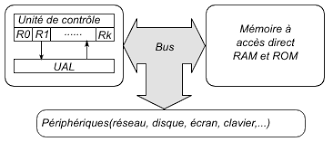
**Modèle d'architecture séquentielle (von Neumann)**



**SYNTHÈSE**

**Mots-clés**

Registre, unité de contrôle, unité arithmétique et logique, microprocesseur, langage machine, langage d’assemblage (ou assembleur), instruction, opérande, opcode.

**Notions**

* Architecture de von Neumann.
* Organisation de la mémoire.

**Compétences**

Dérouler l’exécution d’une séquence d’instructions simples du type langage machine.

**Table des matières**

[1. Introduction 2](#_Toc182581849)

[2. Le modèle de von Neumann 3](#_Toc182581850)

[2.1. Organisation 3](#_Toc182581851)

[2.2. Rôle des composants 3](#_Toc182581852)

[3. Langage machine 4](#_Toc182581853)

[3.1 Instruction machine 4](#_Toc182581854)

[3.2 Le langage d'assemblage ou assembleur 4](#_Toc182581855)

[Annexe - Cycle d’un programme 5](#_Toc182581856)

**Table des illustrations**

[**Figure 1** : Code machine 6800 2](#_Toc182582259)

[**Figure 2** : Ecran du simulateur SDK6800/11 2](#_Toc182582260)

[**Figure 3** : Modèle de von Neumann 3](#_Toc182582261)

[**Figure 4** : Organisation pour une puce de 108 transistors 3](#_Toc182582262)

[**Figure 5** : Extrait d’un programme écrit en assembleur 6802 4](#_Toc182582263)

**Sources**

**Bibliographie**

Architecture de l’ordinateur E Lazard Pearson 2006

NSI 1er Bordas 2021

NSI 1er Hachette 2021

**Webographie**

Assembleur 6800 <https://bit.ly/2T406t5>

MC6800 Datasheet <https://bit.ly/3CXNAjZ>

# Introduction

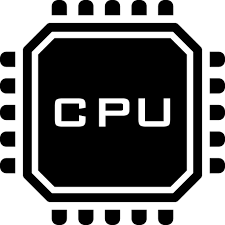
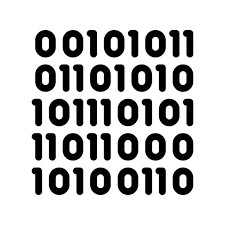
|  |  |  |  |
| --- | --- | --- | --- |
| **Premier calculateur**  Les programmes de l'[***EDSAC***](https://fr.wikipedia.org/wiki/Electronic_Delay_Storage_Automatic_Calculator) (**1949**), premier [calculateur à programmes enregistrés](https://fr.wikipedia.org/wiki/Ordinateur_%C3%A0_programme_enregistr%C3%A9) fonctionnant selon le modèle de **von Neumann**, étaient rédigés en utilisant des mnémoniques alphabétiques d'une lettre pour chaque instruction. La traduction était alors faite à la main par les programmeurs, une opération longue, fastidieuse et entachée d'erreurs. [Wikipédia](https://en.wikipedia.org/wiki/EDSAC) | | | [*EDSAC*](https://fr.wikipedia.org/wiki/Electronic_Delay_Storage_Automatic_Calculator) |
| [*Micral N*](https://fr.wikipedia.org/wiki/Micral) | **Premier micro-ordinateur**  En **1973**, François Gernelle invente le **premier micro-ordinateur** au monde : le [**Micral N**](https://fr.wikipedia.org/wiki/Micral)**.** Autonome et destiné à fonctionner sur le terrain pour réaliser des calculs, il pesait 8kg, ne disposait ni de clavier, ni d’écran, mais innovait en utilisait un microprocesseur.  La **console système** ci-contre (munie d’**interrupteurs** et de **LED**) permettait d’afficher et de modifier directement le contenu de la mémoire **en code machine**, c’est-à-dire que les **codes** **numériques** correspondant aux **instructions** étaient entrés **un par un** en mémoire par l'utilisateur à l'aide d'interrupteurs. | | | |
| **Code machine**  ***« Le langage machine, ou code machine, est la suite de bits qui est interprétée par le processeur*** *d’un ordinateur exécutant un programme informatique****. »***  Exemple : ci-contre, un programme partiel écrit en **code machine 6800** affichant le texte "*Hello World"*. 6800 indique que le code est destiné au microprocesseur 6800 de Motorola.  Pour simplifier la lecture du programme, on écrit en hexadécimal les codes et l’adresse de leur position en mémoire.  Exemple : le même programme représenté en hexadécimal    **Figure 1** : Code machine 6800 | | **Mémoire**  Adresses2  : Données2  0000000000000000 : 11001110  0000000000000001 : 00000000  0000000000000010 : 00011101  0000000000000011 : 11111111  | : |  0000000000101001 : 00000000  0000000000101010 : 00000000  0000000000101011 : 11111011  0000000000101100 : 00000000  | : | | | | |
| Très rapidement (**année 50**), la programmation des calculateurs a été réalisée en **langage d’assemblage** (assembleur).  **Assembleur**  *« L’****assembleur*** *est le* ***langage de plus bas niveau*** *qui représente le langage machine sous une forme* ***lisible par un humain****, dans laquelle des expressions symboliques, appelées* ***mnémoniques****, remplacent les codes numériques. »*  Exemple : *Hello World* codé en **assembleur 6800** et *code machine* correspondant.    **Figure 2** : Ecran du simulateur SDK6800/11  Adresse Opcode Opérande(s) ; Hello World  0000 *CE 00 1D*  ldx #msg  0003 *FF 00 29* stx pstring  0006 *FE 00 29* loop ldx pstring  0009 *A6 00* ldaa 0,x  000B *27 0F* beq fol  000D *08* inx  000E *FF 00 29* stx pstring  0011 *FE 00 2B* ldx pscreen  0014 *A7 00* staa 0,x  0016 *08* inx  0017 *FF 00 2B*  stx pscreen  001A *20 EA* bra loop  001C *01* fol nop  001D *48 65 6C 6C 6F 20 57 6F 72 6C 64 00* msg .str "Hello World"  0029 *00 00* pstring .word 0  002B *FB 00* pscreen .word $FB00  .end | | | | |

