

TD2 – Instructions de contrôle et de test, écriture en mémoire vidéo, instructions logiques, lecture au clavier et affichage à l'écran

But de ce TD :

Manipulation de l'instruction de saut inconditionnel : jmp.

Manipulation des instructions de saut conditionnel : jcc , éventuellement loop.

Manipulation de la notion de label.

Manipulation de l'instruction de test : cmp.

Manipulation des instructions : neg.

Ecrire directement en l'adresse B8000h.

Manipulation des instructions : not, and, or, xor.

Services d'interruption: int 10h, int 21h

Comparaisons et sauts

Les instructions suivantes permettent de vérifier si le contenu de la variable a est pair ou impair.

```
; -----  
; début du module  
; -----  
    mov ax,a  
    mov bl,2  
    div bl  
si:  
    cmp ah,0  
    jne alors  
alors:  
    mov estPair,0    ; impair  
    jmp finsi  
sinon:  
    mov estPair,1    ; pair  
finsi:  
; -----  
; fin module  
; -----
```

Les labels **si** et **finsi** permettent simplement d'augmenter la lisibilité du code.

Ex1. Soient `a` et `valAbs` deux variables de taille un octet. Ecrivez un code qui stocke dans `valAbs` la valeur absolue du contenu de `a`.

Testez votre code dans `td` avec les valeurs suivantes : `a = 25` ; `a = -30` ; `a = 200`.

Qu'observez-vous ?

Ex2. Soient `a`, `aMin` et `aMax` trois variables de taille deux octets et `estIn` une variable de taille un octet. On suppose avoir toujours `aMin < aMax`. Ecrivez deux codes qui stockent dans `estIn` :

– la valeur 1 si $a_{\text{Min}} \leq a < a_{\text{Max}}$;

– la valeur 0 sinon.

Le premier code ne manipule que des entiers non signés. Le second est adapté aux entiers signés.

Testez vos codes dans `td`.

Boucles

Pour les boucles, on peut utiliser l'instruction `loop` ou utiliser une autre structure

```
; -----  
; version tant que  
; somme des 10 premiers entiers  
; Debut module  
; -----  
    mov ax,0  
    mov cx,1  
tantque:  
    cmp cx,10  
    ja fintantque  
    add ax,cx  
    add cx,1  
    jmp tantque  
fintantque:  
    mov sum,ax  
; -----  
; fin module  
; -----
```

```
; -----  
; version repeat until  
; somme des 10 premiers entiers  
; Debut module  
; -----  
    mov ax,0  
    mov cx,0  
repeat:  
    add ax,cx  
    inc cx  
until:  
    cmp cx,10  
    jng repeat  
finrepeat:  
    mov sum,ax  
; -----  
; fin module  
; -----
```

```

; -----
; version avec loop : c'est un cas particulier
; somme des 10 premiers entiers
; Debut module
; -----
    mov ax,0
    mov cx,10
repeat:
    add ax,cx
    loop repeat
finrepeat:
    mov sum,ax
; -----
; fin module
; -----

```

Ex3 : Soient N et somme deux variables de taille deux octets. On considère le contenu de N toujours positif. Ecrivez un code qui stocke dans somme la somme des entiers positifs de 0 à N
Testez votre code dans td avec les valeurs suivantes : N = 25 ; N = 361 ; N = 400.
Qu'observez-vous ?

Ex4 : Soient a et sqrt deux variables de taille deux octets. Ecrivez un code qui stocke dans sqrt la racine carrée entière par défaut de a. On suppose le contenu de a toujours positif.

A titre d'exemple :

- pour a = 81 ; sqrt = 9
- pour a = 90 ; sqrt = 9
- pour a = 99 ; sqrt = 9
- pour a = 100 ; sqrt = 10

Indication:

- on partira d'une valeur du résultat égal à 1 (result = 1);
- on élèvera cette valeur au carré (resultCarré = result * result);
- tant que le résultat élevé au carré est inférieur au nombre dont on cherche la racine carré, on incrémente le résultat de 1:
 - **TQ** (resultCarré < Nombre) **FAIRE**
 - result <----- result+1
 - resultCarré <----- result * result
 - **FIN TQ**
- etc ...

Ecrire directement en l'adresse B8000h

Pour bien comprendre cette partie du TD, il est conseillé de faire une petite recherche ou une relecture des différents **modes d'adressage**. Nous vous donnons ici quelques uns:

- adressage direct : mov ax, [100] autre exemple mov bx, es: [100]
- adressage basé: mov ax, [bx]

- adressage indexé: `mov ax, [di]`
- adressage indexé et basé: `mov ax, [bx+si]`
- adressage basé avec déplacement: `mov ax, [bx+15h]`
- adressage indexé avec déplacement: `mov ax, [di+15h]`
- adressage basé et indexé avec déplacement: `mov ax, [bx+si+15h]`
- etc ...

