PREMIERE GENERALE

NUMERIQUE ET SCIENCES INFORMATIQUES

# ARCHITECTURE D'UN ORDINATEUR

- ▷ **Histoire de l'informatique**
- ▷ Représentation des données : types et valeurs de base
- ▷ Représentation des données : types construits
- ▷ **Architectures matérielles et systèmes d'exploitation**
- ▷ Interactions entre l'homme et la machine sur le Web
- ▷ Langages et programmation
- ▷ Algorithmique

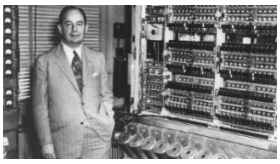
**John von Neumann** (1903-1957), mathématicien et physicien américain d'origine hongroise, a apporté d'importantes contributions tant en mécanique quantique, en théorie des ensembles, en informatique, en sciences économiques ainsi que dans d'autres domaines des mathématiques et de la physique.

**Gordon Earle Moore** est le cofondateur avec Robert Noyce et Andrew Grove de la société Intel en 1968, premier fabricant mondial de microprocesseurs. Il est connu pour avoir publié une loi empirique portant son nom, la loi de Moore, le 19 avril 1965 dans le magazine Electronics.

## 1 – COMPOSANTS D'UN ORDINATEUR

### 1.1 – Introduction

Le principe de fonctionnement des ordinateurs actuels repose sur des travaux réalisés au milieu des années 40 par une équipe de chercheurs de l'université de Pennsylvanie. Ces travaux concernaient la conception d'un ordinateur, l'**EDVAC**, dans lequel les programmes et les données étaient stockés dans la mémoire de l'ordinateur. Cette architecture, qui utilise une zone de stockage unique pour les programmes et les données, est appelée modèle de **Von Neumann**.

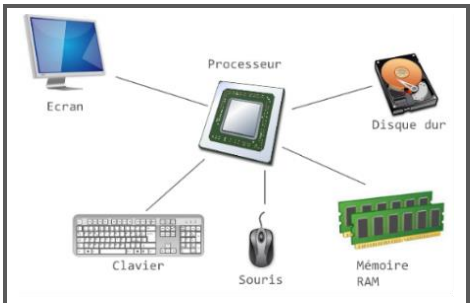


Von Neumann et l'EDVAC

### 1.2 – Composition d'un ordinateur

Un ordinateur est constitué **de composants électroniques** capables de faire fonctionner des programmes informatiques. On parle de **hardware** pour désigner l'ensemble des **éléments matériels** et de **software** pour la **partie logicielle**.

Les composants matériels sont **organisés autour de la carte mère** comportant le microprocesseur, la mémoire RAM, quelques autres circuits intégrés et de composants électroniques.



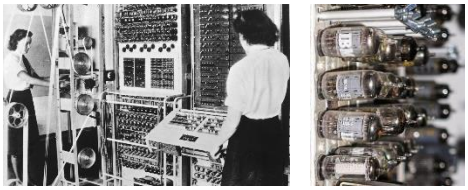

Composants essentiels d'un ordinateur

## 1.3 – Un peu d'électronique

Un ordinateur traite uniquement des données binaires constituées de « 1 » et des « 0 ». 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.

Réplique du 1er transistor

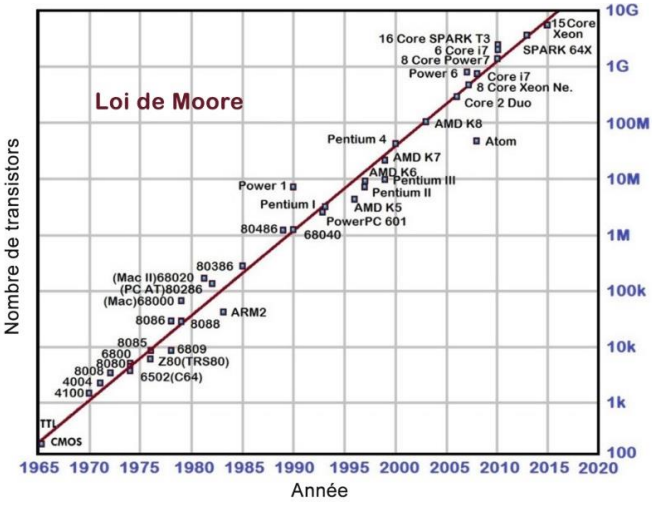
Les premiers ordinateurs sont antérieurs au transistor. En effet, ces premiers ordinateurs, par exemple le Colossus qui date de 1943, étaient conçus à base de **tubes électroniques** (on parle aussi de tubes à vide) qui, bien que beaucoup plus gros et beaucoup moins fiables que les transistors fonctionnent sur le même principe que ce dernier.



