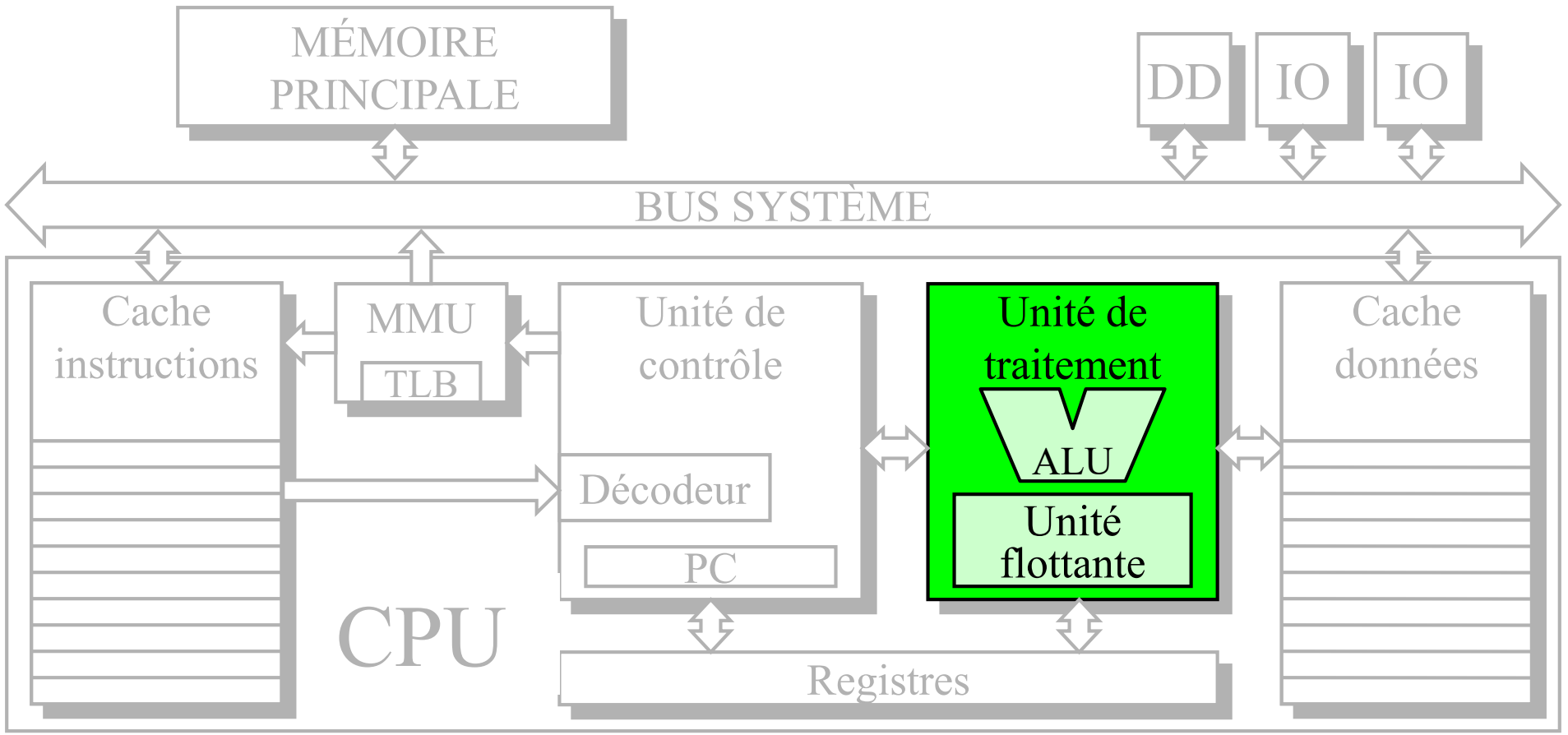
Arithmétique I

- **Opérations arithmétiques – Addition et Soustraction**
- **Nombres entiers et flottants**



Rappel sur l'écriture des entiers en base 2

- **Entiers positifs**

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

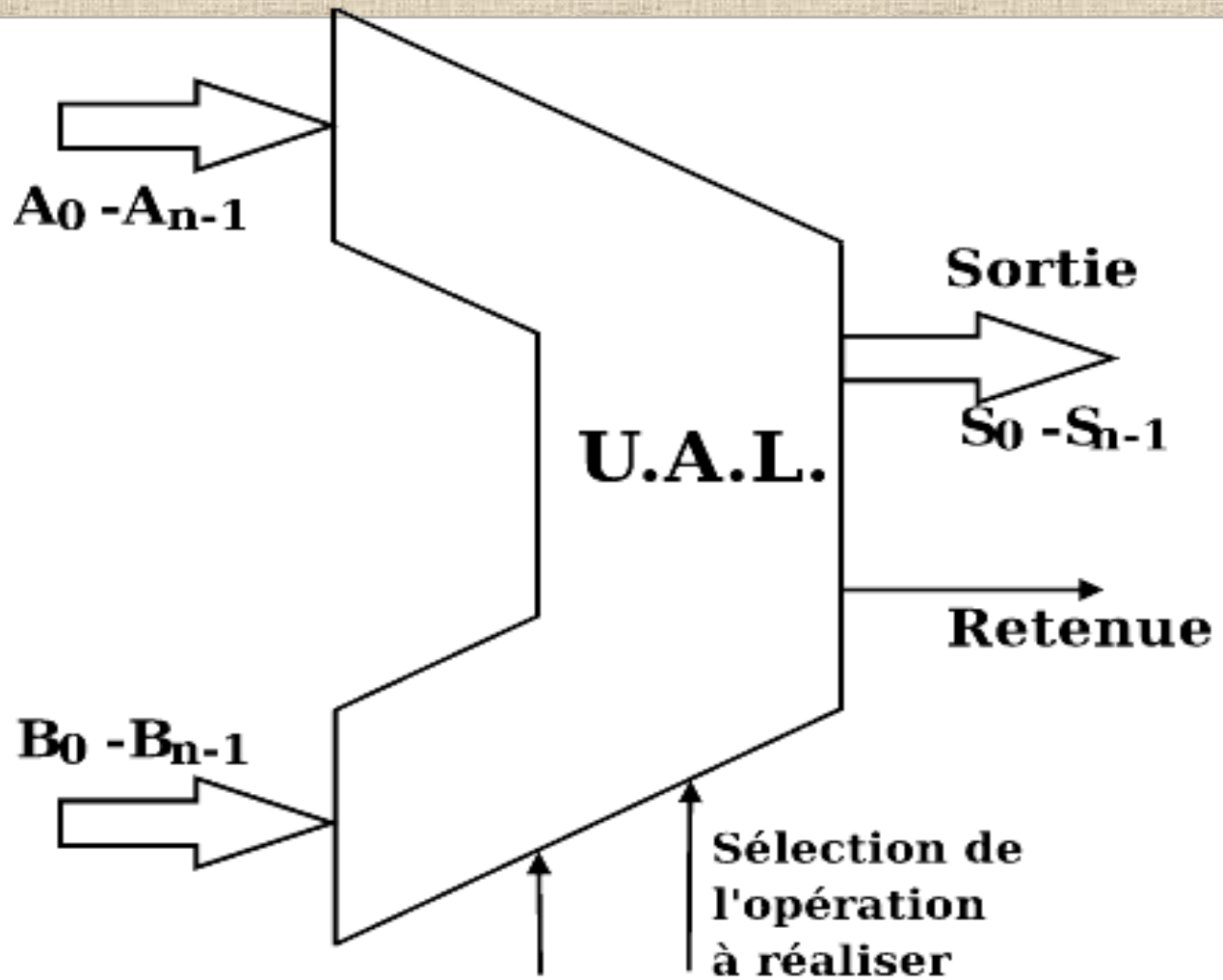
- **Entiers relatifs (complément à 2)**

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

Unité Arithmétique & Logique

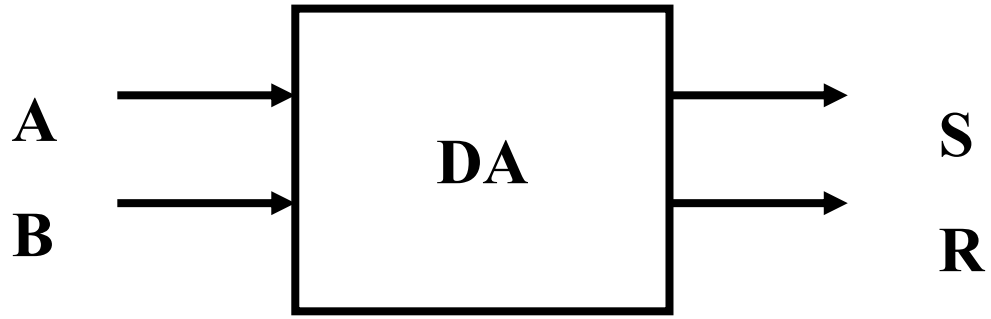
- **Cœur du microprocesseur**
- **Opérations simples sur mots binaires**
  - **Inversion (NON)**
  - **ET/OU**
  - **Addition**
- **Les UAL modernes font bien plus…**

L’UAL (2)


A0 -An-1
B0 -Bn-1
U.A.L.
Sortie
S0 -Sn-1
Retenue
Sélection de
l'opération
à réaliser

Demi-Additionneur

- **Le demi additionneur est un circuit combinatoire qui permet de réaliser la somme arithmétique de deux nombres A et B chacun sur un bit.**
- **A la sotie on va avoir la somme S et la retenu R (Carry).**



Pour trouver la structure (le schéma) de ce circuit on doit en premier dresser sa table de vérité

Demi-Additionneur

- **En binaire l'addition sur un seul bit se fait de la manière suivante:**

$$\begin{cases} 0+0 = 00 \\ 0+1 = 01 \\ 1+0 = 01 \\ 1+1 = 10 \end{cases}$$

•La table de vérité associée :

A	B		R	S
0	0		0	0
0	1		0	1
1	0		0	1
1	1		1	0

De la table de vérité on trouve :

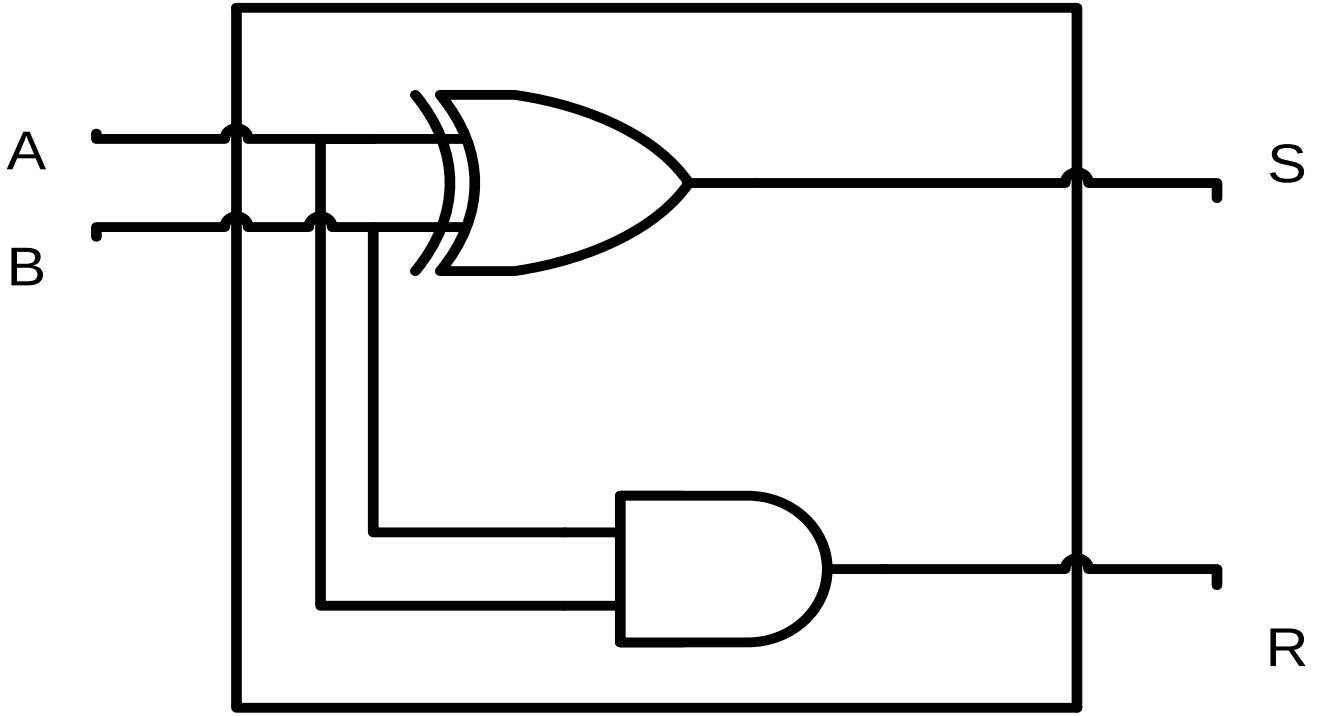
$$R = A.B$$

$$S = \overline{A}.B + A.\overline{B} = A \oplus B$$

Demi-Additionneur

$$R = A.B$$

$$S = A \oplus B$$



Additionneur complet

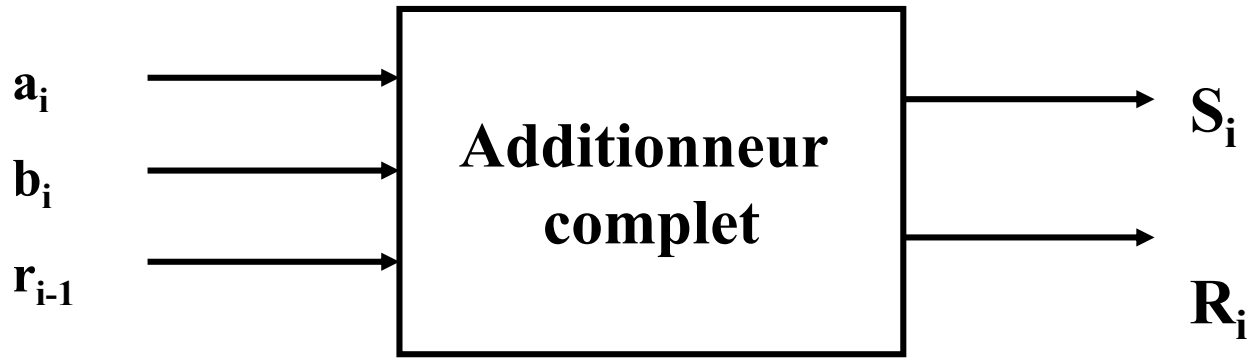
- **En binaire lorsque on fait une addition il faut tenir en compte de la retenue entrante.**

r_4	r_3	r_2	r_1	$r_0 = 0$
	a_4	a_3	a_2	a_1
+	b_4	b_3	b_2	b_1
r_4	s_4	s_3	s_2	s_1

	r_{i-1}	
	a_i	
+	b_i	
r_i	s_i	

Additionneur complet 1 bit

- **L'additionneur complet un bit possède 3 entrées :**
  - **a_i : le premier nombre sur un bit.**
  - **b_i : le deuxième nombre sur un bit.**
  - **r_{i-1} : le retenue entrante sur un bit.**
- **Il possède deux sorties :**
  - **S_i : la somme**
  - **R_i la retenue sortante**



