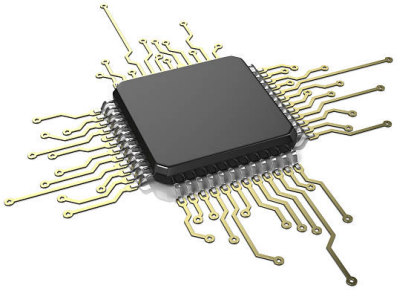
Chapitre 2

Circuits logiques

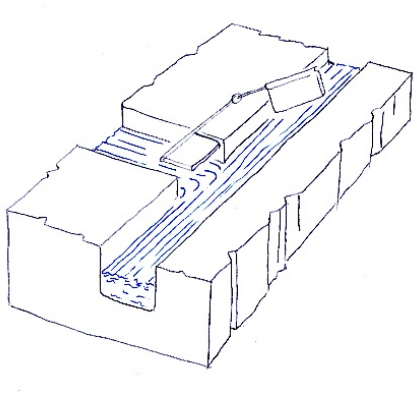


2.1 Transistors et Portes Logiques

2.1.1 Quelques mots sur les transistors

Le **transistor** est la brique de base avec laquelle sont construits les circuits électroniques tels que les micro-processeurs.
C'est un **semi-conducteur**, en d'autres termes un isolant qui peut devenir - sous certaines conditions - conducteur de courant électrique. Dans nos ordinateurs, les transistors utilisés sont de type MOS (Metal-Oxide-Semiconductor).

Un transistor possède trois broches appelées drain, grille et source. Il se comporte comme un interrupteur électrique entre la source et le drain qui serait commandé par la grille. Le dessin ci-dessous illustre de manière imagée la façon dont fonctionne un **transistor de type N** :



Lorsqu'une tension électrique (typiquement 2,9V) est appliquée sur la grille, la source et le drain sont connectés.
Si au contraire, la grille est mise à une tension de 0V, le circuit entre la source et le drain est ouvert (le courant ne circule plus).

Il existe également des **transistors de type P**, pour lesquels le fonctionnement est inversé : le drain et la source sont connectés lorsque la tension appliquée à la grille est 0V, et non connectés sinon.

On représente ces deux types de transistors de la manière suivante :

Type *N* Type *P*

2.1.2 Portes logiques

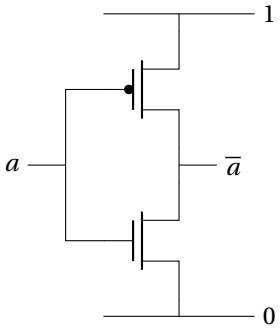
Des transistors bien agencés entre eux permettent de réaliser des **portes logiques** représentant les **opérateurs booléens** de base comme le *and*, le *or*, le *not*, le *nand*...

Porte *not*

La porte la plus simple est la porte not de la négation, dont on rappelle la table de vérité ci-contre.

Entrée	**Sortie**
a	$\overline{a}$
0	1
1	0

Une telle porte possède 1 entrée et 1 sortie. Elle est réalisée à l'aide de deux transistors : un transistor de type N et un transistor de type P.



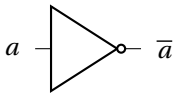
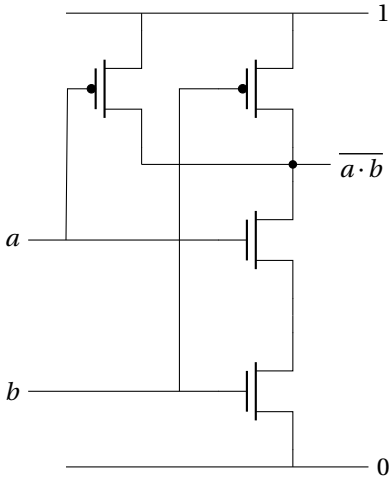


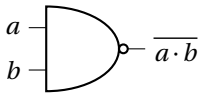
Schéma et symbole de la porte *not* (inverseur)

Porte *nand*

L'opérateur *nand* (non-et) est défini par $a \text{ nand } b = \overline{a \cdot b} = \overline{a} + \overline{b}$.
Avec 4 transistors (2 types N et 2 types P), on peut fabriquer la porte logique correspondante, comportant 2 entrées et 1 sortie.

a	b	$\overline{a \cdot b}$
0	0	1
0	1	1
1	0	1
1	1	0

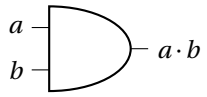




Porte *and*

En combinant ces deux portes, on obtient logiquement la porte *and* associée à l'opérateur élémentaire *and* (*et*) :

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1



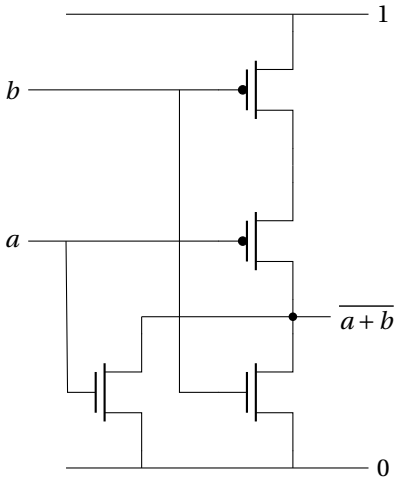
Il faut donc 6 transistors (3 types N et 3 types P) pour fabriquer une porte *and*.

