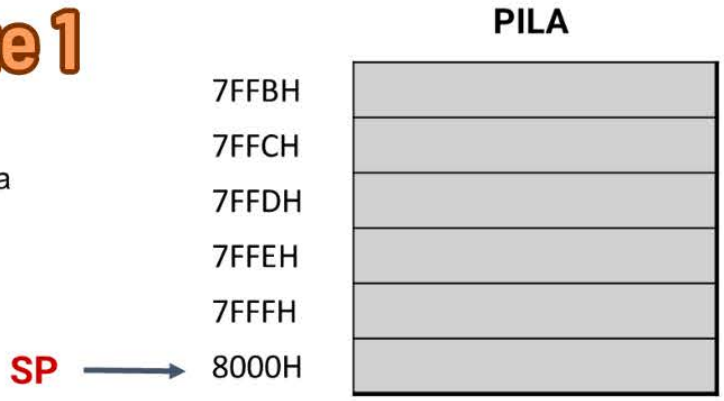
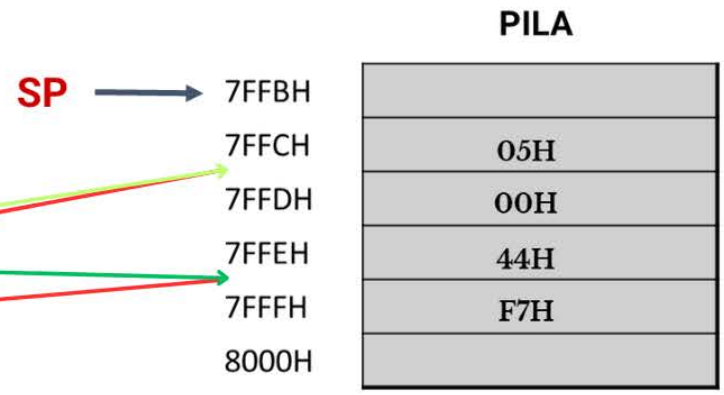
PILA
parte 1

- PUSH: apilaba un elemento de 16 bits en la pila
- POP: desapilaba un elemento de 16 bits desde la pila




```
ORG 1000H
    ... ; Definición de mis variables

ORG 2000H
    MOV AX, F744H    ; AX = F744H
    MOV BX, 5        ; BX = 5
    PUSH AX
    PUSH BX
    POP AX
    POP BX
    HLT
    END
```



¡Acabamos de hacer un *swap* (intercambio) entre AX y BX!

```
ORG 1000H
    ... ; Definición de mis variables
```
DATOS

```
ORG 3000H
    ... ; Definición de subrutina
```
SUBRUTINAS

```
ORG 2000H
    ... ; Programa principal
    HLT
    END
```
PROG. PRINC.

Se llaman con la sentencia **CALL**
Se vuelve de ellas con la sentencia **RET**

Sin subrutinas

```
ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, NUM1
    MOV BX, NUM2
        MOV CX, 0 ; Por el momento el resultado es 0
        LOOP: ADD CX, BX
        DEC AX
        JNZ LOOP
    HLT
END
```

Con subrutinas

```
ORG 3000H
        MUL: MOV CX, 0 ; Por el momento el resultado es 0
        LOOP: ADD CX, BX
        DEC AX
        JNZ LOOP
    RET

ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, NUM1
    MOV BX, NUM2
    CALL MUL
    MOV RES, CX
    HLT
        END
```

Podemos usar la subrutina cuantas veces queramos!

PILA
parte 2
Pasaje de parámetros

En el ejercicio anterior pasábamos el **valor** a través de **registros**

```
ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, NUM1
    MOV BX, NUM2
    CALL MUL
    MOV RES, CX
    HLT
        END
```

Por valor por pila

```
ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, NUM1
    MOV BX, NUM2
    ; Apilamos antes de llamar
    PUSH AX
    PUSH BX
    CALL MUL
    MOV RES, CX
    HLT
        END
```

Pero podríamos pasar la **referencia** a través de **registros**

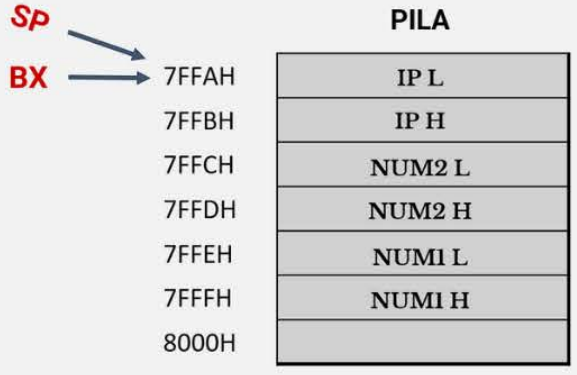
```
ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, OFFSET NUM1
    MOV BX, OFFSET NUM2
    CALL MUL
    MOV RES, CX
    HLT
        END
```

Por referencia por pila

```
ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, OFFSET NUM1
    MOV BX, OFFSET NUM2
    ; Apilamos antes de llamar
    PUSH AX
    PUSH BX
    CALL MUL
    MOV RES, CX
    HLT
        END
```

Recuerden que cuando hacemos un **CALL** se apila el IP. Por lo que si hacemos un POP estamos tomando su valor y no nos interesa!

```
ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, NUM1
    MOV BX, NUM2
    ; Apilamos antes de llamar
    PUSH AX
    PUSH BX
    CALL MUL
    MOV RES, CX
    POP AX
    POP BX
    HLT
        END
```




```
MOV BX, SP      ; BX = SP
ADD BX, 2       ; Posiciono en NUM2
MOV DX, [BX]    ; Tomo NUM2 en DX
ADD BX, 2       ; Posiciono en NUM1
MOV AX, [BX]    ; Tomo NUM1 en AX
LOOP: ADD CX, DX
        DEC AX
        JNZ LOOP
RET
```

PILA
parte 3

Pasaje de parámetros

Por pila y referencia

Ahora, además de lo anterior, hay que tener en cuenta que lo que estamos tomando de la pila son **direcciones** en vez de valores!

```
ORG 2000H
    ; Inicializamos AX y BX
    MOV AX, OFFSET NUM1
    MOV BX, OFFSET NUM2
    ; Apilamos antes de llamar
    PUSH AX
    PUSH BX
    CALL MUL
    MOV RES, CX
    POP AX
    POP BX
    HLT
        END
```