Additionneur complet 1 bit

Table de vérité d'un additionneur complet sur 1 bit

a_i	b_i	r_{i-1}		r_i	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

$$S_i = \overline{A}_i.\overline{B}_i.R_{i-1} + \overline{A}_i.B_i.\overline{R}_{i-1} + A_i.\overline{B}_i.\overline{R}_{i-1} + A_i.B_i.R_{i-1}$$

$$R_i = \overline{A}_i B_i R_{i-1} + A_i \overline{B}_i R_{i-1} + A_i B_i \overline{R}_{i-1} + A_i B_i R_{i-1}$$

Additionneur complet 1 bit

Si on veut simplifier les équations on obtient :

$$S_i = \overline{A_i}.\overline{B_i}.R_{i-1} + \overline{A_i}.B_i.\overline{R}_{i-1} + A_i.\overline{B}_i.\overline{R}_{i-1} + A_i.B_i.R_{i-1}$$

$$S_i = \overline{A_i}.(\overline{B_i}.R_{i-1} + B_i.\overline{R}_{i-1}) + A_i.(\overline{B}_i.\overline{R}_{i-1} + B_i.R_{i-1})$$

$$S_i = \overline{A_i}(B_i \oplus R_{i-1}) + A_i.(\overline{B_i \oplus R_{i-1}})$$

$$S_i = A_i \oplus B_i \oplus R_{i-1}$$

$$R_i = \overline{A_i} B_i R_{i-1} + A_i \overline{B}_i R_{i-1} + A_i B_i \overline{R}_{i-1} + A_i B_i R_{i-1}$$

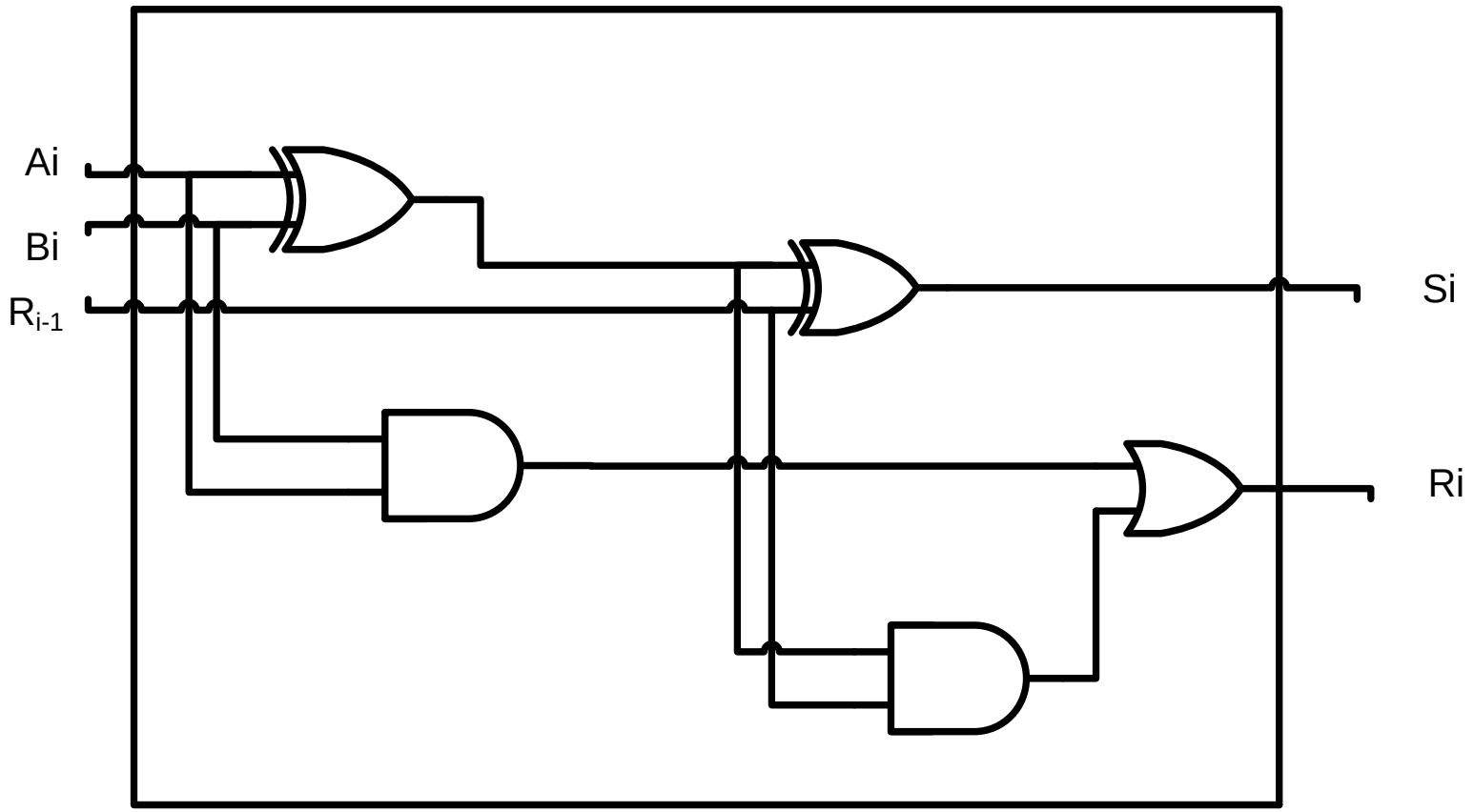
$$R_i = R_{i-1}.(\overline{A_i}.B_i + A_i.\overline{B}_i) + A_i B_i (\overline{R}_{i-1} +_i R_{i-1})$$

$$R_i = R_{i-1}.(A_i \oplus B_i) + A_i B_i$$

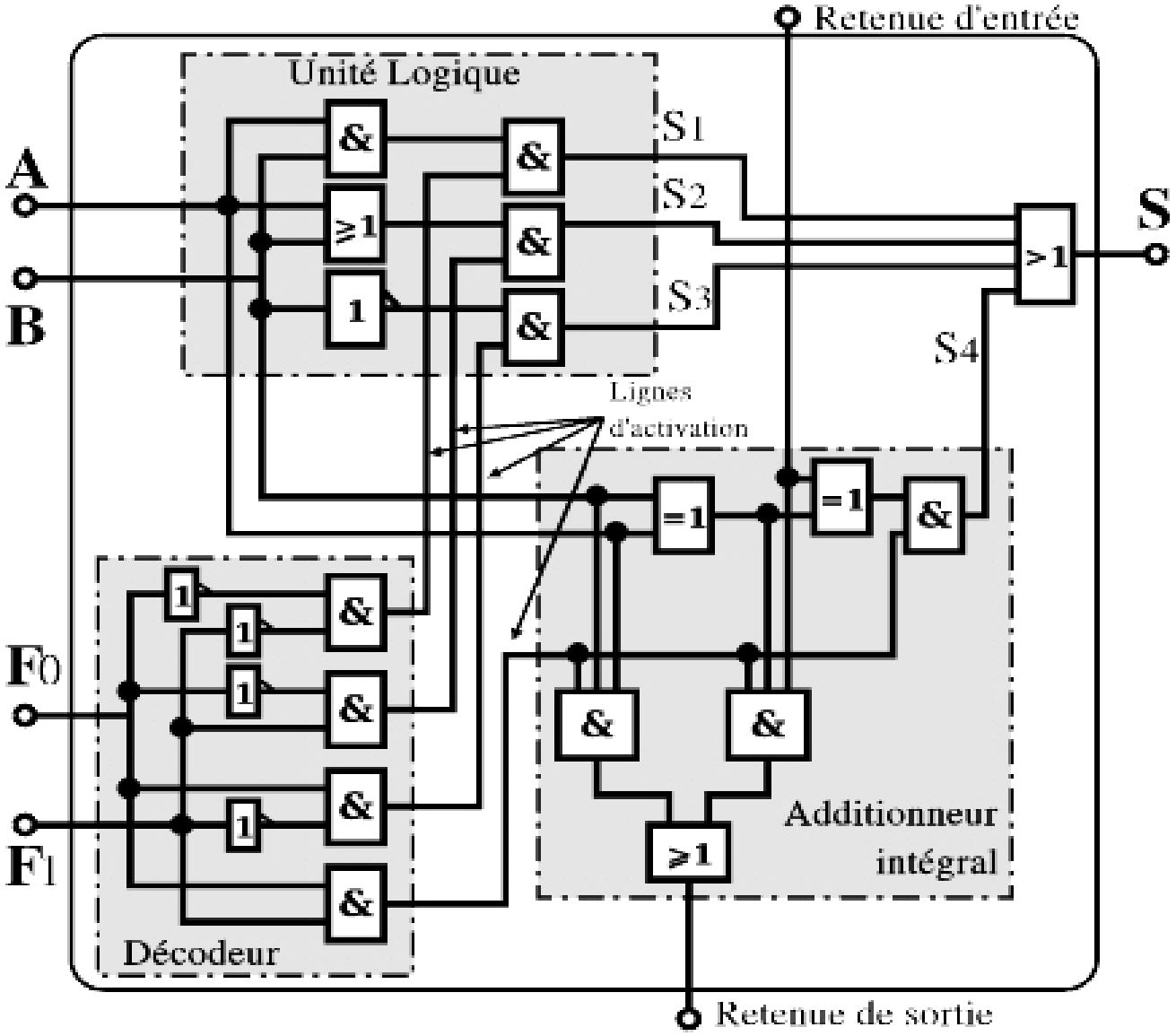
Schéma d'un additionneur complet

$$R_i = A_i.B_i + R_{i-1}.(B_i \oplus A_i)$$

$$S_i = A_i \oplus B_i \oplus R_{i-1}$$



Une UAL 1 bit


Retenue d'entrée
Unité Logique
A
B
S1
S2
S3
S4
S
Lignes
d'activation
F0
F1
Décodeur
Additionneur
intégral
Retenue de sortie

En utilisant des Demi Additionneurs

$$R_i = A_i.B_i + R_{i-1}.(B_i \oplus A_i)$$

$$S_i = A_i \oplus B_i \oplus R_{i-1}$$

Si on pose $X = A_i \oplus B_i$ et $Y = A_i B_i$

On obtient :

$$R_i = Y + R_{i-1}.X$$

$$S_i = X \oplus R_{i-1}$$

et si on pose $Z = X \oplus R_{i-1}$ et $T = R_{i-1}.X$

On obtient :

$$R_i = Y + T$$

$$S_i = Z$$

• On remarque que X et Y sont les sorties d'un demi additionneur ayant comme entrées A et B

• On remarque que Z et T sont les sorties d'un demi additionneur ayant comme entrées X et R_{i-1}