Le **code source** (écrit par le programmeur) n’étant pas destiné à être exécuté par le processeur, un programme de **traduction** automatique (l’**assembleur**) est nécessaire.

****



**Assembleur**



**Code source**

**Code machine**

**Traducteur**

Bien que plus facile à manipuler que les "codes machines", l'assembleur est **fastidieux à écrire,** car comme on le voit ci-dessus il faut aligner un grand nombre d'instructions pour obtenir un résultat, même simple. De plus, il **ne s'adresse qu'à un seul modèle de processeur**. Tout changement de machine nécessite une **réécriture** plus ou moins complète du **code**.

Pour pallier ces défauts, des **langages évolués** comme le **C**, le **PHP** ou le **Python** ont été développés. Ils permettent au programmeur de se concentrer sur l'algorithmique des applications. Comme les instructions ne sont plus compréhensibles par l'ordinateur, une phase de traduction est nécessaire. C'est le rôle des **interpréteurs** et des **compilateurs**. (**Annexe 1**)

Aujourd'hui, l'assembleur reste utilisé pour écrire des parties des **systèmes d'exploitation**, **gestionnaires de périphériques,** etc.

# Le modèle de von Neumann



J. von Neumann

Munie d’un **processeur**, de dispositifs **d’entrée/sortie**, d’une **mémoire** et de **bus**, l’architecture de von Neumann (1945) révolutionne la conception des ordinateurs, car elle permet d’enregistrer des données au cours de l’exécution d’un programme exécuté dans sa **mémoire** physique. Ce modèle est révolutionnaire à l’époque, car il permet de stocker les données et les programmes dans la même mémoire.

