MODÈLE D'ARCHITECTURE SÉQUENTIELLE
(MODÈLE DE VON NEUMAN)

1. Algèbre de Boole

Le transistor est l'élément de base des circuits logiques. Un circuit logique permet de réaliser une opération booléenne. Ces opérations booléennes sont directement liées à l'algèbre de Boole (Georges Boole, mathématicien Britannique 1815-1864). L'étude de l'algèbre de Boole dépasse le cadre de ce cours, vous devez juste savoir qu'un circuit logique prend en entrée un ou des signaux électriques (chaque entrée est dans un état "haut" (symbolisé par un "1") ou à un état "bas" (symbolisé par un "0")) et donne en sortie un ou des signaux électriques (chaque sortie est aussi dans un état "haut" ou à un état "bas"). Il existe deux catégories de circuits logiques :

- les circuits combinatoires (les états en sortie dépendent uniquement des états en entrée)
- les circuits séquentiels (les états en sortie dépendent des états en entrée ainsi que du temps et des états antérieurs)

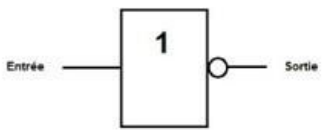
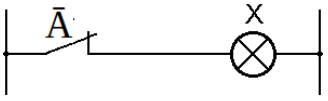
Dans la suite nous nous intéresserons principalement aux circuits combinatoires.

1.1. Les fonctions logiques de base https://www.youtube.com/watch?v=WUJ-6OqL7VU

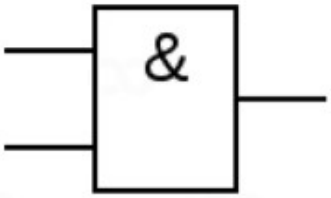
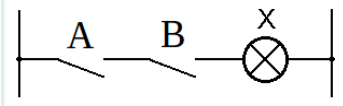
Voici les foncions logiques élémentaires à partir desquelles on peut créer des circuits plus complexes.
Les états binaires 0 et 1 d'une variable A peuvent être représentés par un interrupteur dans un circuit électrique.

Une porte **NON (NOT).**

La valeur retournée **X** par la fonction **NOT** est toujours la valeur inverse (complémentaire) de celle de la variable. Nous l'écrivons : **X = Ā** et nous lisons : **X** égale **A barre**.

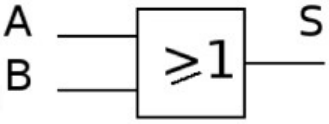
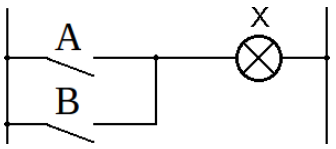
Symbole américain	Symbole européen	Table de vérité	Circuit électrique
		Entrée | Sortie A | Non A , Ā 1 | 0 |	

Une porte **ET (AND)**

Symbole américain	Symbole européen	Table de vérité			Circuit électrique
		Entrées		Sortie	
		A	B	A **ET** B	
		0	0		
		0	1		
		1	0		
		1	1		

Remarque :

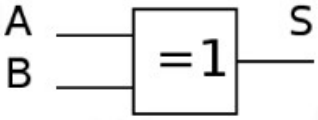
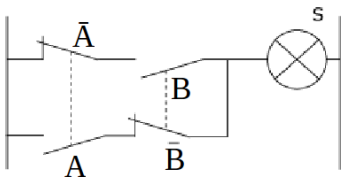
Une porte **OU** (**OR**)

Symbole américain	Symbole européen	Table de vérité	Circuit électrique
			

Entrées		Sortie
A	B	A **OU** B
0	0	
0	1	
1	0	
1	1	

Remarque :