En utilisant des Demi Additionneurs

$$X = A_i \oplus B_i$$

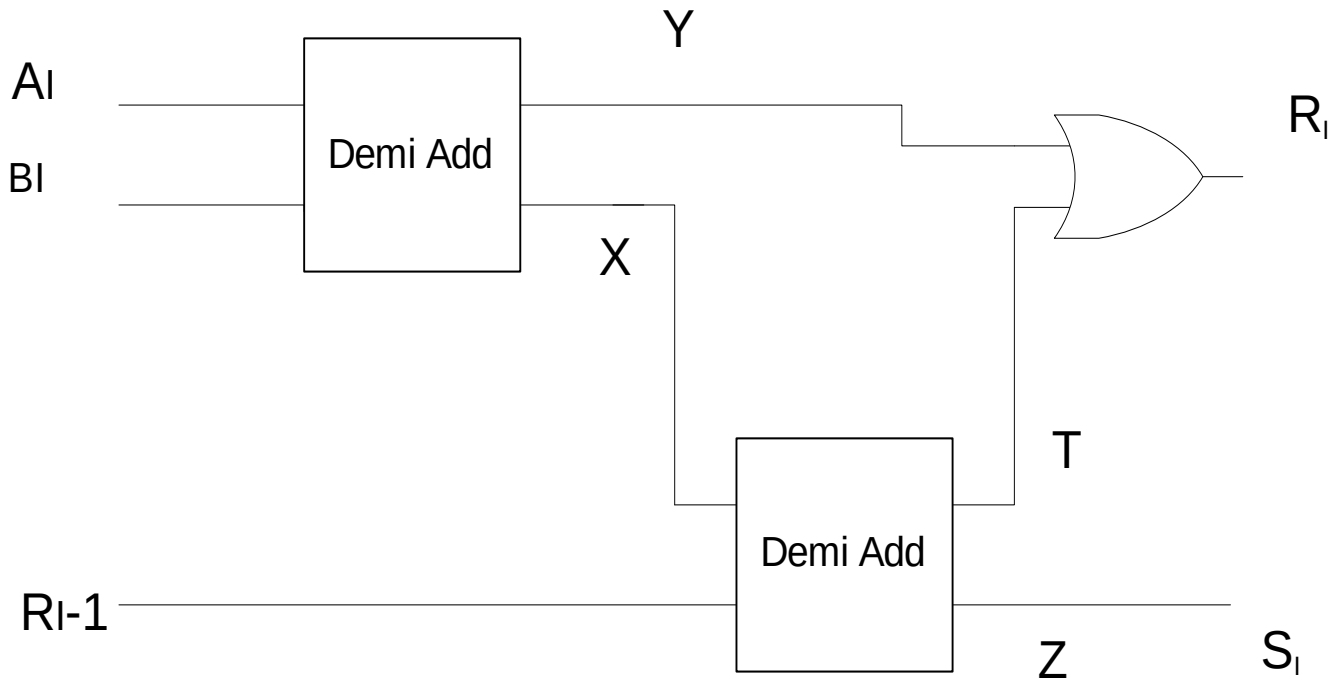
$$Y = A_i B_i$$

$$Z = X \oplus R_{i-1}$$

$$T = R_{i-1}.X$$

$$R_i = Y + T$$

$$S_i = Z$$



Additionneur sur 4 bits

- **Un additionneur sur 4 bits est un circuit qui permet de faire l'addition de deux nombres A et B de 4 bits chacun**
  - **$A(a_3a_2a_1a_0)$**
  - **$B(b_3b_2b_1b_0)$**

  En plus il tient en compte de la retenue entrante

- **En sortie on va avoir le résultat sur 4 bits ainsi que la retenue (5 bits en sortie)**
- **Donc au total le circuit possède 9 entrées et 5 sorties.**
- **Avec 9 entrées on a 2^9=512 combinaisons !!!!!!**

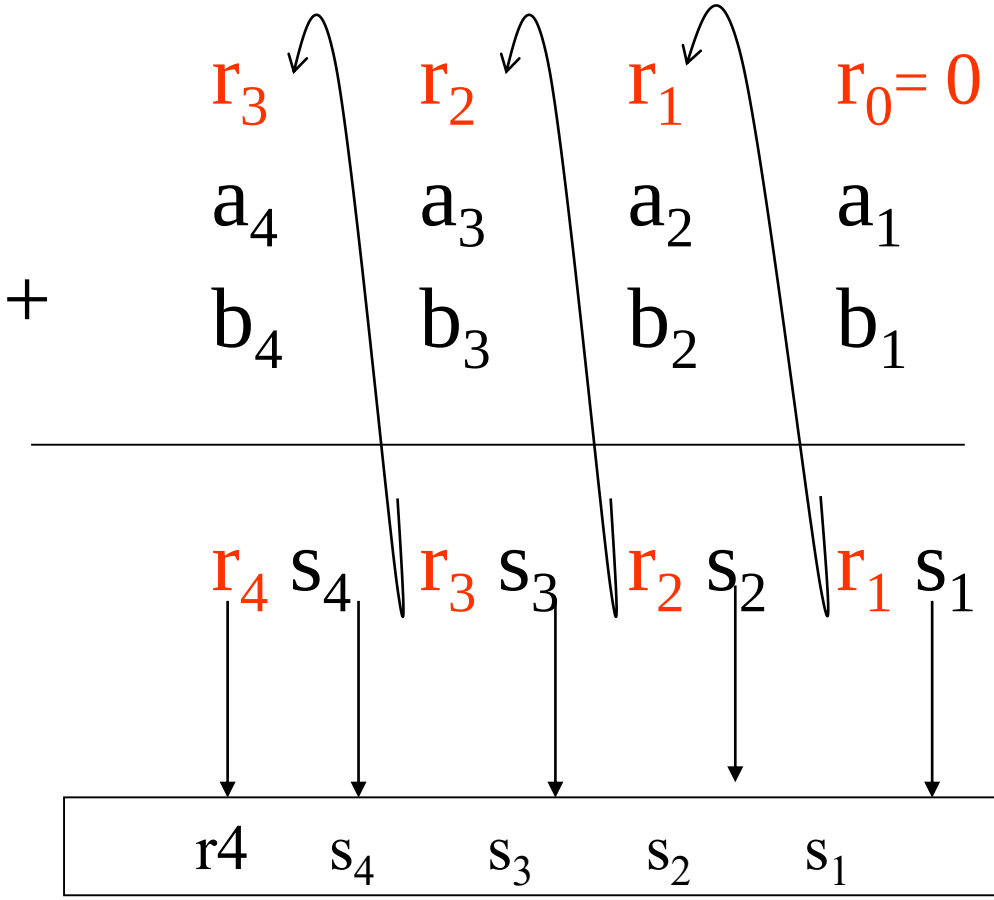
Comment faire pour représenter la table de vérité ?????

- **Il faut trouver une solution plus facile et plus efficace pour concevoir ce circuit ?**

Additionneur sur 4 bits

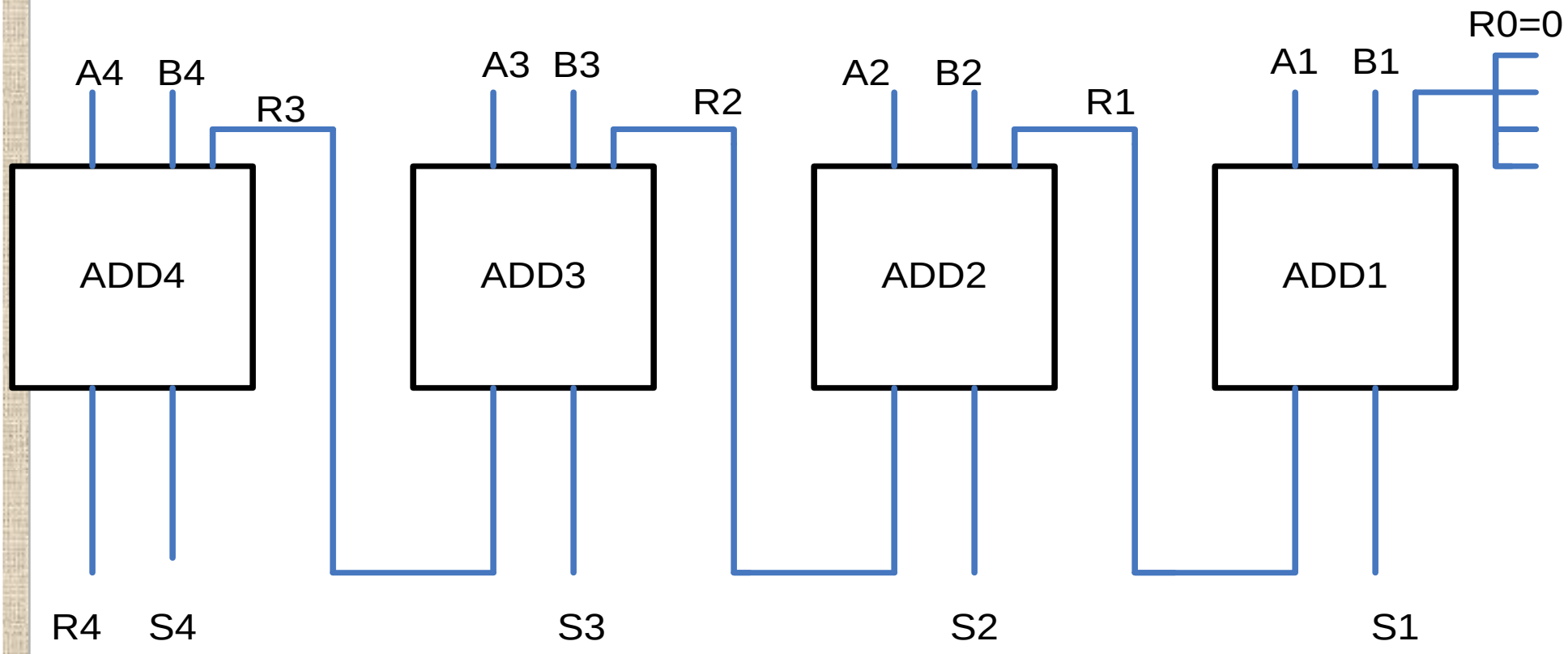
• Lorsque on fait l'addition en binaire , on additionne bit par bit en commençant à partir du poids fiable et à chaque fois on propage la retenue sortante au bit du rang supérieur.

L'addition sur un bit peut se faire par un additionneur complet sur 1 bits.



Résultat final

Additionneur 4 bits (schéma)



Additionneur à retenue anticipée

- **L'inconvénient des structures précédentes est le temps nécessaire à la réalisation de l'addition. Ce temps est en effet conditionné par la propagation de la retenue à travers tous les additionneurs élémentaires.**

- **Dans un additionneur à retenue anticipée on évalue en même temps la retenue de chaque étage. Pour cela on détermine pour chaque étage les quantités Pi et Gi suivantes:**
  - **$p_i = a_i \oplus b_i$ (propagation d'une retenue)**
  - **$g_i = a_i.b_i$ (génération d'une retenue)**

Additionneur à retenue anticipée

$p_i = a_i \oplus b_i$ (propagation d'une retenue)

$g_i = a_i.b_i$ (génération d'une retenue)

$S_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$

$c_i = g_{i-1} + p_{i-1}.g_{i-2} + p_{i-1}.p_{i-2}.g_{i-3} + \ldots\ldots\ldots\ldots + p_{i-1}.p_{i-2}.p_{i-3}\ldots.p_0.c_0$

Additionneur à retenue anticipée

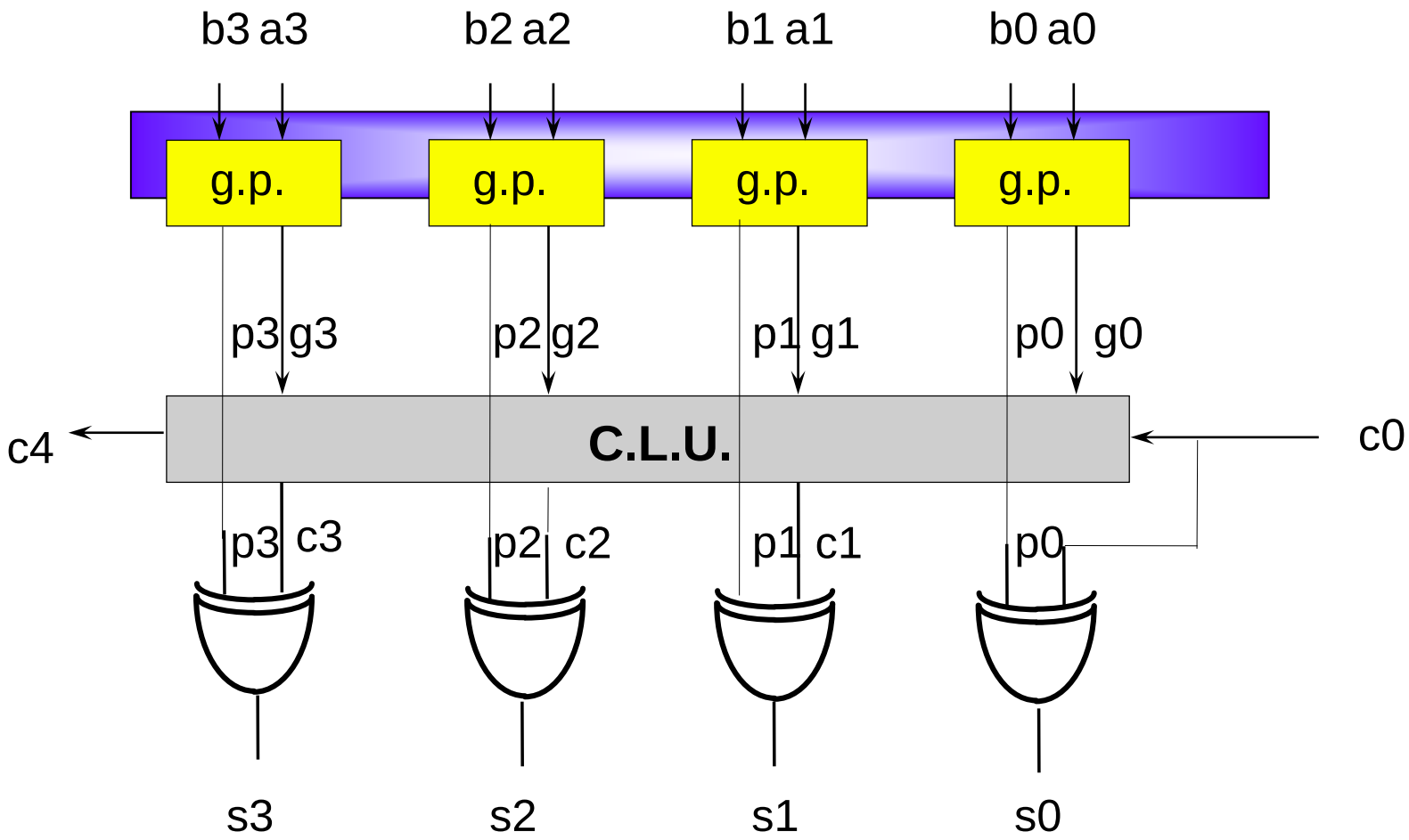
$p_i = a_i \oplus b_i$ (propagation d'une retenue)
$g_i = a_i.b_i$ (génération d'une retenue)

La retenue entrante à l'ordre i vaut 1 si :
- soit l'étage i-1 a généré la retenue ($g_{i-1} = 1$)
- soit l'étage i-1 a propagé la retenue générée à l'étage i-2 ($p_{i-1}=1$ et $g_{i-2}=1$)
- soit les étages i-1 et i-2 ont propagé la retenue générée à l'étage i-3 ($p_{i-1}=p_{i-2}=1$ et $g_{i-3}=1$)
.....
- soit tous les étages inférieurs ont propagé la retenue entrante dans l'additionneur ($p_{i-1}=p_{i-2}=...=p_0=c_0=1$).

