Mme Fatima-Zahra BELOUADHA
Filière Génie Informatique
Ecole Mohammadia d'Ingénieurs

Architecture des ordinateurs

1ère année. Semestre 1

Utilisation de l'assembleur en TP
Exercices avec Solutions

CONTENU :

- Introduction à la programmation en assembleur (TASM)
- Debug en bref
- Entrées/Sorties et Opérations de calcul
- Chaînes de caractères
- Tableaux
- Gestion de l'écran

Introduction à la programmation en assembleur (TASM)

Le langage assembleur est classé dans la catégorie des langages "de bas niveaux". Il est intimement lié au microprocesseur et aux circuits d'un ordinateur.

1. Structure générale d'un programme assembleur

Un programme TASM (Turbo Assembleur) est composé de 5 sections dans l'ordre suivant :

Section	Description	Exemples
Dosseg	Provoque la mise en ordre des différents segments du programme en fonction de la convention d'organisation Microsoft afin d'assurer un fonctionnement correct.	
. Model	Permet de choisir l'un des modèles mémoires suivants : **- tyni :** CS et DS utilisent un même segment de 64ko. **- small :** CS et DS utilisent un segment de 64ko chacun. **- medium :** CS utilise plusieurs segments de 64 Ko et DS utilise un seul segment de 64ko. **- compact :** est l'inverse du modèle medium. **- large :** CS et DS utilisent plusieurs segments de 64ko chacun. **- hogue :** c'est le modèle large avec des structures de données qui dépassent les limites d'un seul segment.	**.Model small**
. stack	Permet de déterminer la **taille de la pile .** Elle est utilisée lors de l'usage d'une pile.	**.stack 200h** ; la taille de la pile est de 512o.
. data	Marque le début du segment de données. Cette section est utilisée uniquement pour déclarer des données s'il y en a.	**. data** **nombre dw ?** ; nombre codé sur un mot non ; initialisé. **Compteur db 0** ; compteur codé sur 1 octet ; initialisé à 0. **Message db 'bonjour$'** ; message initialisé à ; "bonjour".
. code	Marque le segment code (d'instructions).	**.code** **mov ax,0** ; mettre 0 dans AX. **mov bx,1** ; mettre 1 dans BX. **add ax, bx** ; somme de AX et BX.

Remarques :

1. Tout fichier assembleur doit avoir l'extension .asm.

2. Tout programme assembleur doit se terminer par la directive end. Le code qui suit le end sera ignoré par l'assembleur.

2. Syntaxe d'une ligne de code Assembleur (TASM)

• Syntaxe d'une ligne de déclaration

<étiquette> <type de donnée> <initialisation> <;commentaire>

- L'étiquette joue le rôle d'identificateur.

- Le type de donnée peut être égal à :
  - db pour une donnée qui se code sur un octet.
  - dw pour une donnée qui se code sur un mot (16 octets).
  - dd pour une donnée qui se code sur un double mot (32 octets).

- L'initialisation reflète :
  - soit un contenu concret initialisant la donnée déclarée. Dans le cas de nombres, ils doivent être repérés par rapport à leurs bases. Par défaut, ils sont considérés comme des décimaux.
  - soit un point d'interrogation ? indiquant que la donnée n'a pas été initialisée.

- Le commentaire commence par un point virgule et se termine à la fin de la ligne.

Exemples:

```
x db 12h                    ; x est initialisé à 12 en hexadécimal
y db 12d                    ; y est initialisé à 12 en décimal
z db 12                     ; z est initialisé à 12 en décimal
message db 'bonjour$'       ; message est une chaîne de caractères qui se codent sur un
                            ; octet chacun et qui est initialisée à "bonjour". Le caractère $
                            ; indique la fin de la chaîne de caractères.
M db 13,10,'bonjour',l3,10,[$] ; M est initialisé à la chaîne "bonjour" et est précédé et suivi
                            ; d'un retour à la ligne (13,10 : retour chariot et inline).
Tableau db 7 dup (0)        ; Tableau de 7 éléments codés chacun sur un octet et
                            ; initialisés à 0. Le tableau peut ne pas être initialisé en
                            ; mettant ? à la place de 0.
Liste db 1,2,3              ; Liste est un tableau énuméré contenant 1,2 et 3.
```

• Syntaxe d'une ligne de code (instruction)

<étiquette:> <instruction> <opérandes> <;commentaire>

- L'étiquette joue le rôle de référence à un emplacement mémoire (une partie du code comme par exemple les *procédures).* Il s'agit d'un identificateur suivi de deux points:.
- La partie instruction représente l'une des instructions prédéfinies de l'assembleur.
- La partie opérandes reflète les opérandes de l'instruction en question. Deux opérandes doivent être séparés par une virgule.
- Le commentaire commence par un point virgule et se termine à la fin de la ligne.

