I. Machine de Turing.

Dans les années 1930, quatre mathématiciens cherchent à définir ce qui est calculable : Stephen Kleene, Emil Post, Alonzo Church et Alan Turing. On parle alors de la calculabilité. Alan Turing va lier la notion de calcul à la notion de machine.

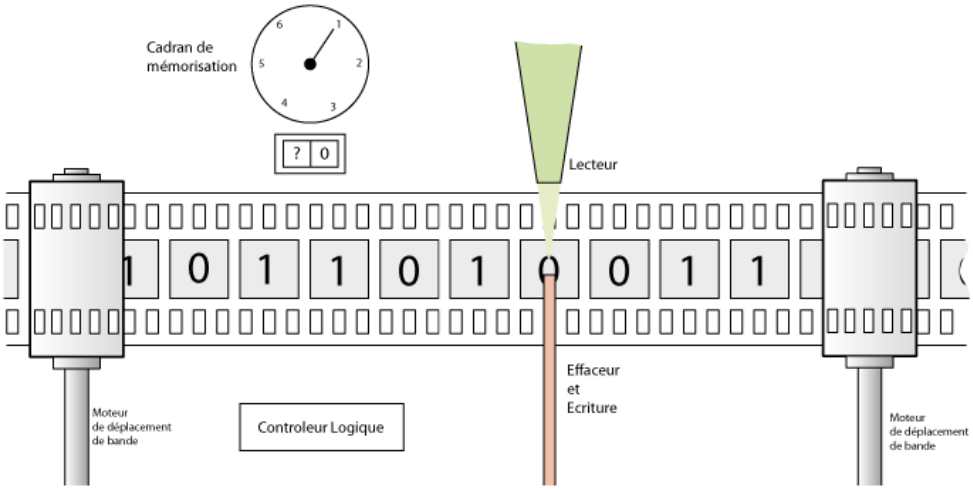
En 1936, Turing imagine une **machine abstraite (ou virtuelle)** capable de fonctionner sans intervention humaine. Cette machine, appelée **machine de Turing**, est la plus élémentaire possible et est destinée à mettre en œuvre des mécanismes de calcul, numériques ou symboliques, comme le font les ordinateurs actuels, qui n'existaient pas encore en 1936.

Cette machine de Turing est une sorte de version minimaliste d'une machine à écrire, contrôlée à l'aide d'un programme. Elle se caractérise par :

- **Un ruban infini** (en tout cas, aussi long que nécessaire) sur lequel la machine peut lire des données et en écrire d'autres. Ce ruban est divisé en cases, chacune pouvant contenir un symbole.

- **Une tête de lecture/écriture** positionnée à tout moment sur une des cases du ruban.

- **Un ensemble fini d'états** : quand la machine lit un symbole, elle réagit en fonction de son état actuel et du symbole lu en changeant d'état. Elle peut alors changer d'état, modifier le symbole sur le ruban et déplacer la tête de lecture d'un cran vers la droite ou vers la gauche (elle n'est pas forcée d'effectuer toutes ces actions).



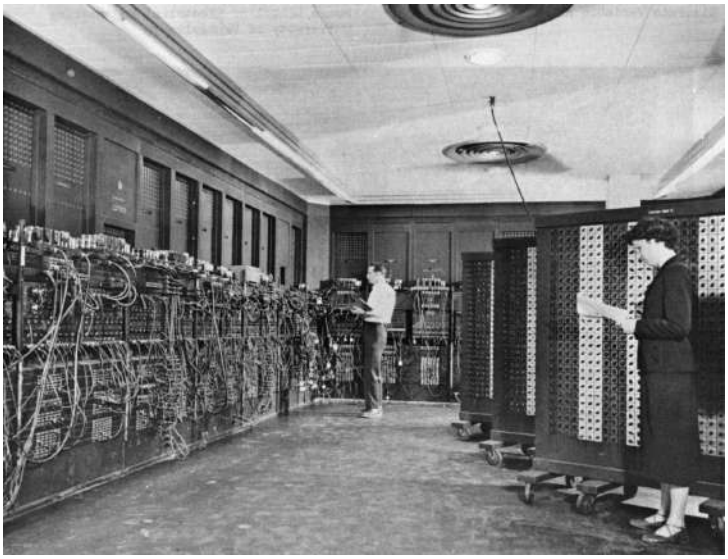
Nous ne détaillerons pas plus le fonctionnement d'une machine de Turing dans ce cours. Il faut retenir que la machine de Turing est un concept fondamental en informatique théorique et en théorie de la calculabilité. Elle permet de définir rigoureusement ce qu'est un algorithme et ce qui peut être calculé de manière mécanique.
Elle peut calculer tout ce que n'importe quel ordinateur physique peut calculer (même les plus puissants) et **ce qu'elle ne peut pas calculer ne peut pas, non plus, l'être par un ordinateur**. Les limites de la machine de Turing sont également les limites de tout ordinateur physique. Ainsi, elle résume à elle seule le concept d'ordinateur

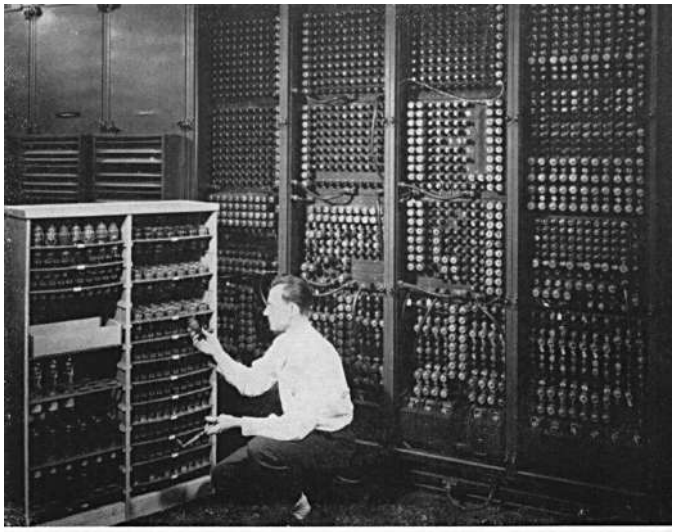
II. Modèle d'architecture de Von Neumann.

