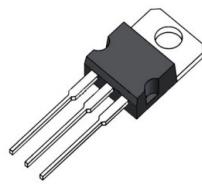
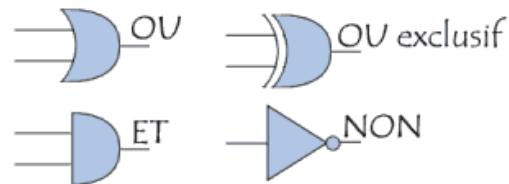


L'ordinateur : Architecture Von Neumann

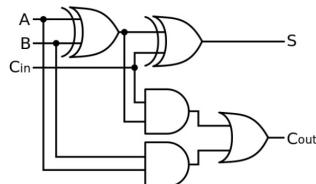
A la base il y a le transistor, à l'échelle du nanomètre



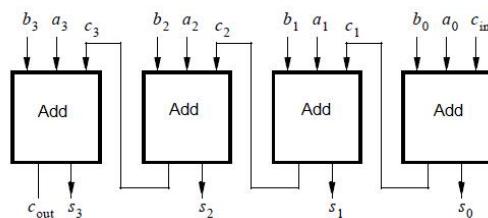
Une combinaison de transistors (sous forme de circuit intégré) permet d'obtenir des circuits logiques



La combinaison de circuits logiques permet d'obtenir des circuits plus complexes (exemple : l'additionneur)

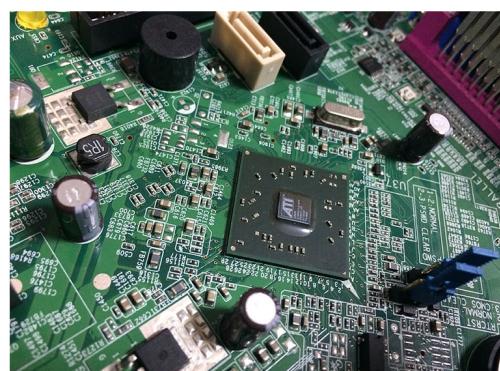


Et ainsi de suite...(exemple : un additionneur 4 bits composé de 4 additionneur 1 bit)



Au sommet de cet édifice, se trouvent:

La mémoire vive (RAM, Random Access Memory) et le microprocesseur (CPU, Central Processing Unit).