$c_i = g_{i-1} + p_{i-1}.g_{i-2} + p_{i-1}.p_{i-2}.g_{i-3} + ... + p_{i-1}.p_{i-2}.p_{i-3}....p_0.c_0$

$c_1 = g_0 + p_0.c_0$
$c_2 = g_1 + p_1.g_0 + p_1.p_0.c_0$
$c_3 = g_2 + p_2.g_1 + p_2.p_1.g_0 + p_2.p_1.p_0.c_0$
$c_4 = g_3 + p_3.g_2 + p_3.p_2.g_1 + p_3.p_2.p_1.g_0 + p_3.p_2.p_1.p_0.c_0$

Additionneur à retenue anticipée



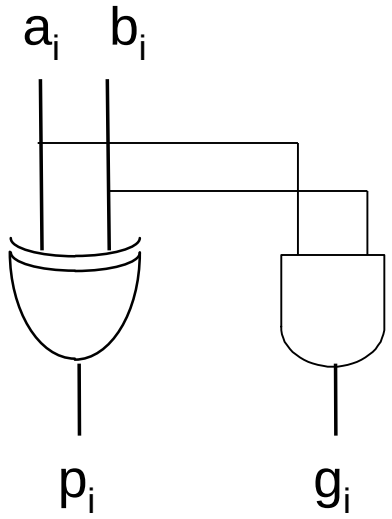
CLU : Carry Look-ahead Unit

$S_i = p_i \oplus c_i$

Bloc g.p.

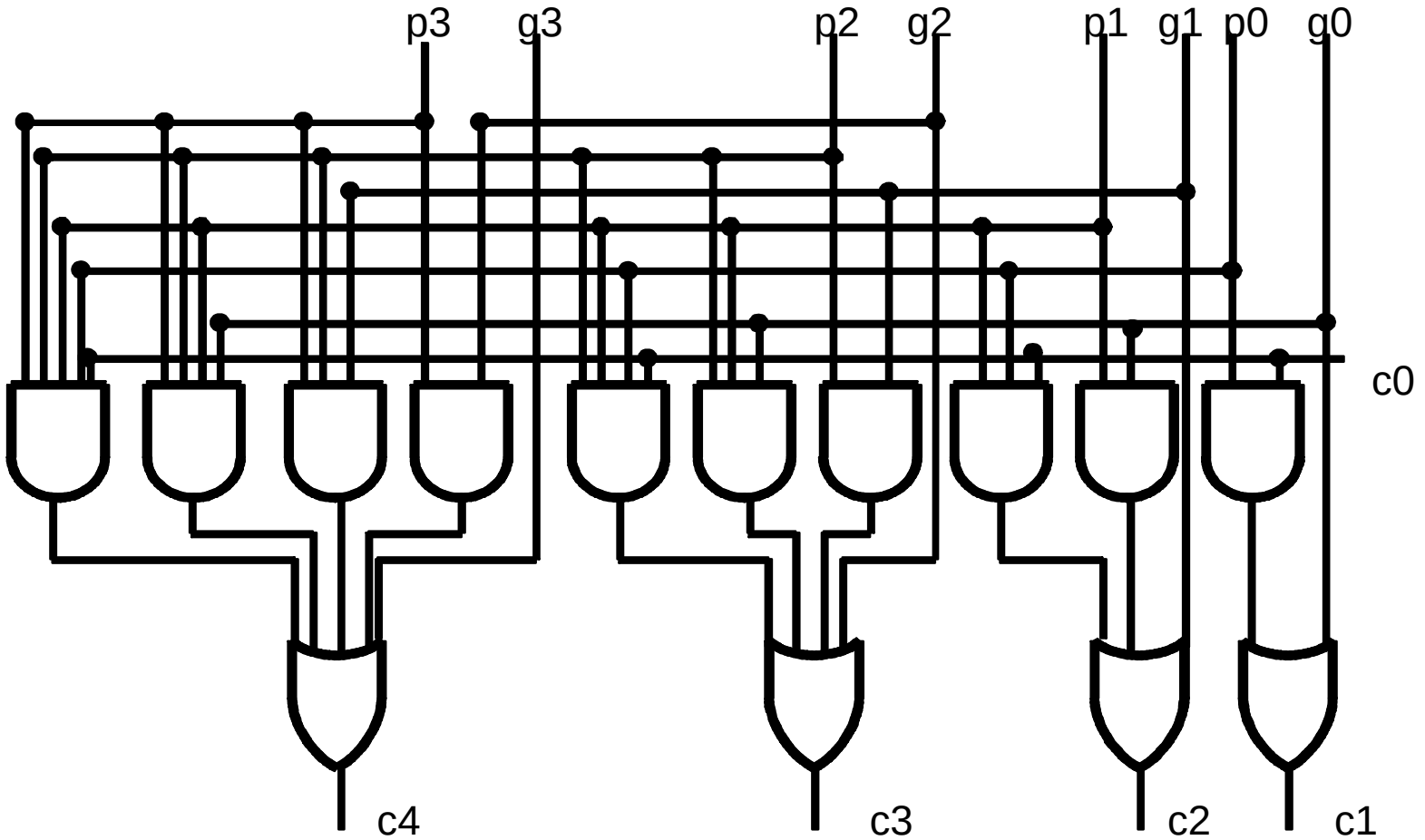
$p_i = a_i \oplus b_i$ (propagation d'une retenue)

$g_i = a_i.b_i$ (génération d'une retenue)



Bloc CLU

$c_i = g_{i-1} + p_{i-1}.g_{i-2} + p_{i-1}.p_{i-2}.g_{i-3} + \ldots\ldots\ldots\ldots + p_{i-1}.p_{i-2}.p_{i-3}\ldots.p_0.c_0$

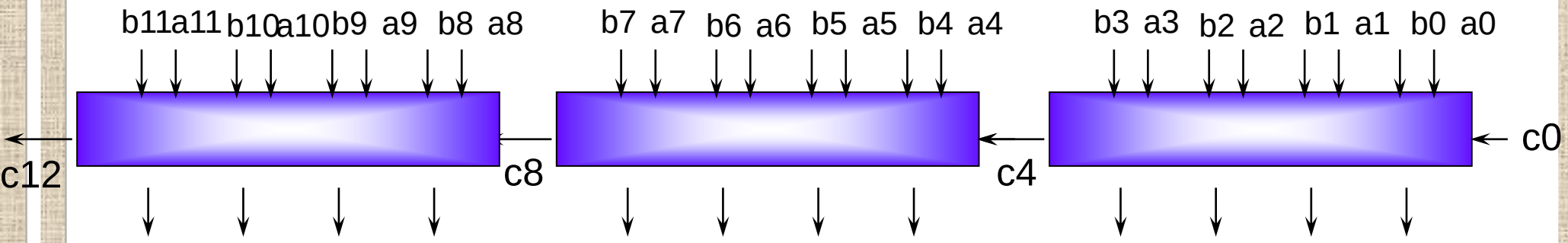


Délai : 2 portes

C.L. Adder (n>4)

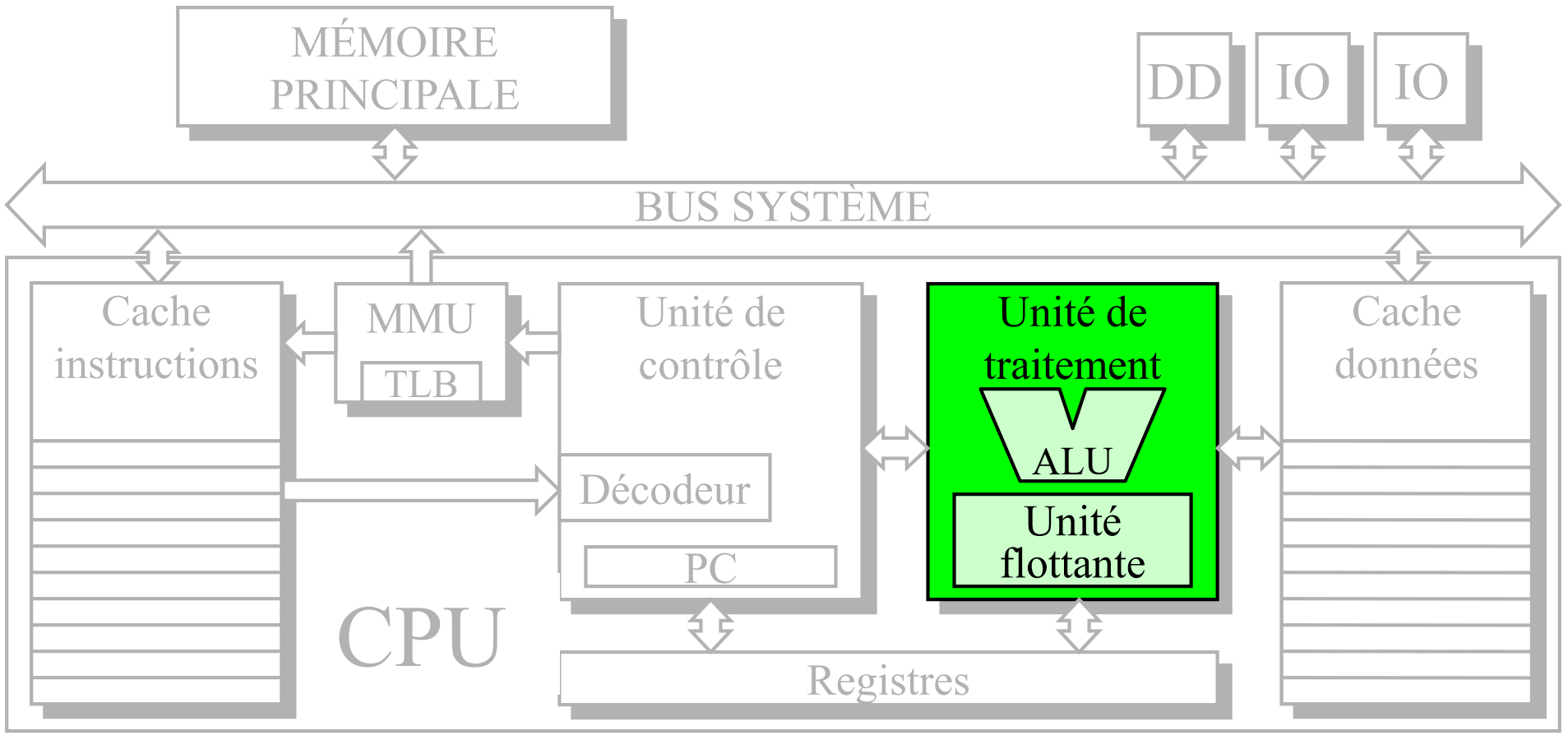
En pratique : n = 4

Pour n >4 :
Arbre de C.L.A multi-niveau (au détriment de la vitesse)



Arithmétique II

- **Opérations arithmétiques – Multiplication et Division**
- **Unité de Traitement**



Multiplication - Nombres entiers

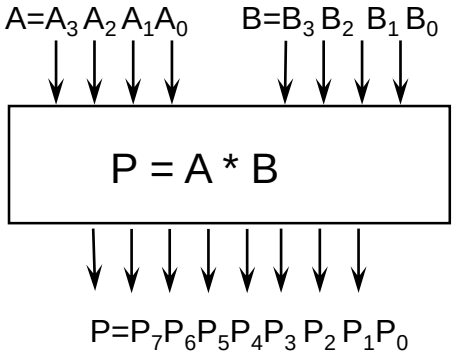
Multiplication de deux nombres entiers de n bits:

Binaire:

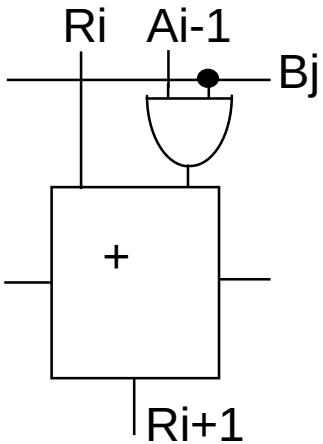
```
1000 x                    (8)
1001                      (9)
1000 +
0000  +
0000   +
1000    +
1001000              (72)
```

Note: résultat -> 2n bits!

