nombre d'additions?

$$2^{i+j} - 2^i = 2^{i+j-1} + 2^{i+j-2} + \ldots + 2^i$$

recodage de Booth

une séquence de j bits à 1 $= j$ additions

comment réduire le nombre d'additions?

$2^{i+j} - 2^i = 2^{i+j-1} + 2^{i+j-2} + \ldots + 2^i$

une séquence de j additions peut être remplacée par

– une addition pour le rang $i + j$ et
– une soustraction pour le rang i

exemple

000111110011100 se recode en $00100001\overline{1}0100\overline{1}00$

– 1 indique une addition

– $\overline{1}$ indique une soustraction

algorithme

$P \leftarrow 0$ // $P = A \times B$, $A = a_{n-1} \ldots a_0$

$A_{-1} \leftarrow 0$ // contient a_{i-1} à l'étape i

faire n fois

  si $A_0 A_{-1} = 01$ alors

    $P \leftarrow P + B$

  sinon si $A_0 A_{-1} = 10$ alors

    $P \leftarrow P - B$

  décaler P,A

multiplication en réseau (parallel multiplier)

utiliser un circuit pour chaque élément $x_i y_j$

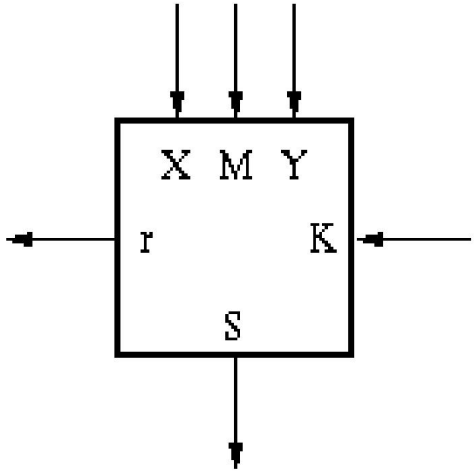
				x_3	x_2	x_1	x_0
				y_3	y_2	y_1	y_0
				x_3y_0	x_2y_0	x_1y_0	x_0y_0
			x_3y_1	x_2y_1	x_1y_1	x_0y_1	
		x_3y_2	x_2y_2	x_1y_2	x_0y_2		
	x_3y_3	x_2y_3	x_1y_3	x_0y_3			
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

multiplieur réseau

Chaque élément $x_i y_j$:

– multiplie x_i par y_j $(x_i \wedge y_j)$
– additionne la somme partielle provenant de $x_{i+1} y_{j-1}$
– tient compte de la retenue provenant de $x_{i-1} y_j$
– envoie la retenue à $x_{i+1} y_j$
– envoie la somme à $x_{i-1} y_{j+1}$

