Ministère de l'Enseignement supérieur et de la Recherche scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique
Département d'Intelligence Artificielle et Sciences des Données

Rapport du Projet du la matière Architecture des ordinateurs II:

"Programmation de quantum de temps à l'aide de l'interruption périodique 1CH"

Chargée de cours:

Mme S.BOUCHENE

Chargée de Travaux Pratiques (TP):

Mr. A.HAOUARI

Préparé par:

Nom:	**BENSAFIA,**	**BARAGH,**
Prénom:	**Younès,**	**Yasser Abdeldjalil,**
Matricule:	**212132032981,**	**202032021716,**
Section:	**ACAD A,**	**ACAD A,**
Groupe:	**Groupe 4.**	**Groupe 4.**

Année universitaire 2022/2023

I) Objectif du TP:

L'Objectif de ce TP c'est d'étudier l'interruption 1CH et son implémentation et l'écriture des programmes et la vérification de leur bon fonctionnement en effectuant les tests nécessaires.

II) Logiciels & Matériels utilisés:

Le matériel utilisé est le suivant:

- Personal computer
- GUI Turbo Assembler

III) Méthode:

1. Programme **1_tache**:
   a. DATA SEGMENT:

```
; DEBUT DU DATA SEGMENT (DS)
data SEGMENT
    msg db " 1 sec ecoulee..... ", "$" ; MESSAGE 1
    msg2 db "****Debut du quantum de temps logiciel****", "$" ; MESSAGE 2
    dert db "Deroutement fait....",10,10,13,"$" ; MESSAGE 3
    compt dw 18 ; COMPTEUR POUR LES 1 SECONDES ECOULEES
    retLine db 10,13,"$" ; POUR LE RETOUR A LA LIGNE
data ENDS
; LA FIN DU DATA SEGMENT (DS)
```

   b. STACK SEGMENT (Pour la pile):

```
;;STACK SEGMENT
maPile segment stack
    dw 128 dup(?)
    tos label word
maPile ends
```

   c. CODE SEGMENT:

   c'est la ou toutes nos procédures existes.

```
; DEBUT CODE SEGMENT (CS)
code SEGMENT
    assume CS:code, DS: data, SS: maPile ;ASSUME
    ; LA FONCTION QUI FAIT LE DEROUTEMENT GRACE L'FONCTION AH=25h DU INT 21H
    derout_1CH PROC NEAR
    ; 25h/21h installer un nouveau vecteur ; ES/ AL : numero du vect a installer ; DS:DX: les adresses cs et ip de la nouvelle routine d it
    push ds ;SAUVGARDER LE DS DANS LA PILE AVANT D'INSTALLER LE VECTEUR
    mov ax , seg new
    mov ds , ax    ; CSnew DANS DS
    mov dx , offset new ; IPnew DANS DX
    mov ax , 251CH ;MOV AH,25H; MOV AL,1CH;
    int 21H
    pop ds
    ret
    derout_1CH ENDP
    ; PROCEDURE AFFICHER QUI VA AFFICHER TOUS LES MESSAGES DE CE PROGRAMME

affiche PROC NEAR
mov bp,sp; BP REJOINT SP
mov dx,[bp+2]; [BP + 2] VA CONTENIR TOUJOURS L OFFSET DU MESSAGE QU ON AFFICHER
mov ah , 09h; INT POUR AFFICHER UNE CHAINE DE CARACTERES /DX: OFFSET DU PREMIER CARACTERE DE LA CHAINE
```

i. La fonction **derout_1CH**:

```
derout_1CH PROC NEAR
; 25h/21h installer un nouveau vecteur ; ES/ AL : numero du vect a installer ; DS:DX: les adresses cs et ip de la nouvelle routine d it
push ds ;SAUVGARDER LE DS DANS LA PILE AVANT D'INSTALLER LE VECTEUR
mov ax , seg new
mov ds , ax    ; CSnew DANS DS
mov dx , offset new ; IPnew DANS DX
mov ax , 251CH ;MOV AH,25H; MOV AL,1CH;
int 21H
pop ds
ret
derout_1CH ENDP
; PROCEDURE AFFICHER QUI VA AFFICHER TOUS LES MESSAGES DE CE PROGRAMME
```

