

# 11. Circuits combinatoires importants

---



## Principes de fonctionnement des ordinateurs

Jonas Lätt

Centre Universitaire d’Informatique



Trouvé une erreur sur un transparent? Envoyez-moi un message

- sur Twitter @teachjl ou
- par e-mail [jonas.latt@unige.ch](mailto:jonas.latt@unige.ch)

Jonas Lätt



# Contenu du cours

## Partie I: Introduction

## Partie II: Codage de l'information

## Partie III: Circuits logiques

## Partie IV: Architecture des ordinateurs

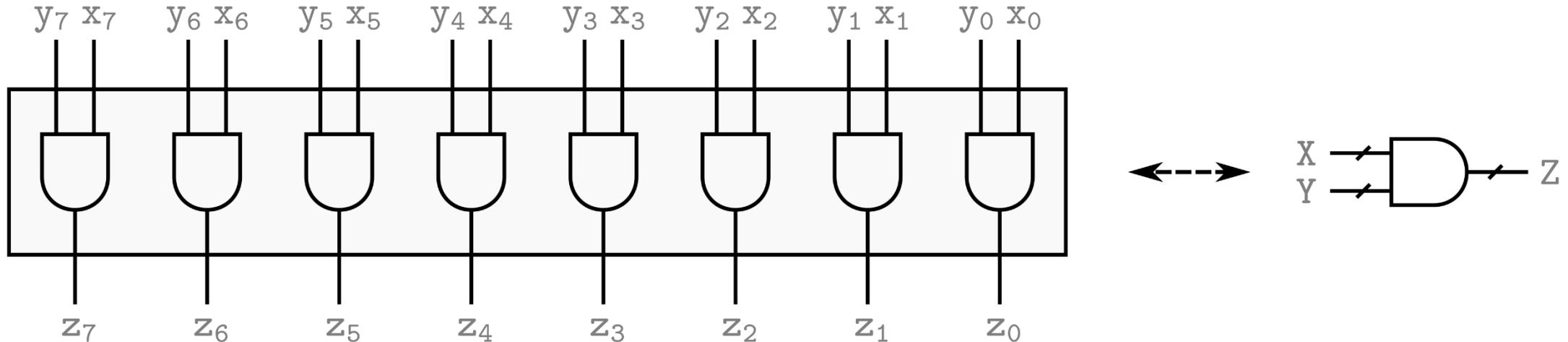
1. Introduction
2. Histoire de l'informatique
3. Information digitale et codage de l'information
4. Codage des nombres entiers naturels
5. Codage des nombres entiers relatifs
6. Codage des nombres réels
7. Codage de contenu média
8. Portes logiques
9. Circuits logiques combinatoires et algèbre de Boole
10. Réalisation d'un circuit combinatoire
- 11. Circuits combinatoires importants**
12. Principes de logique séquentielle
13. Réalisation de la bascule DFF
14. Architecture de von Neumann
15. Réalisation des composants
16. Code machine et langage assembleur
17. Architecture d'un processeur
18. Performance et micro-architecture
19. Du processeur au système

# Les portes bit-par-bit: AND à k bits

**Entrée:** des mots à k bits. **Sortie:** des mots à k bits.

**Réalisation:** répétition d'une porte logique.

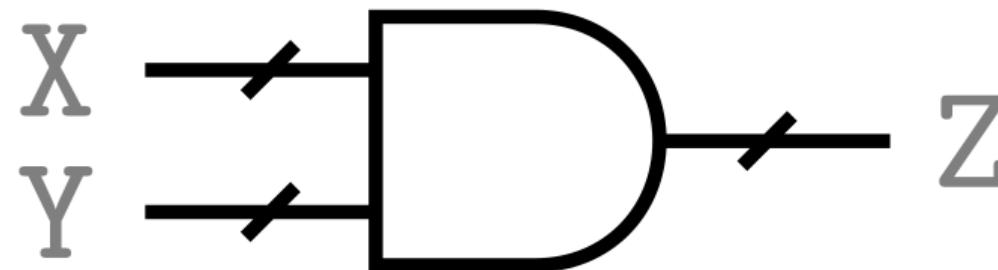
Exemple: AND à k bits ( $k=8$ )



# Les portes à k bits: Symboles

**Z = X AND Y**

( $z_i = x_i \cdot y_i$  pour  $i = 0 \dots k-1$ )





# Les portes à k entrées et 1 sortie

---

- Possible pour toute porte associative et commutative (exemple: AND et OR).

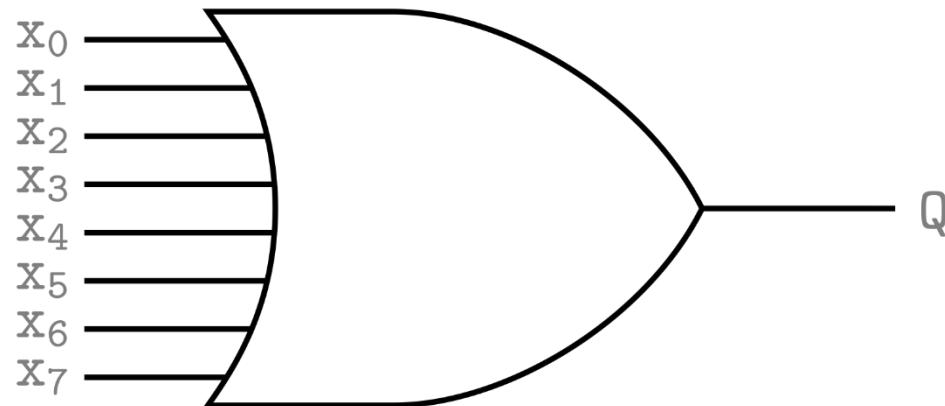
Exemple: Le OR 8-vers-1  $\text{OR}(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$

$$\text{OR}(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) = x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$$

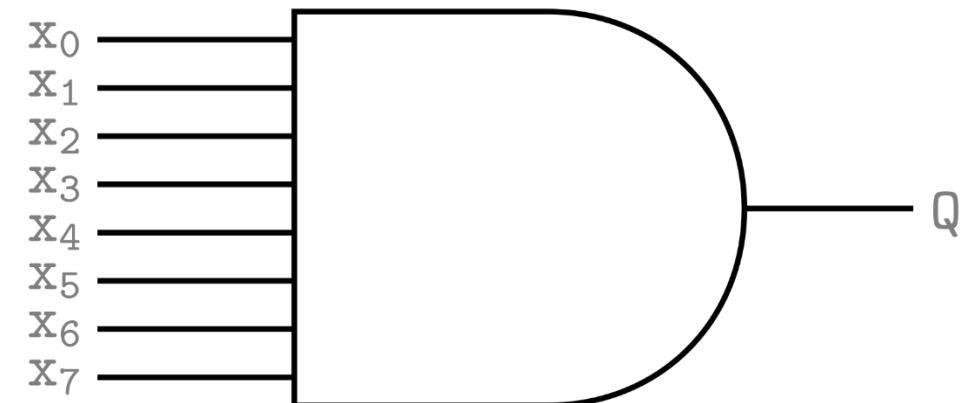
# AND et OR k-vers-1: Symboles

**Entrée:** des mots à k bits. **Sortie:** un seul bit.

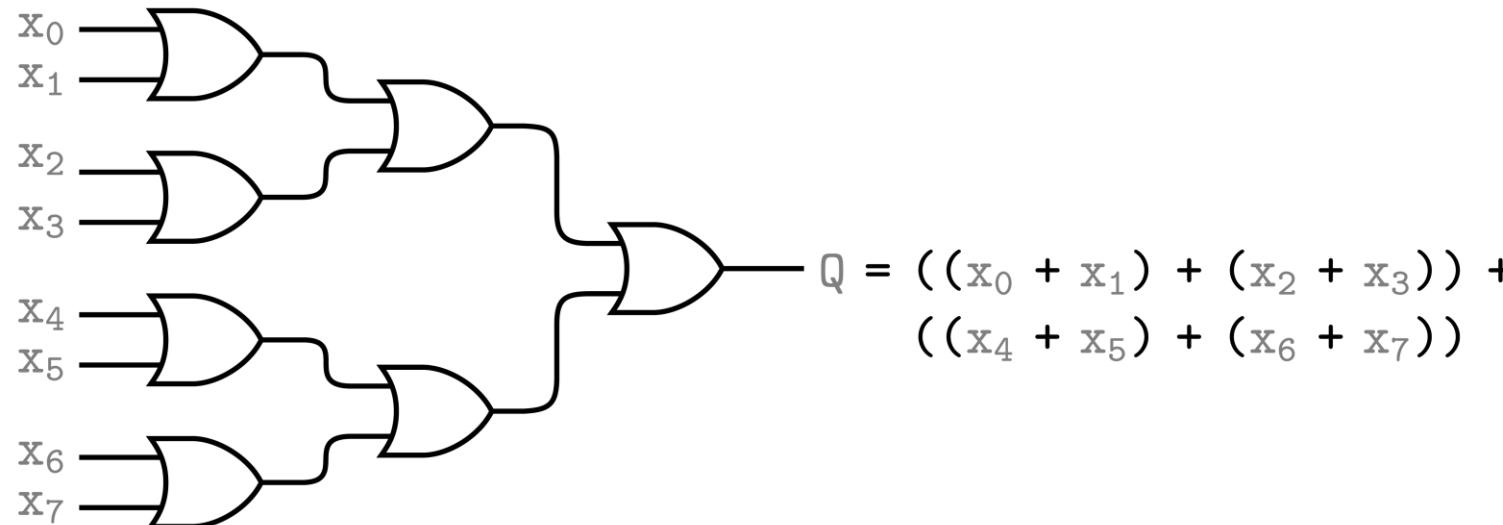
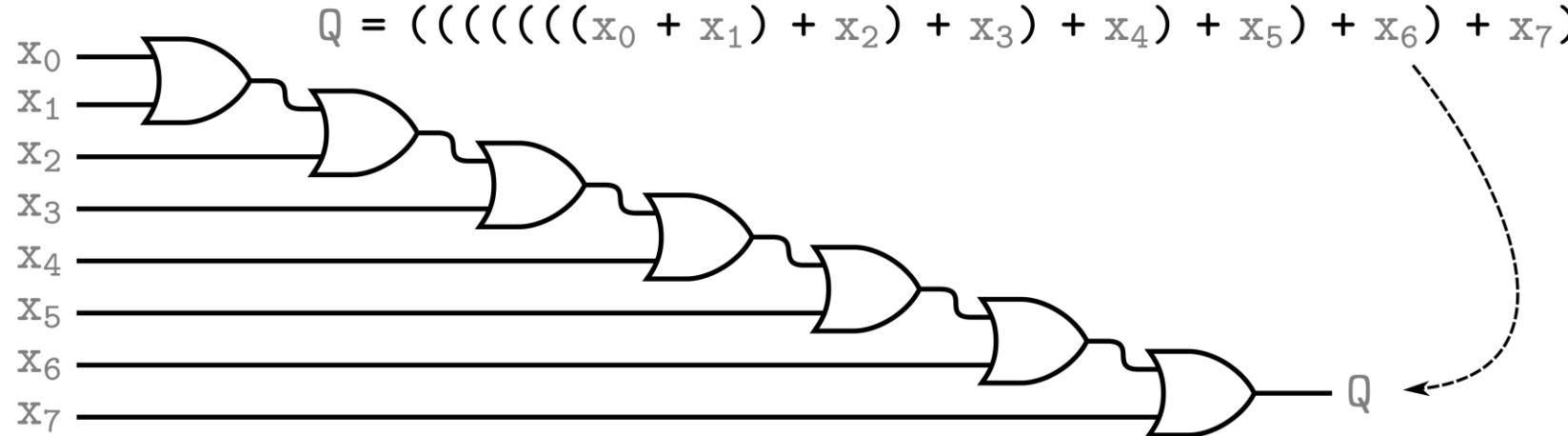
OR 8-vers-1