circuit de base

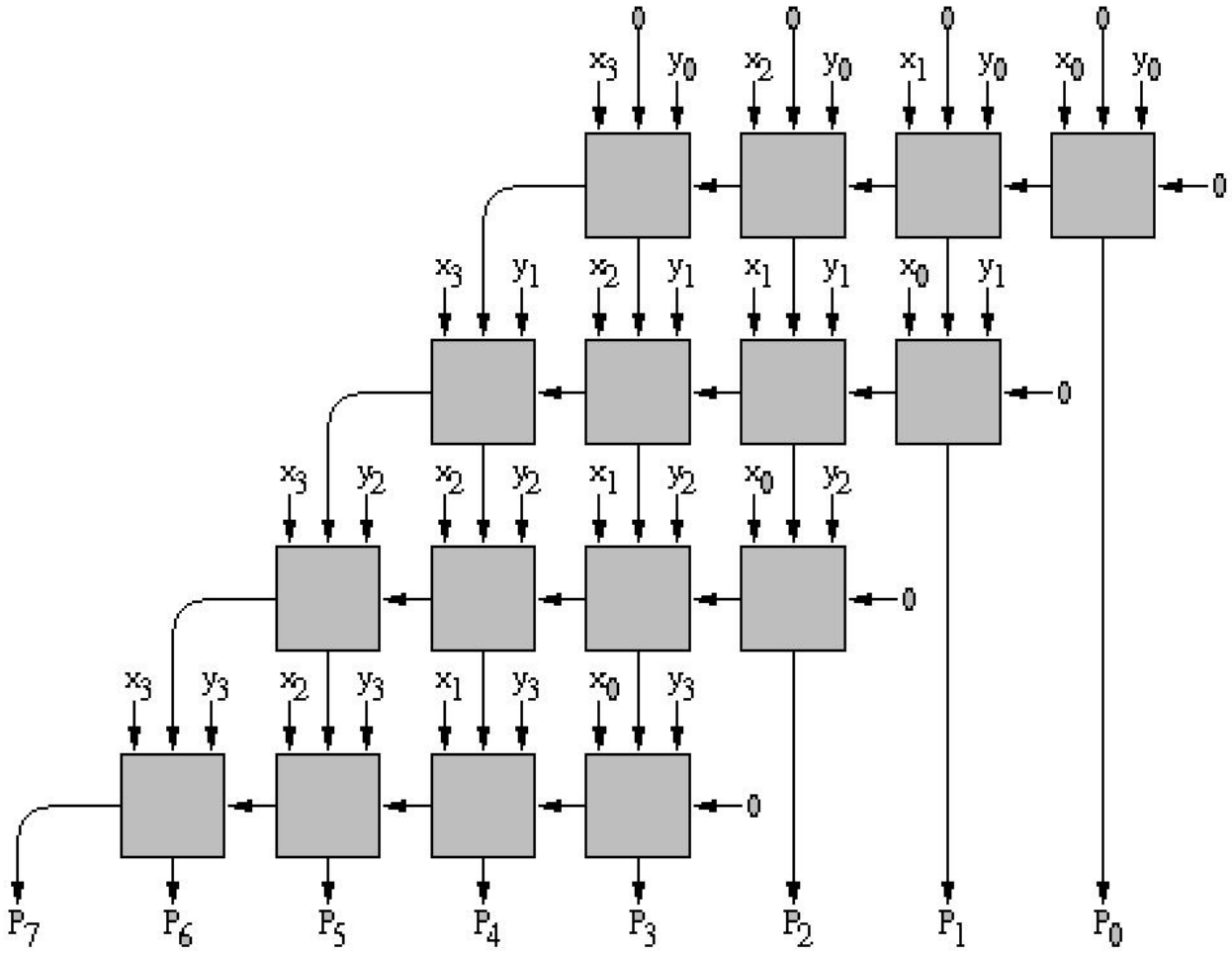


$(r,S) = X \times Y + K + M$

S est la somme

r est la retenue sortante

réalisation



multiplieur réseau

intérêt :

principe du pipeline

les étages disponibles servent à la multiplication suivante

le calcul de séquences de multiplications est accéléré

division

basée sur l'addition et la soustraction

l'amélioration des diviseurs porte sur la réduction du nombre d'opérations intermédiaires