1. Un peu d'histoire.

La première réalisation concrète d'une "pseudo-machine" de Turing (et donc de ce qu'on peut considérer comme le premier ordinateur) est réalisée **en Allemagne en 1941 avec le Z3** (de l'ingénieur allemand Konrad Zuse).
Il était réalisé à partir de relais électromagnétiques (électroaimant qui permet l'ouverture et la fermeture d'un circuit électrique par un deuxième circuit).

La première réalisation électronique est **l'ENIAC conçu par les physiciens américains J.W. Mauchly et J. Eckert en 1945**. Il utilise alors des tubes électroniques qui sont des composants lourds, volumineux, fragiles et qui dégagent beaucoup de chaleur.





Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Le coût de fabrication et d'utilisation le réserve aux militaires qui financent l'ENIAC pour des calculs de balistique jusqu'en 1955. Malgré ses pannes fréquentes (une des causes était la combustion d'insectes sur un tube électronique chaud qui a entrainé le terme de "bug" en informatique), il permettait de faire des calculs qu'un homme faisaient à la main en plusieurs jours en quelques secondes (ces mêmes calculs sont faits de nos jours en quelques microsecondes).

Aujourd'hui, à la base de la plupart des composants d'un ordinateur, on retrouve **le transistor.**
La découverte du transistor a été faite fin 1947 par 3 américains (John Bardeen, William Shockley et Walter Brattain) et a conduit à l'abandon du tube électronique.
Le transistor ne présente que des qualités par rapport aux tubes : il est beaucoup plus petit, léger, robuste, fonctionne instantanément quand il est mis sous tension et consomme beaucoup moins d'électricité.

Actuellement, dans un ordinateur, les transistors sont regroupés au sein des circuits intégrés, ils sont gravés sur des plaques de silicium et les connexions entre les millions de transistors qui composent un circuit

intégré sont, elles aussi, gravées directement dans le silicium avec des gravures de plus en plus fines pour atteindre aujourd'hui les 2 nm (2×10^{-9} m ou 2×10^{-6} mm).

Les transistors se comportent comme des interrupteurs qui laissent ou non passer le courant électrique. Il n'y a pas d'autre état possible, **le courant passe ou ne passe pas. On parle d'état haut et on le symbolise avec un 1 quand le courant passe et on parle d'état bas et on le symbolise avec un 0 quand il ne passe pas.**
Bien sûr, ce ne sont pas des 0 et des 1 qui passent dans l'ordinateur c'est le courant qui circule ou qui ne circule pas ! Un ordinateur travaille "virtuellement" avec ces 2 chiffres les 0 et le 1 donc en base binaire.

Le transistor permet donc de réaliser des opérations booléennes. Ces opérations font partie de "l'algèbre de Boole" du nom du mathématicien Britannique Georges Boole (1815-1864).

Tout circuit électronique prend en entrée un ou plusieurs signaux électriques symbolisés par 0 ou 1 et donne, en sortie un ou plusieurs signaux électriques. Dans ce cours nous ne nous intéresserons qu'à des circuits dits **"combinatoires"** c'est à dire dont les états en sortie ne dépendent que des états en entrée (il existe aussi des circuits dits séquentiels mais nous ne les aborderons pas).

2. Architecture de Von Neumann.

L'architecture des ordinateurs actuels repose sur le modèle de Von Neumann. Von Neumann (mathématicien et physicien americano-hongrois 1903-1957) a intégré l'équipe qui a travaillé dans le projet ENIAC en 1944 et il publia un rapport sur la conception d'un autre ordinateur en 1945 (l'EDVAC) dans lequel il décrit un schéma d'architecture de **calculateur organisé en 3 éléments :**

- **unité arithmétique,**
- **unité de commande et**
- **une mémoire contenant programmes et données.**

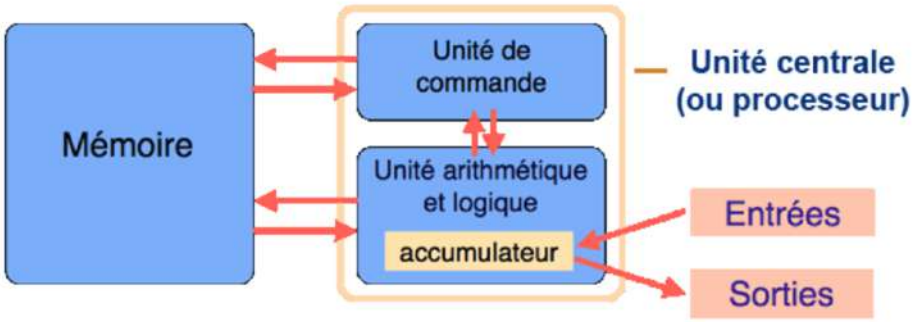
Il décrit des principes de réalisation pour ces éléments, notamment les opérations arithmétiques. Ce modèle (dit de Von Neumann) marque une transition profonde avec les pratiques antérieures et reste encore valable de nos jours.

- La première innovation est la **séparation entre l'unité de contrôle** qui est chargée d'organiser le flot des instructions et **l'unité arithmétique et logique** qui est chargée de l'exécution de ces instructions.
- La seconde innovation est l'idée **du programme enregistré,** fini les cartes à trous ou les rubans. Les instructions et données sont enregistrées dans la mémoire (qui était inférieure à 5 ko). Un compteur (ou pointeur d'instruction) contient l'adresse de l'instruction en cours d'exécution, il est incrémenté après l'exécution de l'instruction et modifié par les instructions de branchement.