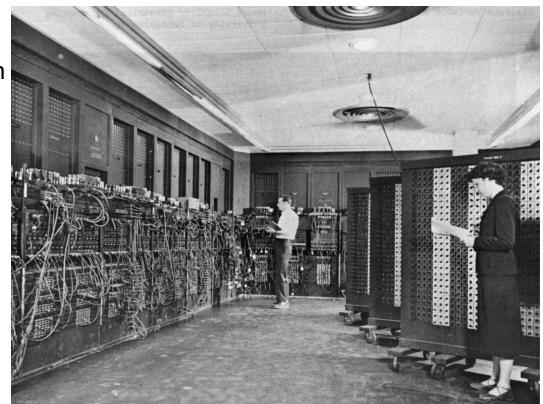
1. Von Neumann

Architecture

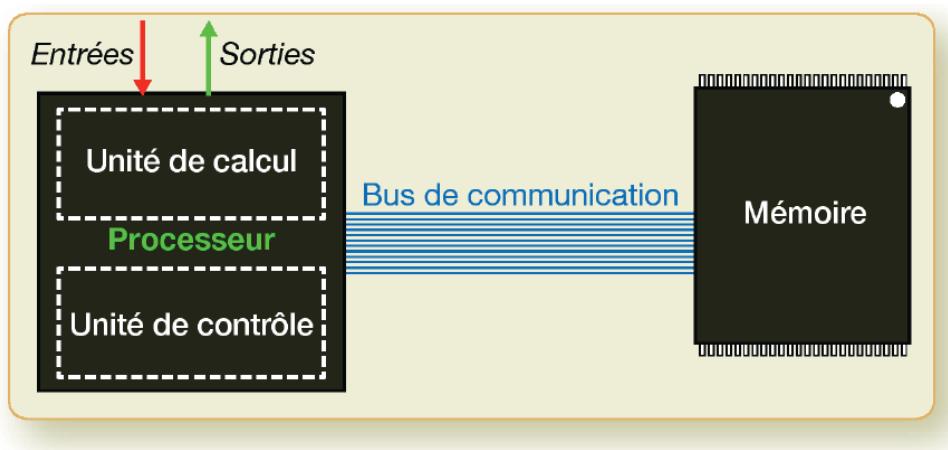


Dans les années 1940, à l'Université de Pennsylvanie, John von Neumann (1903-1957) a conçu, avec Presper Eckert et John Mauchly, deux des premiers ordinateurs : l'ENIAC (ci-contre à droite), puis l'EDVAC.

Ces ordinateurs étaient organisés selon l'architecture de Von Neumann, encore utilisée dans la quasi-totalité des ordinateurs actuels : La séparation du processeur et de la mémoire, reliés par des bus de communication.



La principale innovation à l'époque est que la mémoire contient les données sur lesquelles on calcule mais aussi le programme (auparavant écrit sur des cartes perforées ou sur un ruban papier) qui décrit le calcul effectué, donné sous la forme d'une séquence d'instructions.



- La **mémoire** est composée de plusieurs milliards de circuits mémoires un bit. Ces cases mémoire sont organisées en groupes de multiples de 8 bits (généralement 8, 16, 32 ou 64 bits). Le nombre de ces cases définit la taille de la mémoire de l'ordinateur. Comme il faut distinguer ces cases les unes des autres, on donne à chacune un numéro : son adresse.

```
In [21]: #python affiche l'adresse de la case mémoire dans laquelle est stockée la valeur 5 (ici au format hexadécimal)
x = 2
print (hex(id(x)))
```

0x5a9ad8c0

- **Le microprocesseur** est le "coeur" d'un ordinateur, il exécute les instructions. Il est schématiquement constitué de 3 parties :
 - *Les registres* qui peuvent 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.
 - *L'unité arithmétique et logique (UAL)* , chargée de l'exécution de tous les calculs que peut réaliser le microprocesseur(grâce à des circuits logiques comme l'additionneur par exemple).
 - *L'unité de contrôle* permet d'exécuter les instructions (les programmes)
- **Les bus**(des ensembles de fils) font circuler Les données entre les différentes parties d'un ordinateur, notamment entre la mémoire vive et le CPU. Il en existe, sans entrer dans les détails, 3 grands types :
 - *Le bus d'adresses* qui fait circuler des adresses (par exemple l'adresse d'une donnée à aller chercher en mémoire)
 - *Le bus de données* qui fait 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. Langage machine

Un simulateur CPU : <http://www.peterhigginson.co.uk/AQA/>
<http://www.peterhigginson.co.uk/AQA/>

Voici la capture d'écran d'un simulateur qui reproduit le fonctionnement du processeur et de la mémoire.

Exercice 1:

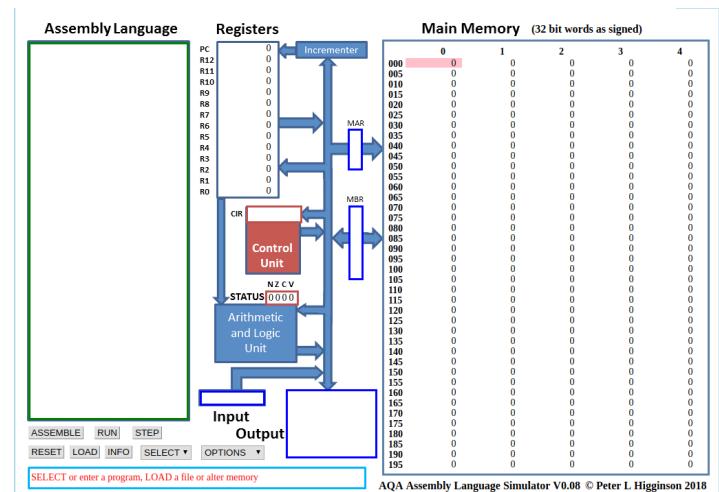
1. Quelle est la taille de la mémoire vive ?
2. De combien de registres le processeur dispose-t-il ?
3. Où se trouve l'UAL ?
4. Où se trouve l'Unité de Commande ?
5. Que représentent les flèches bleues ?