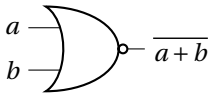
Remarque. Il est théoriquement possible de réaliser une porte *and* à l'aide de « seulement » 4 transistors, mais pour des raisons techniques - dans le but de minimiser le courant consommé - c'est l'assemblage d'une porte *nand* et d'une porte *not* qui est couramment utilisé.

Portes *nor* et *or*

Avec 4 transistors, on peut également obtenir la porte *nor* (*non-ou*), associée à l'opérateur *nor*, défini par $a \text{ nor } b = \overline{a+b} = \overline{a} \cdot \overline{b}$.

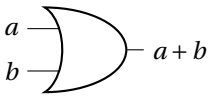
a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0





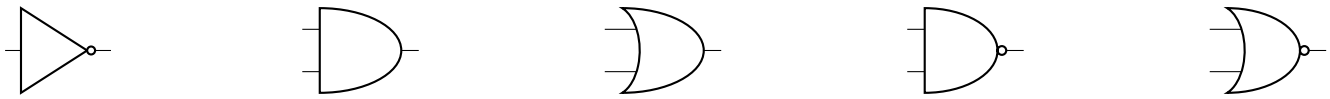
En combinant une porte *nor* et une porte *not*, on obtient la porte *or* :

a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	1



Récapitulatif

Nous venons de découvrir les 5 portes logiques de bases que sont les portes *not*, *or*, *and*, *nand* et *nor*.



NOT *AND* *OR* *NAND* *NOR*

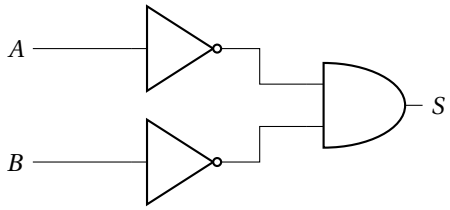
Il est temps de brancher ces portes logiques entre elles pour obtenir des **circuits logiques** (ou **circuits combinatoires**).

2.1.3 Circuits combinatoires

La théorie, c'est bien, mais rien ne vaut la pratique pour appréhender ces nouvelles notions relativement obscures.
Pour « jouer » avec les portes logiques, on dispose du logiciel *Logisim* qui permet d'assembler des portes logiques afin de créer des circuits combinatoires.

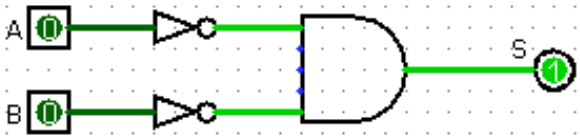
Premier exemple

Question 01 Réaliser le circuit suivant avec *Logisim* :



- a et b sont les entrées : on utilisera l'objet 
- S est la sortie : on utilisera également  mais réglé en « Output »
- Désactiver le mode *Three-state* des entrées et sortie

Le circuit devrait ressembler à ceci une fois terminé :



Pour modifier les valeurs d'entrée, il suffit de cliquer dessus avec la petite main 

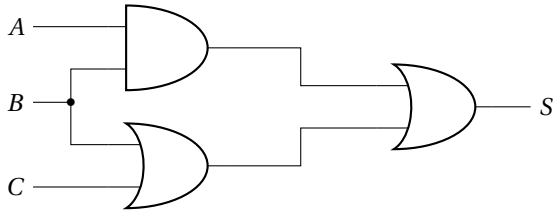
Question 02 Déterminer l'expression booléenne de S en fonction de A et B.

Question 03 Donner la table de vérité de S.

Question 04 Par quel circuit, comprenant uniquement 1 porte *NOT* et 1 porte *OR*, peut-on remplacer le circuit précédent ? À quelle porte logique est-il équivalent ?