Grâce à l'interruption 21h (AH=25h) on a pu installer un vecteur qui va faire le travail de new, et dans ce cas on sera obligé de sauvegarder le DS dans la pile car il va changer sa valeur. (cette interruption, elle a besoin de DS(SEGMENT NEW), DX(OFFSET NEW), AL (Numéro d'interruption).

ii. **new:**

```
; DEROUTEMENT DU 1CH AVEC NEW
    new:
    dec compt ; DEC POUR 1 SEC
    jnz fin   ; IL VA ATTENDRE UNE SECONDE POUR AFFICHER
    push offset msg ; MESSAGE 1
    call NEAR PTR affiche
    pop bx ; POUR QU ON AURA PAS DE PROBLEME CONSERNANT LE IRET ET LA PILE
    mov compt, 18 ;UNE AUTRE SECONDE
    fin: iret ; PSW/CS/IP
```

dans **new:** on fait la décrémentation pour qu'on puisse faire les 1 secondes écoulée (c'est au niveau de compt) autrement dit on fait une solution pour qu'il va attendre une seconde pour qu'il va afficher.
(compt = 18 car $1 \div 55ms \simeq 18$)

iii. **La fonction Affiche:**

```
affiche PROC NEAR
mov bp,sp; BP REJOINT SP
mov dx,[bp+2]; [BP + 2] VA CONTENIR TOUJOURS L OFFSET DU MESSAGE QU ON AFFICHER
mov ah , 09h; INT POUR AFFICHER UNE CHAINE DE CARACTERES (DX; OFFSET DU PREMIER CARACTERE DE LA CHAINE
int 21h
ret
affiche endp
```

La fonction Affiche, l'idée c'est quoi ?, l'idée c'est d'au lieu affiche un seul message (À chaque fois on fait appelle à l'interruption 21h) on a trouver une fonction qui vérifie la notion du MODULARITÉ, avant d'appeler la fonction affiche on va empiler l'offset du message à afficher et à l'intérieur de la fonction on le déplier dans le registre DX.

iv. le programme Principale:

```
; PROGRAMME COMMENCE ICI >>>>>
start: mov ax , data
       mov ds , ax
       mov ax , maPile
       mov ss , ax
       lea sp, tos

       call NEAR PTR derout_1CH
       push offset dert
       call near ptr affiche
boucle:
         push offset msg2
         call near ptr affiche
         ;DELAI DU TEMPS
         mov cx, 02Bfh
         boucle_externe : mov si, 0cfffh
              boucle_interne :            dec si
                                          jnz boucle_interne

         loop boucle_externe

         push offset retLine ; RETOUR A LA LIGNE 10,13
         call near ptr affiche ;AFFICHER LE RETOUR A LA LIGNE

jmp boucle ;POUR QUE IL VA TOUJOURS AFFICHER LE MESSAGE2
code ENDS
; FIN DU CODE SEGEMENT
END start
```

Les instructions `mov ax, data` et `mov ds, ax` sont utilisées pour initialiser le registre de segment de données (`ds`) avec la valeur du registre général `ax`. Cela permet d'accéder aux données dans la section de données du programme, L'instruction `mov ax, maPile` déplace la valeur de l'étiquette `maPile` dans le registre `ax`. Cela peut être utilisé pour initialiser le registre de segment de pile (`ss`) avec cette valeur, l'instruction `mov ss, ax` initialise le registre de segment de pile (`ss`) avec la valeur contenue dans `ax`. Cela indique l'emplacement de la pile. L'instruction `lea sp, tos` calcule l'adresse effective de `tos` (sommet de la pile) et le charge dans le registre de pile (`sp`). Cela initialise la pile avec l'adresse de `tos`. L'instruction `call NEAR PTR derout_1CH` appelle une routine de sous-programme nommée `derout_1CH`. L'utilisation de `NEAR PTR` indique que l'appel de la routine est une instruction d'appel de près.
L'instruction `push offset dert` empile l'adresse de l'étiquette `dert` sur la pile. Cela prépare l'argument pour l'appel de la routine de sous-programme suivante. L'instruction `call near ptr affiche` appelle une autre routine de sous-programme nommée `affiche`. L'utilisation de `near ptr` indique que l'appel de la routine est une instruction d'appel de près.