Exemple:

Mov bx,ax ; mettre le contenu du registre AX dans BX.

3. Exemple d'un premier programme en assembleur

```
Dosseg
.model small          ; choix du modèle small
.data
        n db 2        ; nombre initialisé à 2
        m db ?        ; donnée 8 bits non initialisée.

. code
        mov ax,@data      ; ces deux instructions servent à initialiser
        mov ds,ax         ; l'adresse du segment de données.
        mov al,n          ; met le contenu de n dans al
        call addition     ; appelle la procédure addition
         jmp fin          ; se déplace à l'emplacement libellé fin

addition:                 ; procédure addition
        mov bl,n          ; met le contenu de n dans bl.
        add al,bl         ; calcule la somme de al et bl.
        mov m,al          ; met le résultat dans m.
        ret               ; retour à la prochaine instruction après l'appel de procédure.

fin:                      ; procédure fin qui termine l'exécution du programme.
        mov ah,4ch        ; met la valeur 4ch dans ah.
        int 21h           ; appelle l'interrupteur Dos avec la valeur 21 h
end                       ; fin du code.
```

4. Installation du Turbo Assembleur TASM

Pour installer le logiciel TASM, il suffit de copier le répertoire TASM sur votre poste de travail.

Remarque :

Il est conseillé de rajouter dans le path le chemin du répertoire TASM (C:\tasm si vous l'avez installé dans le dossier C) afin de pouvoir accéder aux commandes Tasm à partir de n'importe quel répertoire.
Pour changer le path, il faut soit changer les variables d'environnement à partir du panneau de configuration soit le faire en mode DOS. Pour cela, il faut :

- Basculer d'abord en mode DOS : Cliquer sur *Tous les programmes/Accessoires/Invites de commandes*.
- Changer la variable PATH : Taper *set path=C:\TASM*

5. Assemblage et exécution d'un programme assembleur (TASM)

Avant d'exécuter un programme, il faut d'abord l'assembler puis éditer les liens.

Pour assembler le programme source portant l'extension .asm, il faut utiliser toujours sous DOS la commande tasm. Si le fichier source ne contient pas d'erreur syntaxique, un fichier .objet sera créé. Par exemple :

tasm nom_fichier.asm

Pour éditer les liens, il faut utiliser sous DOS la commande tlink. le fichier exécutable .exe sera alors créé. Par exemple :

tlink nom_fichier.obj

Enfin pour exécuter un programme, il suffit de taper le nom de son exécutable.

Debug en bref

Debug est un utilitaire Dos qui permet d'écrire de petits programmes en langage d'assemblage dont le code et les données partagent un seul segment de 64Ko.

Commandes Debug

1- Execution de Debug

Debug
L'exécution de cette commande sous Dos donne la main pour utiliser les commandes Debug.

2- L'éditeur

A adresse
Permet de commencer la saisie d'un programme qui sera stocké à partir de l'adresse de déplacement indiquée dans un segment alloué aléatoirement.
Pour terminer, il suffit de taper Entrée (Retour chariot).

3- Sauvegarde de programmes

N nom Fichier.com
Permet d'attribuer un nom de fichier au programme saisi (en cours).

W
Sauvegarde sur disque dur le programme en cours dont le nombre d'octets doit être déjà indiqué dans le registre CX (la commande R CX permet d'affecter une valeur à CX).

4- Exécution de programmes

G
Cette commande permet d'exécuter le programme en cours.

5- Chargement d'un programme (2 manières de faire sont possibles)

Debug nom_fichier.com
Permet de charger en mémoire un programme sauvegardé sur le disque dur. Cette commande est utilisée au moment où l'on appelle debug.

N nomfichier.com (indiquer le nom du fichier)
L (le charger en lecture)

6- Désassemblage de programmes

U
Cette commande permet de désassembler le programme en cours (donner le code machine correspondant).

U adresse 1 adresse2

Permet de désassembler une partie du programme en cours comprise entre les adresses de déplacement adressel et adresse2.

7- Mise à jour de programmes

A adresse

Permet de modifier ou de reprendre la saisie à partir de l'adresse indiquée.

8- Trace de programmes

T

Permet de suivre le cheminement du programme en cours (visualiser les contenus de registres et d'indicateurs suite à l'exécution de chaque instruction). Il suffit de continuer à taper T pour voir ce qui se passe à chaque étape.

9- Points d'arrêts

G = adresse1 adresse2

Permet d'exécuter une partie du programme en cours comprise entre adresse1 et adresse2. Pour continuer à suivre le cheminement du programme, il suffit d'utiliser la commande T.

