Ordinateurs et Systèmes d'Exploitation
Exercices – VII

1. Les instructions suivantes effectuent des transferts de 16 bits de données. Décomposer chacune de ces instructions en deux instructions `mov` réalisant la même opération mais avec des opérandes de 8 bits.

   (a) `mov ax, 1234h`

   (b) `mov ax, [1234h]`

   (c) `mov bx, cx`

   (d) `mov [bx], cx`

   (e) `mov word [bp + si - 6], 32h`

   Solution

   (a)
   ```
   mov al, 34h
   mov ah, 12h
   ```

   (b)
   ```
   mov al, [1234h]
   mov ah, [1235h]
   ```

   (c)
   ```
   mov bh, ch
   mov bl, cl
   ```

   (d)
   ```
   mov [bx], cl
   mov [bx + 1], ch
   ```

   (e)
   ```
   mov byte [bp + si - 6], 32h
   mov byte [bp + si - 5], 0h
   ```

2. Quel sera le contenu du registre `ax` après la suite d'instructions suivante ?

   ```
   mov ax, 100h
   mov bx, ax
   mov [bx + 1], ax
   mov [100h], ax
   mov si, [bx + 1]
   mov al, [si]
   mov ah, [bx + si - 101h]
   ```

   Solution Le registre `ax` contiendra la valeur 0001h.

Instruction	ax	al	ah	bx	si	[100h]	[101h]	[102h]
`mov ax, 100h`	0100h	00h	01h					
`mov bx, ax`	0100h	00h	01h	0100h				
`mov [bx + 1], ax`	0100h	00h	01h	0100h			00h	01h
`mov [100h], ax`	0100h	00h	01h	0100h		00h	01h	01h
`mov si, [bx + 1]`	0100h	00h	01h	0100h	0101h	00h	01h	01h
`mov al, [si]`	0101h	01h	01h	0100h	0101h	00h	01h	01h
`mov ah, [bx + si - 101h]`	0001h	01h	00h	0100h	0101h	00h	01h	01h

3. Pourquoi les instructions suivantes sont-elles incorrectes ?

   (a) `mov ax, cl`

   (b) `mov [ax], cl`

   (c) `mov [bp + si], 99h`

   (d) `mov 1234h, ax`

   (e) `mov dx, [bp + bx]`

   (f) `mov dx, bp + si`

   (g) `mov dx, [bp - si - 9]`

   Solution

   (a) Les tailles des deux opérandes sont différentes : 8 bits pour `cl` et 16 bits pour `ax`.

   (b) Le registre `ax` ne peut pas être utilisé pour l'adressage indirect.

   (c) Le contexte ne permet pas de déterminer la taille des opérandes. Il faut préciser celle-ci par un marqueur de taille : `byte` pour 8 bits, `word` pour 16 bits.

   (d) On ne peut pas transférer des données dans une opérande constante !

   (e) Les registres `bp` et `bx` ne peuvent pas être utilisés ensembles dans l'adressage indirect indexé.

   (f) La seconde opérande ne correspond à aucun mode d'adressage. Il faut utiliser une instruction spécifique (`add`) pour calculer la somme des contenus de deux registres.

   (g) On ne peut prendre que la somme de deux registres dans l'adressage indirect indexé, pas leur différence.

4. Écrire un programme assembleur 80x86 destiné à compter le nombre d'octets non nuls contenus dans une suite d'octets consécutifs du segment de données. L'offset du premier octet est initialement contenu dans le registre BX et le nombre d'octets de la suite se trouve dans le registre CX. Le nombre d'octets non nuls devra être contenu dans le registre AX en fin d'exécution.

   Solution Il suffit de balayer l'ensemble des octets de la suite, et de tester pour chacun d'entre eux s'il est non nul. Le cas échéant, on augmente d'une unité le registre AX.

```
          SECTION .text
          mov ax, 0
boucle:   cmp [bx], 0
          je suivant
          inc ax
suivant:  inc bx
          loop boucle
          ret
```

5. Écrire un programme assembleur 80x86 capable de calculer le produit scalaire de deux vecteurs de nombres entiers. Les vecteurs sont donnés par des suites d'octets consécutifs du segment de données, pointées initialement par les registres AX et BX. Le registre CX contient le nombre de composants des vecteurs (ils ont la même dimension). En fin d'exécution, le registre DX devra contenir le produit scalaire de ces deux vecteurs.

   Solution Il suffit de parcourir simultanément les suites d'octets respectivement pointées par AX et BX. Pour chaque couple d'octets rencontré, on ajoute leur produit au registre DX.

```
          SECTION .text
          mov bp, ax
          mov dx, 0
          mov si, 0
boucle:   mov al, [bp + si]
          mul byte [bx + si]
          add dx, ax
          inc si
          loop boucle
          ret
```