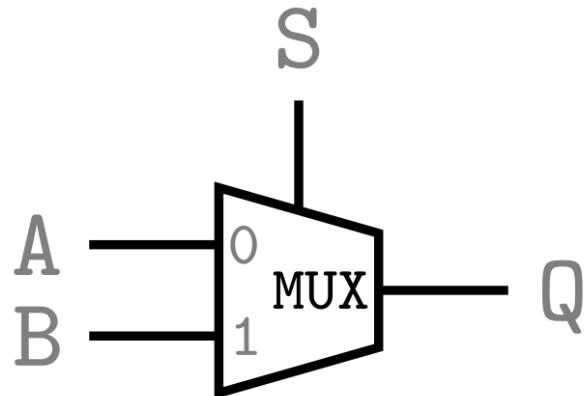
AND 8-vers-1



# Portes k-vers-1: Réalisation à plusieurs niveaux



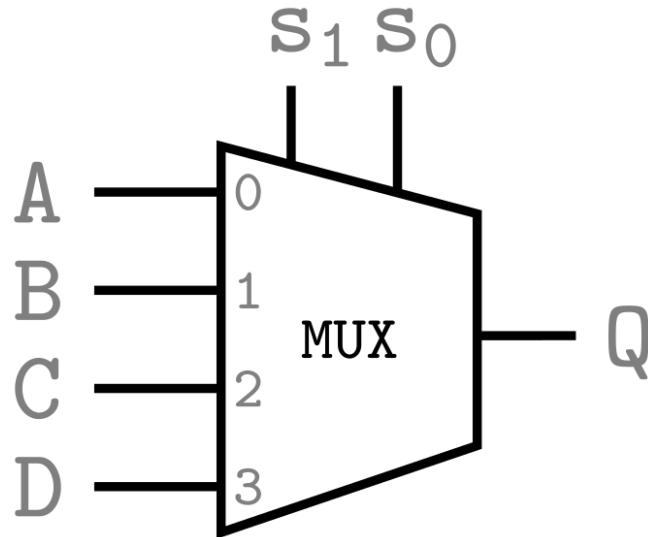
# Multiplexeur à 2 bits: rappel



*si  $S$  vaut 0  
 $Q = A$   
autrement  
 $Q = B$*

Commentaire: réalisation, au niveau électronique,  
d'une structure de contrôle sélective (if-else).

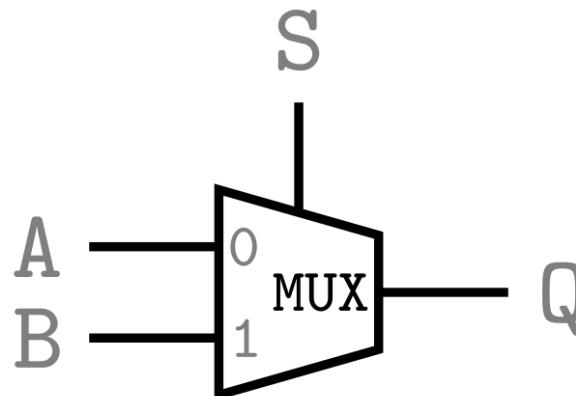
# Multiplexeurs à 4 bits



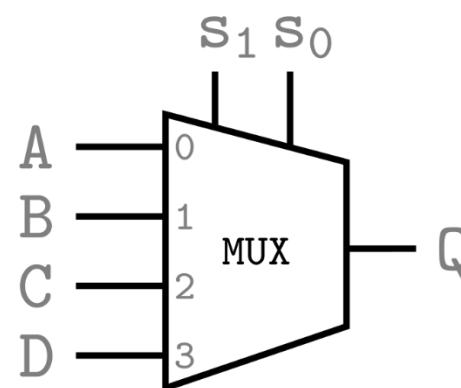
Les deux bits de sélection  
représentent un nombre entre 0 et 3,  
qui indique la ligne à sélectionner.

# Multiplexeurs à n bits

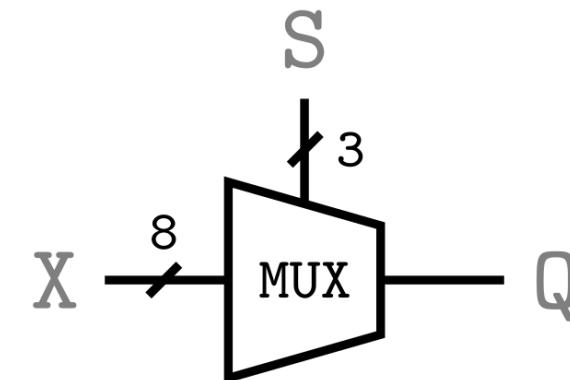
Multiplexeur à deux bits:



Multiplexeur à quatre bits:



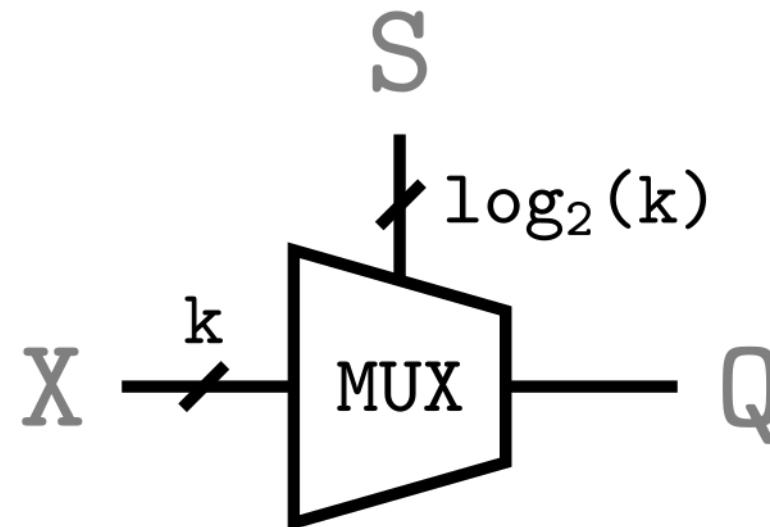
Multiplexeur à 8 bits:



# Multiplexeurs à n bits

Multiplexeur à k bits:

La ligne S a une largeur de  $\log_2(k)$  bits.

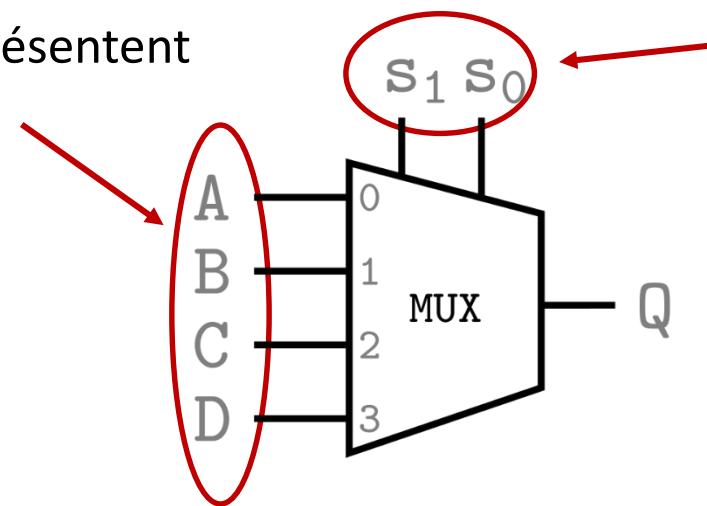


# Lignes de données et lignes de contrôle



Les entrées ne s'utilisent pas toutes dans le même but.