division

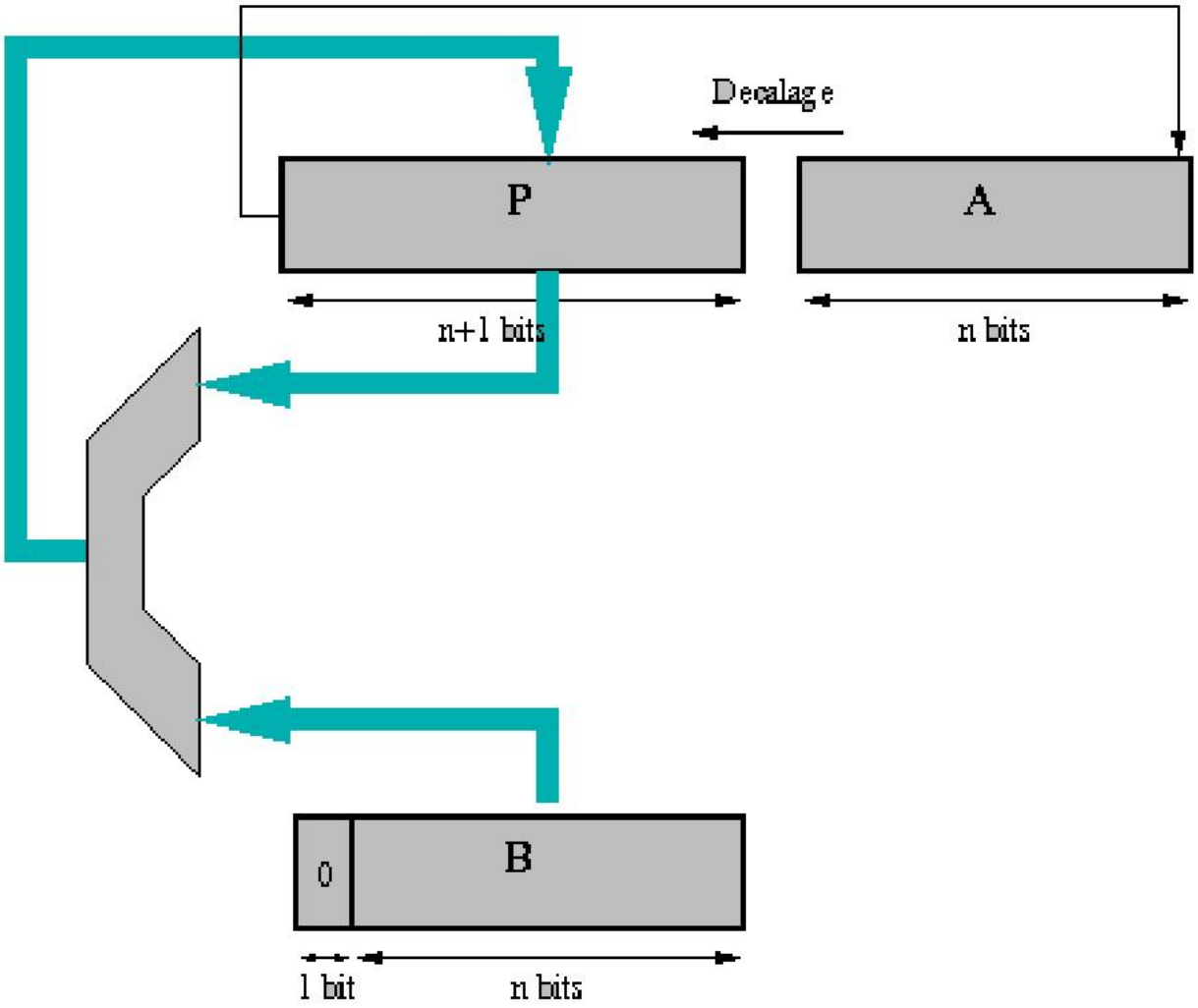
A/B sur n bits non signée

– un registre A contenant le dividende
– un registre B contenant le diviseur
– un registre intermédiaire P

le résultat

– le registre A contient le quotient
– le registre P contient le reste

réalisation



division avec restauration

1. $P \leftarrow 0$
2. faire n fois
   (a) décalage des registres P, A d'un bit vers la gauche :
   $P_0 \leftarrow A_{n-1}$
   (b) $P \leftarrow P - B$
   (c) si $P < 0$ alors
   le bit de poids faible de A est mis à 0
   $P \leftarrow P + B$ //étape de restauration
   sinon
   le bit de poids faible de A est mis à 1

exemple sur 4 bits

P	A	B
0000	0111	0011
0000	1110	décalage
1101	1110	soustraction
0000	1110	restauration
0001	1100	décalage
1110	1100	soustraction
0001	1100	restauration
0011	1000	décalage
0000	1001	soustraction
0001	0010	décalage
1110	0010	soustraction
0001	0010	restauration

division sans restauration

changement de l'itération :

1. si $P < 0$

   décalage des registres P, A d'un bit vers la gauche :

