Informatique et Création Numérique

1977 : ATARI 2600

- Processeur : 6507 (MOS Technology) 8 bits à 1,19 MHz
- Pas de vrai GPU
- RAM : 128 octets
- Jeu sur cartouches ROM : de 2 à 32 ko
- Résolution : 160 × 192 en 128 couleurs

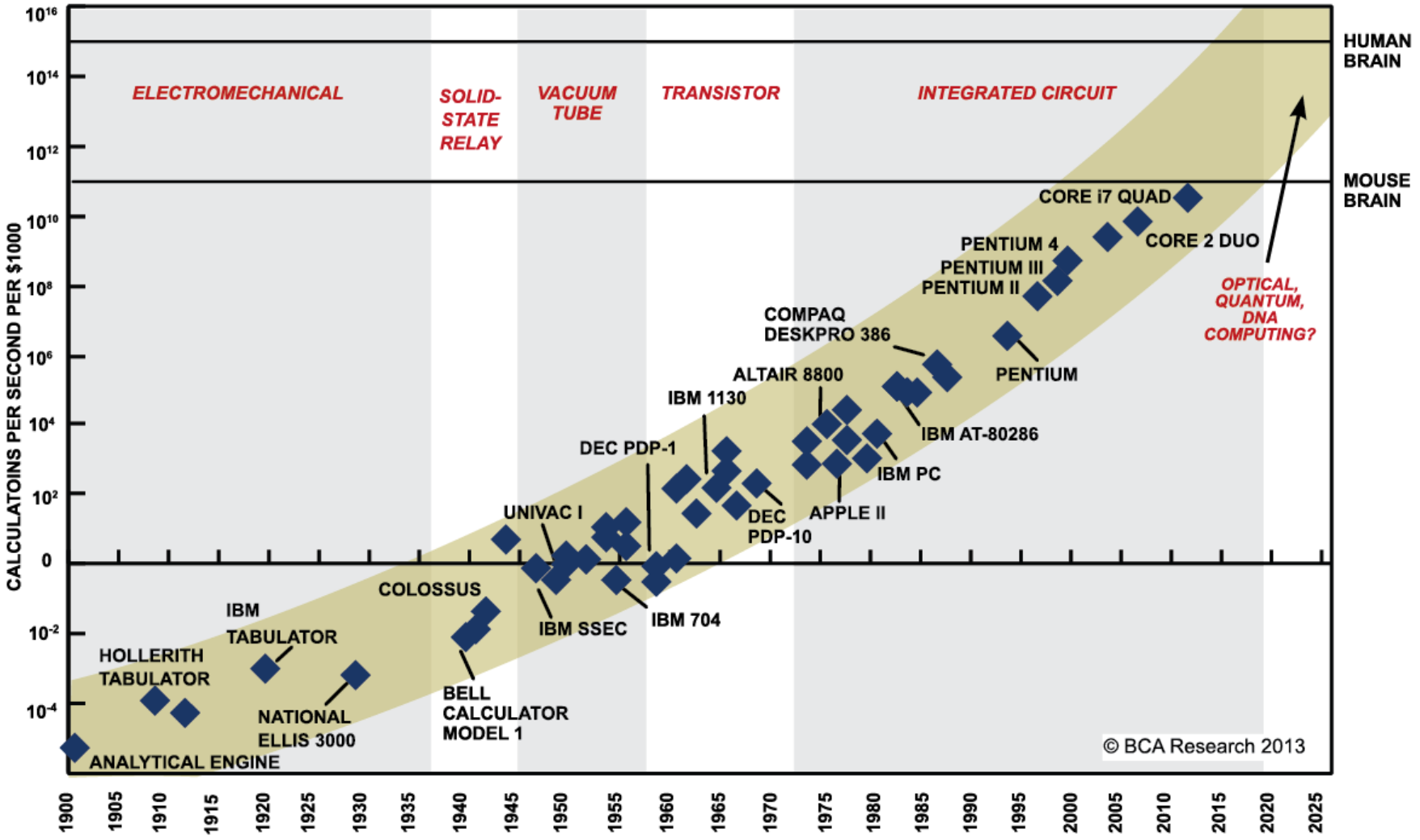
Star Wars: The Empire Strikes Back (Parker Brothers, 1982)

2014 : SONY PS4

- Processeur : Jaguar (AMD) 8 cœurs 64 bits à 1,6 GHz
- GPU : AMD Radeon HD 7000 à 800 MHz (18 cœurs graphiques)
- RAM : 8 Go
- Disque dur de 500 Go (ou 1 To) + lecteur Blu-Ray
- Résolution : Full HD (1920 × 1080) en True Color (16,7 millions de couleurs)