Les **lignes de données** représentent des entrées «classiques».



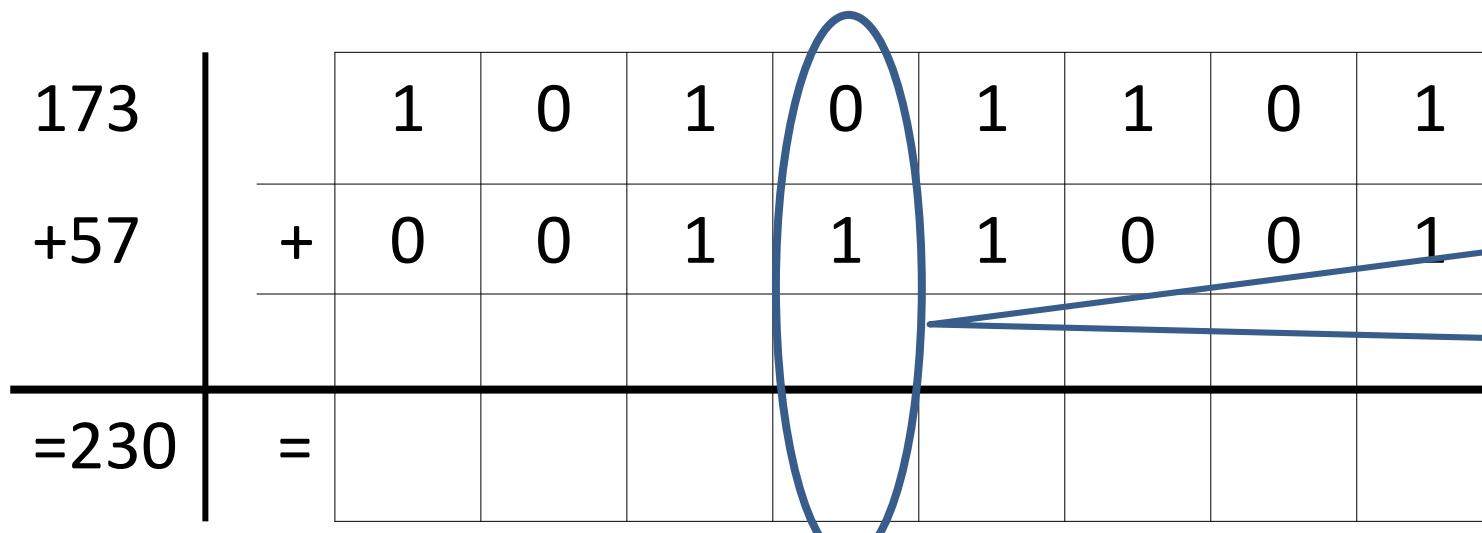
Les **lignes de contrôle** permettent de modifier le comportement du circuit. Dans le multiplexeur: lignes de **sélection** pour les lignes de données.