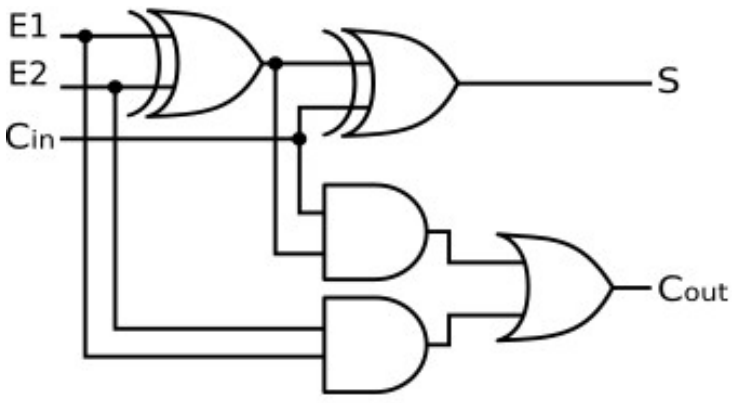
Une porte **OU EXCLUSIF** (**XOR**)

Symbole américain	Symbole européen	Table de vérité	Circuit électrique
			

Entrées		Sortie
A	B	A **XOR** B
0	0	
0	1	
1	0	
1	1	

1.2. Circuits plus complexes

En combinant les portes logiques, on obtient des circuits plus complexes. Par exemple en combinant 2 portes "OU EXCLUSIF", 2 portes "ET" et une porte "OU" on obtient un additionneur :



Entrées			Sorties	
E1	E2	C_{int}	S	C_{out}
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

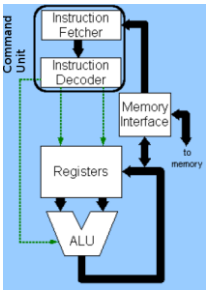
En combinant plusieurs fois le type de circuit décrit ci-dessus, on obtient des additionneurs capables d'additionner des nombres sur X bits.

Une chose est très importante à bien comprendre : à la base nous avons le transistor, une combinaison de transistors (sous forme de circuit intégré) permet d'obtenir des circuits logiques, la combinaison de circuits logiques permet d'obtenir des circuits plus complexes (exemple : l'additionneur), et ainsi de suite...

1. Les composants de base du cœur d'un ordinateur

1.1. Le microprocesseur CPU (Central Processing Unit)

1.1.1. Qu'est-ce que c'est ?



CPU

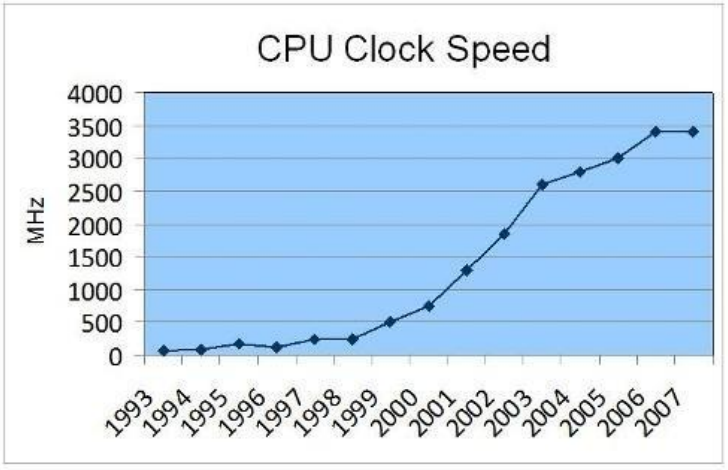
Le microprocesseur est le "cœur" d'un ordinateur : les instructions sont exécutées au niveau du CPU. Il est schématiquement constitué de 3 parties :

- Les registres permettent de mémoriser de l'information (donnée ou instruction) *au sein même* du CPU. Leur nombre et leur taille sont variables en fonction du type de microprocesseur. Dans la suite on nommera ces registres R1, R2, R3...
- L'unité arithmétique et logique (UAL ou ALU en anglais) est chargée de l'exécution de tous les calculs que peut réaliser le microprocesseur. Nous allons retrouver dans cette UAL des circuits comme l'additionneur (voir plus haut)
- L'unité de commande permet d'exécuter les instructions (les programmes). C'est en quelques sorte le chef d'orchestre du CPU