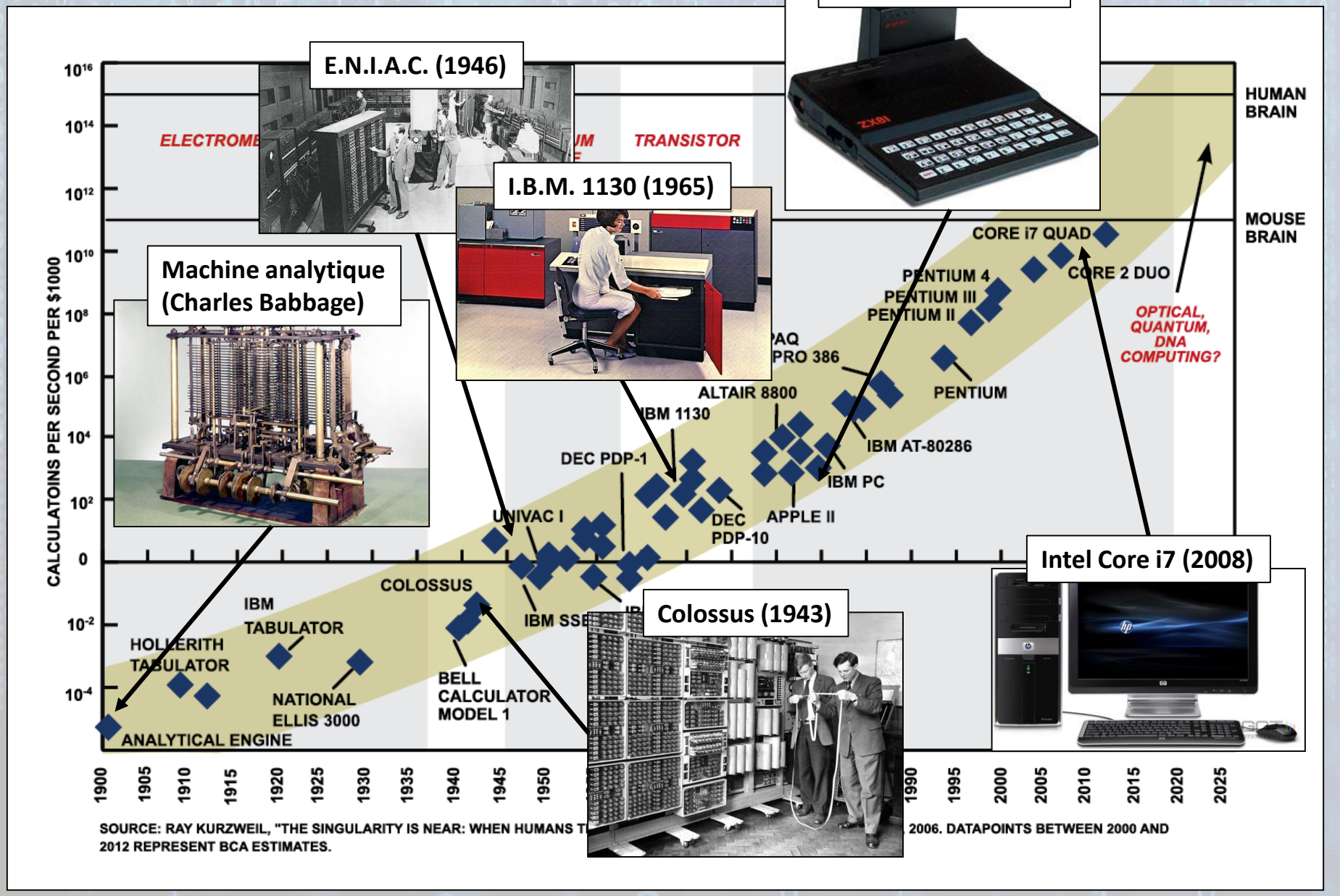
Star Wars: Battlefront (Electronic Arts, 2015)

Evolution de la technologie : loi (ou conjecture) de Moore



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Evolution de la technologie : loi (ou conjecture) de Moore



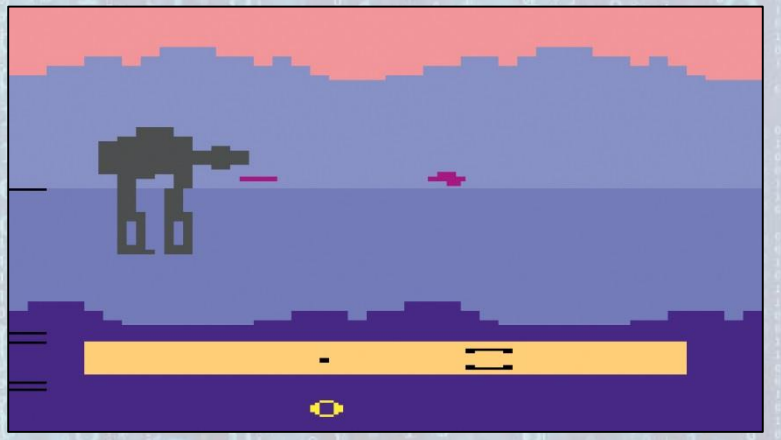
SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS T… 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

- **Evolution de la technologie :**
  - **Taille en diminution**
  - **Consommation en diminution (*pour une taille donnée*)**
  - **Puissance de calcul en augmentation**
  - **Complexité des systèmes en augmentation**

  Exemple : **La puissance du temps de calcul nécessité par une recherche Google à l'heure actuelle correspond à celle nécessaire à l'ensemble du programme spatial Apollo, qui a duré 11 ans et a lancé 17 missions (*Source :* Udi Manber et Peter Norvig, Google)**

- **Principe de base resté inchangé :**
  - **Programmation impérative (au plus bas niveau)**
  - **Architecture de Von Neumann (le plus souvent)**
  - **Peut être modélisé par une machine de Turing**