## Organisation

|  |
| --- |
| **Processeur (CPU)**    **Figure 3** : Modèle de von Neumann |

## Rôle des composants

* **Le CPU (Central Processing Unit)**

Le CPU (processeur ou Unité centrale de traitement) se charge des **calculs mathématiques** et des **transferts de données**. Son organisation interne (simplifiée) est représentée ci-dessous.

|  |  |
| --- | --- |
| **Figure 4** : Organisation pour une puce de 108 transistors | * **L'unité de contrôle**   Elle **récupère en mémoire la prochaine instruction à exécuter** et les données sur lesquelles elle doit opérer puis les envoie à l'unité arithmétique et logique.    Registres du 6800   * **L'unité arithmétique et logique (ALU)**   Effectue les opérations arithmétiques (addition, soustraction, etc.) et logiques (Et, OU, décalage, etc.)   * **Les registres**   Les registres sont des mémoires internes très rapides. Ils sont nombreux dans les processeurs modernes. Certains ont un rôle particulier.  - Le registre d'instruction (**IR**) contient l'instruction courante à décoder et à exécuter.  - Le compteur programme (**PC**) indique l'emplacement mémoire de la prochaine instruction à exécuter,  - le ou les **accumulateurs** (sur les anciens microprocesseurs) sont utilisés pour les calculs.  - le registre d'état (**SR**) indique si une opération à renvoyé zéro, un résultat négatif, etc.  - le pointeur de pile (**SP**) indique une zone de la mémoire utilisée par les sous-programmes. |

|  |  |
| --- | --- |
| FFFF  F001 | Mémoire non volatile |
| F000  0000 | Sommet de la pile ↓  ↑ Début du prog. |

**Espace mémoire de**

**216 = 65536 octets du 6800**

* **La mémoire**

La mémoire contient à la fois **les programmes** et les **données**. On distingue habituellement deux types de mémoires :

- la **mémoire vive** ou volatile perd son contenu dès que l'ordinateur est éteint (ex. : **RAM** pour Random-access Memory);

- la mémoire **non volatile** est celle qui conserve ses données quand on coupe l'alimentation (ex. : **ROM** pour Read Only Memory).

La mémoire d'un ordinateur peut être vue comme un **tableau de cases mémoires** élémentaires, appelées mot mémoire. Selon les ordinateurs, la taille de ces mots peut varier de 8 à 64bits. Chaque case possède une **adresse unique** à laquelle on se réfère pour accéder à son contenu (en écriture ou en lecture). Traditionnellement, ce tableau est représenté verticalement comme ci-contre.

|  |  |
| --- | --- |
| Avertissement | La programmation d'un microprocesseur en **langage machine ou en langage d’assemblage (assembleur)** nécessite de connaître l'organisation de l'espace mémoire qu'il peut adresser. Cet espace est constitué de mémoire volatile et non volatile. |

* **Les entrées-sorties**

Elles permettent la communication avec le processeur pour lui fournir des données (clavier, souris, etc.) et afficher ou transmettre les résultats des calculs (écran, carte réseau, etc.).

# Langage machine

## 3.1 Instruction machine



Motorola [MC6800L](https://fr.wikipedia.org/wiki/Motorola_6800) (1975)

L'instruction-machine est une **chaîne binaire de p bits** composée de **deux parties** :

- le champ **code opération (opcode),** composé de **m bits,** indique au processeur le type de traitement à réaliser (addition, lecture d'une position mémoire, etc.). Le nombre d'opérations différentes autorisées pour un processeur (**2m**) définit le **jeu d'instructions** du processeur.

- le champ **opérande(s),** composé de **p – m bits,** permet d'indiquer la nature des données sur lesquelles le code opération doit s'appliquer. La façon de désigner un opérande dans une instruction peut prendre différentes formes : on parle alors de **mode d'adressage** des opérandes.

Exemple : dans l'exemple ci-dessus **8616** est un **opcode** du jeu d'instructions du processeur 6800. **0C16** est un **opérande**.

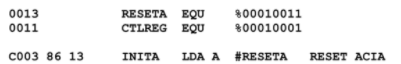
|  |  |
| --- | --- |
| Opcode | Opérande(s) |
| 86 | 0C |

Les instructions du langage machine sont rangées dans **quatre catégories.** Les exemples sont donnés en assembleur 6800.

