Le bit est l'unité la plus simple dans un système de numération, ne pouvant prendre que deux valeurs, désignées le plus souvent par les chiffres 0 et 1. Le mot *bit* est la contraction des mots anglais *binary digit*, qui signifient *chiffre binaire*, avec un jeu de mots sur *bit*, petit morceau. (Wikipedia)

0.1 Notation positionnelle: représentation des entiers positifs

Le système binaire (du latin *binārĭus* : double) est le système de numération utilisant la base 2. Il permet de représenter des nombres à l'aide de la numération de position avec seulement deux chiffres : le 0 et le 1.

0.1.1 Base 10

Nous représentons quotidiennement les nombres avec la base 10: les 9 chiffres permettent par leur position dans un nombre de représenter tous les nombres entiers.

Exemple: En base 10 ou décimale, l'entier naturel 123 se décompose de la façon suivante:

$$\mathbf{1} \times 10^2 + \mathbf{2} \times 10^1 + \mathbf{3} \times 10^0$$

La position des chiffres indique si on doit les considérer comme des unités, des dizaines, des centaines. On dispose des dix chiffres de 0 à 9 pour multiplier chacune des puissances de 10.

Utilisons une représentation dans un tableau:

Puissance de 10 ou **Poids**	10^3	10^2	10^1	10^0
Coefficient	0	1	2	3

1. Donner de la même façon la décomposition du nombre 1912.

0.1.2 Base 2

Les ordinateurs calculent en base 2 ou binaire. Cela vient de leur fonctionnement: les constituants de base que sont leurs transistors savent ou non laisser passer le courant. Il n' y a donc que deux états. Le mode de calcul est le même mais la base est 2, on ne dispose que de deux chiffres: 0 et 1.

Exemple: Le nombre 1101_2 et égal à $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1 = 13$.

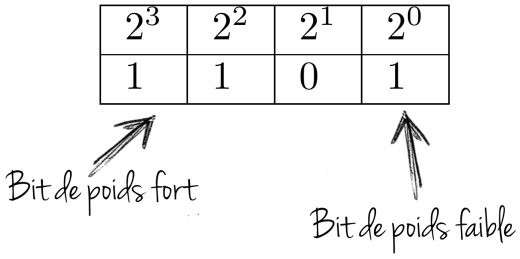


Figure 1: On a représenté ce nombre avec 4 bits.

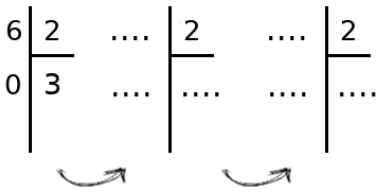
C'est la représentation utilisée pour les entiers machines des processeurs modernes. C'est la représentation utilisée pour les entiers machines des processeurs modernes.

1. `Conversion binaire vers décimal`. Donner la décomposition avec les puissances de 2 des nombres binaires 1101_2, 1111_2. En déduire leur valeur décimale.

2. `Conversion décimal vers binaire`.Voici un algorithme permettant de convertir un nombre entier n écrit en base 10 vers la base 2, il repose sur des divisions euclidiennes successives:

Algorithme 1 : Conversion

```
tant que n ≠ 0 faire
    Diviser n par 2
    Noter le reste r_i
    Poser n égal au quotient
fin
Renvoyer r_p....r_2 r_1 r_0
```



Exemple pour $n = 6$

$0 \leqslant i \leqslant p$, p nombre de chiffres nécessaires à l'écriture de n en base 2.

Autrement dit, il suffit alors d'écrire les restes obtenus **de droite à gauche** dans l'ordre de leur apparition pour obtenir la conversion.

(a) Compléter l'exemple pour convertir 6 en base 2 et sur 4 bits (`binary digit`).

(b) Quel sont les restes possibles dans une division euclidienne lorqu'on divise par 2 ?

(c) Convertir les nombres 15 puis 192 en base 2 sur 4 bits.