---

# Un circuit important: l'additionneur binaire

# Addition de deux nombres

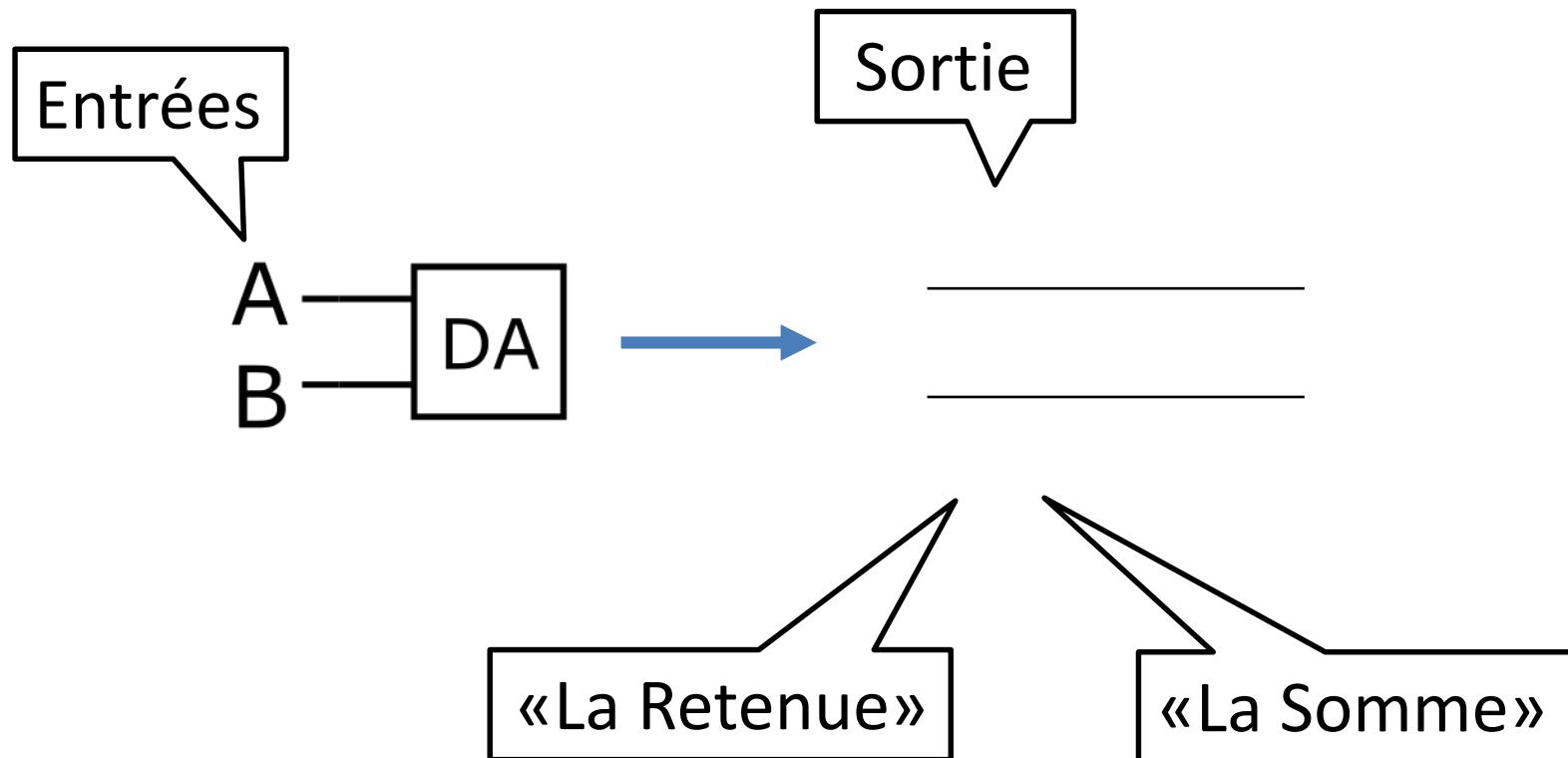


- Chaque colonne:  
Addition de 3 bits
- Calculé par un **circuit additionneur**

# Demi-Additionneur: additionne 2 bits

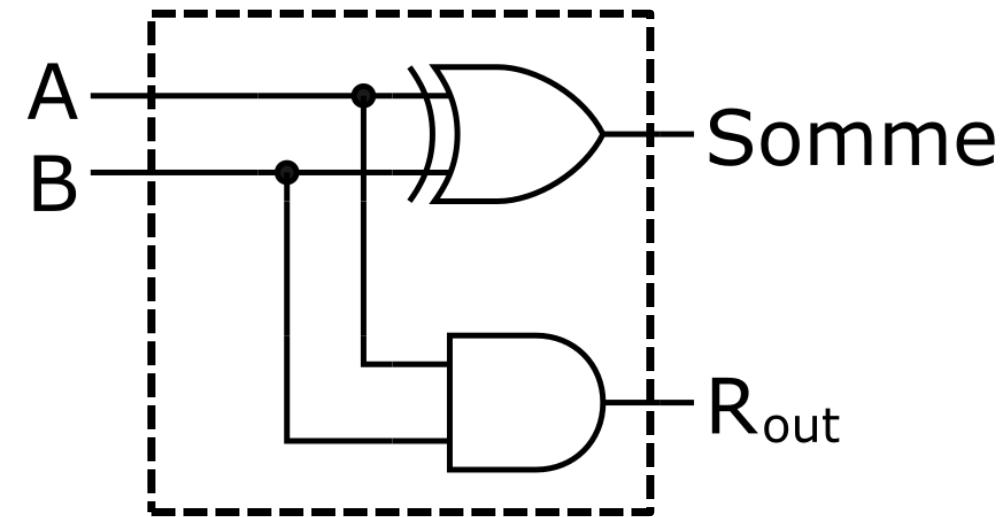
**Entrée:** Deux valeurs binaires A, B

**Sortie:** La somme arithmétique des entrées.



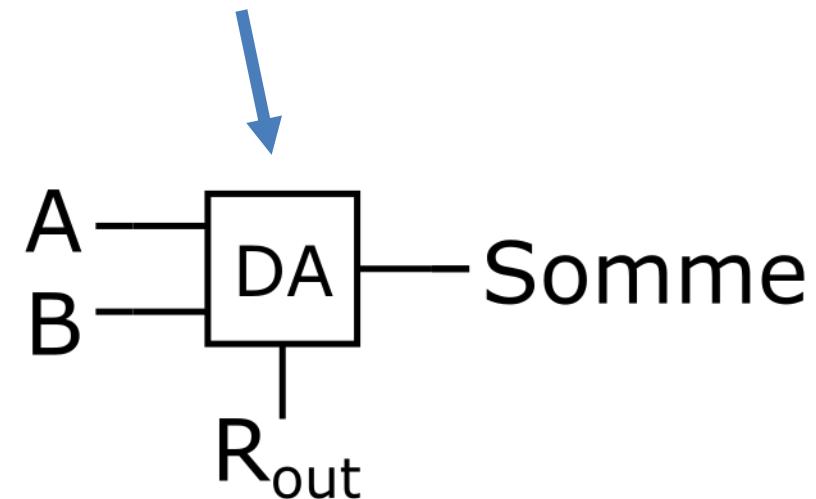
# Demi-Additionneur

A	B	Retenue	Somme
0	0		
0	1		
1	0		
1	1		



$$\text{Somme} = A \oplus B$$

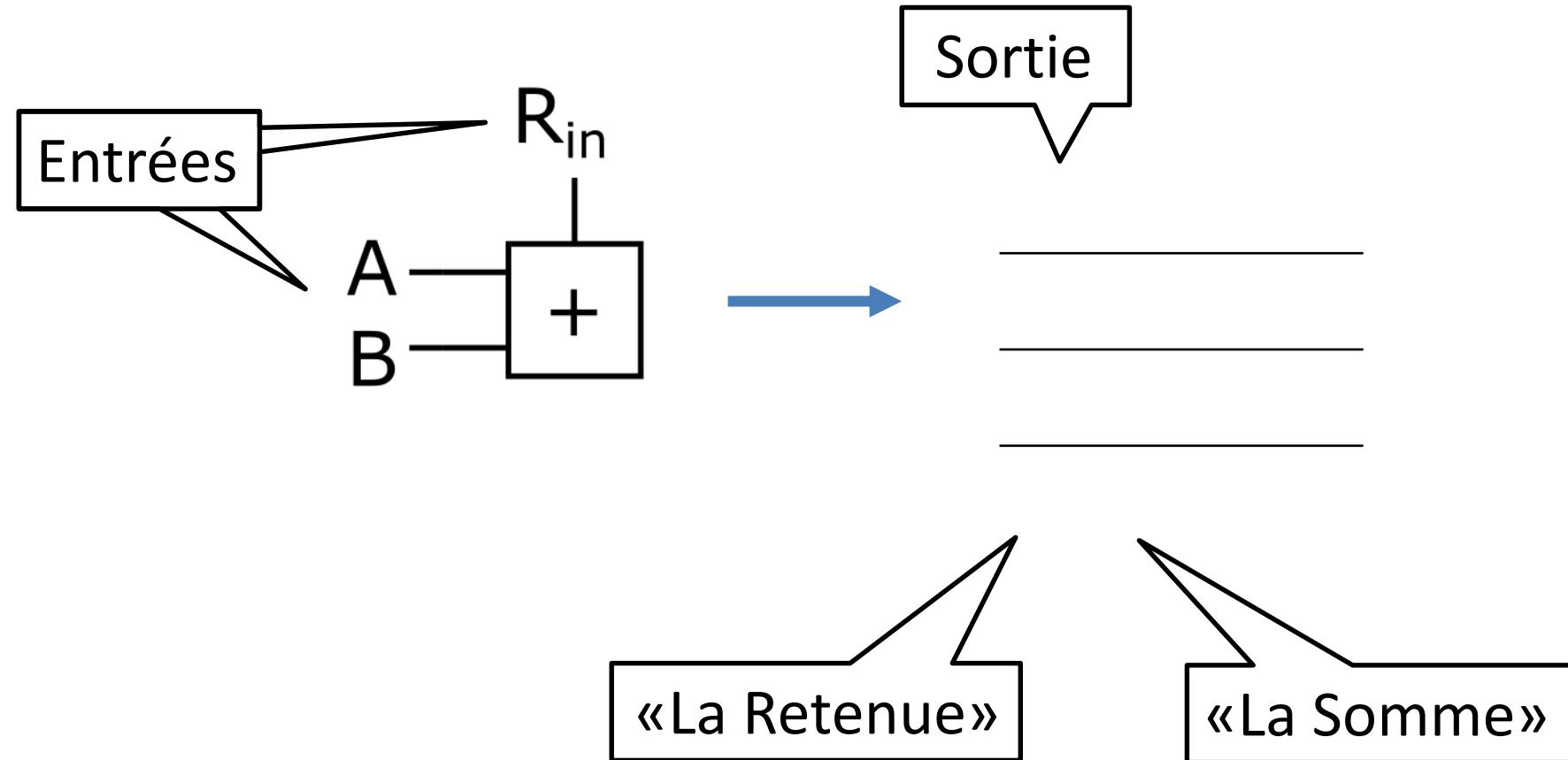
$$\text{Retenue} = AB$$



# Additionneur complet: additionne 3 bits

**Entrée:** Trois valeurs binaires A, B,  $R_{in}$

**Sortie:** La somme arithmétique des entrées.

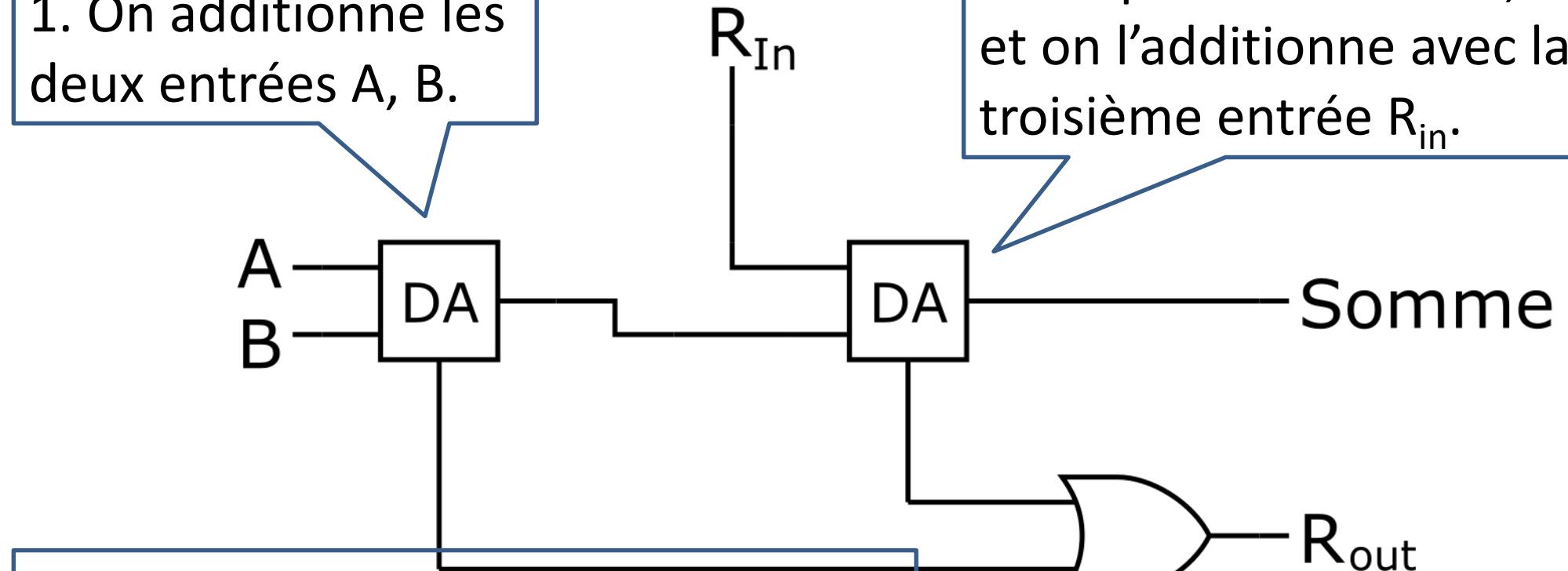


# Additionneur complet: circuit

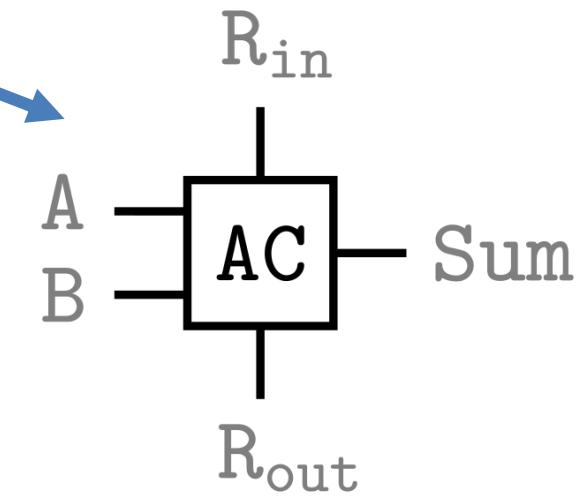
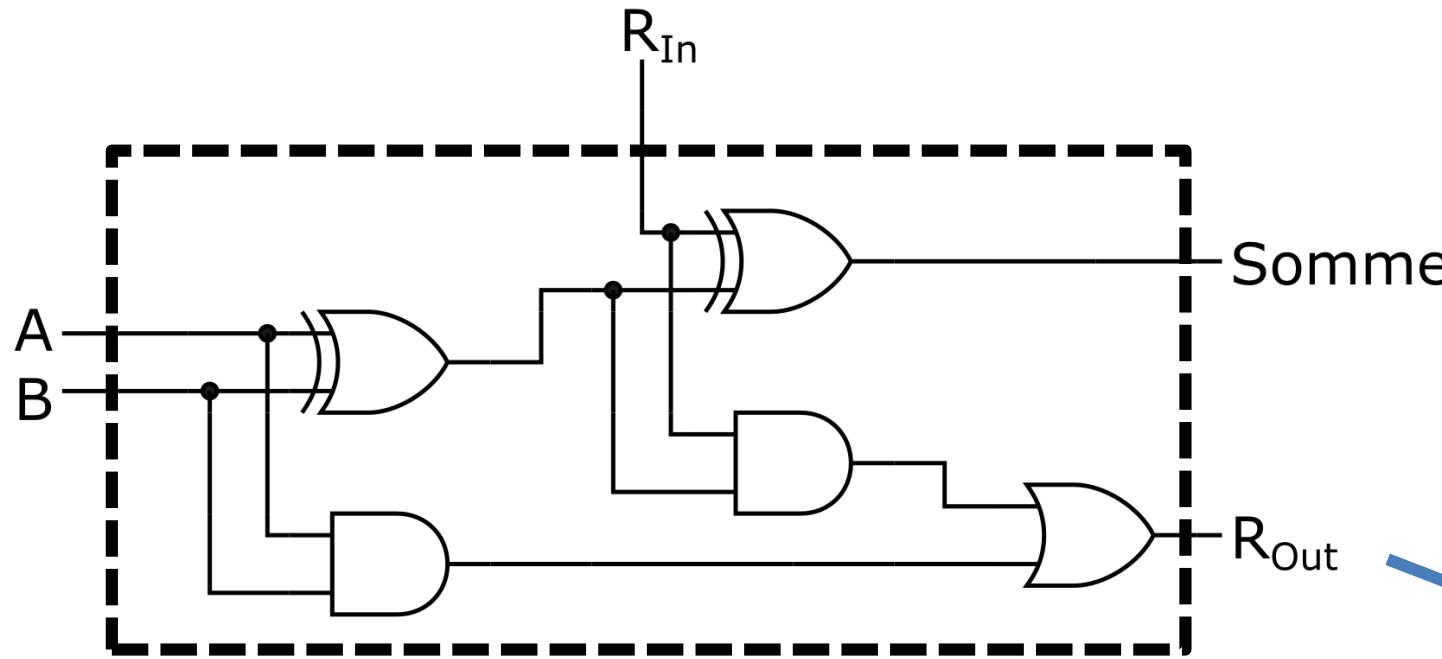
1. On additionne les deux entrées A, B.

