

# Arquitectura de computadoras

*Genaro Camele*

# Repaso

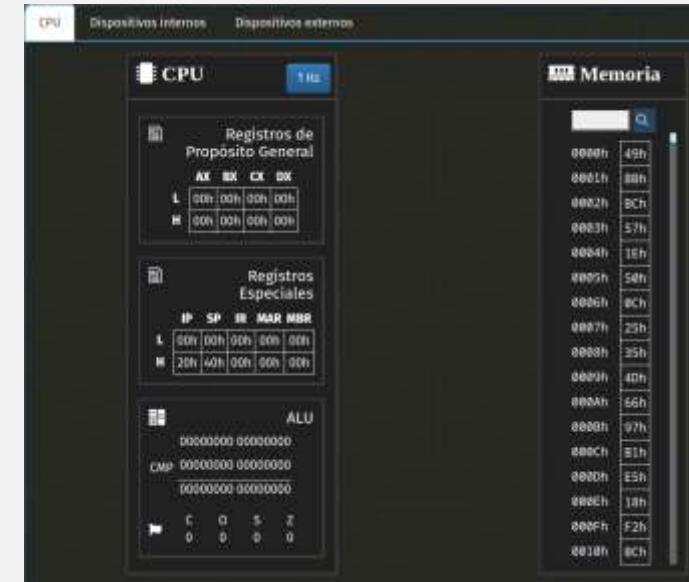
*Pila y subrutinas*

# Repasso

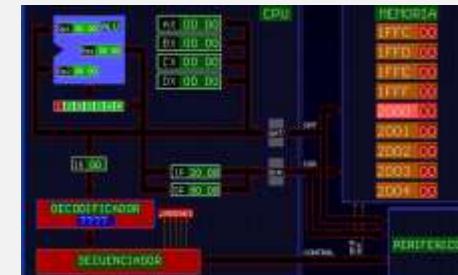
*Assembler*

Habíamos visto el **Lenguaje ensamblador**

- Un lenguaje común y corriente que se sufre
- Sus instrucciones son un mapeo directo a las instrucciones binarias de cada procesador
- Instrucciones para manejar **registros, memoria y flags**
- Teníamos al simulador **VonSim** ([vonsim.github.io](https://vonsim.github.io))



Para los masoquistas también está el simulador **MSX88**



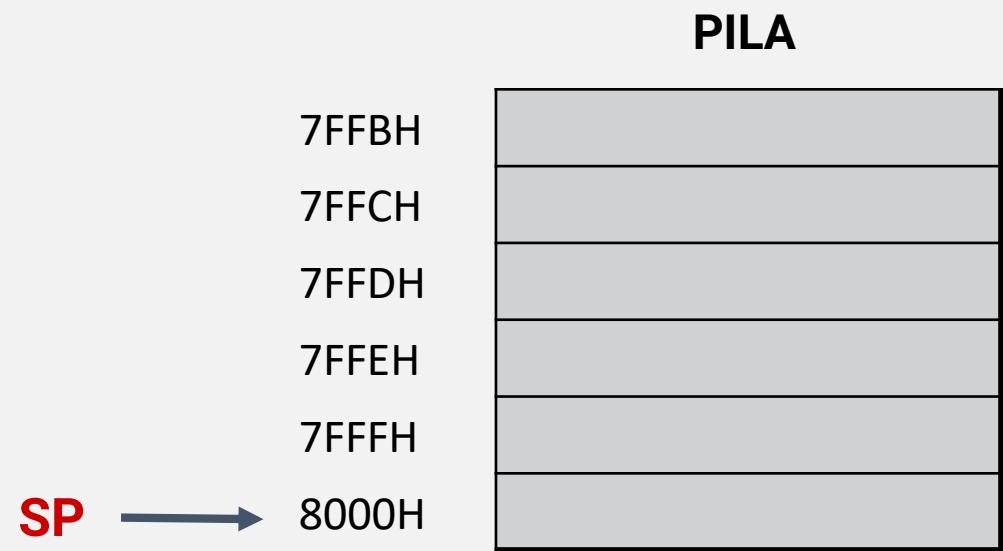
# Repaso

*Pila*

Como (casi) último concepto de Organización de computadoras vimos la **Pila**

Teníamos dos operaciones:

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



# Ejemplo

Pila

ORG 1000H

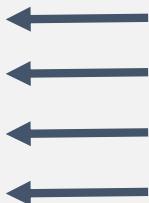
... ; Definición de mis variables

ORG 2000H

MOV AX, F744H ; AX = F744H  
MOV BX, 5 ; BX

= 5

PUSH AX  
PUSH BX  
POP AX  
POP BX  
HLT  
END



SP → 7FFBH  
SP → 7FFCH  
SP → 7FFDH  
SP → 7FFEH  
SP → 7FFFH  
SP → 8000H

| PILA |
|------|
|      |
| 05H  |
| 00H  |
| 44H  |
| F7H  |
|      |

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

# Repasso

## *Subrutinas*

Al igual que en los lenguajes de programación de alto nivel. En Assembler también contamos con modularización de código:  
las **Subrutinas**

Podemos armar módulos/funciones/procesos que realicen una tarea específica

Se llaman con la sentencia **CALL**

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

ORG 1000H  
... ; Definición de mis variables

ORG 3000H  
... ; Definición de subrutina

ORG 2000H  
... ; Programa principal  
HLT  
END

DATOS

SUBRUTINAS

PROG. PRINC.

# Repasso

*Subrutinas*

Escribir un programa que multiplique dos números > 0 (en AX y BX) y almacene el resultado en CX

**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!

# Pasaje de parámetros

*Por registros*

A diferencia de otros lenguajes, el pasaje por parámetros no puede hacerse explícitamente

~~function mul(num1: integer; num2: integer) ...~~

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
```

~~function mul(**var** num1: integer; **var** num2: integer) ...~~

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
```

# Pasaje de parámetros

*Por pila*

Bien, pasamos los parámetros por registros, pero podríamos usar también la pila!

**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
```

**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
```

# Pasaje de parámetros

Por pila

No todo es color de rosa con la pila, vamos a tener que “tomar” los parámetros de forma diferente

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**

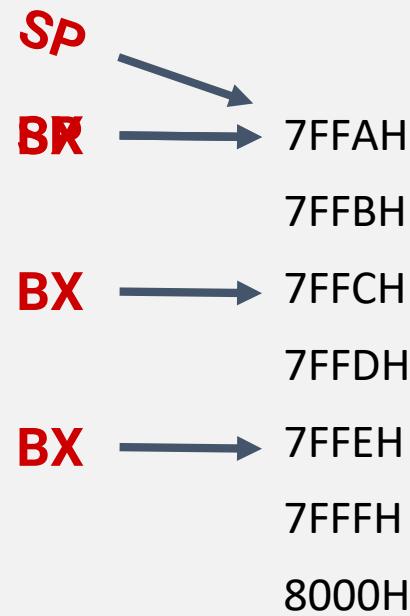
MOV RES, CX

POP AX

POP BX

HLT

END



**ORG 3000H**

~~MOV BX, SP~~, 0 ; BX = Resultado

~~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**

# Pasaje de parámetros

*Por pila y referencia*

¿Y si lo hacemos por pila, pero ahora por referencia en vez de valor?

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**

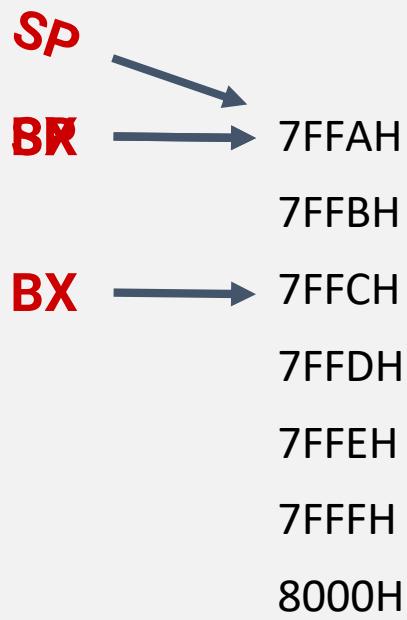
MOV RES, CX

POP AX

POP BX

HLT

END



PILA

**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

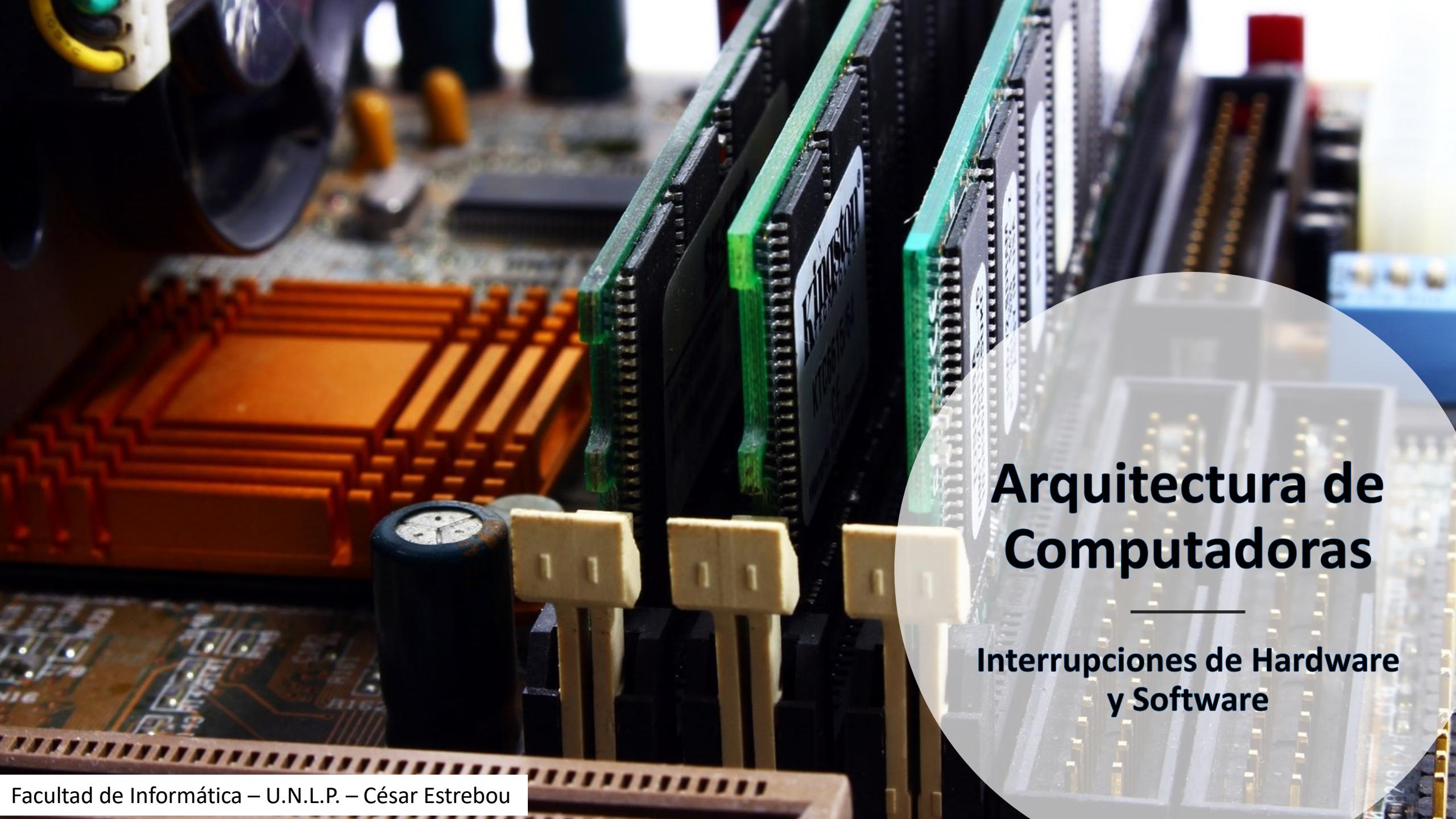
[...];

Repetimos con NUM1

[...]; Seguimos

como siempre

**RET**

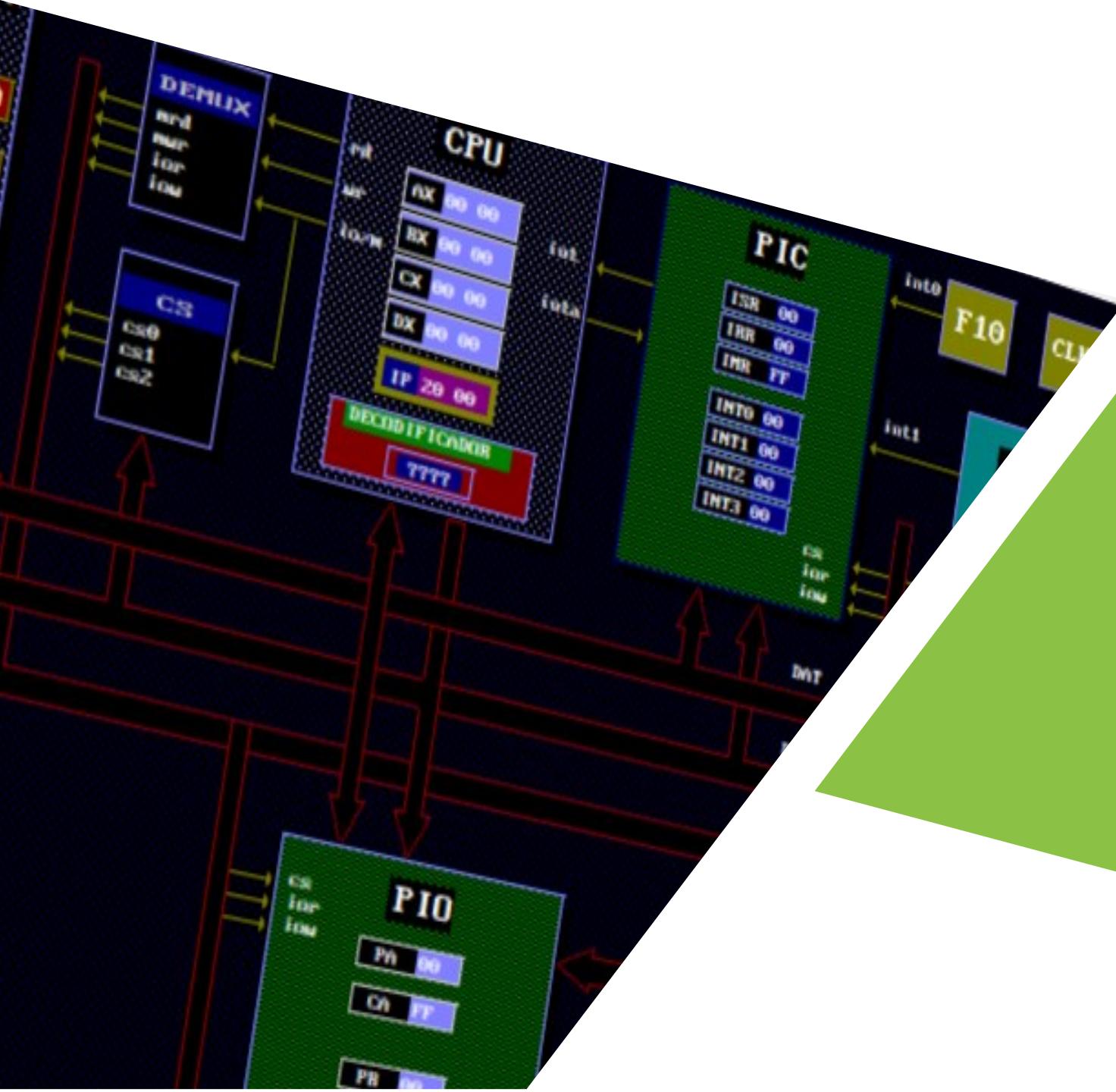


# Arquitectura de Computadoras

---

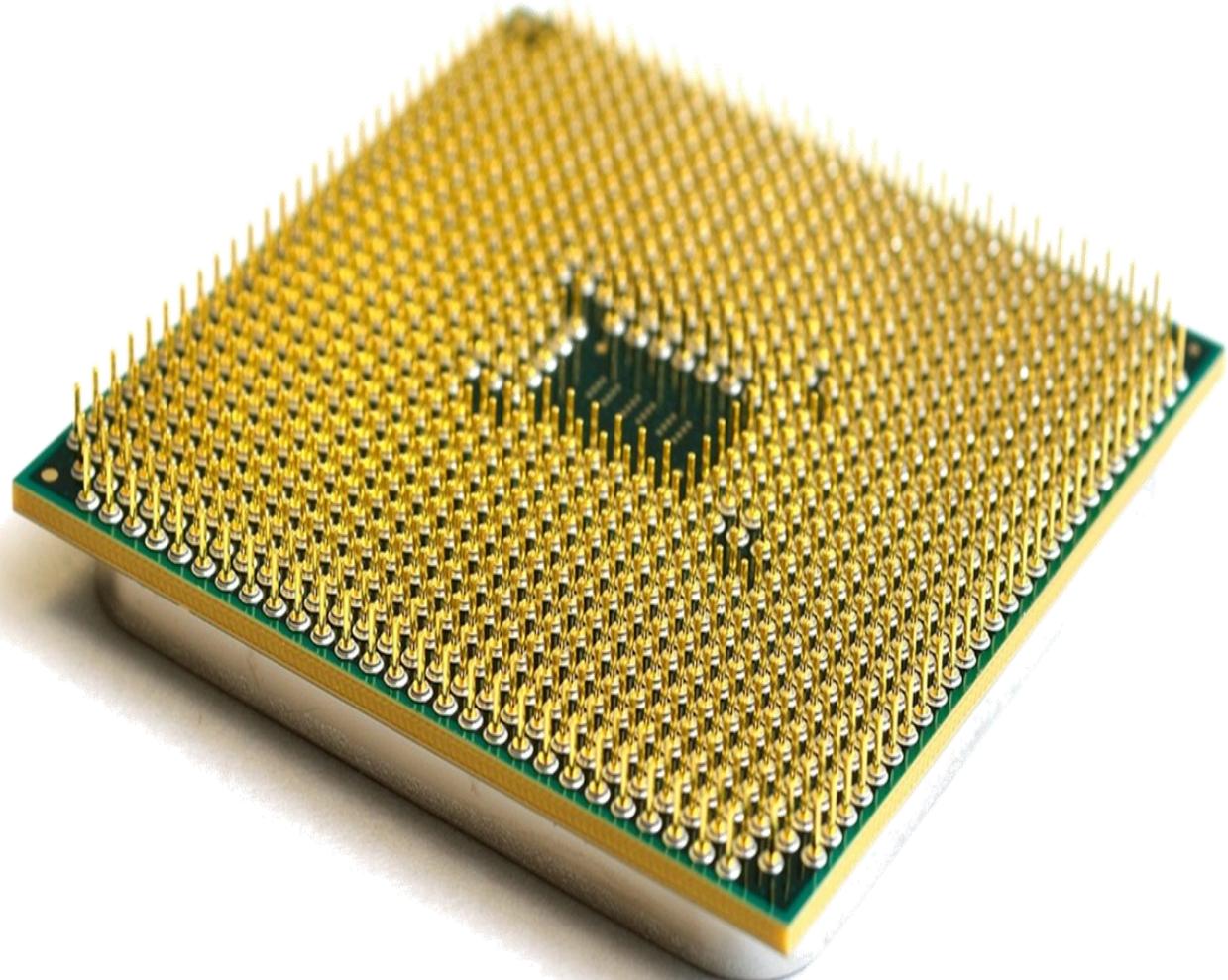
Interrupciones de Hardware  
y Software

# Interrupciones de Hardware y Software



# Agenda

## ► Temas



### 01 Interrupciones por Software

Vector de Interrupciones. Mecanismo e instrucciones asociadas. Interrupciones de Software soportadas. Ejemplos

### 02 Entrada/Salida

Dispositivos de E/S. Esquema. Instrucciones asociadas

### 03 Interrupciones por Hardware

Conceptos. Ciclo de interrupción de Hardware. Controlador de Interrupciones (PIC). Tecla F10 y Temporizador(Timer)

### 04 Ejemplos

Soluciones de varios ejercicios de la práctica

# Interrupciones

**Una interrupción es una situación específica...**

que requiere la ejecución de un programa especial denominado “Rutina de Servicio de Interrupción” (ISR) para resolverla

**Suspende el programa en ejecución...**

y guarda su contexto. Invoca la ejecución de una ISR mediante un mecanismo específico

**Reanuda el programa interrumpido...**

restaurando el contexto guardado y continua la ejecución en el punto de interrupción



# Interrupciones - Tipos



## Tipos de Interrupciones

Según la fuente que produce la interrupción se clasifican en 3 categorías

### Interrupciones por Software

- Producidas por el propio programa en ejecución
- También denominadas “llamadas al sistema”
- Ejecutan rutinas que dependen del hardware pero nos abstraen de él.
- Ej: leer de teclado, escribir en pantalla

### Interrupciones por Hardware

- Se producen de manera asincrónica a la ejecución del procesador
- Ligadas a requerimientos externos al procesador
- Normalmente responden a pedidos de E/S
- Ej: Dispositivo listo para envío/recepción de datos

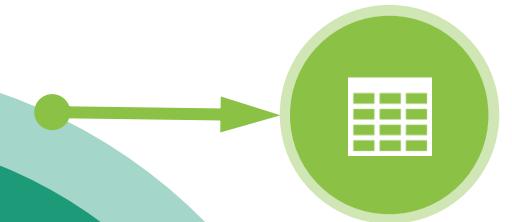
### Excepciones

- Producidas de manera sincrónica a la ejecución del procesador
- Por lo general causadas por anomalías en la ejecución instrucciones.
- Ej: división por cero, violación de memoria, desbordamiento

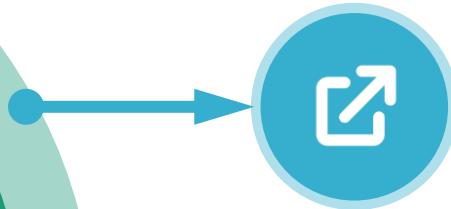
# Interrupciones - Mecanismo de Llamado



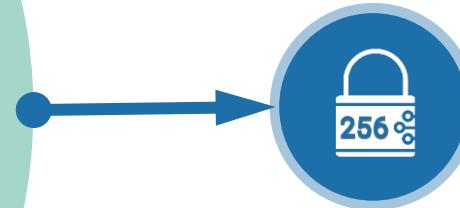
Las interrupciones utilizan una **tabla** para asociar el **número** de interrupción (que no cambia) con la **dirección** de una subrutina (que cambia)



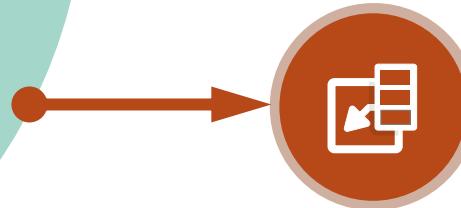
Esta tabla se la denomina IVT (Interrupt Vector Table) o “vector de interrupciones”



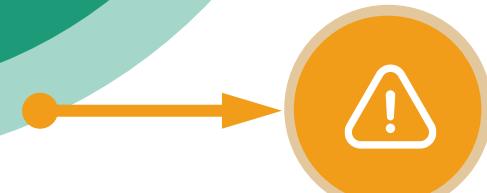
La tabla es externa e independiente al programa. La rutina puede estar dentro o fuera del programa



Tiene 256 lugares. A lo sumo se pueden tener 256 interrupciones



Cada lugar ocupa 4 bytes: 2 bytes altos siempre en 0, 2 bytes bajos contienen la dirección de la subrutina



Ocupa 1024 bytes iniciales ( $256 * 4$  bytes). No ubicar datos ni instrucciones e 0 a la 1023 de la memoria.