Autres lectures conseillées: la notion de **tableaux à 2 dimensions** (comment sont-ils traités en assembleur ?).

La **zone mémoire à partir de 0B800h** (0B800 :0000) permet de stocker la représentation de l'écran en mode texte. Chaque caractère à l'écran est stocké sur **2 bytes**.

Le byte de poids fort représente l'attribut du caractère CRVBIRVB:

- C, clignote
- RVB, caractéristiques RVB de l'arrière plan (background)
- I, intensité de l'avant-plan (foreground)
- RVB, caractéristiques RVB de l'avant-plan

Le byte de poids faible représente le code ASCII du caractère

Par exemple, pour écrire le caractère '0' au milieu de l'écran,

```
...
mov AX ,0B800h
mov ES ,AX ; utilisation de l'extra segment
mov DH ,00000111b
mov DL , '0'
mov ES:[80*2*12+2*40],DX ; mode d'adressage ??
```

Ex4: Ecrivez un code assembleur qui permet d'afficher un message donné à l'écran. On écrira directement en l'adresse B8000h (ne pas utiliser les services d'interruption).

Instructions logiques

Ex5: Soit `var1` une variable de taille 2 octets et `puissDeux` une variable de taille un octet. On considère que le motif binaire stocké dans `var1` est celui d'un nombre non signé. Ecrivez un code qui stocke dans `puissDeux`:

- 0 si `var1` est impair
- 1 si `var1` est multiple de 2
- 2 si `var1` est multiple de 4
- 3 si `var1` est multiple de 6
-
- 15 si `var1` est multiple de 32768

Rédigez ce code sans utiliser l'instruction `DIV` et sans modifier le contenu initial de `var1`.

Testez votre code avec les valeurs suivantes :

- `var1 : 25`
- `var1 : 7428`
- `var1 : 40960`
- `var1 : 23552`

Indications:

- utilisation d'un masque;
- stocker la valeur de la variable dans un registre;
- utilisation de l'instruction `and` (entre le masque et la variable)
- utilisation du `zero flag`;
- utilisation de l'instruction `shl`.

Aide supplémentaire:

```
; -----  
; Que fait le code qui suit  
; Debut module  
; -----  
    mov result,0  
    mov ax,1  
    mov cx,16  
boucle:  
    mov bx,14  
    and bx,ax  
    jnz suite  
    inc result  
    shl ax,1  
    dec cx  
    jnz boucle  
suite:  
; -----  
; fin module  
; -----
```

De nombreux services d'interruption permettent la lecture d'un caractère au clavier.

Citons par exemple: `int 16h`, `service 00h` pour lire un caractère au clavier.

Pour plus d'informations sur ce sujet, référez-vous, par exemple, à `helpc`.

Documentation:

Lire un caractère `Int 16h`, `Fct 00h BIOS`

Cette fonction lit un caractère dans le buffer clavier. Si aucun caractère n'y est stocké, la fonction attend qu'un caractère ait été entré. Le caractère lu est retiré du buffer clavier.

Entrée:

AH = 00h

Sortie:

AL = 0 : Code clavier étendu, dans ce cas:

AH = Code clavier étendu différent de 0

Touche normale actionnée, dans ce cas

AL = Code ASCII de la touche

AH = Code clavier de la touche

Remarques:

. Le code ASCII d'un caractère est défini indépendamment de tel ou tel modèle de clavier, alors que le code clavier ne s'applique que sur le type de clavier connecté sur le PC

. Le contenu des registres CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

```
; -----  
; Lecture d'un caractère au clavier  
; -----  
.model small  
.stack 100h  
.code  
main proc  
    mov ax,@data  
    mov ds,ax  
    xor ax,ax  
    mov ah,0    ; utilisation de l'interruption  
    int 16h    ; pour lire un caractère au clavier  
    mov ax,4C00h  
    int 21h  
main endp  
end main
```

```

; -----
; Ecrire un caractère en couleur
; int 10h, fonction 09h
; -----
.model small
.stack 100h
.code
main proc
    mov ax,@data
    mov ds,ax
    xor ax,ax
    mov ah,09h    ;numéro de la fonction
    mov bh,00h    ;numéro de la page écran
    mov cx,0001h  ;nombre d'écritures successives du caractère
    mov al,51     ;code ASCII du caractère
    mov bl,00000001b ;attribut du caractère
    int 10h
    mov ax,4C00h
    int 21h
main endp
end main

```

Ex6: Ecrivez un programme qui lit deux nombres entiers, positif ou négatif, compris entre -15 et +15; qui stocke la plus petite valeur dans une variable `iMin` et la plus grande dans une variable `iMax`. Le programme calcule ensuite la somme des nombres compris entre `iMin` et `iMax` inclus, puis stocke le résultat dans une variable `result`. `iMin`, `iMax` et `result` sont ensuite affichés à l'écran.