Colossus et ses tubes

L'invention du transistor va révolutionner la conception des ordinateurs en permettant la miniaturisation des composants électroniques et notamment de diminuer la taille des microprocesseur.

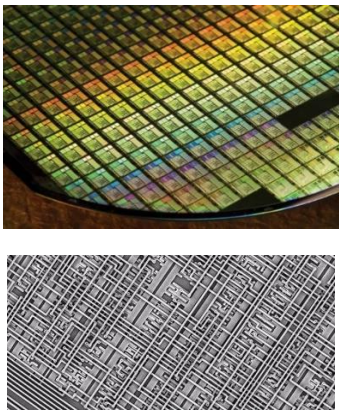
En 1975, Gordon E. Moore (cofondateur de la société Intel) énonça : « *Dans les microprocesseurs, le nombre de transistors sur une puce va doubler tous les deux ans* ». Il fondait sa prédiction sur le constat (empirique) de l'évolution effective relevée entre les années 1965 et 1975. Elle s'est, pour l'instant, toujours avérée exacte.




Loi de Moore

Aujourd'hui, la taille d'un transistor n'est que de **7 nanomètres**, soit la taille de quelques dizaines d'atomes de silicium. Cette miniaturisation permet :

- Une augmentation de la **fréquence de fonctionnement** car les distances entre les composants sont plus petites.
- Une Baisse de la **tension d'alimentation** et donc de la **consommation** énergétique.
- Une baisse des **coûts** car il est possible d'intégrer plusieurs composants sur une même puce.

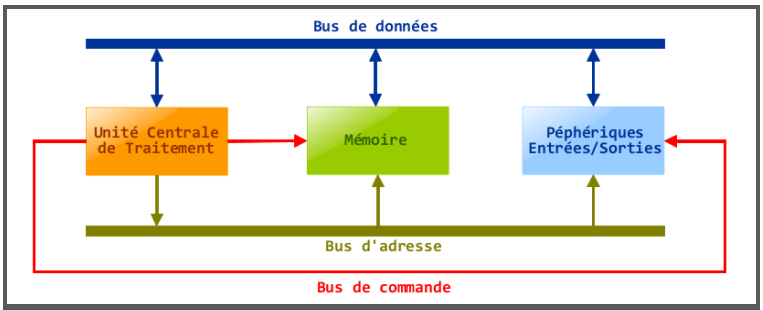


Wafer de processeurs

# 2 – MODELE DE VON NEUMANN

## 2.1 – Présentation

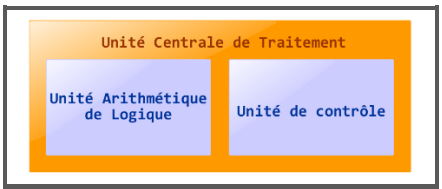
L'architecture d'un ordinateur selon le modèle de Von Neumman est décrit sur la figure ci-contre :



