0.1 Architecture de Von Neumann

L'entrée de *John Von Neumann* dans l'histoire de l'informatique se fait pendant la seconde guerre mondiale.

0.1.1 Un premier modèle avec l'ENIAC

Le 10 décembre 1945, *Mauchly* et *Eckert* mettent en opération l'ENIAC et commence à travailler sur des calculs liés à la conception d'armes thermonucléaires. L'ENIAC n'avait cependant pas de mémoire pour stocker les programmes, qui devaient être câblés physiquement via des commutateurs et des câbles, ce qui constituait une faiblesse du projet.

Sous l'influence de Von Neumann, qui rejoint *Mauchly* et *Eckert* alors qu'il travaille sur le Projet Manhattan, l'ENIAC est à partir de 1947 reconverti en ordinateur à programme enregistré.

Le *First Draft of a Report on EDVAC* est un document qui décrit un schéma d'architecture de calculateur, organisé en trois éléments (unité arithmétique, unité de commande et mémoire contenant programme et données).

0.1.2 Architecture de Von Neumann

Dans l'architecture de Von Neumann, un ordinateur possède 4 parties :

1. L'unité arithmétique et logique (UAL ou ALU) ou unité de traitement pour effectuer les opérations de base.
2. L'unité de contrôle chargée de la partie séquentielle des oérations.
3. La mémoire où sont stockées données et programme.
4. Les dispositifs d' entrées-eorties notées `E/S` ou `I/O`.

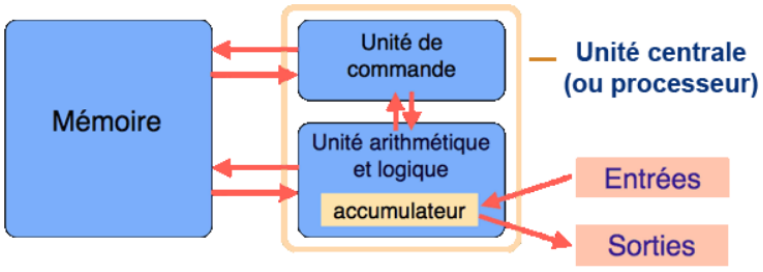


Figure 1: Architecture de Von Neumann

Les bus font le lien entre ces parties : ce sont des ensemble de fils conduisant les impulsions électriques. L'UAL et l'unité de contrôle sont regroupées dans le CPU (Central Processing Unit) ou processeur.

Ce modèle est mis en oeuvre avec l'EDVAC.

L'idée qu'un programme peut être traité comme une donnée par d'autres programmes, présente dans la machine de Turing, trouve ici sa concrétisation.

CC-BY-SA

0.2 Notation positionnelle : représentation des entiers positifs

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)

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.

0.2.1 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.

D'une manière plus générale, la connaissance de l'écriture $x_{p-1}x_{p-2}.....x_1x_0$ dans une base b d'un entier n permet de calculer cet entier par une évaluation de la somme :

$$n = \sum_{k=0}^{p-1} x_k b^k = x_{p-1}x^{p-1} + x_{p-2}x^{p-2} + ... + x_1x^1 + x_0x^0$$

0.2.2 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 ou *bits* pour *binary digit* : 0 et 1.

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

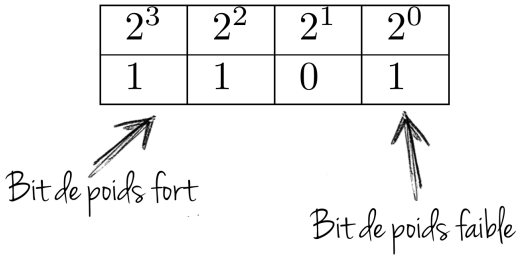


Figure 2: On a représenté ce nombre avec 4 bits.

C'est la représentation utilisée pour les entiers machines des processeurs modernes. D'une manière générale, un nombre binaire sur n bits pourra coder un nombre entier naturel $N = 2^{n-1}$. En effet en comptant le 0, il y a 2^n combinaisons.

CC-BY-SA

Domaine couvert en binaire 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 $0 \leqslant x \leqslant 2^{32} - 1$.

