



PARTIE 1 : Modèle d'architecture séquentielle (Von Neumann)	2
1. Présentation	2
1.1. Le transistor	2
2. L'architecture de Neumann	2
2.1. Les structures principales.....	2
2.2. Les sous structures.....	3
3. Le rôle de l'horloge	3
3.1. Cadence du processeur	3
3.2. Cycle d'instruction	3
4. Le rôle de la mémoire	4
4.1. Différents types de mémoire	4
4.2. Les registres	4
4.3. Mémoire centrale et mémoire cache.....	4
5. Le langage assembleur.....	4
5.1. Présentation.....	4
5.2. TP assembleur.....	5

PARTIE 1 : Modèle d'architecture séquentielle (Von Neumann)

1. Présentation

À la base de la plupart des composants d'un ordinateur, on retrouve le transistor. Ce composant électronique a été inventé fin 1947 par les Américains John Bardeen, William Shockley et Walter Brattain.

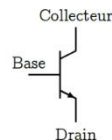
1.1. Le transistor

L'invention du transistor a été un immense progrès, mais les premiers ordinateurs sont antérieurs à cette invention. En effet, ces premiers ordinateurs, par exemple le Colossus qui date de 1943, étaient conçus à base de tubes électroniques (on parle aussi de tubes à vide) qui, bien que beaucoup plus gros et beaucoup moins fiable que les transistors fonctionnent sur le même principe que ce dernier.



Les 3 bornes sont nommées :

- B pour la base (ou grille)
- C pour le collecteur (ou source)
- E pour l'émetteur (ou drain)



Le transistor



Un tube électronique

Il existe une grande variété de transistors aux fonctionnements différents. Cependant, de manière générale, un transistor est un dispositif semi-conducteur.

Il se comporte comme un interrupteur entre les bornes C et E commandé par la borne B

Autre aspect historique qu'il est important de préciser : on ne trouve plus, depuis quelque temps déjà, de transistors en tant que composant électronique discret (comme le transistor de la photo ci-dessus).

Dans un ordinateur, les transistors sont regroupés au sein de ce que l'on appelle des circuits intégrés.

Dans un circuit intégré, les transistors sont gravés sur des plaques de silicium, les connexions entre les millions de transistors qui composent un circuit intégré sont, elles aussi, gravées directement dans le silicium :

<https://www.youtube.com/watch?v=NFr-WyytNfo>

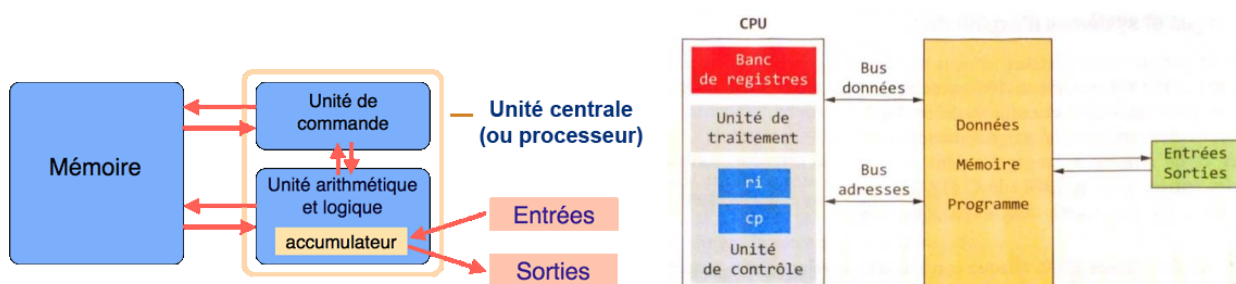
2. L'architecture de Neumann

L'architecture de von Neumann a été conçue en juin 1945 par le mathématicien John von Neumann dans le cadre du projet EDVAC (Electronic Discrete Variable Automatic Computer) : il s'agit d'un des tout premiers ordinateurs. Ce modèle est toujours d'actualité : il régit toujours l'architecture des ordinateurs malgré quelques évolutions.

