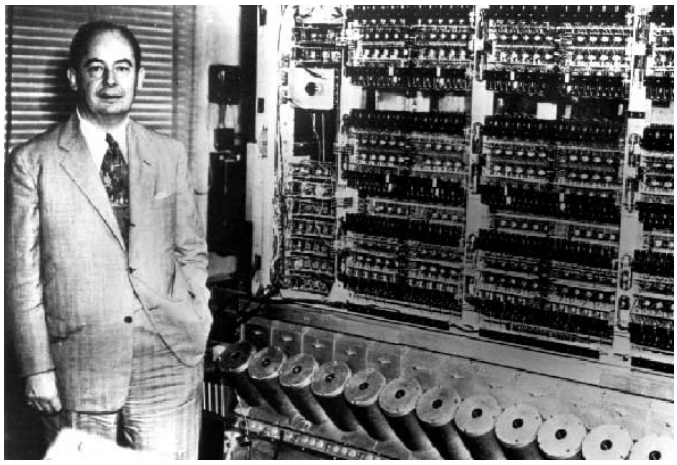


“

Il serait presque vain de dresser la liste des sujets sur lesquels John von Neumann a travaillé et de ceux dans lesquels il a permis des avancées certaines voire a agi en pionnier: découverte de l’ADN, théorie des jeux, théorie des ensembles, mécanique quantique, économie, sans parler de ses travaux sur l’intelligence artificielle qui préfigurent l’informatique actuelle, au point que «l’architecture de von Neumann» est encore aujourd’hui la base même du fonctionnement de la plupart des ordinateurs.

”

ARCHITECTURE SEQUENTIELLE :

- *Constituants d’une machine*
- *Langage machine*

Numérique et Sciences Informatiques
1ère

Support de cours :
Jean-Christophe BONNEFOY

Objectifs :

- Distinguer les rôles et les caractéristiques des différents constituants d’une machine.
- Dérouler l’exécution d’une séquence d’instructions simples du type langage machine.

1. Introduction

Nous allons voir ici comment fonctionne une "**machine**" informatique. Pour bien poser les choses, il faut prendre conscience que ce que nous appellerons machine peut correspondre à une grande gamme d'appareils informatiques : un **ordinateur** personnel bien sûr, mais aussi **smartphone**, ordinateur de bord de voiture, **robot** ou **objet connecté, etc..**.

En simplifiant, tous ces appareils ont plusieurs points communs :

- Ils perçoivent des signaux en entrée : clavier, souris, écran tactile, ondes radio, signaux électriques transmis par câble, signaux provenant de capteurs (vitesse, température etc.)
- Ils disposent d'un processeur qui effectue un traitement sur les signaux reçus en entrée.
- Ils produisent des signaux en sortie : sur un écran, sur une sortie audio, sur une carte réseau, sur une interface dédiée à des moteurs électriques etc.

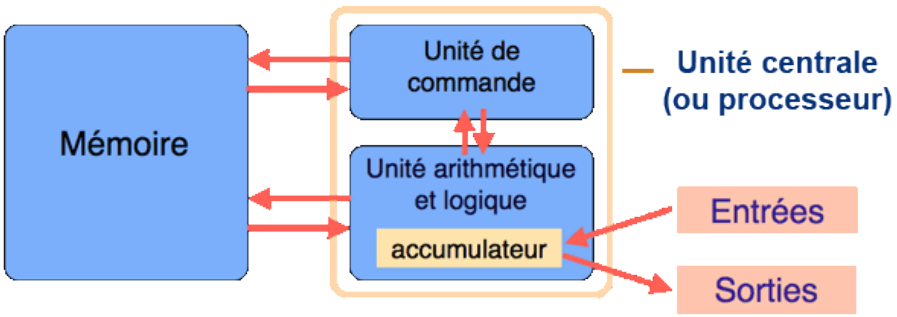
Nous allons nous pencher sur le premier modèle qui a été fait de ces machines informatiques : le modèle d'architecture séquentielle. On notera que ce modèle d'architecture a été conçu durant la seconde guerre mondiale à des fins initialement militaires (calculs balistiques, premier projet géré par Eckert) et a été finalisé par **John Von Neumann** en 1945.