* les instructions **arithmétiques et logiques.** Exemples : ADD, SUB, AND, OR etc.
* les instructions **de transfert de données.** Exemples : LDA, STA, TAB, etc.)
* les instructions **d'entrées-sorties**
* les instructions **de rupture de séquence d'exécution** (saut conditionnel et inconditionnel). Exemples : BNE, BRA, JMP, etc.

<https://webge.fr/6800.html>

## 3.2 Le langage d'assemblage ou assembleur



**Figure 5** : Extrait d’un programme écrit en assembleur 6802

Les notions sur le langage d'assemblage sont illustrées avec l'assembleur du microprocesseur 68xx.

**3.2.1 Instruction**

Une instruction du langage d'assemblage est l'équivalent du langage machine dans lequel chaque champ de l'instruction-machine est remplacé par un **mnémonique** alphanumérique (**ldaa**, **adda**, jmp, etc.). L'instruction estalors composée de **champs, séparés par un ou plusieurs espaces.** On identifie un champ étiquette (label), un champ opération (opcode), un champ opérande (operand) éventuellement complété par un champ commentaire (comment).

**[label] [opcode][operand] [comment]**

; This is a comment

Exemple : loop1 adda #3 ; Add value 3 to Accumulator A

jmp loop1 ; Jump to address ‘label’

.end ; end of program

**3.2.2 Directive d'assemblage**

Une directive d'assemblage n'est pas une instruction-machine, mais un ordre destiné à l'assembleur. Elle **simplifie l'écriture et la lisibilité du code** en assembleur grâce à la déclaration de constantes, de variables, etc.

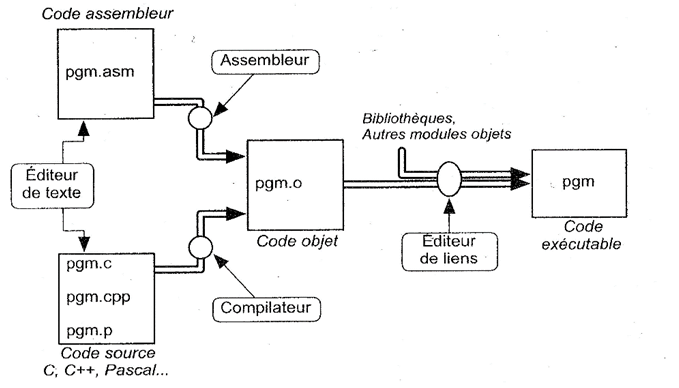
Exemple : .org ; cette directive fixe le début du code machine à une position en mémoire

# Annexe - Cycle d’un programme

**Compilateur** : un **compilateur** est un programme qui traduit le code source écrit dans un langage de programmation de haut niveau (comme C++) en un langage machine ou un code binaire que l'ordinateur peut comprendre et exécuter. [*Wikipédia*](https://fr.wikipedia.org/wiki/Interpr%C3%A8te_(informatique))

**Interpréteur** : un **interpréteur** est un programme qui exécute directement les instructions écrites dans un langage de programmation de haut niveau (PHP), sans les compiler en code machine. [*Wikipédia*](https://fr.wikipedia.org/wiki/Interpr%C3%A8te_(informatique))