Deuxième exemple

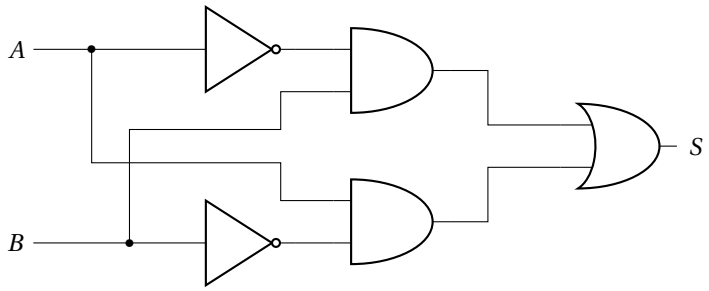
Question 05 On considère le circuit suivant :



1. Donner l'expression booléenne de S en fonction de A, B et C.
2. Donner la table de vérité de S.
3. En déduire une expression de S simplifiée.

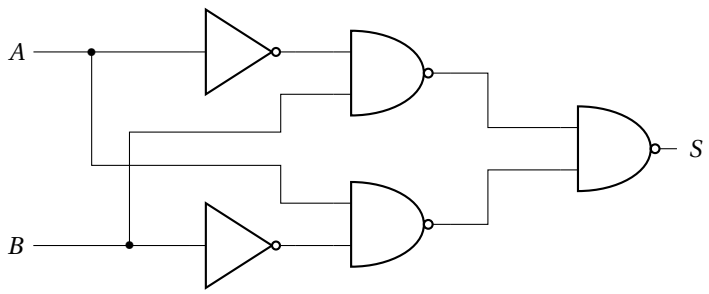
Porte *xor*

Question 06 On considère le circuit suivant :



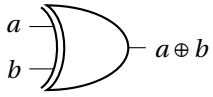
1. Donner l'expression booléenne de S en fonction de A et B.
2. Donner la table de vérité de S.

Question 07 Même exercice, avec des portes *NAND* :

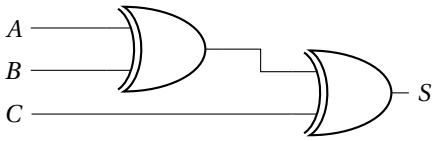


Les deux circuits équivalents vus précédemment constituent l'opérateur *XOR* (ou exclusif), noté symboliquement $\oplus$.
La porte logique correspondante ainsi que la table de vérité du *XOR* sont données ci-dessous.

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



Question 08 Compléter la table de vérité de S dans le circuit logique suivant :



A	B	C	S
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

L'opérateur XOR a de multiples applications :

- en **cryptographie** : on peut coder un message binaire M à l'aide d'une clé binaire K de même longueur, en effectuant un *XOR* « bits à bits ». Pour décoder le message, il suffit d'utiliser la même clé K et le même procédé.
- en **électricité domestique** : pour contrôler l'allumage d'une ampoule via 2 interrupteurs, on peut utiliser un montage dit « va-et-vient » qui n'est rien d'autre qu'une implémentation mécanique de l'opérateur *XOR*. Chacun des deux interrupteurs peut alors allumer ou éteindre l'ampoule quelle que soit la position de l'autre interrupteur.
- en **informatique** : parmi les nombreuses applications en informatique, il y en a une qui va nous intéresser tout particulièrement. C'est l'**addition** de nombres binaires.

2.2 Un exemple de circuit logique : l'additionneur

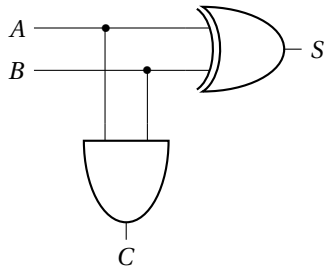
Comment expliquer à notre ordinateur que 10 + 10 = 100 ? (en binaire, rassurez-vous !)
Tout repose sur deux opérateurs logiques : le *ET* et le *XOR*.

2.2.1 Semi-additionneur

L'addition binaire répond aux règles suivantes :

Bit 1	Bit 2	Somme	Retenue
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On remarque sans peine que la somme correspondant au *XOR* et la retenue au *ET*.
Ainsi, avec deux portes logiques *XOR* et *ET*, on peut réaliser l'addition de deux bits :



☞ *S représente la somme des bits A et B, et C la retenue (**Carry** en anglais)*

Question 09 Réaliser ce circuit dans *Logisim*. Vérifier le fonctionnement de l'addition !

Question 10 Donner les tables de vérité de *S* et *C* via le logiciel : *Project => Analyse Circuit => Table*

Le circuit que l'on vient de réaliser s'appelle un **semi-additionneur**.
Semi seulement, car il ne prend pas en compte une éventuelle **retenue en entrée**. Par exemple :

		1	*1*	
	1	0	0	1
+	0	0	1	1
	1	1	0	0

Les additions sur les 2ème et 3ème colonnes nécessitent la connaissance des retenues des opérations précédentes.
Il est donc nécessaire d'ajouter une entrée à notre additionneur.