# Interrupciones – Mecanismo de Llamado



## Vector de interrupciones

| posición | 1º byte | 4 bytes para dirección de subrutina         | Ult. byte |
|----------|---------|---|-----------|
| 0        | 0       | {dirección de subrutina asociada a INT 0}   | 3         |
| 1        | 4       | {dirección de subrutina asociada a INT 1}   | 7         |
| 2        | 8       | {dirección de subrutina asociada a INT 2}   | 11        |
| ...      | ...     | ...   | ...       |
| 254      | 1016    | {dirección de subrutina asociada a INT 254} | 1019      |
| 255      | 1020    | {dirección de subrutina asociada a INT 255} | 1023      |

El vector de interrupciones es una división lógica implementada en la memoria física

La dirección de memoria asociada a la posición del vector se obtiene multiplicando por 4



# Interrupciones por Software





# Interrupciones por Software

¿Por qué es útil una interrupción por software (INT) si contamos con subrutinas (CALL)?

```
    H  
MOV BX, OFFSET TEXTO  
MOV AL, LEN_TEXTO  
INT 7  
INT 0  
END  
  
ORG 2000H  
MOV BX, OFFSET TI  
MOV AL, LEN_TE  
CALL IMPRIMIR  
INT 0  
END
```

## Uso de Código Ya Escrito

Podemos usar “servicios” del sistema operativo (monitor del MSX88) que ya están implementados: Imprimir una cadena, Leer un carácter Terminar el programa

## Independencia del Hardware

Sistema operativo define una interface que no cambia, desarrollando rutinas que funcionan independientemente de la implementación del fabricante de los dispositivos instalados.

## Actualizaciones Transparentes

En distintas versiones del sistema operativo las subrutinas que proveen servicios pueden cambiar, ser optimizadas o corregidas y nuestro programa no necesita saberlo

## Adaptación a la Evolución del Hardware

Independientemente de los cambios del hardware y de las diferentes implementaciones de los fabricantes, el sistema operativo mantendrá la funcionalidad, aprovechando los avances en el desarrollo del hardware

# Interrupciones por Software



## Invocación (pseudo-código)



- INT {nº de interrupción}:
- Salva flags de estado en la pila.
- Salva dirección de retorno en la pila.
- Busca dirección de rutina en la posición {nº de interrupción} del vector
- Asigna registro IP con la dirección de la subrutina
- (Ejecuta rutina)

Toda rutina de interrupción debe preservar los valores de los registros previos al llamado. Debe realizar PUSH de los registros que usa, al iniciar la rutina y realizar POP de los mismos al finalizar

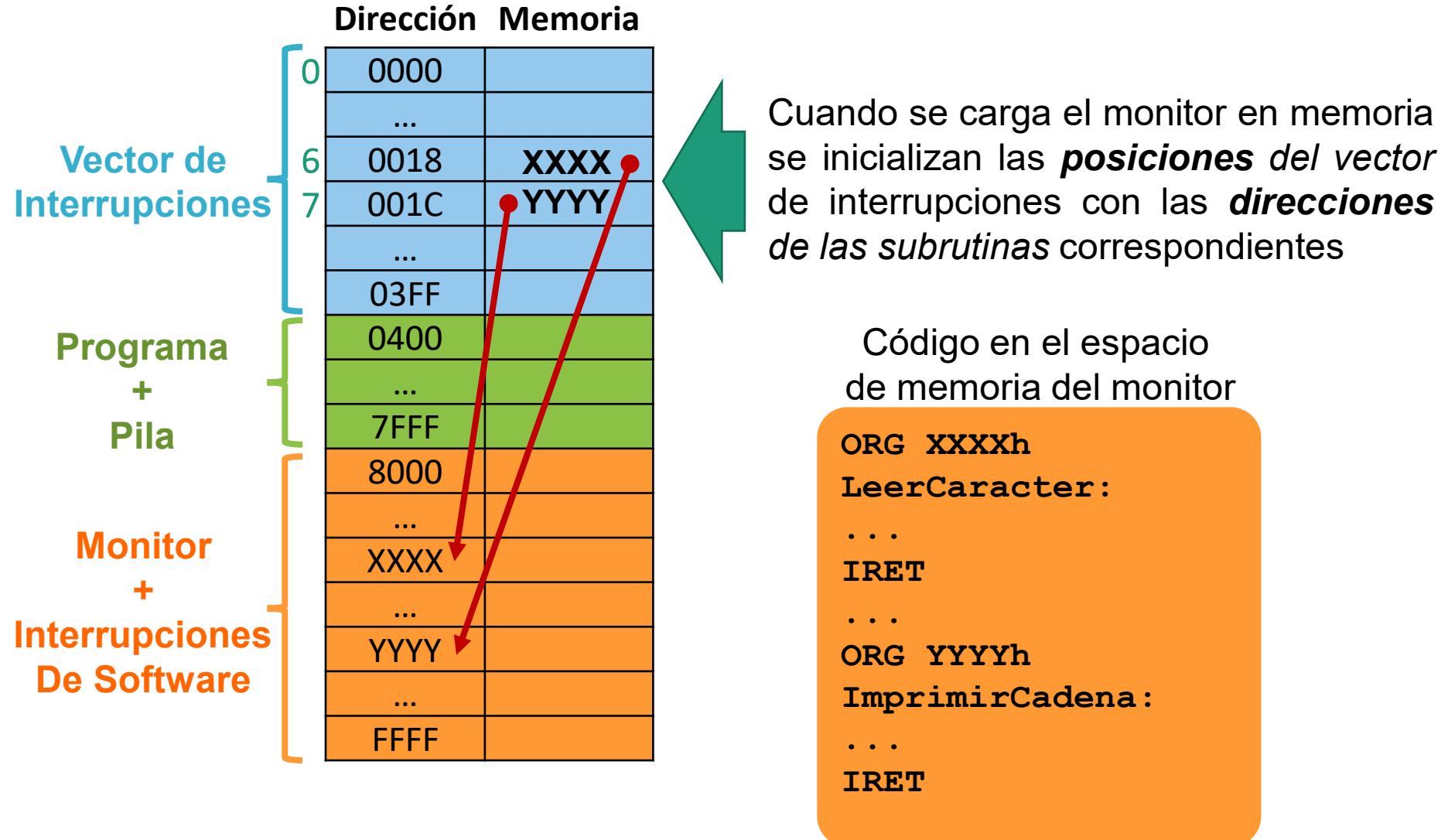
# Interrupciones por Software



- Para retornar de una subrutina normal (CALL) usamos la instrucción RET que desapila la dirección de retorno cuando esta finalizaba.
- Para volver de una subrutina de interrupción (INT) usamos la instrucción IRET, que además de la dirección de retorno, restaura desde la pila los flags de estado.
- IRET (pseudo-código)
  - (esta al final de la subrutina de INTERRUPCIÓN)
  - Repone los flags de estado desde la pila.
  - Asigna registro IP con la dirección de retorno desde la pila.

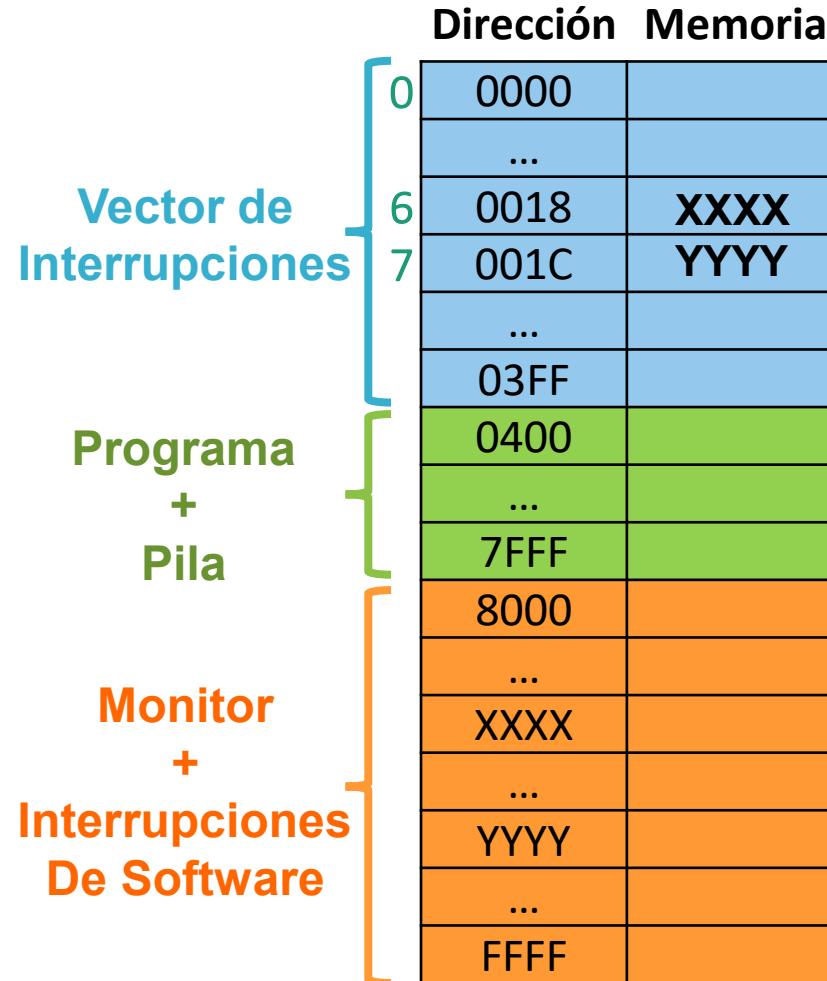


# Interrupciones por Software - Esquema





# Interrupciones por Software - Esquema



Para invocar desde nuestro programa a la rutina *LeerCaracter* usamos la instrucción **INT 6** y para *ImprimirCadena* usamos **INT 7**

Código en el espacio de memoria del monitor

```
ORG xxxxh
LeerCaracter:
...
IRET
...
ORG yyyyh
ImprimirCadena:
...
IRET
```



# Interrupciones por Software

## Interrupciones por software soportadas en el MSX88



INT 0

Finalizar programa. Similar a HALT

No requiere parámetros. Uso:

INT 0



INT 3

Depuración de código (no lo usamos)

No requiere parámetros. Uso:

INT 3



INT 6

Lee 1 carácter del teclado sin eco (no sale en pantalla). Requiere una dirección de memoria para almacenar el carácter

Uso:

BX = dirección almacenamiento

INT 6



INT 7

Escribe un string en pantalla. Requiere un puntero al inicio del string y la cantidad de caracteres a imprimir. Uso:

BX = dirección (OFFSET) almacenamiento

AL = cant. Car. a escribir

INT 7

# Interrupciones por Software



## Ejercicio 1

Escritura de datos en la pantalla de comandos.

Implementar un programa en el lenguaje assembler del simulador MSX88 que muestre en la pantalla de comandos un mensaje previamente almacenado en memoria de datos, aplicando la interrupción por software INT 7



# Interrupciones por Software

## Ejercicio 1:

```
1. ORG 1000H  
2. MSJ DB "ARQUITECTURA DE COMPUTADORAS-"  
3. DB "FACULTAD DE INFORMATICA-"  
4. DB 55H U  
5. DB 4EH N  
6. DB 4CH L  
7. DB 50H P  
8. FIN DB ?  
9. ORG 2000H  
10. MOV BX, OFFSET MSJ  
11. MOV AL, OFFSET FIN-OFFSET MSJ  
12. INT 7  
13. INT 0  
14. END
```

¿Son números, letras o algo más?  
¿De que depende?

Luego de la ejecución de esta instrucción en la consola del MSX88 aparece el texto:  
“ARQUITECTURA DE COMPUTADORAS-  
FACULTAD DE INFORMATICA-UNLP”

Termina la ejecución del programa, igual que HALT

# Interrupciones por Software



## Ejercicio 4

Lectura de datos desde el teclado.

Escribir un programa que solicite el ingreso de un número (de un dígito) por teclado e inmediatamente lo muestre en la pantalla de comandos, haciendo uso de las interrupciones por software INT 6 e INT 7.



# Interrupciones por Software

## Ejercicio 4:

1. ORG 1000H
2. MSJ DB "INGRESE UN NUMERO:"
3. FIN DB ?
4. ORG 1500H
5. NUM DB ?
6. ORG 2000H
7. MOV BX, OFFSET MSJ
8. MOV AL, OFFSET FIN-OFFSET MSJ
9. INT 7
10. MOV BX, OFFSET NUM
11. INT 6
12. MOV AL, 1
13. INT 7
14. MOV CL, NUM
15. INT 0
16. END

Imprime en la consola: "INGRESE UN NUMERO:"

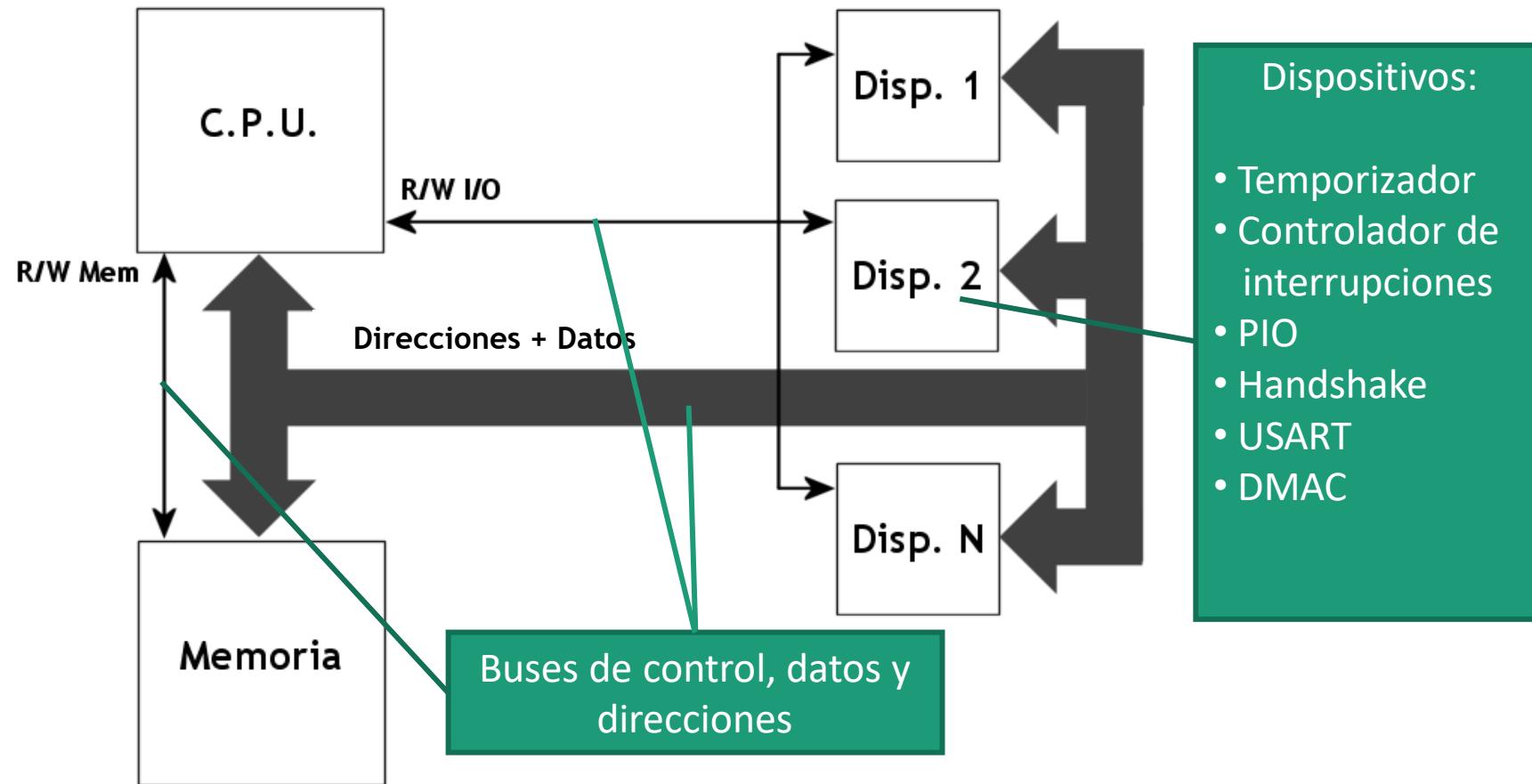
Espera el ingreso de un carácter desde la consola

Imprime en la consola el carácter recién ingresado

Comprobación visual de que se ingresó el carácter

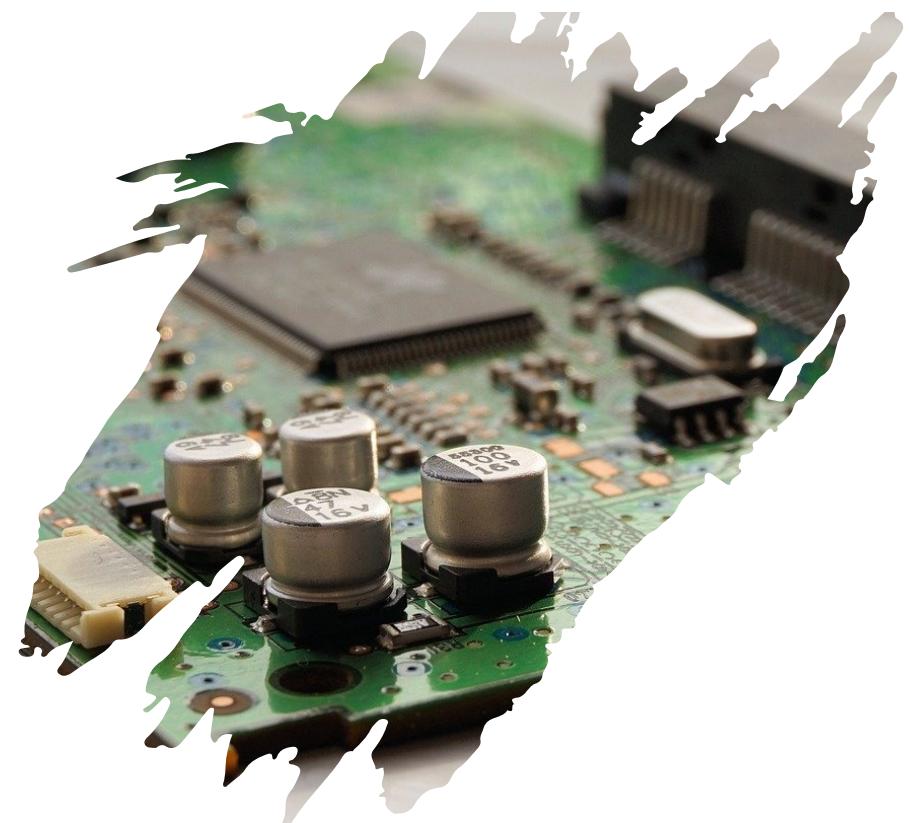
# Entrada/Salida del MSX88 - Dispositivos

Esquema de arquitectura del MSX88:



# Entrada/Salida del MSX88 - Dispositivos

- Denominamos dispositivo al hardware adicional a la CPU (similar a esta) con una función bien específica
- Generalmente son mucho mas lentos que la CPU
- Tienen registros propios (similar a la CPU) que utilizan para realizar las funciones para las que están diseñados.
- Los registros están “cableados” a los buses del sistema, y se leen/escriben de manera similar a la que se lee/escribe la memoria.