Ce modèle est simpliste au regard de la complexité et de la diversité des machines actuelles, néanmoins il doit permettre de mieux appréhender comment fonctionne une machine informatique en la présentant sous une version simplifiée.

2. L'architecture Von Neumann

2.1. L'architecture Von Neumann

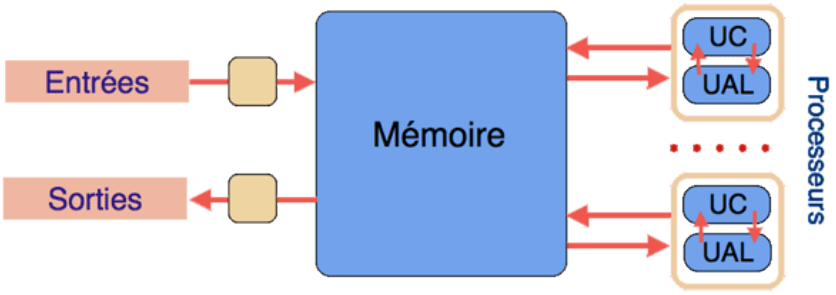
Cette architecture repose sur l'architecture suivante :



La première innovation est la séparation nette entre **l'unité de commande**, qui organise le flot de séquencement des instructions, et **l'unité arithmétique**, chargée de l'exécution proprement dite de ces instructions. La seconde innovation, la plus fondamentale, est l'idée du programme enregistré : les instructions, au lieu d'être codées sur un support externe (ruban, cartes, tableau de connexions), sont enregistrées dans la mémoire selon un codage conventionnel. Un compteur ordinal contient l'adresse de l'instruction en cours d'exécution ; il est automatiquement incrémenté après exécution de l'instruction, et explicitement modifié par les instructions de branchement. Un emplacement de mémoire peut contenir indifféremment des instructions et des données, et une conséquence majeure (dont toute la portée n'avait probablement pas été perçue à l'époque) est qu'un programme peut être traité comme une donnée par d'autres programmes.

2.2. Et aujourd'hui ?

Plus de soixante ans après son invention, le modèle d'architecture de von Neumann régit toujours l'architecture des ordinateurs.



Par rapport au schéma initial, on peut noter deux évolutions :

- Les entrées-sorties, initialement commandées par l'unité centrale, sont depuis le début des années 1960 sous le contrôle de processeurs autonomes (canaux d'entrée-sortie et mécanismes assimilés). Associée à la multiprogrammation (partage de la mémoire entre plusieurs programmes), cette organisation a notamment permis le développement des systèmes en temps partagé.

- Les ordinateurs comportent maintenant des processeurs multiples, qu'il s'agisse d'unités séparées ou de « cœurs » multiples à l'intérieur d'une même puce. Cette organisation permet d'atteindre une puissance globale de calcul élevée sans augmenter la vitesse des processeurs individuels, limitée par les capacités d'évacuation de la chaleur dans des circuits de plus en plus denses.

Ces deux évolutions ont pour conséquence de mettre la mémoire, plutôt que l'unité centrale, au centre de l'ordinateur, et d'augmenter le degré de parallélisme dans le traitement et la circulation de l'information. Mais elles ne remettent pas en cause les principes de base que sont la séparation entre traitement et commande et la notion de programme enregistré.

L'accès des processeurs à la mémoire se fait à travers un **bus** (non représenté sur la figure), voie d'échange assurant un transfert rapide de l'information. Mais au cours du temps, et pour des raisons technologiques, le débit du bus a crû moins vite que le débit d'accès à la mémoire et surtout que la vitesse des processeurs.

2.3. Mémoires

La mémoire est un tableau dans lequel sont stockés aussi bien les variables que les programmes. Chaque cellule de ce tableau possède une adresse X et la valeur de chaque cellule est notée M[X]. On peut distinguer :

- La mémoire interne vive (RAM : Random Access Memory) : Elle se présente le plus souvent sous la forme de barrettes que l'on peut ajouter ou retirer dans un ordinateur. C'est une mémoire **volatile**, c'est-à-dire que son contenu est effacé dés que l'alimentation électrique est coupée. Elle est accessible en lecture et en écriture.