0.2 Domaine couvert

0.2.1 Codage sur n bits

Les machines ont des processeurs travaillant sur un nombre de bits fixe: un ordinateur avec un processeur 32 bits est optimisé pour travailler sur 32 bits. Cela induit une limitation dans les calculs: au maximum, on ne pourra représenter que $2^{32} = 4294967296$ entiers. Dans le cas de notre processeur 32 bits, le domaine couvert s'étend de: 00000....00000 à 1111....11111 sur 32 bits.

Avec ce type de processeur, on peut coder sur 2^{32} bits. Si on ne code que des entiers x positifs, on aura $0 \leqslant x \leqslant 2^{32} - 1$.

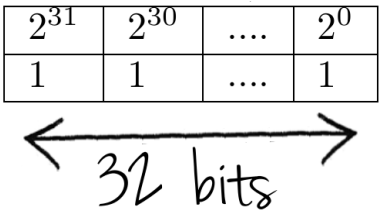


Figure 2: Entier maximum sur 32 bits

0.3 Addition binaire

Unité Arithmétique et Logique C'est l'unité arithmétique et logique (UAL) qui est chargée d'effectuer les calculs dans le processeur et notamment l'addition. En binaire :

- 0+0= 0
- 0+1= 1
- 1+0=0
- 1+1=10 (une retenue est créée)

Poser une addition Poser les additions :

- 357+862
- $101_2 + 011_2$ en base 2 sur 4 bits.

On réalise un additionneur à l'aide de porte logique.

0.4 Circuits et logique booléenne

Les portes logiques sont les opérations les plus basiques que l'on peut réaliser sur un bit. Une porte logique prend en entrée un ou plusieurs bits et génère en sortie un bit de résultat.

Elles sont fabriquées avec des composants électroniques que l'on appelle des transistors. Voyons quelles sont les portes logiques et comment elles sont combinées pour effectuer des calculs. Nous utiliserons des tables vérité à deux entrées pour donner les résultats des principaux opérateurs de logique binaire ou booléenne.

0.4.1 Fonctions logiques

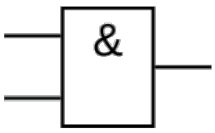
Nous établirons pour chaque table de vérité son expression logique associée. Une fonction logique est souvent exprimée sous forme de somme de produits. On se concentre sur les lignes où la sortie est 1 car ce sont elles qui contribuent à la fonction.

Méthode

- Écrire un produit logique pour chaque ligne où la sortie est 1, ce produit est appelé *minterm*.
- Si une variable vaut 1, elle est utilisée sous sa forme normale A.
- Si une variable vaut 0, elle est utilisée sous sa forme complémentée $\overline{A}$.
- Effectuer l'addition logique de chacun de ces *minterms*.

0.4.2 Porte ET/AND

Soient A et B deux valeurs logiques. A ET B est VRAI si et seulement si les deux opérandes ont la valeur VRAI.

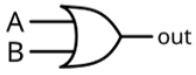
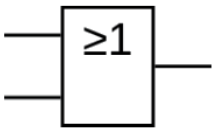
<table>
<tr><th>Type</th><th>Symbole américain</th><th>Symbole européen</th><th>Notation</th><th>Opération booléenne</th><th>Table de vérité</th></tr>
<tr><td>ET</td><td> 
</td><td> 
</td><td>$A \wedge B$</td><td>$A \cdot B$</td><td>
<table>
<tr><th colspan="2">Entrée</th><th>Sortie</th></tr>
<tr><td>A</td><td>B</td><td>A ET B</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
</table>

Expression logique : A.B

0.4.3 Porte OU/OR

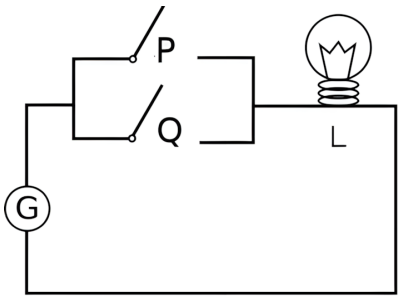
Soient A et B deux valeurs logiques. A OU B est VRAI si et seulement si A est VRAI OU B est VRAI (y compris si les deux sont VRAIS).