2.1. Les structures principales

Dans l'architecture de von Neumann, un ordinateur est constitué de quatre parties distinctes :

- le CPU (Central Processing Unit ou unité centrale de traitement), plus communément appelé le processeur
- la mémoire où sont stockées les données et les programmes
- des bus qui sont en fait des fils qui conduisent des impulsions électriques et qui relient les différents composants
- des entrées-sorties (E/S ou I/O) pour échanger avec l'extérieur



2.2. Les sous structures

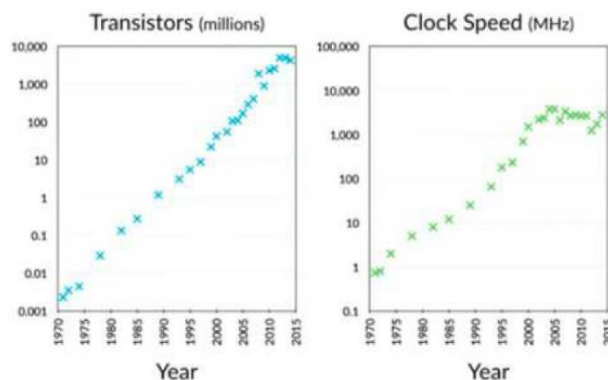
- Les échanges entre la mémoire et les registres du CPU se font via des bus selon une chronologie organisée par l'horloge
- Un programme est enregistré dans la mémoire
- L'adresse (nombre entier) de l'instruction en cours de traitement est stockée dans une mémoire interne au processeur nommée « registre compteur de programme » (cp ou pc).
- La valeur de cette instruction (un entier) est stockée dans une autre mémoire interne : le « registre d'instruction » (ri)
- Le CPU dispose aussi de mémoire interne dans le « banc de registre » où sont placées les données du programme avant utilisation
- l'UAL (unité arithmétique et logique) effectue les opérations arithmétiques et logiques sur les données et les adresses, en interprétant les impulsions électriques sortant de ses circuits combinatoires fabriqués à l'aide portes NAND.

3. Le rôle de l'horloge

3.1. Cadence du processeur

Le CPU dispose d'une horloge qui cadence l'accomplissement des instructions. L'unité est appelé cycle. Lorsqu'on parle d'un processeur cadencé à 3GHz, cela signifie qu'il y a 3 milliards de cycles d'horloge par seconde.

Jusqu'en 2004 environ, la fréquence des processeurs a augmenté. Depuis elle stagne. En effet, au-delà, la chaleur produite devient trop importante et pourrait perturber la lecture des tensions lues aux bornes des circuits de l'UAL, voire détériorer physiquement ses circuits.



3.2. Cycle d'instruction

Dans un processeur, ce que l'on appelle cycle est l'exécution de chacun des cinq instructions suivantes :

- lire l'instruction (LI)
- décoder l'instruction (DI)
- exécuter l'opération dans l'UAL (EX)
- accéder à la mémoire en lecture ou en écriture (M)
- écrire le résultat dans le registre.

Pour gagner du temps, le processeur n'exécute pas les instructions de manière séquentielle mais exécute simultanément plusieurs instructions qui sont à des étapes différentes de leur traitement. C'est le principe du « **pipeline d'instruction** »

1	2	3	4	5	6	7	8	9
LI	DI	EX	M	ER				
	LI	DI	EX	M	ER			
		LI	DI	EX	M	ER		
			LI	DI	EX	M	ER	
				LI	DI	EX	M	ER

4. Le rôle de la mémoire

4.1. Différents types de mémoire

Il existe différents types de mémoires :

- la **mémoire vive** : quand le contenu est perdu lorsque le courant s'arrête : il s'agit des registres, des mémoires cache, de la mémoire centrale.
- les **mémoires persistantes** : les SSD (Solid State Device), les disques magnétiques
- la **ROM** (Read Only Memory) qui ne fonctionne qu'en lecture seule

4.2. Les registres

Un **registre** est un emplacement mémoire interne au processeur pour stocker des opérandes et des résultats intermédiaires lors des opérations effectuées par l'UAL.

4.3. Mémoire centrale et mémoire cache

La **mémoire centrale** est une mémoire vive qui contient les programmes en cours et les données qui sont manipulées. Elle est organisée en cellules qui contiennent chacune une **donnée** ou une **instruction** repérée par un entier : l'**adresse mémoire**.

Le temps d'accès à chaque cellule est le même. On parle alors de **mémoire à accès aléatoire RAM** (Random Access Memory).

Pour pouvoir adapter la très grande vitesse du processeur à celle plus faible de la mémoire centrale, on place entre eux une mémoire plus rapide, la **mémoire cache**. Elle contient les instructions et données en cours d'utilisation car la plupart du temps, les données qui viennent d'être utilisées ont une probabilité plus grande d'être réutilisées.

