

Modèle de Von Neumann

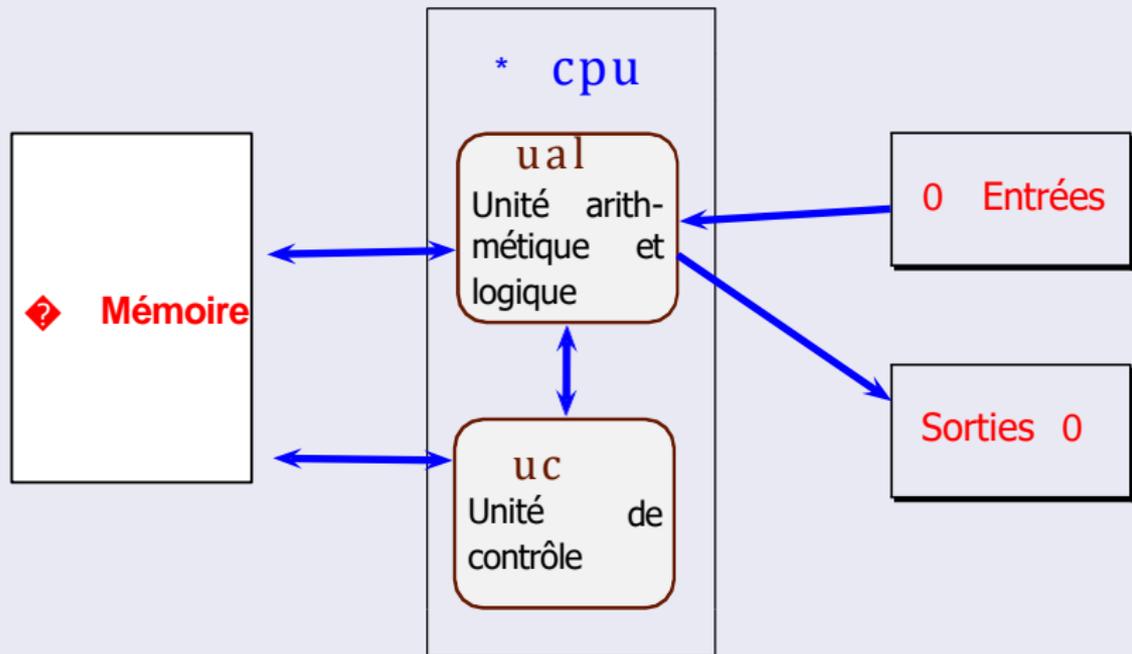
- Les ordinateurs modernes sont construits autour d'un modèle défini par le mathématicien John Von Neumann en 1945 et appelé **Architecture de Von Neumann**.
- Dans ce modèle, l'ordinateur se décompose en 5 parties distinctes :
 - 1 Les dispositifs d'**entrée** des données (ex : clavier, souris, écran tactile, réseau ...),
 - 2 La **mémoire** qui stocke les données et les programmes (ex : mémoire cache, ram, ...)
 - 3 L'**unité arithmétique et logique** qui effectue les opérations (addition, soustraction, comparaison, ...) sur les données.
 - 4 L'**unité de contrôle** qui est chargé de la gestion de l'ordre des opérations (séquençage)
 - 5 Les dispositifs de **sortie** des données (ex : écran, imprimante, ...)

Remarques :

- Dans les ordinateurs modernes, l'UAL et l'unité de contrôle sont regroupés dans le processeur (cpu pour Central Processing Unit en anglais)
- Certains périphériques sont à la fois des dispositifs d'entrée et de sortie. Par exemple, le disque dur car on peut y lire (entrée) et écrire (sortie) des données.
- Par rapport au modèle initial, les ordinateurs actuels possèdent parfois plusieurs processeurs ou coeurs.