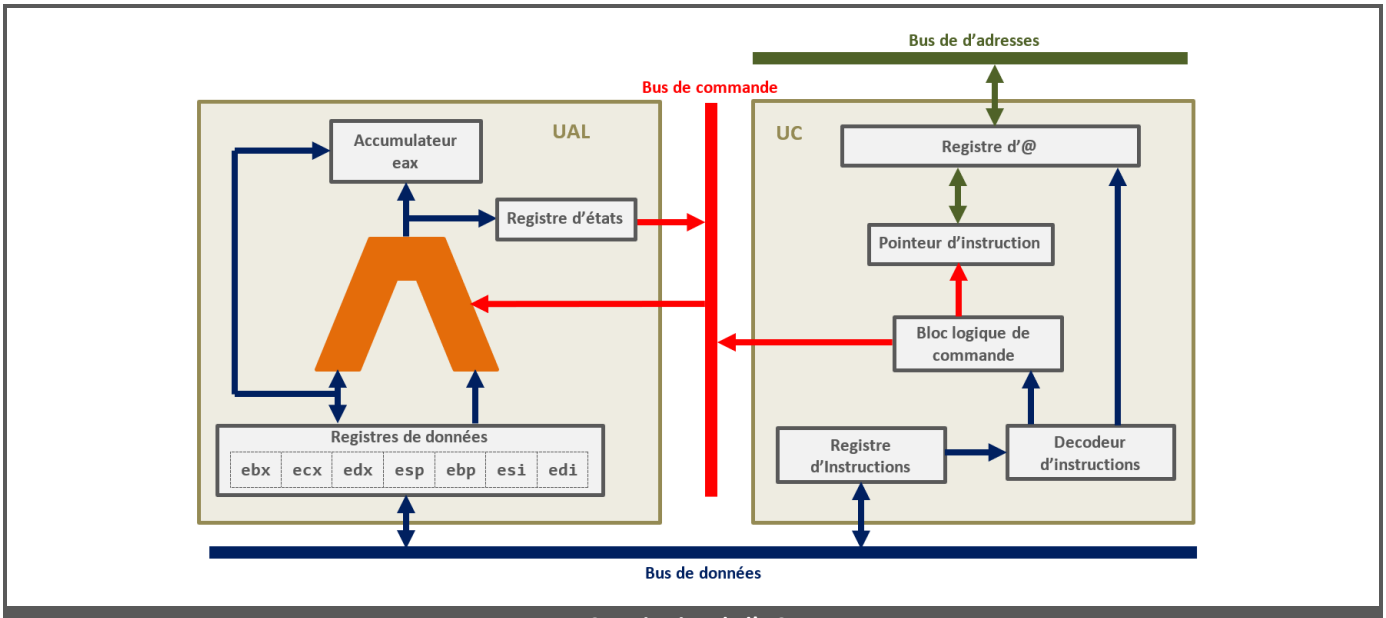

Modèle Van Neumann

## 2.2 – Unité Centrale de Traitement

L'**UCT** ou **U**nité **C**entrale de **T**raitement ( **CPU** : **C**ontrol **P**rocessing **U**nit) est le microprocesseur. Elle est constituée de deux unités importantes : L'**UAL** ou **U**nité **A**rithmétique et **L**ogique ou (**ALU** : **A**rithmetic **L**ogic **U**nit) et l'**UC** ou **U**nité de **C**ontrôle (**CU** : **C**ontrol **U**nit).




Unité Centrale de Traitement




Constitution de l'UCT

L'**UAL** est un circuit électronique qui effectue à la fois des opérations arithmétiques et des opérations logiques sur les nombres binaires. Il est composée de plusieurs **registres** de données et d'un registre spécial appelé **accumulateur** sur lequel vont s'effectuer tous les calculs intermédiaires. À ces registres s'ajoutent tout un tas de circuits électroniques permettant réaliser :

- les **opérations arithmétiques** : addition, soustraction, etc. ;
- les **opérations logiques** : ET, AND, NON, etc. ;
- les **comparaisons** : égalité, inférieur, supérieur, etc. ;
- les **opérations sur les bits** : décalages, rotations ;l
- les **opérations de déplacements mémoire** : copie de ou vers la mémoire

Les entrées d'une UAL sont les données sur lesquelles elle va effectuer une opération (on parle d'opérandes). Ces registres sont chargés avec des valeurs venant de la mémoire de l'ordinateur et c'est l'unité de contrôle qui indique quelle opération doit être effectuée. Le résultat du calcul se trouve toujours dans l'accumulateur. L'UAL peut envoyer des signaux, à partir du registre d'état, pour indiquer des erreurs : division par 0, dépassement de valeur, etc. Elle peut également envoyer des résultats de comparaison.

L'UC joue le rôle de chef d'orchestre de l'ordinateur. C'est ce composant qui se charge de récupérer en mémoire la prochaine instruction à exécuter et les données sur lesquelles elle doit opérer, puis les envoie à l'**UAL**. Cette unité est essentiellement constituée de trois composants :

- Le **registre d'instruction IR** (**I**nstruction **R**egister), qui contient l'instruction courante à décoder et exécuter.
- Le **pointeur d'instruction**, dénommé **IP** (**I**nstruction **P**ointer), qui indique l'emplacement mémoire de la prochaine instruction à exécuter.
- Le **bloc logique de commande** qui contrôle presque tous les mouvements de données de la mémoire vers l'UAL (et réciproquement) ou les périphériques d'entrée-sortie.