* **Cycle d’un programme assemblé ou compilé**

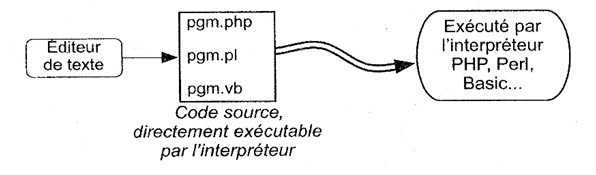
****



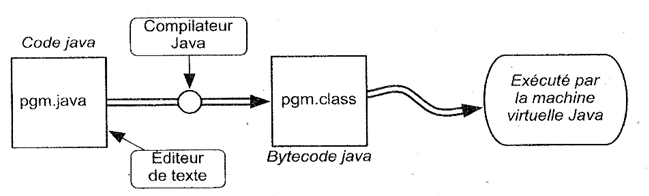
**Exemple de code source 6800 assemblé en code exécutable**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Exécutable*** |  | ***Code source*** |  |
| *Adresse16* | ***opcode16*** *et opérande16* |  | *; label opcode operande(s)* | *comment* |
|  |  |  | *.org $0* | *; Programme* |
| *0000* | **B6** 00 20 |  | Debut **ldaa** v1 | ; [A]<-[v1] |
| *0003* | **BB** 00 21 |  | **adda** v2 | ; [A]<-[A]+[v2] |
| *0006* | **B7** 00 22 |  | **staa** v3 | ; [v3]<-[A] |
| *0009* | **01** |  | fin nop |  |
|  |  |  |  |  |
|  |  |  | *.org $20* | *; Données* |
| *0020* | **0A** |  | v1 .byte 10 |  |
| *0021* | **14** |  | v2 .byte 20 |  |
| *0022* | **0** |  | v3 .byte 0 |  |

* **Cycle d’un programme interprété**

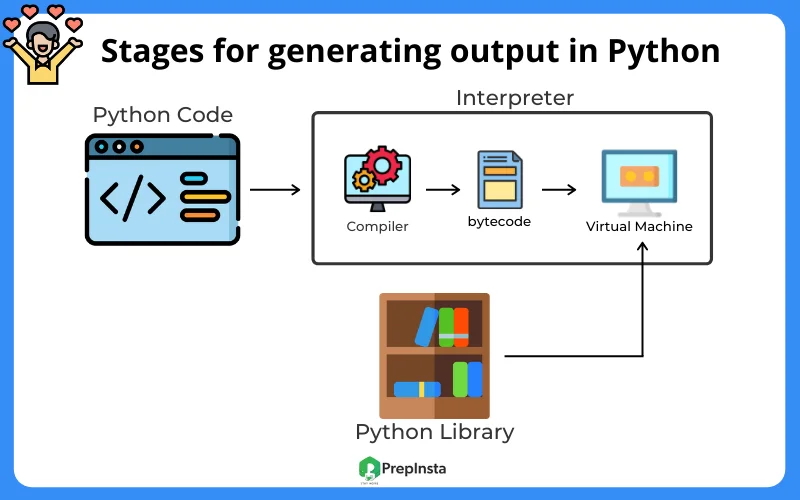
****

* **Cycle d’un programme Java (exécuté sur une machine virtuelle)**

****

* **Cycle d’un programme Python (exécuté sur une machine virtuelle)**



****

* **Un code** dit "**ByteCode**" est produit pour s’exécuter sur une machine virtuelle (Annexe1)

|  |  |
| --- | --- |
| **Informations** | L’affichage du **bytecode** d'une fonction, d’un programme, etc. se fait avec la fonction ***dis()***  Pour avoir **l'adresse d'une variable** ou d'une fonction on utilise ***id()*** en console |

* **Exemple de code source CPython compilé en Bytecode puis interprété par une machine virtuelle**

Ressource : <https://pymotw.com/3/dis/>

**Code source Bytecode**

import dis

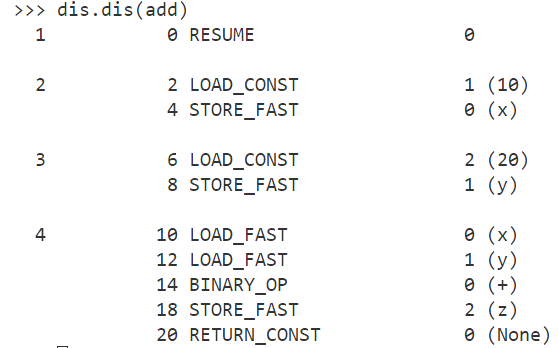
def add():

    x = 10

    y = 20

    z = x+y

dis.dis(add)



* **Adresse** (en décimal, binaire et hexadécimal) et **type d**’une **variable**

|  |  |
| --- | --- |
|  | Explications  a. **2** : Le premier chiffre représente le numéro de ligne dans le code source Python d'origine.  b. **2** : Le deuxième chiffre est l'adresse de l'instruction dans le bytecode.  c. **LOAD\_CONST** : C'est l'opcode, qui indique l'opération à effectuer. Ici, LOAD\_CONST charge une constante sur la pile.  **d.** 1 : C'est l'argument de l'opcode, ici un index dans la table des constantes.  **e.** (10) : valeur chargée, ici 10. |