1.1.2. Evolution du CPU depuis son invention

Pendant des années, pour augmenter les performances des ordinateurs, les constructeurs augmentaient la fréquence d'horloge des microprocesseurs. A chaque top d'horloge, le CPU exécute une instruction : augmenter la fréquence d'horloge augmente le nombre d'instructions exécutées par seconde. Plus la fréquence d'horloge du CPU est élevée, plus ce CPU est capable d'exécuter un grand nombre d'instructions machines par seconde.

Comme on peut le voir, à partir de 2006 environ, la fréquence d'horloge a cessé d'augmenter, pourquoi ? À cause d'une contrainte physique : en effet plus on augmente la fréquence d'horloge d'un CPU, plus ce dernier chauffe. Il devenait difficile de refroidir le CPU, les constructeurs de microprocesseurs (principalement Intel et AMD) ont décidé d'arrêter la course à l'augmentation de la fréquence d'horloge, ils ont décidé d'adopter une nouvelle tactique.



Evolution de la fréquence d'horloge du CPU dans le temps

Puisqu'il n'est plus possible d'augmenter les performances en augmentant la fréquence d'horloge des CPU, et bien augmentons le nombre de cœurs présents sur un CPU ! C'est-à-dire que sur une même puce microprocesseur, il y a plusieurs CPU appelés cœurs. Cette technologie a été implémentée dans les ordinateurs grand public à partir de 2006. Aujourd'hui on trouve sur le marché des CPU possédant plus de 20 cœurs !

Cependant, pour une application qui n'aura pas été conçue pour fonctionner avec un microprocesseur multicœur, le gain de performance sera très faible, voire même nul. En effet, la conception d'applications capables de tirer profit d'un CPU multicœur demande la mise en place de certaines techniques de programmation parallèles. Il faut aussi avoir conscience que les différents cœurs d'un CPU doivent se "partager" l'accès à la mémoire vive : quand un cœur travaille sur une certaine zone de la RAM, cette même zone n'est pas accessible aux autres cœurs, ce qui, bien évidemment va brider les performances.

De plus, on trouve à l'intérieur des microprocesseurs (ou sur la carte mère) de la mémoire "ultrarapide" appelée mémoire cache. Le CPU peut stocker certaines données dans cette mémoire cache afin de pouvoir y accéder très rapidement dans le futur (une sorte d'anticipation pour gagner du temps). En effet, l'accès à la mémoire cache est beaucoup plus rapide que l'accès à la RAM. La mémoire cache ayant un coût assez important, la quantité présente au sein d'un CPU est limitée, les différents cœurs vont donc devoir se partager cette mémoire cache, ce qui peut aussi provoquer des ralentissements.

1.2. La mémoire vive RAM (Random Access Memory)

1.2.1. Qu'est-ce que c'est ?

La mémoire vive permet de stocker des données et des programmes. La mémoire ne gère pas les bits 1 par 1, mais 8 par 8, la mémoire gère donc des octets.

On peut se représenter la mémoire comme une série de cellules, chaque cellule étant capable de stocker 1 octet. Chacune de ces cellules possède une adresse. Les opérations sur la mémoire sont de 2 types : lecture / écriture. Une opération de lecture consiste à aller lire l'octet situé à une adresse mémoire donnée (exemple : 0x7fff9e9bf303) et une opération d'écriture consiste à écrire un octet donné à une adresse mémoire donnée. Cette notion d'adresse mémoire est fondamentale.