- La mémoire interne morte (ROM : Read Only Memory) : Elle sert essentiellement au démarrage de l'ordinateur. C'est une mémoire **non volatile** uniquement accessible en lecture.

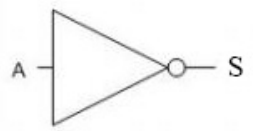
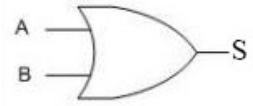
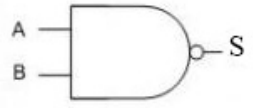
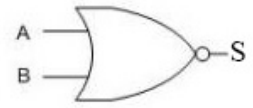
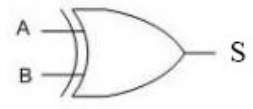
- La mémoire externe de masse : Ce sont les disques durs, les clés USB, les CD…

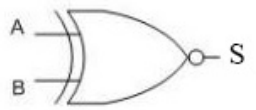
3. Fonctionnement

Une machine n'est pas intelligente. Elle exécute ce qu'on lui demande de faire. Son langage est le **langage binaire.** Son fonctionnement repose sur des circuits électroniques constitués de circuits logiques dont nous avons déjà parlé.

3.1. Circuits et fonctions logiques

Nous avons déjà défini dans la partie traitant de *« l'algèbre booléenne »* les fonctions de base. Il faut retenir que le composant de base est le transistor. Nous allons ici rappeler leurs tables de vérité et ajouter leur symbole universel :

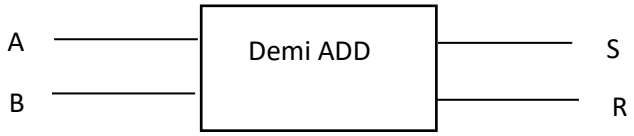
<table>
<tr><th>Portes logiques</th><th>Symbole</th><th>Ecriture booléenne</th><th>Table de vérité</th></tr>
<tr><td>NON
(NOT)</td><td> 
</td><td>$S = \bar{A}$</td><td>
<table>
<tr><td>ENTREE</td><td>SORTIE</td></tr>
<tr><td>A</td><td>S</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>ET
(AND)</td><td> 
</td><td>$S = A.B$</td><td>
<table>
<tr><td colspan="2">ENTREES</td><td>SORTIE</td></tr>
<tr><td>A</td><td>B</td><td>S</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>
<tr><td>OU
(OR)</td><td> 
</td><td>$S = A + B$</td><td>
<table>
<tr><td colspan="2">ENTREES</td><td>SORTIE</td></tr>
<tr><td>A</td><td>B</td><td>S</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>NON ET
(NAND)</td><td> 
</td><td>$S = \overline{A.B} = \bar{A} + \bar{B}$</td><td>
<table>
<tr><td colspan="2">ENTREES</td><td>SORTIE</td></tr>
<tr><td>A</td><td>B</td><td>S</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>NON OU
(NOR)</td><td> 
</td><td>$S = \overline{A + B} = \bar{A}.\bar{B}$</td><td>
<table>
<tr><td colspan="2">ENTREES</td><td>SORTIE</td></tr>
<tr><td>A</td><td>B</td><td>S</td></tr>
<tr><td>0</td><td>0</td><td>1</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>0</td></tr>
</table>
</td></tr>
<tr><td>OU EXCLUSIF
(XOR)</td><td> 
</td><td>$S = A \oplus B = \bar{A}.B + A.\bar{B}$</td><td>
<table>
<tr><td colspan="2">ENTREES</td><td>SORTIE</td></tr>
<tr><td>A</td><td>B</td><td>S</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
</table>

NON OU EXCLUSIF (XNOR)		$S = \overline{A \oplus B} = \overline{A}.\overline{B} + A.B$	

ENTREES		SORTIE
A	B	S
0	0	1
0	1	0
1	0	0
1	1	1

Il nous reste à passer des portes logiques aux calculs mathématiques. La première opération à définir physiquement est l'addition. Les autres opérations s'en déduiront.