10- Consultation et mise à jour de contenus des registres

R registre (Affiche le contenu du registre et donne la main pour le modifier).

11- Consultation du contenu de la mémoire

D

Affiche le contenu de la mémoire. Il suffit de continuer à taper D pour visualiser plusieurs segments mémoire.

D adresse Affiche le contenu de la mémoire à partir de l'adresse indiquée.

12- Modification du contenu de la mémoire

F adresse1 adresse2 Code ASCII

Remplit toutes les cellules du segment mémoire compris entre les adresses adressel et adresse2 par le caractère dont le code ASCII est indiqué.

E adresse

Permet de modifier le contenu de la case mémoire à l'adresse indiquée. Lors de son exécution, cette commande affiche respectivement le code ASCII de l'ancien caractère et son attribut en donnant la possibilité de les modifier.

Remarques :

- Debug considèrent par défaut les nombres comme des hexadécimaux. Aussi, aucun nombre dans le code ne doit être suivi par la lettre faisant référence à sa base.

- Il est conseillé de considérer un programme élaboré en TP comme une séquence .COM. Ainsi, doit-il commencer à l'adresse de déplacement 100. Par contre, un programme exécutable .EXE doit toujours commencer à l'adresse de déplacement 0.

Université Mohammed V - Agdal **Année Universitaire : 2011/2012**
Ecole Mohammadia d'Ingénieurs
Filière Génie Informatique

1ère année.
Série N° 1 d'Exercices

Objectif : Utiliser les Entrées/sorties et les instructions de calcul.

Rappel :

Des fonctions DOS peuvent être appelées au niveau d'un programme écrit en un langage d'assemblage. Elles peuvent, à titre d'exemples, être utilisées pour saisir ou afficher un caractère ou une chaine de caractères et sont utilisées comme suit :

Saisie d'un caractère
Entrée : AH = 1 (Numéro de la fonction Dos à exécuter)
Interruption: 21H (appelée via l'instruction Int 21h)
Sortie: AL= Code ASCII du caractère saisi.

Affichage d'un caractère
Entrée : AH = 2
DL = Code ASCII du caractère à afficher.
Interruption : 21H
Sortie : AUCUNE.

Affichage d'une chaîne de caractères
Entrée : AH = 9
DX = offset de la chaine de caractères à afficher.
(par exemple, l'instruction: mov dx, offset message fait pointer DX sur l'adresse de déplacement de la chaîne *message*.)
Interruption: 21H
Sortie: Aucune

Exercice 1 :

a) Ecrire un programme qui affiche un caractère saisi à partir du clavier.
b) Modifier ce programme en rajoutant des messages (comme "Taper un caractère" et "Voilà le caractère tapé")

Exercice 2 :

a) Ecrire un programme qui calcule la somme de deux chiffres décimaux. On suppose que le résultat est inférieur à 9.
On rappelle que l'instruction ADD calcule la somme de deux opérandes.
b) Modifier ce programme pour calculer la somme de deux décimaux dont le résultat est constitué de 2 chiffres. On prend l'exemple de 3 et 9.

Pour cela, utiliser :

DAA instruction sans opérandes, devant être appelée immédiatement après l'addition, pour faire un ajustage décimal. L'opération d'ajustage permet de coder le résultat, calculé en hexadécimal et contenu dans l'accumulateur, en DCB (décimal Codé Binaire).

AND instruction de **masquage logique.** Elle est similaire à ADD sauf que les sommes qu'elle fournit sont fondées sur les règles suivantes : 1+1=1, 0+1=0 et 0+0=0

SHR instruction de **décalage logique à droite.** Le nombre de fois de décalage doit être indiqué dans CL. Par exemple, si CL=n, l'instruction shr al,cl décale les bits de AL n fois à droite.

Exercice 3 :

Essayer l'ensemble des commandes Debug pour le cas d'un programme qui affiche la lettre A (code ASCII 41 en héxadécimal).

Université Mohammed V - Agdal **Année Universitaire : 2011/2012**
Ecole Mohammadia d'Ingénieurs
Filière Génie Informatique

1ère année.
Série N° 2 d'Exercices

Objectif : Manipuler des chaînes de caractères.

Rappel :

BX Registre qui peut être utilisé comme pointeur sur un tableau. Dans ce cas, il doit être égal à l'offset du tableau en question.

[pointeur] Indique le contenu de l'élément pointé par pointeur. Dans le cas d'un pointeur sur tableau, elle représente le contenu de la case courante sur laquelle pointe le pointeur en question.

INC Instruction qui incrémente son opérande de 1. Lorsqu'il s'agit de pointeur sur tableau. elle le fait avancer vers l'élément suivant.

CMP Instruction qui effectue une comparaison entre ses deux opérandes.