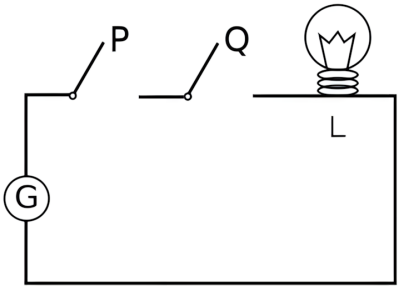
Expression logique : A+B

OU			$A \vee B$	$A + B$	

Entrée		Sortie
A	B	A OU B
0	0	0
0	1	1
1	0	1
1	1	1

Analogie avec les circuits électriques Voici une représentation sous forme de schéma électrique des fonctions logiques **OU** (à gauche) et **ET**:

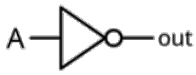
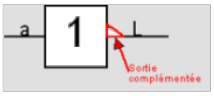




Dans quels cas la lampe s'allume-t-elle ?

0.4.4 Porte NOT/NON

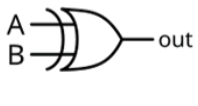
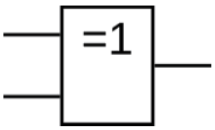
Soit A une valeur logique. NON A est VRAI si A est FAUX, et réciproquement.

NON			$\neg A$	$\overline{A}$	

Entrée	Sortie
A	NON A
0	1
1	0

0.4.5 Porte OU EXCLUSIF/XOR

Soient A et B deux valeurs logiques. A XOR B est VRAI si et seulement si A est VRAI OU B est VRAI mais pas les deux.

OU exclusif (XOR)			$A \oplus B$	$A \oplus B$	

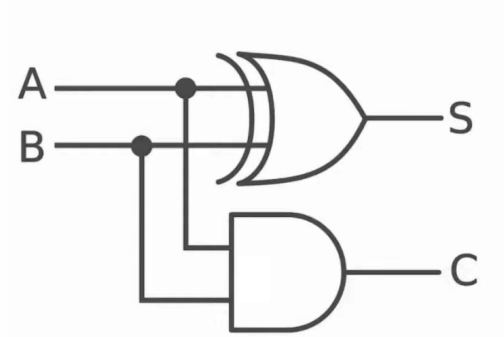
Entrée		Sortie
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Expression logique : $\overline{A}.B + \overline{B}.A = A \oplus B$

0.4.6 Additionneur

Un additionneur est un circuit logique permettant de réaliser une addition.

Demi-additionneur Le fonctionnement d'un demi-additionneur se définit à l'aide de deux expressions booléennes `XOR` et `ET`. Le schéma du demi-additionneur ci-dessous dispose de deux entrées logiques A et B. Les deux sorties correspondent respectivement à la somme de A et B pour S, et C pour la retenue.



A	B	S A + B	C Retenue
0	0		
0	1		
1	0		
1	1		

0.5 Entiers négatifs

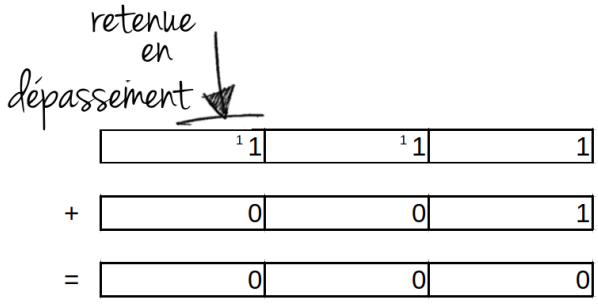
Une idée assez naturelle est de fonctionner comme en décimal : un signe et une valeur absolue. Utiliser le bit de poids fort pour coder le signe, 1 pour négatif et zéro pour positif puis les autres bits pour la valeur absolue. Ce système intéressant par sa simplicité a pour inconvénient de présenter deux zéros 0000_2 et 1000_2.