Multiplieur 4 bits



$$
\begin{array}{rcccccccc}
 & A_3 & A_2 & A_1 & A_0 & & & & \\
* & B_3 & B_2 & B_1 & B_0 & & & & \\
\hline
 & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & & & & \text{PP1} \\
 & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & & & & \text{PP2} \\
 & A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & & & \text{PP3} \\
 & A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & & \text{PP4} \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0 &
\end{array}
$$



PP1 + PP2 = R1 =>
R1 + PP3 = R2 =>
R2 + PP4 = P

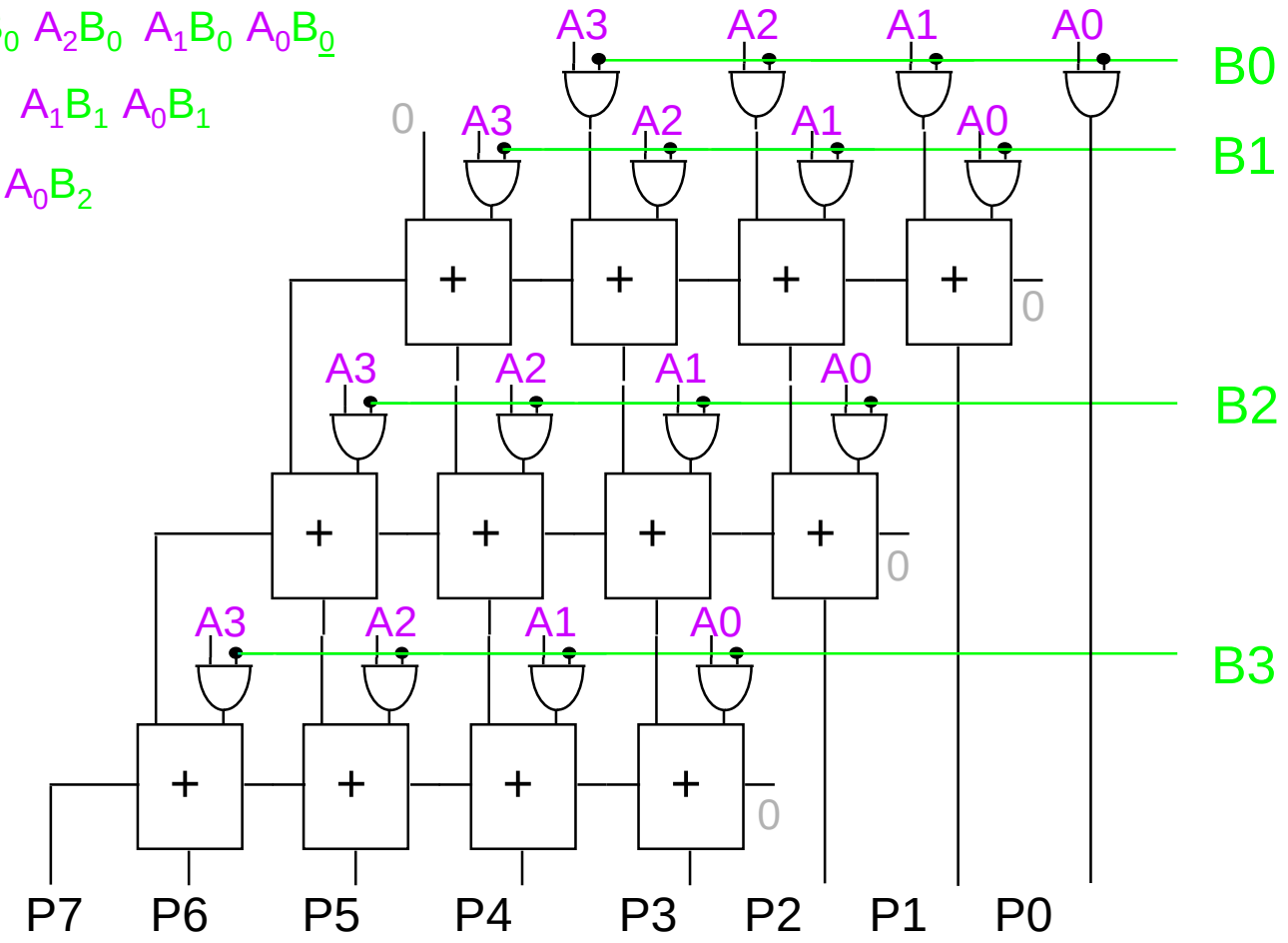
Multiplieur 4 bits

A_3B_0 A_2B_0 A_1B_0 A_0B_0

A_3B_1 A_2B_1 A_1B_1 A_0B_1

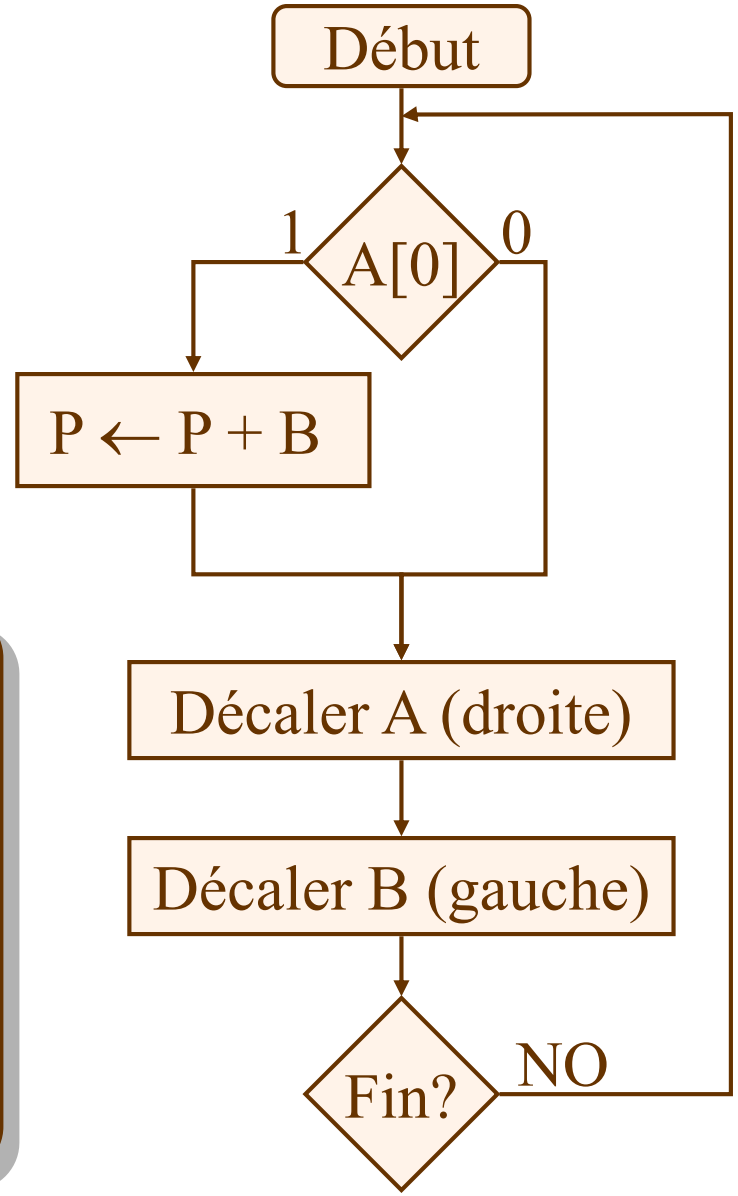
A_3B_2 A_2B_2 A_1B_2 A_0B_2

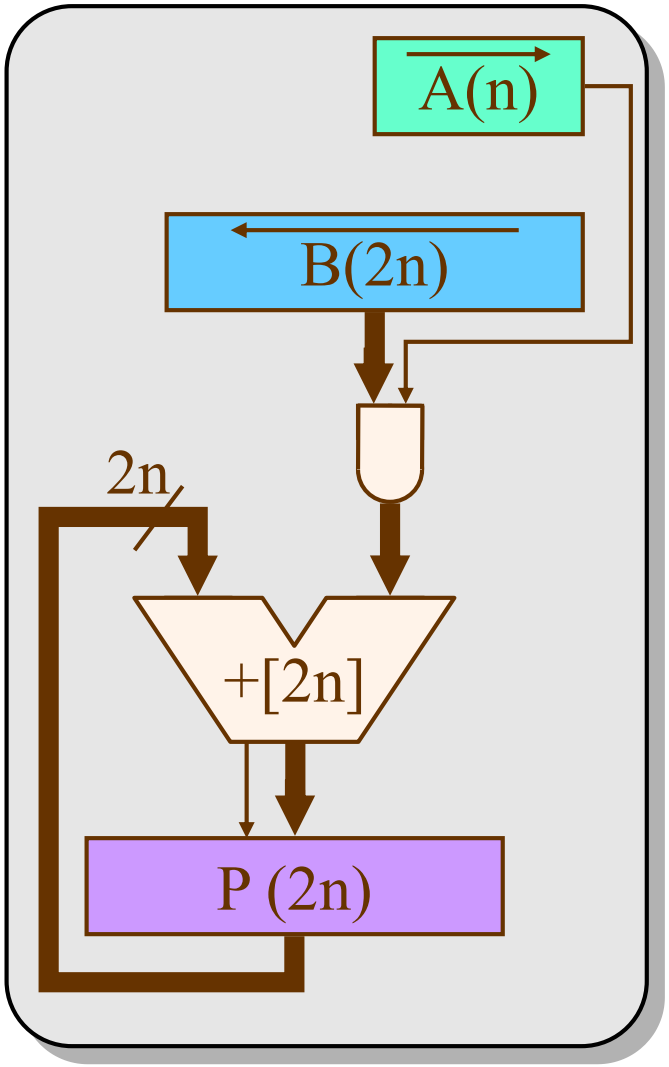
A_3B_3 A_2B_3 A_1B_3 A_0B_3



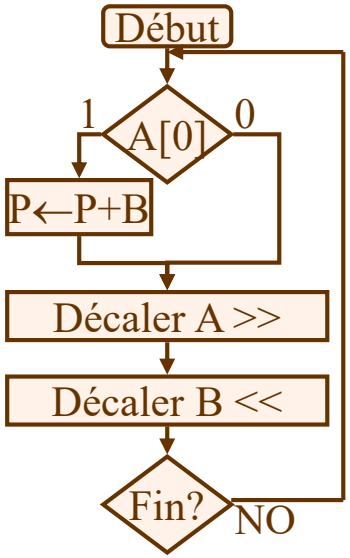
Multiplication - Implémentation I

```
B     1000  x
A     1001
      1000  +
     0000   +
    0000    +
  1000      +
 1001000
```