## 2.3 – Mémoires

La mémoire de l'ordinateur contient à la fois les programmes et les données. Ondistingue deux types de mémoires :

- La **mémoire vive** (ou **volatile**) est celle qui perd son contenu dès que l'ordinateur est éteint. Les données stockées dans la mémoire vive d'un ordinateur peuvent être lues, effacées ou déplacées comme on le souhaite. Le principal avantage de cette mémoire est la rapidité d'accès aux données qu'elle contient. On parle souvent de mémoire **RAM** (**R**andom-**A**ccess **M**emory).
- La **mémoire morte** (ou **non volatile**) est celle qui conserve ses données quand on coupe l'alimentation électrique de l'ordinateur. Contrairement à la RAM, ces mémoires sont souvent beaucoup plus lentes. Il existe plusieurs types de telles mémoires :

| | |
|---|---|
| **ROM** (**R**ead **O**nly **M**émory) | Mémoire dont l'information est enregistrée à la fabrication et ne peut être modifiée. Ce type de mémoire est peu utilisée aujourd'hui. |
| **EEPROM** (**E**lectrically-**E**rasable **P**rogrammable **R**OM) | Mémoire non volatile qui peut être effacée et reprogrammée plusieurs fois. |
| **FLASH** | Mémoire similaire à la mémoire EEPROM mais avec un accès beaucoup plus rapide. |

## 2.5 – Limitations du modèle Van Neumann

Le modèle de von Neumann impose un va-et-vient constant entre l'UCT et la mémoire, soit pour charger la prochaine instruction à exécuter, soit pour récupérer les données sur lesquelles l'instruction courante doit opérer.

Cependant, la différence de vitesse entre les microprocesseurs (nombre d'opérations par seconde) et la mémoire (temps d'accès) est telle qu'aujourd'hui, avec cette architecture, les microprocesseurs modernes passeraient tout leur temps à attendre des données venant de la mémoire : **goulot d'étranglement du modèle de von Neumann**.

Pour tenter de remédier à ce problème, les microprocesseurs intègrent de la **mémoire cache**. Il s'agit de composants très rapides qui stockent les données les plus utilisées chances d'être utilisée afin de limiter l'accés à la mémoire RAM.

D'autres solutions comme les **architectures dites parallèles** constituées de plusieurs CPU qui exécutent chacun un programme, de manière indépendante, sur des données différentes.

Il existe un autre type architecture fréquemment utilisé essentiellement dans les microcontrôleurs dans les systèmes sur puces ou SOC. Il s'agit de la structure dite de **Harvard** dans laquelle la mémoire programme est séparée de la mémoire des données. L'UCT y accède à travers deux bus de communication bien distincts.

## Exercice n°1

**1.** Quel est le rôle de l'Unité Arithmétique et Logique dans un processeur?

| | |
|---|---|
| a | La gestion interne du processeur |
| b | Faire les calculs |
| c | Interfacer les différents périphériques |
| d | Définir la base arithmétique |

**2.** Parmi les affirmations suivantes, laquelle est vraie ?

| | |
|---|---|
| a | La mémoire RAM est une mémoire accessible en lecture seulement |
| b | La mémoire RAM est une mémoire accessible en écriture seulement |
| c | La mémoire RAM est une mémoire accessible en lecture et en écriture |
| d | La mémoire RAM permet de stocker des données après extinction de la machine |

**3.** Dans l'architecture de Von Neumann, le processeur (UCT) contient :

| | |
|---|---|
| a | L'unité de commande et l'unité arithmétique et logique |
| b | L'unité de commande et la RAM |
| c | Seulement l'unité arithmétique et logique. |
| d | Seulement l'unité de commande |

**4.** Que dit la loi de MOORE ?

| | |
|---|---|
| a | Que le nombre de transistors sur une puce va doubler tous les deux ans. |
| b | Que la vitesse des horloges va doubler tous les deux ans. |
| c | Que la puissance des ordinateurs va doubler tous les deux ans. |
| d | Que la taille des ordinateurs va être divisée par deux tous les deux ans. |

**5.** Que-ce qui n'est pas considéré comme un périphérique de l'ordinateur ?

| | |
|---|---|
| a | Disque dur |
| b | Clavier |
| c | CPU |
| d | Moniteur |

**6.** Laquelle des mémoires suivantes est non volatile, donc qui ne s'efface pas quand on éteint l'ordinateur ?

| | |
|---|---|
| a | SRAM |
| b | DRAM |
| c | ROM |
| d | Toutes les réponses sont vraies |

**7.** L'unité qui permet de séquencer, décoder, traduire chaque instruction et générer les signaux d'activation nécessaires pour l'ALU et d'autres unités s'appelle ...

| | |
|---|---|
| a | l'unité arithmétique |
| b | le CPU |
| c | l'unité logique |
| d | l'unité de contrôle |

**8.** Lequel de ces composants n'est pas une mémoire ?

| | |
|---|---|
| a | Barrette de RAM |
| b | Lecteur/Graveur DVD-RW |
| c | Clé USB |
| d | Disque dur externe |

