Introduction à l’informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

UNIVERSITÉ DE GENÈVE

CENTRE UNIVERSITAIRE D’INFORMATIQUE

Chapitre 4
Algèbre de Boole et circuits logiques (partie 2)

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

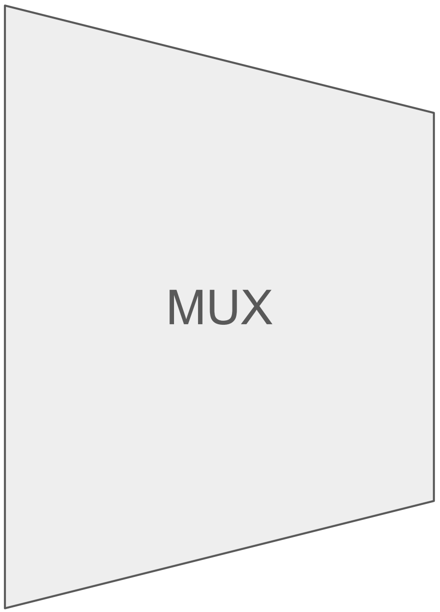
Chapitres du cours

1. Origines des ordinateurs et des réseaux informatiques
2. Codage des nombres
3. Codage des médias
4. **Circuits logiques** ←
5. Architecture des ordinateurs
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Circuits combinatoires

- Circuits combinant plusieurs portes logiques (déjà vu !)
- Limite entre porte logique et circuit logique dépend des auteurs.
- Nous allons ici voir deux circuits combinatoires importants :
  - Le multiplexeur
  - L'additionneur

Le multiplexeur



Multiplexeur à 2 voies

- Il est courant de vouloir exprimer des **conditions** du type : "*Si S est faux, alors la valeur de sortie est A, sinon la valeur de sortie est B.*"

- Permet d'effectuer un **choix** entre A et B grâce à S.

- Équivalent électronique du if/else de programmation :

```
if S == 0:
    Out = A
else:
    Out = B
```

Multiplexeur à 2 voies

- Il est courant de vouloir exprimer des conditions du type : "*Si S est faux, alors la valeur de sortie est A, sinon la valeur de sortie est B.*"

- Permet d’effectuer un choix entre A et B grâce à S.

- Bien entendu, on peut exprimer cela au travers d’une table de vérité.
  Trois entrées : S, A, B.
  Une sortie : Out.

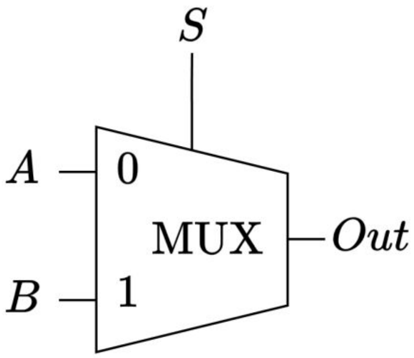
- Vocabulaire : S est la **ligne de contrôle**, A et B sont les **lignes de données**.

Multiplexeur à 2 voies

- On décide que S=0 correspond à la sélection de A et S=1 correspond à la sélection de B.

- On obtient en sortie une copie de l'entrée sélectionnée.

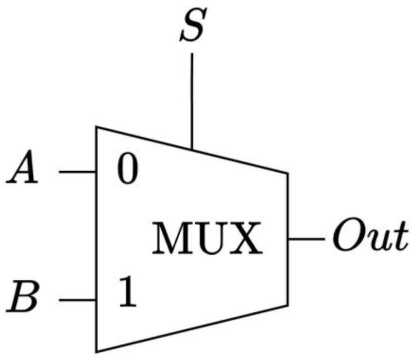
- Symbole :