# Entrada/Salida del MSX88 - Dispositivos

- Escribir en estos registros nos permite:
  - Configurar el dispositivo
  - Enviar datos
- Leer estos registros nos permite:
  - Saber el estado en que se encuentra el dispositivo
  - Recibir datos
- Tipos de Dispositivos del MSX88:
  - PIC: Controlador de Interrupciones Programable
  - Temporizador o Timer: para medir tiempo
  - PIO, Handshake, USART: comunicarnos externa (ej: impresora)
  - DMA: Transferencia eficiente entre memoria y dispositivos

# Entrada/Salida del MSX88 - Dispositivos

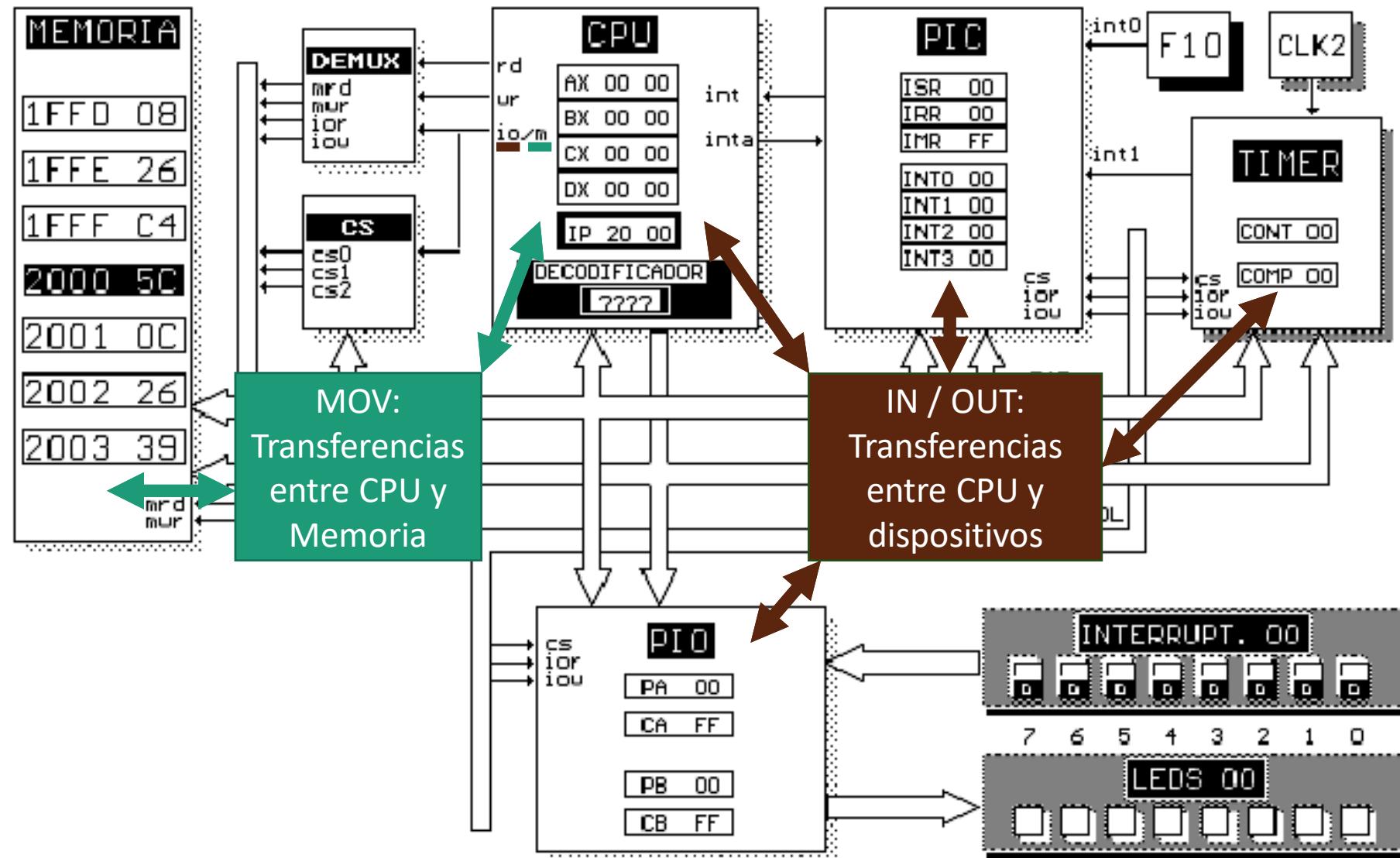
- Como para acceder a memoria y a dispositivos se utilizan señales de control diferentes, también se utilizan instrucciones diferentes
- Para salida (CPU → Dispositivo):
  - OUT DX, AL
  - OUT {número de puerto o dirección de E/S}, AL
- Para entrada (Dispositivo → CPU):
  - IN AL, DX
  - IN AL, {número de puerto o dirección de E/S}

Solo puede utilizarse AL (8bits) y DX

o

AL y un valor inmediato con la dirección del puerto

# Entrada/Salida del MSX88 - Dispositivos



# Interrupciones por Hardware

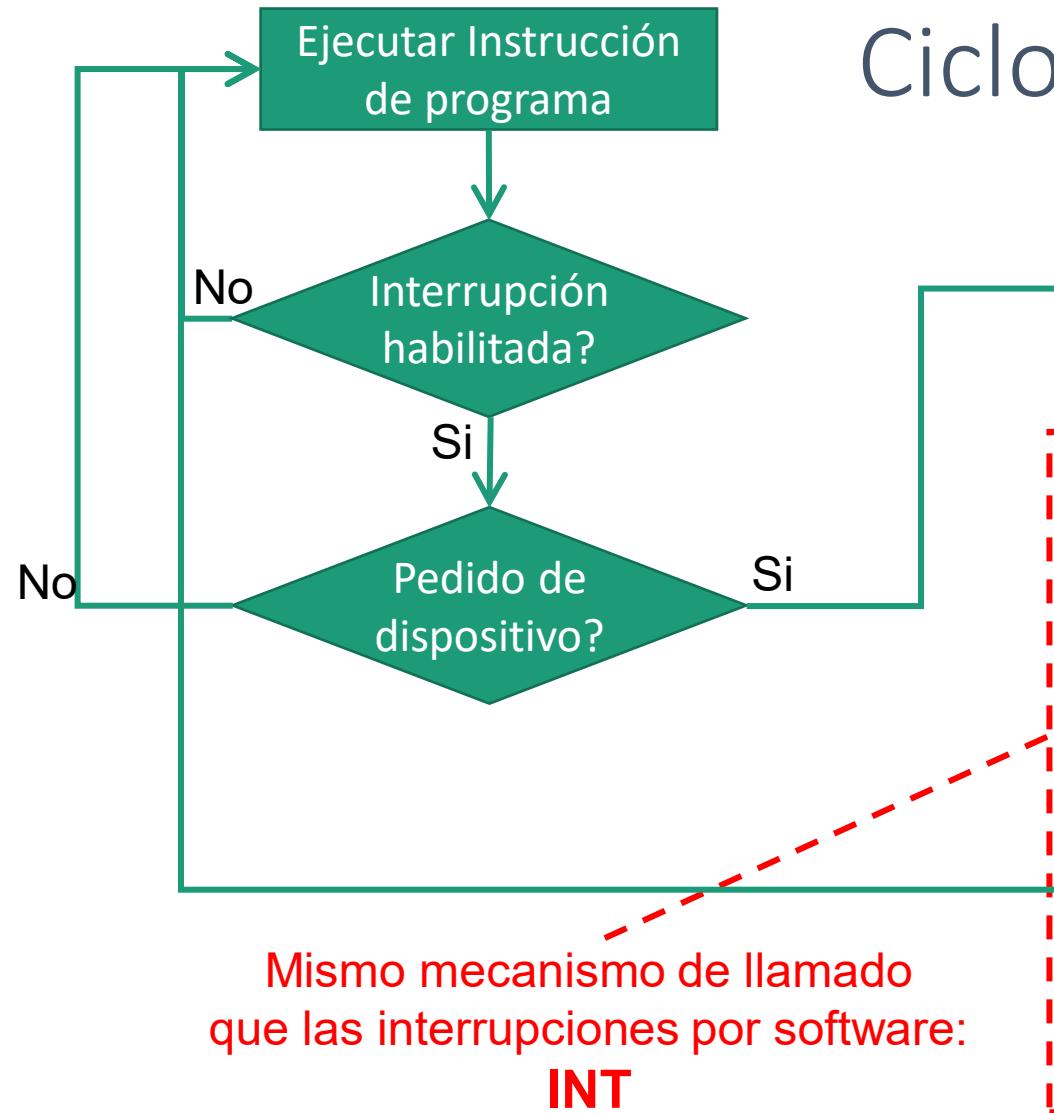
- Es un pedido que realiza un dispositivo (hardware) a la CPU para resolver un evento asociado a este.
- Una interrupción por hardware suspende (interrumpe) el flujo normal de la ejecución de un programa para ejecutar código especial para resolver el requerimiento del dispositivo.
- Para que un dispositivo genere interrupciones, el programador debe inicializarlo adecuadamente.
- Cuando un programa finaliza se debe restablecer el dispositivo para que no continúen las interrupciones
- Ventaja: evita el “polling” de los dispositivos (polling es consulta iterativa hasta que un dispositivo este listo)

# Interrupciones por Hardware - Ciclo

## Ciclo de interrupción

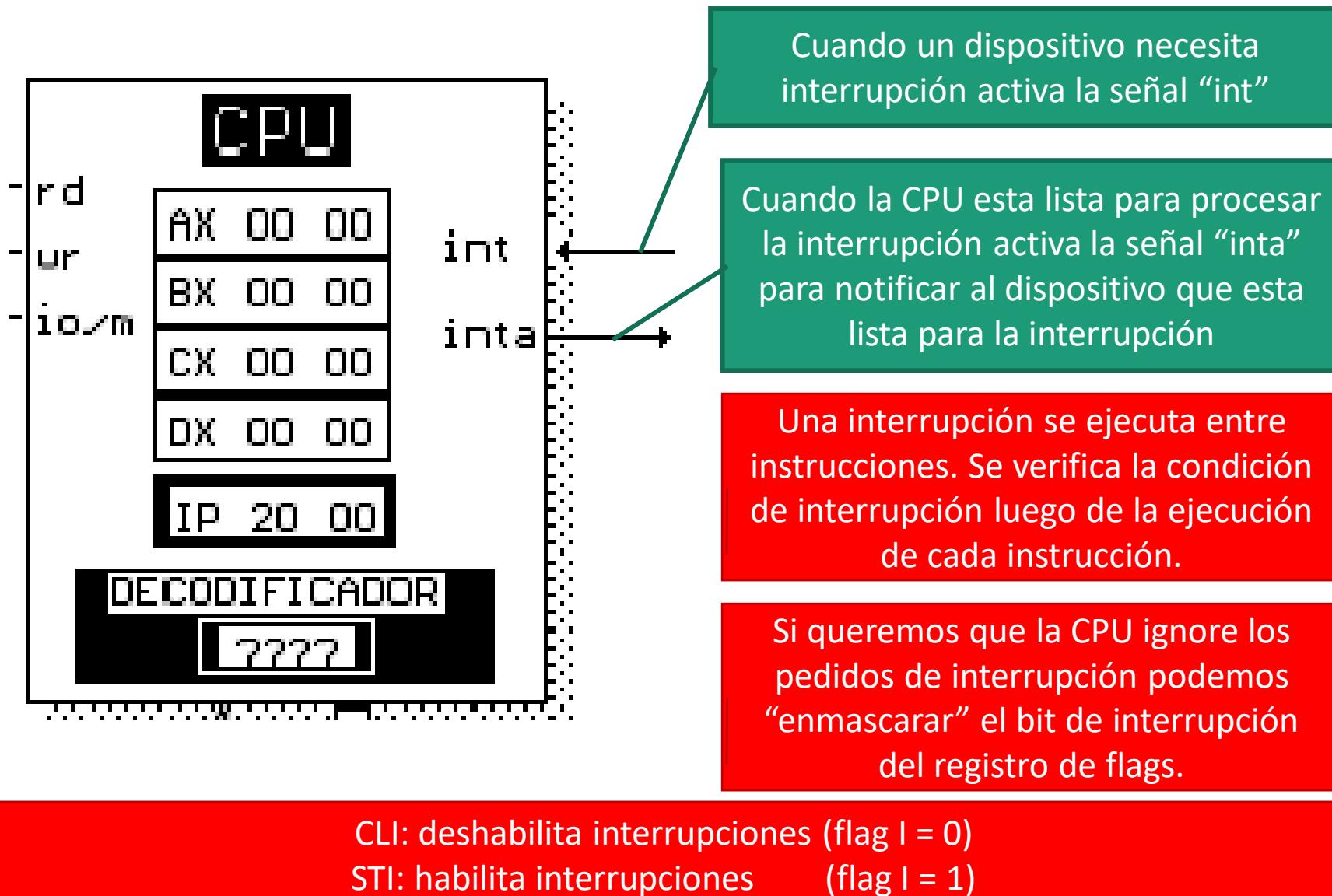
- Completa ejecución de instrucción del programa en curso
- La CPU verifica que hay interrupción de un dispositivo y no esta enmascarada (interrupciones habilitadas)
- Obtiene la posición del vector de interrupción asociada al dispositivo
- Obtiene la dirección de la subrutina de atención de interrupción de la posición del vector
- Ejecuta subrutina:
  - Se atiende el requerimiento del dispositivo
  - Se indica que el dispositivo fue atendido
  - Finaliza subrutina
- Comienza ejecución de instrucción siguiente a la que interrumpió

# Interrupciones por Hardware - Ciclo



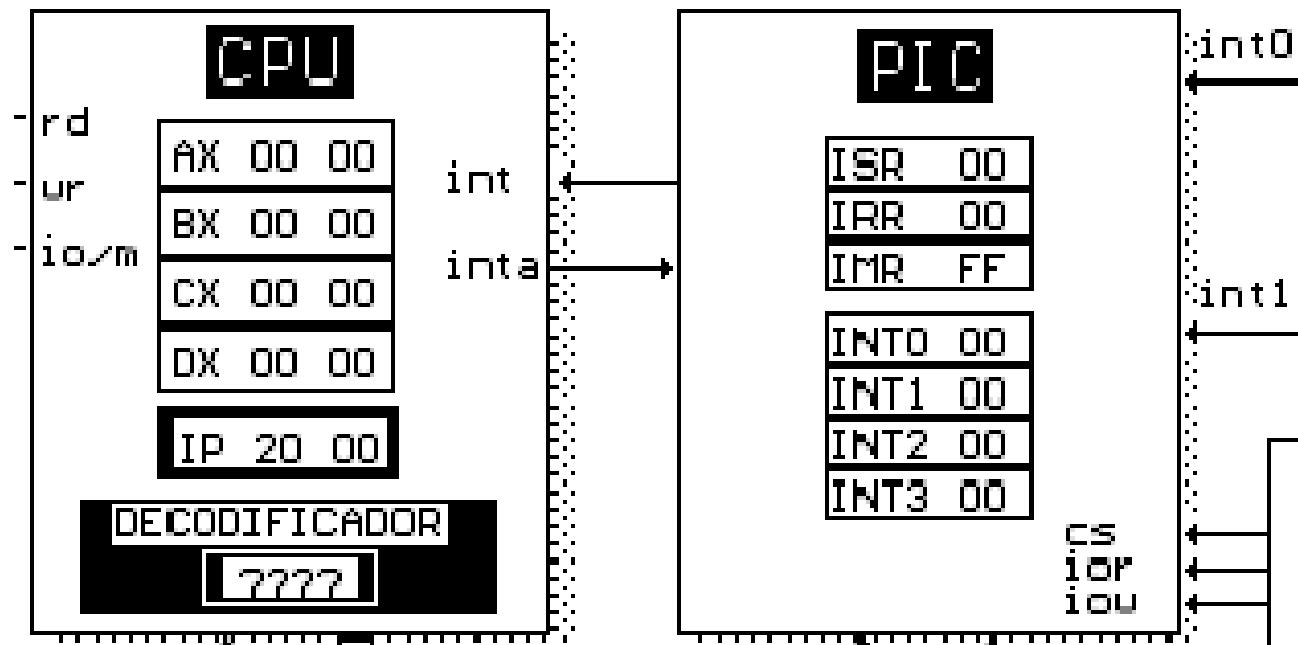
## Ciclo de Interrupción

# Interrupciones por Hardware - Señales



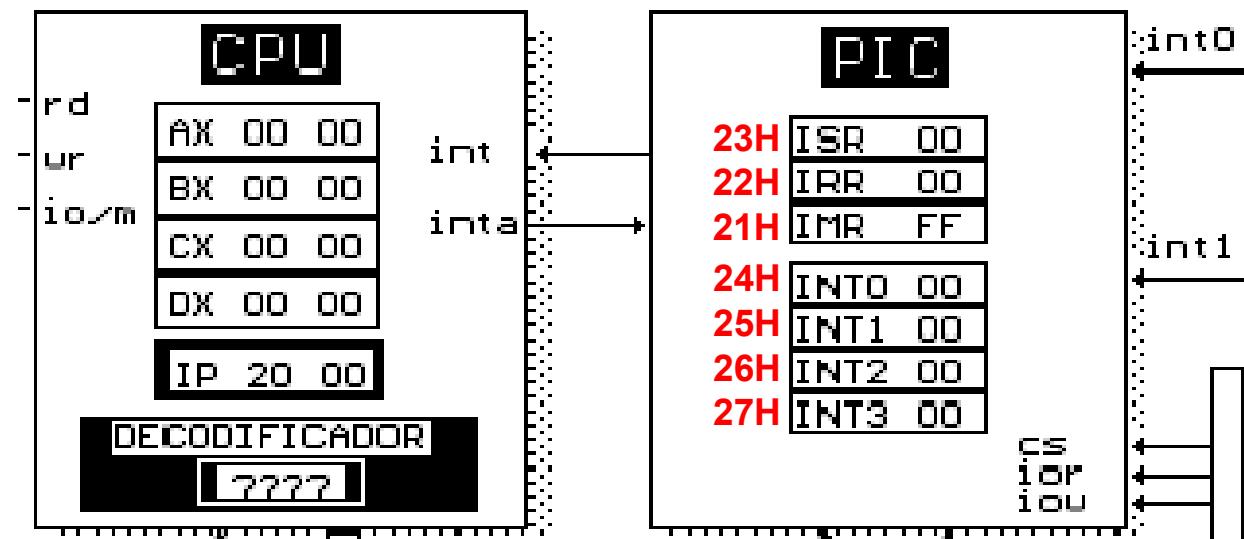
# Interrupciones por Hardware - PIC

- Con el esquema anterior solo podemos atender un dispositivo o fuente de interrupción.
- Necesitamos un dispositivo que nos permita administrar interrupciones para varios dispositivos.



# Controlador de Interrupciones - Conexión

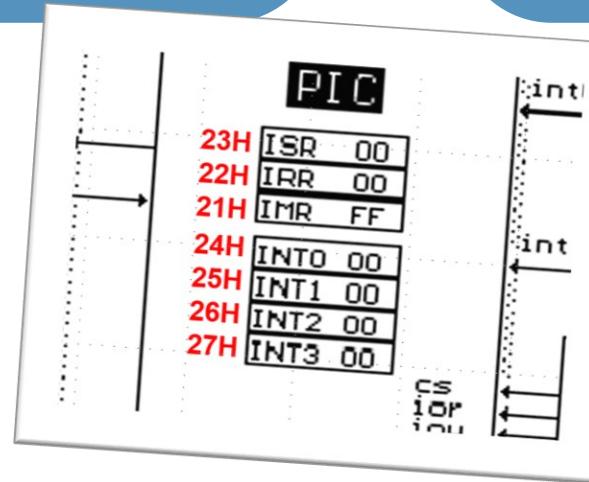
- El dispositivo PIC (Programmable Interrupt Controller) permite controlar hasta 8 fuentes (dispositivos) de interrupción.
- Esta “cableado” o conectado en la dirección de entrada/salida 20H (dirección del 1er registro).
- Tiene 12 registros que permiten configurar y consultar los dispositivos de interrupción.