**9.** « BUS » est le nom que l'on donne en informatique :

| | |
|---|---|
| a | Au Board Unit System |
| b | Aux registres du processeur |
| c | Aux fils qui permettent de transmettre les données d'un composant à un autre |
| d | A l'horloge de la carte mère |

# 3 – LANGAGE MACHINE OU ASSEMBLEUR

## 3.1 – Introduction

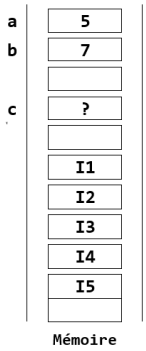
Les programmes stockés dans la mémoire centrale d'un ordinateur sont constitués de suites de « 1 » et de « 0 » directement compréhensibles par l'UCT. Cette suite binaire permet de coder les instructions à réaliser et les données ou l'emplacement des données sur lesquelles les instructions doivent s'appliquer.

Pour progresser dans l'exécution d'un programme, l'UC réalise de manière continue, à une fréquence imposée par l'horloge, le **cycle d'exécution d'une instruction** constitué de trois étapes :

- **Chargement** : l'UC va récupérer, à l'adresse mémoire indiquée par le pointeur d'instruction, le mot binaire qui contient la prochaine instruction à exécuter et le stocke dans le registre d'instuction.
- **Décodage** : La suite de bits contenue dans le registre IR est décodé afin de déduire quelle instruction doit être exécutée et sur quelles données. Cette étape permet également de charger les données (opérandes) qui peuvent être dans les registres de l'UAL ou en mémoire.
- **Exécution** : L'instruction est exécutée par l'UAL s'il s'agit d'une opération arythmétique ou logique ou par l'UC dans le cas d'opérations de branchement.

Exemple : Exécuter l'opération **`c = 2*a + b`**

- **Instruction `I1`** : Charger **a** dans le registre **`r1`**.
- **Instruction `I2`** : Calculer **`r1 = r1 * 2`**.
- **Instruction `I3`** : Charger **b** dans le registre **`r2`**.
- **Instruction `I4`** : Calculer **`r1 = r1 + r2`**.
- **Instruction `I5`** : Stocker **`r1`** dans **`c`**.




## 3.2 – Langage assembleur : jeux d'instruction

Un programme en langage machine est donc une suite très très longue de « 1 » et de « 0 ». Programmer en langage machine est donc extrêmement difficile. Pour pallier cette difficulté, les informaticiens ont remplacé les codes binaires par des **symboles mnémoniques** appelé « **jeux d'instructions** ».

L'opération « Additionner la valeur du registre **`%eax`** avec la valeur du registre **`%ebx`** » est codée en langage machine par la suite binaire **`0b0110000000000011 = 0xC003`**. Cette opération peut également être programmée en langage assembleur par **`addl %eax, %ebx`**.

| Additionner | | %eax | %ebx |
|---|---|---|---|
| C | 0 | 0 | 3 |
| Code opération | | Code opérandes | |

Que se soit en langage machine ou en langage assembleur, une instruction est toujours décomposée en deux champs : le champ **code opération** et le champ **code opérandes**.

| Additionner | %eax | %ebx |
|---|---|---|
| addl | %eax, %ebx | |
| Opération | Opérandes | |

Le langage assembleur est une représentation plus lisible que le langage machine. Le langage assembleur dépend du microprocesseur utilisé et de son jeu d'instructions.

## 3.3 – Jeux d'instruction Y86

Le langage y86 est un langage assembleur simplifié basé sur l'architecture des processeurs x86. Les données en mémoires sont organisées par blocs de 32 bits et utilise la convention **little endian** (stockage des octets en commençant par l'octet de poids faible).

Le y86 utilise 8 registres généraux de 32 bits, dont certains ont des rôles particuliers.

| Registre | Rôle | Identifiant |
|---|---|---|
| `%eax` | Accumulateur | 0 |
| `%ecx` | Registre de comptage utilisé dans les boucles | 1 |
| `%edx` | Registre auxiliaire | 2 |
| `%ebx` | Registre utilisé comme pointeur des tableaux (base index) | 3 |
| `%esp` | Registre qui pointe le sommet de la pile (stack pointer) | 4 |
| `%ebp` | Registre qui pointe la pile d'appel de la fonction courante (frame base pointer) | 5 |
| `%esi` | Registre utilisé lors des copies de suite d'octets (source index) | 6 |
| `%edi` | Registre utilisé lors des copies de suite d'octets (destination index) | 7 |

y86 propose un jeu d'instructions réduit :

- instructions arithmétiques : **`addl`**, **`subl`**, **`andl`**,
- déplacement de données : **`irmovl`**, **`rrmovl`**, **`rmmovl`**, **`mrmovl`**
- sauts conditionnels (ou pas) : **`jmp`**, **`je`**, **`jne`**, **`jg`**, **`jge`**, **`jl`**, **`jle`**,
- gestion pile : **`pushl`**, **`popl`**,
- appel/retour de fonction : **`call`**, **`ret`**,
- divers : **`nop`**, **`halt`**.