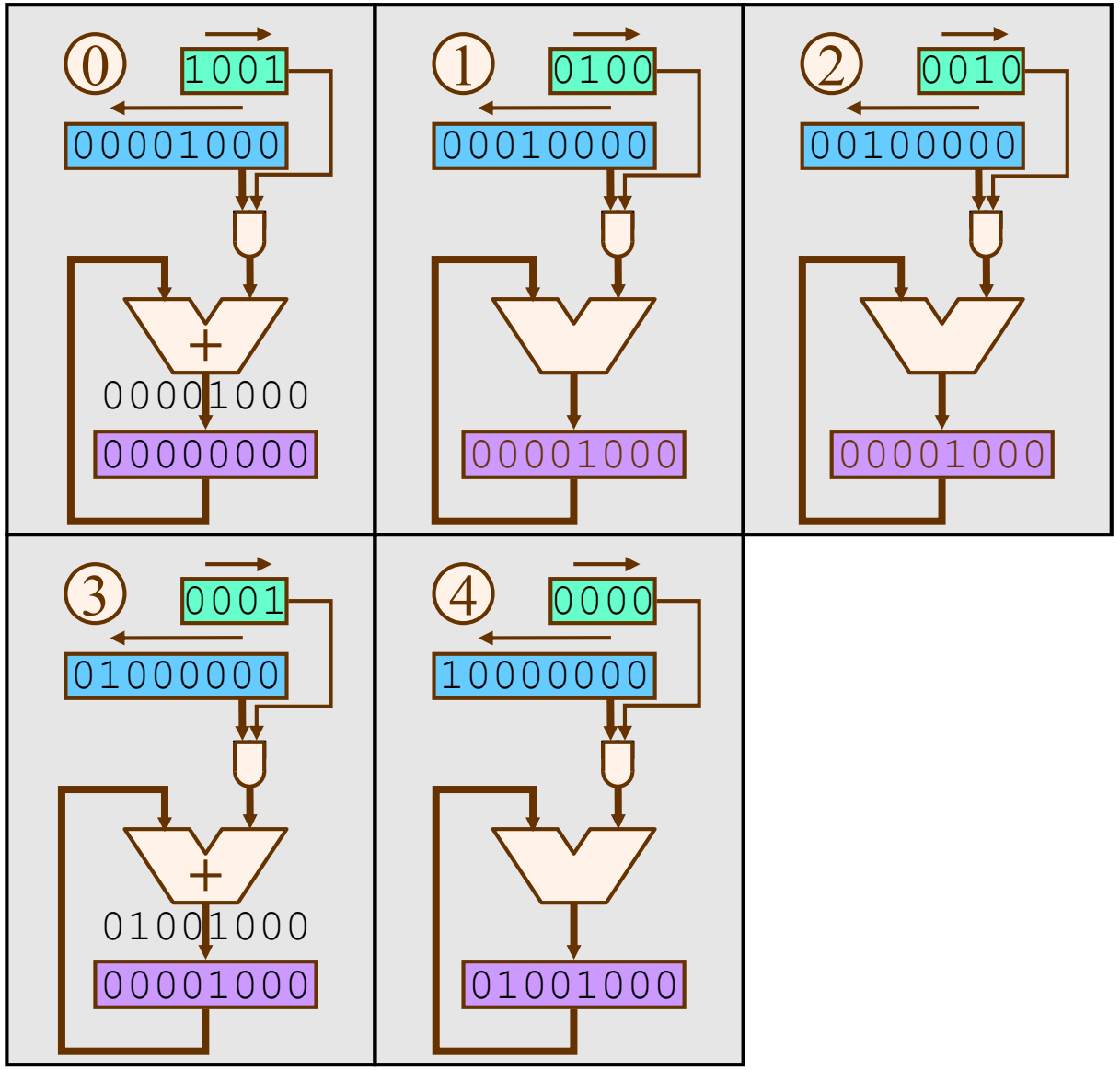




Multiplication - Implémentation I

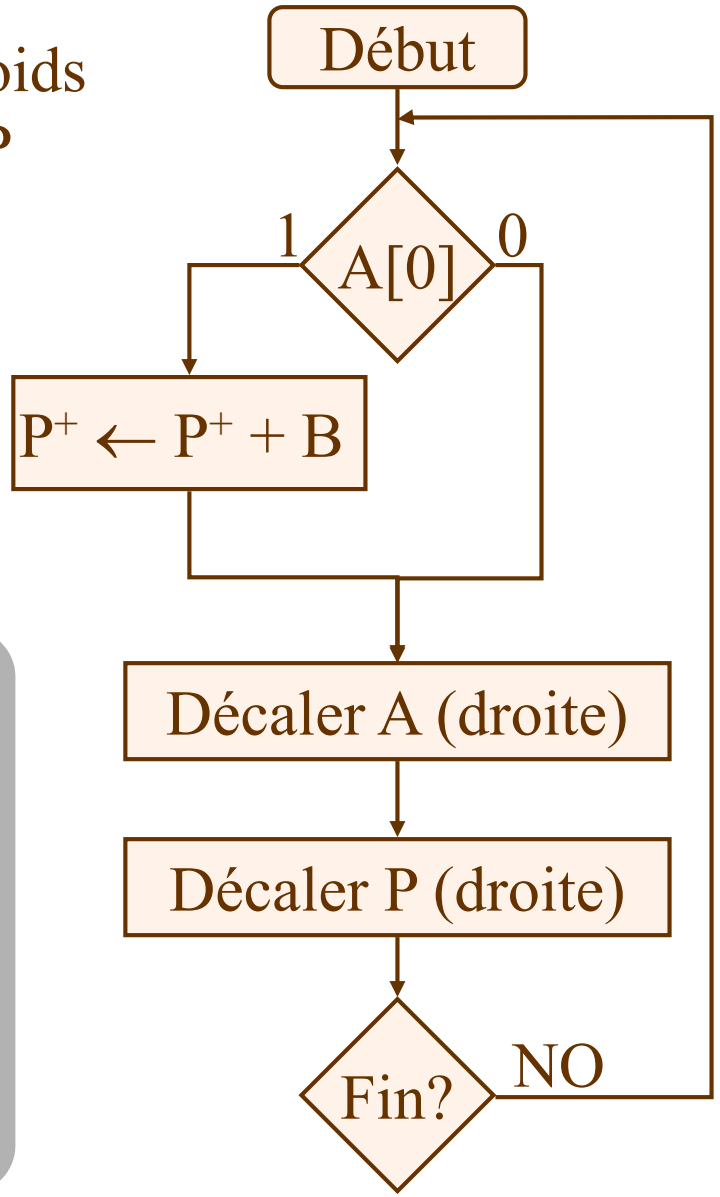


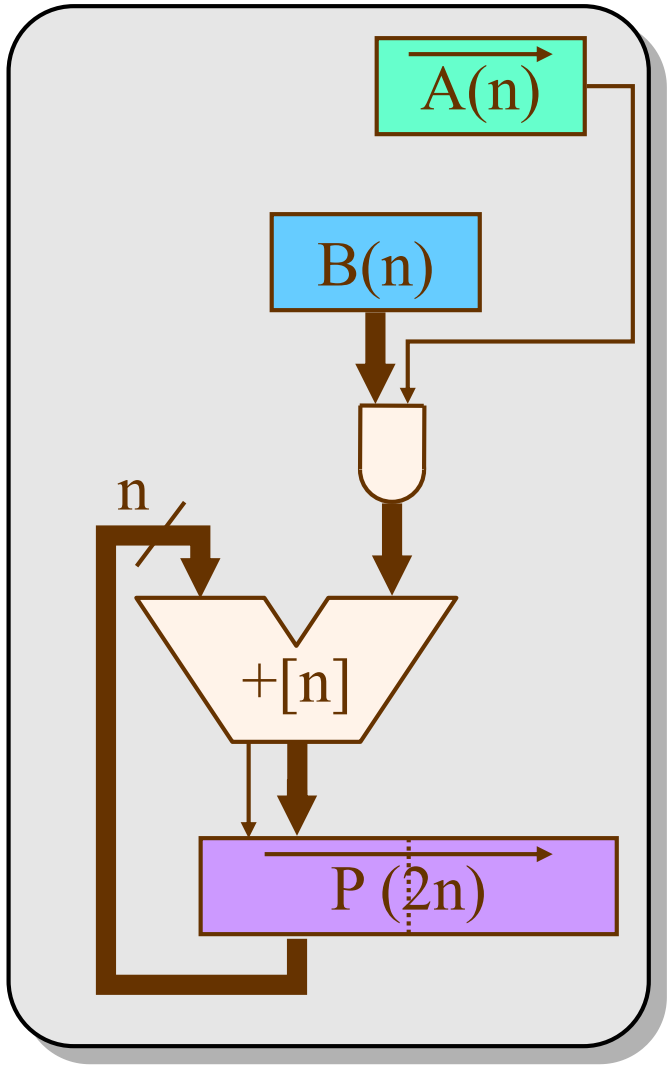

```
B     1000  x
A     1001
-------------
      1000  +
     0000   +
    0000    +
  1000      +
-------------
 1001000
```



Multiplication - Implémentation II

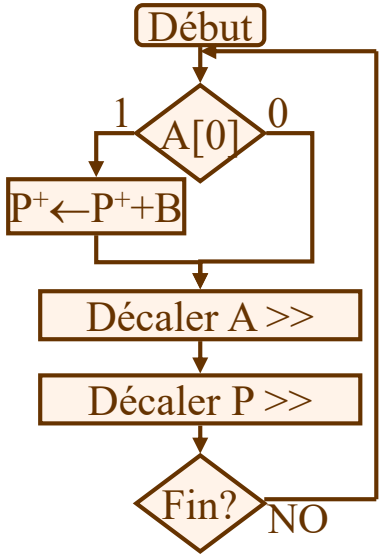
P^+= moitié de poids fort du registre P



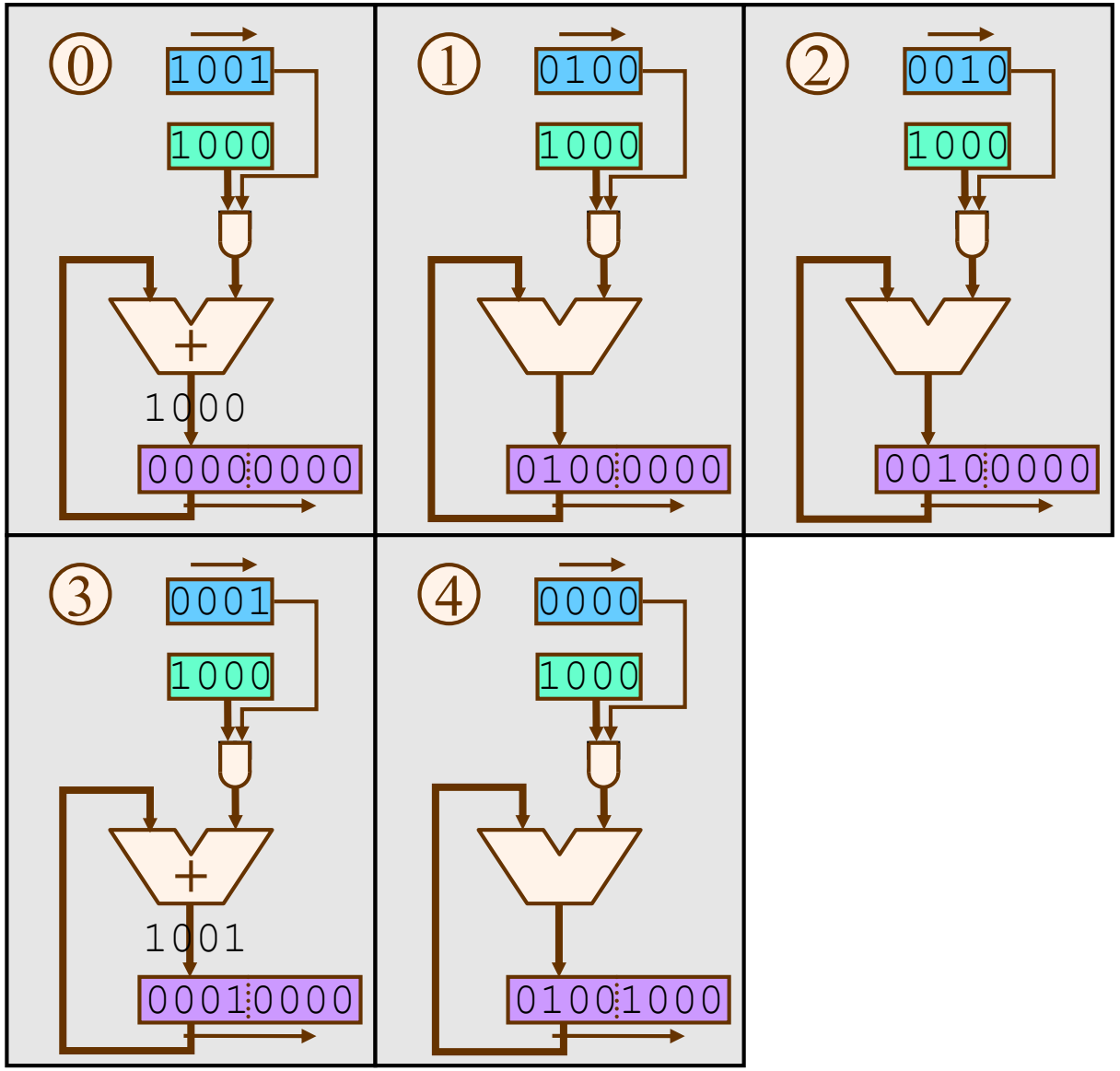



```
B     1000  x
A     1001
-------------
      1000  +
     0000   +
    0000    +
   1000     +
-------------
  1001000
```

Multiplication - Implémentation II



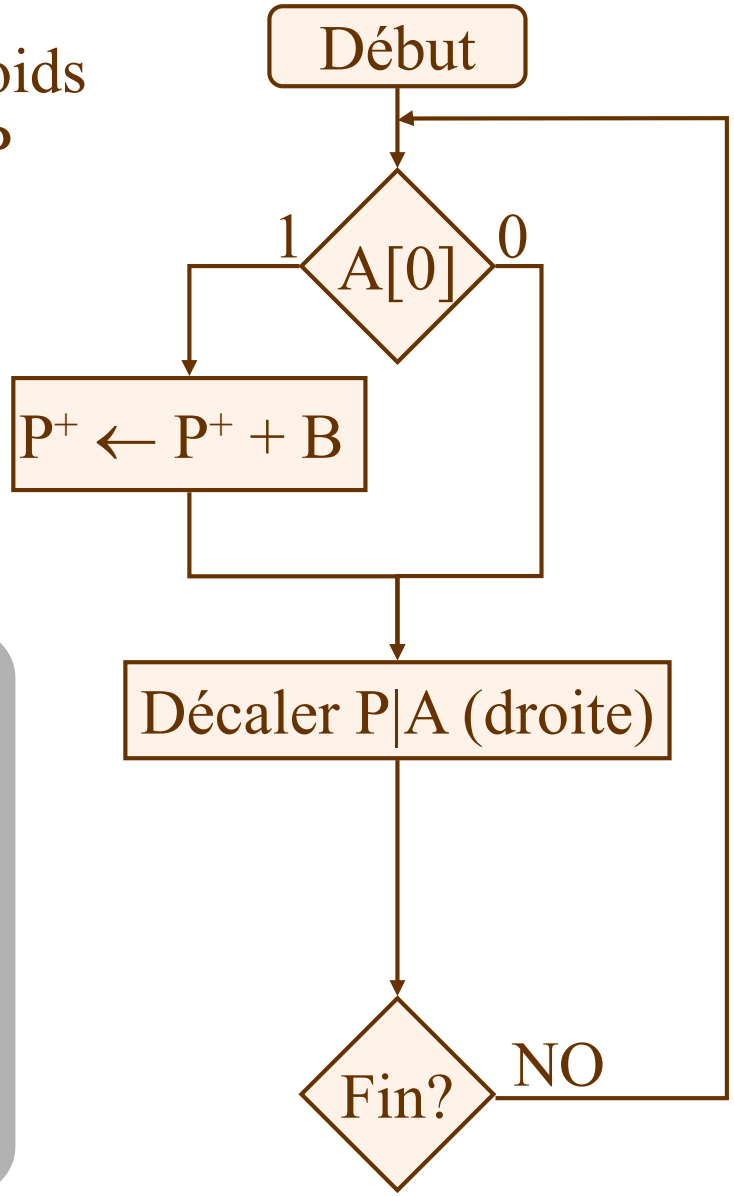

```
B     1000  x
A     1001
      1000  +
     0000   +
    0000    +
  1000      +
-------------
 1001000
```