Le modèle de Von Neumann est formé par **4 composants principaux** :

- **l'unité arithmétique et logique (UAL ou ALU en anglais)**, son rôle est d'effectuer les opérations de base (l'accumulateur est une mémoire interne de l'UAL appelée registre permettant de stocker les résultats intermédiaires lors d'un calcul).
- **l'unité de contrôle** qui permet d'exécuter les instructions des programmes ;
- **la mémoire** qui contient des instructions et des données ;

- **Les entrées et sorties** : périphériques qui permettent une communication entre utilisateur et machine (de nos jours : clavier, souris, écrans…).



Une horloge doit synchroniser les différents éléments qui échangent des informations **par un bus qui est un canal de communication** entre la mémoire, le processeur et les périphériques.
Il existe 3 grands types de bus :

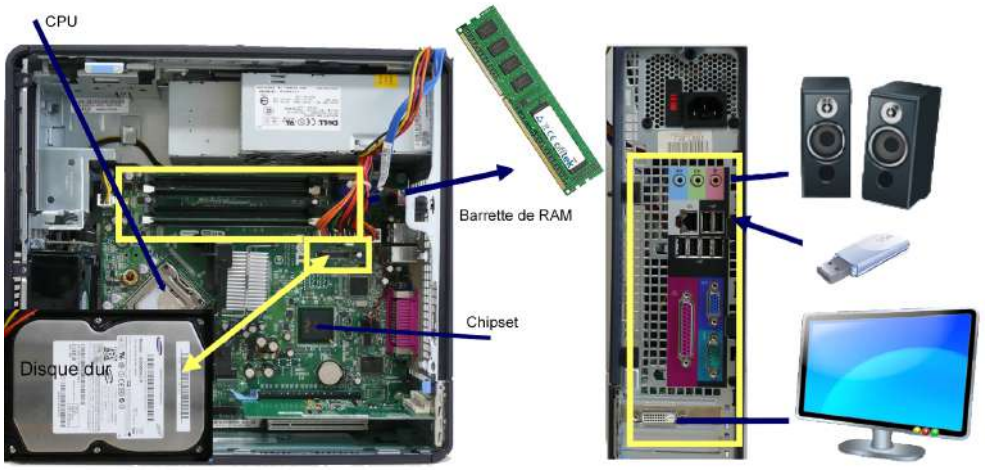
- Le bus d'adresse permet de faire circuler des adresses (par exemple l'adresse d'une donnée à aller chercher en mémoire).
- Le bus de données permet de faire circuler des données.
- Le bus de contrôle permet de spécifier le type d'action (exemples : écriture d'une donnée en mémoire, lecture d'une donnée en mémoire).

3. Un ordinateur de nos jours.

Physiquement, un ordinateur est constitué de plusieurs éléments :

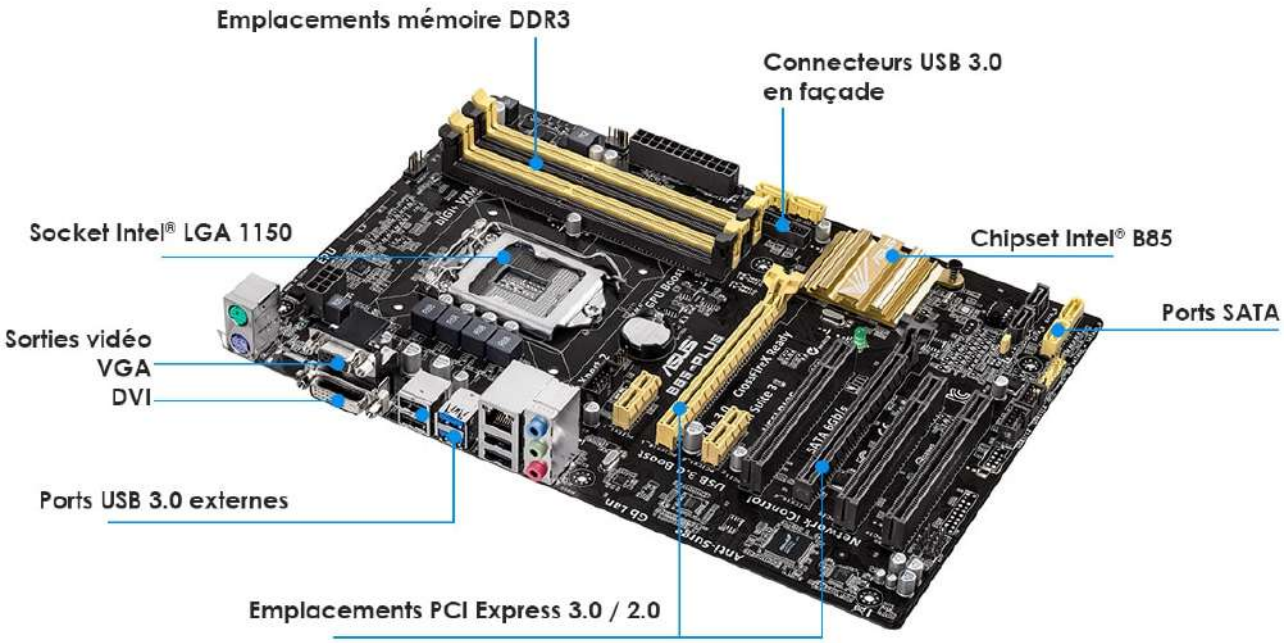
- Une machine (Unité Centrale) : un boitier contenant une carte mère avec un microprocesseur (en anglais Central Processing Unit ou CPU), des barrettes mémoire (RAM), une carte graphique, une carte réseau, des ports de communication (ports Ethernet, usb, Wifi, Bluetooth…) et différents périphériques comme un SSD par exemple.