3. On prend les deux retenues produites et on les somme (une de ces deux valeurs est toujours nulle).

2. On prend la somme, et on l'additionne avec la troisième entrée  $R_{in}$ .

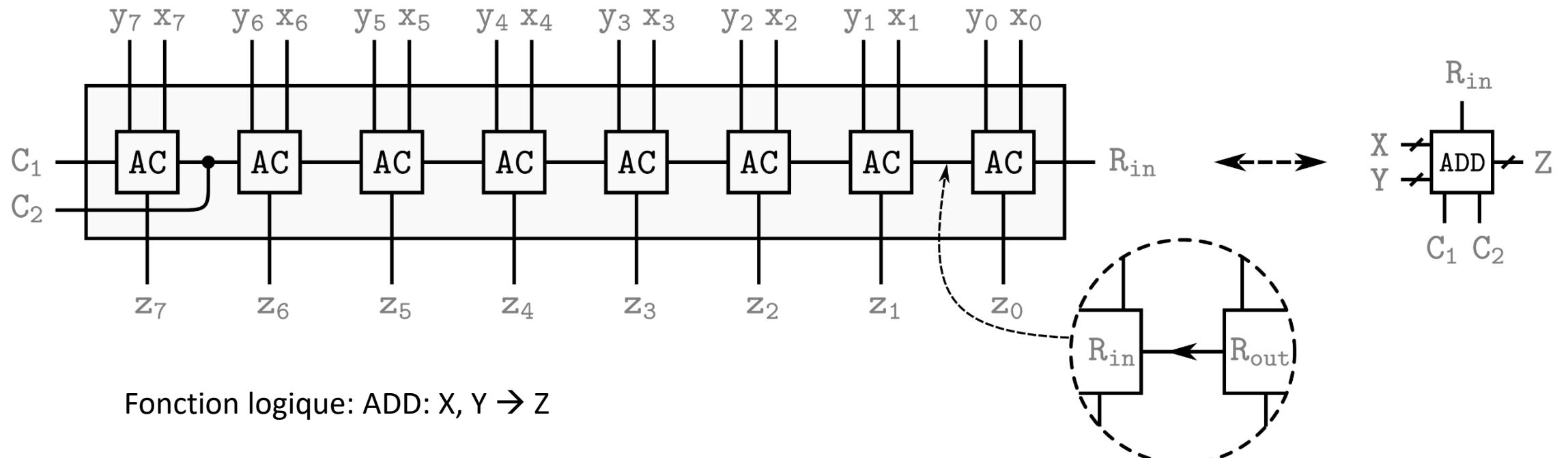


# Additionneur complet: symbole



# Additionneur à 8 bits

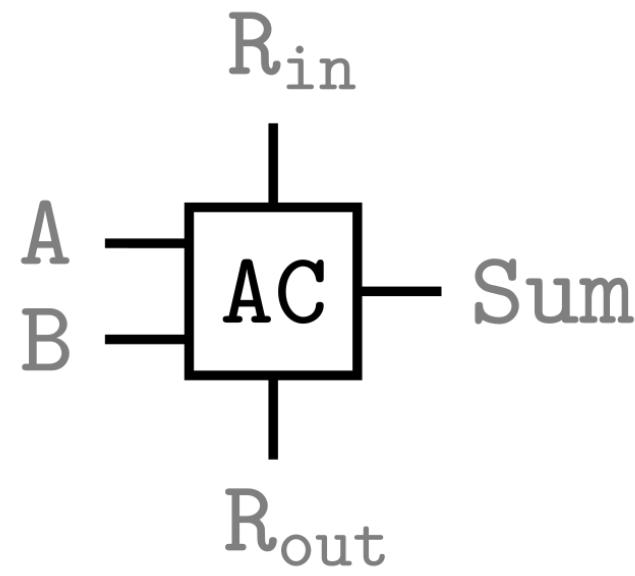
- L'additionneur k-bits enchaîne plusieurs additionneurs complets.
- Dépendance entre les composants: Chaque additionneur à 1-bit connecte sa sortie  $R_{out}$  à l'entrée  $R_{in}$  du prochain additionneur.



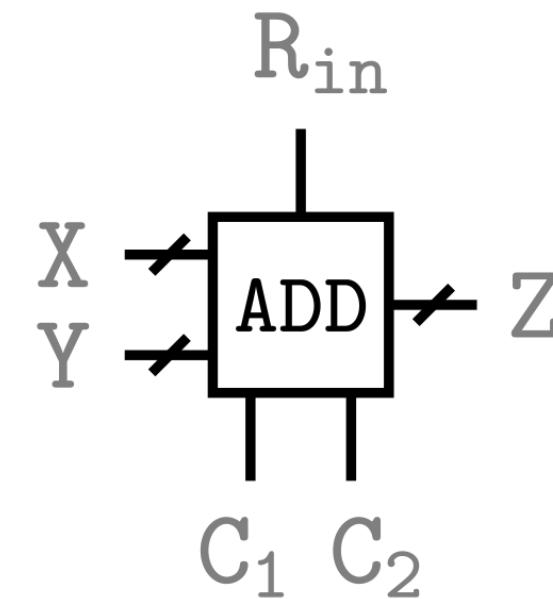
# Notation: additionneur à k bits



Rappel: L'additionneur complet à un bit:

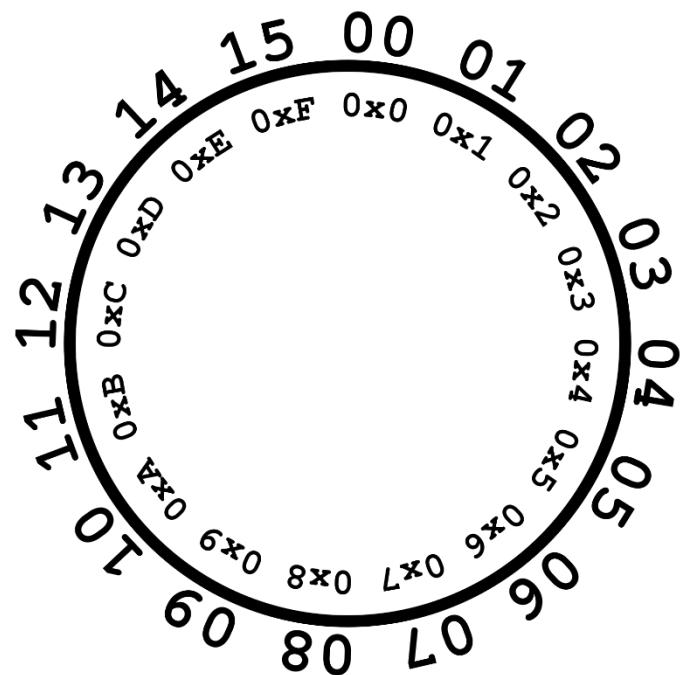


L'additionneur à k bits:

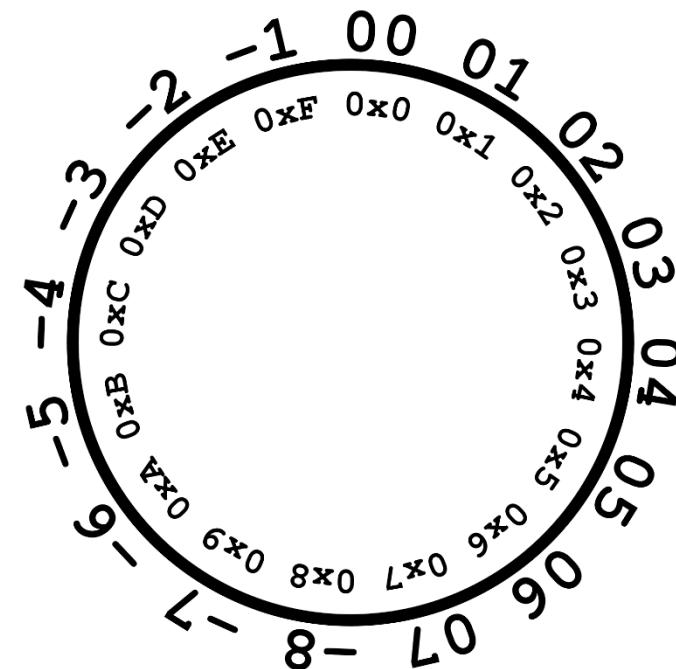


# Comment détecter les débordements?

«Carry Flag CF1»



En  $\mathbb{N}_{(k)}$ , un débordement a lieu lorsqu'on dépasse  $2^k - 1$ .



