Chapitre 3

ARCHITECTURES DES ORDINATEURS II

Les Entrées / Sorties
E/S ou I/O

Entrées / Sorties (Input / Output):

Les opérations d'entrées /Sorties permettent l'échange de données entre le microprocesseur et les périphériques à travers des circuits d'interface d'entrées/sorties (des contrôleurs d'E/S).

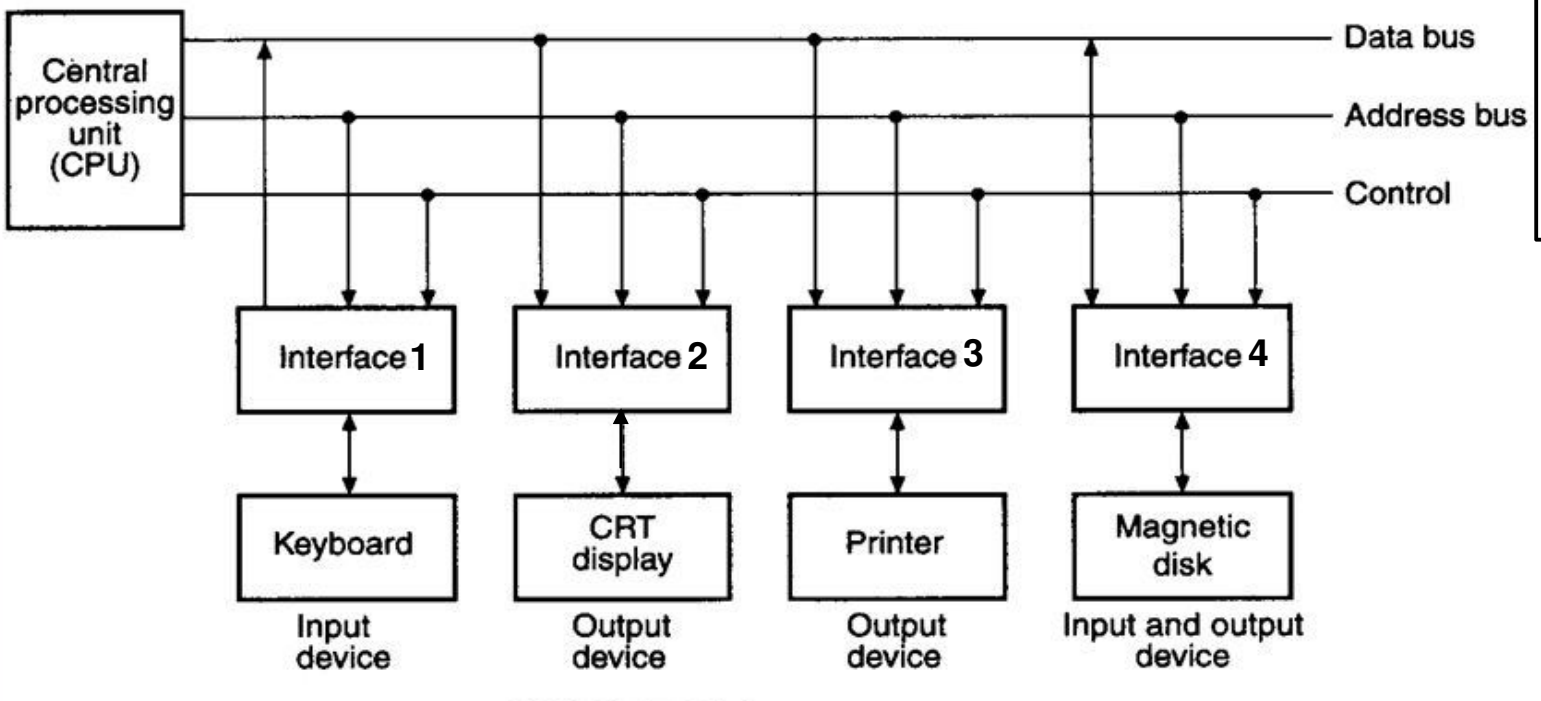
Circuit d'interface (C.I):

C'est un circuit intelligent et programmable, il est spécifique à un type particulier de périphérique. Sa fonction principale est de permettre l'échange de données entre le microprocesseur et un type donné de périphérique et aussi contrôler ce transfert.

caractéristiques des C.I :