JZ étiquette Utilisée immédiatement après une comparaison, elle permet si le résultat de celle-ci est égal à 0, d'aller à l'emplacement repéré par étiquette (une procédure par exemple).
Voir aussi JNZ, JE (equal=), JA (above, >), JB (bellow, <) ...

Fonction de saisie d'une chaîne de caractères

Entrée: AH = 10 ou 0Ah
DX = Offset du tableau devant recevoir la chaîne de caractères.
Interruption : 21H
Sortie : TABLEAU dont le contenu est comme suit :
1ère case : contient le nombre de caractères à lire.
2ème case : contient le nombre de caractères effectivement lus.
Le reste : contient le texte saisi.

Exercice 1 :

Ecrire un programme qui inverse une chaîne de caractères.

Exercice 2 :

a) Ecrire un programme qui affiche une chaîne de caractères saisie à partir du clavier.

b) Modifier ce programme pour qu'il affiche la première position et le nombre d'occurrences d'une lettre dans une chaîne de caractères initialisée dans le programme.

On rappelle que :
LEA pointeur, tableau+n est une instruction qui fait pointer le pointeur sur la n+1ème case du tableau.

c) Améliorer le programme pour l'appliquer à une chaîne de caractère saisie au clavier.

Université Mohammed V - Agdal **Année Universitaire : 2011/2012**
Ecole Mohammadia d'Ingénieurs
Filière Génie Informatique

1ère année.
Série N° 3 d'Exercices

Objectif : Manipuler des tableaux.

Rappel :

SI et **DI** sont des registres qui peuvent être utilisés comme **indices** d'un tableau.

tableau[indice] désigne le contenu de la case dont la position est indice dans le tableau.

Les indices d'un tableau en assembleur commencent à partir de 0.

XCHG est une instruction qui échange le contenu de ses deux opérandes.

Exercice 1 :

Ecrire un programme qui cherche le minimum dans un tableau de chiffres saisis à partir du clavier.

Exercice 2 :

Ecrire un programme qui trie un tableau d'entiers 8bits saisis à partir du clavier.

Utiliser les procédures suivantes :
- Saisie d'un nombre.
- Conversion d'une chaîne de caractères en nombre concret.
- Conversion inverse nombre-chaîne de caractères.
- Tri d'un tableau de nombres.
- Affichage d'un tableau.

Université Mohammed V - Agdal **Année Universitaire : 2011/2012**
Ecole Mohammadia d'Ingénieurs
Filière Génie Informatique

1ère année.
Série N° 4 d'

Objectif : Comprendre et apprendre à gérer l'écran.

Rappel :

Le segment mémoire de la RAM vidéo réservé à l'écran commence à partir de :

- L'adresse B000H:0000H pour des cartes monochromes.
- L'adresse B800H:0000H pour des cartes graphiques couleurs.

Un point de l'écran est codé sur deux octets. Il est donc représenté dans la RAM vidéo par deux octets. Son code ASCII sur le 1er octet et son attribut sur le 2ème.

L'adresse de déplacement d'un point de l'écran peut être calculée par la formule suivante : (ligne*80+colonne)*2.

L'octet attribut est décrit comme suit:

- Le quartet de poids faible représente le code de la couleur du fond.
- Le quartet de poids fort représente le code de la couleur du texte.
- Le bit de poids fort, s'il est à 1 le caractère en question clignotera, s'il est à 0 l'affichage sera alors normal.

Exercie 1 :

Ecrire un programme qui affiche sur l'écran un message à la ligne 12, colonne 20 avec un attribut de couleur. Pour cela, utiliser un accès direct à la mémoire vidéo en mode texte.

On donne :

B800H:0000H	Adresse de la première case mémoire de la RAM viodéo.
STOSB	Instruction qui permet le transfert de AL vers ES:DI (emplacement DI dans l'extra segment ES. Son exécution incrémente automatiquement le registre index DI.
BL	Ce registre peut être utilisé comme registre contenant les données d'un point de l'écran.

Exercie 2 :

Réécrire le programme de l'exercice 1 en utilisant l'interruption 10h du BIOS.
On rappelle les fonctions BIOS suivantes :

Positionnement du curseur

Entrée : AH = 2
DH,DL = Rangée, colonne (Rangées de 0 à 24, colonnes de 0 à 79)
BH = numéro de la page (0 pour modes graphiques; pages de 0 à 4)
Interruption : 10H
Sortie : AUCUNE.

Ecriture de l'attribut / du caractère à la position du curseur

Entrée : AH = 9
BH = numéro de la page (0 à 3 pour modes caractères)
BL = attribut du caractère
CX = compteur des caractères à écrire
AL = Caractère à écrire
Interruption : 10H
Sortie : AUCUNE.