En  $\mathbb{Z}_{(k)}$ , un débordement a lieu lorsqu'on dépasse  $2^{k-1} - 1$ .



# Soustraction

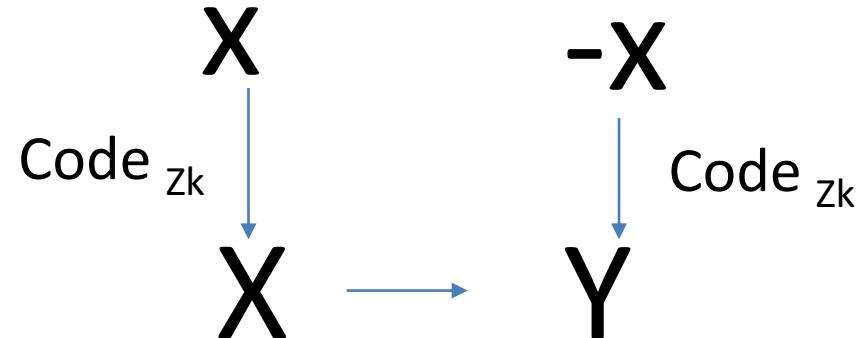
---

## Calcul de $y - x$

- Idée: on prétend que  $x$  et  $y$  sont codés en  $\mathbb{Z}_{(k)}$
- On peut donc calculer la représentation binaire de  $-x$
- Le reste est une simple addition, car

$$y - x = y + (-x)$$

# Calcul du complément à 2



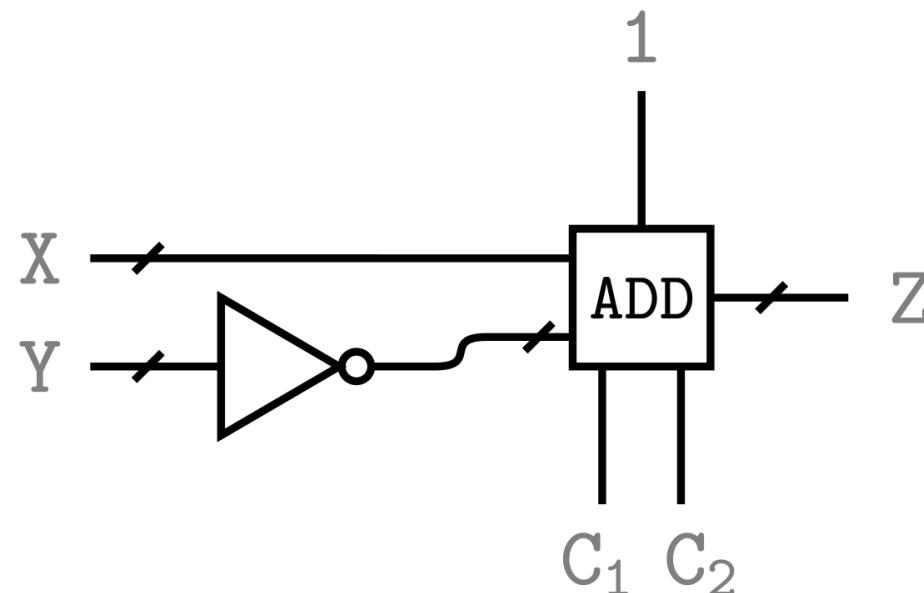
Calcul du complément à 2:

1. Inversion de tous les bits
2. Addition de la valeur 1 au résultat

# Soustraction de deux entiers

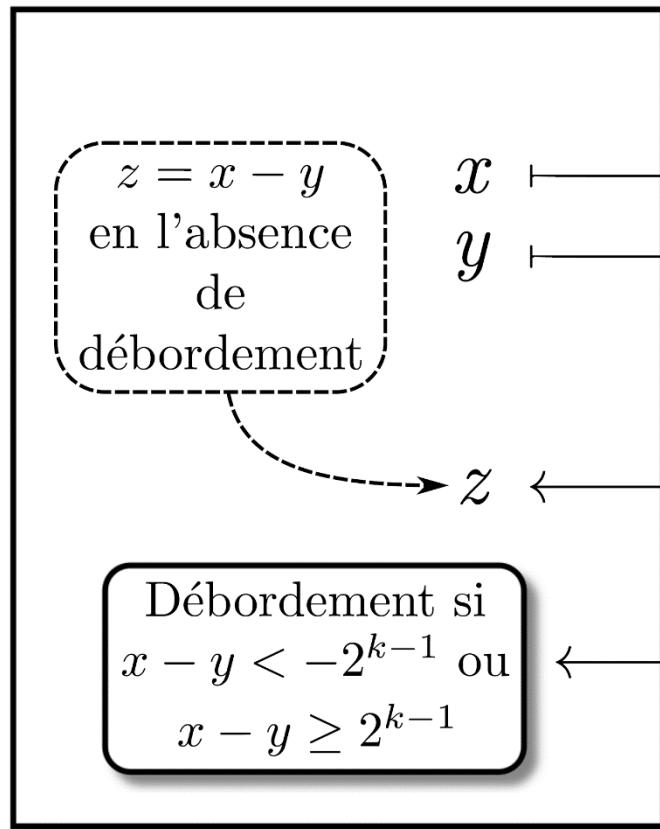


- Soustraction à l'aide du circuit additionneur:
  - Appliquer un NOT au deuxième argument.
  - Additionner la valeur 1.
- Astuce pour ajouter la valeur 1: mettre à 1 la retenue d'entrée de l'additionneur (qui de toute façon ne servait à rien...):

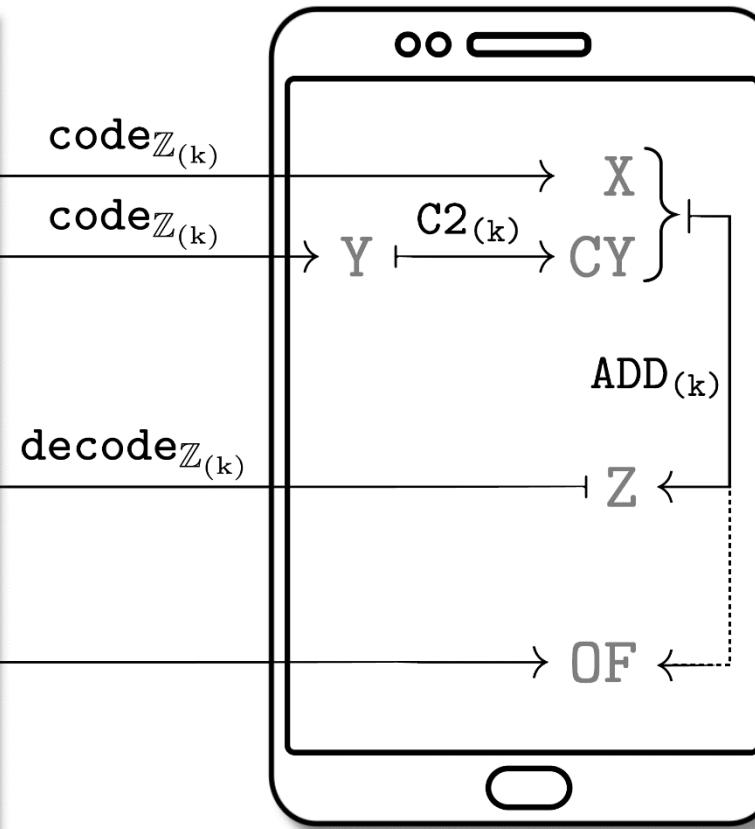


# Soustraction à l'aide du circuit additionneur

Représentation externe:  $\mathbb{Z}_{(k)}$



Représentation interne: mots à k bits



Soustraction  $x - y$  en  $\mathbb{Z}_{(k)}$



# Détection de débordements: Résumé

Espace	Calcul	Opération logique	Débordement (Overflow Flag OF)



# Détection de débordements: Résumé

Espace	Calcul	Opération logique	Débordement
$\mathbb{N}_{(k)}$	$c = a + b$	$\text{ADD}(\text{code}_{\mathbb{N}_{(k)}}(a), \text{code}_{\mathbb{N}_{(k)}}(b))$	$\text{OF} = \text{CF1}$
$\mathbb{Z}_{(k)}$	$c = a + b$	$\text{ADD}(\text{code}_{\mathbb{Z}_{(k)}}(a), \text{code}_{\mathbb{Z}_{(k)}}(b))$	$\text{OF} = \text{CF1} \oplus \text{CF2}$
$\mathbb{N}_{(k)}$	$c = a - b$	$\text{ADD}(\text{code}_{\mathbb{N}_{(k)}}(a), \text{C2}(\text{code}_{\mathbb{N}_{(k)}}(b)))$	$\text{OF} = !\text{CF1}$
$\mathbb{Z}_{(k)}$	$c = a - b$	$\text{ADD}(\text{code}_{\mathbb{Z}_{(k)}}(a), \text{C2}(\text{code}_{\mathbb{Z}_{(k)}}(b)))$	$\text{OF} = \text{CF1} \oplus \text{CF2}$



# Multiplication / Division

---

- Comme dans le cas de l'addition / soustraction, on suit le même procédé que lors d'un calcul manuel sur papier.
- La multiplication s'exprime par une succession d'additions ( $\log_2(k)$  étapes pour des mots à k bits).
- La division s'exprime par une succession de soustractions ( $\log_2(k)$  étapes pour des mots à k bits).
- Nous n'allons pas présenter ces procédés dans ce cours, par manque de temps.