SP → 7FFAH

	PILA
7FFAH	IP L
7FFBH	IP H
7FFCH	DIR NUM2 L
7FFDH	DIR NUM2 H
7FFEH	DIR NUM1 L
7FFFH	DIR NUM1 H
8000H	

```
ORG 3000H
MUL: MOV BX, SP        ; BX = SP
ADD BX, 2              ; Posiciono en DIR de
NUM2
MOV AX, [BX]      ; AX = Dir de NUM2
MOV DX, BX             ; Backup de BX
MOV BX, AX        ; BX = Dir de NUM2
MOV AX, [BX]      ; AX = NUM2
MOV BX, DX             ; Recupero puntero de
pila
[...]                                  ;
Repetimos con NUM1
[...]                                  ; Seguimos
como siempre
RET
```

INTERRUPCIONES
por software

Las **interrupciones por software** nos permiten invocar algunas funciones básicas durante la ejecución de nuestro programa principal

Tenemos 4:

INT 0: detiene el programa. Igual al **HLT**

INT 3: debug. No lo vamos a utilizar

INT 6: lee un caracter desde teclado

INT 7: imprime un string en pantalla

Como dijimos, **INT 0** detiene la ejecución del programa

Es equivalente a lo que conocíamos como **HLT**

```
ORG 1000H
    NUM1 DW 2
    NUM2 DW 8
    RES  DW ?

ORG 2000H
    MOV AX, NUM1
    MOV CX, NUM2
    ADD AX, CX
    MOV RES, AX
    INT 0
        END
```

INT 6 lee un caracter desde teclado
Cuando invocamos la interrupción se guarda el caracter leído en la **dirección** que contiene en ese momento **BX**
Escribir un programa que lea un caracter y lo guarde en la variable *LEIDO*

```
ORG 1000H
    LEIDO DB ?

ORG 2000H
    MOV BX, OFFSET LEIDO   <-
    INT 6                  <-  (presiona la "a")
    INT 0                  <-
        END
```

LEIDO: ~~Basura~~ 61H
BX: ~~Basura~~ 1000H

INT 7 imprime un string en pantalla
Esta interrupción necesita dos cosas: la **dirección** en **BX** desde donde empieza a leer y cuántos caracteres va a imprimir en **AL**

Escribir un programa que imprima la cadena "*Arquitectura de computadoras*" en pantalla

```
ORG 1000H
    MENSAJE  DB "Arquitectura de computadoras"
    FIN      DB ?

ORG 2000H
    MOV BX, OFFSET MENSAJE
    MOV AL, OFFSET FIN - OFFSET MENSAJE
    INT 7
    INT 0
        END
```

BX: ~~Basura~~ 1000H
AL: ~~Basura~~ 1CH (24)

INTERRUPCIONES
por software

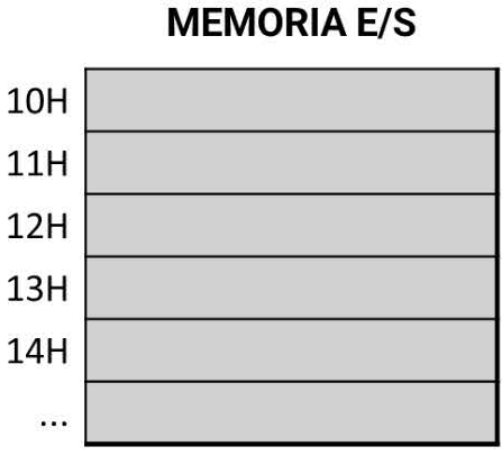
INT 0, INT 6 e INT 7

Escribir un programa que lea 10 caracteres y cuando termine la lectura imprima la cadena completa en pantalla

```
ORG 1000H
    MENSAJE   DB "Ingrese 10 caracteres!"
    FIN       DB ?
    CADENA    DB ?

ORG 3000H
; Subrutina que imprime consigna en la pantalla
PRINT_MSG: MOV BX, OFFSET MENSAJE
    MOV AL, OFFSET FIN - OFFSET MENSAJE
    INT 7
    RET
```

```
ORG 2000H
    CALL PRINT_MSG ; Imprimimos mensaje
    MOV DL, 10 ; Cantidad de caracteres a leer
    MOV BX, OFFSET CADENA ; Donde vamos a insertar lo leido
    LEER: INT 6
        INC BX ; Proxima posicion en la memoria
        DEC DL
        JNZ LEER

    ; Imprimimos lo leido
        MOV BX, OFFSET CADENA
    MOV AL, 10
    INT 7
    INT 0
    END
```

MEMORIA E/S



Si son iguales necesito un mecanismo que permita distinguirlas!