- Des périphériques externes comme un moniteur, un clavier, une souris, une imprimante, etc. qui se connectent aux ports de communication.



La carte mère est un circuit imprimé qui est l'élément central d'un ordinateur. Le flux de données entre le processeur, la mémoire et les périphériques **est géré** par un composant électronique présent sur la carte mère : le **chipset.** Son rôle est de recueillir une certaine partie des informations pour ensuite les renvoyer au processeur afin qu'elles soient traitées. C'est ce composant qui détermine la vitesse des bus de communication qui permettent les échanges d'informations entre les différents composants.

Carte mère détaillée :



L'architecture de Von Neumann a peu évolué depuis sa création. Néanmoins il existe quelques petites différences.

On parle de mémoires (avec un "s") car il existe **plusieurs types de mémoire** dans un ordinateur :

- Les mémoires de grande capacité sont **appelées mémoire de masse**, comme **les SSD** ou les clés USB. Sur ces mémoires, **les données sont conservées** de manière permanente même lorsqu'elles ne sont plus alimentées électriquement. Ces mémoires sont physiquement éloignées du CPU et sont relativement lentes (même les SSD) en lecture comme en écriture.

- **La RAM** (random access memory) ou mémoire vive est une mémoire dite **volatile** : **les données sont perdues dès qu'elle n'est plus alimenté électriquement** (donc dès qu'on éteint l'ordinateur). Elle stocke les données et les programmes exécutés par le processeur. Elle est accessible aussi bien en lecture qu'en écriture. On peut la représenter comme un ensemble de cases de même taille ou chaque case a sa propre adresse et une taille qui lui permet de stocker 1 octet (8 bits), **la mémoire ne gère pas les bits 1 par 1, mais 8 par 8**. La RAM est située sur la carte mère (donc assez proche du processeur) et ses accès sont beaucoup plus rapide que les accès aux mémoires de masse.

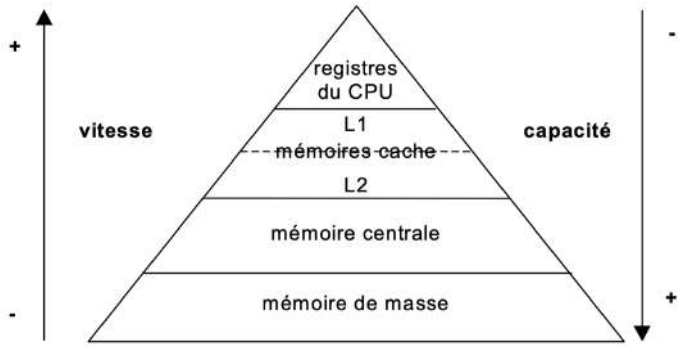
- **La ROM** (read only memory), aussi appelée mémoire morte est une mémoire **accessible uniquement en lecture.** C'est une mémoire non volatile (comme les mémoires de masse) et c'est elle qui contient le nécessaire qui permet à l'ordinateur de démarrer.

- **Les registres** sont des petits composants de stockage **situés dans le CPU**. Ce sont les mémoires caractérisées par une **très grande vitesse d'accès** (le processeur accède aux registres dans un cycle d'horloge du CPU) et qui servent au stockage de nombres et de résultats intermédiaires des programmes qui sont en cours de traitement par le processeur.

- **La mémoire cache** du processeur est une mémoire rapide, **proche des cœurs du CPU** et permettant de **réduire les délais d'attente** des informations stockées en mémoire vive. Elle va stocker les informations les plus récemment et les plus fréquemment traitées par ce dernier. Cette mémoire très rapide (il existe différent niveaux) va ainsi lui permettre d'accéder plus rapidement à ce type de données, et donc d'effectuer avec bien plus de vélocité ses calculs. Les mémoires cache des processeurs peuvent aller jusqu'à 30 Mo de cache, mais une capacité de 4 ou 6 Mo est très convenable pour l'utilisation classique d'un PC. Elle est beaucoup plus rapide que la RAM mais son coût est bien plus élevé.

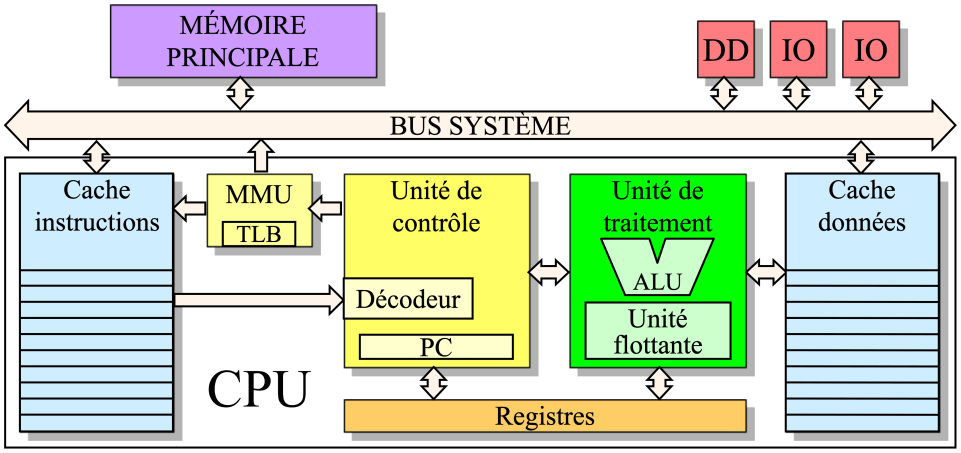
À retenir : Pour les éléments de mémoire, **plus on s'éloigne du CPU, plus les temps d'accès s'allongent et leur capacité de stockage grandit.**

Hiérarchie des différentes mémoires :



