|  |  |  |
| --- | --- | --- |
| **Logo_Lycée** | **Langage machine et assembleur** | **logo%20ac%20orl%E9ans%20toursDescription : Description : Description : pemDescription : Description : Description : pem** |
| **TP**  **Approfondissement** | **Préparation du TP « Assembleur 68xx »** |

*Prof*

**Rappel de l’objectif**

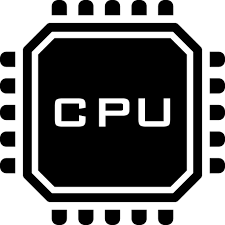
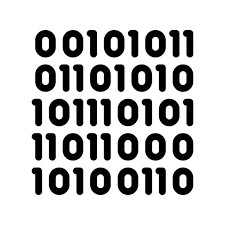
***Dérouler l’exécution d’une séquence d’instructions simples du type langage machine.***

**Utilisation du simulateur**

Pour atteindre cet objectif, vous disposez d’un simulateur. Vous écrivez les instructions du programme en **langage d’assemblage**. L’assembleur intégré au simulateur les convertit en **codes machine** et les place dans la mémoire. L’exécution du programme se fait en mode **pas à pas** (c.-à-d. instruction par instruction) afin d’identifier le contenu des **registres** du **CPU**.



**Logiciel assembleur**



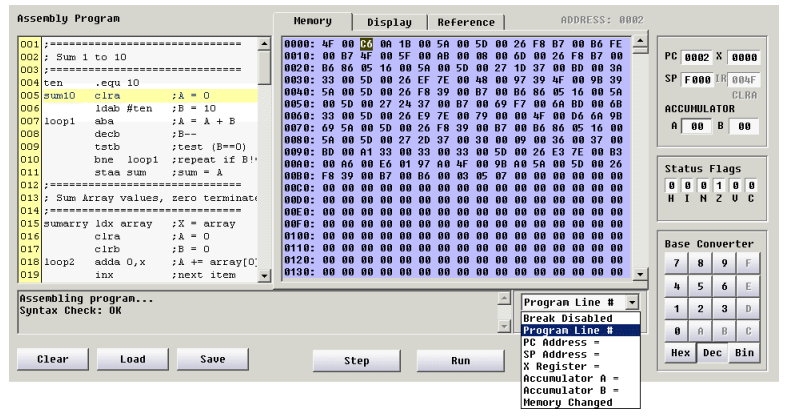
**Code source en langage d’assemblage**

**Code machine**

**en mémoire**

**Interprète**





Registres

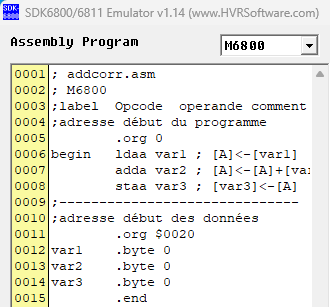
6800

**Premier programme du TP**

**Opération attendue** : **var3 ← var1 + var2** # Les variables var3, var2 et var1 sont des octets, 0 ≤ varx ≤ 255 x ∈ [1,3]

Le programme est réalisable avec **3 instructions** :

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction n°** | **Description** | **Instruction assembleur 6800** | **Opération symbolique** |
| **1** | **Charger** dans l’accumulateur **A** la valeur (**opérande**) située à la position mémoire **var1** | **ldaa** *var1* | **[A] ← [***var1***]** |
| **2** | **Ajouter** le contenu de l’accumulateur **A** à la valeur (**opérande**) située à la position mémoire **var2** (le résultat de l’opération est automatiquement placé dans le registre A) | **adda** *var2* | **[A] ← [A] + [***var2***]** |
| **3** | **Stocker** le contenu de l’accumulateur **A** à la position mémoire **var3** | **staa** *var3* | **[***var3***]** **← [A]** |



Lecture d’une **opération symbolique**

**[]** ⬄ contenu de

Ex : [var1] : contenu de la mémoire à l’adresse var1

**←**: placer dans

Ex : [A] ←[var1] : placer le contenu de la mémoire à l’adresse var1 dans le registre A