S	A	B	Out

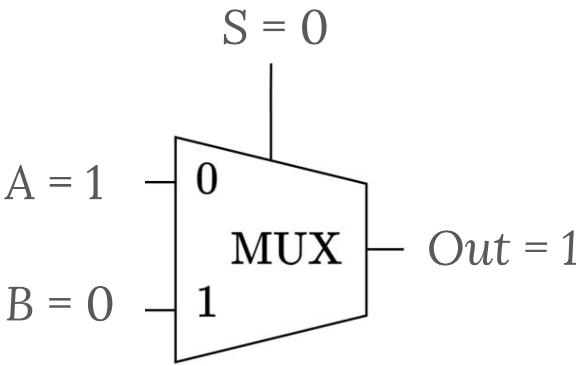
Multiplexeur à 2 voies

- On décide que $S=0$ correspond à la sélection de A et $S=1$ correspond à la sélection de B.
- On obtient en sortie une copie de l'entrée sélectionnée.
- Symbole :

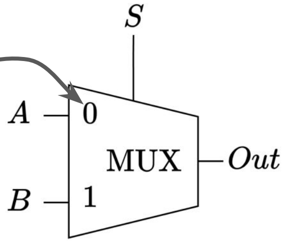


S	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexeur à 2 voies | **Exemple**



Ne veut **pas**
dire que $A = 0$

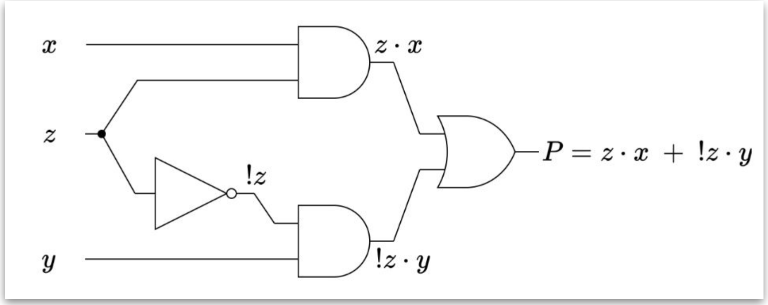


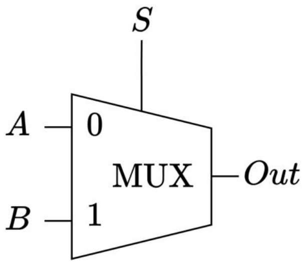
S	A	B	Out
0	1	0	1

Multiplexeur à 2 voies

- Cette table de vérité est équivalente à celle du problème du nombre premier à 3 bits !

- S → z, A → y, B → x, Out → P

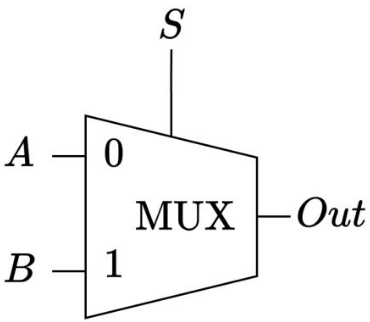




S	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexeur à 2 voies

$Out = SB + !SA$

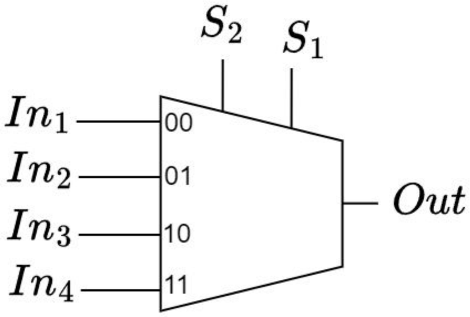


S	A	B	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexeur à *k* voies

Exemple pour *k* = 4 lignes de données.

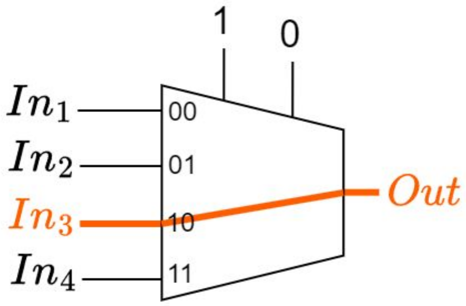
S	Out
00	In_1
01	In_2
10	In_3
11	In_4



Multiplexeur à *k* voies

Exemple pour *k* = 4 lignes de données.