Toujours sur l'aspect technologique, 1 bit d'une cellule est l'association d'un transistor et d'un condensateur. Un condensateur est un composant électronique qui peut être soit chargé (on stocke alors un "1"), soit déchargé (on stocke alors un "0"). Un condensateur n'est pas capable de conserver sa charge pendant très longtemps, il doit donc être alimenté électriquement parlant afin de conserver cette charge. Voilà pourquoi la mémoire vive est une mémoire volatile : toutes les données présentes en mémoire sont perdues en cas de coupure de courant. Pour conserver les données une fois l'ordinateur éteint, il faut faire appel à d'autres types de mémoire (voir paragraphe suivant).

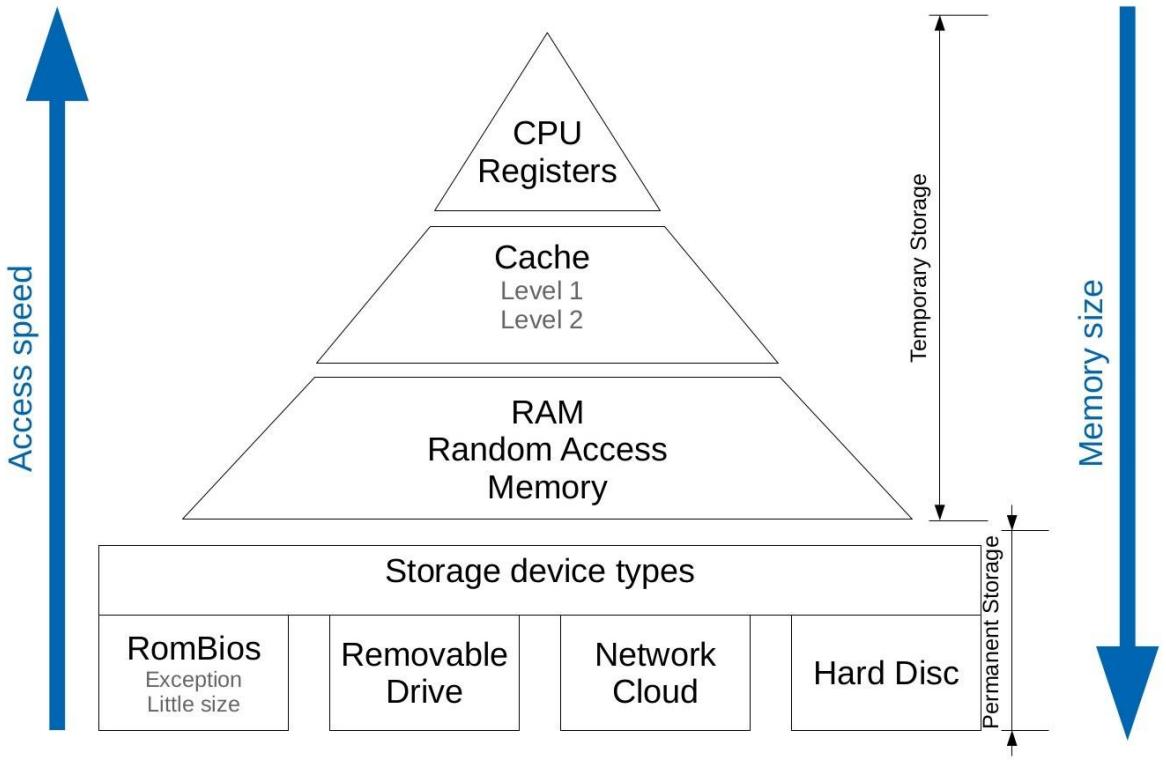
1.2.2. Les autres types de mémoire

Il existe un autre type de mémoire dans un ordinateur, la mémoire ROM (Read Only Memory) aussi appelée "mémoire morte". Comme son acronyme l'indique, cette mémoire est uniquement utilisable en lecture. Au contraire de la RAM, la ROM n'est pas volatile. Elle est notamment utilisée pour stocker les informations nécessaires au démarrage d'un ordinateur (BIOS).

Il existe également les disques durs internes ou externes, HDD (Hard Disk Drive) ou SSD (Solid State Drive) ainsi que les clés USB ou encore les DVD.