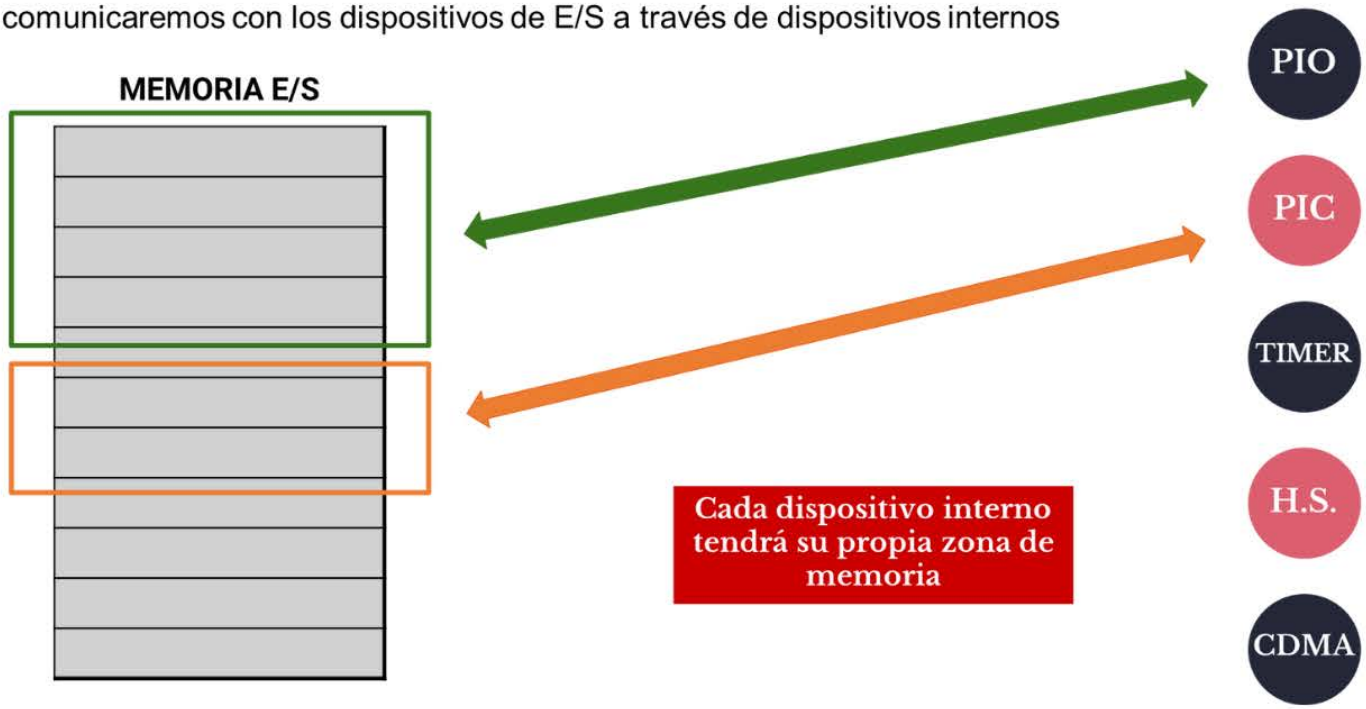
- Para leer desde la memoria E/S usaremos **IN**, para escribir en ella **OUT**. Ambas instrucciones **solo se pueden usar** con el registro *AL*
- Ej. lectura: leer el dato que está en la posición 40H de E/S

  IN *AL, 40H*
- Ej.escritura: poner el valor 30 en la posición 50H de E/S

  ~~***OUT*** *50H, 30*~~

  MOV AL, 30
  OUT *50H, AL*

Nos comunicaremos con los dispositivos de E/S a través de dispositivos internos

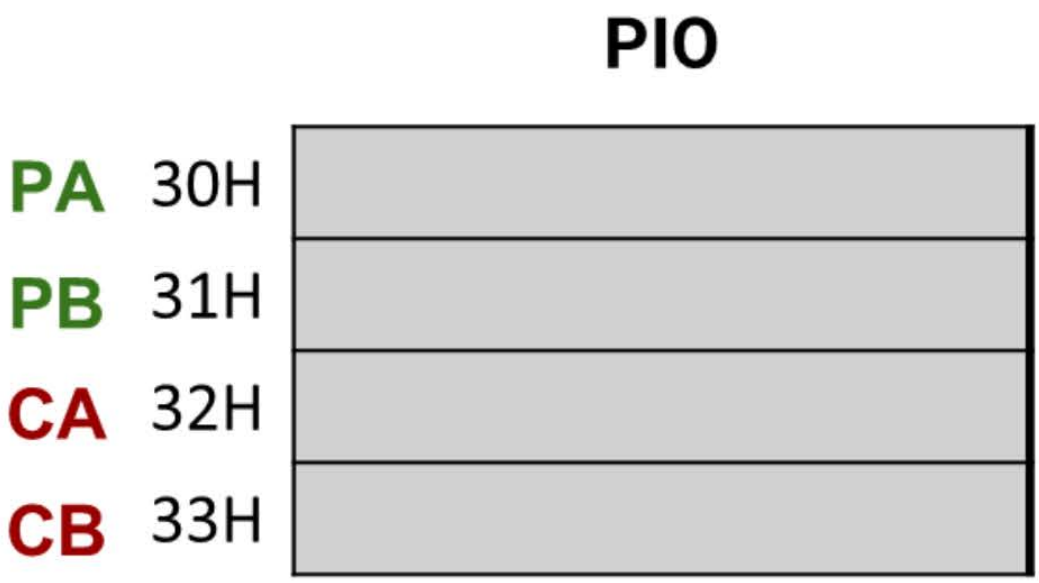


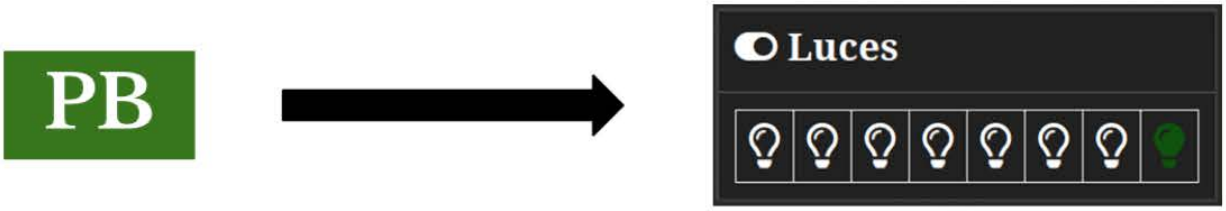
PIO
Estructura

Consta de 2 puertos paralelos configurables

Ocupa 4 celdas en la memoria de E/S:

- 2 de **datos** llamados PA y PB
- 2 de **configuración** llamados CA y CB (ya veremos para qué sirven)





PIO
Funcionamiento

Los puertos funcionan de la siguiente manera

- Cada celda (también llamado *registro*) consta de 8 bits
- Debemos **configurar** cada bit de **datos** como entrada o salida
- En los puertos de **configuración** debemos poner un 0 para que ese bit en el puerto de **datos** sea de salida, 1 para que sea de entrada

Ej.: queremos que el **PA** tenga todos los bits como entrada excepto el menos significativo

- Debemos configurar **CA**
- Todos en 1 excepto el menos significativo (11111110)

```
1   ORG 2000H
2
3   ; CONFIGURAR PA (ENTRADA) Y PB (SALIDA)
4   MOV AL, 1111111b
5   OUT 32h, AL ; CA = 1111111
6   MOV AL, 00000000b
7
8   ; LEEMOS EL ESTADO DE LAS LLAVES (PA)
9   LOOP: IN AL, 30H
10
11  ;MANDAR LA INFORMACION DE LAS LLAVERS
12  OUT 31H, AL
13
14  JMP LOOP
15
16  INT 0
17  END
18
```

PIO
Entrada

Leer el estado de las llaves y prender las luces de aquellas llaves que estén en 1. Recuerden:

- Las llaves están ligadas al puerto **PA**. Las luces al **PB**.
- Queremos todos los bits de **PA** de entrada y todos los de **PB** de salida!