# Controlador de Interrupciones - Registros

- EOI (End of Interrupt, 20H):

- solo se usa el bit 5 de este registro. Se debe poner a 1 para indicar que finalizo la atención del dispositivo que pidió interrupción

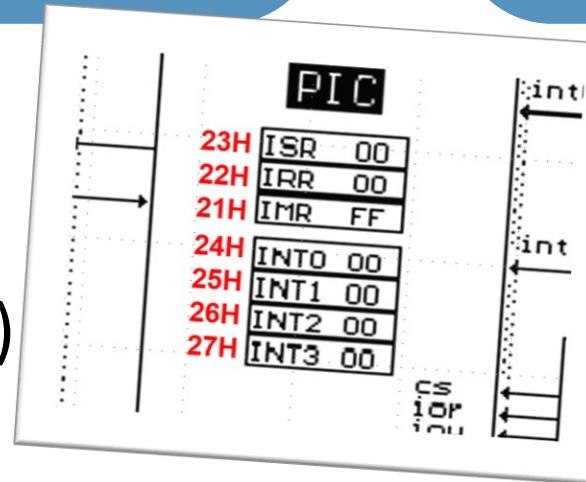


- IMR (Interrupt Mask Register, 21H):

- cada bit se asocia a una fuente de interrupción diferente:
    - 0 → habilita
    - 1 → deshabilita
  - Si un dispositivo pide interrupción y el bit asociado esta en 1, no se propaga la interrupción a la CPU (no confundir con el flag I de la CPU, estos bits inhabilitan a nivel PIC)

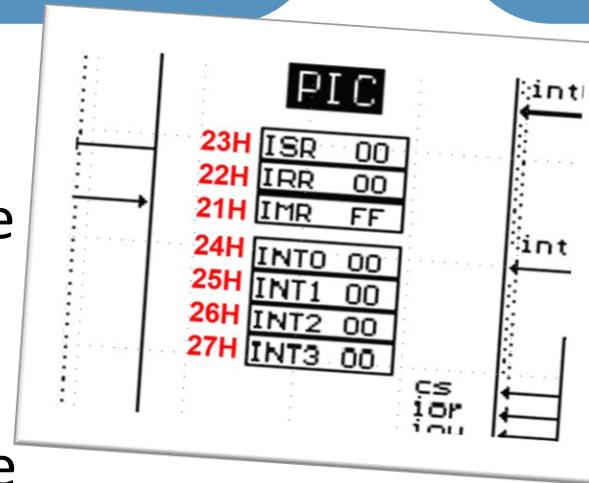
# Controlador de Interrupciones - Registros

- IRR (Interrupt Request Register, 22H):
  - cada bit se asocia a una fuente de interrupción diferente (bit 0 a dispositivo 0, bit 1 a dispositivo 1, ...)
  - indica que un dispositivo necesita atención (puede haber varios pedidos simultaneos)
- ISR (Interrupt Service Register, 23H):
  - cada bit se asocia a una fuente de interrupción diferente
  - indica que la CPU esta atendiendo un pedido de interrupción (solo un dispositivo a la vez):
    - Se pone el bit en 1 cuando la CPU acepta el pedido
    - Se pone el bit en 0 cuando la CPU indica al PIC que termino la interrupción (hay que escribir un 1 en el bit 5 del registro EOI)

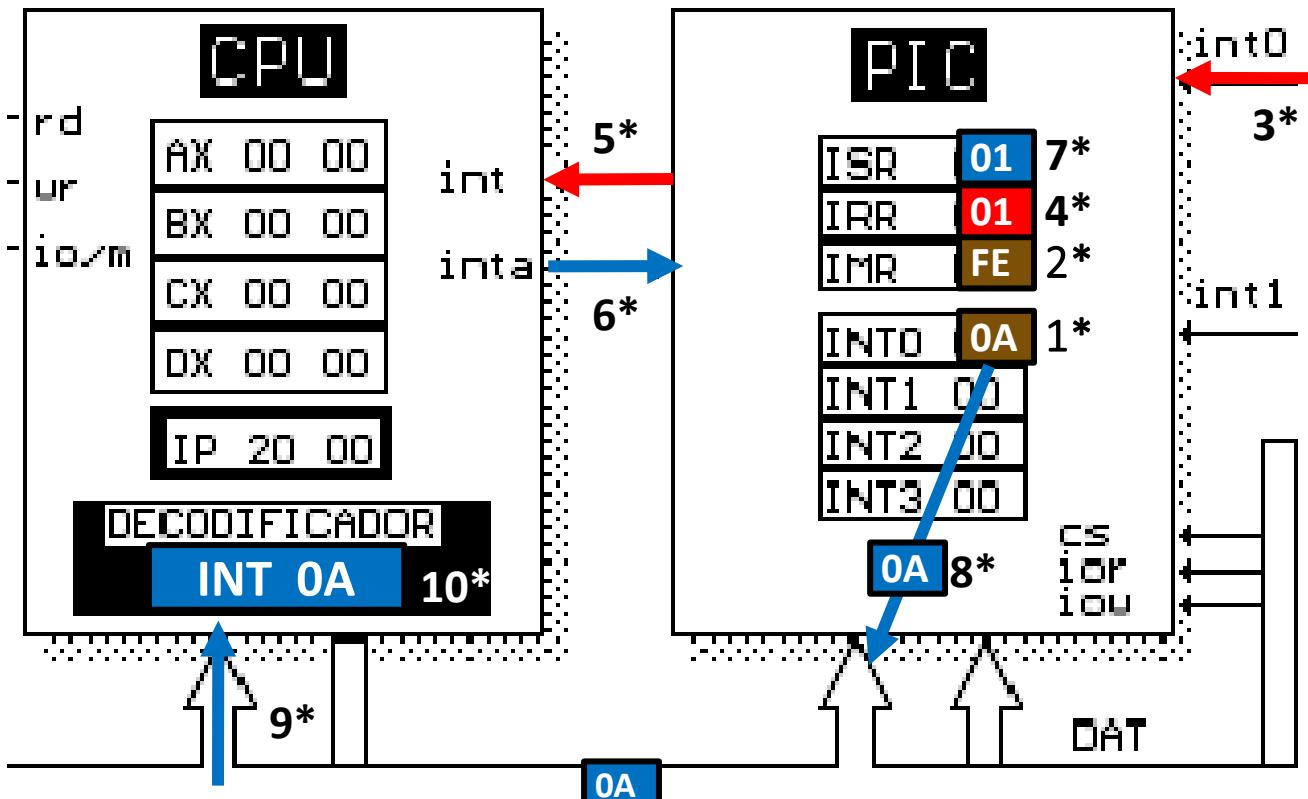


# PIC - Registros

- INT0 , 24H:
  - este registro contiene la posición del vector interrupción donde se encuentra la dirección de la subrutina de atención para el dispositivo asociado a la entrada 0 del PIC.
  - se utiliza el mismo mecanismo que la interrupción por software con la diferencia que el número de interrupción se saca de este registro
- INT1, 25H: idem pero para dispositivo 1
- INT2, 26H: idem pero para dispositivo 2
- ... ...
- INT6, 2AH: idem pero para dispositivo 6
- INT7, 2BH: idem pero para dispositivo 7



\* Orden de ejecución



### Inicialización:

0\*. Escribir en la posición 10 (0Ah) del vector de interrupción la dirección de la subrutina que atenderá al dispositivo 0

1\*. Asociar al registro INT0 del PIC la posición del vector  
2\*. Habilitar el dispositivo 0 en IMR(Feh) interrumpir

### Secuencia de Interrupción:

3\*. El dispositivo 0 activa señal de interrupción del PIC

4\*. El registro IRR del PIC muestra el pedido (01h)

5\*. Como el dispositivo 0 está habilitado por IRR (Feh) el PIC propaga la señal a la CPU

6\*. La CPU indica al PIC que acepta la interrupción (siempre y cuando el flag I esté en 1)

7\*. El PIC muestra en ISR la interrupción aceptada (01h)

8\*. El PIC escribe en el bus de datos el valor del registro INT0 (0Ah o 10) para que la CPU invoque a la subrutina

9\*. La CPU obtiene del bus la posición del vector

10\*. La CPU ejecuta la instrucción INT 0Ah (10) por lo que se obtiene la dirección de la rutina y la ejecuta.

11\*. Al finalizar la subrutina esta pone un 1 en el bit 5 del registro EOI del PIC. Esto actualiza los estados de los registros ISR e IRR

# Controlador de Interrupciones - Registros

## Tecla F10

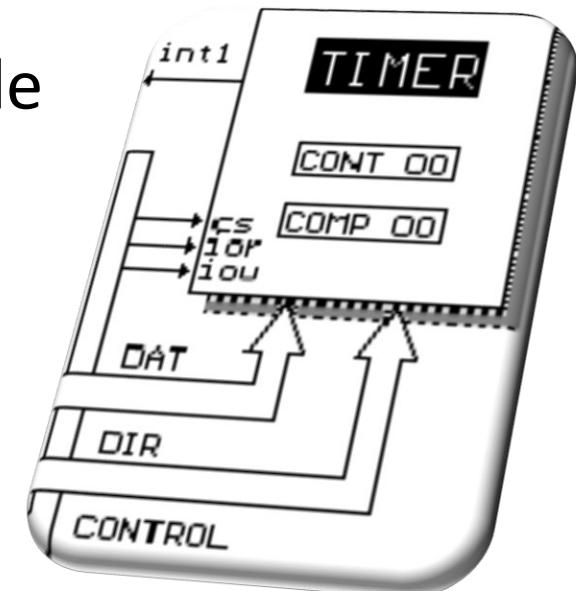


- Genera una interrupción al presionar la tecla F10
- Simula un evento de un dispositivo a través de esta tecla
- No requiere programación del dispositivo, solo la programación del PIC
- Asociada a la entrada int0 del PIC

# Dispositivos del MSX88

## TIMER (Temporizador)

- Interrumpe a la CPU transcurridos una cantidad de segundos desde su programación
- Conectado al puerto 10H de entrada/salida
- Asociado a la entrada int1 del PIC
- Tiene 2 registros:
  - CONT (contador ascendente): 10H
  - COMP (comparador): 11H
- Cuando el valor de CONT coincide con el valor de COMP genera interrupción.
- En cada interrupción es necesario restablecer el valor del registro CONT al valor original de programación para mantener la frecuencia de conteo



# Acerca de INT

## Poner atención al contexto en el que utilizamos la palabra INT

- “int” es la señal que entra a la CPU para indicar que hay una interrupción
- “INT” es la instrucción que ejecuta una interrupción.  
Ej: INT 7
- “Int0” es la señal de entrada asociada al PIC para el dispositivo 0 (Tecla F10)
- “INT0” es el nombre que le damos al registro del PIC asociado al dispositivo 0 (dirección 24H)
- “INT 0” es la interrupción por software que termina un programa
- “INT0” podría ser una constante de un programa.  
Ej: INT0 EQU 24H

# Interrupciones por Hardware - Ejemplos



## **Interrupción por hardware usando el TIMER**

Implementar a través de un programa un reloj segundero que muestre en pantalla los segundos transcurridos (00-59 seg) desde el inicio de la ejecución.

# Interrupciones por Hardware - Ejemplos

## Interrupción por hardware usando el TIMER

Para imprimir en pantalla solo puede usarse texto. Si se contabiliza usando un entero, se requiere de conversión de número a texto

Ejemplo:

Para convertir un valor numérico como el 29 se requiere:

- División para extraer la unidad y sumar para convertir en carácter:

$$29 \text{ DIV } 10 = 2 \rightarrow 2+30H = 32H \rightarrow "2"$$

- Resto para extraer la unidad y sumar para convertir en carácter:

$$29 \text{ MOD } 10 = 9 \rightarrow 9+30H = 39H \rightarrow "9"$$

# Interrupciones por Hardware - Ejemplos

## Interrupción por hardware usando el TIMER

Para evitar hacer las divisiones podemos usar una estrategia en vez de contar los segundos con un número:

$$8+1 \rightarrow 9, \quad 59+1 \rightarrow 0$$

Se puede “contar en caracteres”:

$$\text{“0”}+1 = \text{“1”} \rightarrow 30H+1=31H$$

$$\text{“1”}+1 = \text{“2”} \rightarrow 31H+1=32H$$

$$\text{“2”}+1 = \text{“3”} \rightarrow 32H+1=33H$$

...

$$\text{“8”}+1 = \text{“9”} \rightarrow 38H+1=39H$$

# Interrupciones por Hardware - Ejemplos

## Interrupción por hardware usando el TIMER

La solución sería utilizar un string o 2 variables para almacenar la cuenta en 2 caracteres inicializado con los caracteres “0” (30H)

Tres formas distintas de declararlos:

```
SEG DB 30H; ASCII del "0"  
DB 30H; ASCII del "0"
```

```
Decena DB "0"; Código ASCII 30H  
Unidad DB "0"; Código ASCII 30H
```

```
Decena DB "0"; Código ASCII 30H  
DB "0"; Código ASCII 30H
```

# Interrupciones por Hardware - Ejemplos

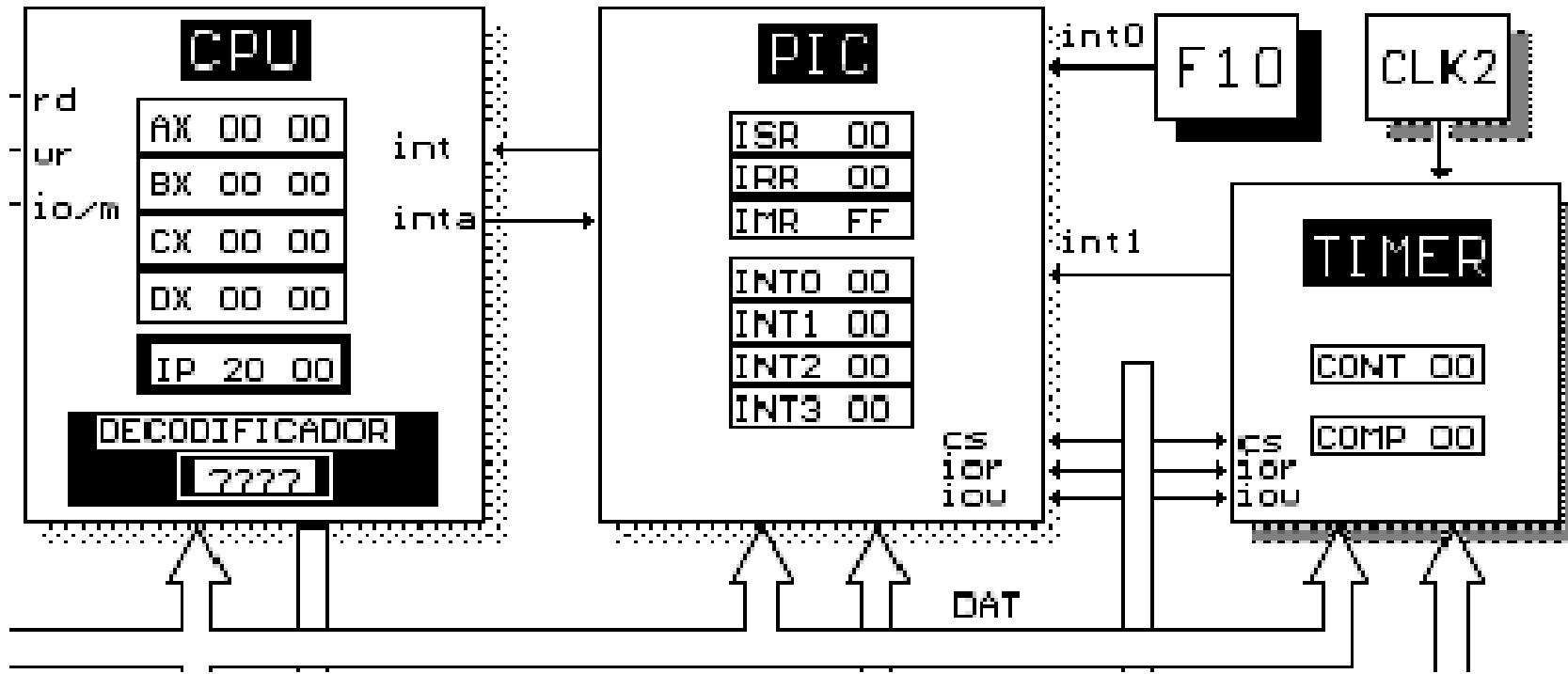
## Interrupción por hardware usando el TIMER

La solución es simple pero requiere unos ajustes:

“9”+ 1 da un carácter inválido porque el resultado tiene que ser en dos caracteres: “1” (31H) y “0” (#30H) formarían el “10”

Para solucionar el problema hay que tener en cuenta 2 casos:

- Sumar la unidad y verificar si se pasó del carácter “9” (39H). Si se pasa de la cuenta se obtiene el carácter 3AH
- Detectada esta condición hay que corregir la unidad asignándola en “0” y sumar 1 a la decena. Para los valores “0” y “9”



*Seudocódigo para configurar la interrupción del temporizador*

### Pasos para programar TIMER:

1. Deshabilitar interrupciones de CPU
2. Programar PIC:
  1. Guardar en INT1 del PIC la posición del vector que tiene la subrutina
  2. Habilitar Interrupción solo para TIMER (bit 1 del registro IMR del PIC)
3. Programar TIMER:
  1. Inicializar CONT en 0
  2. Inicializar COMP en 1
4. Habilitar interrupciones de la CPU

Pregunta: ¿Tendríamos el mismo resultado si programamos CONT con 5 y COMP con 6?

```

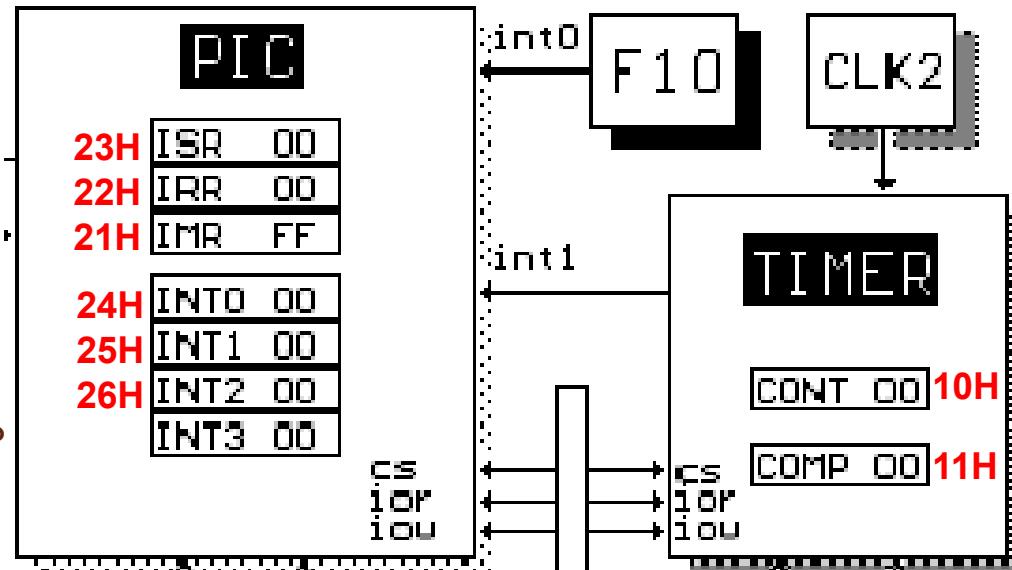
1. TIMER EQU 10H
2. PIC EQU 20H
3. EOI EQU 20H
4. N_CLK EQU 10
5. ORG 40
6. IP_CLK DW RUT_CLK

7. ORG 1000H
8. SEG DB 30H ; ASCII del "0"
9. DB 30H
10.FIN DB ?

11.ORG 3000H
12.RUT_CLK: PUSH AX ;Rutina interrup.
13. INC SEG+1
14. CMP SEG+1, 3AH
15. JNZ RESET
16. MOV SEG+1, 30H ; "0"
17. INC SEG
18. CMP SEG, 36H ; "6"
19. JNZ RESET
20. MOV SEG, 30H ; "0"
21.RESET: INT 7 ;Imprime tiempo
22. MOV AL, 0
23. OUT TIMER, AL;Reinicia CONT
24. MOV AL, EOI
25. OUT PIC, AL;PIC:Fin interrup
26. POP AX
27. IRET ;CPU Fin interrup.

28.ORG 2000H
29. CLI
30. MOV AL, 0FDH ;Máscara para Disp
31. OUT PIC+1, AL ;PIC: registro IMR
32. MOV AL, N_CLK
33. OUT PIC+5, AL ;PIC: reg. INT1
34. MOV AL, 1
35. OUT TIMER+1, AL ;TIMER: reg. COMP
36. MOV AL, 0
37. OUT TIMER, AL ; TIMER: reg. CONT
38. MOV BX, OFFSET SEG
39. MOV AL, OFFSET FIN-OFFSET SEG
40. STI
41.LAZO: JMP LAZO
42.END

```



# Arquitectura de computadoras

*Genaro Camele*

# Interrupciones

# Interrupciones

*Tipos*

Las interrupciones permiten pausar la ejecución del programa principal para realizar una operación específica

Vamos a ver interrupciones de dos tipos

## SOFTWARE

- Se invocan desde el código
- Las vamos a ver en medio segundo

## HARDWARE



(para después)

# 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 carácter desde teclado

**INT 7**: imprime un string en pantalla

Veamos algunos ejemplos....

# Interrupciones por Software

*INT 0*

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

A partir de ahora vamos a  
utilizar INT 0 en lugar de  
HLT

# Interrupciones por Software

*INT 6*

**INT 6** lee un carácter desde teclado

Cuando invocamos la interrupción se guarda el carácter leído en la **dirección** que contiene en ese momento **BX**

Escribir un programa que lea un carácter y lo guarde en la variable **LEIDO**

**ORG 1000H**

LEIDO DB ?

LEIDO: ~~Basura~~ 61H

BX: ~~Basura~~ 1000H

**ORG 2000H**

MOV BX, **OFFSET** LEIDO ←

**INT 6** ←

(presiona la “a”)

**INT 0** ←

END

# Interrupciones por Software

INT 7

**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 ?

**BX:** ~~Basura~~ 1000H

**AL:** ~~Basura~~ 1CH (24)

**ORG 2000H**

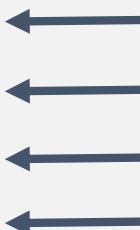
MOV BX, **OFFSET** MENSAJE

MOV AL, OFFSET FIN - OFFSET MENSAJE

**INT 7**

**INT 0**

END



Imprime!