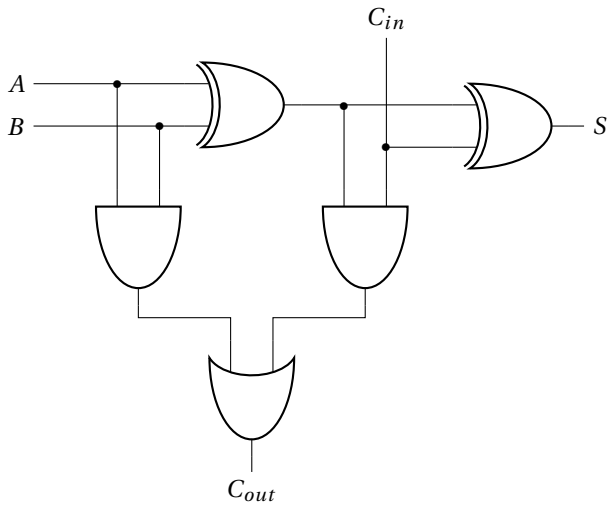
2.2.2 Additionneur complet

L'addition binaire avec retenue en entrée répond aux règles suivantes, où A et B sont les bits d'entrée, C_{in} et C_{out} sont les retenues d'entrée et de sortie, et S est la somme de A et B.

Entrées			**Sorties**	
A	B	C_{in}	S	C_{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

Le circuit correspondant peut être construit en assemblant deux semi-additionneurs en cascade.

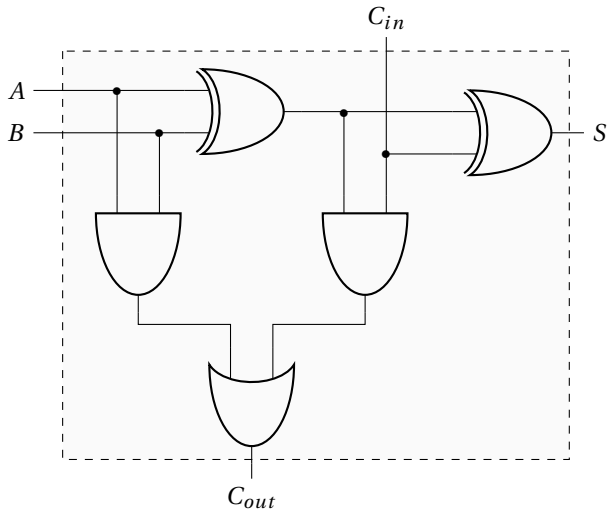
- Le premier semi-additionneur calcule d'abord la somme de A et B : $A \oplus B$.
- Le second calcule la somme du premier résultat et de C_{in} : $(A \oplus B) \oplus C_{in}$.
- La retenue C_{out} vaut 1 s'il y a au moins une retenue à une des deux sommes effectuées : $(A \cdot B) + C_{in}$.

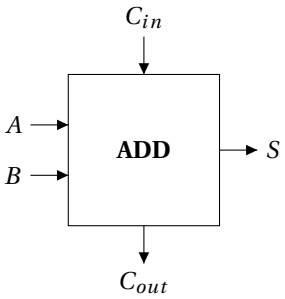


Question 11 Réaliser ce circuit combinatoire avec Logisim et afficher la table de vérité de C_{out} et S.

2.2.3 Additionneur 4 bits

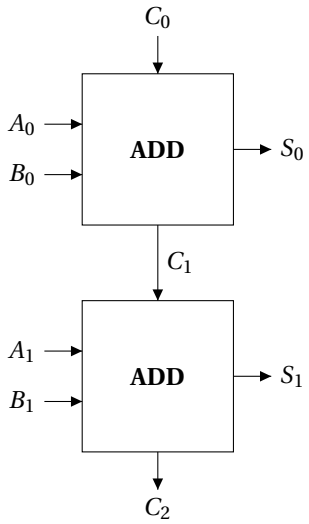
Le circuit précédent permet de réaliser l'addition de deux nombres binaires de 1 bit, avec une retenue en entrée.
On peut « enfermer » ce circuit dans une « boîte » correspondante à un additionneur complet à 1 bit avec retenue en entrée.





En chaînant deux additionneurs à 1 bit, on obtient un additionneur à 2 bits :

c_2	c_1	c_0
	A_1	A_0
+	B_1	B_0
	S_1	S_0



Question 12 Réaliser l'additionneur à 2 bits sur Logisim.
On utilisera le circuit *Adder* disponible dans le logiciel.

☞ ***Régler le paramètres « Data bits » du circuit Adder à 1***

Question 13 Réaliser un additionneur à 4 bits sur Logisim.

Question 14 Avec votre additionneur, vérifier que le calcul suivant est bien effectué :

		1	*1*	
	1	0	0	1
+	0	0	1	1
	1	1	0	0

2.2.4 Indicateurs

À l'issue d'une opération arithmétique comme l'addition, il est souvent utile - pour des raisons que nous découvrirons plus tard - de connaître certaines caractéristiques du résultat obtenu.
On appelle ces caractéristiques des **indicateurs** ou **flags** (drapeaux) en anglais.