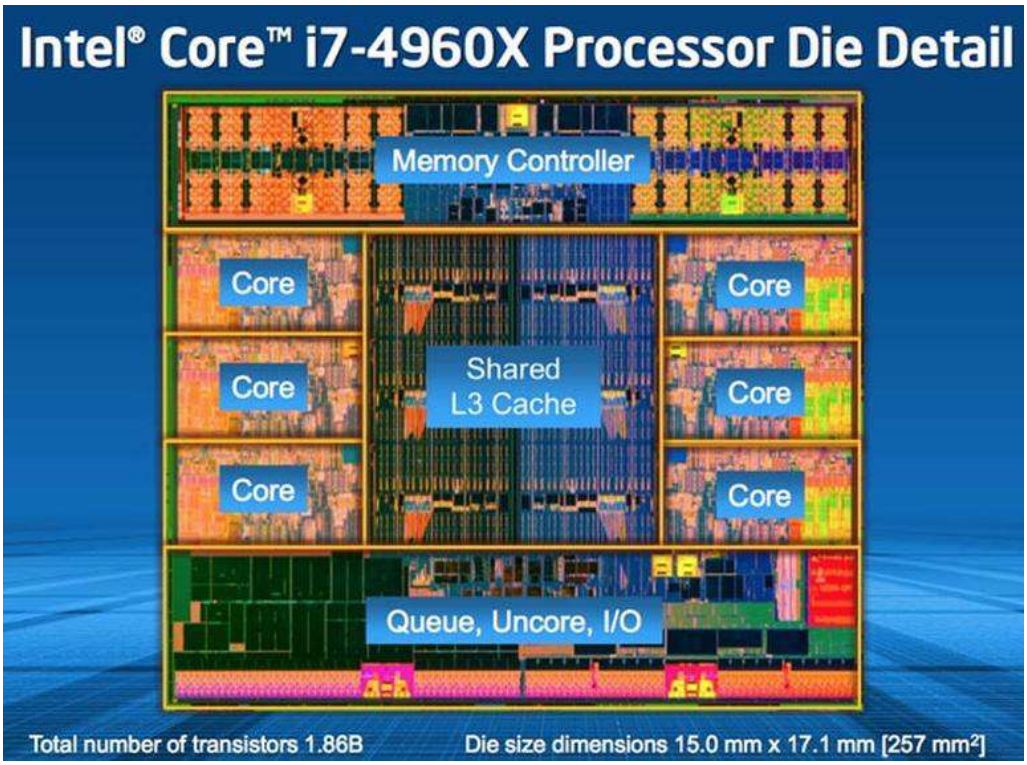
Support	Temps d'accès	Débit
Registres	1 ns	-
Cache	2-3 ns	-
Mémoire RAM	5-60 ns	1-20 Gio/s
SSD	0,05-0,1 ms	500 Mio/s - 7 Gio/s

On peut résumer le fonctionnement d'un ordinateur de nos jours par le schéma suivant :



La vitesse de calcul des ordinateurs n'a cessé d'augmenter. Pour faire augmenter cette vitesse, le principal levier était la fréquence d'horloge mais la température des processeurs augmente avec l'augmentation de cette fréquence. Depuis de début des années 2000, les fabricants de processeurs ont cessé d'augmenter les fréquences, ils augmentent le nombre de cœurs (et donc le nombre de registres qui sont situés dans ces

cœurs). On parle alors de **processeurs multi-cœurs (multi-core en anglais).** Cela ne résout pas tous les problèmes puisque ces cœurs doivent se partager les mêmes mémoires cache, la même RAM et il faut que les programmes soient écrits pour pouvoir utiliser ce type d'architecture si on veut profiter des différents cœurs.



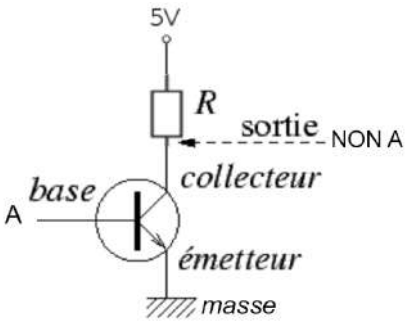
III. Circuits et fonctions booléennes.

1. Les portes logiques.

Le transistor est un interrupteur contrôlé électroniquement, sans partie mécanique. Si on branche plusieurs transistors à la suite, avec la sortie d'un transistor branché à la base d'un autre, on obtient un circuit où un transistor peut contrôler ceux qui le suivent. Avec un grand nombre d'enchaînements de ce type, on arrive à faire des circuits plus évolués.

Définition : Une fonction qui prend en entrée un ou plusieurs bits et qui produit en sortie un bit est appelée porte logique.

a. Porte NON : Le plus simple des circuits combinatoires est **la porte NON** (**NOT** en anglais) **qui inverse l'état en entrée.**
Elle prend un seul bit en entrée et sa sortie vaut 0 quand l'entrée vaut 1 et 1 quand l'entrée vaut 0.
On peut obtenir ce circuit avec un seul transistor ainsi branché :



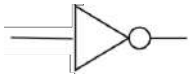
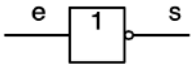
Si la tension d'entrée A est 0, le transistor se bloque et n'est pas conducteur (l'interrupteur est ouvert) et la tension de sortie est proche de celle avant la résistance, donc à un niveau haut c'est à dire 1.
Si la tension d'entrée A est 1, le transistor bascule, il est alors conducteur (l'interrupteur est fermé) et la tension de sortie est proche de la masse c'est à dire 0.

La résistance R est conçue pour limiter le courant à travers le transistor, dans ce dernier cas.

On peut représenter ça dans ce que l'on appelle **la table de vérité** de la **porte NON**.