# 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

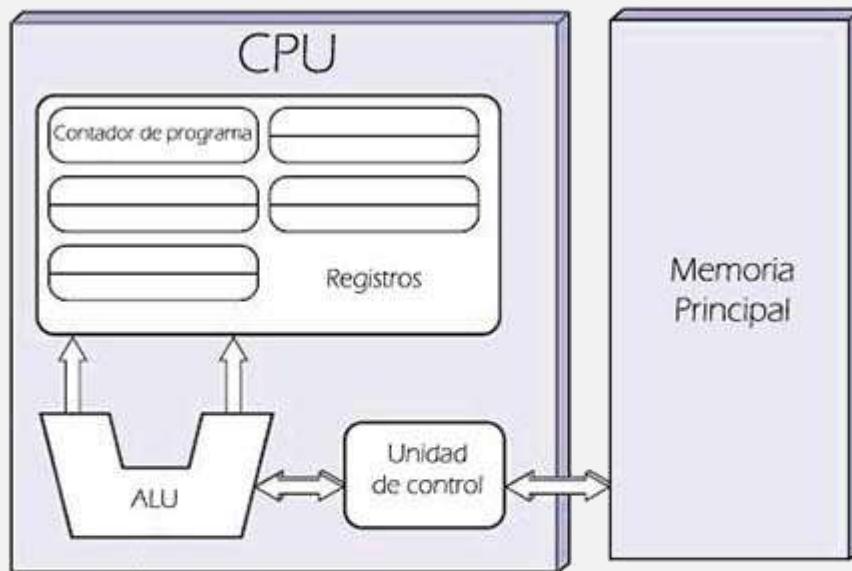


**Entrada/Salida**

# Memoria E/S

*Definición*

Hasta ahora teníamos este esquema:



**MOV  
ADD  
SUB**

...

**Memoria de E/S**



?

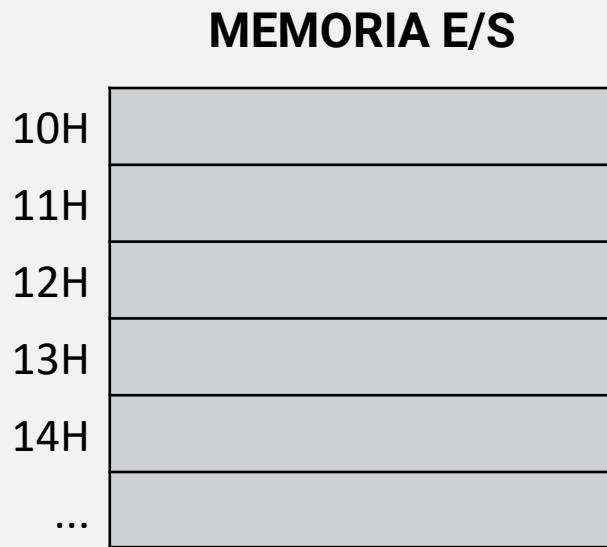


**Dispositivos de E/S**

# Memoria E/S

*Lectura y escritura en E/S*

La memoria de E/S es igual a la memoria común!



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**~~

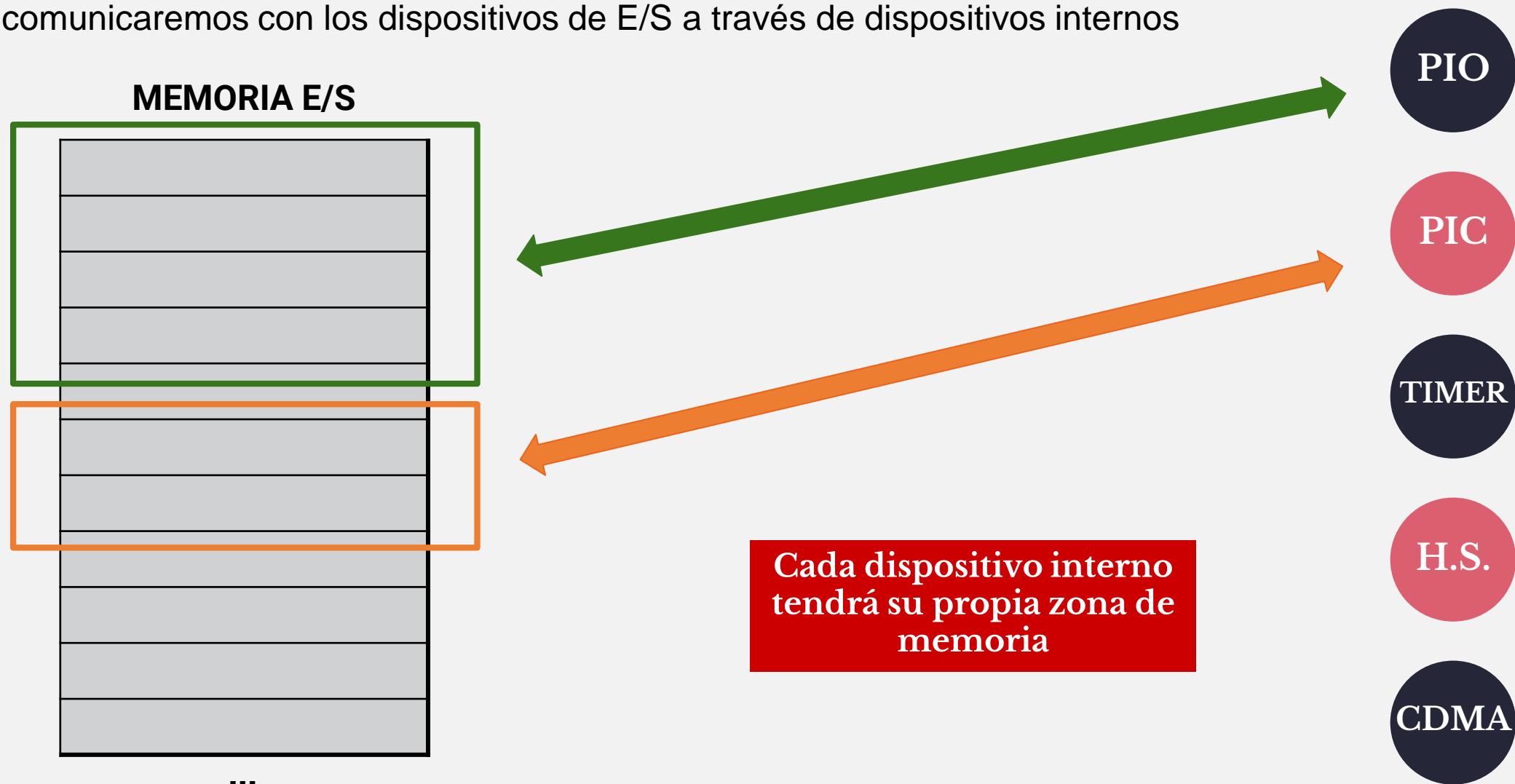
**MOV AL, 30**

**OUT 50H, AL**

# Memoria E/S

*Dispositivos internos*

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





PIO

*Puerto Paralelo de E/S*

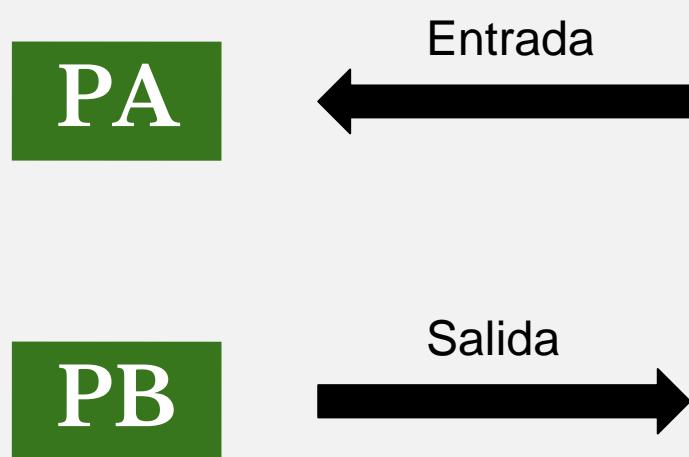
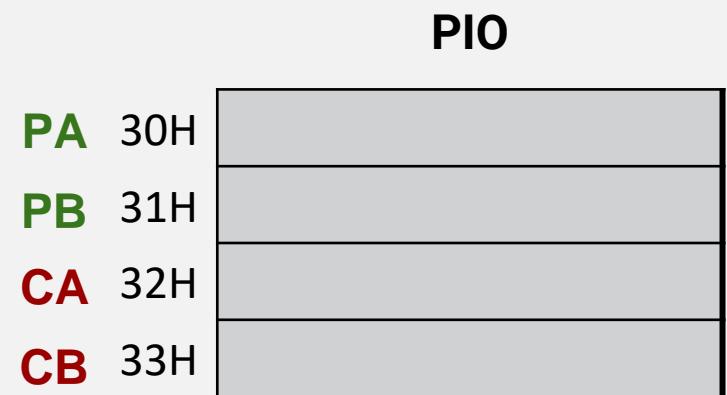
# PIO

*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)

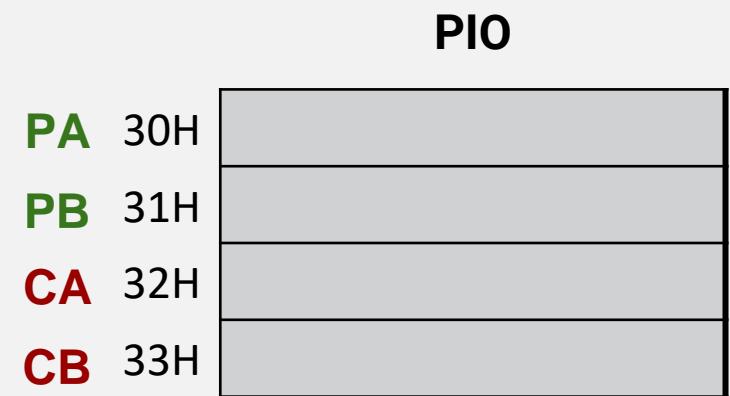


# PIO

## *Fucionamiento*

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)

```
MOV AL, 11111110b  
OUT 32H, AL ; CA = 11111110
```

# PIO

*Ejemplo salida*

1. Prender todas las luces. Recordar que:

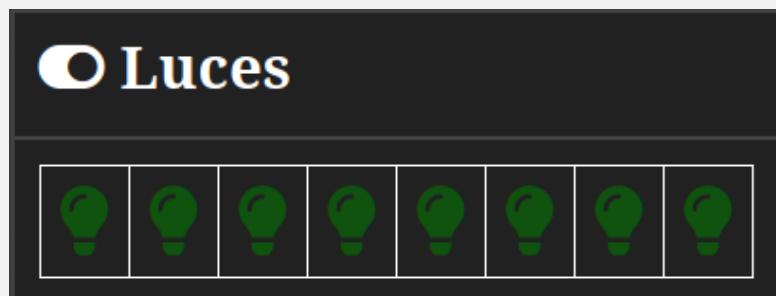
- Las luces están ligadas al puerto **PB**. 1 significa encendida
- Las queremos a todas de salida!

*MOV AL, 00000000b*

*OUT 33H, AL ; CB = 00000000*

*MOV AL, 11111111b*

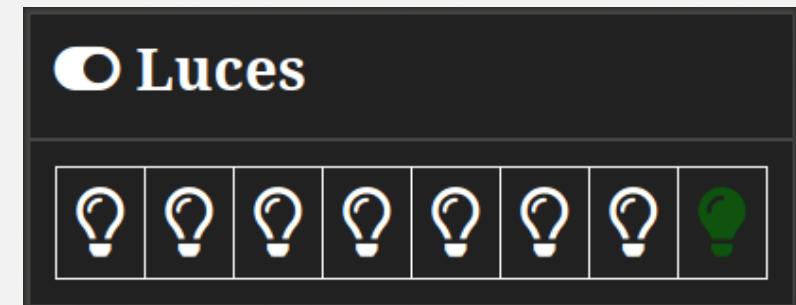
*OUT 31H, AL ; PB = 11111111*



2. Prender solo la primera (desde derecha)

*MOV AL, 01H*

*OUT 31H, AL ; PB = 00000001*



Si! Podemos usar  
hexadecimales y  
decimales!

# PIO

*Ejemplo 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
```

3. Escribimos en **PB**