1. Configuramos **PA** y **PB**

```
MOV AL, 11111111b
OUT 32H, AL ; CA = 11111111
MOV AL, 00000000b
OUT 33H, AL ; CB = 00000000
```

2. Leemos **PA**

```
IN AL, 30H
```

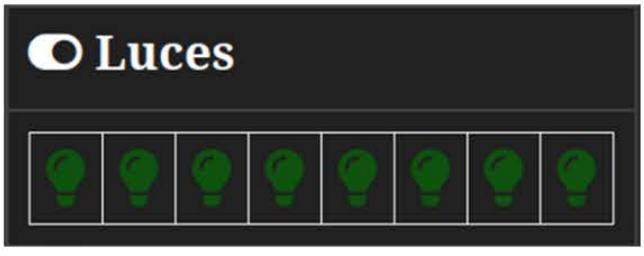
Solo queda hacerlo infinitas veces!

PIO
Salida

1. Prender todas las luces. Recordar que:

- Las luces están ligadas al puerto **PB**. 1 significa encendida
- Las queremos a todas de salida!

```
MOV AL, 00000000b
OUT 33H, AL ; CB = 00000000
MOV AL, 11111111b
OUT 31H, AL ; PB = 11111111
```



INTERRUPCIONES por HARDWARE

Nos vamos a manejar con 4 dispositivos externos

F10

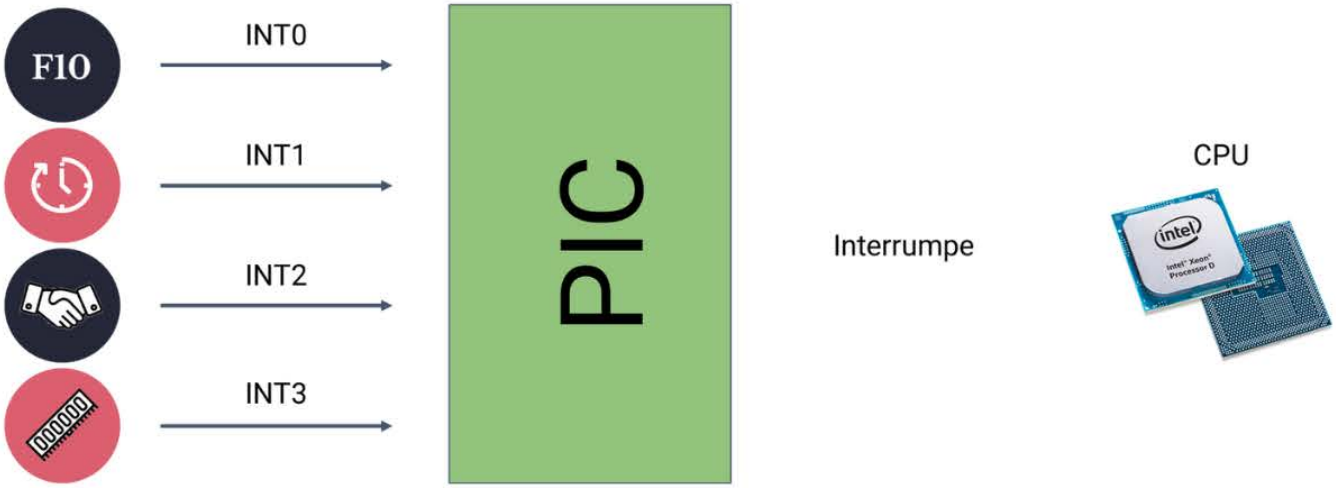
Timer

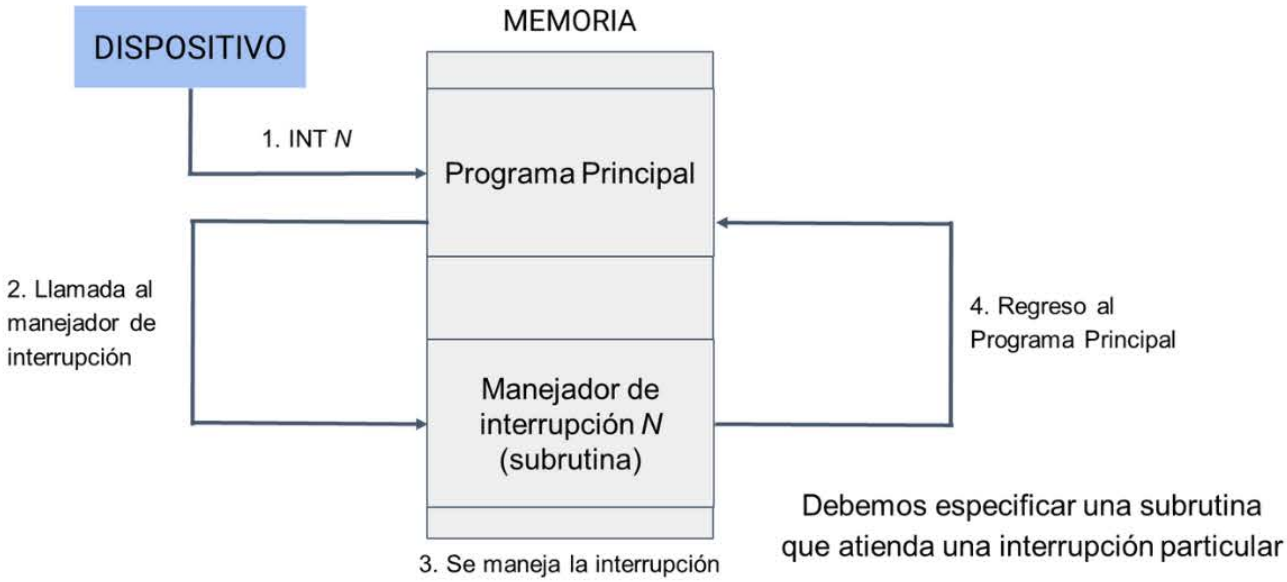
Handshake

CDMA

Cada uno va a tener la posibilidad de interrumpir al CPU cuando lo necesiten

Los dispositivos interrumpen a la CPU a través del PIC





Debemos especificar una subrutina que atienda una interrupción particular

PIC

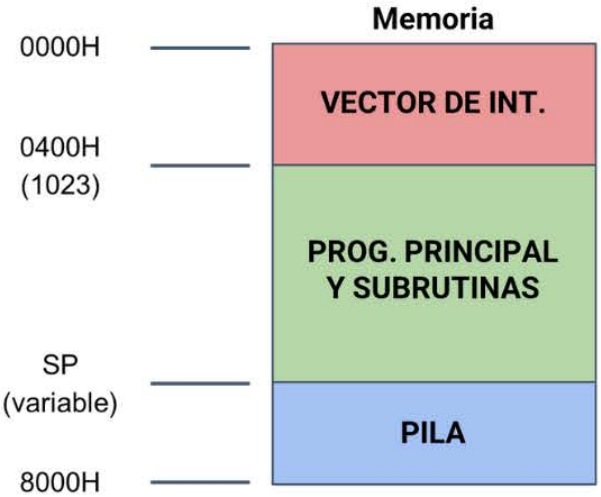
El vector de interrupciones

Partamos desde un ej. simple: contar las veces que se presionó la tecla F10 en DL

Vamos a realizar los siguientes pasos:

1. Escribir la subrutina que se ejecutará cuando se produzca la interrupción (que finaliza con **IRET**)
2. Elegir un ID de interrupción (cualquiera menos 0, 3, 6 ó 7)