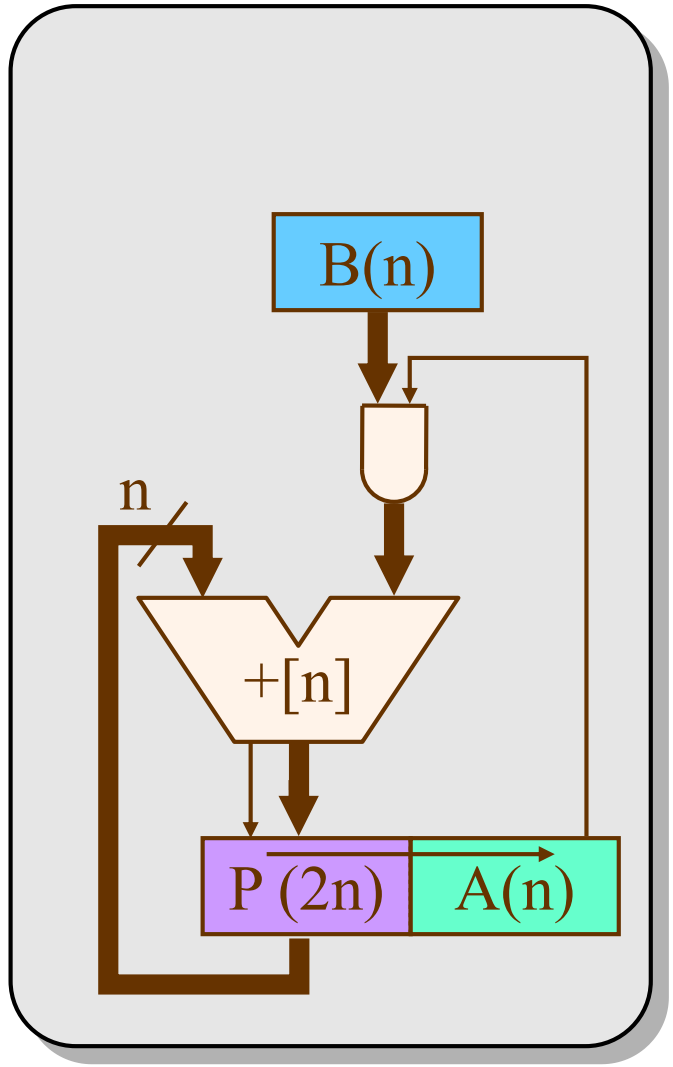


Multiplication - Implémentation III

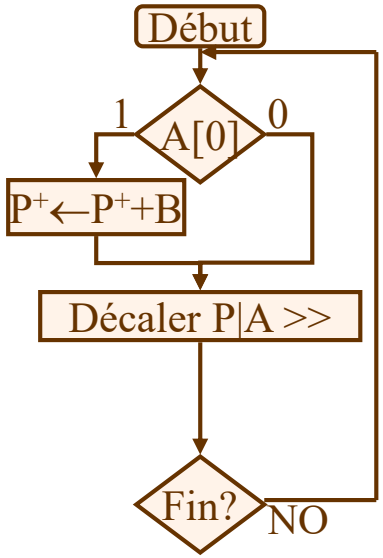
P^+= moitié de poids fort du registre P

```
B     1000  x
A     1001
-----------
      1000  +
     0000   +
    0000    +
  1000      +
-----------
 1001000
```

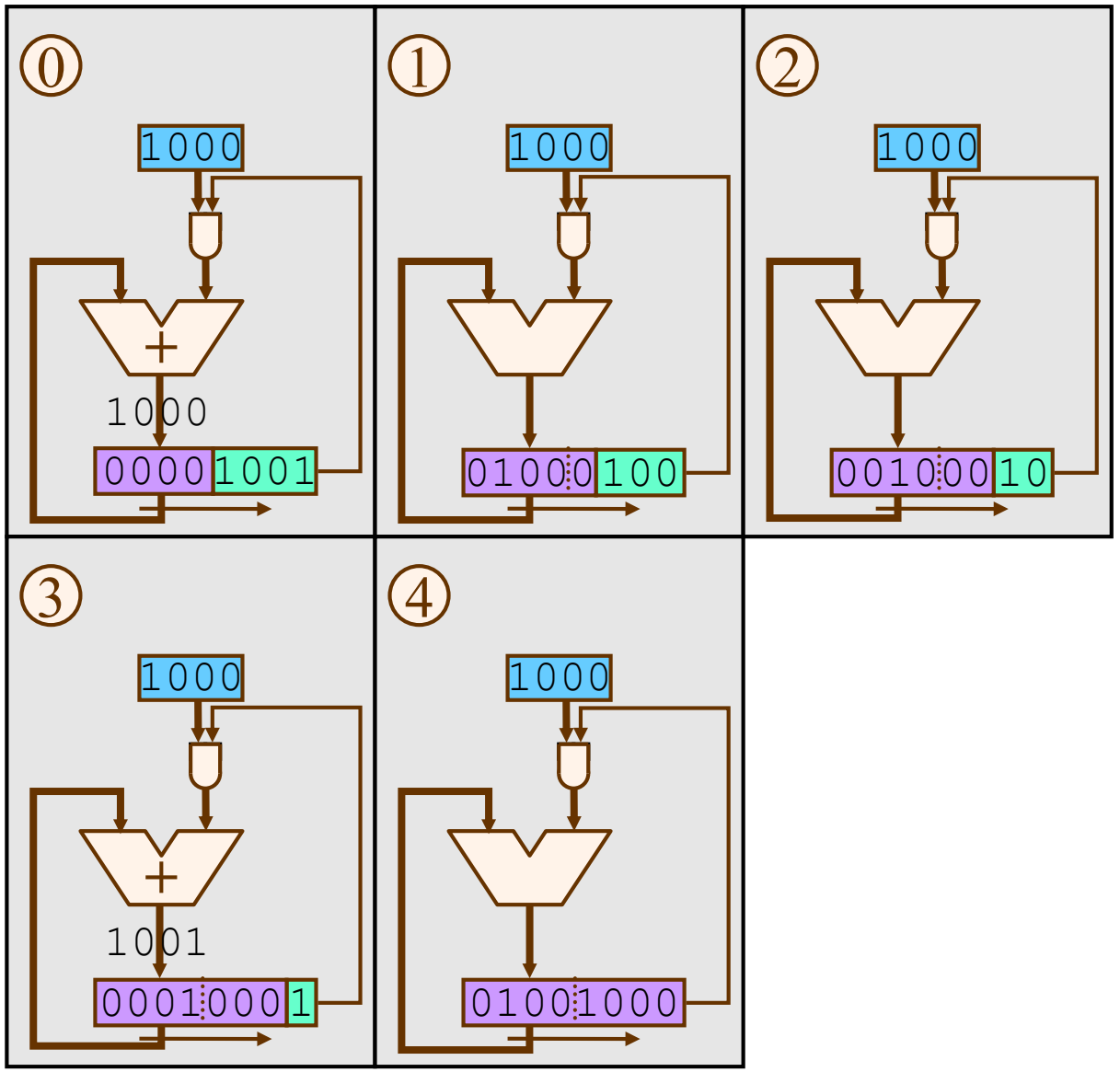




Multiplication - Implémentation III




```
B     1000  x
A     1001
-------------
      1000  +
     0000   +
    0000    +
   1000     +
-------------
  1001000
```



Division Binaire

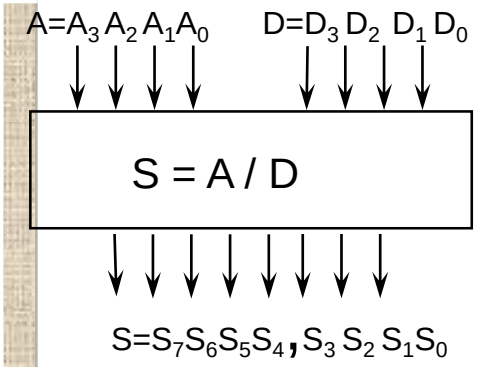
Exemple : 11 / 9

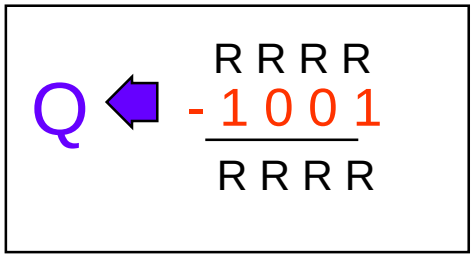
```
1 1      |  9
         |-------
  2 0    |  1 , 2 2 ...
    2 0  |
      2  |
```

```
                1 0 1 1          | 1 0 0 1
              - 1 0 0 1          |---------
                ---------------  | 1 ,0 0 1
1  ⇐          0 0 1 0 0        |
                -   1 0 0 1      |
0  ⇐          ---------------    |
              0 1 0 0 0          |
                -   1 0 0 1      |
0  ⇐          ---------------    |
              1 0 0 0 0          |
            -   1 0 0 1          |
1  ⇐          ---------------    |
                  0 1 1 1        |
```