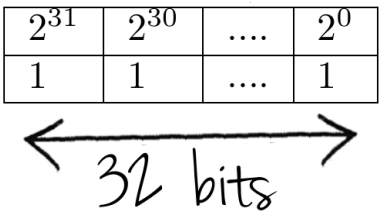


Figure 3: Entier maximum sur 32 bits

0.2.3 Base hexadécimale ou base 16

L'écriture d'un nombre binaire est fastidieuse et prompte à de nombreuses erreurs de retranscription.

On utilise en informatique la base 16 ou hexadécimale pour une écriture plus concise d'un nombre. La base hexadécimale comporte 16 symboles. Les 10 premiers sont les 10 chiffres de la base 10. Pour les six restants, ce sont les 6 premières lettres (majuscules) de l'alphabet.

Exemple 1 : base 16 vers base 10 Pour convertir le nombre $A5E_{16}$ vers la base 10, la méthode est identique au binaire : on utilise le plynôme caractéristique.

$$A5E_{16} = 10 \times 16^2 + 5 \times 16^1 + 14 \times 16^0 = 2654$$

Exemple 2 : base 10 vers base 16 On applique également la méthode des soustractions ou des divisions successives.

Exemple 3 : base 16 et base 2 La conversion entre base binaire et base hexadécimale est immédiate. Ce qui est loin d'être le cas entre les bases binaire et décimale. Un chiffre en base 16 correspond à regroupement de 4 bits ou quartet en binaire :

$$A5E_{16} = 1010\ 0101\ 1110$$

La méthode s'adapte facilement de la base 2 vers la base 16.

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

0.3 Entiers négatifs

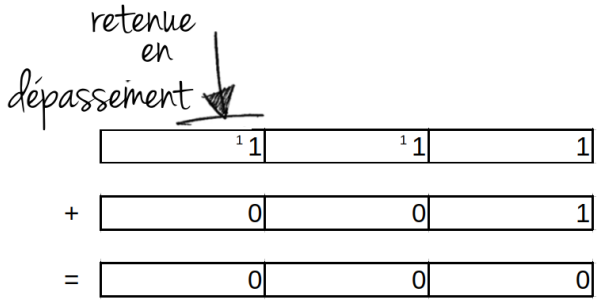
0.3.1 Représentation avec un signe et sa valeur absolue

Pour un entier représenté sur n bits, les $n-1$ bits de poids faible sont réservés à la représentation de la valeur absolue de l'entier tandis que le bit de poids fort permet d'indiquer le signe de cet entier. Le bit de poids fort est à 1 si et seulement si l'entier est négatif. Cette reprsentation pose problème avec notamment la possibilité de deux représentations pour zéro.

0.3.2 Représentation avec complément à deux

Phénomène de dépassement

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



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.

Complément à deux

Avant de représenter un entier relatif, il est nécessaire de définir le nombre de bits qui seront utilisés pour cette représentation (souvent 8, 16 , 32 ou 64 bits)

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. Calculer $2^n - 1 - x$, c'est prendre le complementaire à 1 de x. C'est à dire changer ses 0 en 1 et ses 1 en 0.

   En effet, sur n bits: $2^n - 1 =$ 11111....111 et **soustraire x à 111111....111 revient à changer 0 en 1 et 1 en 0** dans la représentation de x (vérifiez-le sur un exemple).

2. ajouter 1.

CC-BY-SA

En bref Avec n bits, on peut représenter à la fois des nombres positifs et négatifs en utilisant la première moitié des combinaisons pour les positifs et la seconde moitié pour les négatifs :

- Nombres positifs : Ils sont représentés de manière standard en binaire, de 0 à $2^{n-1}-1$.
- Nombres négatifs : Ils sont représentés par les valeurs de 2^{n-1} à 2^n-1, mais en interprétant ces bits comme des nombres négatifs via le complément à deux.

Remarques

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 : 1 pour un nombre négatif et 0 pour un nombre positif. Pour représenter un nombre négatif N, on code $|N|$ et on prend son complément à deux.

On peut aussi remarquer que c'est la valeur de $2^n + N$ qui est stockée pour représenter N, mais il faudra convertir depuis la base 10 dans ce cas, ce qui semble moins pratique.