Les principaux indicateurs utilisés sont les indicateurs N, Z, C et O décrits ci-dessous :

- Indicateur **N** (Negative) : il indique si le résultat est négatif.

  ↪ *Les entiers étant représentés en compléments à 2, il est égal au bit de poids fort du résultat*

- Indicateur **Z** (Zéro) : il indique si le résultat est égal 0.

  ↪ *Il est égal au complémentaire du* OR *(c'est-à-dire un* NOR*) de tous les bits du résultat*

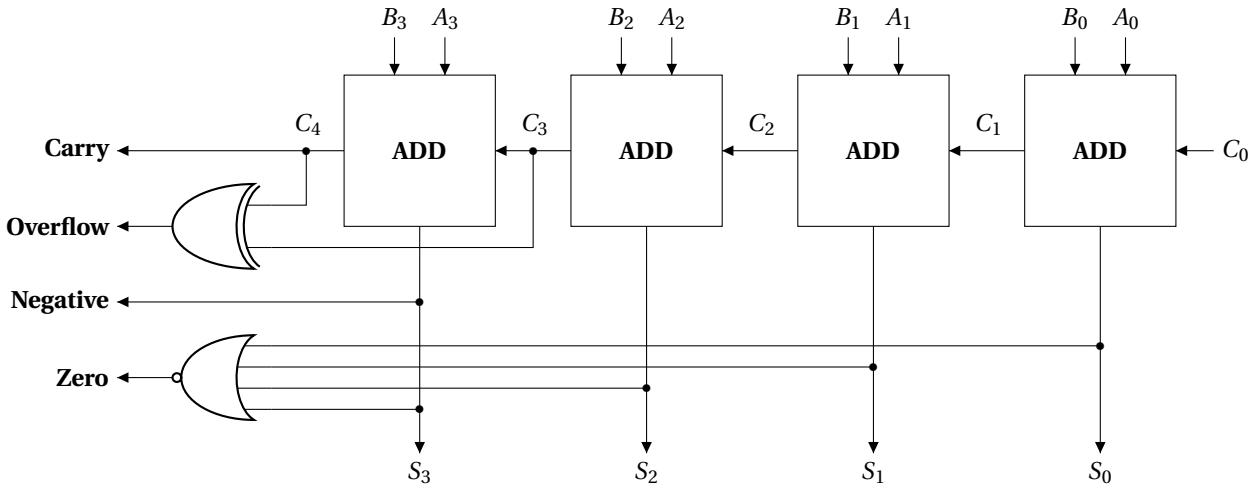
- Indicateur **C** (Carry) : il indique si l'opération a provoqué une retenue.

  ↪ *Il est égal à la retenue du dernier additionneur*

- Indicateur **O** (Overflow) : il indique un débordement lors d'une addition de nombres signés.

  ↪ *Il y a débordement lorsque la somme de deux nombres positifs est supérieure à* 2^{k-1} *(pour un additionneur k bits) ou lorsque la somme de deux nombres négatifs est inférieure à* $-2^{k-1}-1$. *Il est égal au* XOR *des deux dernières retenues*

Pour un additionneur 4 bits, le schéma (tourné de 90°) est le suivant :



Question 15 Compléter l'additionneur créé à la question précédente.

Question 16 Effectuer les opérations suivantes à l'aide de l'additionneur 4 bits, et noter les résultats et indicateurs obtenus :

- 3 + 2
- 4 + 3
- 1 + (-2)
- 5 + 4

Question 17 Créer un additionneur 8 bits avec indicateurs à l'aide de Logisim. Ne pas se plaindre : les ordinateurs actuels sont dotés d'additionneurs 64 bits!!

2.2.5 Et la soustraction ?

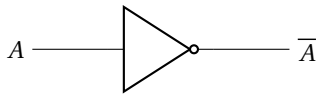
Une fois l'additionneur construit, il n'est pas difficile d'effectuer la soustraction de deux nombres, sachant que :

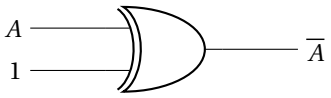
« Soustraire, c'est ajouter l'opposé »

L'opposé d'un nombre négatif est, en binaire, son complément à 2. Pour trouver le complément à 2, il suffit de (revoir le cours sur la représentation des nombres négatifs au besoin) :

- Inverser tous les bits
- Ajouter 1

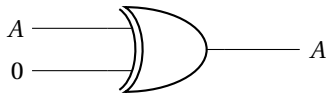
Pour inverser tous les bits, on peut appliquer un *NOT* à chaque bit ou calculer le *XOR* de chaque bit avec 1 :