Voici un classement de ces mémoires par vitesse d'accès par le microprocesseur :

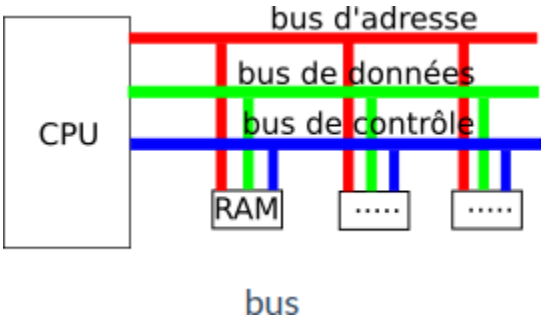
Mémoire	Temps d'accès	Débit	Capacité
registres	1 ns		Kilo octet
mémoire cache	2-3 ns		Méga octet
mémoire vive (RAM)	5-60 ns	De 1 à 20 GO/s	Giga octet
disque dur SSD	2-12 ms	27 Mo /s à 3 Go /s	Tera octet
disque dur HD	3-20 ms	de 10 à 320 MO/s	Tera octet
DVD	140 ms	de 2 à 22 MO/s	Giga octet



1.3. Les bus : moyen de communication entre le CPU et la RAM

Le système permettant cette circulation est appelé bus (physiquement, ce sont des fils reliant les différents composants). Il en existe 3 grands types :

- Le **bus d'adresses** 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).

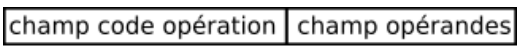


2. Assembleur : le langage du microprocesseur

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, en java, en C etc... En effet, comme tous les autres constituants d'un ordinateur, le CPU gère uniquement 2 états (toujours symbolisés par un "1" et un "0"), les instructions exécutées au niveau du CPU sont donc codées en binaire. L'ensemble des instructions exécutables directement par le microprocesseur constitue ce que l'on appelle le "langage machine" ou l'assembleur.

Une instruction machine est une chaîne binaire composée principalement de 2 parties :

- le champ "code opération" qui indique au processeur le type de traitement à réaliser. Par exemple le code "00100110" donne l'ordre au CPU d'effectuer une multiplication.
- le champ "opérandes" indique la nature des données sur lesquelles l'opération désignée par le "code opération" doit être effectuée.