3. Poner la dirección de la subrutina en el ***Vector de interrupciones*** (ya veremos qué es esto)
4. Configurar el **PIC**
   a. Bloquear las interrupciones con la sentencia **CLI**
   b. Poner el ID en el PIC para la interrupción que nos interesa
   c. Desenmascarar la interrupción
   d. Desbloquear las interrupciones con la sentencia **STI**

```
ORG 3000H
; Subrutina que atiende la interrupción de F10
CONTAR: INC DL
        ; ACA FALTA ALGO!
        IRET
```



- Va de la posición 0 (0000H) a la 1023 (0400H)
- Consta de 1024 posiciones de memoria
- Lo usaremos para asociar las interrupciones con una subrutina a ejecutar

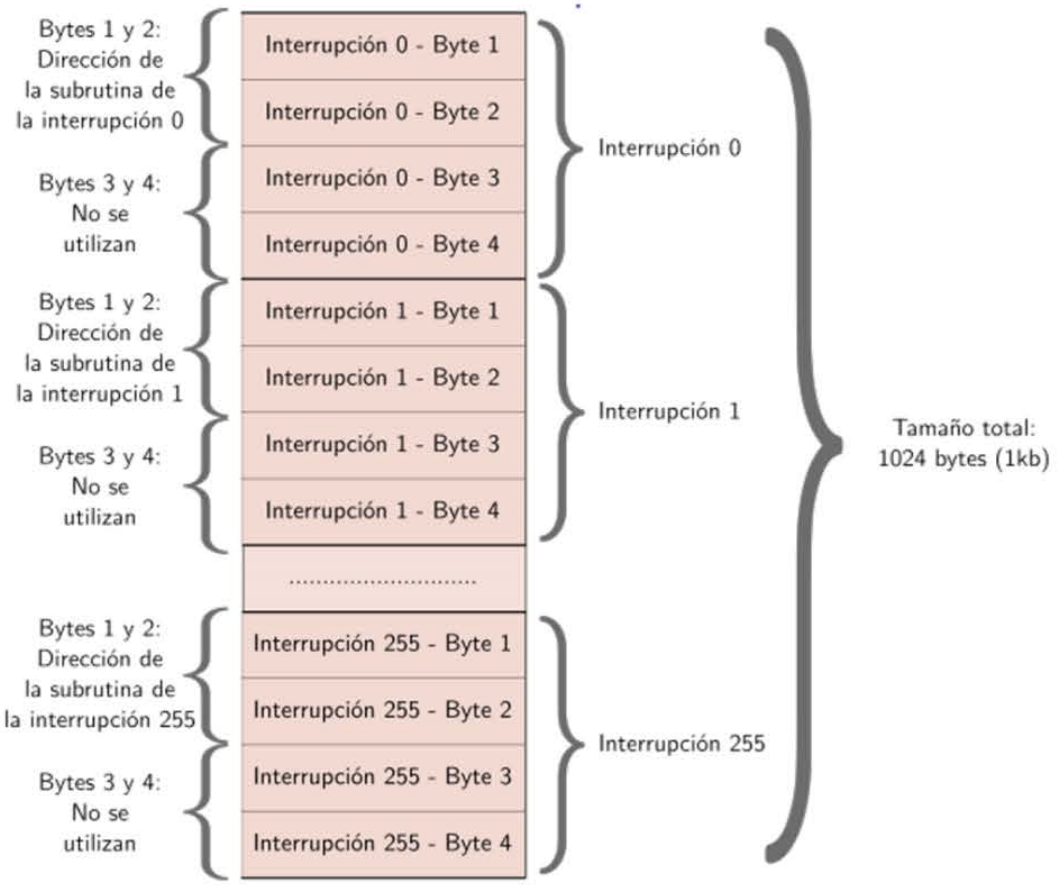
PIC

El vector de interrupciones

Contar las veces que se presionó la tecla F10 en DL

2. Seleccionar un *ID* para la interrupción

- Seleccionar un *ID* es crucial ya que se usará para asociar una interrupción con una subrutina
- Cuando ocurre una interrupción la máquina toma el *ID* que elegimos y busca la dirección de la subrutina a ejecutar en la posición *ID * 4* del Vector de Int.
- Vamos a seleccionar como *ID* el 5.
- Cuando toquemos F10, se interrumpirá nuestro programa y se fijará en la posición 20 del Vec. de Int. para obtener la dirección de la subrutina a ejecutar