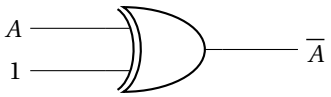




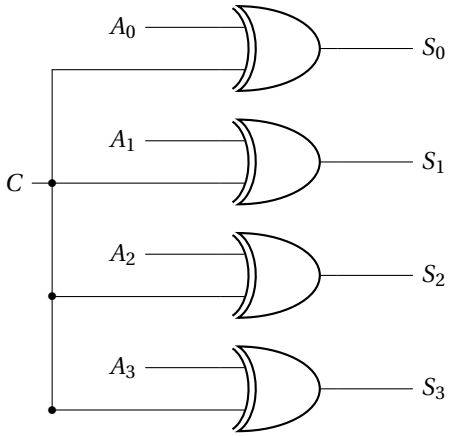
▷ ***Quel est l'intérêt d'utiliser une porte* XOR *?***

L'intérêt est avant tout pratique. En modifiant la valeur de la seconde entrée, on peut choisir d'inverser ou non le bit :





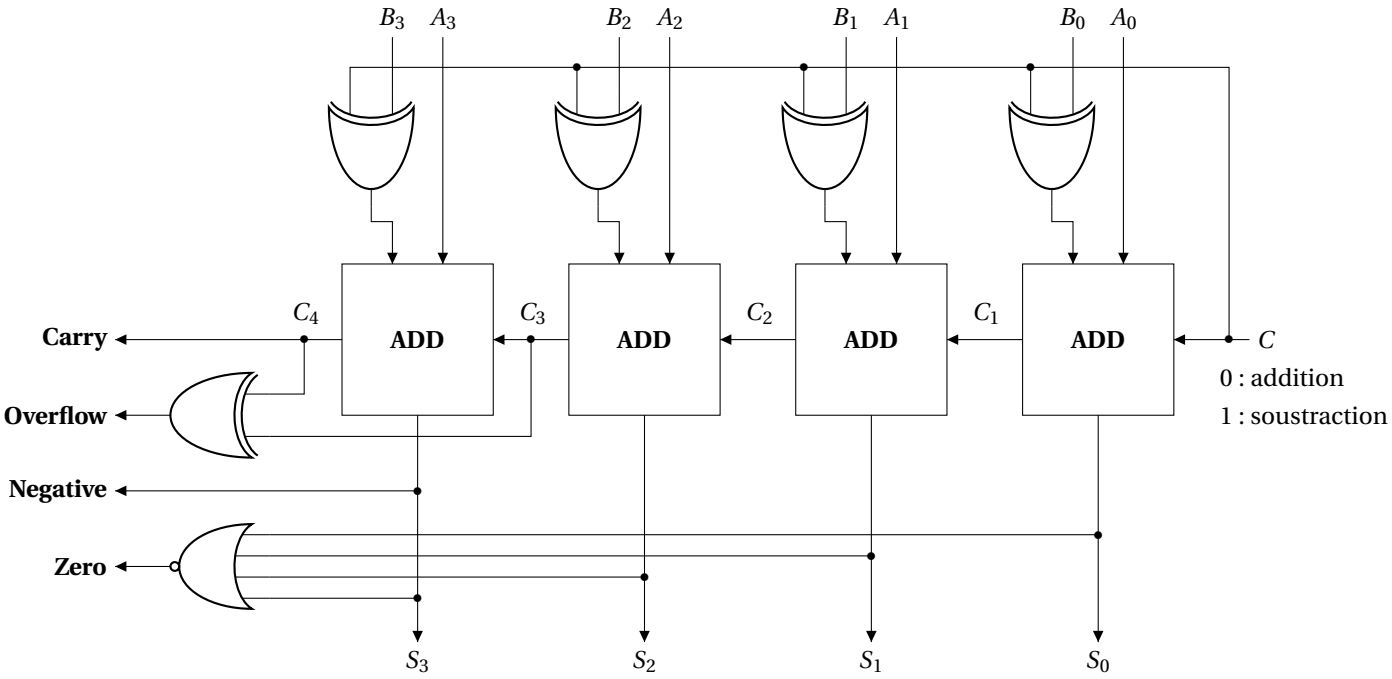
Voici par exemple un inverseur à 4 bits, muni d'une commande C.
Si $C = 1$, on inverse les bits. Si $C = 0$, les bits ne sont pas inversés.



Mais l'intérêt ne s'arrête pas là. Il reste à ajouter 1 pour obtenir l'opposé de notre nombre.

▷ ***J'ai compris ! Si C = 1, on ajoute 1. Si C = 0, on n'inverse rien, donc on ajoute 0. Au final, on ajoute simplement C.***

C'est exact ! Voici donc l'astucieux additionneur / soustracteur à 4 bits, commandé via la commande C.