Calculer 3+(-2) en base 2 avec ce modèle sur 4 bits.

0.5.1 Phénomène de dépassement

Sur la *figure 1.3*, on additionne à capacité fixe sur 3 bits les deux nombres binaires 111_2 et 001_2:

Sur 3 bits, il n' y a rien après le 3^{eme} bit et la dernière retenue disparait simplement : elle est en *dépassement*. Ce qui peut apparaitre comme un problème peut être utilisé avec avantage pour représenter les entiers relatifs. En effet, à capacité fixe sur n bits, l'entier le plus grand pouvant être représenté est $2^n - 1$. Or nous venons de voir sur 3 bits que $2^3 - 1 + 1 = 0$. Ceci est généralisable à n bits et on a:



$$(2^n - 1) + 1 = 0$$

Ce qui prouve que $2^n - 1$ est une représentation de -1 sur n bits. Généralisons cette propriété à tout entier x.

0.5.2 Complément à deux

Définition Sur n bits, le complément à deux d'un entier x est $2^n - x$.
C'est une représentation de $-x$. En effet:

1. $x + (-x) = x + 2^n - x = 2^n$, c'est à dire 0 sur n bits.
2. De plus $-(-x) = -(2^n - x) = 2^n - (2^n - x) = x$

Comment utiliser cette représentation alors que 2^n n'est pas représentable sur n bits ? Il suffit d'utiliser l'astuce d'écriture expliquée sur 3 bits en 1.2.2:

$$(2^n - x) = (2^n - x - 1 + 1) = (2^n - 1 - x + 1).$$

Par conséquent, pour obtenir le complément à deux de x, on applique l'algorithme suivant:

1. prendre le complementaire à 1.En effet,sur n bits:

   $$2^n - 1 = 11111....111$$

   soustraire x à 111111....111 revient à changer 0 en 1 et 1 en 0 (on prend le *complément à 1*).
2. ajouter 1.

En bref : complément à deux et représentation machine des nombres relatifs

Pour représenter les relatifs, on utilise les binaires signés où le bit de poids fort est utilisé pour représenter le signe:
1 pour un nombre négatif et 0 pour un nombre positif.

Exemple

1. Il est nécessaire de fixer à l'avance le nombre de bits de la représentation.
2. Le bit de poids fort est réservé au signe. Pour représenter un nombre négatif N , on code $|N|$ et on prend son complément à deux.

Exemples:

- Sur 8 bits le nombre $\overline{10001001}^2$ est égal à -9.
- Sur 4 bits: voici la représentation de tous les entiers:

suite de bits	0000	0001	0010	0011	0100	0101	0110	0111
signification sur entiers naturels	0	1	2	3	4	5	6	7
signification sur entiers relatifs	0	1	2	3	4	5	6	7
suite de bits	1000	1001	1010	1011	1100	1101	1110	1111
signification sur entiers naturels	8	9	10	11	12	13	14	15
signification sur entiers relatifs	-8	-7	-6	-5	-4	-3	-2	-1

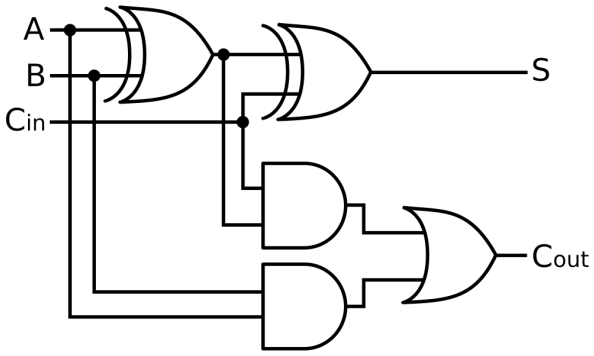
0.6 Python

Réalisés par ces circuits électroniques que l'on appelle des portes logiques ou portes booléennes, les ordinateurs s'appuient sur la théorie de Boole pour effectuer des opérations logiques grâce à des opérateurs booléens. Un opérateur booléen (on dit aussi fonction booléenne) est une fonction mathématique prenant en entrée un ou plusieurs booléens et donnant en sortie un unique booléen.

