Numérique et science informatique
Classe de première

Lycée Hoche

année scolaire 2025-2026

Contents

1 Un peu d'histoire

"L'ordinateur est né pendant la Seconde Guerre mondiale des travaux de plusieurs ingénieurs et théoriciens.
Il n'y a pas eu un pôle unique de développement mais plusieurs centres indépendants qui ont chacun tâtonné en essayant de construire des machines semblables à aucune autre existant à l'époque. Avant-guerre, Alan Turing (Angleterre) et Claude Shannon (États-Unis) avaient posé les fondements de la théorie du calcul, ouvrant la voie aux réalisations : Konrad Züse (Allemagne), John Atanasoff (États-Unis), Alan Turing et son équipe (Angleterre) ont développé entre 1940 et 1945 les premiers calculateurs.

À base de relais électriques ou de tubes électroniques, programmables ou non, ces prototypes ont montré la faisabilité du concept d'ordinateur. Souvent isolés, ces inventeurs n'ont pas été soutenus mais ont permis la construction après-guerre des premiers ordinateurs, lorsque les ingénieurs ont de nouveau pu s'y intéresser.

Même si des réalisations antérieures, plus ou moins semblables, ont existé, on attribue le titre de premier ordinateur (de manière quand même un peu arbitraire) à l'ENIAC (Electronic Numerical Integrator and Computer), œuvre de John W. Mauchly et de Prosper Eckert au sein de l'université de Pennsylvanie (États-Unis), finalisé en 1946.

En raison de la très large publicité qui a été faite, l'ENIAC a attiré de nombreux visiteurs qui s'emploieront, de retour dans leurs laboratoires, à développer leurs propres machines, faisant ainsi avancer les connaissances. L'un de ces visiteurs les plus connus a été le mathématicien John von Neumann, qui a rédigé à la suite un rapport où il exposait ses vues sur l'organisation interne d'un ordinateur. Il a mis, entre autres, l'accent sur le concept de programme enregistré. L'architecture décrite dans son rapport est à la base des ordinateurs depuis 65 ans.(d'après "architecture d'un ordinateur",Emmanuel Lazare)

2 Du transistor au CPU

2.1 Tube à vide et circuit intégré

A la base de la plupart des composants d'un ordinateur, on trouve le **transistor**.Ce composant électronique a été inventé par J.Bardeen , W.Shockley et W Brattain (Etats-unis , 1947).
L'invention du transistor fut un immense progrès , même si les premiers ordinateurs sont antérieurs à cette invention.

Les premiers ordinateurs étaient conçus à base de tubes électroniques .Bien que ces derniers soient plus gros et moins fiables qu'un transistor, leur fonctionnement ressemble à ce dernier.
Aujourd'hui les transistors sont regroupés au sein de ce qu'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.

2.2 Circuits logiques

Le transistor est l'élément de base des circuits logiques .Un circuit logique permet de réaliser une opération booléenne.Un circuit logique prend en entrée un ou des signaux électriques (chaque entrée est dans un état "haut" symbolisé par un "1" ou dans 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 "bas").
Il existe deux catégories de circuits logiques:

- **Les circuits combinatoires** : les états en sortie dépendent principalement des états en entrée.
- **Les circuits séquentiels**:Les états en sortie dépendent des états en entrée ainsi que du temps et des états antérieurs.

2.3 La mémoire vive (RAM)

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**:

Un bloc d'adresses

...								
adr yy								
adr1000								
adr1001								
adr1002								
...								
adr xx								
...								

Les opérations sur la mémoire sont de deux types:

- **Lecture**:Une opération de lecture consiste à aller lire l'octet situé à une certaine adresse mémoire **xx**.
- **Ecriture**:Une opération d'écriture consiste à écrire un octet donné à une certaine adresse mémoire **yy**.