On définit d'abord un circuit élémentaire appelé demi-additionneur 1 bit, qui réalise l'addition binaire de 2 nombres à 1 bit :



La table de vérité associée à ce circuit est donc :

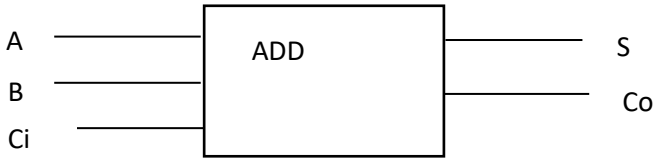
A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On en déduit les équations logiques :

$$S = A \oplus B$$

$$R = A.B$$

Pour passer à l'additionneur complet sur n bits, il faut tenir compte de la retenue et la propager au rang d'après. En résumé, cela revient à considérer le module suivant :



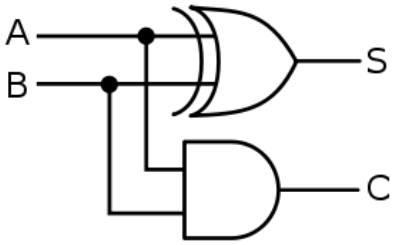
La table de vérité associée à ce circuit est donc :

A	B	Ci	S	Co
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

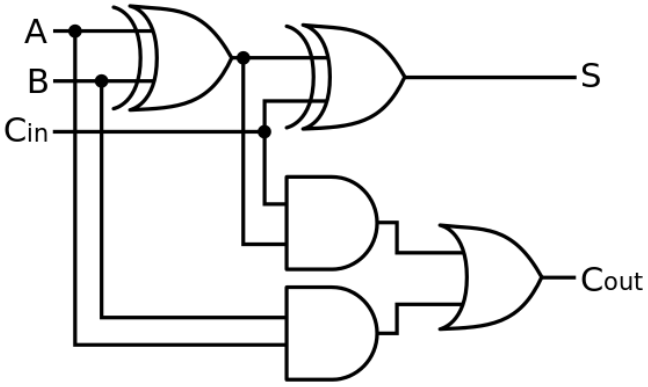
$$S = (A \oplus B) \oplus Ci$$

$$Co = (A.B) + ((A \oplus B).Ci)$$

Il ne nous reste plus qu'à donner le « câblage » logique. On appelle cela le **Logigramme**. On peut réaliser ces schémas à l'aide d'outil numérique qui permettent en plus de simuler. On prendra par exemple le logiciel *Logisim* qui se trouve sur internet à cette adresse : http://www.cburch.com/logisim/



Câblage du demi additionneur 1 bit



Câblage de l'additionneur 1 bit

3.2. Langage machine

Les instructions d'un programme écrit dans un langage de programmation de haut niveau, c'est-à-dire plus proche de langage humain, sont traduites pour pouvoir être comprises par la machine. Le langage de base est l'assembleur. Il existe presque autant de langages assembleur que de processeurs : un programme écrit en assembleur ne sera compris que par le processeur destinataire contrairement à un programme écrit en langage Python.

En assembleur, dans sa forme la plus simple, une instruction est stockée dans une cellule mémoire désignée par une adresse. Elle est constituée de 2 parties :

- Un code opération (CO) qui indique quelle opération doit être effectuée
- Une adresse (ADR) désignant les opérandes de cette opération

Pour travailler, le microprocesseur utilise de petites zones où il peut stocker des données. Ces zones portent le nom de registres et sont très rapides d'accès puisqu'elles sont implantées dans le microprocesseur lui-même et non dans la mémoire vive. Ces registres sont propres au microprocesseur et cela ne rentre pas dans le cadre du programme de NSI.

Exemple d'instruction en assembleur : MOV R0, #1

Prenons un exemple de programme simple en python :

```
x = 1
x =x +2
```

Et sa traduction en assembleur

```
MOV R0, #1
STR R0, x
LDR R0, x
MOV R1, #2
ADD R0, R0, R1
STR R0, x
```

Chaque opération machine correspond à un code binaire en interne.