Exemple 1 Par exemple, pour 4 bits :

- Les nombres positifs vont de 0000_2 (0) à 0111_2 (7).
- Les nombres négatifs vont de 1000_2 (-8) à 1111_2 (-1).

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

Exemple 2 Représentation de l'entier -73 sur 8 bits :

- Avec le complément à 2
  - $73 = 01001001_2$
  - Complément à 1: 10110110_2
  - Ajout de 1 : 10110111_2
- Avec le calcul de $2^n + N$
  $2^8 - 73 = 183 = 10110111_2$

CC-BY-SA

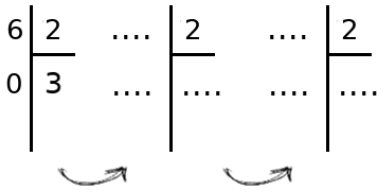
0.4 Exercices

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`. Pour convertir un nombre décimal dans une autre base (binaire mais aussi hexadécimale ou autre) il faut le diviser par la base d'arrivée. On effectue des divisions entières successives jusqu'à ce que le quotient soit nul.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.

CC-BY-SA

0.5 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. et 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.

0.5.1 Fonctions logiques

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.

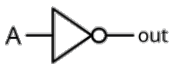
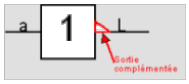
Méthode

- 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* : expression logique qui correspond à une combinaison spécifique des variables d'entrée d'une fonction logique, où chaque variable apparaît exactement une fois, soit sous sa forme directe A, soit sous sa forme complémentée $\overline{A}$.

Dans la suite, A et B sont deux valeurs logiques.

0.5.2 Porte NOT/NON

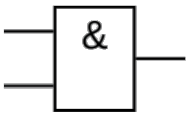
NON A est VRAI si A est FAUX, et réciproquement.

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

Entrée	Sortie
A	NON A
0	1
1	0

0.5.3 Porte ET/AND

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

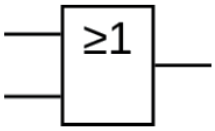
Type	Symbole américain	Symbole européen	Notation	Opération booléenne	Table de vérité
ET			$A \wedge B$	$A \cdot B$	

Entrée		Sortie
A	B	A ET B
0	0	0
0	1	0
1	0	0
1	1	1

Expression logique : A.B

0.5.4 Porte OU/OR

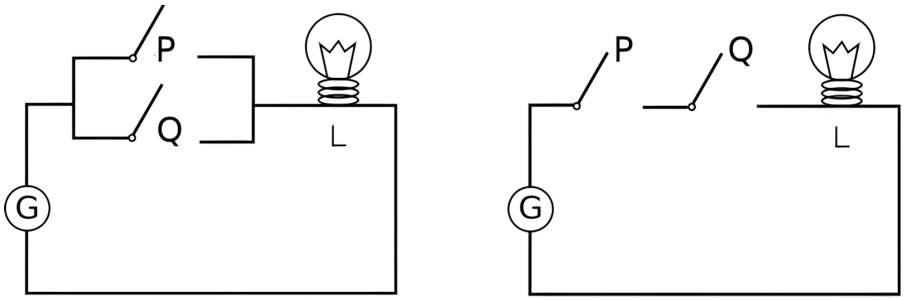
A OU B est VRAI si et seulement si A est VRAI OU B est VRAI (y compris si les deux sont VRAIS).

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

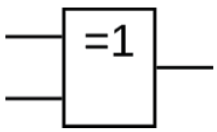
Expression logique : A+B

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 ?



0.5.5 Porte OU EXCLUSIF/XOR

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

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

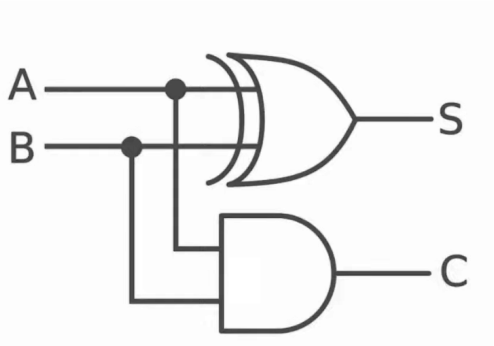
Entrée		Sortie
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

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

0.5.6 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		

Le demi-additionneur ne fait que la moitié du travail : il ne peut additionner que deux bits A et B et ne tient pas compte d'une éventuelle retenue d'entrée usuellement appelée *Carry In* (`Cin`) provenant d'une addition précédente.