**Python vs assembleur 6800**

**Opération attendue** : **var3 ← var1 + var2** # Les variables var3, var2 et var1 sont des octets, 0 ≤ varx ≤ 255

|  |  |
| --- | --- |
| **Python** | **Assembleur MC6800** |
| *# Organisation du programme source*  *# Indentation avec 4 espaces pour identifier les blocs de code*  **# Données**  var1 = 10  *Déclaration et initialisation des*  var2 = 20 *variables*  ***type ?***  **# Traitement**  var3 = var1 + var2  Choisi par **type(x)**  Python en console pour  le connaître  Ex : int  *Position du programme*  *en mémoire ? non précisée dans le fichier source*  **Python gère les types des variables et la mémoire à l’aide du système d’exploitation** | *; Organisation du programme source*  *; 4 champs séparés par au moins un espace*  *;* ***label******opcode******opérande******commentaires***  .**org $0000** ; position du code en mémoire  begin ldaa var1  *adda var2 ;* ***Traitement***  *staa var3*  **.org $0020** ; position des données (variables) en mémoire  var1 .byte 10 ; déclaration et initialisation  *var2 .byte 20*  *var3 .byte 0*  *.end type*  **Le programmeur doit gérer les types et la mémoire** |

**Vocabulaire (point de vue matériel)**

* **Déclarer une variable** : la définition (ou déclaration) d'une variable permet de **réserver un emplacement mémoire** pour la variable.
* **Initialiser une variable** : l'initialisation consiste à donner la valeur de la variable au moment même où on la déclare.
* **Type d’une variable** : place occupée par une variable en mémoire (dépend du langage de programmation et de la machine).
* Les **directives** d’**assemblage** sont des pseudo-instructions. Elles ne correspondent à aucune instruction-machine ; ce sont des **ordres destinés à l’assembleur** (**Ex** : .org, .byte et .end ci-dessus).
* **Byte** ou **octet** est composé de **8bits**

**En assembleur, le code source est "compilé" en code machine** (illustré en annexe)



**Code machine Code source**

**Adresse opcode opérande ; label opcode operande comment**

.org $0000

**0000 B6 00 20** begin **ldaa** var1; [A]<-[var1]

**0003 BB 00 21 adda** var2; [A]<-[A]+[var2]

**0006 B7 00 22 staa** var3; [var3]<-[A]

.org $0020

**0020 10** var1 .byte 10

**0021 20** var2 .byte 20

**0022 0** var3 .byte 0

**En CPython, le code est interprété**

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

* **Un code** (**ByteCode**) est produit pour s’exécuter sur une machine virtuelle (illustré sur l’annexe)



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

**Code source Bytecode**

import dis

def main():

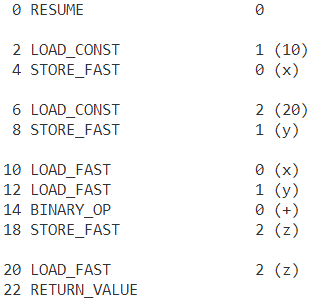
    x = 10

    y = 20

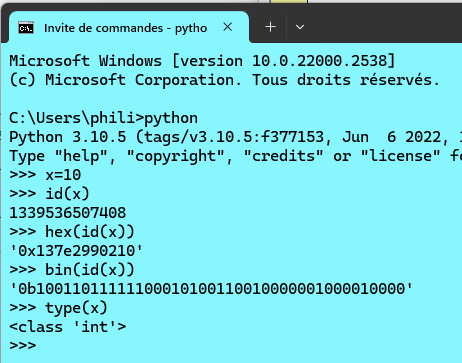
    z = x+y

if \_\_name\_\_ == "\_\_main\_\_":