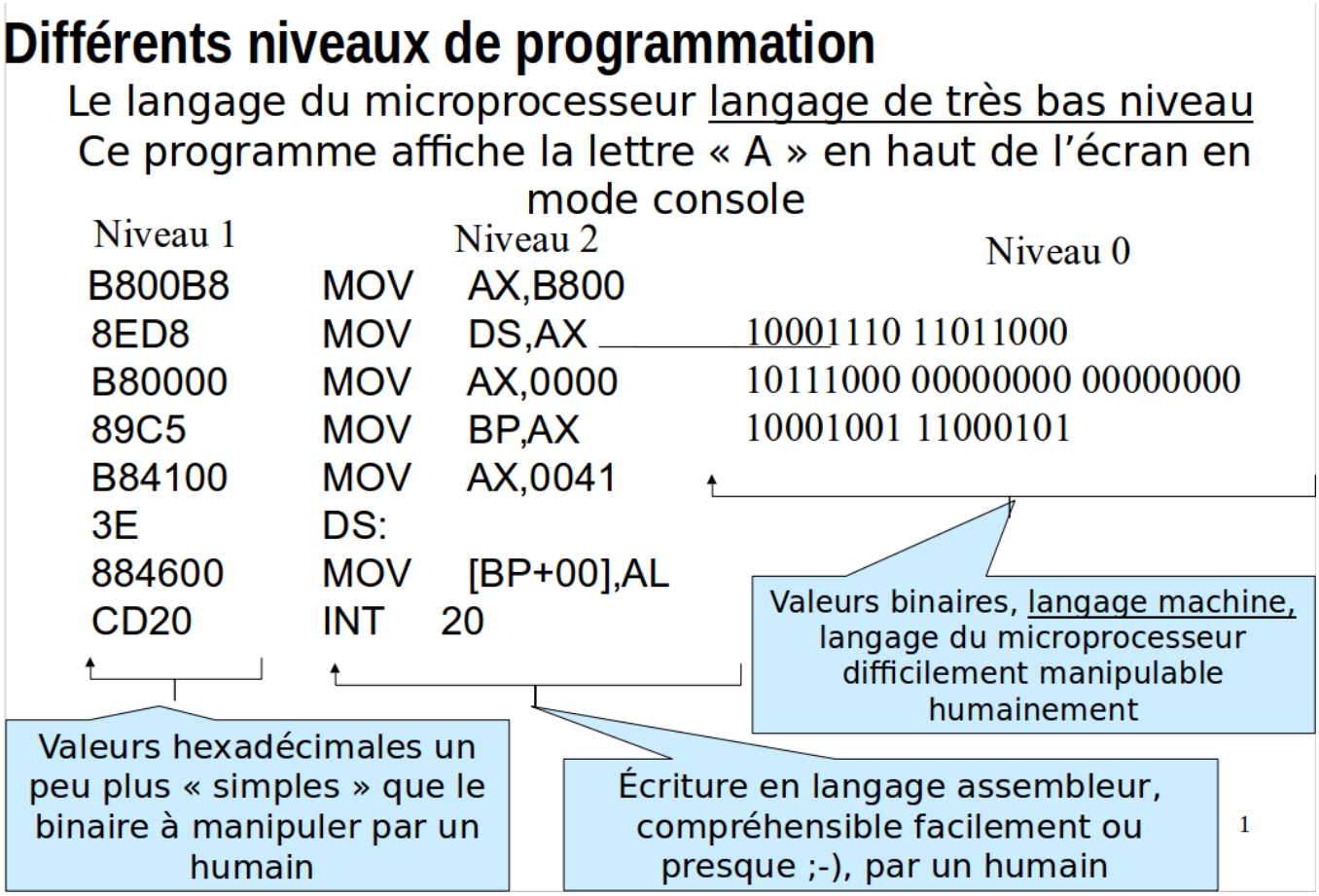
instruction machine

Évidemment le microprocesseur est incapable d'interpréter la phrase "additionne le nombre 125 et la valeur située dans le registre R2 , range le résultat dans le registre R1" tout cela doit être codé sous forme binaire.
Un programme en langage machine est donc une suite très très longue de "1" et de "0", humainement impossible à programmer : sur les dizaines de milliers de "1" et de "0" qui composent un programme en langage machine de taille modeste, une seule erreur, et votre programme ne fonctionne pas...imaginez la difficulté pour retrouver l'erreur !

Bref programmer en langage machine est extrêmement difficile, pour pallier cette difficulté, les informaticiens ont remplacé les codes binaires abscons par des symboles mnémoniques (plus facile à retenir qu'une suite de "1" et de "0"). Nous avons toujours des instructions machines du genre "additionne le nombre 125 et la valeur située dans le registre R2 , range le résultat dans le registre R1", mais au lieu d'écrire "11100010100000100001000001111101", nous pourrons écrire "**ADD R1,R2,#125**". (on retrouve les deux champs : le champ code opération puis celui des opérandes séparés par une virgule)
Dans les 2 cas, la signification est identique : "additionne le nombre 125 et la valeur située dans le registre R2 , range le résultat dans le registre R1".

Le processeur est uniquement capable d'interpréter le langage machine, **un programme appelé "assembleur"** assure donc le passage de "ADD R1,R2,#125" à "11100010100000100001000001111101". Par extension, on dit que l'on programme en assembleur quand on écrit des programmes avec ces symboles mnémoniques à la place de suites de "0" et de "1".