5. Le langage assembleur

5.1. Présentation

Les programmes peuvent être écrits dans différents types de langages :

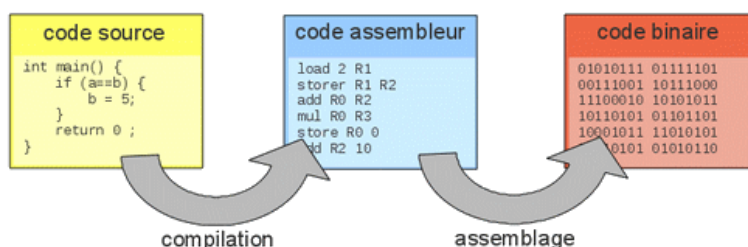
- langage de **haut niveau** : plus proche du langage humain, par exemple Python. Il est éloigné du langage machine (de bas niveau). Il dépend peu du processeur et du système d'exploitation utilisé.
- langage de **bas niveau** plus proche de la machine.

On parle également de niveau d'abstraction d'un langage. Plus celui-ci est proche de notre langage naturel et plus son niveau d'abstraction est élevé. Plus le langage est proche de la machine (binaire) plus celui-ci est de bas niveau.

Du langage haut niveau vers le langage machine

Lorsque vous exécutez un programme Python, celui-ci est traduit sous forme d'instructions en langage bas niveau de type "assembleur".

En fait, sans rentrer dans le détail, il s'agit en fait de pseudo-code d'assemblage qui au moment de l'exécution est traduit en langage machine. Tout langage de programmation, pour être exécuté par une machine doit être à un moment où à un autre traduit en langage binaire.



Compilateur ou interpréteur

Il existe plusieurs manières de procéder :

La première consiste à traduire le programme dans son ensemble une fois pour toute et générer un fichier avec le code binaire prêt à être exécuté. Il s'agit de la **méthode dite de compilation réalisée par un compilateur**. Le langage C est un exemple de **langage compilé**.

La deuxième méthode consiste à traduire les instructions en langage binaire au fur et à mesure de la lecture du programme. Il s'agit de la **méthode dite d'interprétation réalisée par un interpréteur**. Python est un **langage interprété**.

5.2. TP assembleur

Le CPU est incapable d'exécuter directement des programmes écrits, par exemple, en Python. 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".

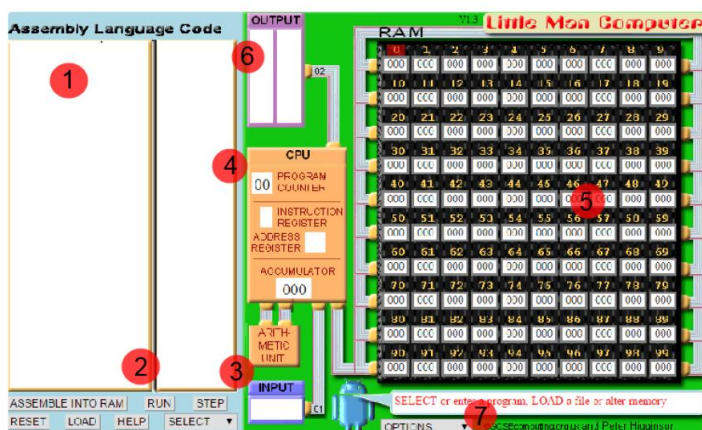
Nous allons utiliser un simulateur nommé « Little Man Computer » <https://peterhigginson.co.uk/lmc/>

Il représente le fonctionnement interne d'un microprocesseur dans un langage de bas niveau : l'assembleur. Il nous permettra d'avoir une première approche du fonctionnement du modèle de Von Neumann Le LMCest généralement utilisé pour enseigner aux élèves, car il modélise un ordinateur simple avec une architecture de Von Neumann et qui possède toutes les fonctionnalités élémentaires d'un ordinateur moderne. Il peut être programmé en code machine (bien que sous forme décimale plutôt que binaire) ou en assembleur.

Dans ce simulateur, on retrouve :

- Program Counter (PC) : ce registre contient la ligne à laquelle nous sommes.
- Accumulator (ACC) : C'est l'endroit où on ira chercher les valeurs à additionner, soustraire etc.
- Instruction Register (IR) : le type d'instruction.
- Address Register (AR) : L'adresse du registre où travailler

La mémoire a 100 cases numérotées de 00 à 99. Elles contiennent les valeurs en décimal avec 3 chiffres.