Ensuite, nous avons une boucle étiquetée `boucle`, qui est utilisée comme point de saut pour revenir après chaque itération de la boucle. L'instruction `push offset msg2` empile l'adresse de l'étiquette `msg2` sur la pile. Cela prépare l'argument pour l'appel de la routine de sous-programme suivante.
L'instruction `call near ptr affiche` appelle la routine de sous-programme `affiche` pour afficher le message associé à `msg2`.
Ensuite, nous avons une boucle avec l'étiquette `boucle_externe`, qui est utilisée pour effectuer un délai de temps en exécutant des itérations de boucle.
L'instruction `mov cx, 02Bfh` initialise le registre de compteur de boucle (`cx`) avec la valeur `02Bfh` (695 en décimal), ce qui détermine le nombre d'itérations de la boucle externe.
À l'intérieur de la boucle externe, nous avons une boucle étiquetée `boucle_interne`, qui est utilisée pour créer un délai en effectuant des itérations de boucle.
L'instruction `dec si` décrémente la valeur du registre source-index (`si`).
L'instruction `jnz boucle_interne` effectue un saut vers l'étiquette `boucle_interne` si la valeur du registre `si` n'est pas zéro.
L'instruction `loop boucle_externe` décrémente automatiquement le registre de compteur de boucle (`cx`) et saute à l'étiquette `boucle_externe` si la valeur de `cx` n'est pas encore égale à zéro. Cela permet de répéter la boucle externe jusqu'à ce que le compteur atteigne zéro.
Ensuite, nous avons une instruction `push offset retLine` qui empile l'adresse de l'étiquette `retLine` sur la pile. Cela prépare l'argument pour l'appel de la routine de sous-programme suivante.
L'instruction `call near ptr affiche` appelle la routine de sous-programme `affiche` pour afficher un retour à la ligne.
Enfin, nous avons une instruction `jmp boucle` qui effectue un saut à l'étiquette `boucle`. Cela garantit que le programme revient toujours à la boucle, assurant ainsi l'affichage continu du message associé à `msg2`.

2. Programme 1_tache2 (5 minutes exécution):
Vous allez trouver la vidéo qui montre les 5 minutes d'exécution:

(192) PROJET ARCHITECTURE DES ORDINATEURS 2 (Programme 1_tache s'arrête après 5 min d'exécution). - YouTube

C'est la même chose avec le programme **1_tache** juste ce qu'il va changer c'est le **new:** et le rajout de la variable 'msg' et le rajout de l'étiquette 'exit'

```
    min dw 300 ; COMPTEUR DES 5 MINUTES
data ENDS
```

min = 300 car $(5 * 60 = 300)$.

```
; DEROUTEMENT DU 1CH AVEC NEW
new:
dec compt ; DEC POUR 1 SEC
jnz fin   ; IL VA ATTENDRE UNE SECONDE POUR AFFICHER
push offset msg ; MESSAGE 1
call NEAR PTR affiche
dec min
jz exit
pop bx ; POUR QU ON AURA PAS DE PROBLEME CONSERNANT LE IRET ET LA PILE
mov compt, 18 ;UNE AUTRE SECONDE
fin: iret
```

```
dec min
jz exit
```

'exit' se trouve à la fin du **CODE SEGMENT:**

```
; PROGRAMME FINI ICI >>>
exit: mov ax,4c00h
      int 21h
code ENDS
END start
```

la fonction $AH = 4ch$ de l'interruption $21h$. (Elle cause l'arrêt du programme).