```
OUT 31H, AL
```

Solo queda hacerlo infinitas veces!

# Interrupciones

*Por hardware*

# Dispositivos vs CPU/Memoria

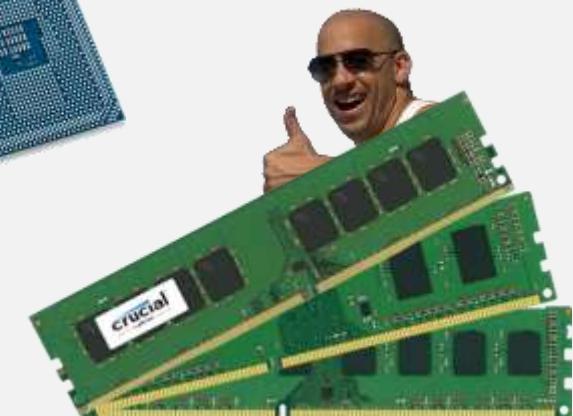
## DISPOSITIVOS

~1.000 ops/seg



## CPU/MEMORIA

~1.000.000 ops/seg



¡Los dispositivos deberían esperar a la CPU y no viceversa!

# Dispositivos

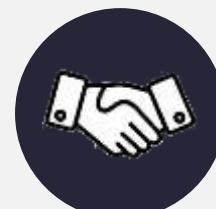
Nos vamos a manejar con 4 dispositivos externos



*F10*



*Timer*



*Handshake*



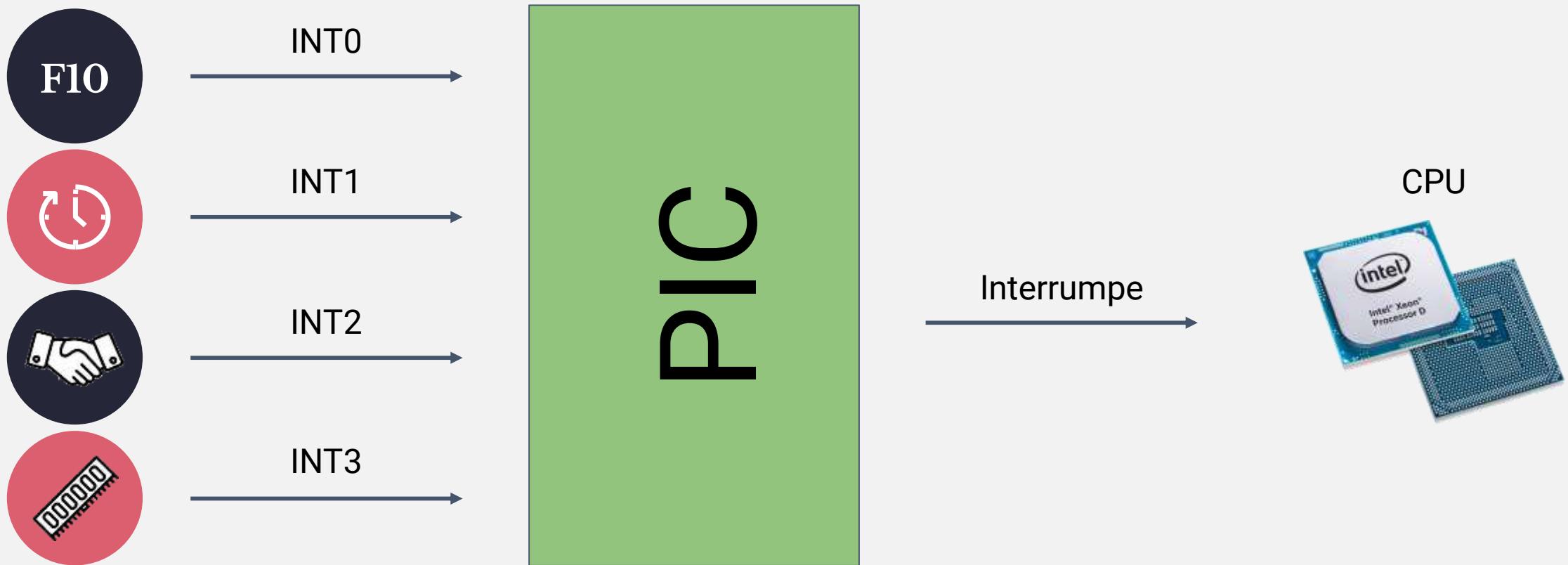
*CDMA*

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

# PIC

*Programmable Interface Controller*

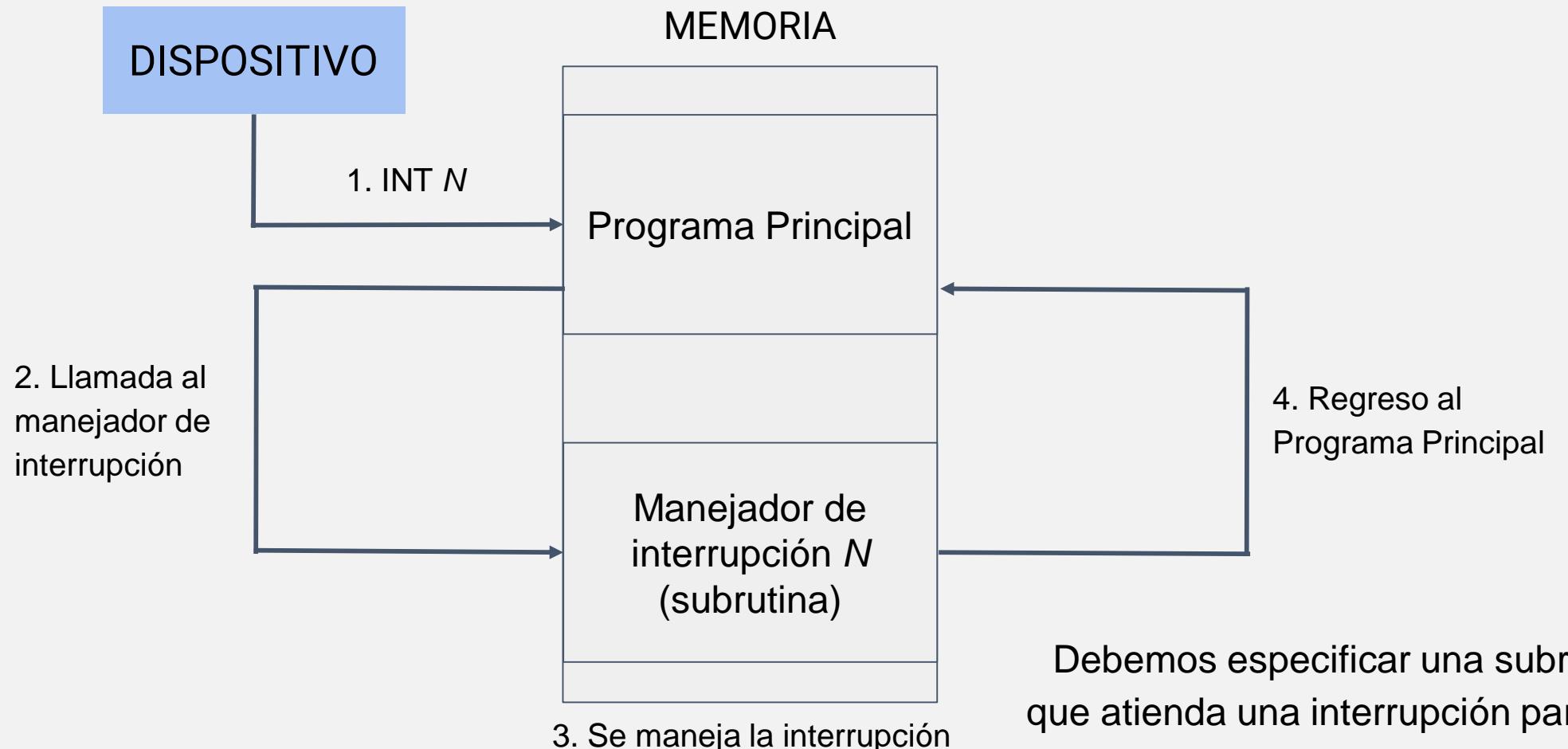
Los dispositivos interrumpen a la CPU a través del PIC



# PIC

*Programmable Interface Controller*

¿Cómo funciona?



# PIC

*Ejemplo*

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**

# PIC

*Ejemplo*

Contar las veces que se presionó la tecla F10 en DL

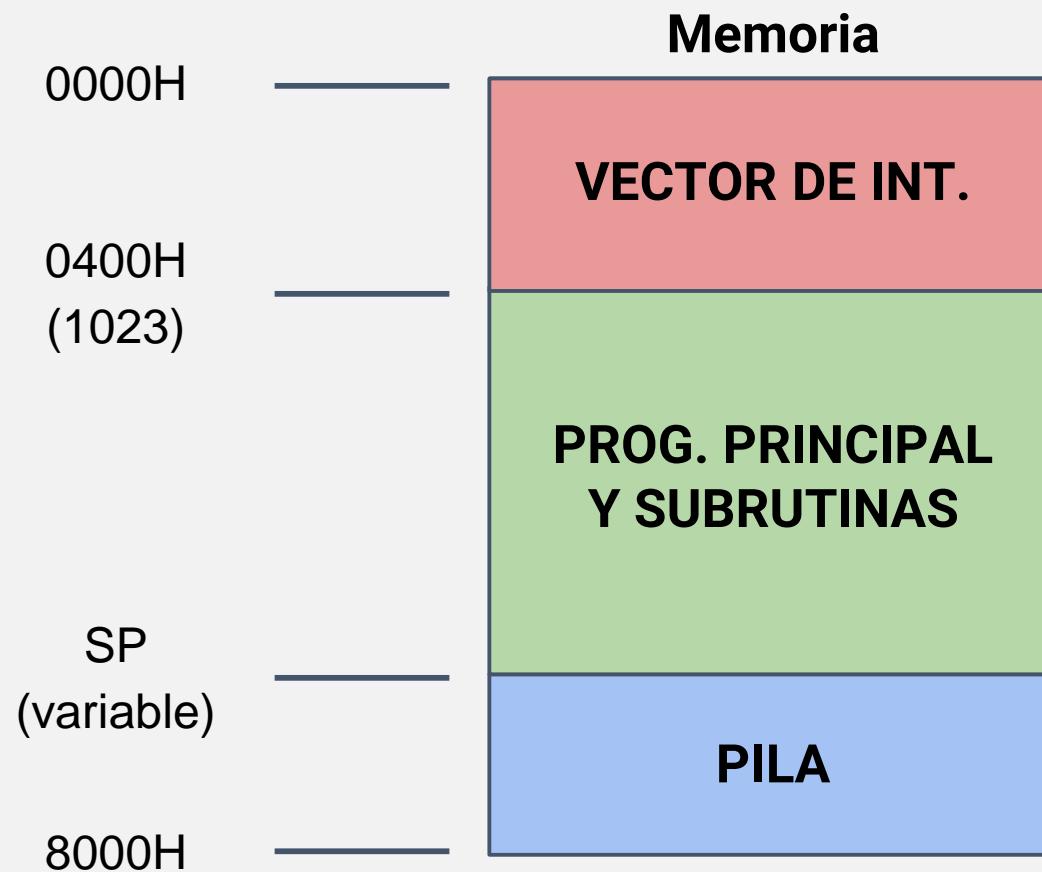
1. Escribir la subrutina que se ejecutará cuando se produzca la interrupción (que finaliza con **IRET**)

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

# PIC

*El vector de interrupciones*

Hasta ahora conocíamos una memoria con lugar para el programa, las subrutinas y la pila, pero...



- 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

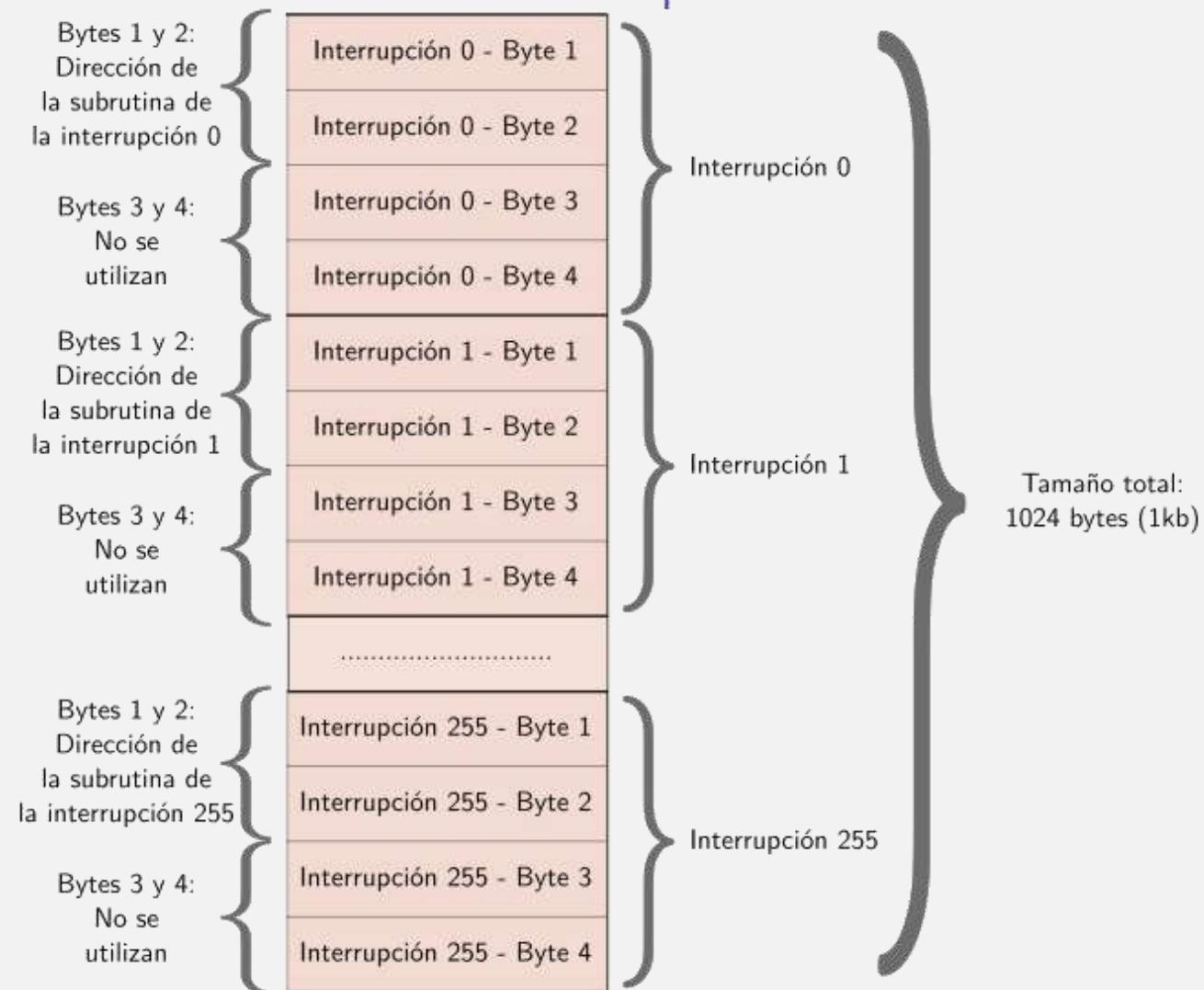
# PIC

## *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 dirección 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 posición 20 = 3000H

...

# PIC

*Ejemplo*

Contar las veces que se presionó la tecla F10 en DL

## 4. Configurar el PIC

| PIC   |     |
|-------|-----|
| EOI   | 20H |
| IMR   | 21H |
| IRR   | 22H |
| ISR   | 23H |
| INT 0 | 24H |
| INT 1 | 25H |
| INT 2 | 26H |
| INT 3 | 27H |

- 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**

# PIC

*Ejemplo*

El **PIC** contiene los siguientes campos

**PIC**

|                  |  |  |
|------------------|--|--|
| <b>EOI</b> 20H   |  | Le avisa al <b>PIC</b> que la interrupción ya fue atendida |
| <b>IMR</b> 21H   |  | Para habilitar o deshabilitar alguna interrupción          |
| <b>IRR</b> 22H   |  | Indica cuáles dispositivos externos solicitan interrumpir  |
| <b>ISR</b> 23H   |  | Indica cuál dispositivo externo está siendo atendido       |
| <b>INT 0</b> 24H |  | Contiene <i>ID</i> asignado al F10                         |
| <b>INT 1</b> 25H |  | Contiene <i>ID</i> asignado al Timer                       |
| <b>INT 2</b> 26H |  | Contiene <i>ID</i> asignado al Handshake                   |
| <b>INT 3</b> 27H |  | Contiene <i>ID</i> asignado al CDMA                        |

# 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 la subrutina de la interrupción debemos poner el valor 20H en el **EOI**

**MOV AL, 20H**

**OUT 20H, AL ; EOI = 20H**

**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

# PIC

*Ejemplo*

Debemos configurar lo que nos interesa!

Cuando termina la interrupción  
avisamos al **EOI**

**ORG 3000H**

; Subrutina que atiende la interrupción de F10

CONTAR: INC DL

; Aviso al EOI que termina la  
subrutina

**MOV AL, 20H**

**OUT 20H, AL** ; EOI = 20H

**IRET**

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**

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 dirección 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 posición 20 = 3000H**

**CLI**

**MOV AL, 11111110b**

**OUT 21H, AL**

**MOV AL, 5**

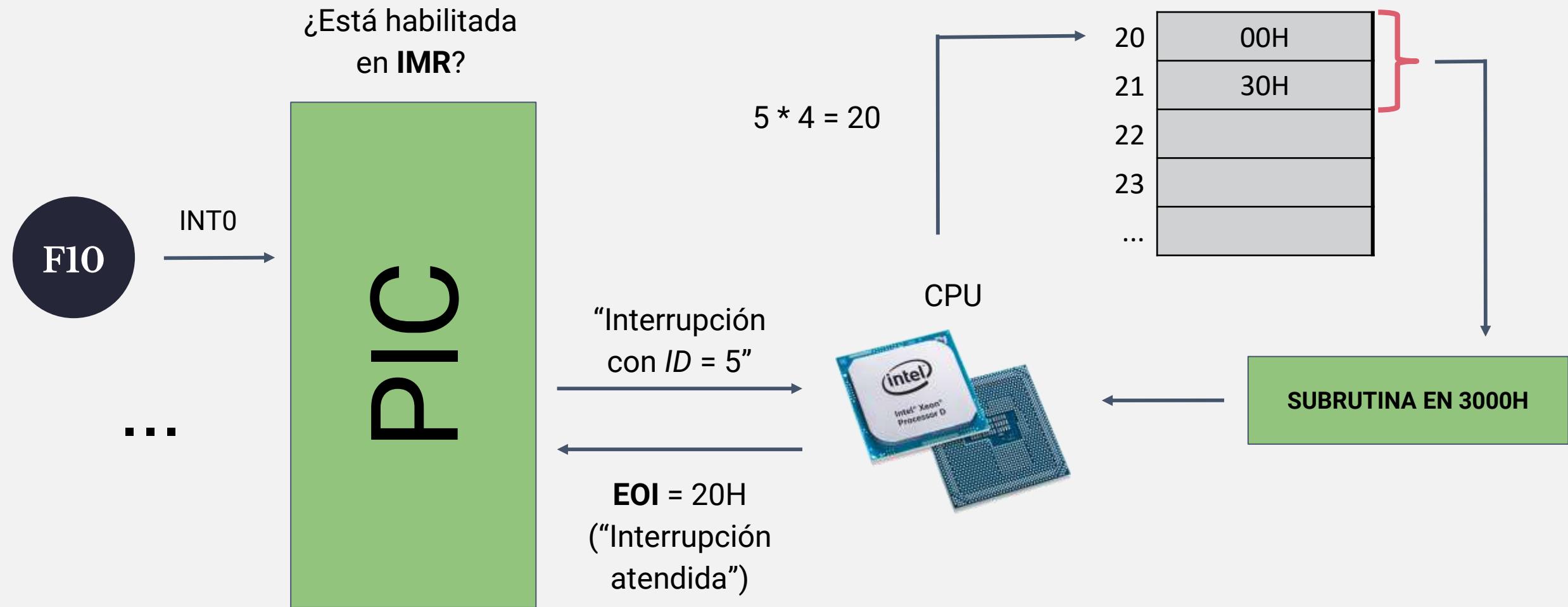
**OUT 24H, AL**

**STI**

...

# Interrupciones por Hardware

*Resumen*



# Trucazo

*Constantes*

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**

...

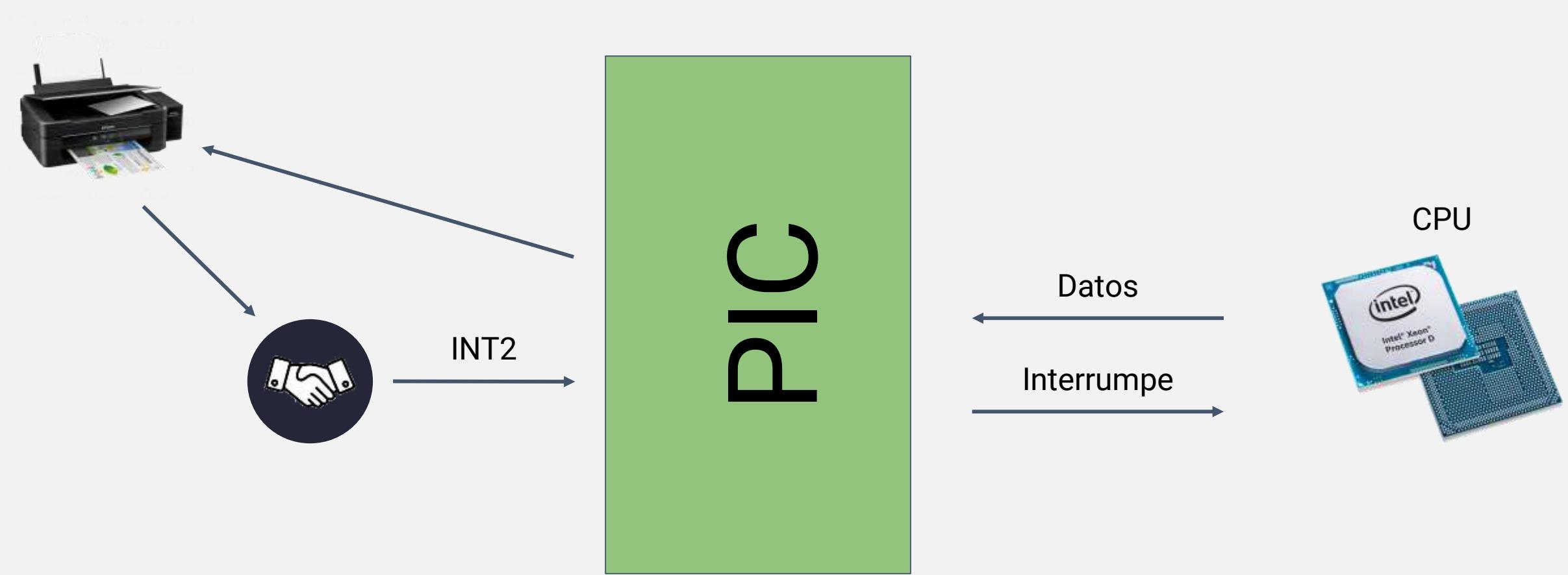


Handshake

# Handshake

*Definición*

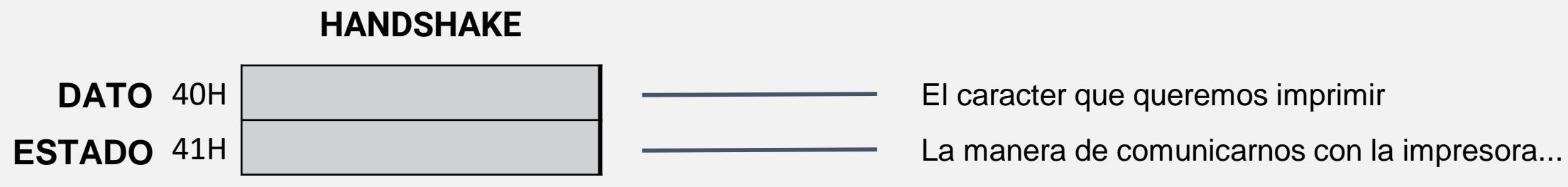
Es una abstracción de la impresora



# Handshake

*Registros*

Así como el timer tiene **COMP** y **CONT** el handshake tiene sus propios registros



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

# Handshake

*Registros*

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

**SALIDA**



¿INT?

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

**ENTRADA**



¿INT?

STROBE      BUSY

- **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)

1. Debemos configurar ¿En qué configuramos el bit de **INT**? En 0! No queremos interrupciones!
2. Consultaremos constantemente si está libre Chequear si el bit **Busy** = 0
3. Cuando la impresora esté libre mandamos el carácter a **DATO** (40H)

# 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 = 0xxxxxxxx
```

```
; 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 carácter a **DATO** (40H)

# Handshake

## Ejercicio 2

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

### 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
```

# Handshake

## Ejercicio 2

### **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 = 1xxxxxxxx

; 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

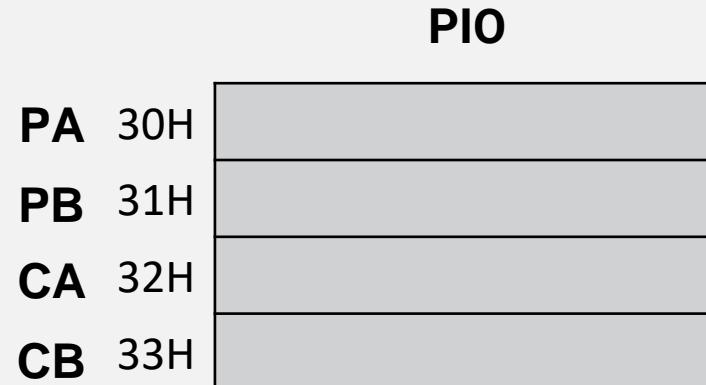
# Impresora

*Esta vez por PIO*

# Impresora

*Configuración por PIO*

Recordemos la estructura del PIO



Como en el HandShake el estado  
será de escritura y lectura

# Registro de Estado

Veamos cuáles bits del registro **estado** son de **entrada** y cuáles de **salida**...



- **Bit 0 (busy)** - 1 si está ocupada la impresora, 0 si está libre
- **Bit 1 (strobe)** - seteando el bit en 1 le avisamos a la impresora que dejamos un carácter en **DATO** para que lo imprima

# Impresora

*Ejercicio*

Escribir un programa que envíe datos a la impresora a través del **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!
3. Consultaremos constantemente si está libre Chequear si el bit **Busy** = 0
4. Cuando la impresora esté libre mandamos el carácter 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

Del punto 3 al 6 debemos repetirlo para cada carácter

# Impresora

*Ejercicio*

Escribir un programa que envíe datos a la impresora a través del **PIO**

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

**ORG 1000H**

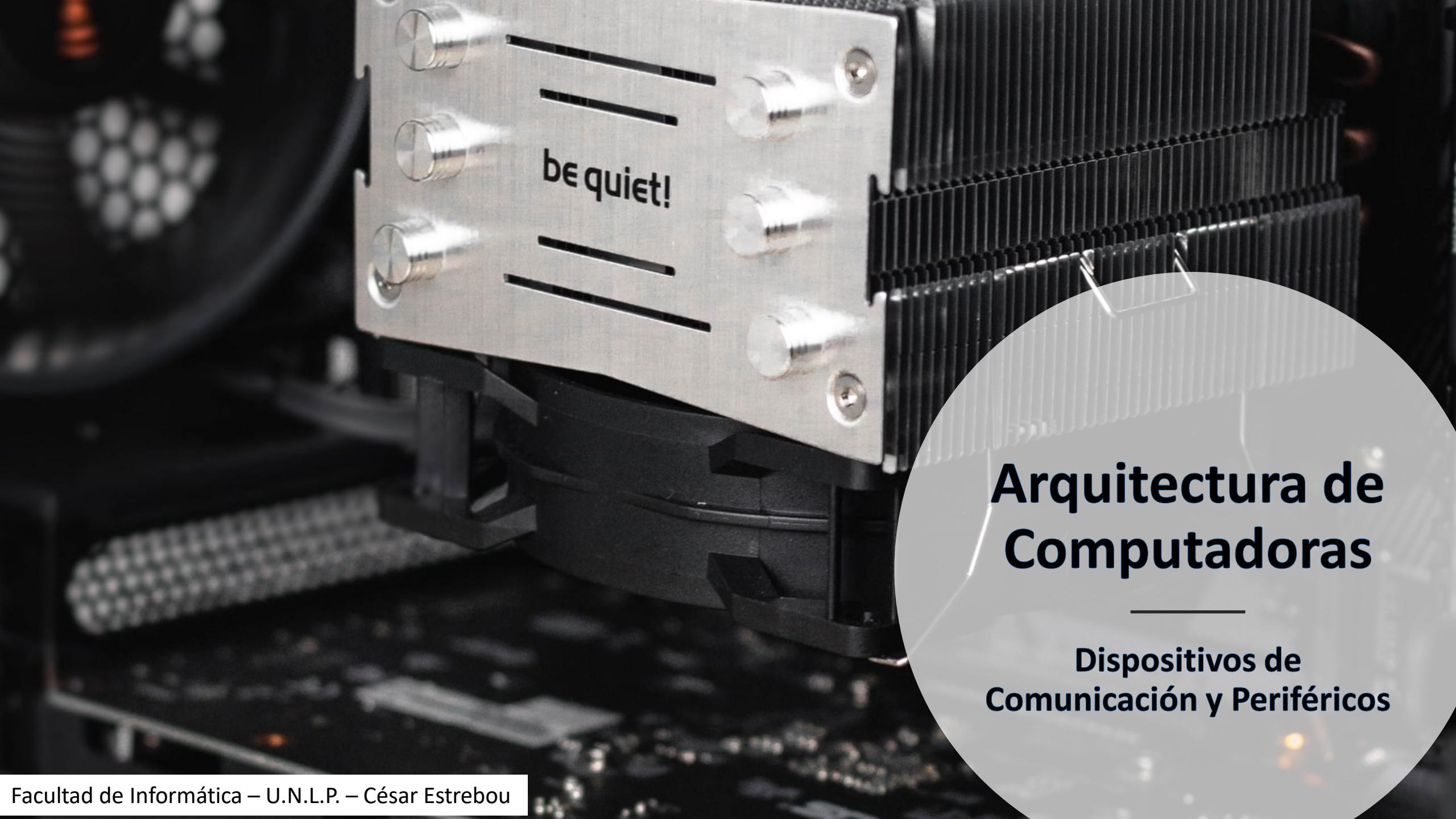
```
MENSAJE DB "Imprimiendo con el PIO!"  
FIN DB ?
```