S	Out
00	In_1
01	In_2
10	In_3
11	In_4



Multiplexeur à *k* voies | **Taille de la ligne de contrôle**

- Sélectionne l'une des *k* entrées, selon le signal de contrôle *S*.

- Question : quelle doit être la taille (nombre de bits) de *S* afin de pouvoir sélectionner n'importe laquelle des *k* entrées ?

- Réponse :

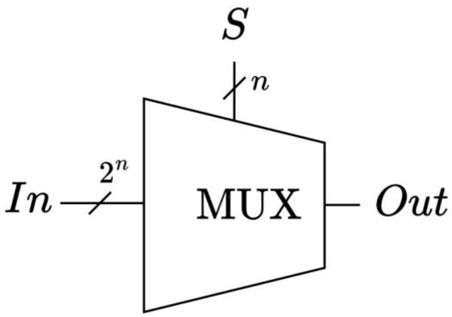
- Exemple :

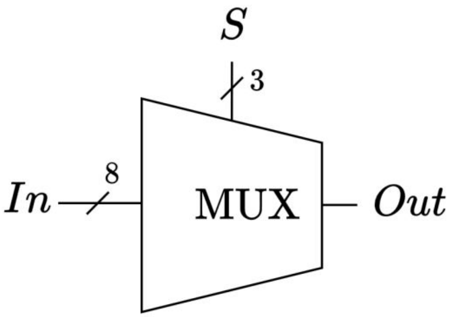
Multiplexeur à *k* voies | **Taille de la ligne de contrôle**

- Sélectionne l’une des *k* entrées, selon le signal de contrôle *S*.
- Question : quelle doit être la taille (nombre de bits) de *S* afin de pouvoir sélectionner n’importe laquelle des *k* entrées ?
- Réponse : il faut $\log_2(k)$ bits. Autrement dit, si la ligne de contrôle comprend *n* bits, on peut sélectionner une entrée parmi $k = 2^n$.
- Exemple : avec 3 bits pour la sélection, on peut avoir 8 voies.

Multiplexeur à *k* voies | **Symboles**

Plutôt que de dessiner toutes les voies sur les schémas, on indique le nombre de bits concernés avec une barre oblique. Exemples :



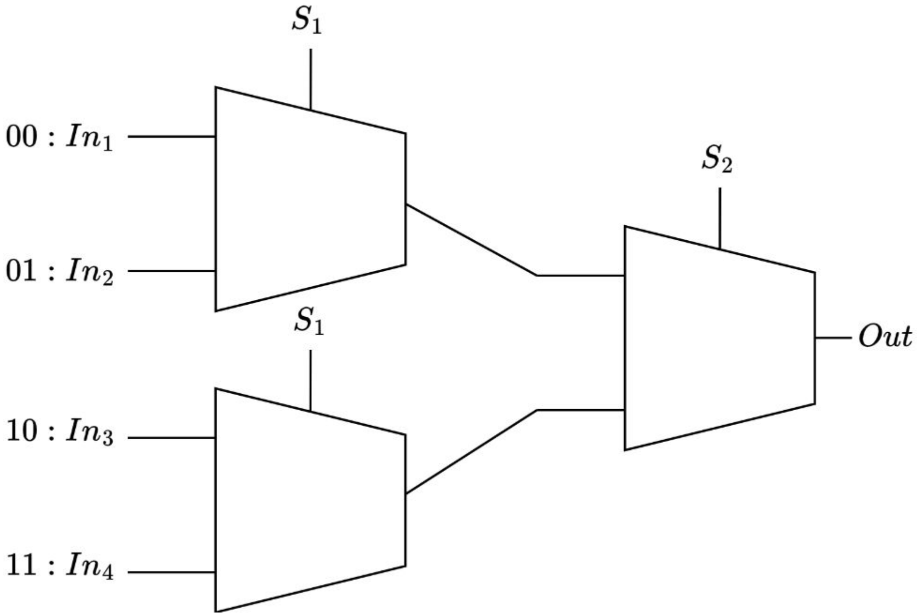


Multiplexeur à *k* voies | **Parcours du signal**

Question : combien de portes logiques le signal doit-il traverser pour générer un output ?