Voyez-vous en quoi le circuit est astucieux ?

Question 18 Créer ce circuit sur Logisim et tester les opérations suivantes :

- 1 + 5
- 4 - 3
- 2 - 5
- 4 - (-5)
- -6 - 6

2.3 Décodeur et Multiplexeur

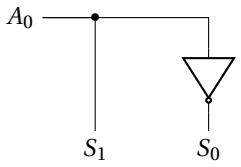
Les **décodeurs** et **multiplexeurs** sont des circuits relativement élémentaires mais très souvent utilisés. Il s'agit de deux briques de base pour la construction de circuits plus élaborés. Ils sont en particulier présents dans chaque **circuit mémoire**.

2.3.1 Décodeur

Un décodeur k bits possède k entrées et 2^k sorties. La sortie dont le numéro est donné par les entrées est active (valeur 1) alors que toutes les autres sorties sont inactives (valeur 0).

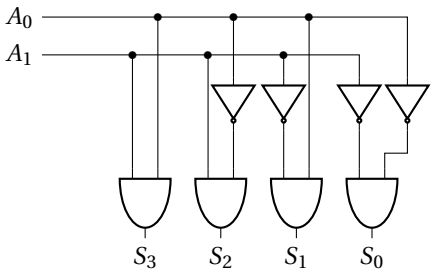
Décodeur 1 bit

Entrées	**Sorties**	
A_0	S_0	S_1
0	1	0
1	0	1



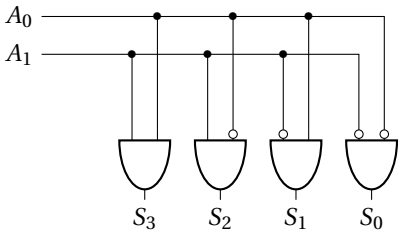
Décodeur 2 bits

Entrées		**Sorties**			
A_1	A_0	S_0	S_1	S_2	S_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



On parle de décodeur « *1 parmi n* » car on active une seule entrée parmi celles disponibles.
La sortie activée correspond au nombre binaire $A_k A_{k-1} \dots A_1 A_0$ donné en entrée.

Remarque. Pour inverser l'entrée d'une porte logique, on peut utiliser le symbole ∘. Le circuit précédent est simplifié (seulement schématiquement, car le circuit reste en tout point identique).



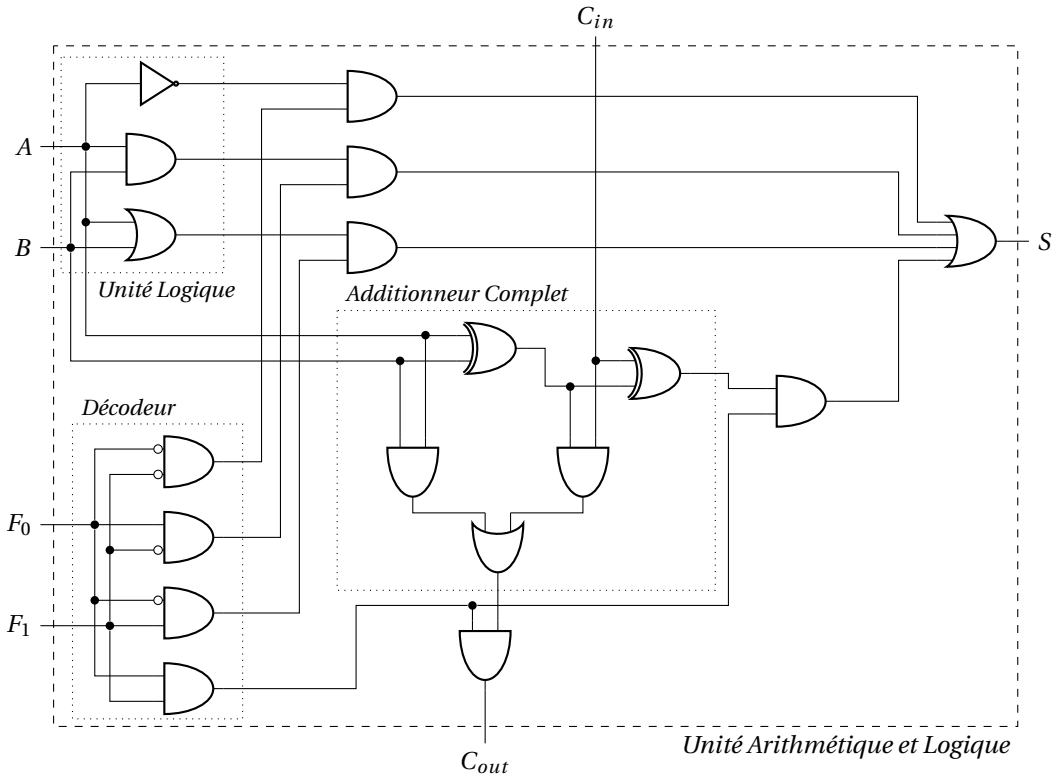
Question 19 Réaliser ce circuit sur Logisim et vérifier son fonctionnement.