Dans ce procédé , on n'est pas obligé de passer par une cellule pour accéder à une autre (comme dans l'accès séquentiel).
Cette faculté d'accéder directement à chaque cellule explique le nom **RAM**(Random access memory).

Aspect physique:

On peut réaliser un bit d'une cellule par l'association d'un transistor et d'un **condensateur**.
Un condensateur peut être soit *chargé* (on stocke alors un "1") soit *déchargé* (on stocke alors un "0").
Un condensateur ne peut conserver longtemps sa charge et il doit être alimenté électriquement afin de conserver cette charge.

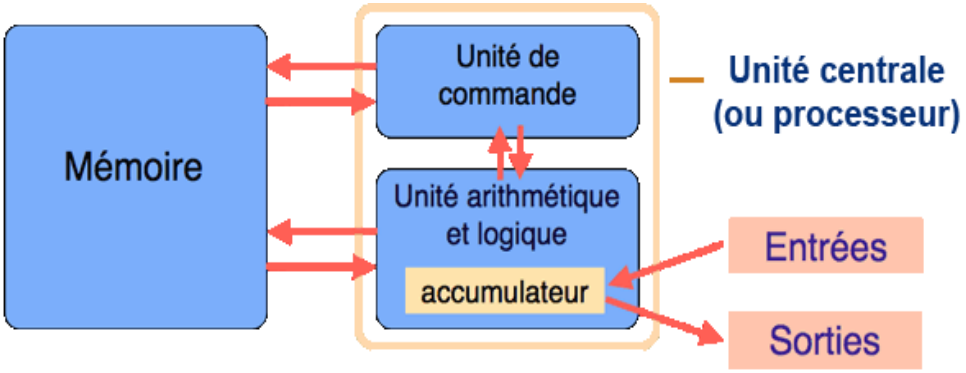
La mémoire vive est donc une mémoire volatile:Toutes les données de la mémoire vive sont perdues en cas de coupure de courant.

3 Le microprocesseur (CPU)

3.1 Organisation générale

Les instructions qui composent les programmes sont exécutées par le CPU (central processing unit).Il est schématiquement constitué de trois parties:

- L'unité arithmétique et logique (**UAL**) est chargée de l'exécution de touts les calculs que peut réaliser le microprocesseur.Nous retrouvons dans cette UAL des circuits comme l'additionneur.
- L'unité de commande permet d'exécuter les instructions (les programmes).
- Une toute petite quantité de mémoire appelée **les registres** , qui permettent de mémoriser de l'information transitoirement pour opérer dessus ou avec.Leur nombre et leur taille et leur rôle sont variables en fonction du type de microprocesseur.Certains registres jouent des rôles particuliers dans le fonctionnement du processeur et portent des noms en conséquence :on les notera R1,R2,R3.



le modèle de La machine de Von NeumannSource:Interstices

Les données doivent circuler entre les différentes parties d'un ordinateur, notamment entre la mémoire vive et le CPU.Le système permettant cette circulation est appelé **le bus**.Il existe trois 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..)

3.2 Fonctionnement

Un processeur donné est capable d'exécuter un certain nombre d'opérations de base, celles pour lesquelles il dispose d'un circuit électronique qui les réalise.

L'ensemble des instructions exécutables directement par le microprocesseur (instructions machines) constitue ce que l'on appelle le "**langage machine**" du processeur.

Chaque instruction machine correspond à une configuration binaire composée principalement de deux parties:

- Le champ "code opération" (opcode) qui indique au processeur le type de traitement à réaliser.Par exemple, sur un certain modèle de processuer , le code '00100110' donne l'ordre d'effectuer une multiplication.
- Le champ 'opérandes' indique la nature des données sur lesquelles l'opération désignée par le "code opération" doit être effectuée.
  Un opérande peut être de 3 natures différentes:
  - L'opérande est une valeur immédiate: l'opération est effectuée directement sur la valeur donnée dans l'opérande.
  - L'opérande est un registre du CPU : l'opération est effectuée sur la valeur située dans un des registres (R0,R1,R2..) .
    L'opérande indique de quel registre il s'agit.
  - L'opérande est une donnée située en mémoire vive:l'opération est effectuée sur la valeur située en mémoire vive à l'adresse xx.Cette adresse est indiquée dans l'opérande.