Porte NON (NOT)	
A	non A
1	0
0	1

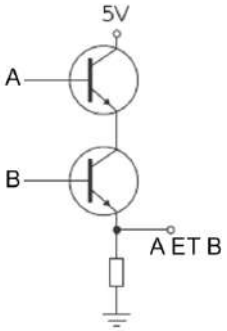
On symbolise la porte NON (NOT) par les symboles :

 (notation américaine) et 

(notation européenne)

Un seul transistor suffit à obtenir une porte NON et en combinant plusieurs transistors, on peut alors fabriquer d'autres portes logiques.

b. Porte ET : En combinant **2 transistors en série**, on peut fabriquer la **porte ET** (**AND** en anglais).
Si on met deux transistors en série, alors le second, quel que soit son état (bloquant ou passant), aura un comportement qui dépendra du premier. Pour que le second laisse passer le courant, il faut obligatoirement que le premier laisse passer le courant lui-aussi :

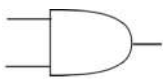
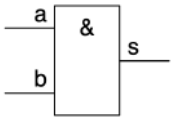


- si les entrée A et B sont nulles (tension nulle), alors la sortie est nulle ;
- si A est nulle mais que B n'est pas nulle, alors la sortie est nulle ;
- si B est nulle mais que A n'est pas nulle, alors la sortie est nulle ;
- si A et B ne sont pas nulle, alors la sortie n'est pas nulle (la tension à la sortie est ici de 5 V).

Ce dispositif permet de fabriquer une **porte ET : le courant passe uniquement si l'entrée A et l'entrée B sont sous tension.** On peut alors représenter **la table de vérité de la porte ET** :

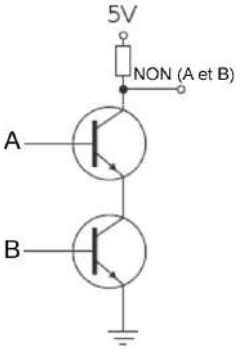
Porte ET (AND)		
A	B	A et B
0	0	0
1	0	0
0	1	0
1	1	1

On symbolise la porte ET (AND) par les symboles :

 (notation américaine) et 

(notation européenne)

c. **Porte NON ET :** En "déplaçant" la résistance et la sortie, on obtient la porte **NON ET (NAND en anglais)** :

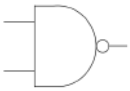
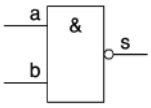


- si les entrée A et B sont nulles alors la sortie n'est pas nulle ;
- si A est nulle mais que B n'est pas nulle, alors la sortie n'est pas nulle ;
- si B est nulle mais que A n'est pas nulle, alors la sortie n'est pas nulle ;
- si A et B ne sont pas nulle, alors la sortie est nulle.

On peut représenter **la table de vérité de la porte NON ET (NAND) :**

Porte NON ET (NAND)		
A	B	non(A et B)
0	0	1
1	0	1
0	1	1
1	1	0

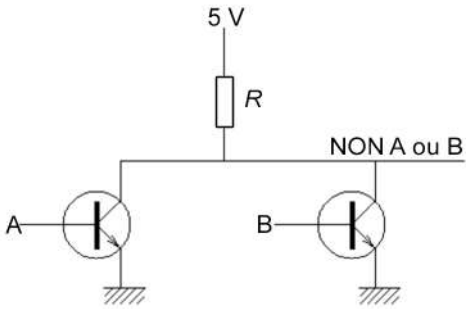
On symbolise la porte NON ET (NAND) par les symboles :

 (notation américaine) et 

(notation européenne)

d. Porte NON OU :

Sur le même principe, le montage de deux transistors en parallèle permet d'obtenir une porte **NON OU (NOR** en anglais) :

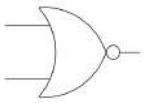
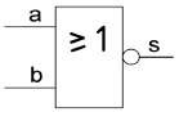


- si les entrée A et B sont nulles alors la sortie n'est pas nulle ;
- si A est nulle mais que B n'est pas nulle, alors la sortie est nulle ;
- si B est nulle mais que A n'est pas nulle, alors la sortie est nulle ;
- si A et B ne sont pas nulle, alors la sortie est nulle.

On peut représenter **la table de vérité de la porte NON OU (NOR) :**

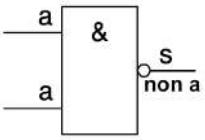
Porte NON OU (NOR)		
A	B	non(A ou B)
0	0	1
1	0	0
0	1	0
1	1	0

On symbolise la porte **NON OU (NOR)** par les symboles :

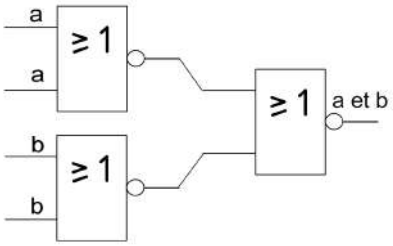
 (notation américaine) et 

(notation européenne)

Les portes NAND et NOR sont fondamentales dans les circuits électroniques car **tout circuit peut être conçu en utilisant uniquement ces 2 portes** (**elles sont dites complètes**). Par exemple, on peut fabriquer la porte NOT à partir d'une porte NAND en reliant les 2 entrées de cette porte comme ci-contre :



ou encore la porte ET avec 2 portes NOR :

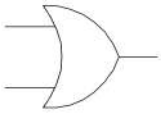
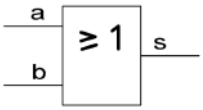


e. Porte OU :

On peut obtenir la porte OU (OR en anglais) sur le même principe avec plusieurs portes NAND ou encore avec un inverseur (porte NON) et la porte NON OU (NOR). On obtient alors **la table de vérité de la porte OU (OR) :**

Porte OU (OR)		
A	B	A ou B
0	0	0
1	0	1
0	1	1
1	1	1

On symbolise la porte OU (OR) par les symboles :

 (notation américaine) et 

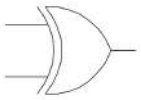
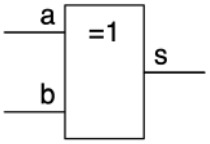
(notation européenne)

f. Porte OU EXCLUSIF :

On trouve encore d'autres portes logiques "de base" comme XOR (OU EXCLUSIF) ou encore XNOR (pour NOT XOR). La table de vérité de la porte **XOR (OU EXCLUSIF)** est :

Porte XOR		
A	B	A xor B
0	0	0
1	0	1
0	1	1
1	1	0

On symbolise la porte **OU EXCLUSIF (XOR)** par les symboles :

 (notation américaine) et 

(notation européenne)

La porte XOR est plus complexe à réaliser, mais comme pour les autres portes logiques, on peut la réaliser à partir de transistors.