Intelligent : implique plusieurs modes de fonctionnement
Programmable : ceci signifie que chacun des modes de fonctionnement s'obtient par de simples instructions (codage)

CPU Connection to I/O Devices



M.C

CIRCUITS D'INTERFACES VUS PAR LE MICROPROCESSEUR

Le processeur voit un circuit d'interface comme étant un ensemble de registres appelés **ports d'E/S** (I/O ports):

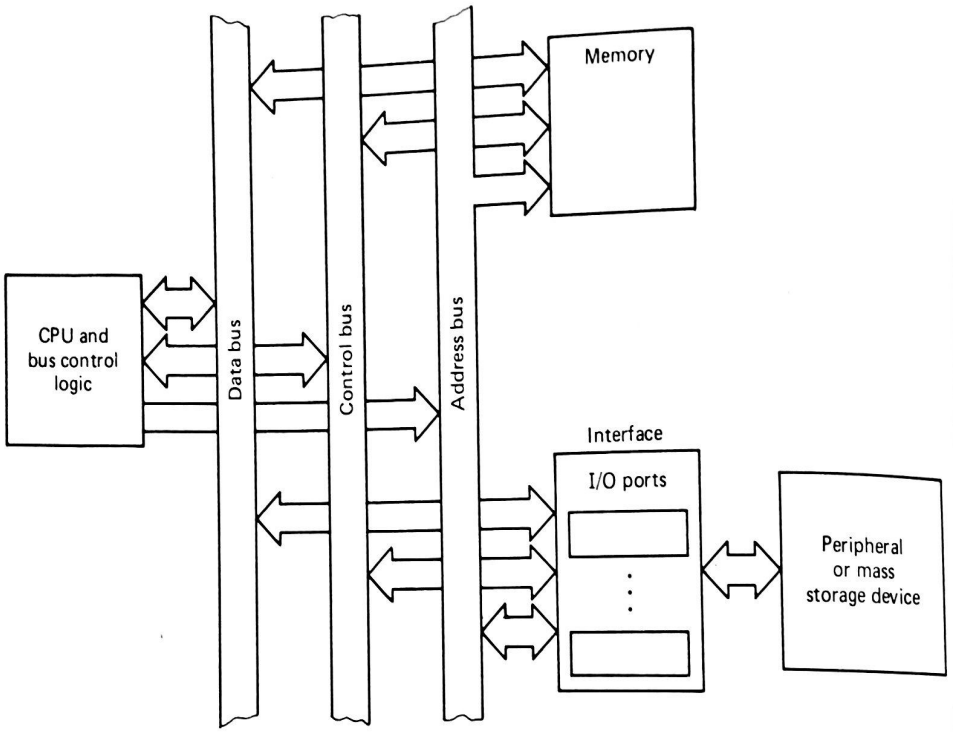
En général, il existe quatre types de ports: *(il s'agit d'une classification académique)*
Chaque port est caractérisé par une adresse et un contenu

Port d'entrée : c'est le port où le périphérique d'entrée (exemple le clavier) place la donnée en vue d'être lue par le processeur. Ce port est à lecture.

Port de sortie : c'est le port où le processeur place la donnée en vue d'être acheminée au périphérique de sortie (exemple écran). Ce port est à écriture.

Port d'état : c'est dans ce port où le processeur trouve tous les états du C.I. Il s'agit d'un port de lecture.

Port de contrôle : le C.I est programmé pour un mode de fonctionnement donné dans le port de contrôle. C'est un port d'écriture par le processeur.



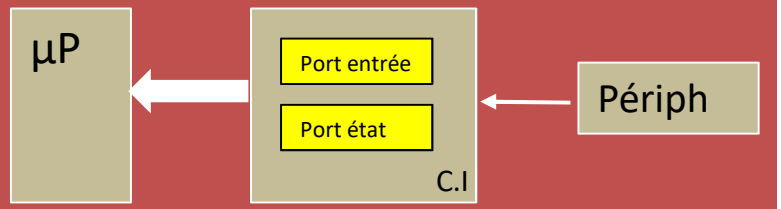
TYPES D'OPERATIONS D'E/S

Il existe 2 types d'opérations:

IN : pour la lecture d'un port
OUT : pour l'écriture dans un port

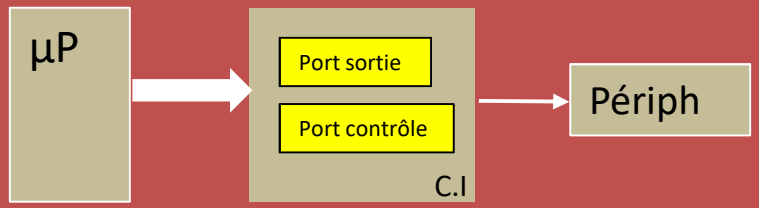
Ports à lecture:

Port d'entrée
Port d'état



Ports à écriture:

Port de sortie
Port de contrôle



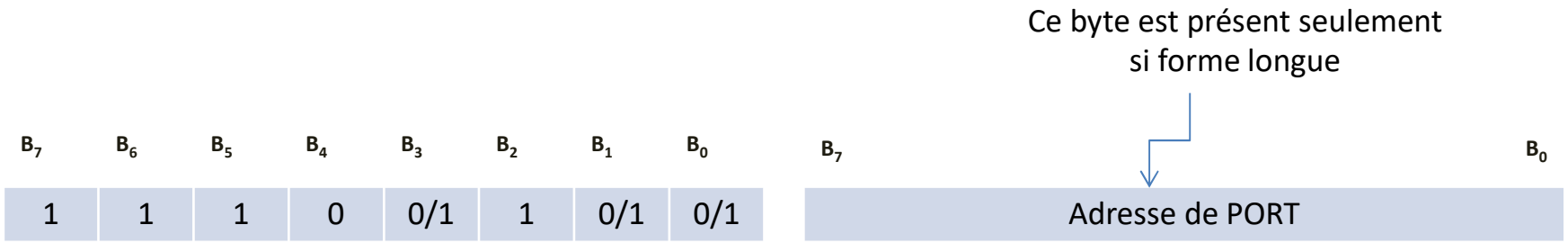
ETUDE DES INSTRUCTIONS IN ET OUT

instruction	forme	syntaxe	description
IN	Forme longue, transfert Byte	IN AL , Port	(AL) ← (Port)
	Forme longue, transfert word	IN AX , Port	(AX) ← (Port+1:Port)
	Forme courte, transfert Byte	IN AL , DX	(AL) ← ((DX))
	Forme courte, transfert word	IN AX , DX	(AX) ← ((DX)+1:(DX))
OUT	Forme longue, transfert Byte	OUT Port , AL	(Port) ← (AL)
	Forme longue, transfert word	OUT Port , AX	(Port+1:Port) ← (AX)
	Forme courte, transfert Byte	OUT DX , AL	((DX)) ← (AL)
	Forme courte, transfert word	OUT DX , AX	((DX)+1:(DX)) ← (AX)

* Port est une adresse d'E/S (donc une constante) sur 8 bits allant 00H à FFH

** Aucun flag n'est affecté par les instructions IN et OUT

Code machine de IN et OUT



$B_0 = 0$ transfert byte
$B_0 = 1$ transfert word

$B_1 = 0$ IN
$B_1 = 1$ OUT

$B_3 = 0$ forme Longue
$B_3 = 1$ forme courte

ETUDE COMPARATIVE ENTRE LA FORME LONGUE ET LA FORME COURTE POUR LES INSTRUCTIONS IN ET OUT :

	Forme Longue	Forme Courte
Espace d'adressage des ports	256 adresses de port de 00H à FFH	65536 adresses de port de 0000H à FFFFH
Taille code machine	2 bytes	1 byte
Temps d'exécution	10 cycles machine	8 cycles machine

Exemple de code machine:

in al , 60h	E4 60
out dx , ax	EF

Exemple:

Soit le circuit d'interface d'un périphérique d'entrée ayant les caractéristiques suivantes:

@port d'état : 60H
@port d'entrée : 61H

Le bit B3 du port d'état : 0 signifie que le port d'entrée est vide
1 signifie que le port d'entrée est plein

- le port d'entrée vide veut dire le périphérique d'entrée n'a pas envoyé de donnée au circuit d'interface.

- le port d'entrée plein veut dire le périphérique d'entrée a envoyé une donnée au circuit d'interface, celle-ci est dans le port d'entrée est attend d'être lue par le processeur.

Question :

Coder en assembleur l'opération de lecture d'une donnée de ce périphérique d'entrée et son stockage dans la mémoire à l'adresse usthb.