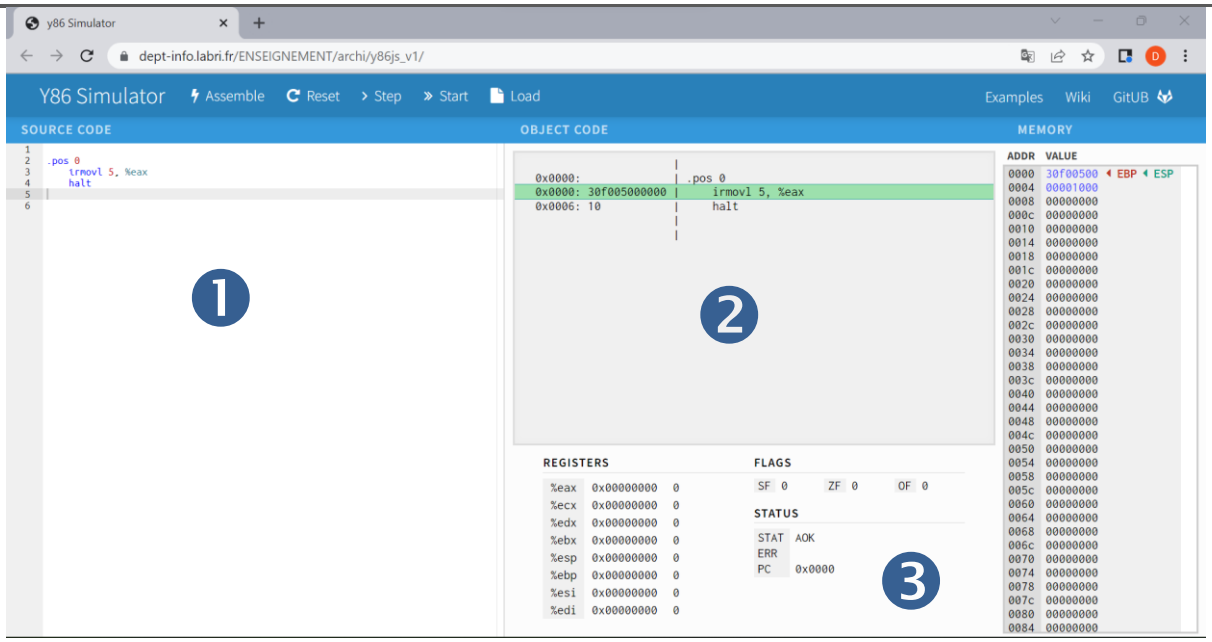
| N° octets | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| `nop` | 0 | 0 | | | | |
| `halt` | 1 | 0 | | | | |
| `rmovl rA, rB` | 2 | 0 | rA | rB | | |
| `Irmovl V, rB` | 3 | 0 | F | rB | V | |
| `rmovl rA, D(rB)` | 4 | 0 | rA | rB | D | |
| `mrmovl S(rB), rA` | 5 | 0 | rA | rB | S | |
| `addl rA, rB` | 6 | 0 | rA | rB | | |
| `subl rA, rB` | 6 | 1 | rA | rB | | |
| `andl rA, rB` | 6 | 2 | rA | rB | | |
| `xorl rA, rB` | 6 | 3 | rA | rB | | |
| `jxx D` | 7 | X | D | | | |
| `call D` | 8 | 0 | D | | | |
| `ret` | 9 | 0 | | | | |
| `pushl rA` | A | 0 | rA | 8 | | |
| `popl rA` | B | 0 | rA | 8 | | |

`V : Valeur - D : destination - S : Source`

# 4 – SIMULATEUR Y86

## 4.1 – Présentation du simulateur Y86

Ce simulateur permet de simuler le fonctionnement d'un microprocesseur et d'observer l'exécution de programmes en langage assembleur. Développé par le LABRI (LABoratoire de Recherche en Informatique) de l'Université de Bordeaux, il est disponible à https://dept-info.labri.fr/ENSEIGNEMENT/archi/y86js_v1/.