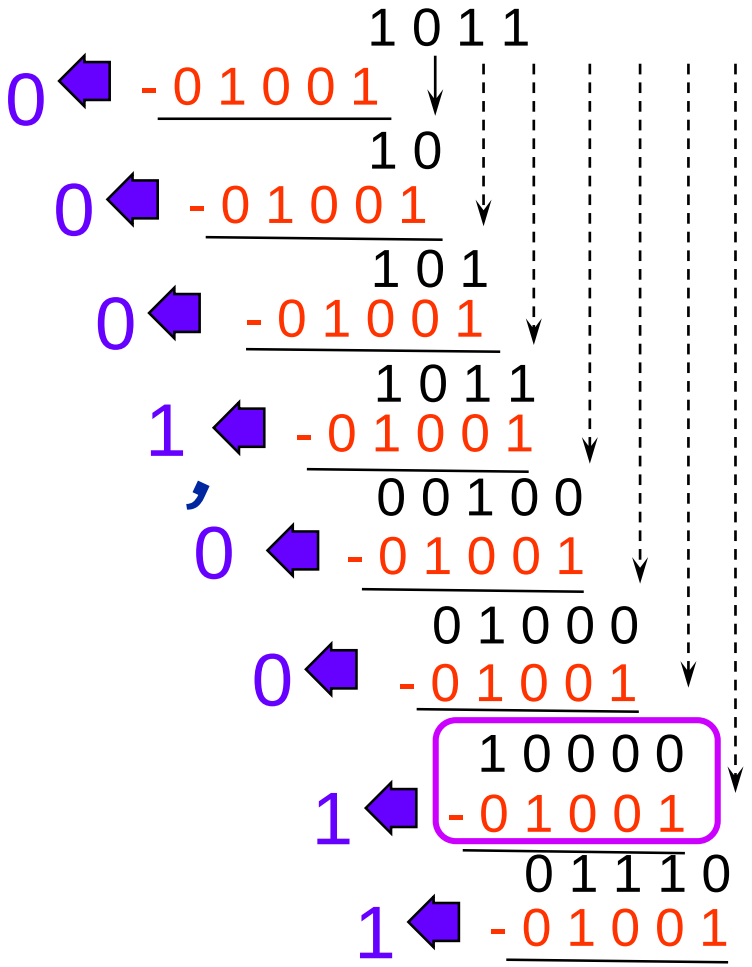
Soustraction 5 bits

Diviseur

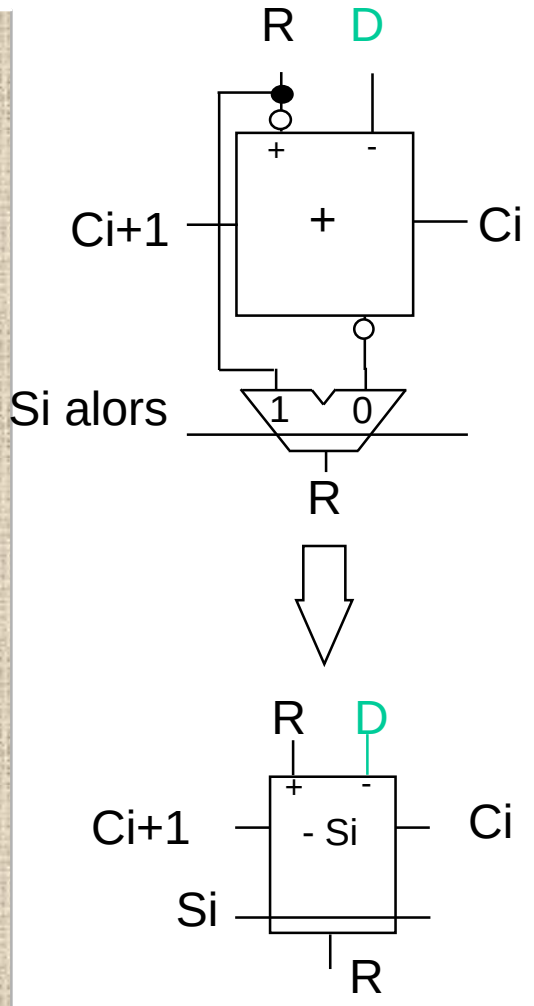


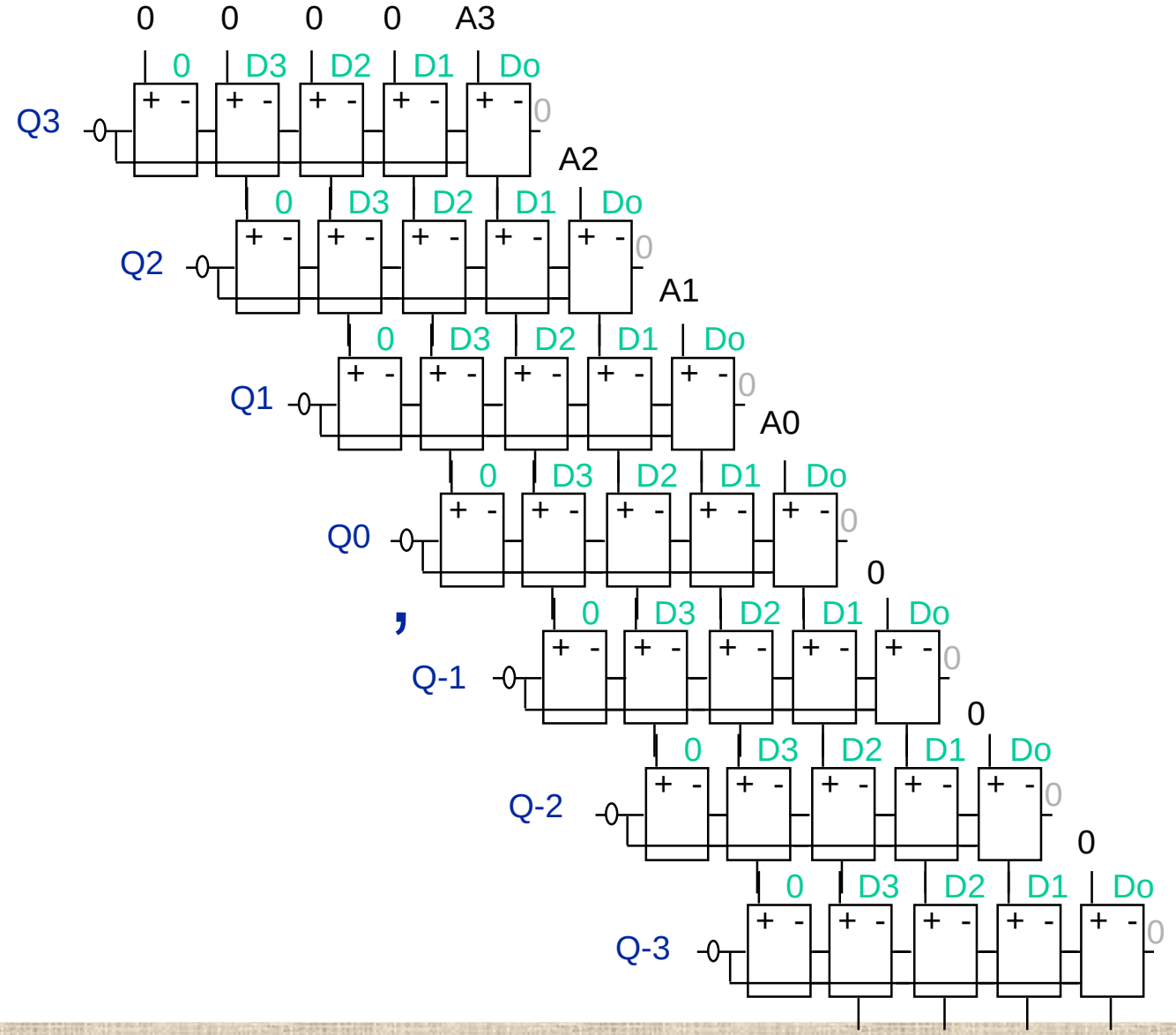


Si R>D alors Q=1 et R-D
sinon Q=0 et R



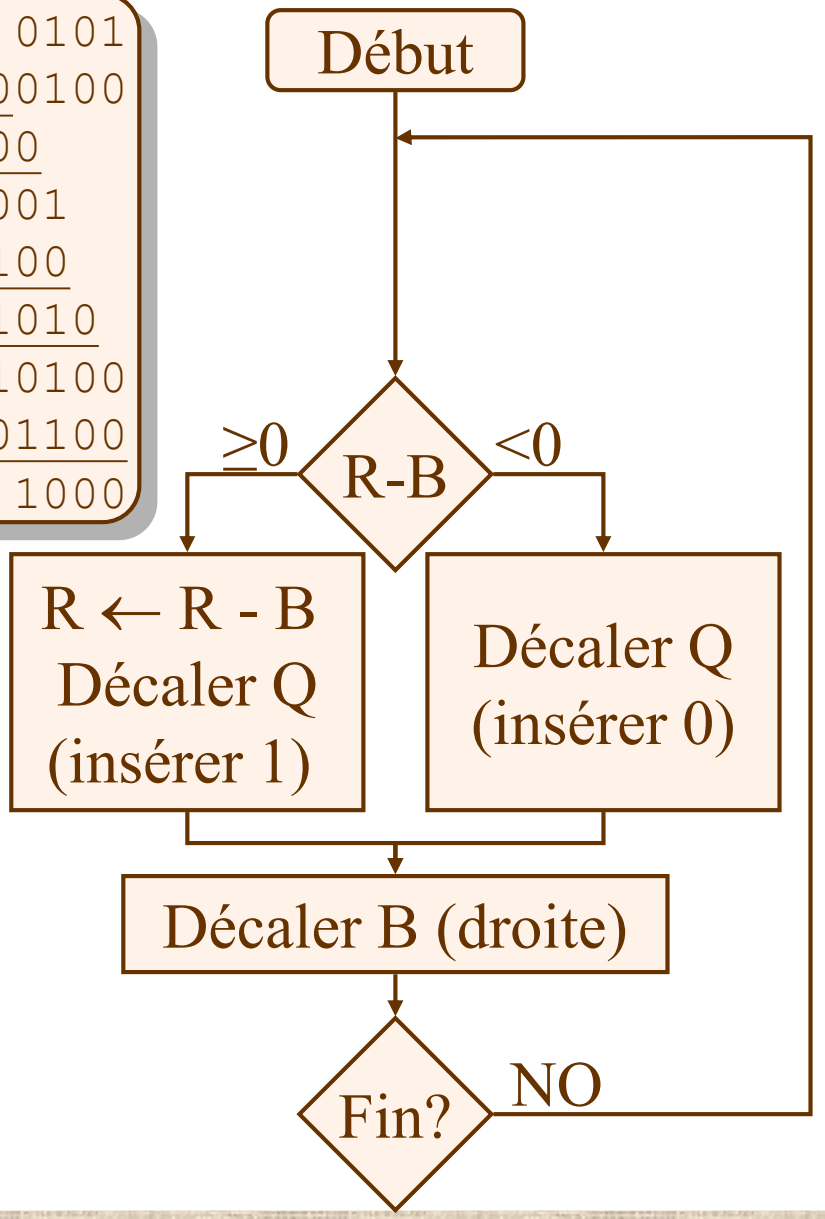
Diviseur


R
D
Ci+1
Ci
Si alors
1
0
R
R
D
Ci+1
- Si
Ci
Si
R

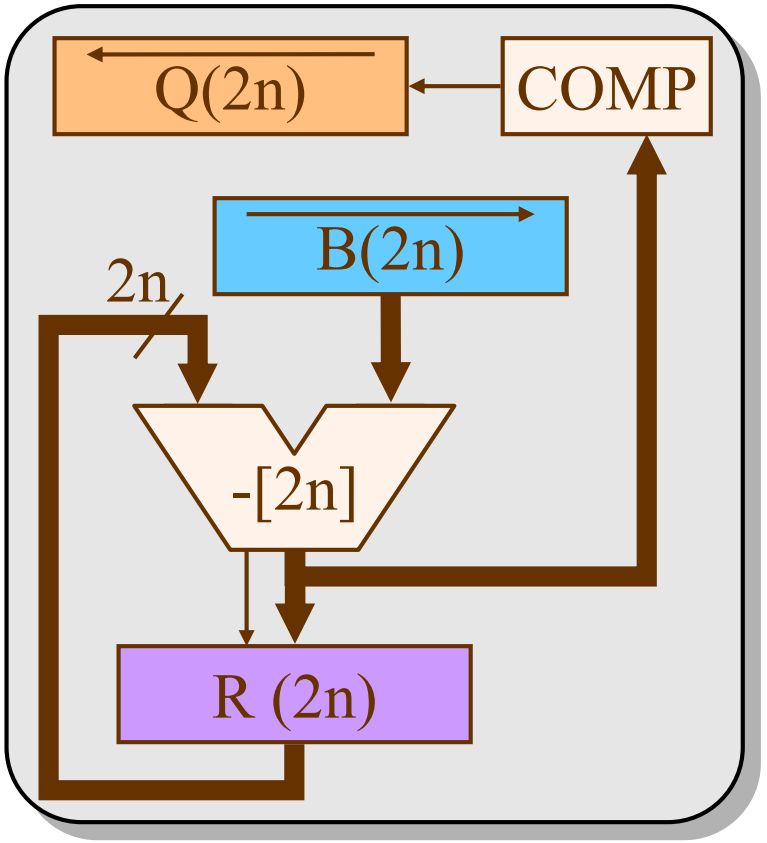

0 0 0 0 A3
0 D3 D2 D1 Do
Q3
A2
Q2
A1
Q1
A0
Q0
0
Q-1
Q-2
Q-3

Division - Implémentation

```
      |    0101
1100  |01000100
       01000
        10001
      - 01100
          1010
          10100
      -   01100
           1000
```

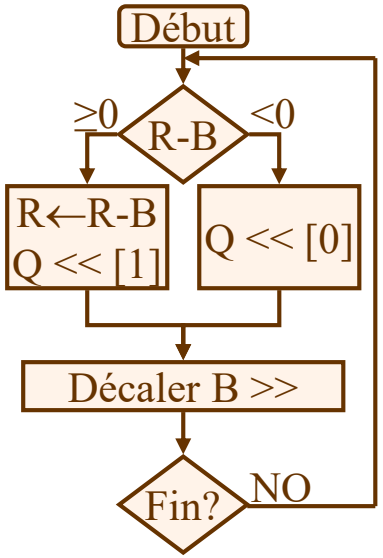


B est stockée dans la moitié
de poids fort du registre B
A est stockée dans le registre R

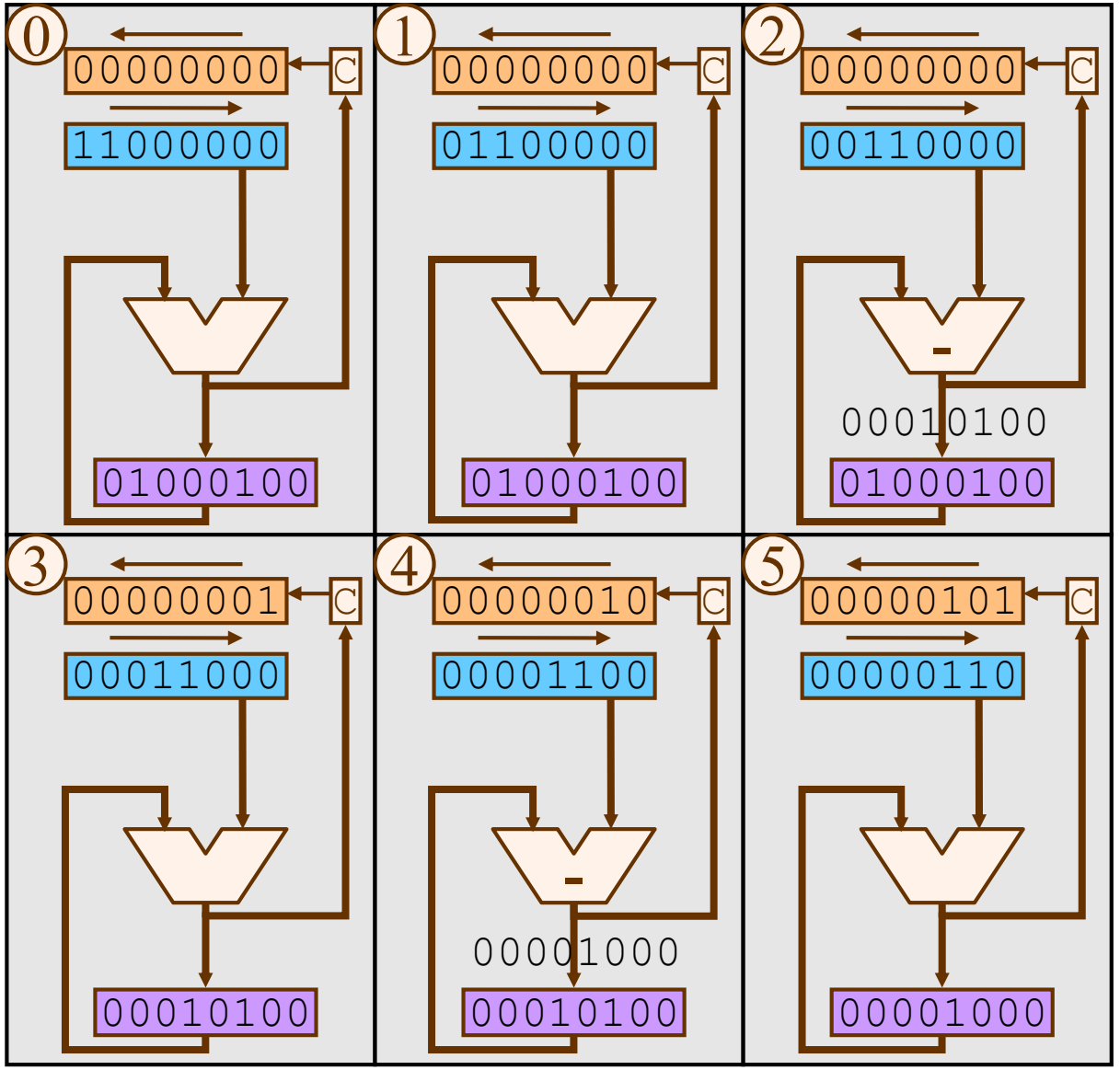


On peut simplifier la machine
comme pour la multiplication

Division - Implémentation




```
      |     0101
1100  |01000100
       01000
         10001
      - 01100
           1010
           10100
      -    01100
            1000
```



Division - Nombres signés

Approche « standard »: convertir les nombres négatifs en positifs, diviser, convertir au négatif si nécessaire.

Pour déterminer le signe des résultats, la relation suivante doit toujours être vraie:

Dividende = Quotient x Diviseur + Reste

Règle: dividende et reste doivent avoir le même signe!

+7 / +2: quotient = +3, reste = +1;
+7 / -2: quotient = -3, reste = +1;
-7 / +2: quotient = -3, reste = -1;
-7 / -2: quotient = +3, reste = -1;

Note: la valeur absolue reste la même!

Multiplication flottante

Nettement plus compliquée que la multiplication entière:

$1.110 \times 10^{10} \times 9.200 \times 10^{-5}$

1) Additionner les exposants: $10+(-5) = 5$

   Notation avec excès: $137 + 122 = 259$??????

   $137 + 122 - 127 = 132 = 5 + 127$

2) Multiplier les mantisses: $1.110 \times 9.200 = 10.212000$

3) Normaliser: $10.212000 \times 10^5 = 1.0212000 \times 10^6$

4) Arrondir: $1.0212000 \times 10^6 = 1.021 \times 10^6$

5) Trouver le signe: $+1.021 \times 10^6$

Multiplication flottante - Algorithme