Le programme exécuté se trouve en RAM , tout comme les données.Un registre particulier du processeur , nommé IP (instruction pointer) ou PC (Program counter) , contient l'adresse de la cellule RAM de la prochaine instuction à exécuter.

Un deuxième registre , IR (instruction register) , joue un rôle important en raison de sa connexion physique au reste du processeur:placer dans ce registre la configuration électronique qui dénote une instructon provoque l'activation du circuit dédié à la réalisation de l'opération sous-jacente.

Le CPU a un **fonctionnement cyclique**:

- Il copie dans le registre IR le contenu de la RAM à l'adresse pointée par IP.
- Il décode l'instruction contenue dans IR:ceci provoque l'activation du circuit électronique qui réalise l'opération visée.
- il exécute l'instruction décodée;ceci met aussi à jour la valeur de IP pour continuer dans le programme. (...)

La capacité du processeur à exécuter tous les programmes s'explique par ce fonctionnement très souple et par le fait que le jeud'instructions de base est suffisamment riche pour être universel (on dit que le langage machine est Turing-complet).

Dans le modèle de von Neumann, il y a une seule mémoire vive pour le programme et les données : c'est par sa copie dans le registre IR qu'une configuration électronique initialement présente en RAM joue le rôle d'une instruction.
La même configuration pourrait être interprétée comme une donnée (entier, etc.) dans un autre contexte.La réciproque vaut aussi : une grande famille d'attaques informatiques consiste à exploiter des défauts dans le flux de contrôle d'un programme pour faire exécuter par le processeur des configurations électroniques qui étaient censées être des données,notamment des données choisies par l'utilisateur-attaquant.
Pour ces raisons de sécurité, les processeurs modernes sont dotés de la capacité de marquer des portions de la mémoire comme non-exécutables, s'écartant ainsi du principe d'origine du modèle de von Neumann.

3.3 Exemples d'instructions CPU

Historiquement, les instructions machine sont relativement basiques ; on peut se contenter d'à peine plus que ce qui est nécessairepour l'universalité. Pour des raisons de performance, les processeurs modernes savent exécuter nativement des opérations pluscomplexes, comme la composée addition-produit ou des opérations sur des vecteurs de petite dimension.

Les opérations fondamentales sont de trois sortes.

- Les instructions arithmétiques (addition, soustraction, multiplication...) effectuent des calculs mathématiques, soit sur des entiers,soit sur des flottants.
  Par exemple, on peut avoir une instruction consistant à additionner la valeur contenue dans le registre R1 et le nombre 789 et ranger le résultat dans le registre R0.
- Les instructions de transfert de données permettent de transférer une donnée d'un registre du CPU vers la mémoire vive et vice versa.
  Par exemple, on peut avoir une instruction consistant 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.Sont également essentielles les instructions de rupture de séquence ou instructions de saut.
  Elles permettent d'agir sur le registreIP qui contient l'adresse de la prochaine instruction à exécuter.

  Les instructions arithmétiques ou de transfert de données, outre leur effet spécifique, incrémentent la valeur de IP : de cette façon, l'exécution du programme avance séquentiellement.

- Les instructions de saut permettent de réaliser les autres structures de contrôle d'exécution, notamment l'alternative et la boucle.On distingue les instructions de saut inconditionnel, qui modifient toujours IP à la valeur donnée en opérande, et les instructions de saut conditionnel, qui ne font cette modification que selon certaines circonstances, et sinon ont le comportement habituel d'incrémentation de IP.