Développement d'un jeu



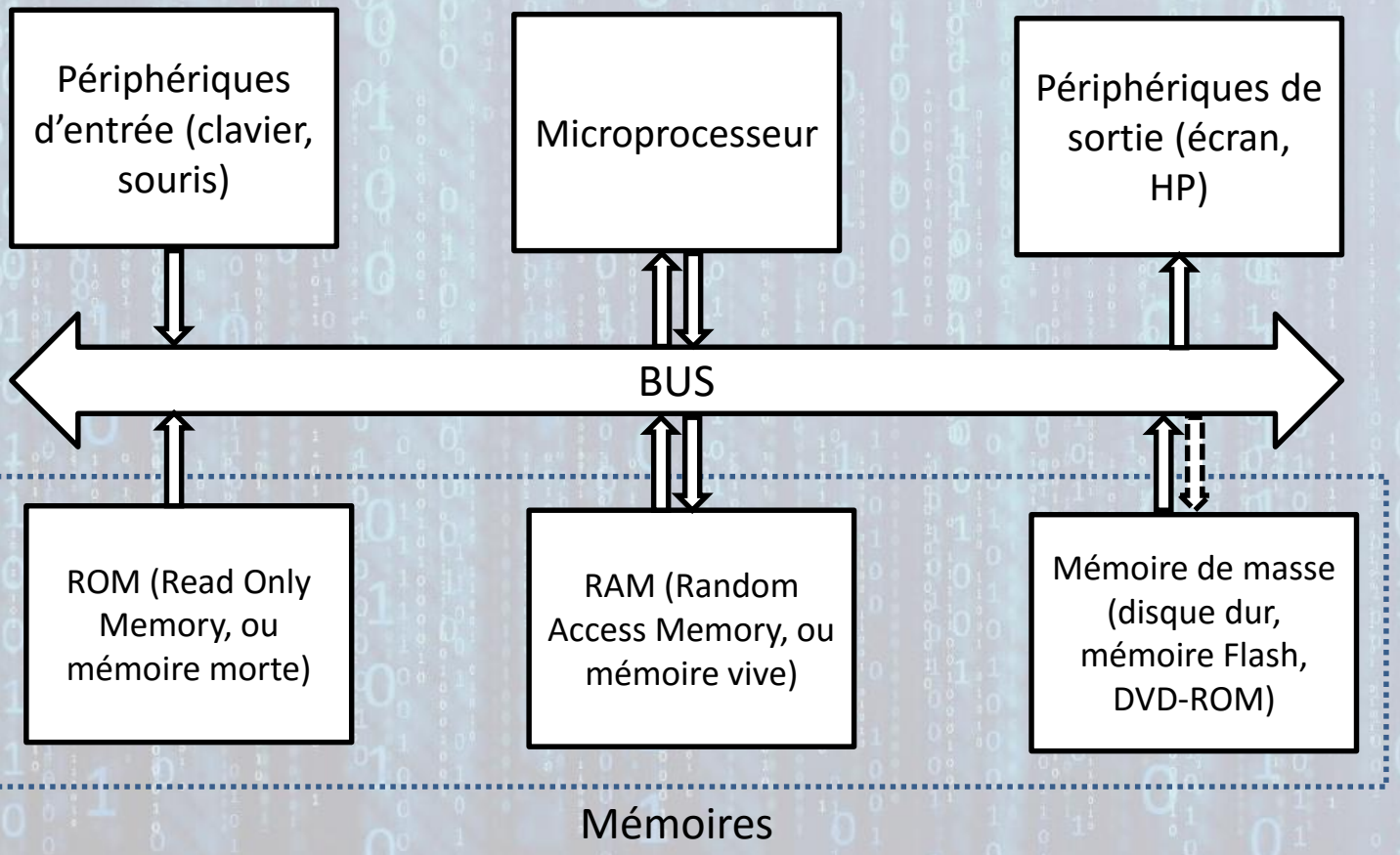
- Une personne seule
- Quelques jours à quelques semaines

⇒ €€€

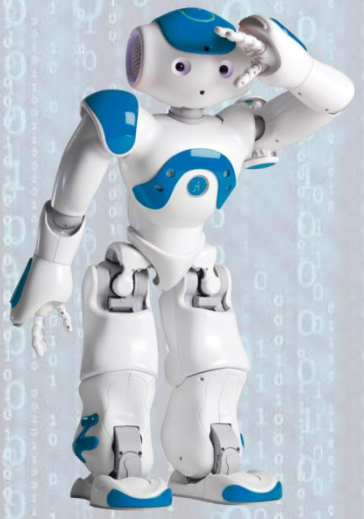
- Plusieurs dizaines de personnes
- Quelques mois à quelques années

⇒ **€€€**

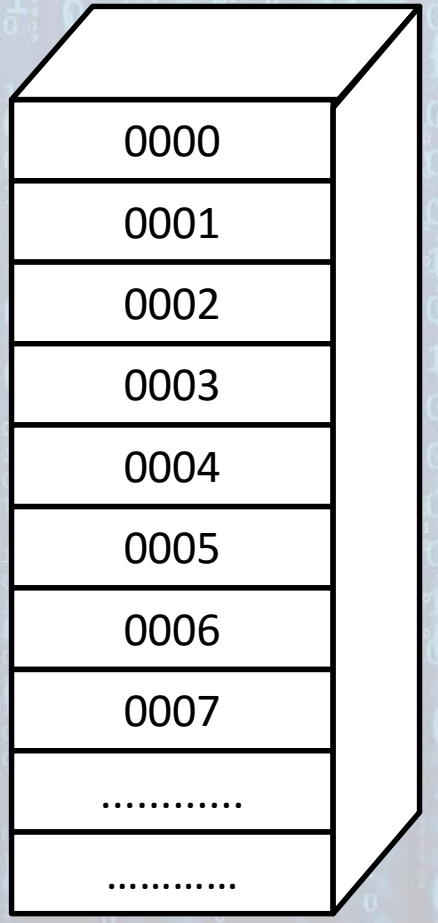
Principe (très simplifié !) d'un ordinateur