0.6 Python

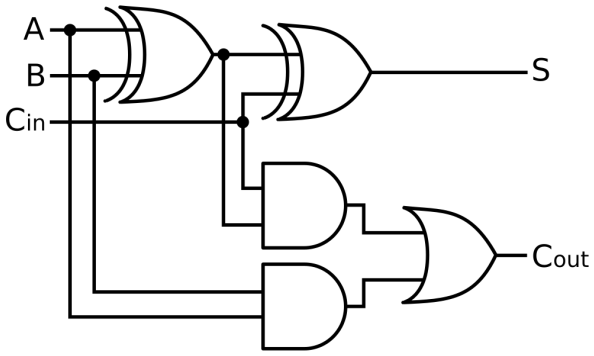
Réalisés par ces circuits électroniques que l'on appelle des portes logiques ou portes booléennes, les ordinateurs s'appuient sur la théorie de Boole pour effectuer des opérations logiques grâce à des opérateurs booléens. Un opérateur booléen (on dit aussi fonction booléenne) est une fonction mathématique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

CC-BY-SA

0.7 Exercices

Exercice 1 Dans le cours, nous avons réalisé un demi-additionneur sur 1 bit. On réalise un additionneur 1 *bit* complet en tenant compte d'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.

CC-BY-SA

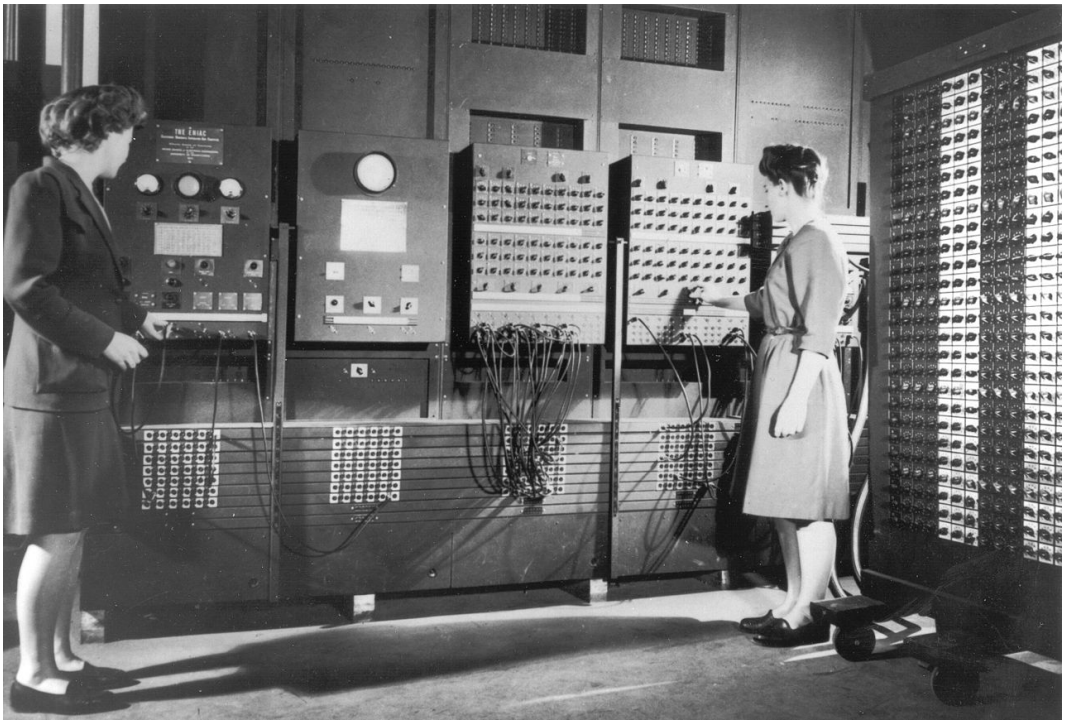


Figure 4: ENIAC

CC-BY-SA