**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 carácter a carácter 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 carácter  
OUT PB, AL ; Lo envío 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 posición  
CMP BX, OFFSET FIN ; Chequeo si llegue al final  
JNZ POLL

```
INT 0  
END
```



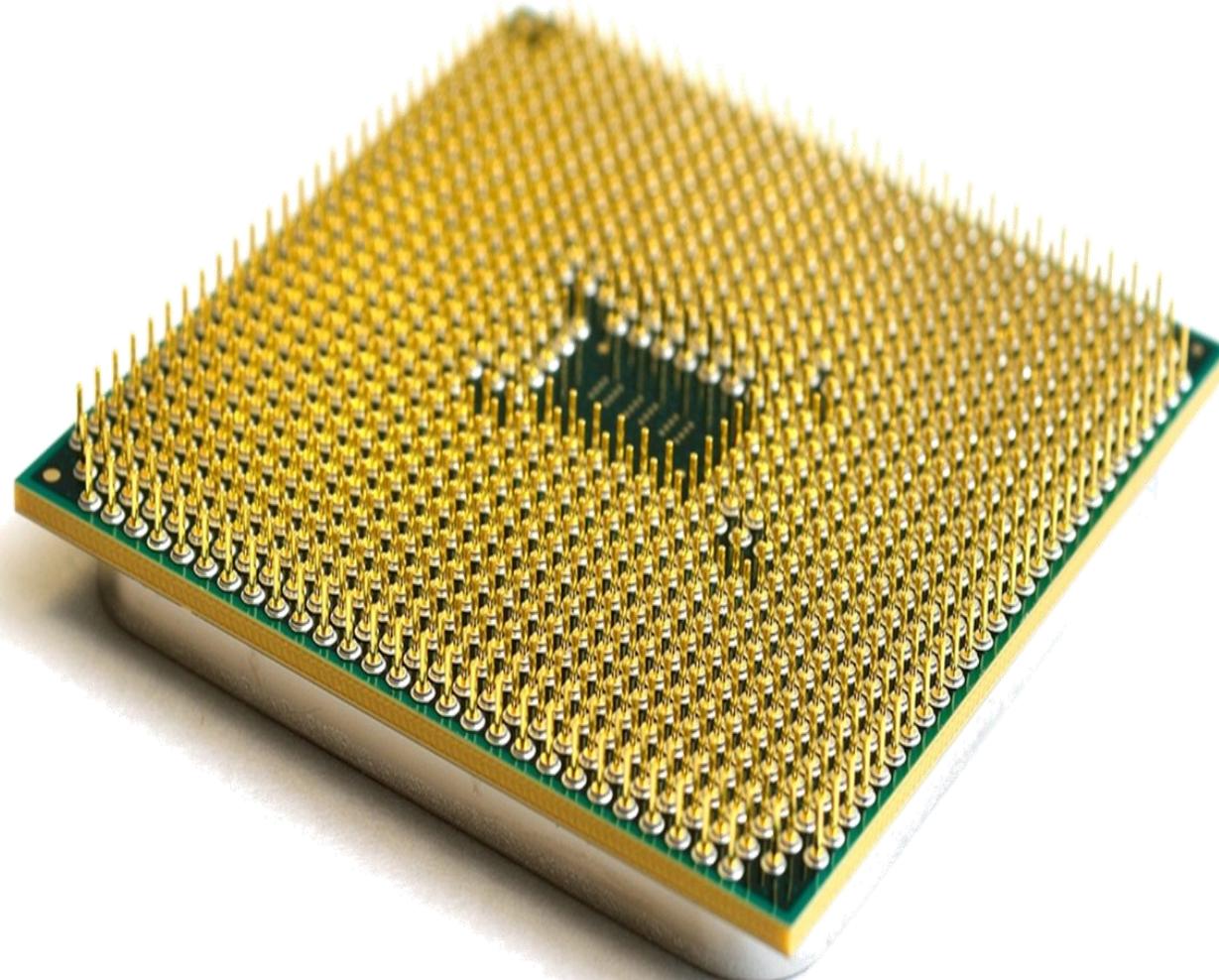
# Arquitectura de Computadoras

---

Dispositivos de  
Comunicación y Periféricos

# Agenda

## ► Temas



### 01 Periféricos del MSX88

Introducción. Barra de interruptores, barra de leds, impresora paralelo, impresora serie

### 02 Entrada / Salida Con PIO

Descripción y funcionamiento. Puertos y configuración . Ejemplos con interruptores/leds e impresora

### 03 Entrada / Salida con Handshake

Descripción y funcionamiento. Puertos y configuración. Ejemplos con y sin interruptores

### 04 Entrada / Salida con USART

Descripción y funcionamiento. Puertos y configuración. Protocolos. Ejemplos con impresora

### 05 Transferencias con DMA

Descripción y funcionamiento. Puertos y configuración. Transferencia por bloque y demanda. Ejemplos



# Periféricos del MSX88

# Periféricos

Periféricos del MSX88 (Hardware/dispositivos “fuera” del MSX88):

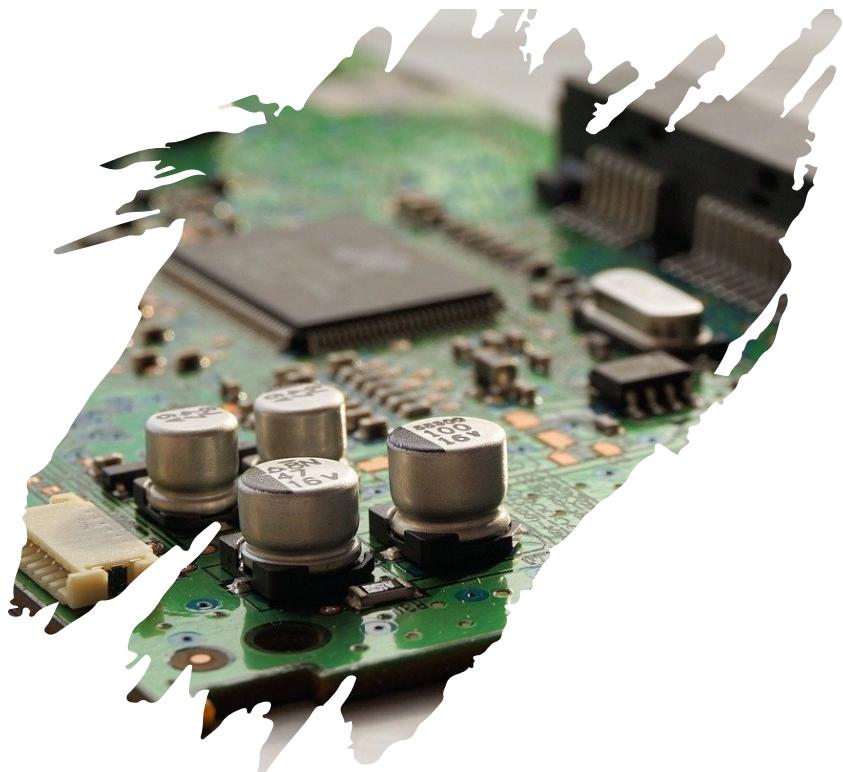
- Barra de Leds
- Interruptores
- Impresora:
  - Paralela
  - Serie

Comunicación:

- Necesitamos hardware /`dispositivos internos ( adicionales) para poder comunicarnos con hardware/dispositivos externos
- Necesitamos entender como funciona cada dispositivo interno y cada periférico

# Periféricos

Dispositivos del MSX88 para comunicación y transferencia de datos



## PIO

Dispositivo de comunicación **general** (PIO, Peripheral Input/Output o Entrada/Salida Periférica) que permite conectarnos con **cualquier** periférico conectado.

## Handshake

Dispositivo de comunicación **dedicado** que permite comunicarnos con periféricos con comunicación **paralela** de manera eficiente. Ej Impresora paralela

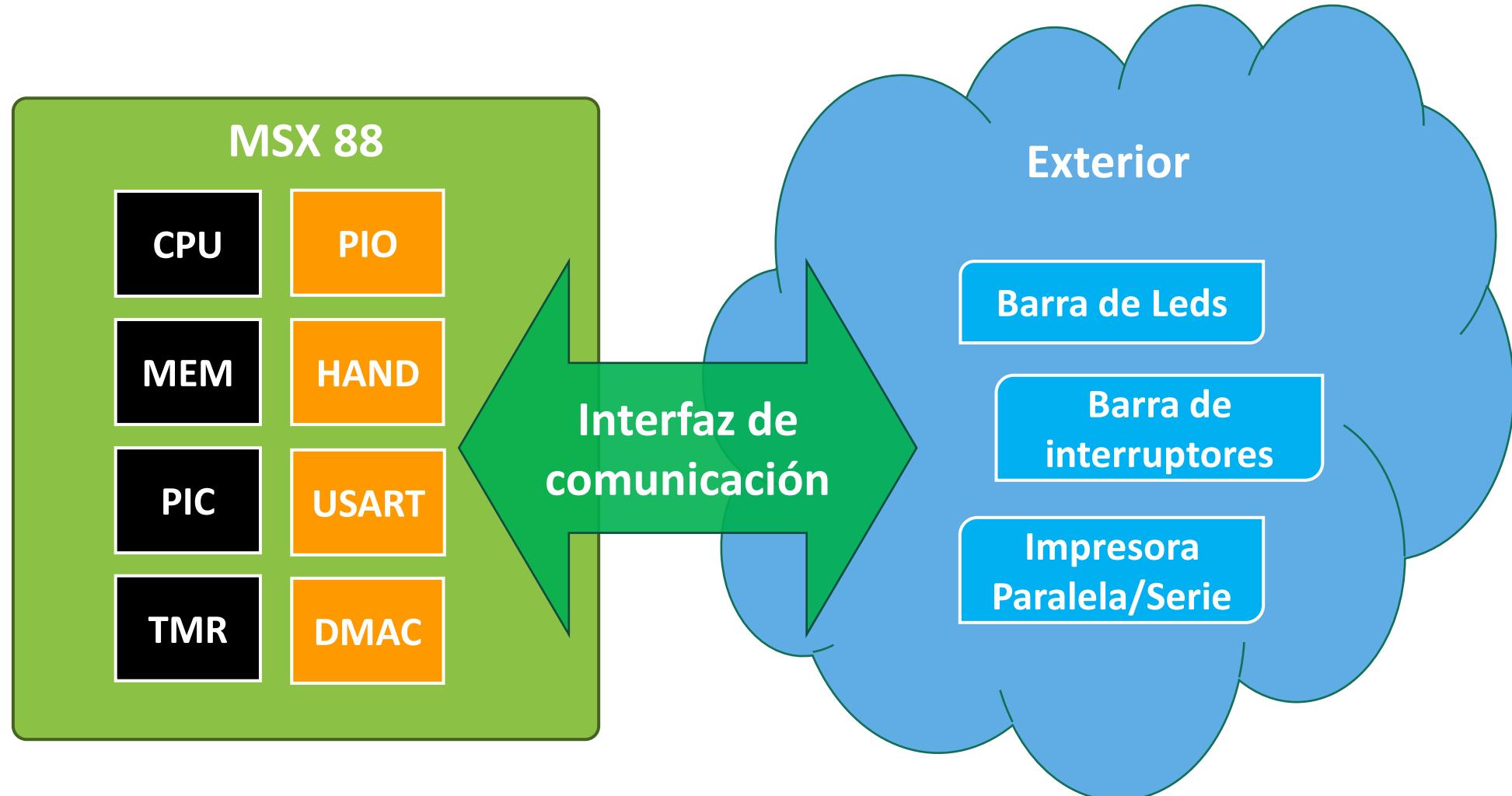
## USART

Dispositivo de comunicación **dedicado** que permite comunicarnos con periféricos con comunicación **serie** de manera eficiente. Ej: Impresora serie

## CDMA

Dispositivo (CDMA, Controlador de Acceso Directo a memoria) que permite hacer **transferencias** eficientes entre **memoria** y **dispositivos**. Ej: Memoria-Memoria o Memoria-Dispositivo

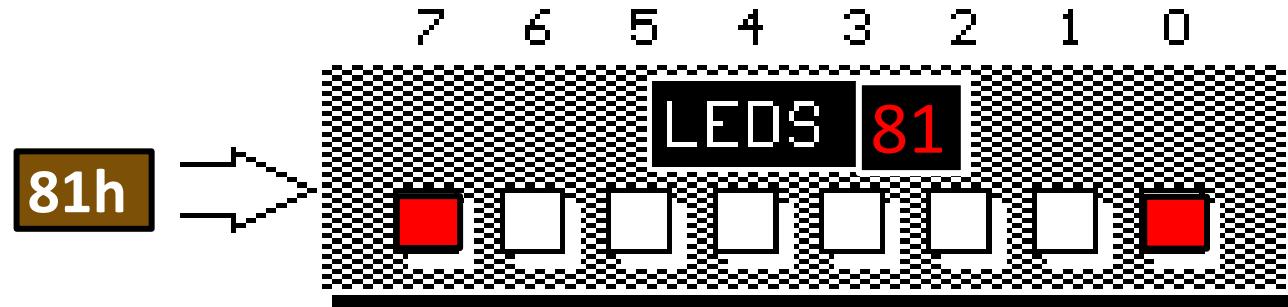
# Periféricos



# Periféricos – Barra de leds



- Es un grupo de 8 leds (diodos emisores de luz)
- Se conecta en el MSX88 a un puerto que permite controlar el estado de cada led (8 bits = 8 leds)
- Para encender un led se pone en 1 el bit deseado en el puerto



Si queremos encender el bit 7 y 0 y mantener apagados los demás hay que enviar a la barra de leds el valor 81h, 129 o 10000001b

# Periféricos – Barra de leds



- La funcionalidad de los leds es didáctica
- En aplicaciones reales, con el **hardware adecuado**, con 1 bit es posible controlar dispositivos que tienen 2 estados (apagado/encendido):



## Bombas Eléctricas

Bombas para riego/tanques



## Motores eléctricos

Apertura y cierre de puertas



## Interruptores eléctricos

Encendido y apagado de luces



## Accionamiento de Alarmas

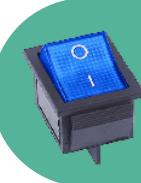
Activar/desactivar alarmas



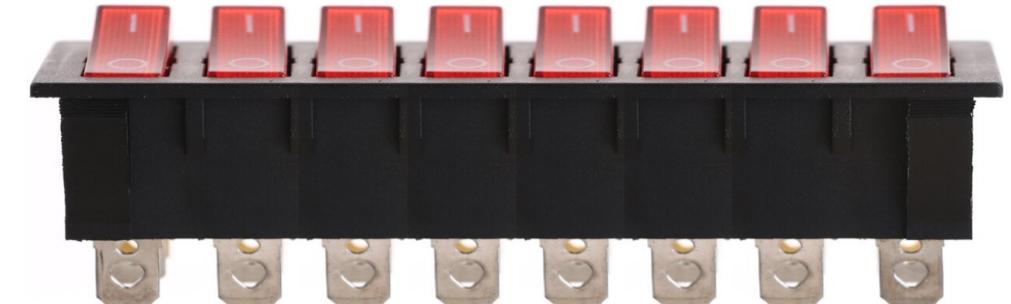
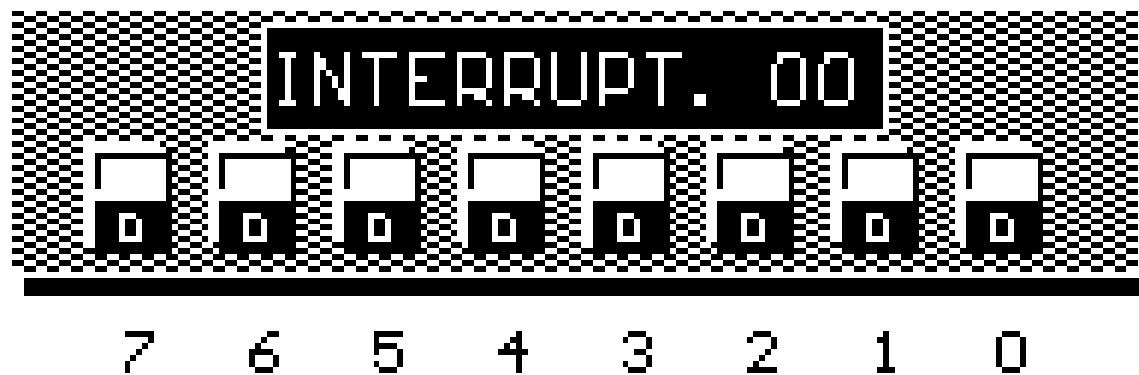
## Refrigeración

Control de ventilación

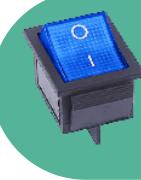
# Periféricos – Microinterruptores



- Es una barra de 8 interruptores (botones con 2 estados)
- Se conecta en el MSX88 a un puerto que permite “sensar” (saber el estado) de los interruptores (8 bits = 8 interruptores)
- Cuando se activa/desactiva un interruptor modifica el bit asociado en el puerto de entrada



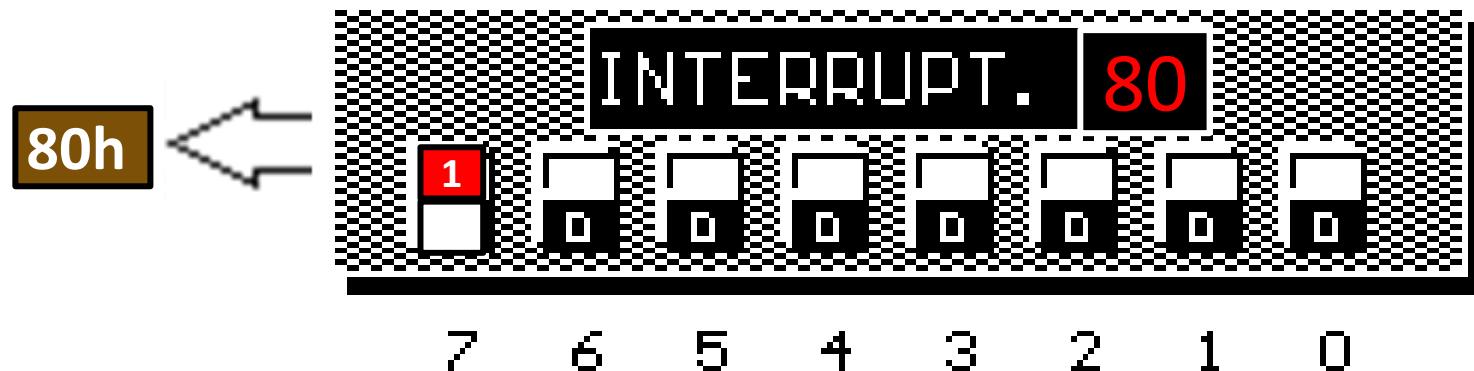
# Periféricos – Microinterruptores



Los interruptores son externos y para controlarlos es necesario ejecutar el comando que simula una presión:

- M{nro de bit} : invierte bit {nro de bit}
- M{valor hexa}: establece todos los valores simultáneamente

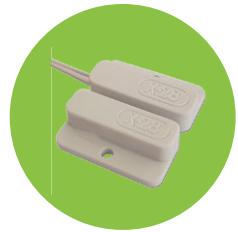
Ejecutamos el comando: M7



# Periféricos – Microinterruptores



- La funcionalidad de los microinterruptores es didáctica
- En aplicaciones reales, con el hardware adecuado, con 1 bit es posible sensar dispositivos que tienen 2 estados (apagado/encendido):



**Detección de Cierre y Apertura**  
Sensores para puertas y ventanas



**Detección de Partículas**  
Sensores para Humos y Gases



**Detección de Nivel Alto/Bajo**  
Sensores para líquidos



**Detección de luminocidad**  
Sensores de luz



**Detección de Presencia**  
Sensores infrarrojos y de radar



**Detección de Interruptores**  
Pulsadores, Interruptores, Pad de Teclas

# Periféricos – Impresora Paralela



## Características:

- Tiene 20 columnas e imprime a una velocidad de 1 carácter cada 5 segundos
- Tiene un cola de impresión (**buffer**) de 5 caracteres:
  - Cuando la cola está llena, la impresora activa una señal (**ocupada**) para indicar que no puede recibir caracteres
  - Cuando esta lista para imprimir un carácter, lo saca del **buffer**, lo imprime y desactiva la señal (desocupada)



# Periféricos – Impresora Paralela



## Comunicación:

- 8 líneas de entrada (**data**) para recibir un carácter ASCII
- 1 línea de salida (**busy**) para indicar si esta ocupada o disponible para recibir un carácter e imprimirlo:
  - Línea en 1 → impresora ocupada
  - Línea en 0 → impresora libre
- 1 línea de entrada (**strobe**) para indicarle cuando hay un carácter para imprimir en **data**:
  - Toma carácter de **data** cuando pasa de 1 a 0



# Periféricos – Impresora Paralela



# Periféricos – Impresora Paralela



Pseudo-código para enviar un carácter:

Mientras la línea **busy** este en 1 (\*1):

Esperar sin hacer nada

Enviar a **data** el carácter a imprimir (\*2)

Enviar un 1 a la linea **strobe** (\*3)

Enviar un 0 a la línea **strobe** (\*3)