Principe (très simplifié !) d'un ordinateur

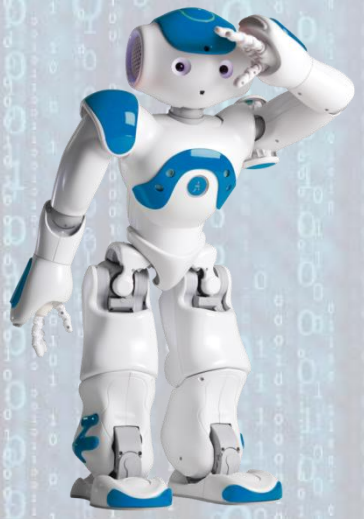


Microprocesseur

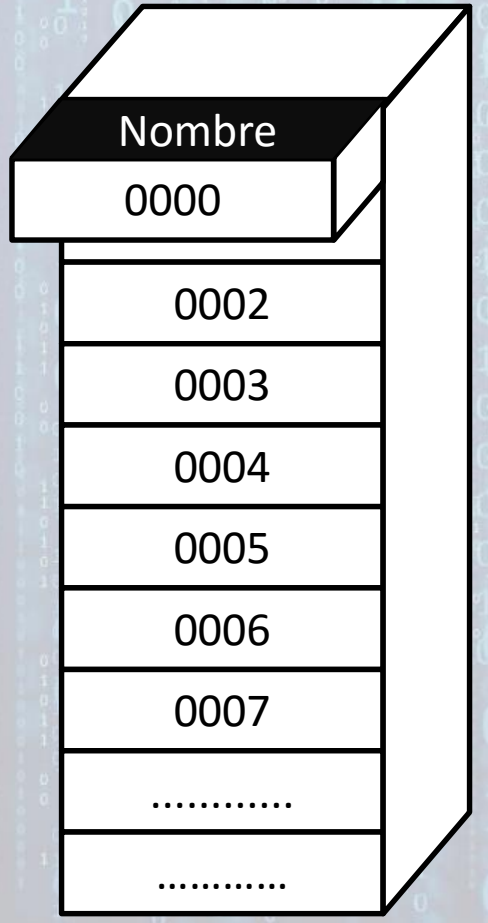


Mémoire

Principe (très simplifié !) d'un ordinateur

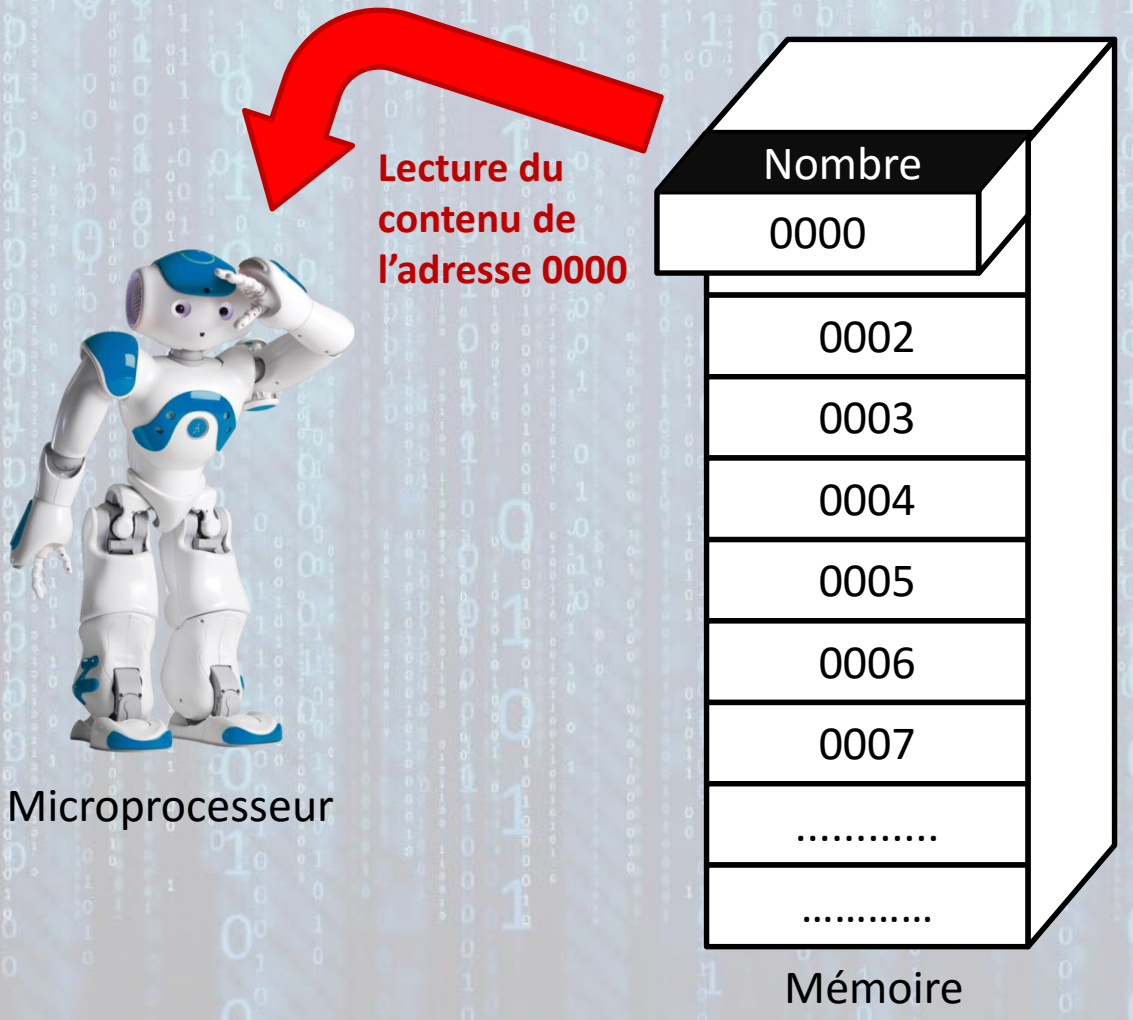


Microprocesseur



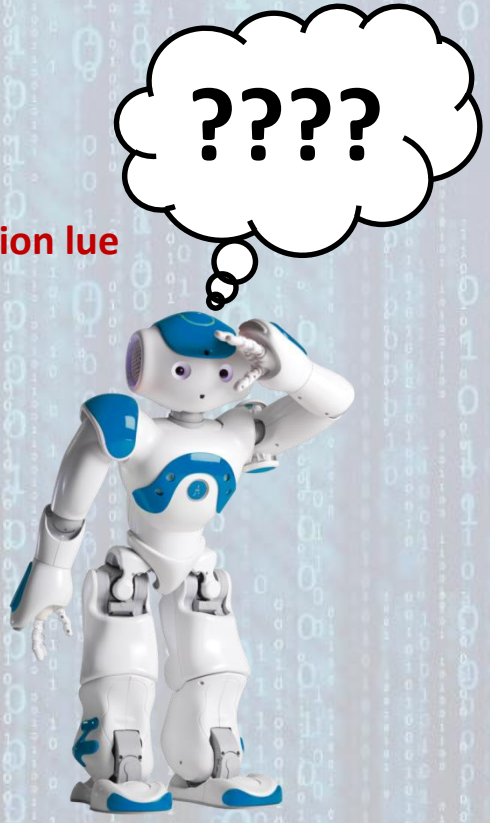
Mémoire

Principe (très simplifié !) d'un ordinateur

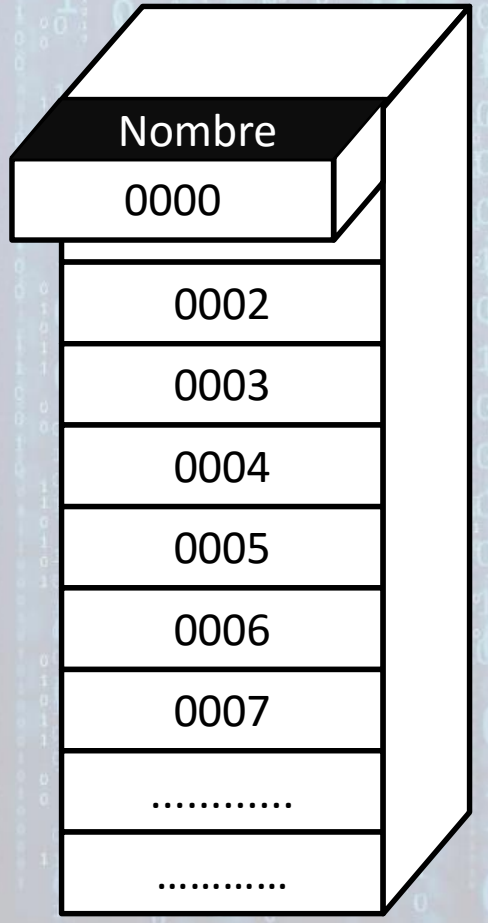


Principe (très simplifié !) d'un ordinateur

Décodage de l'instruction lue