Exercice 3 :

Afficher un message au début de l'écran en utilisant l'une des commandes debug.

Solutions

```
;---------------------------------------------------------------------------
; Série 11 - Exercice 1
;---------------------------------------------------------------------------
; SAISIE_Affichage Caractère.ASM
;---------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small
.data
message db 'Tapez un caractère : $'
.stack
.code
mov ax,@data                    ; ces deux instructions servent à
mov ds,ax                       ; initialiser l'adresse du segment de code.
call affichage_message
call saisie
call affichage
call fin

saisie :
      MOV AH,01h
      INT 21h             ; lit un caractere et le place dans AL
 ret
Affichage :
 MOV DL, AL
 MOV AH,02h
      INT 21h             ; lit un caractere et le place dans AL
 ret
Affichage_message:
 MOV DX, offset message
 MOV AH,09h
      INT 21h             ; affiche une chaine de caracteres pointé par DX
 ret
fin:

      MOV AH,4Ch
      INT 21h             ; retour au DOS

end                                   ;fin du code.


;---------------------------------------------------------------------------
; Série 12 - Exercice 2
;---------------------------------------------------------------------------
; Somme de 2 chiffres saisis > 9 (ex : 3+9)
;---------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small
```

```
.data
n db ?
message1 db 'Tapez un chiffre :',10,13, '$'
message2 db 10,13, 'La somme est : $'
.stack
.code
mov ax,@data                ; ces deux instructions servent à
mov ds,ax                   ; initialiser l'adresse du segment de code.
call affichage_message1
call saisie
sub al, 48
mov n, al
call affichage_message1
call saisie
sub al, 48
add al,n
daa
mov s, al

call affichage_message2
mov al,s
mov cl, 4
shr al, cl
add al, 48
call affichage
mov al,s
and al, 0Fh
add al, 48
call affichage
call fin


Affichage :
 MOV DL, AL
 MOV AH,02h
      INT 21h          ; affiche un caractere placé dans AL
 ret
Affichage_message2:
 MOV DX, offset message2
 MOV AH,09h
      INT 21h          ; affiche une chaine de caracteres pointé par DX
 ret
fin:
      MOV AH,4Ch
      INT 21h          ; retour au DOS

end                                ;fin du code.


;-------------------------------------------------------------------------
```

```
; Série 13 - Exercice 3
;-------------------------------------------------------------------------
; Affichage de la lettre A avec debug
;-------------------------------------------------------------------------
MOV AL,41
MOV DL, AL
MOV AH,2
INT 21
MOV AH,4C
INT 21
```

```
;-------------------------------------------------------------------------
; Série 21 - Exercice 1
;-------------------------------------------------------------------------
; Inverse d'une chaine
;-------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small
.data
n db 1
message1 db 'ma chaine$'
message2 db 'Résultat de conversion : $'
.stack 200h
.code
mov ax,@data                    ; ces deux instructions servent à
mov ds,ax                       ; initialiser l'adresse du segment de code.

call affichage_message2

mov bx,offset message1
loop1 :
inc bx
inc n
cmp n,9
jnz loop1

loop2 :
mov al, [bx]
call affichage
dec bx
dec n
cmp n,0
jnz loop2

call fin

;saisie :
;       MOV AH,01h
;       INT 21h              ; lit un caractere et le place dans AL
;ret
Affichage :
 MOV DL, AL
 MOV AH,02h
      INT 21h               ; affiche un caractere placé dans AL
 ret
;Affichage_message1:
;MOV DX, offset message1
;MOV AH,09h
;       INT 21h              ; affiche une chaine de caracteres pointé par DX
;ret
Affichage_message2:
```

```
 MOV DX, offset message2
 MOV AH,09h
      INT 21h              ; affiche une chaine de caracteres pointé par DX
 ret
fin:

      MOV AH,4Ch
      INT 21h              ; retour au DOS

end                                   ;fin du code.


;-------------------------------------------------------------------------
; Série 22 - Exercice 2a
;-------------------------------------------------------------------------
; Saisie/Affichage chaine
;-------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small

.data
chaine db 10 DUP ('$')
message1 db 'Taper une chaine : $'
message2 db 10, 13, 'la chaine est : $'

.stack 200h

.code
mov ax,@data                  ; ces deux instructions servent à
mov ds,ax                     ; initialiser l'adresse du segment de code.

call affichage_message1
call saisieCh


call affichage_message2
call affichageCh

call fin

saisieCh:
      mov dx, offset chaine
      MOV AH,0Ah
 INT 21h              ; lit une chaine de caracteres pointée par dx
 ret
Affichage_message1:
 MOV DX, offset message1
 MOV AH,09h
      INT 21h              ; affiche une chaine de caracteres pointé par DX
 ret
```

```
Affichage_message2:
 MOV DX, offset message2
 MOV AH,09h
      INT 21h            ; affiche une chaine de caracteres pointé par DX
 ret
AffichageCh:
 MOV DX, offset chaine
 inc dx
 inc dx
 MOV AH,09h
      INT 21h            ; affiche une chaine de caractères pointé par DX
 ret
fin:
 MOV AH,4Ch
      INT 21h            ; retour au DOS

End                      ;fin du code.


;-------------------------------------------------------------------------
; Série 22 - Exercice 2c (2b amélioré)
;-------------------------------------------------------------------------
; Saisie/Nombre d'occurrences dans une chaine
;-------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small

.data
c db ?
d db '$'
chaine db 20 DUP ('$')
message1 db 'Taper une chaine : $'
message2 db 10,13,'Taper le caractère à chercher: $'
message3 db 10, 13, 'Première position : $'
message4 db 10,13,'Nombre doccurences : $'

.stack 200h

.code
mov ax,@data             ; ces deux instructions servent à
mov ds,ax                ; initialiser l'adresse du segment de code.

call affichage_message1
call saisieCh

call affichage_message2
call saisie
mov bl,al
mov c,al
```

```
call affichage_message3
mov SI,1
loop1:
inc si
cmp bl,chaine[si]
jnz loop1
mov ax, si
dec ax
dec ax
add al, 48
call affichage

call affichage_message4
lea bx, chaine+si
mov DI,1
loop2:
inc bx
mov ax, [bx]
cmp al, d
jz result
cmp al,c
jz maj
continue:
jmp loop2

maj:
inc di
jmp continue

result:
mov ax, di
add al, 48
call affichage

call fin

saisieCh:
      mov dx, offset chaine
      MOV AH,0Ah
 INT 21h              ; lit une chaine de caracteres pointée par dx
 ret
saisie:
      MOV AH,01h
      INT 21h              ; lit un caractere et le place dans AL
 ret
Affichage:
 MOV DL, AL
 MOV AH,02h
      INT 21h              ; affiche un caractere placé dans AL
```

```
 ret
Affichage_message1:
 MOV DX, offset message1
 MOV AH,09h
      INT 21h              ; affiche une chaine de caractères pointé par DX
 ret
Affichage_message2:
 MOV DX, offset message2
 MOV AH,09h
      INT 21h              ; affiche une chaine de caractères pointé par DX
 ret
Affichage_message3:
 MOV DX, offset message3
 MOV AH,09h
      INT 21h              ; affiche une chaine de caracteres pointé par DX
 ret
Affichage_message4:
 MOV DX, offset message4
 MOV AH,09h
      INT 21h              ; affiche une chaine de caracteres pointé par DX
 ret
;AffichageCh:
;MOV DX, offset chaine
;inc dx
;inc dx
;MOV AH,09h
;       INT 21h              ; affiche une chaine de caracteres pointé par DX
;ret
fin:
 MOV AH,4Ch
      INT 21h              ; retour au DOS

end                                ;fin du code.
```

```
;-------------------------------------------------------------------------
; Série 31 - Exercice 1
;-------------------------------------------------------------------------
; Minimum d'un tableau saisi
;-------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small
.data
n dw 0
tab db 10 DUP (?)
min db ?
message1 db 'Nombre d_elements : $'
message2 db 10, 13, 'Saisir les elements du tableau : $'
message3 db 10, 13, 'Minimum : $'
message4 db 'OK $'
.stack 200h
.code
mov ax,@data                    ; ces deux instructions servent à
mov ds,ax                       ; initialiser l'adresse du segment de code.

call affichage_message1
call saisie
sub al, 48
mov ah,0
mov n,ax

call affichage_message2

mov SI, 0
loop1 :
call saisie
sub al, 48
mov tab[SI],al
inc SI
mov ax, SI
cmp ax, n
jnz loop1

call affichage_message3
mov SI,0
mov al, tab[SI]
mov min, al
inc SI
loop2 :
mov al, min
cmp al,tab[SI]
ja change
inc SI
mov ax, SI
cmp ax, n
```

```
jnz loop2

mov al, min
add al, 48
call affichage

call fin

change :
mov al, tab[SI]
mov min, al
call affichage_message4
ret

saisie :
      MOV AH,01h
      INT 21h          ; lit un caractere et le place dans AL
 ret
Affichage :
 MOV DL, AL
 MOV AH,02h
      INT 21h          ; affiche un caractere placé dans AL
 ret
Affichage_message1:
 MOV DX, offset message1
 MOV AH,09h
      INT 21h          ; affiche une chaine de caracteres pointé par DX
 ret
Affichage_message2:
 MOV DX, offset message2
 MOV AH,09h
      INT 21h          ; affiche une chaine de caracteres pointé par DX
 ret
Affichage_message3:
 MOV DX, offset message3
 MOV AH,09h
      INT 21h          ; affiche une chaine de caracteres pointé par DX
 ret
Affichage_message4:
 MOV DX, offset message4
 MOV AH,09h
      INT 21h          ; affiche une chaine de caracteres pointé par DX
 ret
fin:

      mov al, min
add al, 48
call affichage
MOV AH,4Ch
      INT 21h          ; retour au DOS
```

```
end                                    ;fin du code.
```

```
;-------------------------------------------------------------------------
; Série 41 - Exercice 1
;-------------------------------------------------------------------------
; Affichage sur l'écran _ STOSB
;-------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small

.data
debut dw 0B800h
pos dw 1960
attribut db 0E4h                ; fond jaune, texte rouge
.stack 200h

.code
mov ax,@data                    ; ces deux instructions servent à
mov ds,ax                       ; initialiser l'adresse du segment de code.

;mov ah,00h                     ; initialiserle mode graphique
;mov al, 03h                    ; numéro du mode (80x25 texte et 16 couleurs)
;int 10h

mov es,debut
mov di,pos
mov al,'F'
stosb
mov al,attribut
stosb


call fin

;saisieCh:
;       mov dx, offset chaine
;       MOV AH,0Ah
;INT 21h                ; lit une chaine de caracteres pointée par dx
;ret
;saisie:
;       MOV AH,01h
;       INT 21h                 ; lit un caractere et le place dans AL
;ret
;Affichage:
;MOV DL, AL
;MOV AH,02h
;      INT 21h                ; affiche un caractere placé dans AL
;ret
;Affichage_message1:
;MOV DX, offset message1
;MOV AH,09h
;       INT 21h                 ; affiche une chaine de caracteres pointé par DX
```

```
;ret
;Affichage_message2:
;MOV DX, offset message2
;MOV AH,09h
;       INT 21h             ; affiche une chaine de caracteres pointé par DX
;ret
;Affichage_message3:
;MOV DX, offset message3
;MOV AH,09h
;       INT 21h             ; affiche une chaine de caracteres pointé par DX
;ret
;Affichage_message4:
;MOV DX, offset message4
;MOV AH,09h
;       INT 21h             ; affiche une chaine de caracteres pointé par DX
;ret

fin:
 MOV AH,4Ch
      INT 21h               ; retour au DOS

end                         ;fin du code.



;-------------------------------------------------------------------------
; Série 42 - Exercice 2
;-------------------------------------------------------------------------
; Affichage sur l'écran : fonction 10H
;-------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small

.data
attribut db 0E4h                ;fond jaune, texte rouge
.stack 200h

.code
mov ax,@data                    ; ces deux instructions servent à
mov ds,ax                       ; initialiser l'adresse du segment de code.

call position
call affichage


call fin

position:
 mov bh,0
 mov dh,12
```

```
 mov dl,20
      MOV AH,02h
 INT 10h              ; lit une chaine de caracteres pointée par dx
 ret

Affichage:
 mov bh,0
 mov bl, attribut
 MOV cx,4
 mov AL, 'G'
 MOV AH,09h
      INT 10h              ; affiche un caractere placé dans AL
 ret
fin:
 MOV AH,4Ch
      INT 21h              ; retour au DOS

end                         ;fin du code.
```

```
;--------------------------------------------------------------------------
; Série 43 - Exercice 3

Sous debug taper e b800:0000
Et donner les valeurs des attributs

;--------------------------------------------------------------------------
```