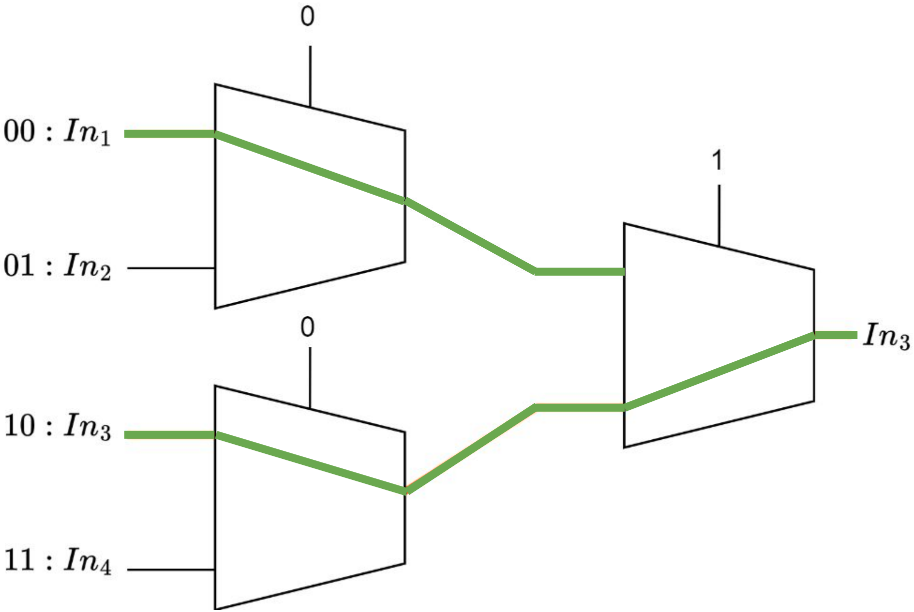
Multiplexeur à *k* voies | **Parcours du signal - Intuition**

Question : combien de portes logiques le signal doit-il traverser pour générer un output ?



Multiplexeur à *k* voies | **Parcours du signal**

Question : combien de portes logiques le signal doit-il traverser pour générer un output ?



Multiplexeur à *k* voies | **Parcours du signal**

- Question : combien de portes logiques le signal doit-il traverser pour générer un output ?

- Réponse : ce nombre est proportionnel au nombre de bits de *S*.

- Raison : multiplexeur à *k* voies peut être vu comme un arbre de multiplexeurs à 2 voies. Le nombre d'étages de cet arbre est de $\log_2(k)$.

Multiplexeur à *k* voies | **Parcours du signal**

- Le signal doit traverser un nombre de portes proportionnel à log(*k*) ou encore au nombre de bits de *S*.

- Ce fait trouvera son importance plus tard dans le cours, car des multiplexeurs sont utilisés pour sélectionner des données dans la mémoire de l’ordinateur.

  La vitesse d’accès aux éléments de la mémoire est d’autant plus lente que les adresses sont longues.

Multiplexeur à *k* voies

- Question : quelle est le nombre de lignes de la table de vérité d'un multiplexeur à 4 voies ?

- Début de la table de vérité pour *k* = 4 voies :

S_1	S_2	In_1	In_2	In_3	In_4	Out
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0

(Diapositive hors champ)

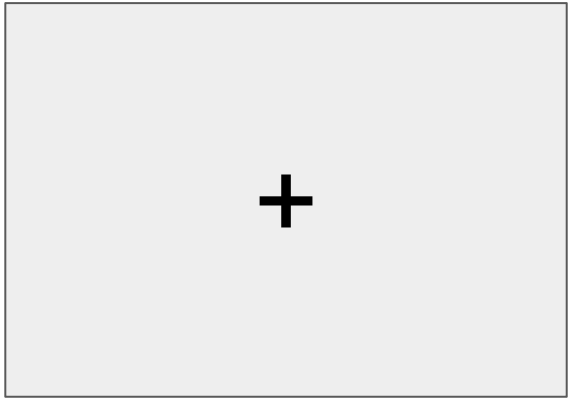
Multiplexeur à k voies

- Question : quelle est le nombre de lignes de la table de vérité d'un multiplexeur à 4 voies ?