PIC

Ejemplo

Contar las veces que se presionó la tecla F10 en DL

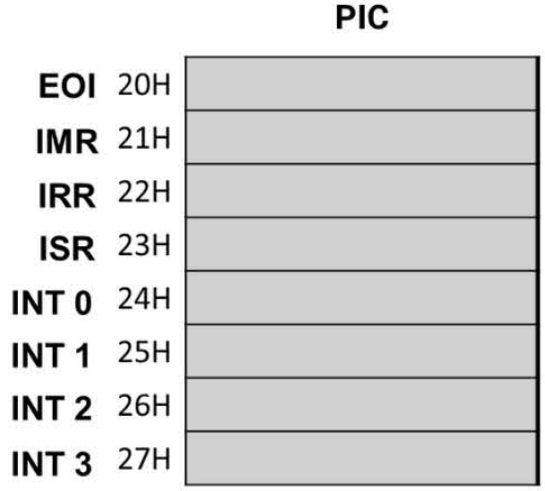
3. Poner la dirección de la subrutina en el ***Vector de interrupciones***

```
ORG 3000H
; Subrutina que atiende la interrupción de F10
CONTAR: INC DL
        ; ACA FALTA ALGO!
    IRET
```

```
ORG 2000H
; Tomo direccion de la subrutina
MOV AX, CONTAR ; AX = Dir de contar (3000H)

; Pongo la dir en el vector de int.
MOV BX, 20 ; 5 * 4 = 20 en Vec. de Int.
MOV [BX], AX ; En la posicion 20 = 3000H
...
```

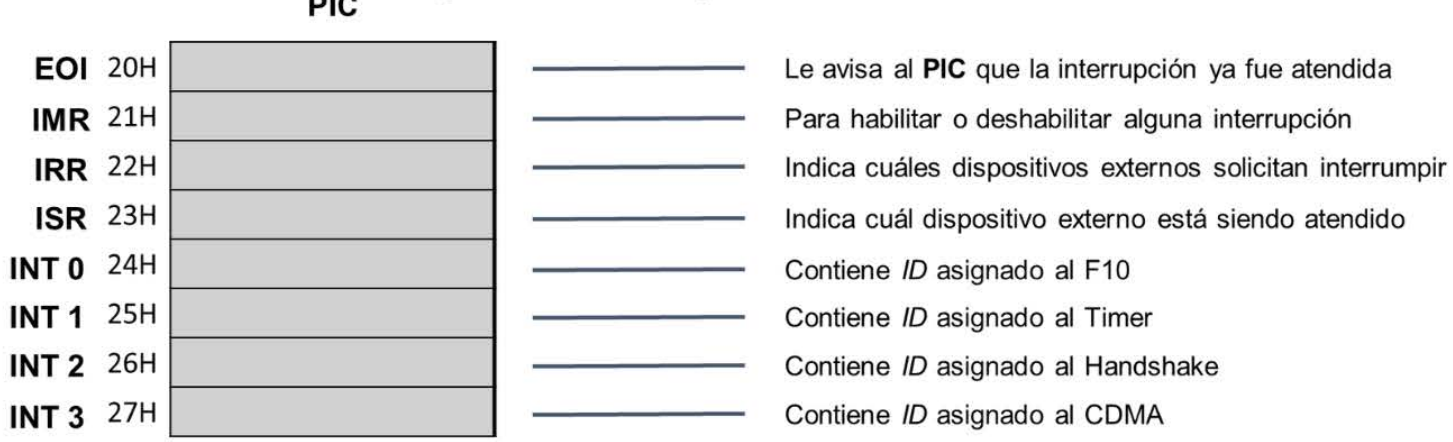
Contar las veces que se presionó la tecla F10 en DL

4. Configurar el **PIC**



- Se maneja desde la memoria de E/S así que para configurar haremos uso de **IN** y **OUT**
- El **PIC** permite configurar el resto de las cosas que nos quedaron pendientes
- Las sentencias **CLI** y **STI** bloquean y habilitan, respectivamente, las interrupciones
- Cuando configuremos el **PIC** debemos **siempre** debemos hacerlo entre **CLI** y **STI**

El **PIC** contiene los siguientes campos



PIC

Ejemplo

¿Cómo funcionan los campos configurables?

PIC

EOI 20H

- El **PIC** nos avisa que un dispositivo nos quiere interrumpir. Nosotros le avisamos que ya atendimos la interrupción
- Antes de volver de las subrutina de la interrupción debemos poner el valor 20H en el **EOI**

```
MOV AL, 20H
OUT 20H, AL ; EOI = 20H
```

Cuando termina la interrupción avisamos al **EOI**

PIC

IMR 21H

- Nos permite definir qué interrupciones vamos a atender y cuáles ignorar
- 1 significa deshabilitada, 0 habilitada

1 1 1 1 — Totalmente al pedo

1 0 1 0 — INT 3, INT 2, INT 1 e INT 0

Configuramos el **IMR** para atender solo INT 0

```
MOV AL, 11111110b
OUT 21H, AL
```

Configuramos el *ID* que habíamos elegido para F10 (INT 0)

```
MOV AL, 5
OUT 24H, AL
```

```
ORG 3000H
; Subrutina que atiende la interrupción de F10
CONTAR: INC DL
        ; Aviso al EOI que temina la subrutina
        MOV AL, 20H
        OUT 20H, AL ; EOI = 20H
    IRET
```

Todo esto entre CLI y STI

PIC

Ejemplo

Contar las veces que se presionó la tecla F10 en DL

4. Configurar **PIC**

```
ORG 3000H
; Subrutina que atiende la interrupción de F10
CONTAR: INC DL
        MOV AL, 20H
            OUT 20H, AL ; EOI = 20H
        IRET
```

```
ORG 2000H
; Tomo direccion de la subrutina
MOV AX, CONTAR ; AX = Dir de contar (3000H)
; Pongo la dir en el vector de int.
MOV BX, 20 ; 5 * 4 = 20 en Vec. de Int.
MOV [BX], AX ; En la posicion 20 = 3000H
CLI
MOV AL, 11111110b
OUT 21H, AL
MOV AL, 5
OUT 24H,  AL
STI
...
```

PROGRAMA COMPLETO

```
1   ORG 3000H
2   ; SUBRRUTINA QUE ATIENDE LA INTERRUPCION F10
3   CONTAR: INC DL
4
5   ; AVISA AL PIC QUE TERMINAMOS
6   MOV AL, 20H
7   OUT 20H, AL ; EOI = 20H
8   IRET
9   ; VOLVEMOS CON IRET
10
11  ORG 2000H
12  ; SELECCIONAMOS ID 10 PARA EL F10
13  MOV AX, CONTAR
14  MOV BX, 40 ; 10*4=40
15  MOV [BX], AX ; 40=3000H
16
17  ; CONFIGURAMOS EL PIC
18  CLI
19  MOV AL, 11111110B
20  OUT 21H, AL ; IMR = 11111110
21  MOV AL, 10
22  OUT 24H, AL ; INT 0 = 10
23  STI
24
25  LOOP: JMP LOOP
26
27  INT 0
28  END
29
```