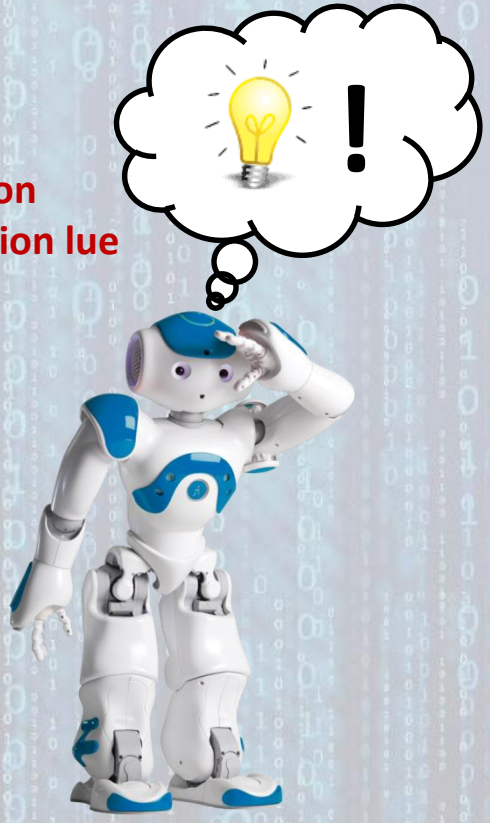
Microprocesseur



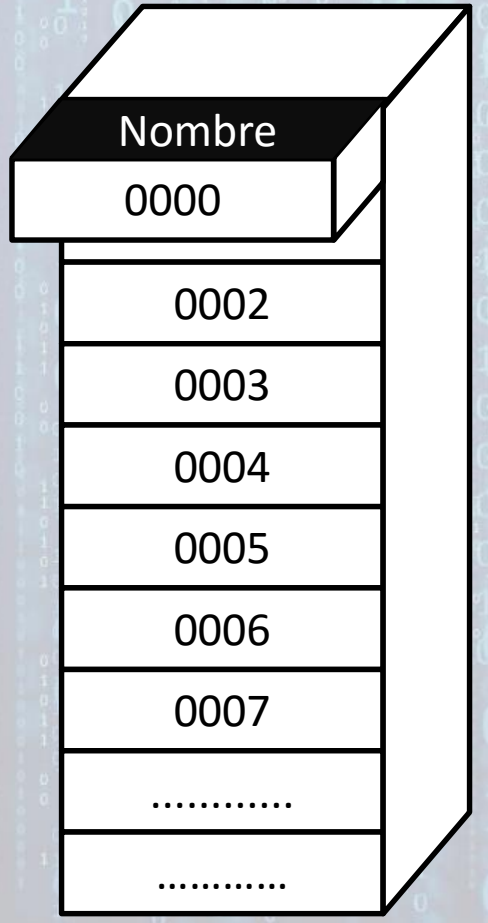
Mémoire

Principe (très simplifié !) d'un ordinateur

Interprétation
de l'instruction lue



Microprocesseur

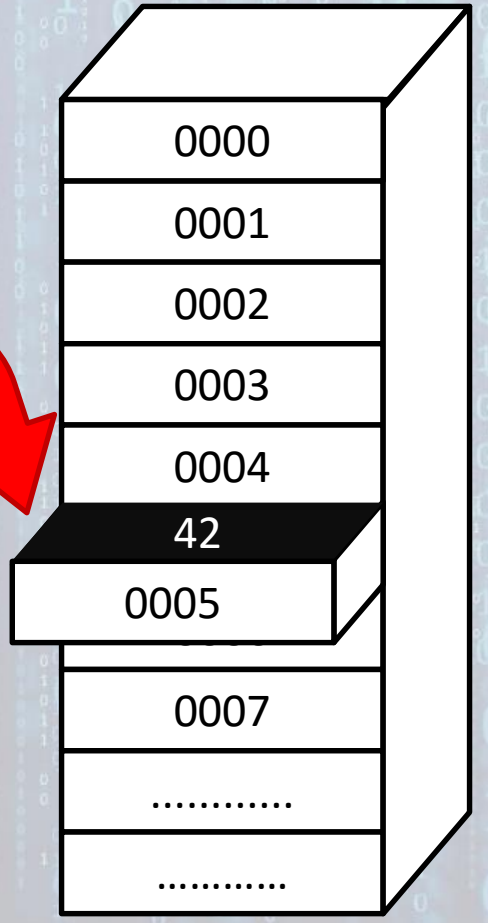


Mémoire

Principe (très simplifié !) d'un ordinateur

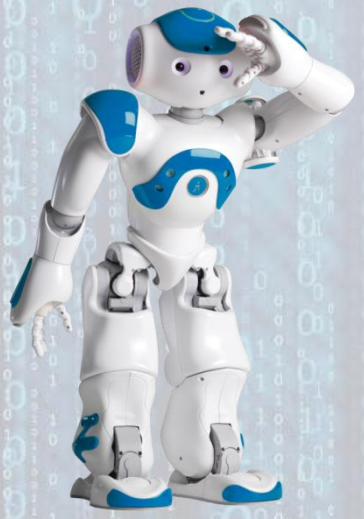
Microprocesseur

Exécution de l'instruction lue (exemple : « PLACER LA VALEUR 42 A L'ADRESSE 0005 »)