- Réponse : il y a $k + \log_2(k)$ colonnes, donc le nombre de lignes est : $2^{k + \log2(k)} = k \cdot 2^k$

(Diapositive hors champ)

L'additionneur



Le circuit additionneur

- Objectif : prendre deux mots (états binaires) en input, et obtenir un mot en output qui représente la somme des inputs (interprétés comme des entiers).

- Inputs :
    - Bits du nombre a : a_0, a_1, … ,a_{n-1}
    - Bits du nombre b : b_0, b_1, … ,b_{n-1}

- Outputs :
    - Bits de la somme s_0, s_1, … ,s_{n-1}
    - Bit de la retenue R.

Rappel sur l'addition binaire

- Exemple de $1001_2 + 0011_2$

1	0^1	0^1	1
0	0	1	1
1	**1**	**0**	**0**

- Difficulté : la retenue

Addition de deux bits | **Retenue**

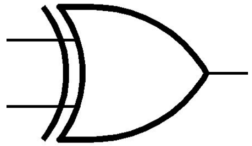
- Quatre cas possibles pour **une** colonne donnée.
  - 0 + 0 = **0**
  - 0 + 1 = **1**
  - 1 + 0 = **1**
  - 1 + 1 = 1**0**

Pas présent sur la colonne concernée

- La table de vérité pour l’addition sur **une** colonne correspond donc au XOR.

Addition de deux bits

- La table de vérité pour l’addition sur une colonne correspond donc au XOR.
- Ne prend pas en compte la retenue !
- XOR(a,b) = a $\oplus$ b

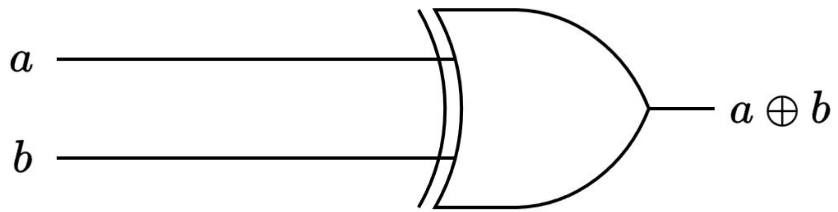


a	**b**	**a $\oplus$ b**
0	0	0
0	1	1
1	0	1
1	1	0

Addition de deux bits

Ne prend pas en compte la retenue. Comment modifier ce circuit pour également avoir un output de retenue ?

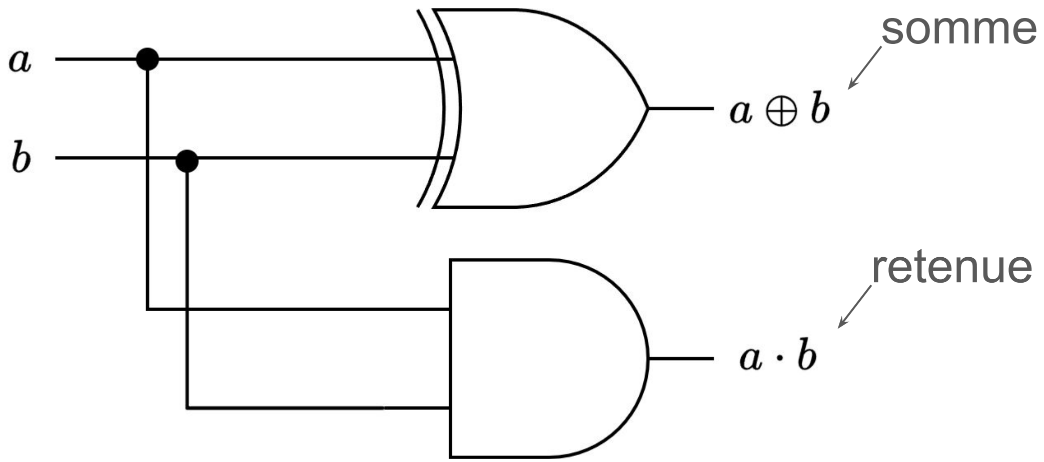
a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0