Trucazo

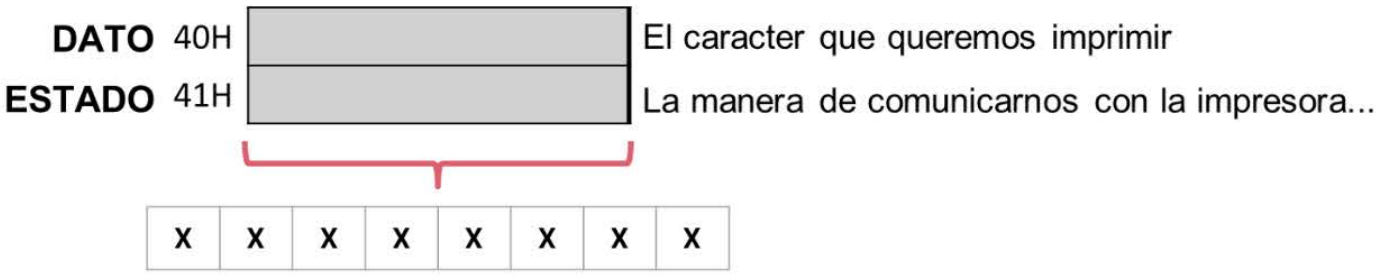
Como recordar tantas direcciones fijas es dificil se puede hacer uso de constantes!

```
EOI EQU 20H
IMR EQU 21H
INT0 EQU 24H

ORG 2000H
...
CLI
MOV AL, 11111010b
OUT IMR, AL ; 21H = 11111010

MOV AL, 5
OUT INT0, AL ; 24H = 5
STI
...
```

HANDSHAKE

Registros

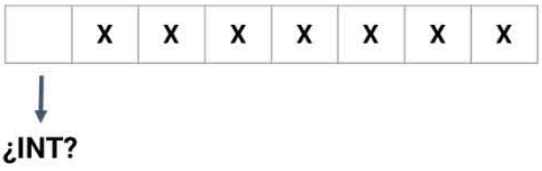
HANDSHAKE



Estos bits tiene diferentes significados dependiendo de si los pusimos en **entrada** o **salida**

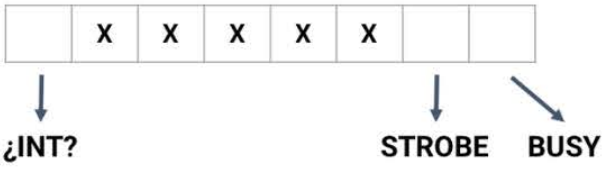
Los bits del registro **estado** tiene diferentes significados dependiendo de si los pusimos en **entrada** o **salida**

SALIDA



- **Bit 7 (Interrupción)** - 1 si queremos por interrupción, 0 por polling/consulta de estado

ENTRADA



- **Bit 0 (busy)** - 1 si está ocupada la impresora, 0 si está libre
- **Bit 1 (strobe)** - 1 si el strobe está activado, 0 si está desactivado
- **Bit 7 (Interrupción)** - 1 si es por interrupción, 0 si es por polling/consulta de estado

HANDSHAKE

Ejercicio 1

Escribir un programa que envíe datos a la impresora a través del Handshake. La comunicación se debe establecer por **consulta de estado** (polling)

```
HAND_DATO    EQU 40H
HAND_ESTADO  EQU 41H

ORG 1000H
MENSAJE DB "El Handshake la rompe"
FIN          DB ?
```

```
ORG 2000H
; Configuro el Handshake para el polling
IN AL, HAND_ESTADO ; Tomo estado actual
AND AL, 07FH ; 7FH = 01111111
OUT HAND_ESTADO, AL ; Estado = 0xxxxxxx
```

```
; Recorremos el mensaje y lo enviamos caracter
; a caracter hacia la impresora
MOV BX, OFFSET MENSAJE ; Para recorrer el mensaje
POLL: IN AL, HAND_ESTADO ; Tomo el estado actual
   AND AL, 1 ; Chequeo el primer bit
   JNZ POLL ; Mientras sea 1 sigo en el loop
   MOV AL, [BX] ; Tomo el caracter
   OUT HAND_DATO, AL ; Lo envio al registro de datos
   INC BX ; Avanzo a la siguiente posicion
   CMP BX, OFFSET FIN ; Chequeo si llegue al final
   JNZ POLL
INT 0
END
```

HANDSHAKE

Ejercicio 2

Escribir un programa que envíe datos a la impresora a través del Handshake. La comunicación se debe establecer por **interrupción**

1. Debemos configurar ¿En qué configuramos el bit de **INT**? En 1! ~~No~~ queremos interrupciones!
2. Ya no consultaremos constantemente si está libre Nos interrumpirá cuando esté libre!
3. Cuando la impresora nos interrumpa mandamos el caracter a **DATO** (40H)

```
ORG 3000H
; Recorremos el mensaje y lo enviamos caracter
; a caracter hacia la impresora
IMPRIMIR: PUSH AX; Salvo AX por las dudas
MOV AL, [BX] ; Tomo el caracter
OUT HAND_DATO, AL ; Lo envio al registro de datos
INC BX ; Avanzo a la siguiente posicion

; Chequeo si llegue al final del string
CMP BX, OFFSET FIN
JNZ CONTINUA

; En caso de que llegue aca significa que llegamos al final del string.
Debemos desactivar las interrupciones por Handshake y por el PIC
IN AL, HAND_ESTADO ; Tomo estado actual
AND AL, 07FH ; 7FH = 01111111
OUT HAND_ESTADO, AL ; Estado = 0xxxxxxx
```

```
; NOTA: no hace falta las sentencias CLI y STI porque estamos
haciendo esto antes de enviar el 20H al EOI, por lo que el PIC no nos
va a interrumpir ya que sabe que seguimos atendiendo la interrupcion
MOV AL, 11111111b ; Todo deshabilitado!
OUT IMR, AL

; Aviso al PIC y vuelvo de la subrutina
CONTINUA: MOV AL, 20H
OUT EOI, AL

POP AX ; Recupero lo que habia en AX
IRET
```

```
ORG 2000H
; Configuro el vector de interrupciones. ID = 9
MOV AX, IMPRIMIR
MOV BX, 36 ; 36 = 9 * 4
MOV [BX], AX

; Configuro PIC
CLI
MOV AL, 11111011b ; Solo Handshake habilitado
OUT IMR, AL

MOV AL, 9
OUT INT2, AL ; Mando el ID seleccionado al registro INT2

MOV BX, OFFSET MENSAJE ; Para recorrer el mensaje
STI
```

```
; Configuro el Handshake para interrupcion
IN AL, HAND_ESTADO ; Tomo estado actual
OR AL, 80H ; 80H = 10000000
OUT HAND_ESTADO, AL ; Estado = 1xxxxxxx

; Simulamos un programa en ejecucion para ver que puede
interrumpirnos
POLL: NOP
 NOP ; Esto es el Counter
 NOP ; Esto es Youtube
 NOP ; Esto es el Chrome
JMP POLL

INT 0
END
```