**dis.dis(main**)



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



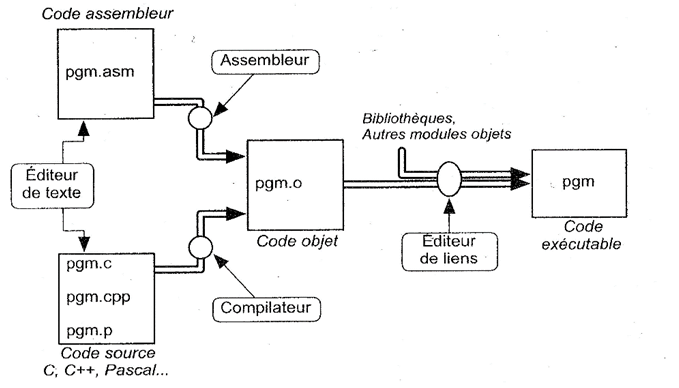
# 

# Annexe - Cycle d’un programme

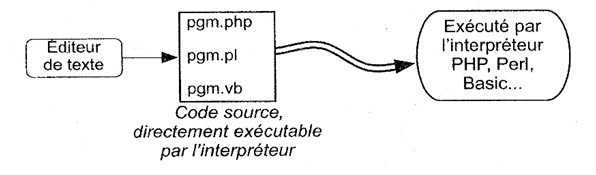
***Compilateur*** *: « en informatique, un* ***compilateur*** *est un programme qui transforme un code source en un code objet. Généralement, le code source est écrit dans un langage de programmation, il est de haut niveau d'abstraction, et facilement compréhensible par l'humain ».* [*Wikipédia*](https://fr.wikipedia.org/wiki/Compilateur)

***Interpréteur*** *: « en*[*informatique*](https://fr.wikipedia.org/wiki/Informatique)*, un* ***interpréteur*** *est un outil ayant pour tâche d'analyser, de traduire et d'exécuter les programmes écrits dans un*[*langage informatique*](https://fr.wikipedia.org/wiki/Langage_informatique)*. L'interprétation repose sur l'exécution (l'interprète), plutôt que sur sa conversion en un autre langage (par exemple le langage machine) ; elle évite la séparation du temps de conversion et du temps d'exécution, qui sont simultanés ».* [*Wikipédia*](https://fr.wikipedia.org/wiki/Interpr%C3%A8te_(informatique))

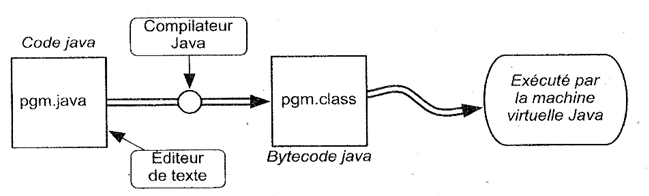
**Cycle d’un programme compilé**

****

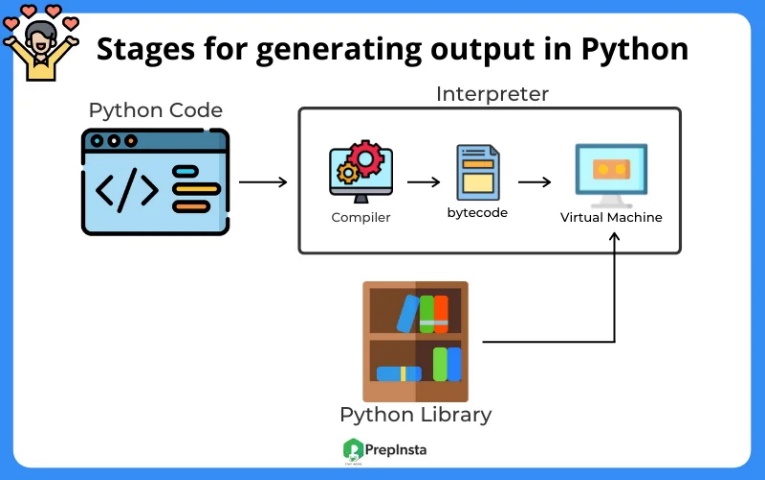
**Cycle d’un programme interprété**

****

**Cycle d’un programme Java**

****

**Cycle d’un programme Python**

****