Addition de deux bits | **Prise en compte de la retenue**

- Pour le cas où *a* et *b* valent 1, une retenue existe !
- Circuit souvent nommé "demi-additionneur".
- Reste à la transmettre à la colonne de gauche, et récupérer celle de droite.

a	**b**	**a ⊕ b**	**ab**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

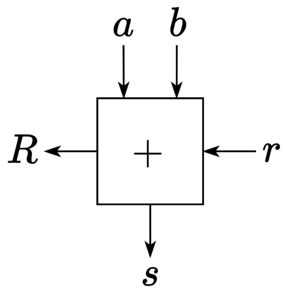


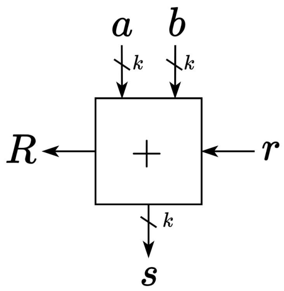
Additionneur | **Cas général (additionneur complet)**

- En plus de a_i et b_i, Le circuit associé à chaque colonne reçoit en input le reste de la colonne de droite, $r_i = R_{i-1}$.

- Le circuit de chaque colonne output la somme s_i et transmet le reste R_i à la colonne de gauche.

- Symbole de l'additionneur complet :



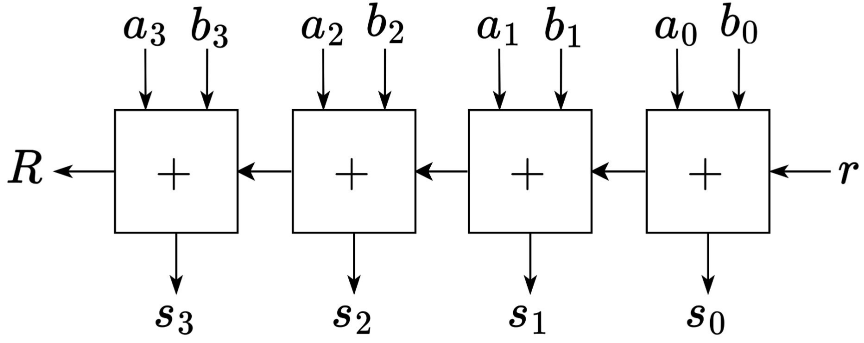


Additionneur | **Cas général**

- En plus de a_i et b_i, Le circuit associé à chaque colonne reçoit en input le reste de la colonne de droite, $r_i = R_{i-1}$.

- Le circuit de chaque colonne output la somme s_i et transmet le reste R_i à la colonne de gauche.

- Exemple à 4 bits :



Bit de débordement

Additionneur | **Table de vérité pour le cas général 1 bit**

r	**a**	**b**	**s**	**R**
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

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour R :
  - Soit *a* et *b* valent 1 tous les deux, peu importe *r* ;
  - Soit *r* vaut 1 et uniquement l'un des bits *a* et *b* vaut 1.

r	**a**	**b**	s	**R**
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**

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour R :
  - Soit *a* et *b* valent 1 tous les deux, peu importe *r* ;
  - Soit *r* vaut 1 et uniquement l'un des bits *a* et *b* vaut 1.

$\Rightarrow R = ab + r(a \oplus b)$

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour s :
  - Soit r vaut 0 et uniquement l'un des deux bits a et b vaut 1 ;
  - Soit r vaut 1 et a = b.

r	a	b	s	R
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

Additionneur | **Table de vérité pour le cas général**

- On trouve l'expression pour s :
  - Soit r vaut 0 et l'un des deux bits a et b vaut 1 (mais pas les deux) ;

    S'exprime $a \oplus b$

  - Soit r vaut 1 et a = b.