# Multiplication / Division par puissances de 2

---



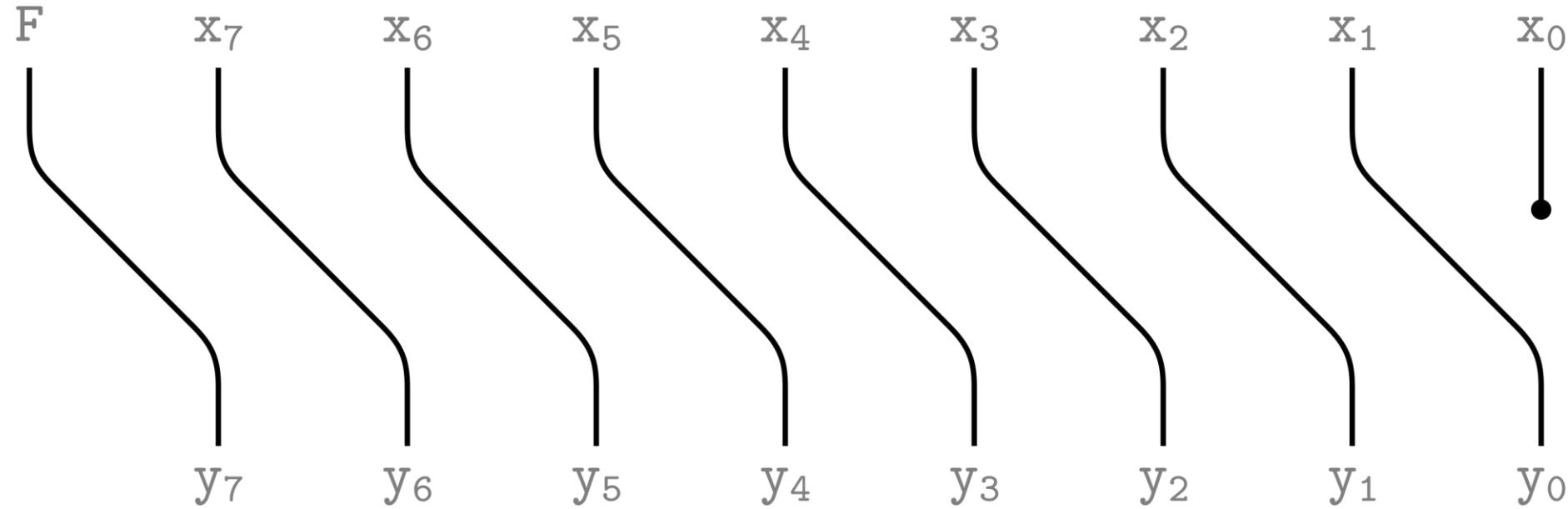
Base 10: Multiplication / Division par une puissance de 10

- $00001234 * 10^3 = 01234000$
- $00001234 / 10^3 = 00000001$  (arrondi vers bas)

Base 2: Multiplication / Division par une puissance de 2

- $00001010_{(2)} * 2^3 = 01010000 \text{ (2)}$
- $00001010_{(2)} / 2^3 = 00000001 \text{ (2) (arrondi vers le bas)}$

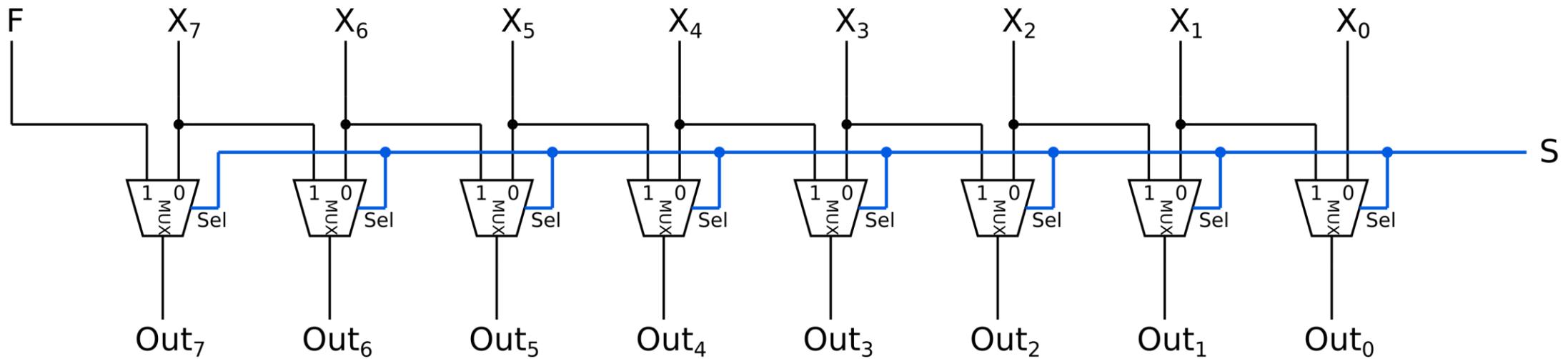
# Multiplication / Division par puissances de 2



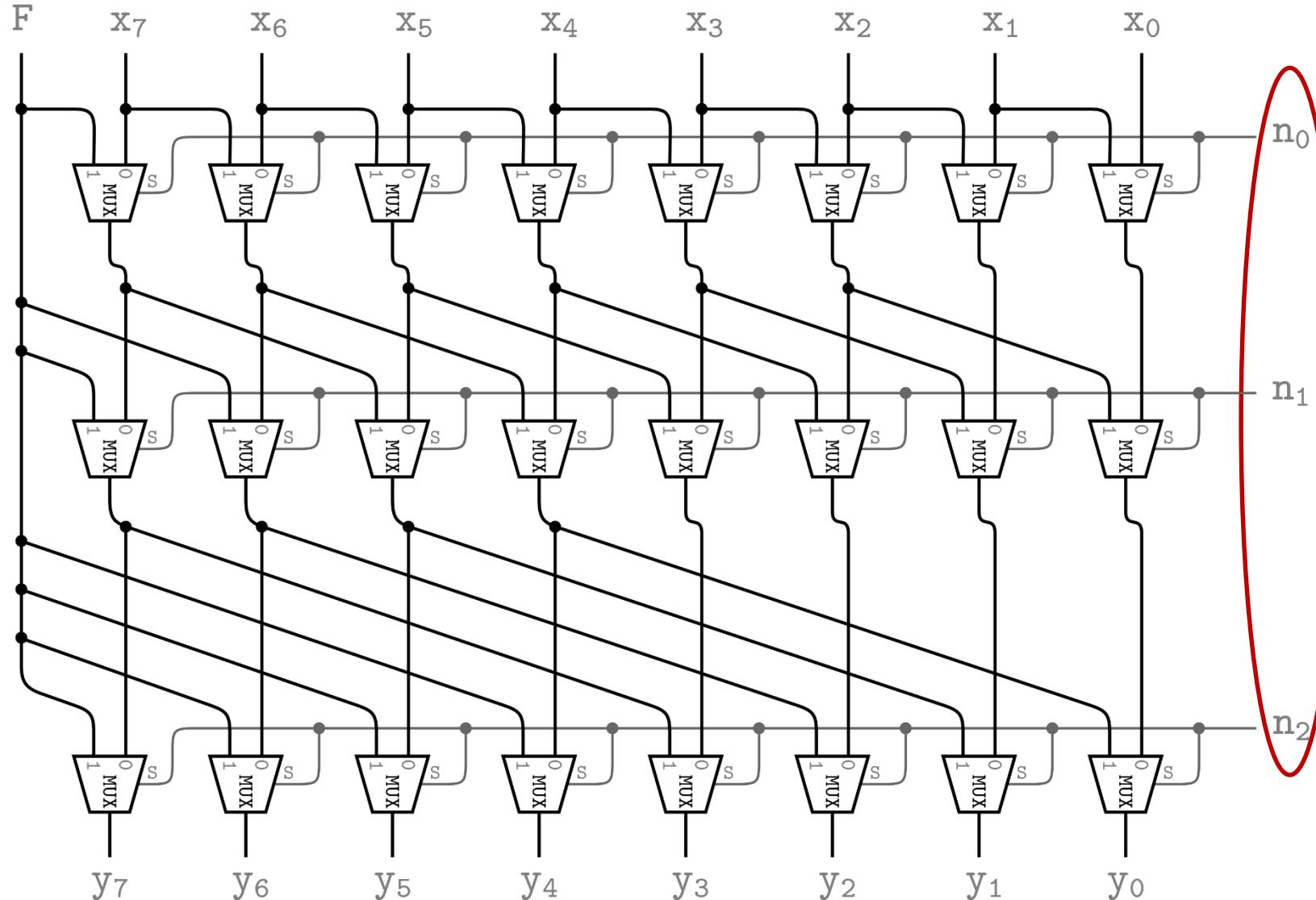
- Le circuit «décalage à droite d'une position».
- Si  $F=0$ : Division entière par 2.