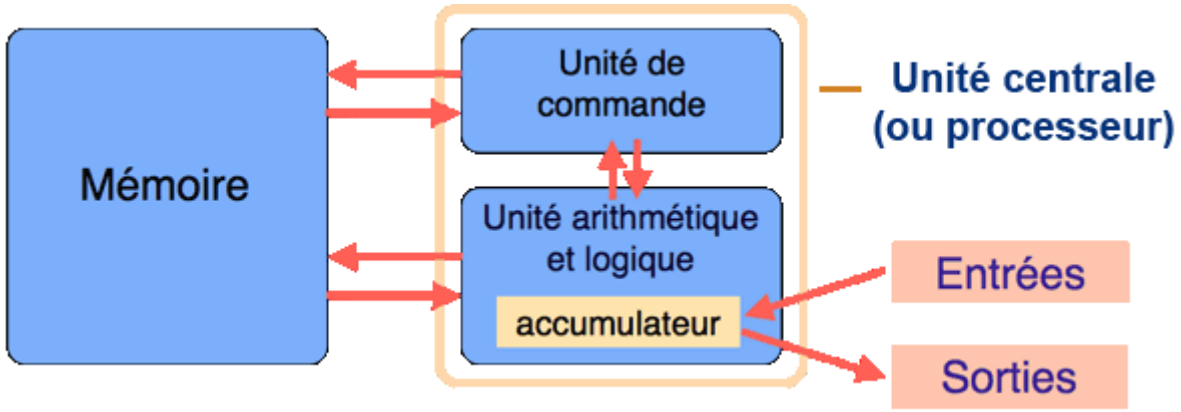
Aujourd'hui plus personne n'écrit de programme directement en binaire (ou hexadécimal), en revanche l'écriture de programme en assembleur est encore chose relativement courante. L'assembleur est donc le langage de plus bas niveau programmable par un humain (c'est à dire le plus proche du CPU).



3. Architecture de Von Neumann

John Von Neumann (mathématicien et physicien américano-hongrois 1903-1957) a proposé en 1945 un modèle d'architecture du fonctionnement d'un ordinateur modélisant les éléments vus dans les paragraphes précédents. Dans ce modèle, les données et les instructions à exécuter par le CPU se partagent la mémoire vive.

Voici le schéma du modèle de Von Neumann :

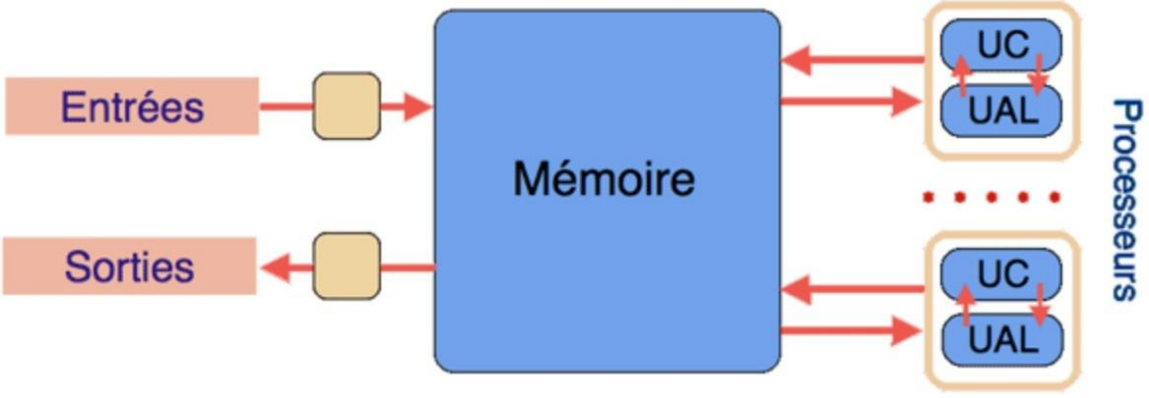


Modèle originel de Von Neumann
https://interstices.info/le-modele-darchitecture-de-von-neumann/

