TP Pointeurs en assembleur

On rappelle qu'un tableau, en assembleur (ainsi qu'en C), est simplement un pointeur vers un ensemble de données. Il appartient à l'utilisateur de savoir à quel endroit le tableau s'arrête, et quelle est la taille des éléments du tableau.

On a en fait déjà vu des implémentations de tableau en TP. Par exemple, dans le code

```
section .data
text: db 'Hello world!', 10
size: equ $ - text
```

On a déclaré un tableau (`text`) dont les éléments sont des octets (`db`) contenant respectivement le code ASCII de `H`, puis le code ASCII de `e`, etc. Nous savons de plus que le tableau fait `size` octets de long.

On peut également déclarer un tableau de façon dynamique : `table: times 100 db 'A'` initialise 100 octets (`db`) avec le code ASCII de `A`. Un tableau peut également être déclaré dans la section `.bss` du code :

```
section .bss
table: times 100 resb 1

; ou, de façon équivalente et plus rapide :
table2: resb 100
```

Cela dit, un tableau déclaré de telle manière ne peut pas être initialisé comme on le faisait dans la section `.data`.

Dans tous les cas, on peut accéder aux éléments du tableau en jouant avec le pointeur `table`. Attention, les adresses seront calculées avec la taille du plus petit registre utilisé. Pensez donc bien à utliser des variables et registres de 32 bits.

Exercice 1 –

1. Codez un programme qui affiche toutes les lettres d'une chaîne de caractères en majuscules. On considère pour cette question que la chaîne de caractères ne contient que des lettres minuscules (pas de ponctuation, d'accentuation ni d'espace).

2. Adaptez le code suivant pour prendre en charge les chaînes de caractères qui comportent des majuscules et des espaces.

Exercice 2 –

1. A partir d'un tableau d'entiers, récupérez la plus petite valeur contenue dans ce tableau.

2. Soustrayez cette valeur à tous les éléments du tableau.

Exercice 3 –

On rappelle le pseudocode du tri par bulle, pour un tableau `table` de taille `n` :

for i from 0 to $n-1$ **do**
  for j from 1 to $n-1$ **do**
    if `table`$[j] <$ `table`$[j-1]$ **then**
      Swap `table`$[j]$ and `table`$[j-1]$
    end if
  end for
end for

Implémenter ce code en assembleur.

Exercice 4 –

1. A partir d'un tableau d'entiers, et d'un tableau vide de même taille, remplissez le tableau vide avec, dans cet ordre, le premier élément du tableau d'origine, puis le dernier, puis le second, puis l'avant-dernier etc. Par exemple, le tableau `1 2 3 4 5 6 7 8` retournera `1 8 2 7 3 6 4 5`. On suppose dans cette question que le tableau est de longueur paire.

2. Même question, mais avec un tableau de taille arbitraire.

Exercice 5 –

1. Déclarez un tableau de 20 octets, tous initialisés à 32 (le code ASCII de l'espace).

2. Insérez dans le tableau, caractère par caractère, le texte rentré par l'utilisateur (s'arrêter lorsque vous détectez un retour à la ligne, code ASCII 10). Si le texte est trop long, alors bouclez sur le tableau en réécrivant à partir du début.

3. Affichez le contenu du tableau. Par exemple, le texte `Bien le bonjour a tous !` en entrée se verra afficher, en sortie, `us !  le bonjour à to`

4. Affichez les 20 derniers caractères tapés, dans l'ordre. Par exemple, le texte `Bien le bonjour a tous !` en entrée se verra afficher, en sortie, `  le bonjour à tous !`