   $P_0 \leftarrow A_{n-1}$

   $P \leftarrow P + B$

   sinon

   décalage des registres P, A d'un bit vers la gauche :

   $P_0 \leftarrow A_{n-1}$

   $P \leftarrow P - B$

division sans restauration

suite :

2. si $P < 0$

   le bit de poids faible de A est mis à 0

   sinon

   le bit de poids faible de A est mis à 1

restaurer si $P < 0$ en fin de division

division avec le complément à 2

1. $B \leftarrow$ diviseur
   $(P,A) \leftarrow$ dividende avec extension du signe
2. décaler P,A d'un bit
3. si B et A sont de même signe
   $P \leftarrow P - B$
   sinon
   $P \leftarrow P + B$
4. si P a changé de signe et $(P,A) \neq 0$
   restauration
   $A_0 \leftarrow 0$
   sinon
   $A_0 \leftarrow 1$
5. répéter n fois depuis 2.
6. complémenter A si A et B étaient de signe différent

exemple : -7/3 sur 4 bits

P	A	B
1111	1001	0011
1111	001	décalage
0010	001	addition
1111	0010	restauration
1110	010	décalage
0001	010	addition
1110	0100	restauration
1100	100	décalage
1111	1001	addition
1111	001	décalage
0010	001	addition
1111	0010	restauration
1111	1110	complémentation

Division SRT

proposé indépendament par Sweeney, Robertson, Tocher

réduit les calculs intermédiaires

Division SRT

proposé indépendament par Sweeney, Robertson, Tocher

réduit les calculs intermédiaires

	1	9	5	2	1	7	
-	1	7			1	2	$\overline{6}$
		2	5	2			
	-	3	4				
		-	8	8			
	+	1	0	2			
			1	4			

$$(120 - 6) \times 17 + 14 = 1952$$

principe

si on est sûr que le reste partiel $<$ diviseur

bit de quotient à 0

si le reste partiel > 0

bit de quotient à 1

soustraction

si le reste partiel < 0

bit de quotient à $\overline{1}$

addition

algorithme

les q_i sont les bits du quotient

injectés dans les bits de poids faible de A

le registre P comporte $n+1$ bits

1. si il y a k 0 dans les bits de poids fort de B

   décaler de k positions vers la gauche les registres P,A et B

2. pour i de 0 à n-1
   si les trois bits de poids fort de P sont égaux
      décaler P,A d'une position à gauche
      $P_0 \leftarrow A_{n-1}$
      $q_i \leftarrow 0$
   sinon
      si $P < 0$
         décaler P,A d'une position vers la gauche
         $P_0 \leftarrow A_{n-1}$
         $q_i \leftarrow \overline{1}$
         $P \leftarrow P + B$
      sinon
         décaler P,A d'une position vers la gauche
         $P_0 \leftarrow A_{n-1}$
         $q_i \leftarrow 1$
         $P \leftarrow P - B$

3. si $P < 0$ (le reste final est négatif)

   $P \leftarrow P + B$

   $A \leftarrow A - 1$

4. décaler le reste de k positions vers la droite

ne fonctionne que sur des non signés

explication

si P contient 3 bits de poids fort identiques

- $P = 0{,}00...$ ou $1{,}11...$ donc $P \in]-1/4, 1/4[$
- diviseur $= 0{,}1...$
- $|P| < |B|$
- donc $q = 0$

sinon $|P| > 1/4$

- si $P < 0$ $q = \overline{1}$
- sinon $q = 1$

exemple 8/3 sur 4 bits

P	A	B	
00000	1000	0011	
00010	0000	1100	1. décalage
00100	0000	1100	2.1 $q_0 = 0$
01000	0000	1100	2.2 décalage
11100	0001	1100	2.2 $q_1 = 1$, soustraction
11000	0010	1100	2.3 $q_2 = 0$
10000	0100	1100	2.4 décalage
11100	010$\overline{1}$	1100	2.4 $q_3 = \overline{1}$, addition
01000	0010	1100	3. P+B et A-1
00010	0010	1100	4. décalage