Autres Exercices

```
;--------------------------------------------------------------------------
; Exercice
;--------------------------------------------------------------------------
; Somme 14h et 32h
;--------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small
.data
a db 14h
b db 32h
s db ?
message2 db 'La somme est : $'
.stack 200h
.code
mov ax,@data                ; ces deux instructions servent à
mov ds,ax                   ; initialiser l'adresse du segment de code.

mov al, a
add al,b
mov s, al
```

```
call affichage_message2
mov al,s
mov cl, 4
shr al, cl
add al, 48
call affichage
mov al,s
and al, 0Fh
add al, 48
call affichage
call fin

Affichage :
 MOV DL, AL
 MOV AH,02h
      INT 21h            ; affiche un caractere placé dans AL
 ret
Affichage_message2:
 MOV DX, offset message2
 MOV AH,09h
      INT 21h            ; affiche une chaine de caracteres pointé par DX
 ret
fin:

      MOV AH,4Ch
      INT 21h            ; retour au DOS

end                                   ;fin du code.


;-------------------------------------------------------------------------
; Somme 3 et 9
;-------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small
.data
a db 3
b db 9
s db ?

message2 db 'La somme est : $'
.stack 200h
.code
mov ax,@data                ; ces deux instructions servent à
mov ds,ax                   ; initialiser l'adresse du segment de code.

mov al, a
add al,b
daa
```

```
mov s, al

call affichage_message2
mov al,s
mov cl, 4
shr al, cl
add al, 48
call affichage
mov al,s
and al, 0Fh
add al, 48
call affichage
call fin


Affichage :
 MOV DL, AL
 MOV AH,02h
      INT 21h            ; affiche un caractere placé dans AL
 ret
Affichage_message2:
 MOV DX, offset message2
 MOV AH,09h
      INT 21h            ; affiche une chaine de caracteres pointé par DX
 ret
fin:

      MOV AH,4Ch
      INT 21h            ; retour au DOS

end                                   ;fin du code.



;---------------------------------------------------------------------------
; Somme 39Fh et 281h
;---------------------------------------------------------------------------
Dosseg
.model small   ; choix du modèle small
.data
a dw 39Fh
b dw 281h
s dw ?
message2 db 'La somme est : $'
.stack 200h
.code
mov ax,@data                  ; ces deux instructions servent à
mov ds,ax                     ; initialiser l'adresse du segment de code.

mov ax, a
```

```
mov bx, b
add al,bl
adc ah,bh
mov s, ax

call affichage_message2
mov ax,s
mov cl, 8
shr ax, cl
add al, 48
call affichage
mov ax, s
mov cl,4
shr al,cl
add al,48
call affichage
mov ax,s
and al, 0Fh
add al, 48
call affichage
call fin


Affichage :
 MOV DL, AL
 MOV AH,02h
      INT 21h            ; affiche un caractere placé dans AL
 ret

Affichage_message2:
 MOV DX, offset message2
 MOV AH,09h
      INT 21h            ; affiche une chaine de caracteres pointé par DX
 ret
fin:

      MOV AH,4Ch
      INT 21h            ; retour au DOS

end                                  ;fin du code.
```