# Généralisation du circuit de décalage

Ce circuit décale d'une position à droite, ou ne fait rien, dépendant de la valeur de S.



# Barrel Shifter: Décalage de n bits



Décalage 1 bit (ou 0) à droite

Décalage 2 bits (ou 0) à droite

Décalage  $n$  bits  
 $n = n_0 \cdot 1 + n_1 \cdot 2 + n_2 \cdot 4$

Décalage 4 bits (ou 0) à droite

**$n$  en base binaire**



# Applications du barrel shifter

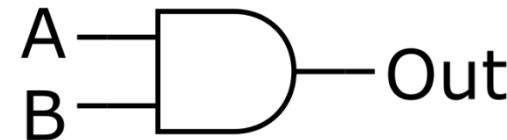
---

- Multiplication et division par puissances de 2.
- Unité de contrôle: décomposition d'une instruction en code machine.
- Algorithmes de chiffrage.
- Protocoles de communication: contrôles de redondance cyclique.



# Délais de propagation

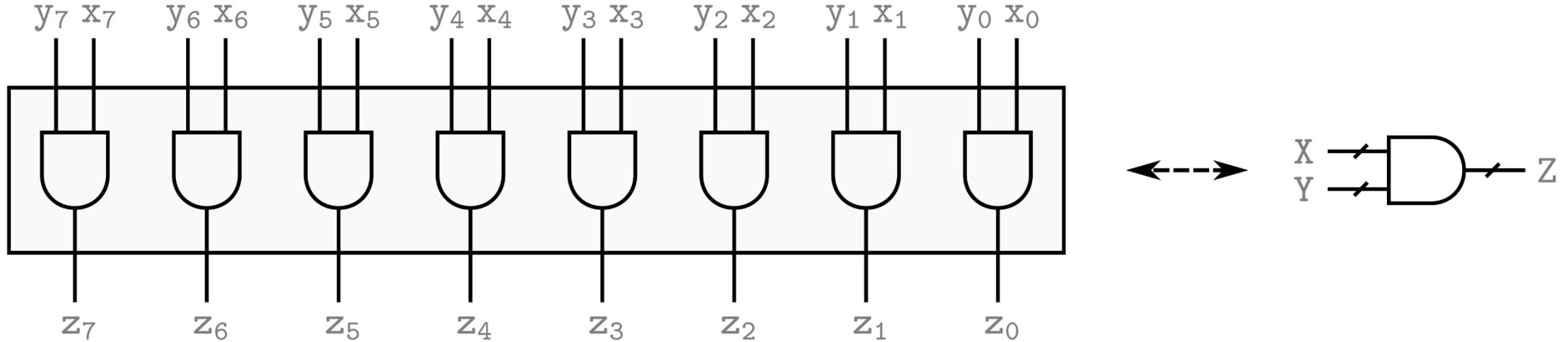
**Théorie:** Les sorties d'un circuit combinatoire dépendent des entrées. Il n'y a pas de notion du temps.



**Pratique:** Quand les entrées changent, il s'écoule un certain laps de temps avant que les sorties reprennent une valeur stable. On parle du **délai de propagation d'un circuit**.

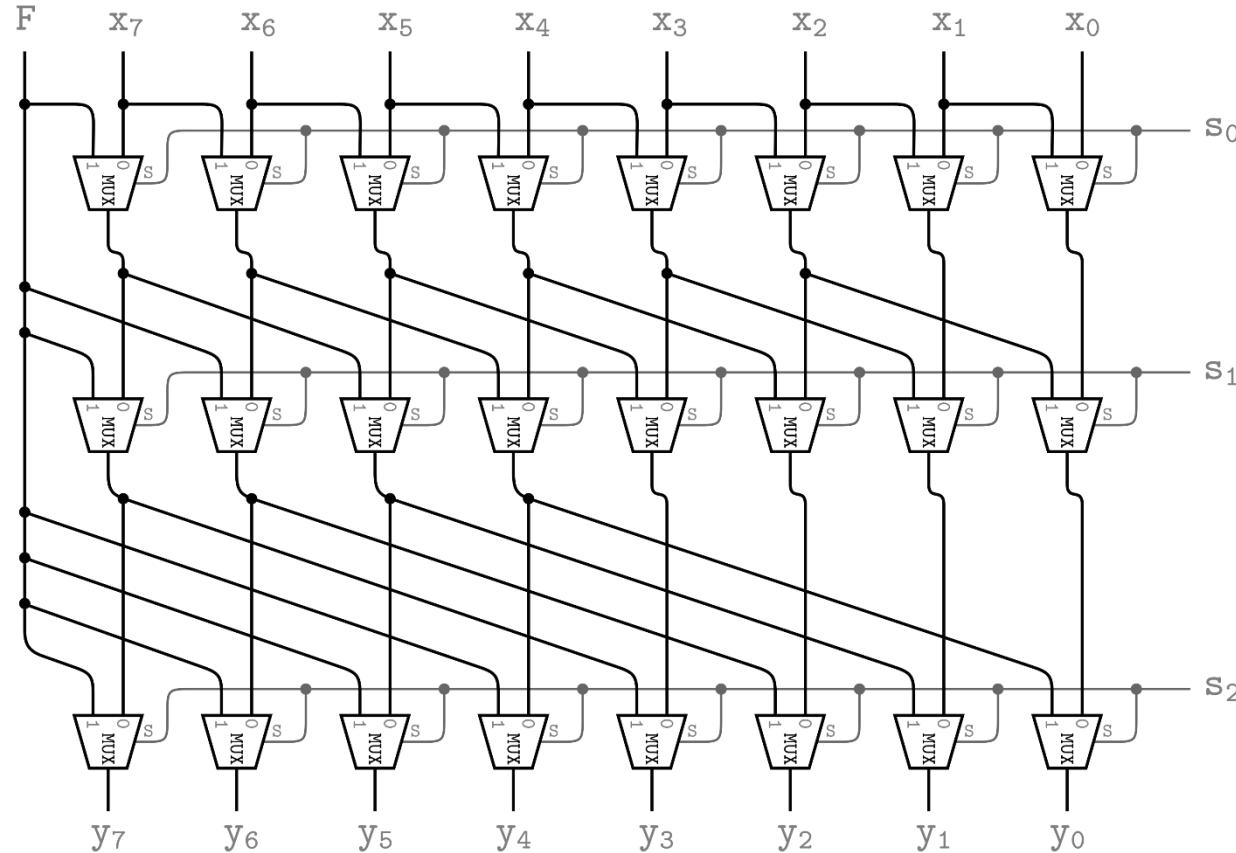
Le délai de propagation dépend du **nombre de portes** qu'un signal doit traverser entre l'entrée et la sortie du circuit.

# Délai de propagation d'une porte k-bit



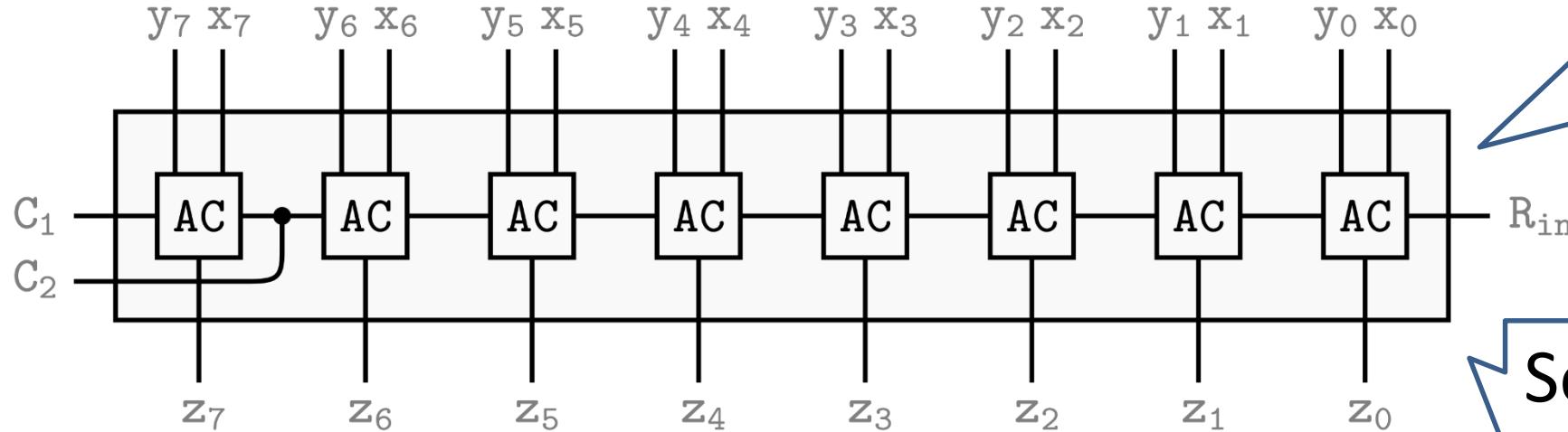
- Toutes les portes travaillent en parallèle.
- Le délai de propagation du circuit est égal au délai d'une porte individuelle.

# Délai de propagation du barrel shifter



- On a  $\log_2(k)$  niveaux
- Le délai de propagation est proportionnel à  $\log_2(k)$ .

# Délai de propagation de l'additionneur



Additionneur à propagation de retenue.

Solution:  
Additionneur à retenue anticipée.

- Chaque additionneur doit attendre la retenue de l'add. précédent.
- Le délai de propagation est proportionnel à  $k$ .