Il faut maintenant passer des portes logiques aux calculs mathématiques. Pour cela, on combine différentes portes logiques pour obtenir des circuits plus complexes.

2. Exemples de circuits combinatoires.

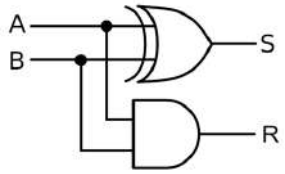
Les circuits électroniques possèdent plusieurs entrées et plusieurs sorties. **Quand les sorties dépendent directement et uniquement des entrées, on parle de circuits combinatoires** (ce ne sont pas les seuls qui existent). La construction des circuits électroniques ressemble à un "jeu de Lego" dans lequel on assemble différents circuits "élémentaires" pour obtenir des circuits plus complexes.

Parmi ces circuits, il y a, par exemple, des décodeurs qui permettent de sélectionner une sortie à partir des entrées, des multiplexeurs, qui permettent de sélectionner une entrée de données à partir des entrées de contrôle ou encore des circuits arithmétiques qui permettent de faire des opérations de base sur des nombres binaires.

a. Circuit demi-additionneur 1 bit.

Un circuit additionneur est un circuit qui admet en entrée 2 nombres binaires de n bits et qui fournit en sortie le résultat de la somme binaire de ces nombres sur n+1 bits. C'est un circuit complexe à réaliser. Le **demi-additionneur** (1 bit) est un circuit beaucoup plus simple. Il **prend en entrée 2 nombres binaires sur 1 bit et il a 2 sorties : somme et retenue,** notées S et R sur 1 bit. Par exemple 1 + 0 donne 01, avec S=1 (la somme) et R=0 (la retenue) ou encore 1 + 1 donne 10 avec S=0 et R=1.

Le schéma d'un demi-additionneur est :



Avec une table de vérité, on vérifie qu'on obtient bien la somme et la retenue sur 1 bit avec ce circuit :

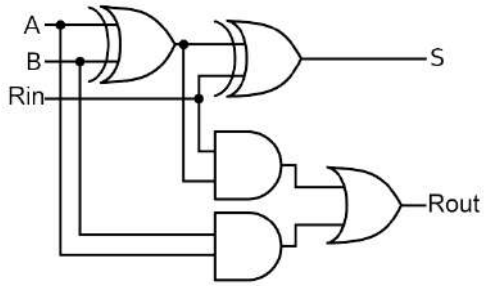
A	B	R (A and B)	S (A xor B)
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	0

On obtient bien les bons résultats (en binaire) : 1 + 1 = 10 ; 1 + 0 = 01 ; 0 + 1 = 01 ; 0 + 0 = 00.

b. Circuit additionneur complet 1 bit.

Un additionneur complet 1 bit, permet d'additionner 2 nombres binaires A et B sur 1 bit en tenant compte de la retenue entrante ("Rin"). En sortie on obtient la somme (S) et la retenue sortante ("Rout").

Le schéma d'un additionneur complet 1 bit est :



Avec une table de vérité, on vérifie qu'on obtient le résultat attendu :

A	B	Rin	Rout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

En mettant en cascade des additionneurs de 1 bit, on obtient des additionneurs capables d'additionner des nombres sur n bits.

Ce qu'il faut retenir, c'est que la base de tout ce qui permet de concevoir nos ordinateurs est le transistor, une combinaison de transistors permet d'obtenir des circuits logiques (sous forme de circuit intégré), la

combinaison de circuits logiques permet d'obtenir des circuits plus complexes (additionneur, décodeur, multiplexeur…), et ainsi de suite... jusqu'à obtenir le processeur et la ram.

Pour simuler un circuit : https://logic.ly/demo

IV. Le langage machine.

Comme nous l'avons déjà vu, un ordinateur exécute des programmes qui sont des suites d'instructions. Le CPU est incapable d'exécuter directement des programmes écrits, par exemple, en Python ou en C. Comme tous les autres composants d'un ordinateur, le CPU gère uniquement 2 états (symbolisés par un "1" et un "0"), **les instructions exécutées au niveau du CPU sont donc codées en binaire**. **Le langage machine est la suite de bits qui est interprétée par le processeur** de l'ordinateur lors de l'exécution d'un programme. C'est le langage natif du processeur, et le seul qui soit reconnu par celui-ci.
Un microprocesseur est incapable d'interpréter une phrase aussi simple que "ajoute le nombre 24 et la valeur située dans le registre R1, range le résultat dans le registre R2", tout doit être codé sous forme binaire.
Un programme en langage machine est donc une suite (très longue) de "1" et de "0", ainsi une seule erreur entre un 1 et un 0, et le programme ne fonctionne pas, et il faudra bien de la patience et du courage pour retrouver l'erreur !
Programmer en langage machine est extrêmement difficile et ce n'est plus du tout utilisé. Pour pallier à cette difficulté, les informaticiens ont remplacé les codes binaires par des mots plus faciles à retenir que des suites de "1" et de "0". Nous avons toujours des instructions machines du genre " ajoute le nombre 24 et la valeur située dans le registre R1, range le résultat dans le registre R2", mais au lieu d'écrire cette phrase avec des 0 et des 1, on peut écrire quelque chose comme "ADD R2, R1, #24". Dans les 2 cas, la signification est identique pour le processeur.
Le processeur n'étant capable d'interpréter que le langage machine (les 0 et les 1), **un programme appelé "assembleur" se charge du passage de "ADD R2, R1, #24" à une suite de 0 et de 1.** On dit alors que l'on **programme en assembleur** quand on écrit des programmes avec ces mots (appelés mnémoniques). Même de nos jours, l'écriture de programmes en assembleur est encore (un peu) utilisée. Des gros programmes ont été écrits entièrement en assembleur pour les ordinateurs, comme le système d'exploitation DOS de l'IBM PC (environ 4000 lignes de code) ou le tableur Lotus 1-2-3. Dans les années 1990, c'était aussi le cas pour la plupart des jeux pour consoles vidéo (par exemple pour la Mega Drive ou la Super Nintendo).
Le langage assembleur est le langage de plus bas niveau que l'on peut encore appréhender et que le processeur sait traduire directement en langage machine.
L'assembleur dépend fortement du type de processeur. Donc il n'y a pas de langage d'assembleur unique et chaque constructeur de processeur a son assembleur. Si un processeur A est capable d'exécuter toutes les instructions du processeur B, on dit que A est compatible avec B (l'inverse n'est pas forcément vrai).