C4 Architecture des ordinateurs

Schéma représentant l'architecture de Von Neumann :

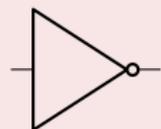


Remarques :

- Le composant de base des ordinateurs est le *transistor*, un composant électronique ne pouvant être que dans deux états. Soit il laisse passer le courant (état **1**), soit il ne le laisse pas passer (état **0**).
- Toutes les données représentées dans un ordinateur le sont donc sous forme de 0 et de 1.
- Dès les années 1850, dans des travaux sur la logique, le mathématicien britannique Georges Boole avait travaillé sur des variables ne pouvant prendre que deux valeurs 0 ou 1.
- On appelle, ces variables des **booléens**. On définit trois opérations de base que nous allons détailler sur les booléens : le **non**, le **et** et le **ou**.

Opérateur **non**

- Inverse la valeur de l'entrée
- Symbole électronique



Américain



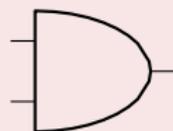
Européen

- Table de vérité

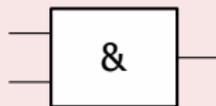
Entrée	Sortie
0	1
1	0

Opérateur **et**

- Vaut 1 lorsque les *deux* entrées valent un
- Symbole électronique



Américain



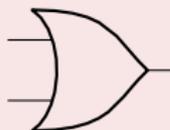
Européen

- Table de vérité

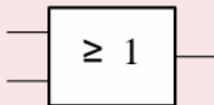
Entrée 1	Entrée 2	Sortie
0	0	0
1	0	0
0	1	0
1	1	1

Opérateur OR

- Vaut 1 lorsque l'une des deux entrées vaut 1
- Symbole électronique



Américain



Européen

- Table de vérité

Entrée 1	Entrée 2	Sortie
0	0	0
1	0	1
0	1	1
1	1	1

Autres portes logiques

Deux autres portes logiques sont fondamentales et bien que pouvant être construire à partir de OR, AND et NOT ont leur propre symbole :

- La porte XOR qui vaut 1 lorsque l'une des entrées vaut un mais pas les deux à la fois. C'est le ou exclusif.
- La porte NAND qui vaut 0 seulement lorsque les deux entrées valent 1. C'est la porte "NON ET"
- La porte NOR qui vaut 1 seulement lorsque les deux entrées valent 0. C'est la porte "NON OU"

Python et les booléens

- Python possède le type de variable booléen, les deux valeurs possibles sont : True et False.
- L'opération **non** s'obtient à l'aide de not
- L'opération **et** s'obtient à l'aide de and
- L'opération **ou** s'obtient à l'aide de or
- Les booléens de python peuvent donc être notamment des résultats de test de condition.

Exemple

```
# Définit une variable booléen ok qui vaut vrai
# lorsque au moins 2 des 3 variables a,b et c sont égales
ok=(a==b) or (a==c) or (b==c)
```

Circuit logique

- En combinant ces portes logiques, on réalise des circuits logiques permettant d'effectuer des opérations (additions, soustractions, comparaison, ...) sur les données stockées dans l'ordinateur.
- Voir TP sur le site de simulation de circuit logique : <https://circuitverse.org/>

Assembleur

- Le langage machine est le seul compréhensible par un processeur, il se compose uniquement de suites de 0 et de 1. Ecrire un programme avec ce langage est extrêmement compliqué pour un être humain.
- Le langage assembleur traduit en code mnémotechnique les instructions machine les rendant plus accessibles à un être humain.
Par exemple `ADD R2,R1,R0` signifie additionner les valeurs des registres R0 et R1 et stocker le résultat dans R2 se traduit en une suite de 0 et de 1.
- Chaque assembleur est spécifique à un processeur (ou à une famille de processeur).
- Dans les langages de haut niveau (comme Python), c'est l'interpréteur (ou le compilateur) qui se charge de la traduction des instructions en langage machine. Ces langages sont donc indépendants des processeurs utilisés.