☞ *Pour inverser une entrée, choisir l'option* Negate *pour l'entrée correspondante*

Exemple d'application

On souhaite créer un circuit combinatoire permettant de réaliser une opération sur deux bits A et B. Cette opération peut être un *NON*, un *ET*, un *OU* ou une *SOMME*.
Le choix de l'opération se fait à l'aide d'un **décodeur**.
On attribue alors un code à chaque opération (on parle d'**opcode**).

Opcode	Sortie
00	*NON A*
01	*A ET B*
10	*A OU B*
11	*SOMME A + B*



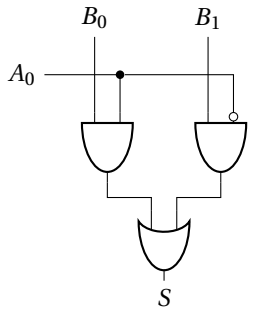
Le circuit obtenu s'appelle une **Unité Arithmétique et Logique**, abregé **UAL**.
Bien entendu, notre circuit reste extrêmement simple comparé aux UAL présentes dans les processeurs actuels et ne doit être considéré qu'à titre d'exemple.

2.3.2 Multiplexeur

Un multiplexeur k bits permet de sélectionner une entrée parmi 2^k disponibles. Il dispose de $k+2^k$ entrées et une seule sortie. Les k premières entrées A_0, A_1, ..., A_{k-1} sont appelées **bits d'adresses** car elles donnent le numéro de l'entrée à sélectionner parmi les entrées B_0, B_1, ..., B_{2^k-1}.
La sortie S est alors égale à cette entrée sélectionnée.

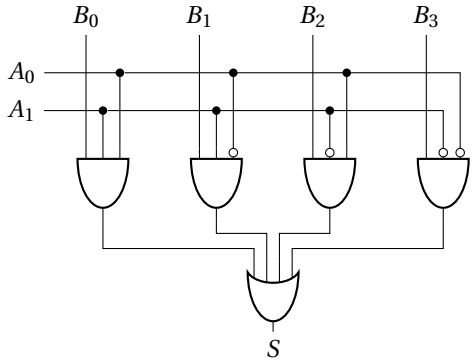
Multiplexeur 1 bit

Entrées	**Sortie**
A_0	S
0	B_0
1	B_1



Multiplexeur 2 bits

Entrées		**Sortie**
A_1	A_0	S
0	0	B_0
0	1	B_1
1	0	B_2
1	1	B_3

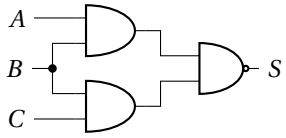


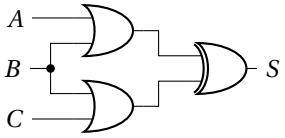
Le multiplexage est utilisé pour transformer des données **parallèles** en données **séries**.
On peut illustrer le fonctionnement avec une box Internet. Elle reçoit des données de plusieurs appareils (ordinateur, tablette, téléphone portable...) qu'elle transmet via une seule sortie (câble téléphone ou fibre).
Les données qu'elle reçoit à destination des différents appareils sont quant à elles **démultiplexées** (via un *démultiplexeur*), c'est à dire reçue sur une entrée et réparties sur plusieurs sorties.

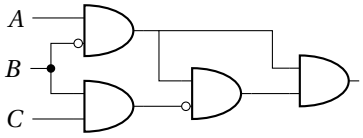
Question 20 Réaliser un multiplexeur 2 bits sur Logisim et vérifier son fonctionnement.

2.4 Exercices

Exercice 01 Donner pour chacun des circuits logiques suivants l'expression logique des sorties en fonction des entrées ainsi que les tables de vérité associées :







Exercice 02 Proposer un circuit logique correspondant aux expressions logiques suivantes :

1. $S = a \cdot b + c$
2. $S = \overline{a} \cdot (b + c)$
3. $S = (a + b) \cdot (a + c)$
4. $S_1 = a + \overline{b}$ et $S_2 = a \oplus b$

Exercice 03 Pour chaque table de vérité suivante, donner l'expression de la sortie S en fonction des entrées et proposer un circuit logique correspondant à cette table de vérité.

Entrées		**Sorties**
A	*B*	*S*
0	0	1
0	1	0
1	0	0
1	1	0

Entrées		**Sorties**
A	*B*	*S*
0	0	1
0	1	0
1	0	0
1	1	1

Entrées			**Sorties**
A	*B*	*C*	*S*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1