Solution :

```
Attendre:   In al , 60h     ; lire le port d'état
            test al , 8     ; test est équivalente à l'instruction and sans conserver le résultat , les flags sont affectés
            Jz attendre     ; tester si B3 du port d'état = 0, si oui attendre
            in al , 61h     ; saisir la donnée envoyée par le périphérique
            mov usthb , al
```

TECHNIQUES D'ENTREES /SORTIES

Il existe 3 techniques :

- **E/S programmées (le polling) :**
  les E/S sont réalisées par scrutation, c'est le microprocesseur qui initie l'opération d'E/S. *(sera développé dans ce chapitre)*

- **E/S par interruption :**
  dans cette technique c'est le périphérique à travers son C.I qui initie l'opération par l'envoi d'une interruption au microprocesseur. *(sera développé à la fin de ce chapitre)*

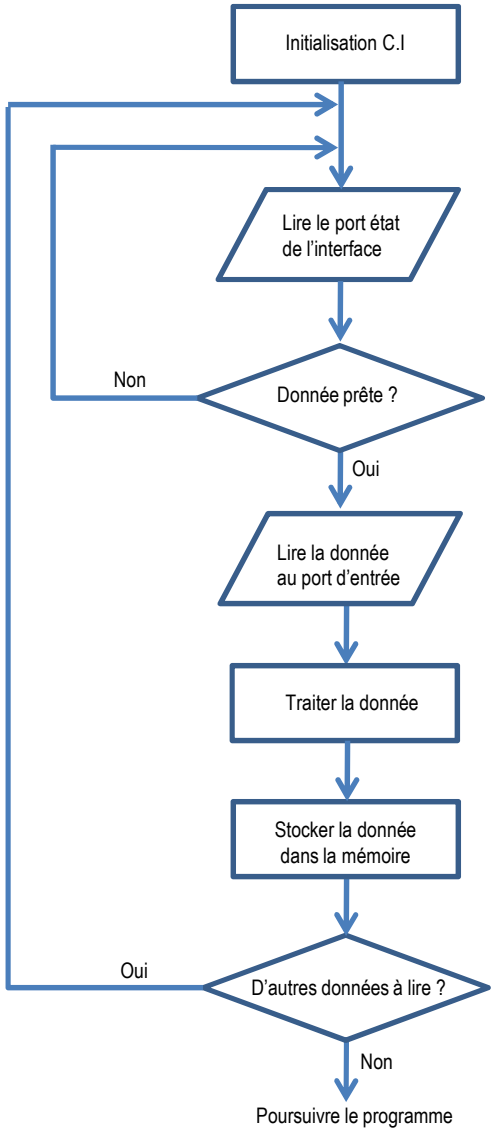
- **E/S transfert par blocs :**
  c'est le contrôleur DMAC (Direct Memory Access), en s'allouant les bus du système, contrôle et réalise l'échange de données entre la mémoire centrale et un périphérique donné sans le recours au microprocesseur. *(sera développé dans le chapitre 6 si le temps le permet)*

E/S programmées (le polling ou méthode d'E/S par scrutation)

La technique d'E/S programmée (programmed I/O) consiste à examiner continuellement l'état de l'interface et effectuer une opération d'E/S quand celle-ci est prête.

Exemple d'un organigramme d'une possible technique d'entrée

Phase 1 : initialisation du C.I en mode E/S programmée (polling) en programmant le port de contrôle
Phase 2 : scruter le C.I pour l'arrivée d'une nouvelle donnée en testant le port d'état
Phase 3 : en absence d'une nouvelle donnée, continuer à scruter le C.I
Phase 4: si nouvelle donnée, lire cette donnée au port d'entrée
Phase 5 : traiter cette donnée dans le processeur
Phase 6: sauvegarder cette donnée en mémoire
Phase 7: d'autres données à lire ?
si oui aller à la phase 2
sinon continuer le traitement du programme