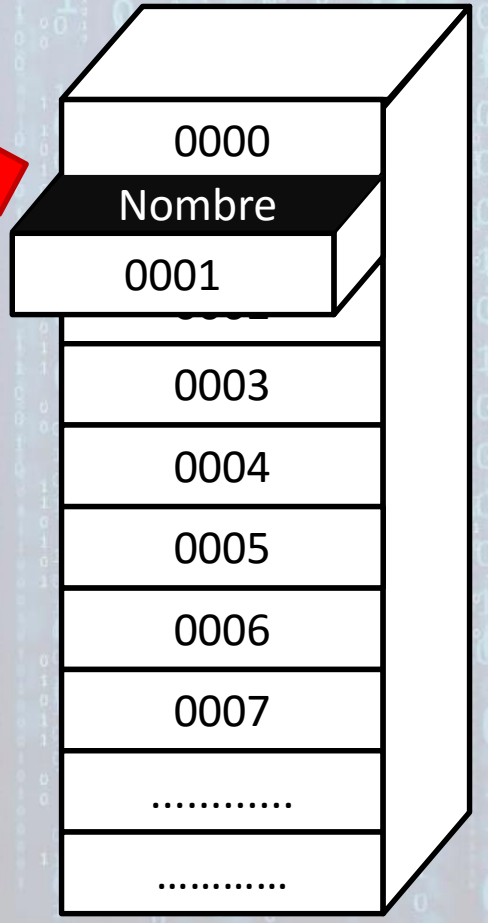
Mémoire

Principe (très simplifié !) d'un ordinateur



Microprocesseur

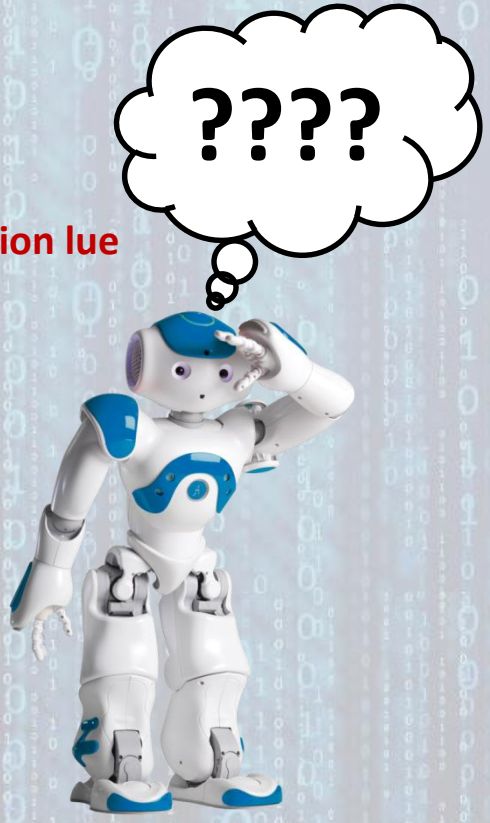
Lecture du contenu de l'adresse 0001



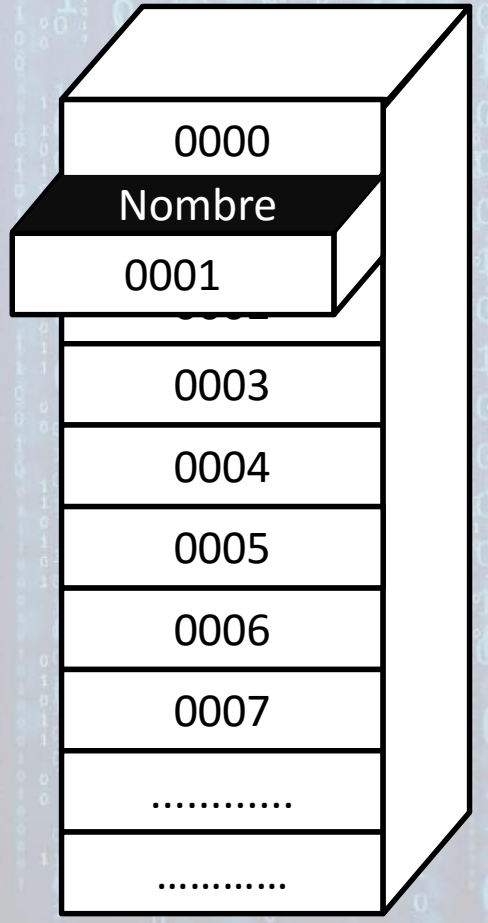
Mémoire

Principe (très simplifié !) d'un ordinateur

Décodage de l'instruction lue



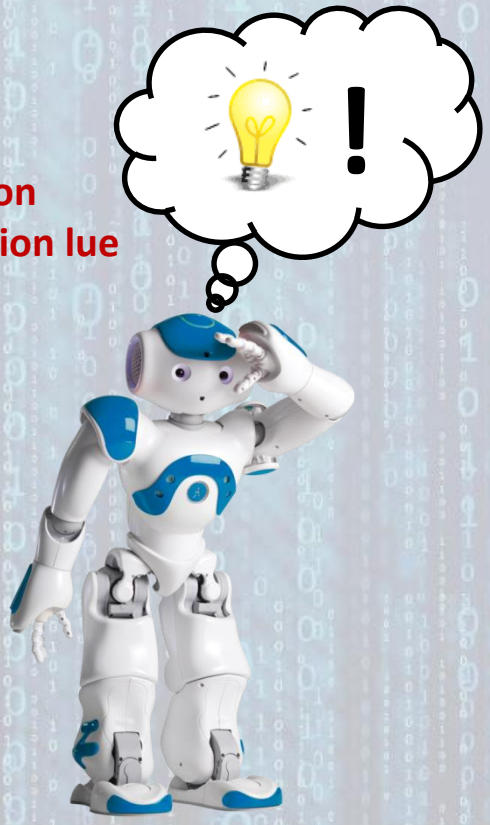
Microprocesseur



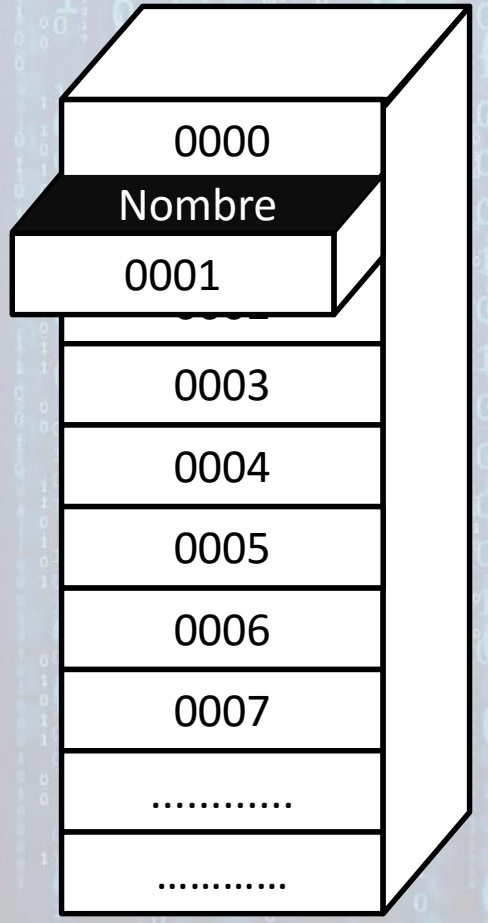
Mémoire

Principe (très simplifié !) d'un ordinateur

Interprétation de l'instruction lue



Microprocesseur