IMPRESORA por PIO

Configuración por PIO

1. ¿Cómo configuramos el **PA** a partir de **CA**? Strobe en 0 (salida) y Busy en 1 (entrada)
2. ¿Cómo configuramos el **PB** a partir de **CB**? Todos de salida!

Del punto 3 al 6 debemos repetirlo para cada caracter

3. Consultaremos constantemente si está libre Chequear si el bit **Busy** = 0
4. Cuando la impresora esté libre mandamos el caracter a **PB** (31H)
5. Hasta que no mandemos el bit de Strobe en 1 no se va a imprimir!
6. Después de enviar el Strobe en 1, debemos volver a ponerlo en 0

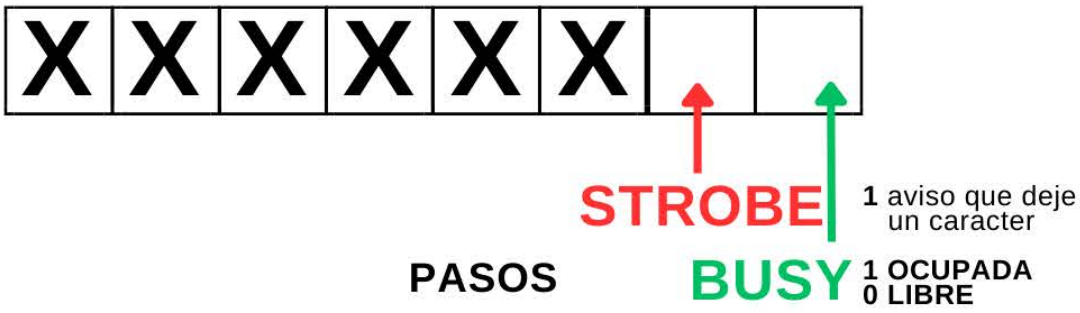
```
PA EQU 30H
PB EQU 31H
CA EQU 32H
CB EQU 33H

ORG 1000H
MENSAJE DB “Imprimiendo con el PIO!”
FIN     DB ?
```

```
ORG 2000H
; Configuro PA y PB a partir de CA y CB
MOV AL, 11111101b ; Str = salida, Busy = entrada
OUT CA, AL ;
MOV AL, 0 ; Todos 0 = Todo de salida!
OUT CB, AL ;
```

```
; Recorro el mensaje y envío caracter a caracter hacia la impresora
MOV BX, OFFSET MENSAJE ; Para recorrer el mensaje
POLL: IN AL, PA ; Tomo el estado actual
        AND AL, 1 ; Chequeo el primer bit
        JNZ POLL ; Mientras sea 1 sigo en el loop
        MOV AL, [BX] ; Tomo el caracter
        OUT PB, AL ; Lo envio al registro de datos
    IN AL, PA ; Tomo el estado actual
        OR AL, 00000010b ; Fuerzo Strobe a 1
        OUT PA, AL ; Mando el nuevo Strobe a la impresora
        AND AL, 11111101b ; Fuerzo Strobe a 0
        OUT PA, AL ; Mando el nuevo Strobe a la impresora
        INC BX ; Avanzo a la siguiente posicion
        CMP BX, OFFSET FIN ; Chequeo si llegue al final
        JNZ POLL
INT 0
END
```

PIO

PA DATOS IMPRESORA
PB DATOS IMPRESORA
CA CONFIGURACION cual entra o sale
CB CONFIGURACION cual entra o sale



1- ***PA CA*** **STROBE** 0 (salida), **BUSY** en 1 (entrada)
2- ***PB CB*** todo salida (0)

POR CADA CARACTER

3- Chequear si el bit **BUSY = 0**
4- Cuando este LIBRE mandamos el caracter **PB(31H)**
5- Hasta que no enviemos el bit de **STROBE** en 1 no imprime
6- Luego de enviar **1** a **STROBE** , volvemos a poner **STROBE** en 0

```
1  PA EQU 30H
2  PB EQU 32H
3  CA EQU 32H
4  CB EQU 33H
5
6  ORG 1000H
7  MENSAJE DB "IMPIMIR POR PIO"
8  FIN DB ?
9
```

```
10  ORG 2000H
11  MOV AL, 11111101b ; strobe salida(0) busy entrada(1)
12  OUT CA, AL
13
14  MOV AL, 0 ; todo salida (0)
15  OUT CB, AL
16
17  ; RECORRER EL STRING
18  MOV BX, OFFSET MENSAJE
19  POLL: IN AL, PA
20        AND AL, PA
21        JNZ POLL ; SI NO ESTA LIBRE SIGO PREGUNTANDO
22
23   ; LA IMPRESORA ESTA LIBRE
24   MOV AL, [BX]
25   OUT PB, AL
```

```
26
27   ; STROBE EN 1
28   IN AL, PA ; TOMO EL ESTADO
29   OR AL, 2 ; FUERZO A 1
30   OUT PA, AL
31
32   ; STOBE EN 0
33   IN AL, PA ; TOMO EL ESTADO
34   AND AL, 0FDH ; FUERZO A 0
35   OUT PA, AL
36
37   INC BX
38   CMP BX, OFFSET FIN
39   JNZ POLL
```