Réponses :

- 1.
- 2.

- 1.
- 2.

- 5.



Assembleur

- 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.
- La compilation d'un programme, c'est la transformation du programme en instructions machines. Un programme existe donc toujours sous deux formes : le code source écrit en Java, C, Python... Et le code compilé écrit en langage machine adapté au processeur utilisé.
- Les instructions exécutées à ce niveau sont ainsi codées en binaire, ce qui est rapidement difficile à lire et à écrire.
- Les premiers programmeurs disposaient de quelques instructions pour représenter du code machine.
- A gauche sur la capture d'écran ci-dessus se trouve une zone où l'on peut exécuter des instructions machines écrites en langage assembleur(Assembly Language). Ces instructions sont propres à chaque machine !
- Aujourd'hui plus personne n'écrit de programme directement en langage machine, en revanche l'écriture de programme en assembleur est encore chose relativement courante.

Instruction machine

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

Exemple adapté au simulateur : MOV R0, #42

- Le champ "code opération" qui indique au processeur le type de traitement à réaliser(ici c'est MOV)
- Le champ "opérandes" indique la nature des données sur lesquelles l'opération désignée par le "code opération" doit être effectuée (ici c'est R0 qui désigne le registre du même nom et #42 qui désigne l'entier 42).

Exercice 2 :

1. Se débrouiller pour exécuter cette instruction dans l'assembleur. Que se passe-t-il ?
2. Dans **OPTIONS** , choisir **binary**. Ou retrouve -t-on la valeur 42 ?
3. Que fait finalement cette instruction ?

Réponses :

- 1.
- 1.
- 3.

Exercice 3:

Utiliser **RESET** pour réinitialiser le simulateur, puis saisir une instruction qui permet de stocker la valeur 127 dans le registre R1 .

Réponse :

Types d'instructions

- Les instructions de transfert de données qui permettent de transférer une donnée d'un registre du CPU vers la mémoire vive et vice versa.
 - Exemples :
 - LDR R1,78 : LOAD, charge la valeur contenue à l'adresse 78 de la mémoire dans le registre R1
 - STR R0,23 : STORE, stocke la valeur contenue dans le registre R0 dans l'emplacement 23 de la mémoire vive
 - MOV R5,#74 : MOVE, place l'entier 74 dans le registre R5 (le caractère # indique une valeur et non une adresse)
- Les instructions arithmétiques (addition, soustraction, multiplication, opérateurs booléens,...).
 - Exemples :
 - ADD R1,R0,#128 : Additionne le contenu du registre R0 et la valeur 128 et place le résultat dans le registre R1
 - ADD R0,R1,R2 : Additionne les contenu des registres R1 et R2 et place le résultat dans le registre R0
 - SUB R0,R1,R2 : Soustrait le contenu du registre R2 au contenu de R1 et place le résultat dans le registre R0
- Les instructions de rupture de séquence : 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 immédiatement supérieure (il lit la 1 puis la 2, etc...). Certaines instructions permettent d'interrompre l'ordre initial (on parle de saut ou de branchements) sous certaines conditions en passant à une instruction située une adresse mémoire donnée.
 - Exemples :
 - B 45 : BRANCH, la prochaine instruction se trouve à l'adresse 45.
 - COMP R0, #23 : COMPARE, compare le contenu du registre R0 avec la valeur 23. Cette instruction CMP est obligatoirement suivie d'une des instructions de branchements conditionnel ci-dessous:
 - BEQ 78 : BRANCH EQUAL, si la valeur comparée est égale à 23, la prochaine instruction à l'adresse 78
 - BNE 41 : BRANCH NOT EQUAL, si la valeur comparée est différente de 23, la prochaine instruction à l'adresse 41
 - BGT 90 : BRANCH GREATER THAN, si la valeur comparée est supérieure à 23, la prochaine instruction à l'adresse 90
 - BLT 125 : BRANCH LESS THAN, si la valeur comparée est inférieure à 23, la prochaine instruction à l'adresse 125
 - HALT : Arrête l'exécution du programme

Exercice 4 :

Partie A : Expliquer ce que font chacune des instructions.