0.7 Exercices

Exercice 1 On réalise un additionneur 1 *bit* complet en preannt en compte une éventuelle retenue de deux bits précédents. Il y a donc en entrée deux bits A et B et une retenue C. On utilise en pratique deux demi-additionneurs.

Compléter la table de vérité de l'additionneur 1 bit.



A	B	C_{in}	S	C_{out}

Exercice 2 : Lois de De Morgan Complétez la table de vérité suivante. Quelles égalités peut-on établir ?

A	B	$A \cdot B$	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$	$A + B$	$\overline{A + B}$	$\overline{A} \cdot \overline{B}$
0	0						
0	1						
1	0						
1	1						

0.8 Références

Une grande partie des documents est issue de Wikipedia.

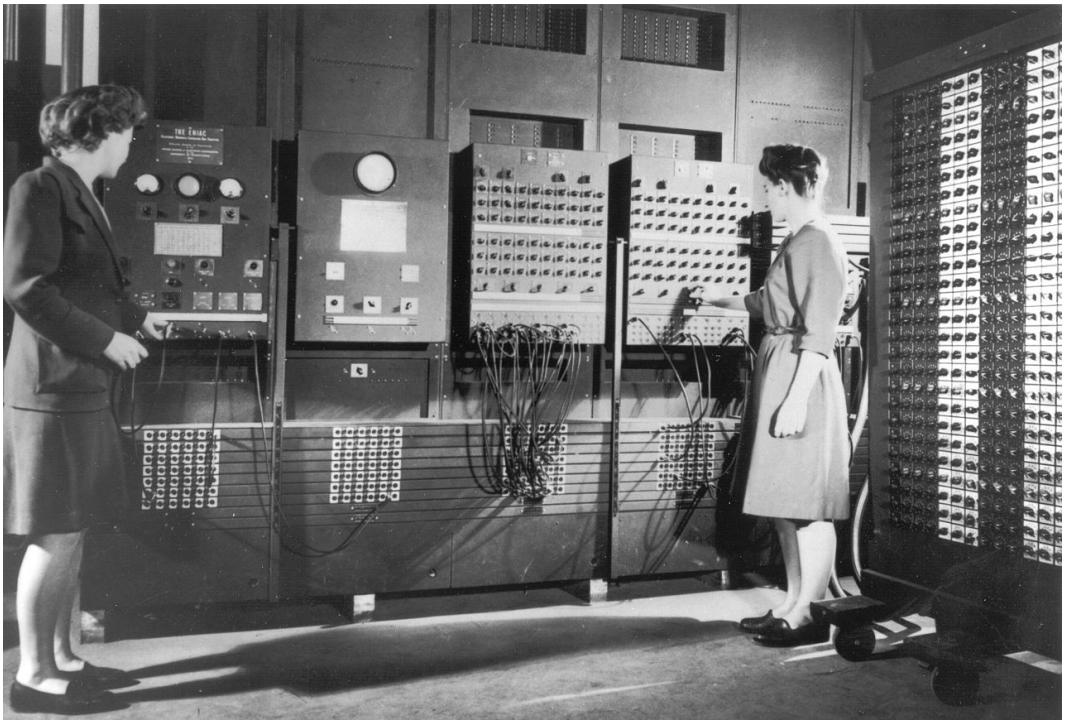


Figure 3: ENIAC