2.Programme_5_tache:

A.DATA SEGMENT:

```
;DATA SEGMENT
data segment
    dert db "Deroutement fait....",10,10,13,"$" ; MESSAGE 1
    tache_p1 db "Tache $" ; MESSAGE 2
    tache_p2 db " est en cours d'execution",10,13,"$" ; MESSAGE 3
    numTache db "1$"
    compt dw 91 ; compteur de boucle pour afficher un msg chaque 5sec
    retLine db 10,13,"$"
data ends
```

B.STACK SEGMENT:

```
;;STACK SEGMENT
maPile segment stack
    dw 128 dup(?)
    tos label word
maPile ends
```

C.CODE SEGMENT:

```
;CODE SEGEMENT
code SEGMENT
    assume CS:code, DS: data, SS: maPile
```

i.Fonction derout_1ch

```
;LA FONCTION QUI FAIT LE DEROUTEMENT GRACE L'FONCTION AH=25h DU INT 21H
derout_1CH PROC NEAR
 ; 25h/21h installer un nouveau vecteur ; ES/ AL : numero du vect a installer ; DS:DX: les adresses cs et ip de la nouvelle routine d it
 push ds ;SAUVGARDER LE DS DANS LA PILE AVANT D'INSTALLER LE VECTEUR
 mov ax , seg new
 mov ds , ax    ; CSnew DANS DS
 mov dx , offset new ; IPnew DANS DX
 mov ax , 251CH ;MOV AH,25H; MOV AL,1CH;
 int 21H
 pop ds
 ret
derout_1CH ENDP
```

Grâce à l'interruption 21h (AH=25h) on a pu installer un vecteur qui va faire le travail de new, et dans ce cas on sera obligé de sauvegarder le DS dans la pile car il va changer sa valeur. (cette interruption, elle a besoin de DS(SEGMENT NEW), DX(OFFSET NEW), AL (Numéro d'interruption).

ii.Fonction d'affichage:

```
    ;FONCTION D'AFFICHAGE DE TOUS LES MESSAGES DU PROGRAMME
affiche PROC NEAR
    mov bp,sp
    mov dx,[bp+2]
    mov ah , 09h
    int 21h
    ret
affiche endp
```

Cette petite procédure est utilisée pour afficher une chaîne de caractères sur la console.

L'idée est d'initialiser le pointeur de base (BP) à la valeur du pointeur de pile (SP), qui est un registre qui pointe vers le sommet de la pile. Ensuite, charge l'adresse de la chaîne de caractères à afficher depuis le deuxième paramètre passé par la fonction appelante, qui est situé à deux octets à partir de BP. L'instruction "mov ah,09h" charge la valeur 09h dans le registre AH, qui est pour l'affichage d'une chaîne de caractères.

iii.Fonction afficheTache:

```
; PROCEDURES POUR AFFICHER TOUS LES MESSAGES DES TACHES // ELLE VA JUSTE CHANGER LES VALEURS DES TACHES
        afficheTache PROC NEAR
            cmp numTache,'6'
            jne debut
            mov numTache,'1'

            push offset retLine
            call near ptr affiche
            pop bx

            debut:
                ;AFFICHAGE MESSAGE 2
                push offset tache_p1
                call near ptr affiche
                pop bx

                ;

                ;CHANGEMENT DES VALEURS 1-->2-->3-->4-->5
                push offset numTache
                call near ptr affiche
                pop bx
                inc numTache

                ;AFFICHAGE MESSAGE 3
                push offset tache_p2
                call near ptr affiche
                pop bx
                ret
            afficheTache endp
```