Sur ce schéma, nous avons :

- La mémoire qui correspond à la RAM
- L'unité arithmétique et logique qui correspond à l'UAL (l'accumulateur est un registre permettant de stocker les résultats intermédiaires lors d'un calcul)
- L'unité de commande qui gère l'exécution des instructions machines. À noter que cette unité de commande est aussi parfois appelée "unité de contrôle"
- Le système "entrée-sortie" qui permet au système CPU+RAM de communiquer avec "le monde extérieur" (clavier, souris, écran, carte graphique, disque dur…)

Encore aujourd'hui, tous les ordinateurs fonctionnent sur ce principe défini par Von Neumann. Mais ce dernier a évolué. Voici un schéma plus actuel :



À noter que John Von Neumann était un véritable génie "touche à tout" puisqu'il a laissé son nom dans l'histoire de la mécanique quantique, dans l'histoire de la théorie des ensembles… et comme nous venons de le voir, dans l'histoire de l'informatique. Il a aussi participé à l'élaboration de la bombe atomique américaine lors de la 2e guerre mondiale (projet Manhattan).

Assembleur (microprocesseur ARM société « *Advanced RISC Machines* »)

Pour simplifier, nous allons considérer un processeur qui contient 14 registres : R0,R1,R2,...,R12 et un registre particulier appelé PC (Program Counter) aussi appelé IP (Instruction Pointer) et parfois nommé compteur ordinal. Ce registre contient l'adresse mémoire de la prochaine instruction à exécuter par le CPU.

Voici une partie des opérations (**le « jeu d'instructions » en anglais « instructions set »**) qu'un microprocesseur d'architecture ARM permet d' effectuer :

- **LDR** R1,78 : Place la valeur stockée à l'adresse mémoire 78 dans le registre R1 (par souci de simplification, nous continuons à utiliser des adresses mémoire codées en base 10) (LD pour load)
- **STR** R3,125 : Place la valeur stockée dans le registre R3 en mémoire vive à l'adresse 125 (ST pout stock)
- **ADD** R1,R0,#128 : Additionne le nombre 128 (une valeur immédiate est identifiée grâce au symbole #) et la valeur stockée dans le registre R0, place le résultat dans le registre R1
- **ADD** R0,R1,R2 :Additionne la valeur stockée dans le registre R1 et la valeur stockée dans le registre R2, place le résultat dans le registre R0
- **SUB** R0,R1,R2 : Soustrait la valeur stockée dans le registre R2 de la valeur stockée dans le registre R1, place le résultat dans le registre R0
- **MOV** R1, #23 : Place le nombre 23 dans le registre R1
- **MOV** R0, R3 : Place la valeur stockée dans le registre R3 dans le registre R0
- **B** 45 : Nous avons une structure de rupture de séquence, la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 45 (B pour break)
- **CMP** R0, #23 :Compare la valeur stockée dans le registre R0 et le nombre 23. Cette instruction CMP doit précéder une instruction de branchement conditionnel BEQ, BNE, BGT, BLT (voir ci-dessous)

CMP R0, #23 **BEQ** 78 La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est **égale** à 23	**CMP** R0, #23 **BNE** 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 23
CMP R0, #23 **BGT** 78 La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est **plus grand** que 23	**CMP** R0, #23 **BLT** 78 La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est **plus petit** que 23

Rem : souvent, au lieu d'utiliser l'adresse mémoire de la prochaine instruction, on utilise des "labels":

```
ex : CMP R0, #23
     BGT lab1
     ...
   lab1:
     ...
```

- **CMP** R0, R1 : Compare la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1. Cette instruction CMP doit précéder une instruction de branchement conditionnel comme précédemment.
- **HALT** : Arrête l'exécution du programme

Le jeu d'instructions d'un microprocesseur ARM (le « Cortex-M3 ») : https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/instruction-set-summary