- ADD R0, R1, #42 :
- LDR R5, #18 :
- CMP R4, #18 :
- BGT 77 :
- STR R0, #15 :
- B 100 :

Partie B : Ecrire les instructions correspondantes aux phrases suivantes

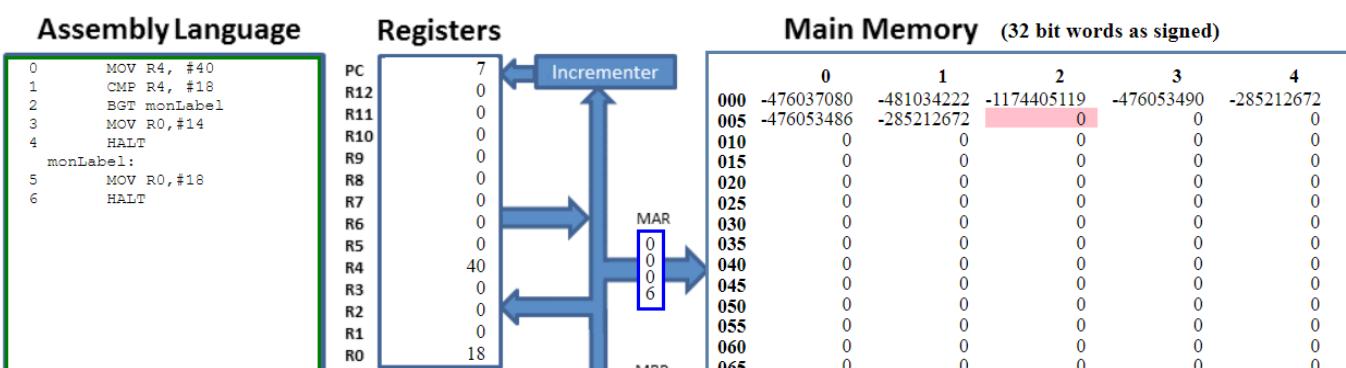
- Additionne la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1, le résultat est stocké dans le registre R5 :
- Place la valeur stockée à l'adresse mémoire 878 dans le registre R0 :
- Place le contenu du registre R0 en mémoire vive à l'adresse 124 :
- La prochaine instruction à exécuter se situe en mémoire vive à l'adresse 478 :
- Si la valeur stockée dans le registre R0 est égale 42 alors la prochaine instruction à exécuter se situe à l'adresse mémoire 85 :

Labels

Les instructions assembleur B, BEQ, BNE, BGT et BLT n'utilisent pas directement l'adresse mémoire de la prochaine instruction à exécuter, mais des "labels"(des étiquettes). Un label correspond à une adresse en mémoire vive (c'est l'assembleur qui fera la traduction "label"→"adresse mémoire"). L'utilisation d'un label évite donc d'avoir à manipuler des adresses mémoires en binaire ou en hexadécimal.

Exercice 5:

Ci-dessous le résultat d'une exécution d'instructions machines.