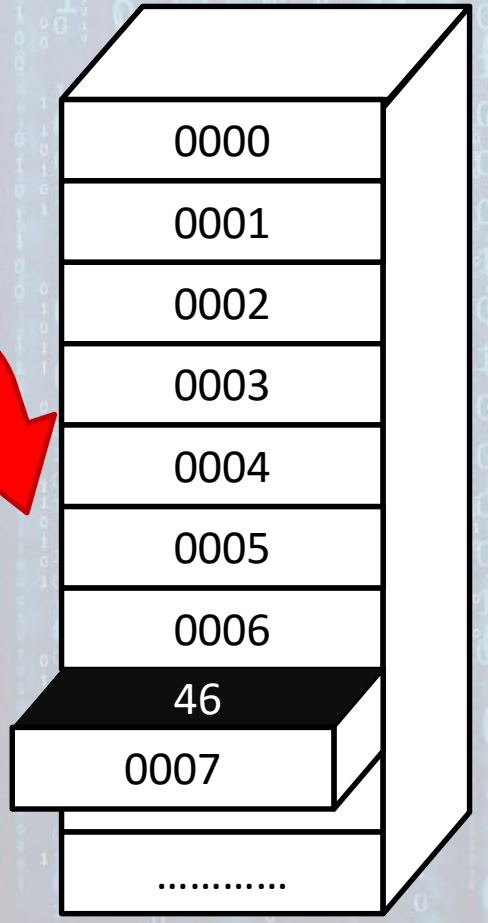


Mémoire

Principe (très simplifié !) d'un ordinateur

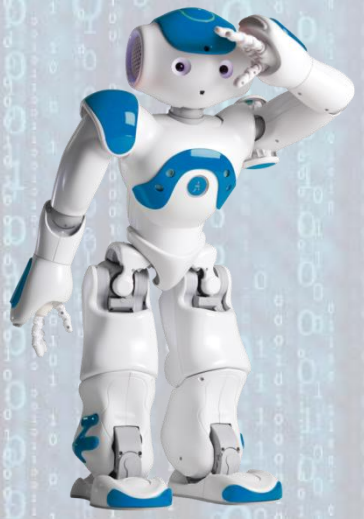
Microprocesseur

Exécution de l'instruction lue (exemple : « LIRE LE CONTENU DE L'ADRESSE 0005, AJOUTER 4 ET ECRIRE LE RESULTAT EN 0007 »)



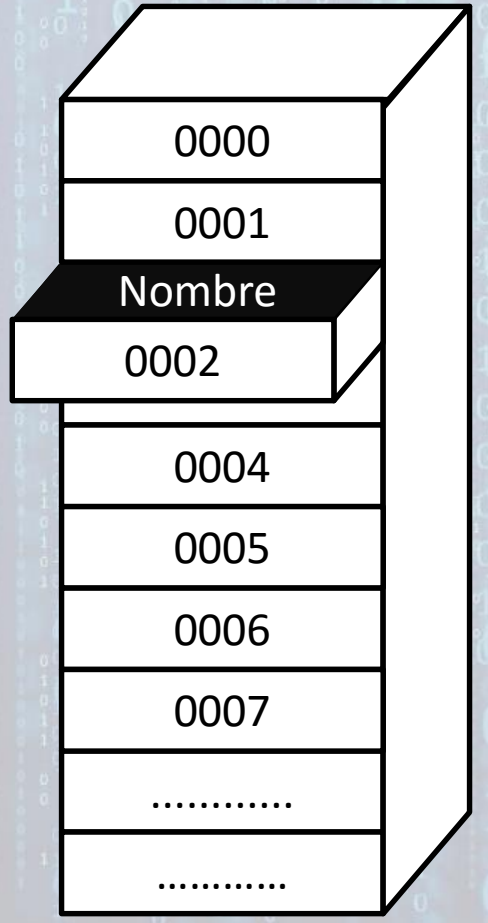
Mémoire

Principe (très simplifié !) d'un ordinateur



Microprocesseur

Etc…, etc…



Mémoire

Principe (très simplifié !) d'un ordinateur

Un programme est donc une suite d'instructions que va exécuter le microprocesseur.