REFLEXION : Dresser un organigramme d'une possible technique de sortie

EXEMPLE DU CODAGE DE LA TECHNIQUE D'ENTREE PAR POLLING

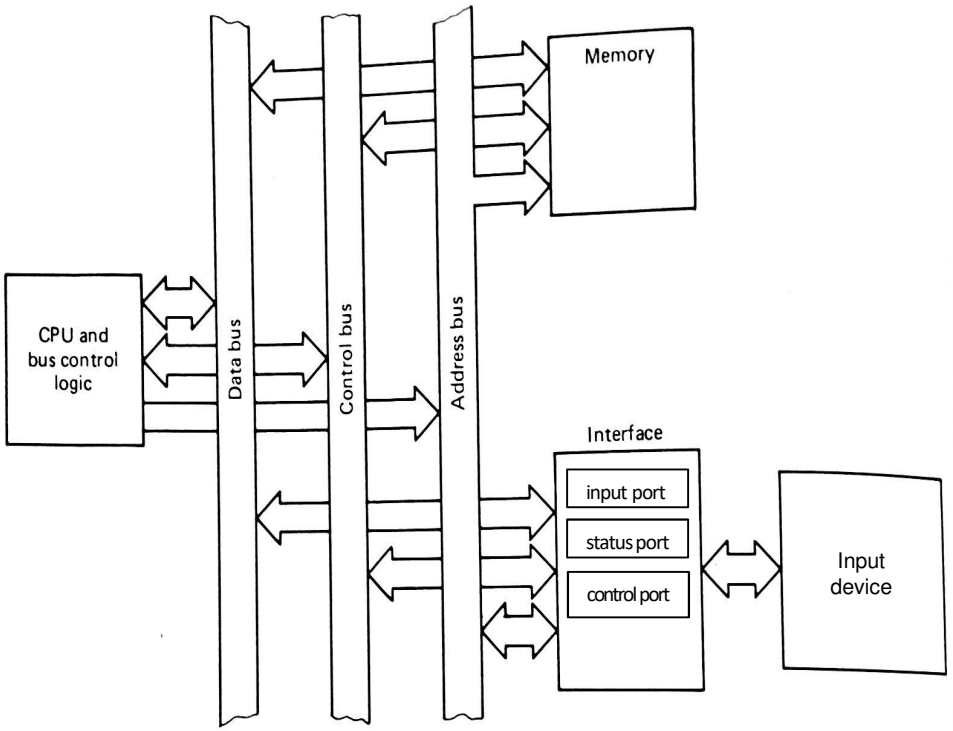
ENONCE:

@ port d'entrée = 03BCH , @ port d'état = 03BDH
@port de contrôle = 03BEH

BIT B_6 du port d'état = 0 port d'entrée vide sinon plein

BIT B_3 du port de contrôle = 1 le C.I fonctionne en mode polling sinon en E/S par IT.

Le périphérique d'entrée envoie 255 données, chacune sur 8 bits, que le processeur doit saisir et stocker en mémoire à partir de la case mémoire mem_in.




```
data            SEGMENT
mem_in    db    255 dup(?)
data            ends

code            SEGMENT
assume  cs: code , ds: data

start:          mov  ax , data
                mov ds , ax

                lea BX , mem_in
                mov cx , length mem_in

; initialisation du C.I en mode polling
                mov dx , 03BEH   ; port de contrôle
                in al , dx
                or al , 00001000B
                out dx , al

;  scrutation du port d'entrée
                dec dx      ; port d'état
attendre:       in al , dx
                test al , 01000000B
                jz attendre

; lecture de la donnée
                dec dx      ; port d'entrée
                in al , dx
                mov  [BX] , al
                inc BX
                inc DX
                loop attendre

; terminer le programme
                mov ax , 4c00h
                int  21H
Code            ends
                end     start
```

REMARQUES IMPORTANTES :

✓ POUR SAISIR UNE NOUVELLE DONNEE ENVOYEE PAR UN PERIPHERIQUE D'ENTREE, ON TESTE AU PREALABLE, DANS LE PORT D'ETAT, **L'ETAT PLEIN** DU PORT D'ENTREE AVANT LA LECTURE DE LA DONNEE DU PORT D'ENTREE

✓ POUR ENVOYER UNE NOUVELLE DONNEE A UN PERIPHERIQUE DE SORTIE, ON TESTE AU PREALABLE, DANS LE PORT D'ETAT, **L'ETAT VIDE** DU PORT DE SORTIE AVANT L'ECRITURE DE LA DONNEE DANS LE PORT DE SORTIE.

EXEMPLE SIMPLIFIE D'UNE OPERATION DE SORTIE EN MODE POLLING

Soit un C.I d'un périphérique de sortie ayant les caractéristiques suivantes:
@ port de sortie : 36H
@port d'état : 37H

Le bit B_4 du port d'état = 1 signifie que le port de sortie est vide sinon il est plein.