    S'exprime $!(a \oplus b)$

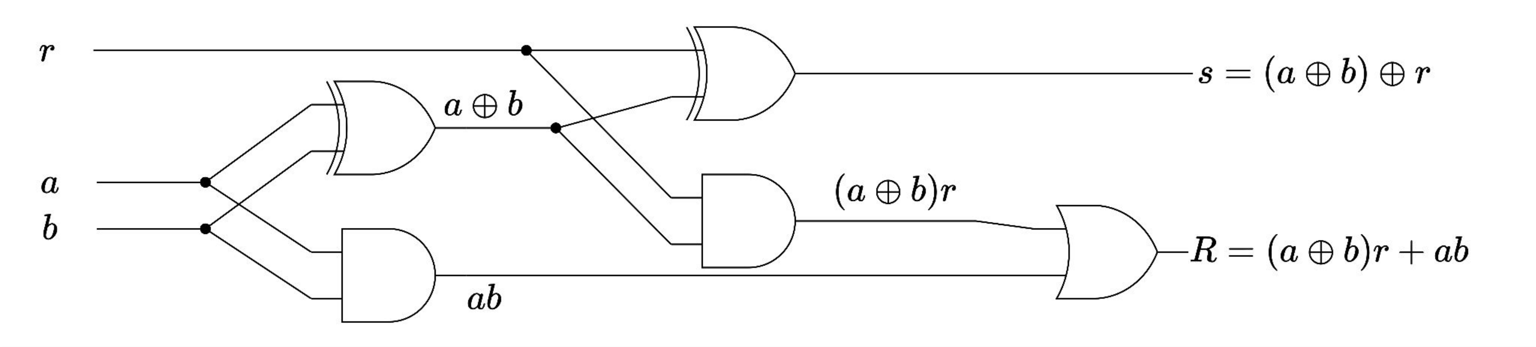
$\Rightarrow s = !r(a \oplus b) + r!(a \oplus b)$.

Mais comme $x \oplus y = x!y + !xy$, on remarque finalement que :

$s = r \oplus (a \oplus b)$

Additionneur | **Additionneur complet**

- R = ab + r(a ⊕ b)
- s = r ⊕ (a ⊕ b)
- Additionneur complet :



Soustracteur | **Rappel sur le complément à 2**

- a - b = a + (-b).
- Profite des avantages du complément à 2.
- Le codage de (-a) est obtenu comme flip(a) + 1.

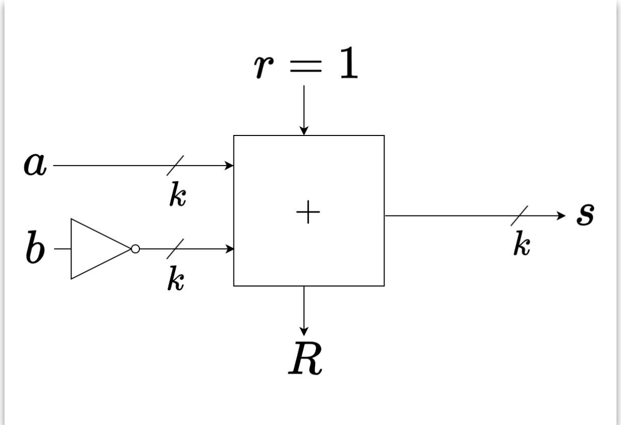
Soustracteur

- a - b = a + (-b) = a + flip(b) + 1

Fourni au circuit comme un "reste initial".

flip(b) = NOT(b).

- Soustracteur :



Note sur les optimisations

- Dans notre additionneur complet, chaque additionneur 1-bit doit attendre le résultat de son voisin avant d’effectuer son calcul, car il a besoin de la retenue.

- Dans les ordinateurs modernes, les additionneurs utilisent des techniques avancées pour anticiper les retenues et gagner du temps.

Logique séquentielle et éléments de mémoire

- Nous reviendrons en fin de semestre sur la façon de réaliser des circuits capables de **mémoriser des valeurs**.

- Cela se fera grâce à des circuits dits "**séquentiels**", où une notion de temps intervient.