Vous distinguerez :

- ❶ La fenêtre pour taper le code.
- ❷ Les deux boutons pour charger le code en mémoire puis exécuter.
- ❸ La fenêtre pour une entrer une valeur (éventuellement)
- ❹ Un indicateur qui montre la progression du code étape par étape.
- ❺ Les emplacements mémoire où sont stockées les instructions et les données, comme spécifié dans l'architecture de Von Neumann. (100 cellules, de 00 à 99).
- ❻ La fenêtre pour la / les sortie (s) lors de l'exécution du code.
- ❼ Les options pour contrôler le déroulement de l'exécution lent à rapide, etc.

MÉTHODE D'UTILISATION :

1. Taper le code
2. SUBMIT
3. ASSEMBLE INTO RAM
4. RUN ou STEP
5. RESET

INSTRUCTIONS

Instruction	Instruction	Code	Remarque
INPUT	INP	901	copie INPUT → ACC
ADD	ADD	1XX	ACC + @XX → ACC
SUBTRACT	SUB	2XX	ACC - @XX → ACC
STORE	STA	3XX	ACC → @XX
LOAD	LDA	5XX	@XX → ACC
BRANCH ALWAYS	BRA	6XX	GOTO XX
BRANCH IF ZERO	BRZ	7XX	IF ACC == 0 : GOTO XX
BRANCH IF ZERO OR POSITIVE	BRP	8XX	IF ACC >= 0 : GOTO XX
OUTPUT	OUT	902	ACC → OUTPUT
END	HLT	000	stoppe le programme

➤ REMARQUE 1 :

- @XX correspond à une adresse mémoire
- XX correspond à un numéro de ligne

✓ EXEMPLE 1 :

a) Tapez dans la fenêtre ❶ les instructions suivantes :

```
INP
STA 20
OUT
HLT
```

b) Choisir la vitesse lente, cliquer sur Submit.

c) Cliquer sur Run

d) Dans la fenêtre de saisie input entrer par exemple la valeur 15, cliquer sur Entrée

e) Observer le mécanisme d'utilisation de la mémoire

PC	Instr	Code	Remarque	IR	AR	ACC	@20
00	INP	901	Entrer valeur dans INPUT (ex : 15)	9	01	15	000
01	STA 20	320	Stocke la valeur de ACC à l'adresse @20	3	20	15	015
02	OUT	902	Affiche le contenu de ACC	9	02	15	015
03	HLT	000	Stoppe le programme	0	00	15	015

✓ EXEMPLE 1 bis :

- f) Saisir les instructions suivantes et observer le mécanisme d'utilisation de la mémoire (Dans la fenêtre de saisie input entrer par exemple la valeur 15)


PC	Instruction	Code	Remarque	IR	AR	ACC	@3
00	LDA A	503	Charge dans ACC le contenu de la variable (associée à le mémoire 03)	5	03	015	015
01	OUT	902	Affiche le contenu de ACC	9	02	015	015
02	HLT	000	Stoppe le programme	0	00	015	015
03	A DAT 015	Pas de code	Variable A (associée à la case mémoire 03 dont le contenu vaut 015)				015


🔪 EXERCICE 1 : Additionner deux valeurs : 15 et 22


PC	Instr	Code	IR	AR	ACC	INPUT	OUTPUT	@10	@11
00	INP	901	9	01	015	15		000	000
01	STA 10	310	3	10	015			015	000
02	INP	901	9	01	022	22		015	000
03	STA 11	311	3	11	022			015	022
04	LDA 10	510	5	10	015			015	022
05	ADD 11	111	1	11	037			015	022
06	OUT	902	9	02	037		37	015	022
07	HLT	000	0	00					

🔪 EXERCICE 2 : Soustraire deux valeurs. 45 – 24

PC	Instruction	Code	IR	AR	ACC	INPUT	OUTPUT	@10	@11
00									
01									
02									
03									
04									
05									
06									
07									

 EXERCICE 3 : Entrer un nombre, le multiplier par deux, 2 fois et afficher le résultat. $(7 \times 2) \times 2 = 28$

 EXERCICE 4 : Entrer deux valeurs. Afficher la plus grande des deux.

 EXERCICE 5 : Entrer une valeur. Si cette valeur est strictement supérieure à 50, afficher 1, sinon afficher 0.

```
a_97 = 50
a_98 = 0
a_99 = 1
a = int(input())
if a - a_97 >= 0:
    print(a_99)
else:
    print(a_98)
```

On va commencer par entrer dans @98 et @99 les valeurs 0 et 1 et dans @97 la valeur 50.

PC	Instruction	Code	IR	AR	ACC	INPUT	OUTPUT	@96	@97	@98	@99
00											
01											
02											
03											
04											
05											
06											
07											
08											
09											
10											
11											
12											
13											
14											
15											
16											

--	--

OU BIEN

--	--