Question: envoyer à ce périphérique de sortie, en mode polling, la donnée 8 bits se trouvant dans la case mémoire usthb.

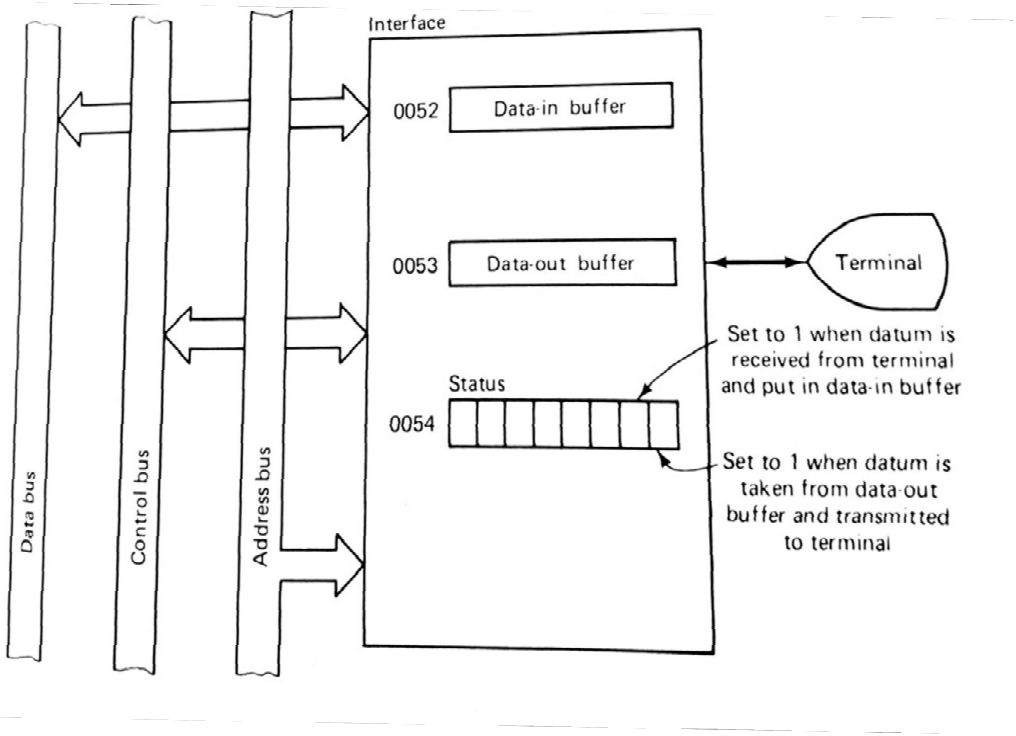
```
not_yet:    in    al , 37H    ;  attendre l'état vide du port de sortie
            Test  al , 10H
            Jz    not_yet
            mov   al , usthb  ;  écriture de la donnée dans le port de sortie
            out   36H , al
```

EXEMPLE COMPLET D'UN PROGRAMME D'E/S PROGRAMMEES

```
DATA_SEG        SEGMENT
    MESSAGE  DB  'BUFFER  OVERFLOW' , 0DH , 0AH
                 .
                 .
                 .
DATA_SEG        ENDS

COM_SEG         SEGMENT
    BUFFER      DB                  82  DUP(?)
    COUNT       DB                  ?
COM_SEG         ENDS
                 .
                 .
                 .
    IN_BUFF     EQU                 52H
    OUT_BUFF    EQU                 53H
    STATUS      EQU                 54H
    RRDY        EQU                 00000010B
    TRDY        EQU                 00000001B

CODE_SEG        SEGMENT
ASSUME  CS: CODE_SEG , DS: DATA_SEG , ES: COM_SEG
                MOV     AX , DATA_SEG
                MOV     DS , AX
                MOV     AX , COM_SEG
                MOV     ES , AX
                .
                .
                .
                MOV     DI , OFFSET BUFFER
                MOV     COUNT, DI
                MOV     CX, 81
                CLD
NEXT_IN:        IN      AL, STATUS
                TEST    AL, RRDY
                JZ      NEXT_IN
                IN      AL, IN_BUFF
                OR      AL, 0
                JPE     NO_ERROR
                JMP     NEAR PRT ERROR
NO_ERROR:       AND     AL, 7FH
                STOSB
                CMP     AL, ODH
                LOOPNE NEXT_IN
                JNE     OVERFLOW
                MOV     AL, 0AH
                STOSB
                SUB     DI, COUNT
                MOV     COUNT, DI
                .
                .
                .
OVERFLOW:       MOV     SI, OFFSET MESSAGE
                MOV     CX, 17
NEXT_OUT:       IN      AL, STATUS
                TEST    AL, TRDY
                JZ      NEXT_OUT
                LODSB
                OUT     OUT_BUFF, AL
                LOOP    NEXT_OUT
                                    .
                .
                .
```