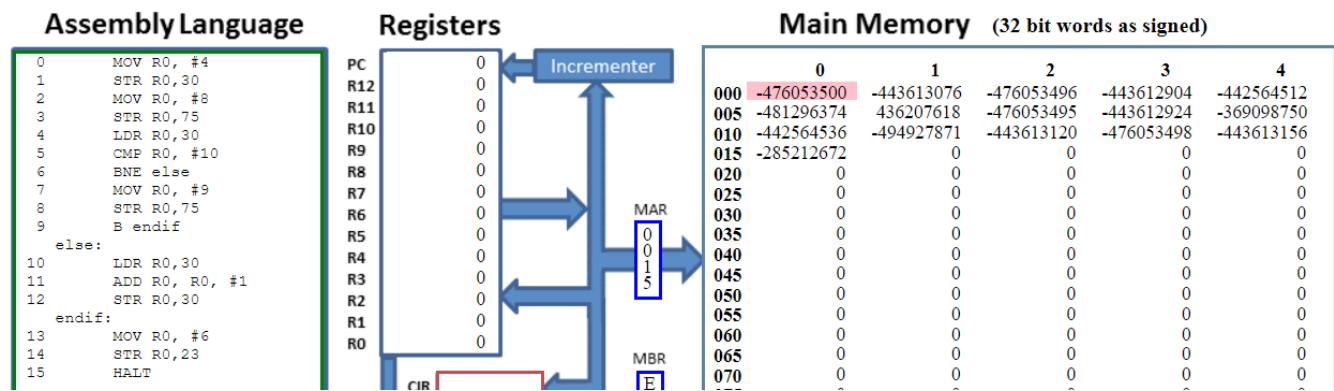
1. Quel est le nom du label utilisé ? :

2. Dans quel cas la partie monLabel: est-elle lue ? :

1. Pourquoi l'instruction HALT de la ligne 4 est-elle indispensable ? :

1. Réinitialiser la mémoire et modifier le programme pour qu'à la fin de l'instruction, le registre R0 contienne la valeur 14 :

Exercice :



1. Saisir le code assembleur dans le simulateur puis L'exécuter pas à pas avec la touche STEP.
2. Décrire ce que fait ce programme
3. A quoi servent les adresses 25,30 et 75 ?
4. Traduire ce programme en langage python.

Réponses :

1.

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

1.

In [1]: #4.

Exercice 6 :

Voici un code python. Le traduire en langage machine !

In []: x=0
while x<=3:
 x=x+1

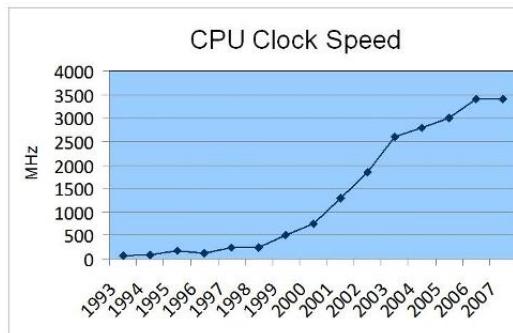
Réponse :

- • • • •

3. Multiprocesseurs

La course au Ghz

- Pour pouvoir exécuter un grand nombre d'instructions par seconde, celles-ci doivent être synchronisées, c'est le rôle de l'horloge.
- Pendant des années, pour augmenter les performances des ordinateurs, les constructeurs augmentaient la fréquence d'horloge des microprocesseurs : Plus la fréquence d'horloge du CPU est élevée, plus ce CPU est capable d'exécuter un grand nombre d'instructions machines par seconde