Un microprocesseur exécute un jeu d'instructions relatif au programme créé par le concepteur. Une instruction réalise une action simple sur le microprocesseur, comme "récupérer un nombre en mémoire", "additionner deux nombres et placer le résultat en mémoire". **Chaque instruction machine est composée d'un nombre en binaire codant l'opération à réaliser codé sur 1 ou 2 octets** (par exemple le code "00100110" donne l'ordre au CPU d'effectuer une multiplication) **suivi des opérandes**, **c'est à dire les données** sur lesquelles l'opération est effectuée.

Les instructions machines sont relativement basiques (instructions de bas niveau), par exemple :

- **Les instructions arithmétiques** (addition, soustraction, multiplication...). On peut avoir une instruction qui ressemble à "additionne la valeur contenue dans le registre R1 et le nombre 789 et range le résultat dans le registre R0".

- **Les instructions de transfert de données** qui permettent de transférer une donnée d'un registre du CPU vers la mémoire vive et vice versa. On peut avoir une instruction qui ressemble à "prendre la valeur située à l'adresse mémoire 487 et la placer dans le registre R2" ou encore "prendre la valeur située dans le registre R1 et la placer à l'adresse mémoire 512" (normalement les adresses mémoires sont codées en binaire…).

- **Les instructions de rupture de séquence** : les instructions machines sont situées en mémoire vive, si, par exemple, l'instruction n°1 est située à l'adresse mémoire 343, l'instruction n°2 sera située à l'adresse mémoire 344, l'instruction n°3 sera située à l'adresse mémoire 345, etc. Au cours de l'exécution d'un programme, le CPU passe d'une instruction à une autre en passant d'une adresse mémoire à l'adresse mémoire suivante, par exemple après avoir exécuté l'instruction n°2 située à l'adresse mémoire 344, le CPU "va chercher" l'instruction suivante à l'adresse mémoire 345. Les instructions de rupture de séquence d'exécution (ou instructions de saut) permettent d'interrompre l'ordre initial en passant à une instruction située une adresse mémoire donnée. Par exemple, on peut avoir une instruction qui ressemble à cela : imaginons qu'à l'adresse mémoire 354 nous avons l'instruction "si la valeur contenue dans le registre R1 est strictement supérieure à 0 alors exécuter l'instruction située à l'adresse mémoire 4521". Si la valeur contenue dans le registre R1 est strictement supérieure à 0 alors la prochaine instruction à exécuter est l'adresse mémoire 4521, dans le contraire, la prochaine instruction à exécuter est à l'adresse mémoire 355.

Il y a 3 possibilités pour accéder à un opérande :

- il est directement saisi,
- il est dans un registre : il faut alors indiquer le nom de ce registre,
- il est dans la mémoire vive : il faut indiquer son adresse

Le but n'est pas d'apprendre à programmer en assembleur, nous allons seulement voir quelques exemples d'instructions en assembleur. Simulateur : http://www.peterhigginson.co.uk/AQA/

Pour additionner 60 avec 15, le code en assembleur sera le suivant :

MOV R0, #60 *Place la valeur 60 (valeur immédiate identifiée grâce au #) dans le registre R0*
ADD R0, #15 *Ajoute le nombre 15 à la valeur stockée dans R0 et replace le résultat dans R0*

Pour placer la valeur stockée à l'adresse mémoire 12 (par souci de simplification, on continue à utiliser des adresses mémoire codées en base 10) dans le registre R1 :

LDR R1, 12

Pour placer la valeur stockée dans le registre R2 en mémoire vive à l'adresse 27 :

STR R2,27

Pour ajouter la valeur stockée dans le registre R1 et la valeur stockée dans le registre R0 puis placer le résultat dans le registre R2 :

ADD R2, R1, R0

Pour soustraire le nombre 87 de la valeur stockée dans le registre R0 et placer le résultat dans le registre R1 :
SUB R1, R0, #87

Pour placer la valeur stockée dans le registre R2 dans le registre R1 :
MOV R1, R2

Pour faire une rupture de séquence si la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 45 :
B 45

Pour comparer la valeur stockée dans le registre R0 et le nombre 31 :
CMP R0, #31
Cette instruction CMP doit précéder une instruction de branchement conditionnel BEQ, BNE, BGT, BLT (tests) :
Par exemple : la prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 **est égale** à 31 :
CMP R0, #31
BEQ 78
La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 **n'est pas égale** à 31 :
CMP R0, #31
BNE 78
La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 **est plus grande** que 31 :
CMP R0, #31
BGT 78
La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est **plus petite** que 31 :
CMP R0, #31
BLT 78

Enfin, pour arrêter l'exécution du programme :
HALT