Ces instructions sont codées sous forme de nombres stockés dans la mémoire de l'ordinateur.

Ces instructions sont très simples :

- Lire le contenu d'une case mémoire.
- Ecrire un nombre dans une case mémoire.
- Faire un calcul entre deux nombres.
- Comparer deux nombres.
- Sauter à une adresse particulière.
- Etc…

Deux points importants :

- Les données et le programme se trouvent au même endroit : la mémoire.
- Pour faire une tâche complexe, il faut de très nombreuses instructions élémentaires.

Principe (très simplifié !) d'un ordinateur

Les instructions sont stockées en mémoire sous forme de nombres binaires.
Un nombre binaire est constitué uniquement des chiffres 0 et 1.
C'est ce que l'on appelle le « ***langage machine*** » : c'est la langue naturelle du Microprocesseur, la seule qu'il comprenne.

Problèmes :

- Chaque microprocesseur a sa « propre langue ».
- Un programme en langage machine est difficilement lisible par un humain.

L'exemple simple vu précédemment (stocker 42 à l'adresse 5, puis rajouter 4 à cette valeur) peut se traduire ainsi en langage machine Z80 (microprocesseur 8 bits) :

00100001
00000101
00000000
00111110
00101010
01110111
11000110
00000100
01110111

Principe (très simplifié !) d'un ordinateur

Une première solution, pour rendre les programmes plus lisibles est de traduire le binaire en ***hexadécimal*** (base 16), mais le résultat reste encore mystérieux :

21 00 05 3E 2A 77 C6 04 77

On peut alors décider de traduire les codes numériques correspondant aux différentes instructions par des mots, que l'on appelle des ***mnémoniques***, et en mettant le tout en forme. On obtient alors le programme en ***assembleur*** :

```
LD HL,0005
LD A,42
LD (HL),A
ADD A,4
LD (HL),A
```

Problèmes :

- Cela reste compliqué à lire et à interpréter par un humain.
- La moindre tâche complexe demande des centaines de lignes en assembleur !
- Si on change de famille de microprocesseur, il faut traduire tout le programme.

Principe (très simplifié !) d'un ordinateur

L'hexadécimal et l'assembleur ne sont que des transcriptions différentes du langage binaire : c'est pourquoi on confond souvent « assembleur » et « langage machine ».

Pour résoudre les problèmes évoqués précédemment, les informaticiens ont inventé d'autres langages, dits « ***langages de haut niveau*** » : BASIC, Fortran, COBOL, Pascal, C, C++, C#, Java, Python, etc… Il en existe plusieurs centaines de différents (plusieurs milliers si on compte toutes les variantes). Ces langages permettent :

- Une plus grande concision, car ils permettent des opérations plus complexes.
- Une meilleure lisibilité (à condition d'être anglophone).
- Une meilleure portabilité (dans une certaine mesure).

Le programme précédent pourrait s'écrire de la façon suivante en BASIC :

```
A = 42
A = A + 4
```

Principe (très simplifié !) d'un ordinateur

Si on veut afficher la table des carrés et des cubes des entiers de 0 à 20, on peut Le faire avec un programme BASIC de 3 lignes :

```
FOR I = 0 TO 20
PRINT I, I^2, I^3
NEXT I
```

Si on veut faire la même chose en assembleur, on obtiendra un listing beaucoup plus complexe, de plusieurs dizaines de lignes de code. Un langage de haut niveau est donc plus intéressant pour développer des applications complexes (comme un jeu, par exemple).

Problème :

Un microprocesseur ne comprend **QUE** le langage machine. Il faut donc traduire le programme que l'on a réalisé dans un langage de haut niveau en langage machine, compréhensible par l'ordinateur.

Deux méthodes sont possibles : la ***compilation*** et l'***interprétation***.

Principe (très simplifié !) d'un ordinateur

Compilation : un programme, appelé ***compilateur*** va lire tout le texte du listing écrit en langage de haut niveau, tout traduire dans le langage machine de la machine cible, et produire un programme en langage machine directement exécutable par l'ordinateur.

Avantage : programme très rapide, car directement dans la langue du microprocesseur.
Inconvénient : difficulté de mise au point (il faut recompiler à chaque modification).

Interprétation : un programme, appelé ***interpréteur*** va lire la première ligne du listing écrit en langage de haut niveau, la traduire dans le langage machine de la machine cible, puis l'exécuter. Il va ensuite lire la deuxième ligne, la traduire, puis l'exécuter, etc…

Avantage : Mise au point très aisée (on modifie, puis on relance directement).
Inconvénient : programme plus lent (temps de traduction + temps d'exécution).

Remarque : Certains langages (comme Java) font un compromis entre les deux méthodes (bytecode).

Que va-t-on faire en ICN cette année ?

Découvrir la notion d'environnement de développement à l'aide de la plateforme Processing.

Apprendre à décomposer une tâche complexe en suite d'instructions simples.

(Re)voir les notions de bases d'algorithmique : tests, boucles, branchements, …

Apprendre la syntaxe du langage en réalisant des programmes de plus en plus complexes.

Pourquoi Processing ?

Processing est gratuit, et il y a une vaste documentation sur le web (y compris en français).

Processing permet de réaliser facilement des applications graphiques par rapport à d'autres langages de programmation.

Processing a une syntaxe apparentée à celle d'autres langages très répandus (comme Java ou C).

Si on sait programmer dans un langage de programmation quelconque, alors on sait programmer tout court. L'apprentissage d'un nouveau langage de programmation est alors une question de jours.

Il n'y a pas un langage meilleur qu'un autre dans l'absolu : tout dépend de ce que l'on veut faire ainsi que des goûts personnels du programmeur.