; Exercice
;---
; SAISIE2.ASM
;---
;
.386
ASSUME CS:code, DS:data, SS:pile

code SEGMENT USE16
main:
	MOV AX,data
	MOV DS,AX		; initialise le segment DS

	MOV BX,0		; initialise l'indice du caractere courant
debut:
	MOV AH,08h
	INT 21h		; lit un caractere et le place dans AL
	CMP AL,13
	JZ fin		; si AL = entree fin du programme
	CMP AL,0
	JZ code_etendu		; si AL=0 alors code etendu
	CMP AL,8
	JZ retour_arriere	; si AL=BS alors touche RETOUR ARRIERE
	CMP BX,30
	JZ plus_que_30		; si BX - 30 = 0 alors nbre max de car
	CMP AL,32
	JS debut		; si AL-32<0 car non imprimable
	CMP AL,127
	JZ debut		; si AL=127 (touche SUPP) ignor,

			; sinon le carctere est imprimable,
			; le nombre max de caracteres non atteint
			; ALORS on sauvegarde le caractere lu
			; et on l'affiche
	MOV chaine[BX],AL	; sauvegarde le caractere lu dans chaine
	MOV AH,02h
	MOV DL,AL
	INT 21h		; affiche le caractere
	INC BX
	JMP debut		; lire prochain

code_etendu:
	INT 21h		; lire code suivant et l'ignorer
			; (la valeur de AH est toujours 08h)
	JMP debut
retour_arriere:
	CMP BX,0
	JZ debut		; si BX = 0 alors debut de la chaine
	DEC BX		; sinon effacer le caractere
	MOV chaine[BX],32	; mettre un blanc … la place du carc effac,

```
        MOV AH,02h
        MOV DL,8            ; retourne le curseur en arriere
        INT 21h
        MOV DL,32           ; affiche un blanc … la place du caractere
        INT 21h
        MOV DL,8            ; retourne le curseur en arriere
        INT 21h
        JMP debut           ; lire carctere suivant
plus_que_30:
        MOV AH,02h
        MOV DL,7            ; bip
        INT 21h
        JMP debut           ; Lire un autre caractere
fin:

        MOV AH,4Ch
        INT 21h             ; retour au DOS
code ENDS
;--------------------------------------------------------------------------
data SEGMENT USE16
        chaine  db  30 DUP(' '),13,10,'$' ; initialise chaine
data ENDS
;--------------------------------------------------------------------------
pile SEGMENT STACK
  remplissage DB 256 DUP (?)
pile ENDS
;--------------------------------------------------------------------------
END main
```