REFLEXION: COMPRENDRE CE QUE FAIT CE PROGRAMME

Gestion de plusieurs périphériques en E/S programmées

Dans le cas où plusieurs périphériques échangent des données avec le processeur en E/S programmée, l'ordre de scrutation des ports d'état de leurs C.I doit être bien choisi.

ETUDE DE CAS:

Considérons l'exemple suivant:

Trois périphériques appelés DEV1, DEV2 et DEV3 qui envoient leurs données au processeur. Les adresses des ports d'état de leurs C.I sont respectivement STATUS1, STATUS2 et STATUS3. Les taches d'entrées pour les 3 périphériques sont réalisées respectueusement par les procédures PROC1,PROC2 ET PROC3. Seul le périphérique 1 peut arrêter le processus d'entrée en forçant dans la procédure PROC1 la variable FLAG à 1, qui est initialement à zéro, . Quand Flag passe à 1, une dernière chance est accordée aux périphériques DEV2 et DEV3 pour envoyer éventuellement leurs dernières données en attente.

Les bits B5 des trois ports d'états STATUS1, STATUS2 et STATUS3 sont utilisés pour indiquer si une donnée est en attente dans leurs ports d'entrée respectifs.

L'ordre de scrutation désiré étant DEV1, DEV2 ensuite DEV3, il est répété jusqu'à ce que la variable FLAG est forcée à 1 par PROC1.

SOLUTION DU PROBLEME

```
            MOV   FLAG , 0
INPUT:      IN    AL , STATUS1
            TEST  AL , 20H
            JZ    DEV2
            CALL  FAR  PTR  PROC1
            CMP    FLAG , 1
            JNZ    INPUT
DEV2:       IN    AL , STATUS2
            TEST  AL , 20H
            JZ    DEV3
            CALL  FAR  PTR  PROC2
            CMP    FLAG , 1
            JNZ    INPUT
DEV3:       IN    AL , STATUS3
            TEST  AL , 20H
            JZ    NONE
            CALL  FAR  PTR  PROC3
NONE:       CMP    FLAG , 1
            JNZ    INPUT
              .
              .
              .
```

Discuter cette solution notamment en terme d'ordre de priorité de la scrutation (priority polling).

REFLEXION :

Transformer ce programme pour que l'ordre de la scrutation des périphériques à travers leurs C.I devienne **circulaire** (**round-robin** arrangement), et l'ordre désiré est: DEV1 DEV2 DEV3 répété jusqu'à ce que Flag = 1.

CRITIQUE DES E/S PROGAMMEES (le polling)

✓ **AVANTAGE**

METHODE TRES SIMPLE A CONCEVOIR

⊘ **INCONVENIENT**

PERTE CONSIDERABLE EN TEMPS DUE A L'ATTENTE DE L'ETAT PRÊT (READY STATE) DE L'INTERFACE.

A TITRE D'ILLUSTRATION :

Soit une secrétaire professionnelle capable de taper sur son clavier 10 caractères par seconde et seulement 10 micro-secondes sont nécessaires au processeur pour lire chaque caractère. Donc, approximativement

$[(10^5 – 10)/10^5]$ x 100% = 99,99% du temps machine est inutilisé.

UNE METHODE PLUS PERFORMANTE POUR REALISER LES E/S SERAIT DONC LA METHODE DES E/S PAR INTERRUPTION

RAPPEL : E/S programmées (polling)

les E/S sont réalisées par scrutation, c'est le microprocesseur qui initie l'opération d'E/S scrutant l'interface pour un état prêt..

E/S par interruption :

dans cette technique c'est le périphérique à travers son C.I qui initie l'opération par l'envoi d'une requête au microprocesseur quand l'interface est prête pour une opération d'E/S.

