

## Semaine 7 – Langage machine – Exercices

### Exercice n°7.1 – Programme à réaliser en NASM 32 bits

En utilisant l'IDE SASM, écrivez un programme en NASM qui demande à l'utilisateur d'entrer un nombre entier en décimal non signé jusqu'à ce qu'il trouve le nombre mystère.

Le nombre mystère est codé en mémoire en dur dans le programme, initialisé en hexadécimal dans la section `.data`.

Travaillez avec des nombres qui ont une taille de 1 octet.

```
Entre un nombre 128
Trop grand, essaie encore
Entre un nombre 64
Trop petit, essaie encore
Entre un nombre 96
Trop grand, essaie encore
Entre un nombre 74
Trop petit, essaie encore
Entre un nombre 85
Trop grand, essaie encore
Entre un nombre 80
Trop grand, essaie encore
Entre un nombre 76
Bravo, le nombre est bien 76
Nombre d'essai(s) : 7
```

### Exercice n°7.2 – Programme à réaliser en NASM 32 bits

En utilisant l'IDE SASM, écrivez un programme en NASM qui demande à l'utilisateur d'entrer un nombre entier en décimal non signé, un second nombre entier en décimal non signé, et ensuite un opérateur (+ ou -).

Travaillez avec des nombres qui ont une taille de *double word* (4 octets).

L'utilisateur doit entrer un opérateur (+ ou -) correctement, vous devez lui redemander d'entrer un opérateur (+ ou -) jusqu'à ce que ce soit correct.

```
Entre un nombre : 7
Entre un second nombre : 9
+ ou - ? *
+ ou - ? /
+ ou - ? +
Le resultat est = 16
```

Votre programme affiche ensuite le résultat de l'addition ou de la soustraction des deux nombres.

```
Entre un nombre : 23
Entre un second nombre : 5
+ ou - ? -
Le resultat est = 18
```

## Exercice n°7.3 – Programme à réaliser en NASM 32 bits

En utilisant l'IDE SASM, écrivez un programme en NASM qui demande à l'utilisateur d'entrer 10 nombres entiers en décimal non signé.

Vous devez stocker ces 10 nombres en mémoire dans un tableau de *double word*. Utiliser la directive `resd` dans la section `.bss` pour ce faire.

Par exemple,

avec les *inputs* ci-dessous,

vous devez pouvoir visualiser en mémoire le tableau en mode *Hex* ou *Smart* :

| Input |  |  |  |
|-------|--|--|--|
| 123   |  |  |  |
| 20    |  |  |  |
| 45    |  |  |  |
| 5486  |  |  |  |
| 218   |  |  |  |
| 4     |  |  |  |
| 728   |  |  |  |
| 649   |  |  |  |
| 40000 |  |  |  |
| 1245  |  |  |  |

  

| Memory                 |   |      |      |
|------------------------|---|------|------|
| Variable or expression | Value   | Type |      |
| tableau                | {0x7b,0x14,0x2d,0x156e,0xda,0x4,0x2d8,0x289,0x9c40,0x4dd} | Hex  | d 10 |

  

| Memory                 |   |       |      |
|------------------------|---|-------|------|
| Variable or expression | Value                                     | Type  |      |
| tableau                | {123,20,45,5486,218,4,728,649,40000,1245} | Smart | d 10 |

L'exécution en fenêtre de commandes Windows donne, selon le même exemple que dans SASM, ceci :

```
Entre le nombre 1
123
Entre le nombre 2
20
Entre le nombre 3
45
Entre le nombre 4
5486
Entre le nombre 5
218
Entre le nombre 6
4
Entre le nombre 7
728
Entre le nombre 8
649
Entre le nombre 9
40000
Entre le nombre 10
1245
```

Ensuite, en parcourant ce tableau en mémoire, trouvez le nombre le plus grand et le plus petit et affichez-les à l'écran.

L'exemple précité donne dans SASM :

Le plus petit est 4

Le plus grand est 40000