Simulateur Y86

Le programme en langage assembleur est edité dans la zone « Source Code » (❶). L'appui sur le bouton « Assemble » permet de transformer ce code en code machine exécutable par le processeur. Ce code machine est disponible dans la zone « Objet Code » (❷).

La zone « Memory » (❹) affiche le contenu de la mémoire et les zones « Register », « Flags » et « Status » (❸), le contenu des registres de données et du registre d'état.

L'exécution du programme est réalisée par l'appui sur le bouton « Start ». Cette exécution peut être réalisée instruction par instruction par l'appui sur le bouton « Step ». On pourra replacer le programme en situation initiale par appui sur le bouton « Reset ».

**Exercice n°2**

1. **Editez** le programme assembleur ci-contre.
2. **Assemblez** et **exécutez** le programme.
3. **Justifiez** le contenu du regsistre **%eax**.
4. **Justifiez** le contenu de la mémore à partir de l'adresse **0x0000**.

```
.pos 0
irmovl 5, %eax
halt
```

Quelques explications :

| | |
|---|---|
| `.pos 0` | Positionne le début du programme à l'adresse **0x0000**. |
| `irmovl 5, %eax` | Charge la valeur **5** dans le registre **%eax**. |
| `halt` | Arrêtre l'exécution du programme |

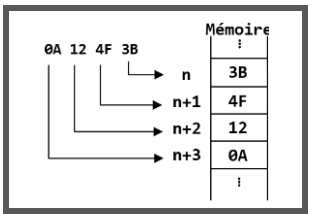
**Exercice n°3**

1. **Editez** le programme assembleur ci-contre.
2. **Vérifiez** la valeur des registres **%eax** et **%ecx** après l'exécution du programme. Dans quel resistre est enregistré le résultat ?
3. Que **remarquez** vous sur le stockage des données en mémoire ?
4. **Précisez** le rôle de l'instruction « `addl %eax, %ecx` ».

```
.pos 0
    irmovl 1000, %eax
    irmovl 500, %ecx
    addl %eax, %ecx
    halt
```

Les données sont stockées en mode **little-endian** c'est-à-dire que l'octet de poids faible est stocké à l'adresse mémoire la plus petite.

En mode **big-endian**, c'est l'octet de poids le plus fort qui est enregistré à l'adresse mémoire la plus petite.



Mode little-endian

## 4.2 – Allocation mémoire

Toutes les variables d'un programme ne peuvent pas être contenues dans des registres. La mémoire RAM est utilisée pour stocker les variables à des adresses disjointes du code binaire du programme. L'instruction suivante permet de stocker le contenu du registre **%eax** à l'adresse mémoire **100** : `rmmovl %eax, 100`

**Exercice n°4**

**Modifiez** le programme de l'exercice 2 afin de stocker le résultat de l'addition à l'adresse **100**.

Il est possible de déclarer l'emplacement des variables à l'aide d'étiquettes (ou labels). La syntaxe est la suivante : « `nom: .type valeur` ». Par exemple : **n** variable de type **long** intialisée à la valeur **10** : `n: .long 10`

**Exercice n°5**

1. **Editez** le programme assembleur ci-contre. **Assemblez** le.
2. **Indiquez** à quelle adresse a éte stockée la variable **x**.
3. **Exécutez** le programme. **Donnez** la valeur de **x** à la fin de l'exécution du programme.
4. **Donnez** le rôle l'instruction « `iaddl 1, %eax` ».
5. **Indiquez** quelle opération arithmétique est réalisée par le programme.

```
.pos 0
    mrmovl x, %eax
    iaddl 1, %eax
    rmmovl %eax, x
    halt
.pos 100
    x: .long 5
```

Le code du programme précédent est stocké à partir de l'adresse **0** jusqu'à jusqu'à l'adresse **18** (**0x12**). La variable **x** se trouve à l'adresse **100** (**0x64**). Les deux zones de mémorisation (programme et données) sont bien disjointes Dans le cas général, on veut des zones disjointes mais qui se suivent. La directive « `.align` » permet de laisser le compilateur déterminer automatiquement l'adresse de début d'allocation mémoire pour les variables. Suivie d'un chiffre, elle permet d'allouer des espaces de taille (en nombre d'octets) définis par le chiffre.