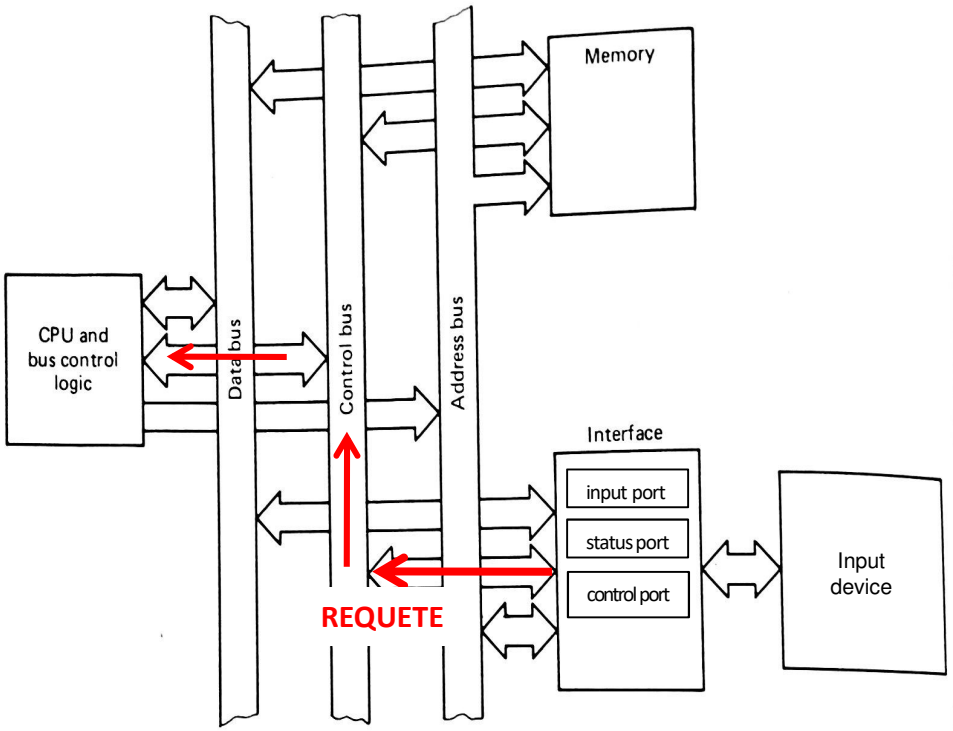
Dans cette nouvelle technique, ce n'est plus le processeur qui scrute continuellement l'interface pour un état prêt causant ainsi une perte importante en terme de temps mais c'est le périphérique à travers son circuit d'interface qui envoie automatiquement une requête au processeur quand l'interface est dans son état prêt (READY STATE).

Cette technique permet de décharger le processeur de la tache d'attente et évite au processeur le temps gaspillé inutilement lors de l'attente de l'état READY de l'interface.

EXEMPLE DU CODAGE DE LA TECHNIQUE D'ENTREE PAR INTERRUPTION

ENONCE:

@ port d'entrée = 03BCH , @ port d'état = 03BDH
@port de contrôle = 03BEH
BIT B_6 du port d'état = 0 port d'entrée vide sinon plein
BIT B_3 du port de contrôle = 1 le C.I fonctionne en mode polling sinon en E/S par IT.
Le périphérique d'entrée envoie 255 données, chacune sur 8 bits, que le processeur doit saisir et stocker en mémoire à partir de la case mémoire mem_in. Pour chaque nouvelle donnée, le processeur reçoit une requête asynchrone du C.I l'informant qu'une nouvelle donnée est disponible dans le port d'entrée.




```
data            SEGMENT
mem_in    db      255 dup(?)
data            ends

code            SEGMENT
assume  cs: code , ds: data

;  routine d'IT lecture d'une donnée
                mov dx , 03BCH      ; port d'entrée
                in al , dx
                mov  [BX] , al
                inc BX
                IRET

start:          mov  ax , data
                mov ds , ax

                lea BX , mem_in
                mov cx , length mem_in

; initialisation du C.I en E/S par IT
                mov dx , 03BEH   ; port de contrôle
                in al , dx
                and al , 11110111B
                out dx , al
                .
                .
                .
Code            ends
                end     start
```

Remarque: Beaucoup de simplifications dans ce programme telles que les omissions de l'installation du vecteur d'IT de la requête, la gestion du registre BX et autres.