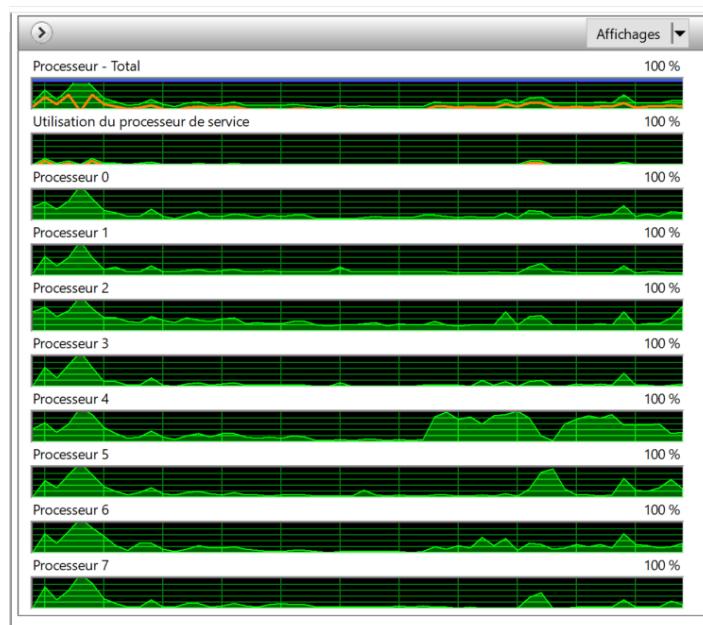


- Comme on peut le voir sur le graphique ci-contre, à partir de 2006 environ, la fréquence d'horloge a cessé d'augmenter, pourquoi ? À cause d'une contrainte physique : Il devenait difficile de refroidir le CPU, les constructeurs de microprocesseurs ont décidé d'adopter une nouvelle tactique.

Multicoeurs

- Dans un processeur, un cœur physique est un ensemble de circuits capables d'exécuter des programmes de façon autonome. Outre les fonctionnalités nécessaires à l'exécution d'un programme sont présents aussi des registres et une unité de calcul.
- Les ordinateurs, tablettes et smartphones récents disposent désormais de processeurs comportant 2, 4, 8, 16 coeurs, voire plus, gravés sur une même puce. Ceci multiplie d'autant le nombres d'instructions exécutables.

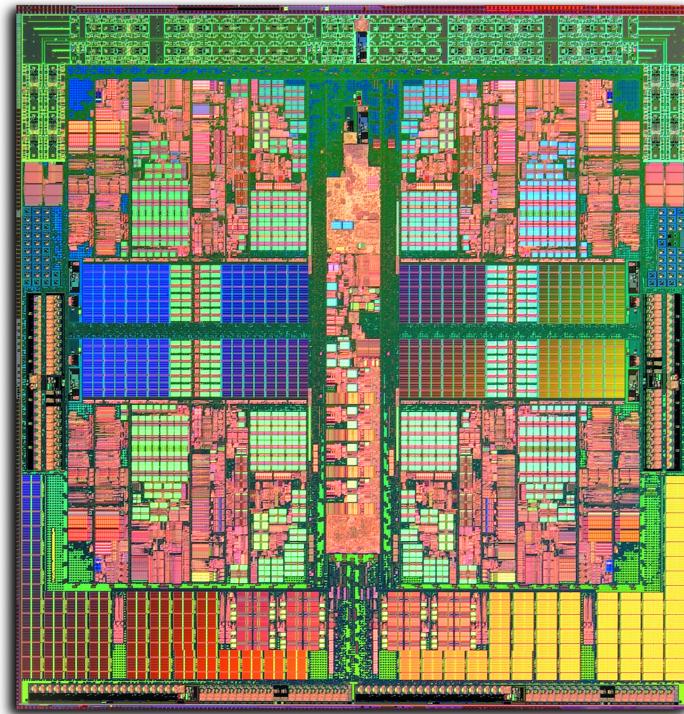
Ci-dessous : une capture d'écran du moniteur de ressources d'un processeur i7



- Cependant, tous ces coeurs utilisent la même mémoire vive. Quand un cœur travaille sur une certaine zone de la RAM, cette même zone n'est pas accessible aux autres coeurs, ce qui peut brider les performances. On trouve à l'intérieur des microprocesseurs de la mémoire "ultrarapide" appelée mémoire cache. Le CPU peut stocker certaines données

dans cette mémoire cache afin de pouvoir y accéder très rapidement. Enfin, il faut aussi que les applications aient été développées pour fonctionner en architecture multicoeurs, ce qui n'est pas toujours le cas...

Ci-dessous une vue agrandie de l'intérieur d'un processeur multicoeurs récent. On distingue la symétrie des 4 coeurs.



Exercice 7 :

Déterminer le nombre de coeurs du processeur de votre machine

Réponse :

FIN