**Exercice n°6**

1. **Editez** le programme assembleur ci-contre. **Assemblez** et **exécutez** le programme.
3. **Justifiez** le valeur du chiffre associé à la directive « `.align` ».
4. A quelle adresse est stockée la variable **x** ?

```
.pos 0
    mrmovl x, %eax
    iaddl 1, %eax
    rmmovl %eax, x
    halt
.align 4
    x: .long 5
```

**Exercice n°7**

1. **Editez** le programme permettant de faire l'opération **z = x + y** avec **x = 8** et **y = 20**.
2. **Editez** le programme permettant de faire l'opération **x - y** avec **x = 8** et **y = 20** et qui stocke le résultat dans le registre **%eac**.
3. **Editez le programme** permettant de faire l'opération **x + y - z** avec **x = 8**, **y = 20** et **z = 10** et qui stocke le résultat dans **%edx** en utilisant des variables commençant à l'adresse **200**.

**Exercice n°8**

Soit le programme assembleur ci-contre

1. **Indiquez** quel sera le contenu des adresses mémoires **0x0030** et **0x0034** à la fin du programme.

```
.pos 0
    irmovl 20, %eax
    irmovl 0x20, %ecx
    rmmovl %eax, x
    rmmovl %ecx, y
    mrmovl x, %ecx
    mrmovl y, %eax
    halt
.pos 0x30
    x: .long 5
    y: .long 12
```

## 4.3 – Registre des drapeaux.

Après chaque opération arithmétique ou logique, le processeur met à jour des indicateurs (drapeaux ou Flags) dans un registre appelé registre des drapeaux. Parmi les différents drapeaux on trouve les 3 indicateurs (1 bit) suivants :

- **ZF** (**Z**ero **F**lag) : indique si le résultat de l'opération est nul (**ZF = 1**) ou non nul (**ZF = 0**).
- **SF** (**S**ign **F**lag) : indique si le résultat de l'opération est négatif (**SF = 1**) ou positif (**SF = 0**).
- **OF** (**O**verflow **F**lag) : indique si le résultat a induit un débordement (**OF = 1**) sur un entier non signé.

**Exercice n°9**

1. **Editez** un programme qui :
   - Charge la valeur **5** dans le registre **%eax**.
   - Effectue un OU-EXLUSIF entre le registre **%eax** et lui-même.
2. **Assemblez** et **exécutez** le programme.
3. **Justifiez** la valeur du drapeau **ZF**.

## 4.4 – Sauts et branchement

Ils permettent d'effectuer un branchement en fonction de l'état flags du registre des drapeaux.

| Saut | Description | Etats drapeaux |
|---|---|---|
| **jmp Dest** | Saut inconditionnel | |
| **jle Dest** | Saut quand inférieur ou égal (**A <= B**) | **OE != SF** ou **ZF = 1** |
| **jl Dest** | Saut quand inférieur (**A < B**) | **OE != SF** |
| **je Dest** | Saut quand égal (**A = B**) | **ZF = 1** |
| **jne Dest** | Saut quand différent (**A != B**) | **ZF = 0** |
| **jge Dest** | Saut quand supérieur ou égal (**A >= B**) | **OE = SF** |
| **jg Dest** | Saut quand supérieur (**A > B**) | **OE = SF** et **ZF = 0** |

**Exercice n°10**

1. **Editez** le programme ci-contre. **Assemblez** et **exécutez le** en mode pas à pas.
2. **Quel** drapeau déclenche l'arrêt du programme.
3. **Précisez** ce que fait le programme.

```
.pos 0
    mrmovl N, %eax
boucle:
    isubl 1, %eax
    jne boucle
    halt
.align 4
N: .long 5
```

## 4.5 – Boucles

**Exercice n°11**

On veut réaliser le programme ci-contre.

```
x = 3
tant que x > 0
    x = x - 1
Fin tant que
```

1. **Editez** le programme en remplaçant les commentaires par les instructions adéquates.

```
.pos 0
TantQue:
    #stocker dans %eax la valeur de x
    #effectuer une opération pour tester si x>0
    #saut vers "FinTantQue"si x< ou égal à 0
    #stocker dans %eax la valeur %eax-1
    #stocker la valeur de %eax dans x
    #reboucler vers "TantQue"
FinTantQue
    halt
.pos 100
    x: .long 3
```

**Exercice n°12**

Soit le programme ci-contre.

1. **Précisez** ce que fait le programme.
2. **Proposez** son équivalent en python.

```
.pos 0
    mrmovl x, %eax

si:
    andl %eax, %eax
    jle sinon
alors:
    isubl 1, %eax
    jmp fin_si
sinon:
    iaddl 1, %eax
fin_si:
    rmmovl %eax, x
    halt
.align 4
x: .long -5
```