TP N° 3

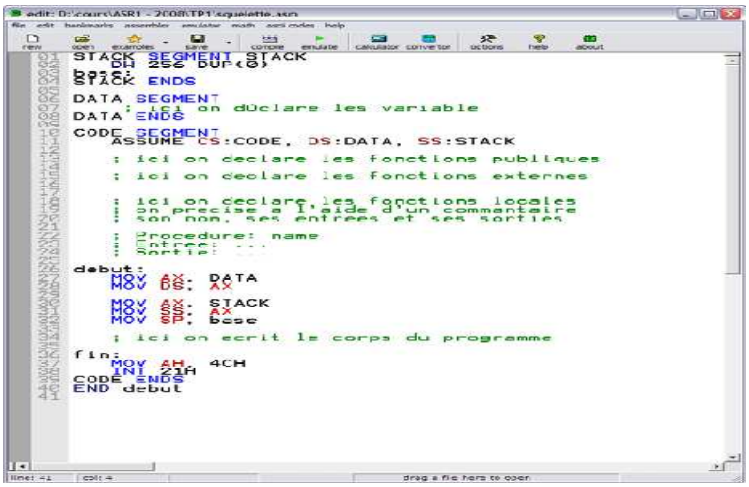
**Programmation en assembleur sous emu8086**

## 1. Objectifs

- Prise en main d'emu8086.
- Ecriture d'un premier programme en assembleur 8086

## 2. Emu86

Pour simplifier l'apprentissage de l'assembleur x86, nous travaillerons sur un émulateur logiciel qui simule un processeur 8086. L'utilisation de cette émulation n'est pas obligatoire car nos programmes, écrits en assembleur 8086, fonctionne normalement sur un processeur moderne comme l'Intel core i7. Cependant, cet émulateur nous fourni, en plus, un environnement avancé pour l'apprentissage des mécanismes et le fonctionnement du processeur. Ainsi, Emu8086 se présente dans un premier temps comme un éditeur de texte classique, avec une colorisation du code assembleur (mode éditeur).




**Structure d'un programme**

```
org 100h
.DATA
  <Déclaration des variables >
.CODE

        Instr 1
        Instr 2
        …
        Instr n
```

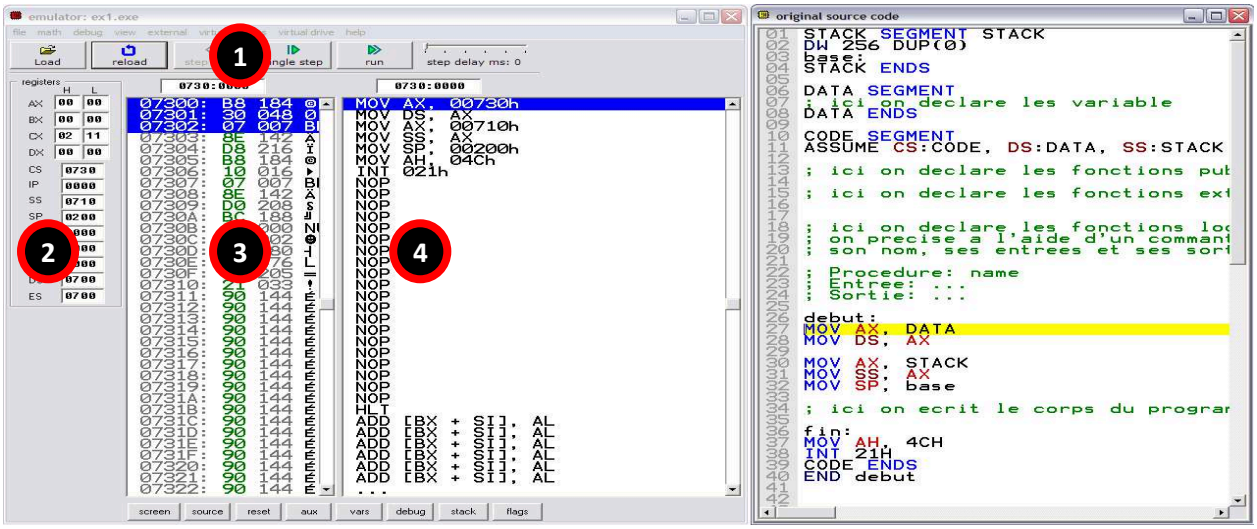
**Exercice 1 :**

*Soit ci-dessous le programme qui calcule la somme des deux variables a et b et enregistre le résultat dans c.*

```
org 100h
.DATA
        a dw 25
        b dw 13
        c dw ?
.CODE
        mov ax, a
        add ax, b
        mov c, ax
        mov ax, 4ch
        int 21h
```

***A.*** *Taper, compiler et exécuter le programme (bouton RUN).*

Deux nouvelles fenêtres viennent de s'ouvrir, il s'agit du mode exécution de l'émulateur. La fenêtre « *original source code* » contient le code tel que vous l'avez écrit. La ligne surlignée en jaune est la prochaine instruction qui va être exécutée.




La fenêtre « *emulator* : » contient 4 zones, schématisées par les 4 cercles numérotés, détaillées ci-dessous :

1. Les boutons de la zone une vont nous permettre d'exécuter notre programme soit *Pas à Pas (single step)*, de revenir en arrière (reload), soit automatiquement (run) avec réglage du délai d'attente entre chaque instruction.
2. La deuxième zone contient une représentation des registres, et permet de visualiser leur valeur hexadécimale tout au long de l'exécution de votre programme.
3. La zone 3 représente la mémoire (octet par octet). Sur une ligne, on retrouve l'adresse mémoire sur 20 bits, la valeur de l'octet en hexadécimale, en décimale et sa représentation en ASCII.
4. La 4eme zone contient le code de votre programme une fois les traductions d'adresses terminées. Il n'y a plus aucune étiquette ni nom de variable. Il s'agit du code réellement exécuté.

**B.** *Exécuter le même programme pas à pas (single step) observer le contenu des registres AX, IP et les valeurs de a, b, c.*

**Exercice 2:**

- Ecrire un programme en assembleur pour chacune des opérations suivantes :
  1- s= a + b + c
  2- m = a * b
  3- d = a div b