Cette procédure affiche une tâche numérotée sur la console, le but est d'organiser l'affichage des 5 tâches par ordre.
En debut on compare la valeur de la variable numTache à la valeur ASCII de '6'. Si elles sont égales, le code saute à l'étiquette "debut".L'étiquette "debut" est appelle la procédure "affiche" avec un pointeur vers la chaîne de caractères "tache_p1", qui est une chaîne de caractères représentant le début de la tâche. Ensuite numTache est affichée en appelant la procédure "affiche" pour indiquer le nombre de tâche. La variable numTache est ensuite incrémentée(pour la tâche suivante, et la procédure appelle à nouveau la procédure "affiche" avec la chaîne de caractères "tache_p2", qui est une chaîne de caractères représentant la fin de la tâche.La deuxième instruction "mov numTache,'1'" charge la valeur ASCII de '1' dans la variable numTache si la variable numTache est égale à 6(fin des tâches). Ensuite, la procédure appelle la procédure "affiche" avec la chaîne de caractères "retLine", qui est une chaîne de caractères de retour à la ligne.

iv. New:

```
  ;DEROUTEMENT DE LA FONCTION 1CH par new
  new:
       dec compt
       jnz fin ; Si compt n'est pas egal,zero, saute au fin
       call NEAR PTR afficheTache ; sinon fait appelle la procedure affiche
       mov compt , 91 ; Reinitialise la variable compt 91
  fin: iret
  ;; LE PROGRAMME COMMENCE D'ICI >>>>>>>
```

dans **new:** on fait la décrémentation pour qu'on puisse faire les 5 secondes entre les 5 tâches (compt = 91 car $91 \div 18.2ms \simeq 5s$)
la procédure "afficheTache" est appelée, ce qui affiche une tâche numérotée sur la console. L'instruction "jnz fin" saute à l'étiquette "fin" si la valeur de la variable "compt" est différente de zéro (affiche rien).

v.Programme principale:

```
start:
       mov ax , data
       mov ds , ax
       mov ax , maPile
       mov ss , ax
       lea sp, tos

       call NEAR PTR derout_1CH ; Appelle la procedure derout_1CH pour installer un nouveau vecteur d'interruption

       push offset dert      ;AFFICHAGE DE MESSAGE 1
       call near ptr affiche
       pop bx  ;POUR PAS AVOIR DES PROBLEMES AU NIVEAU DU STACK SEGMENT

       boucle1:jmp boucle1  ; Saute boucle pour boucler infiniment
code ENDS
END start
```

Le programme principal de ce code commence par initialiser les segments de données, de pile et de code avant d'appeler la procédure "derout_1CH" qui va installer un nouveau vecteur d'interruption. Ensuite, le programme affiche le message "Deroutement fait...." au début d'exécution. Puis, il entre dans une boucle infinie pour que le programme ne s'arrête pas. Cependant, toutes les 5 secondes, le vecteur d'interruption nouvellement installé interrompt le programme principal et appelle la procédure "afficheTache". Qui va simplement afficher les messages correspondants aux différentes tâches (de 1 à 5) et qui sera affichée à l'écran pour indiquer quelle tâche est en cours d'exécution.

4. Programme 5_tache2 (5 minutes exécution):

Vous allez trouver la vidéo qui montre les 5 minutes d'exécution:
(213) PROJET ARCHITECTURE DES ORDINATEURS 2 (Programme 5_tache s'arrête après 5 min d'exécution). - YouTube

C'est la même chose avec le programme **5_tache** juste ce qu'il va changer c'est le **new:** et le rajout de l'étiquette 'exit'.

```
min dw 60 ; COMPTEUR POUR LES 5 MIN
```

min = 60 car $(5 * 12 = 300)$.(60 / 5 = 12)

```
; PROCEDURE POUR AFFICHER UNE SPECIFIQUE TACHE
        new:
        dec compt
        jnz fin ; Si compt n'est pas egal,zero, saute au fin

        call NEAR PTR afficheTache ; sinon fait appelle la procedure affiche
        dec min ;POUR LES 5 MINUTES
        jz exit

        mov compt , 91 ; Reinitialise la variable compt 91  (5s)
        fin: iret
```

'exit' se trouve à la fin du **CODE SEGMENT:**

```
; PROGRAMME FINI ICI >>>
exit: mov ax,4c00h
      int 21h
code ENDS
END start
```

la fonction $AH = 4ch$ de l'interruption $21h$. (Elle cause l'arrêt du programme).