  Une instruction de saut conditionnel permet par exemple d'agir comme suit : si la valeur contenue dans le registre R1 est strictement supérieure à 0 alors la prochaine instruction à exécuter est celle située à l'adresse mémoire 4521

4 Langages de haut niveau

Pour simplifier l'écriture des programmes , on a inventé des langages encore plus proches de la langue courante (langue anglaise) : ce sont les langages de haut niveau comme **Python**, le **C**, **Java**, etc.

Les instructions d'un programme de haut niveau sont traduites en langage assembleur ou en langage machine par un compilateur, pour être comprises par la machine. Un même code source (par exemple en C), sera compilé en divers programmes en assembleur, suivant le processeur utilisé.

Le premier compilateur a été écrit par **Grace Hopper**.

Un programme écrit en langage **C** est d'abord compilé avant d'être exécuté. Ce n'est pas le cas de langages interprétés, comme Python.

5 Initiation à l'asembleur

Programmer en langage machine est extrêmement difficile (très longue suite de 0 et de 1), 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),cela donne l'assembleur.
Par exemple un ADD R1,R2,# 125 sera équivalent à "11100010100000100001000001111101".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 suite de 0 et de 1.

Une machine virtuelle permet de simuler le fonctionnement simplifié d'un microprocessur:

https://www.peterhigginson.co.uk/AQA/

On présente ci-dessous quelques instructions en assembleur ainsi que leur signification.

LDR R1 , 78	Place la valeur stockée à l'adresse mémoire 78 dans le registre R1 (pour simplifier on utilise la représentation en base 10 des adresses mémoires)
STR R3 , 125	Place la valeur stockée dans le registre R3 en mémoire vive à l'adresse 125.
ADD R1 , R0 ,#128	Additionne le nombre 128 et la valeur stockée dans le registre R0 puis 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 et place le résultat dans R0.
SUB R1 , R0 ,#128	Soustrait le nombre 128 de la valeur stockée dans R0 et place le résultat dans R1
SUB R0 , R1 , R2	Soustrait la valeur stockée dans le registre R2 de la valeur stockée dans R1 , place le résultat dans R0
MOV R1 , #23	Place le nombre 23 dans le registre R1
MOV R0 , R1	Place la valeur stockée dans R1 dans le registre R0
B 45	C'est une structure de rupture de séquence , la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 45
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
CMP R0 , R1	Compare la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1.
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 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 78 si la valeur stockée dans le registre R0 est plus grande que 23.
CMP R0 , #23 BLT 78	La prochaine instruction à exécuter se situe à l'adresse 78 si la valeur stockée dans le registre R0 est inférieure à 23.
HALT	arrête l'exécution du programme.

Exercice 1
Que font les instructions? suivantes

```
ADD R0, R1, #42
```

Et

```
CMP R4, #18
BGT 77
```

Exercice 2 Ecrire les instructions suivantes en assembleur:

1. Additionne la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1, le résultat est stocké dans le registre R5
2. 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.

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". Un label correspond à une adresse en mémoire vive. L'utilisation d'un label évite donc d'avoir à manipuler des adresses mémoires en binaire ou en hexadécimale

```
1 CMP R4, #18
2 BGT monLabel
3 MOV R0,#14
4 HALT
5 monLabel:
6 MOV R0,#18
7 HALT
```

5.1 Python et l'assembleur

Considérons un programme écrit en Python

```
1 x = 4
2 y = 8
3 if x == 10:
4 y = 9
5 else :
6 x=x+1
7 z=6
```

Il est possible de le traduire en assembleur:

```
MOV R0, #4
2 STR R0,30
3 MOV R0, #8
4 STR R0,75
5 LDR R0,30
6 CMP R0, #10
7 BNE else
8 MOV R0, #9
9 STR R0,75
10 B endif
11 else:
12 LDR R0,30
13 ADD R0, R0, #1
```

```
14 STR R0,30
15 endif:
16 MOV R0, #6
17 STR R0,23
18 HAL
```