\*1 busy



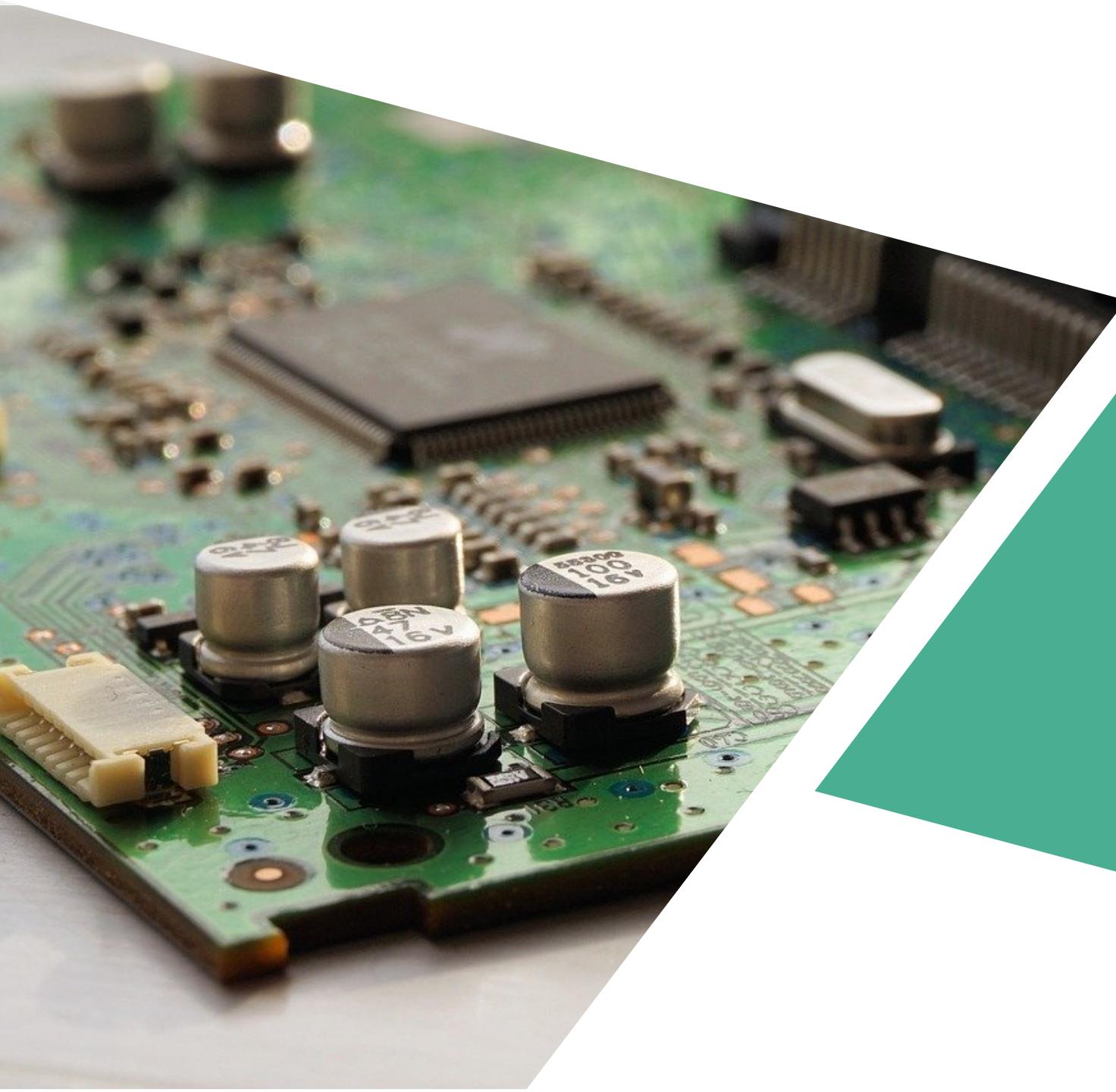
\*3 strobe



\*2 data



Impresora



# Entrada/Salida con PIO

# PIO - Características

## Características:

- Dispone de hasta 16 bits o líneas para comunicarse con periféricos (conector de 16 “cables” de datos al exterior)
  
- El sentido de cada línea es configurable. Puede usarse como entrada o como salida (no son simultáneas)



# PIO - Registros

- Los 16 bits se distribuyen entre dos registros del dispositivo: PA (puerto A) y PB (puerto B)
- Tiene 2 registros adicionales donde se indica el sentido (entrada o salida) de cada uno de los bits:
  - CA (configuración de A) para cada bit de PA
  - CB (configuración de B) para cada bit de PB
  - Un bit en 0 configura como salida, un bit en 1 como entrada
  - La electrónica para que un bit sea de entrada o de salida es diferente. Por esto son necesarios los bits de configuración



# PIO - Registros

Se conecta al MSX88 a partir de la dirección 30H:

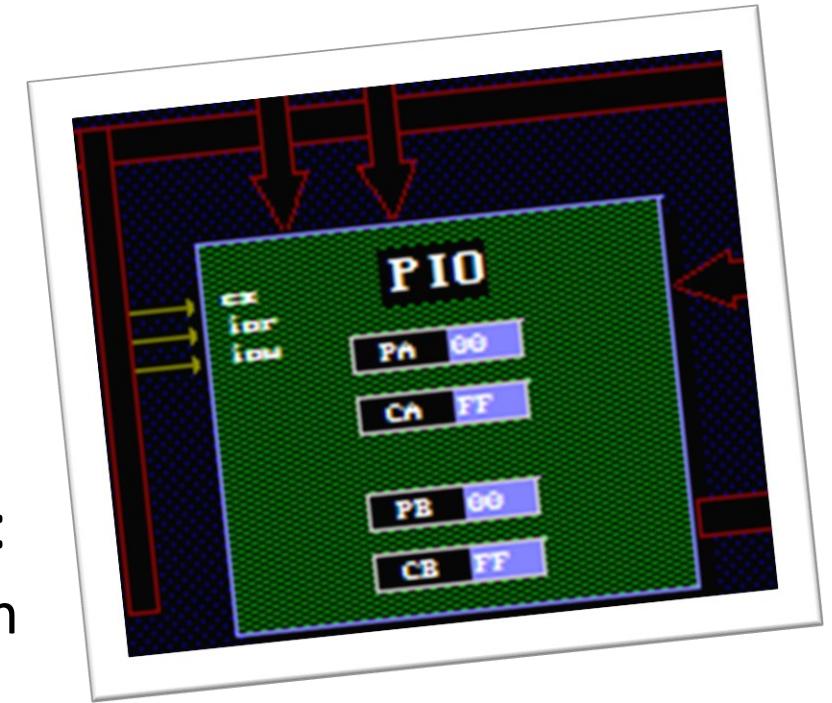
- Puertos de envío y recepción:  
**PA** en la dirección **30H** y **PB** en la dirección **31H**
- Puertos de configuración de dirección de bits  
**CA** en la dirección **32H** y **CB** esta en la dirección **33H**

Puede conectarse la barra de leds e interruptores :

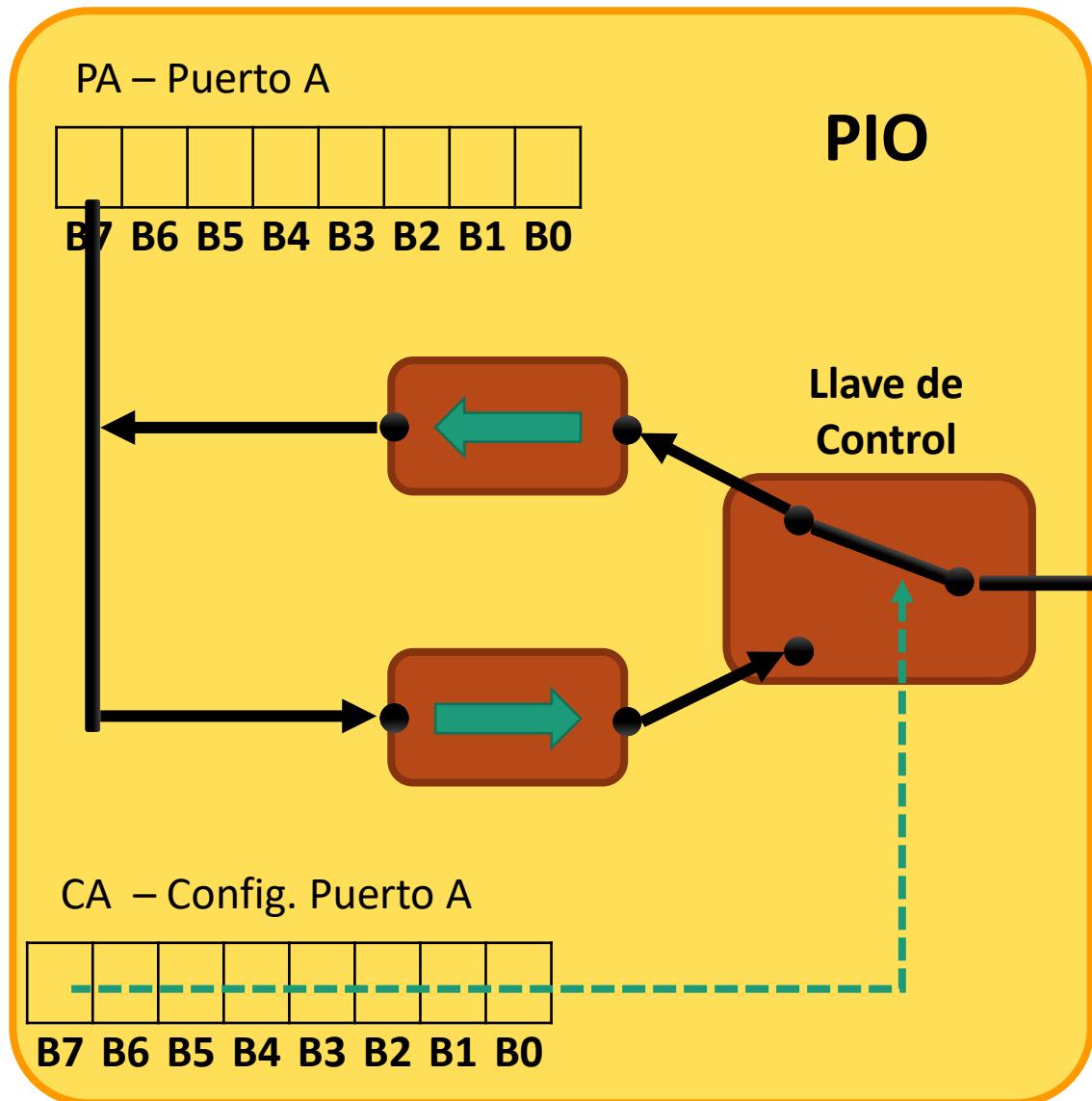
- Comando simulador “P1 C0” muestra la configuración
- En **PA** se conectan los **interruptores** y en **PB** los **leds**

Puede conectarse la impresora:

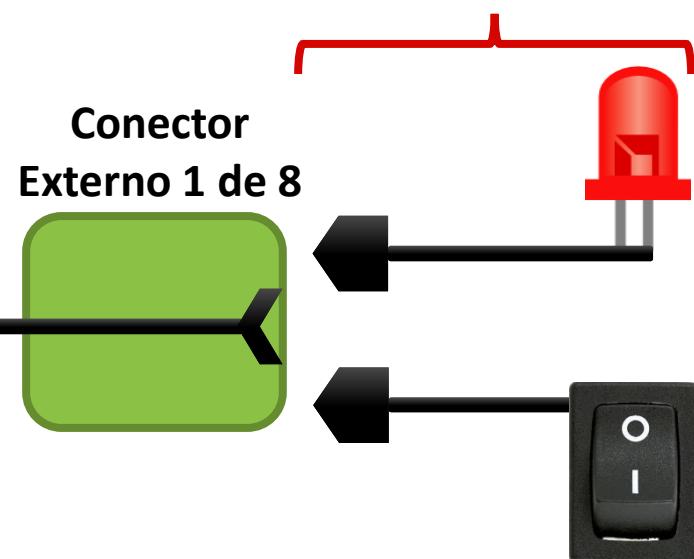
- Comando simulador “P1 C1” muestra la configuración
- En el **bit 0** de PA se conecta la línea **busy**
- En el **bit 1** de PA se conecta la línea **strobe**
- En **PB** se conectan las líneas de **datos**



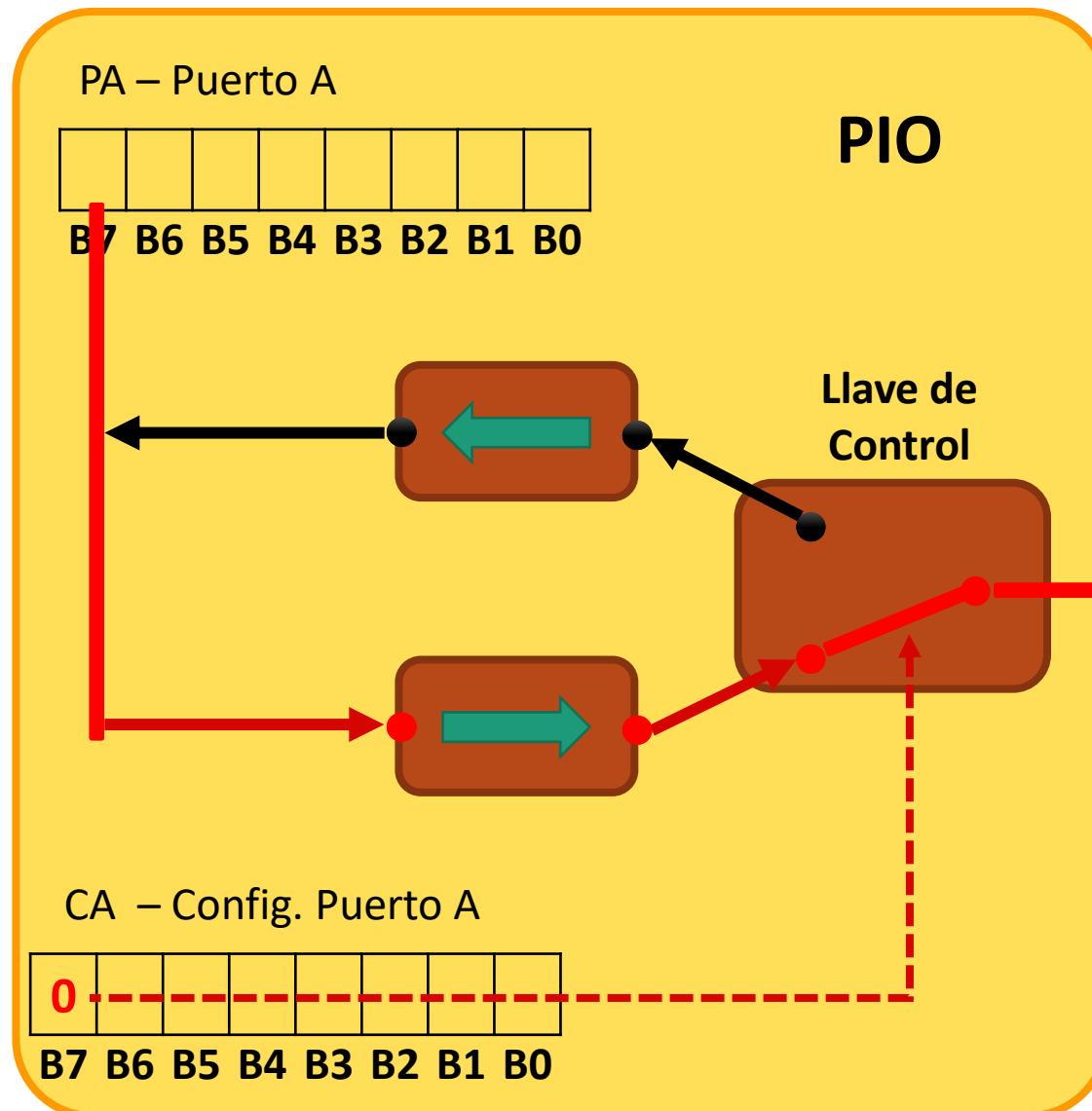
# PIO – Configuración E/S



**Dispositivos que se  
pueden conectar**



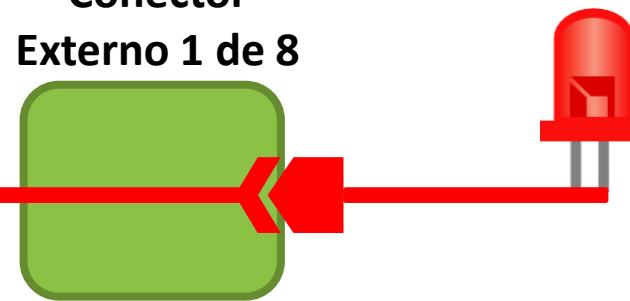
# PIO – Configuración E/S



Luego de configurado CA:

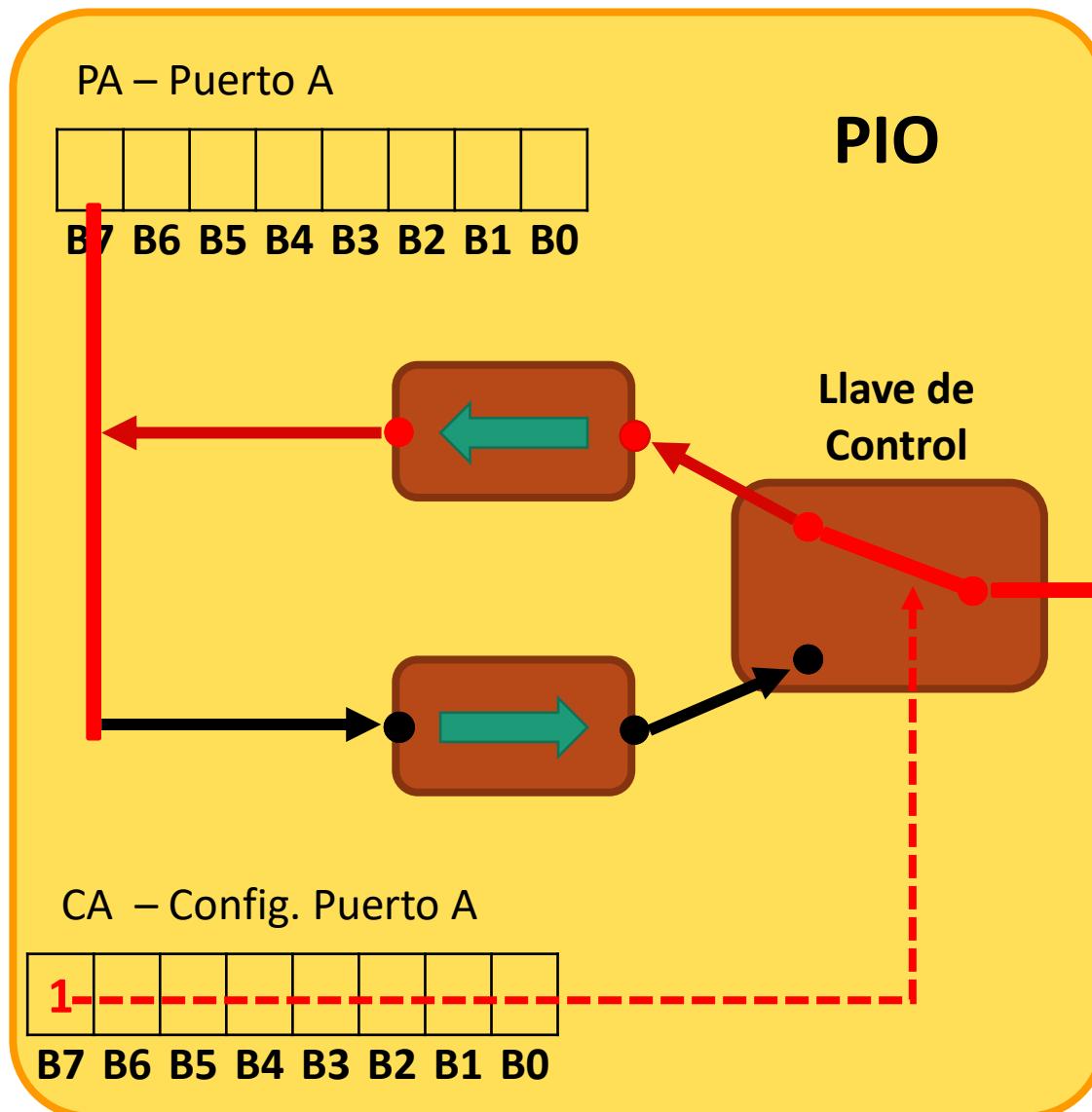
- ✓ Poner 1 en bit 7 de PA enciende el LED
- ✓ Poner 0 en bit 7 de PA apaga el LED

Conector  
Externo 1 de 8



El bit 7 de CA habilita la electrónica necesaria para transmitir al exterior un 1 o 0 a través del bit 7 del PA

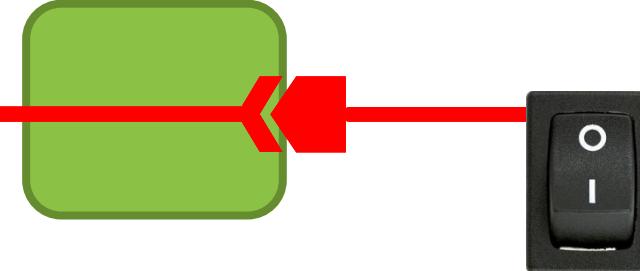
# PIO – Configuración E/S



Luego de configurado CA:

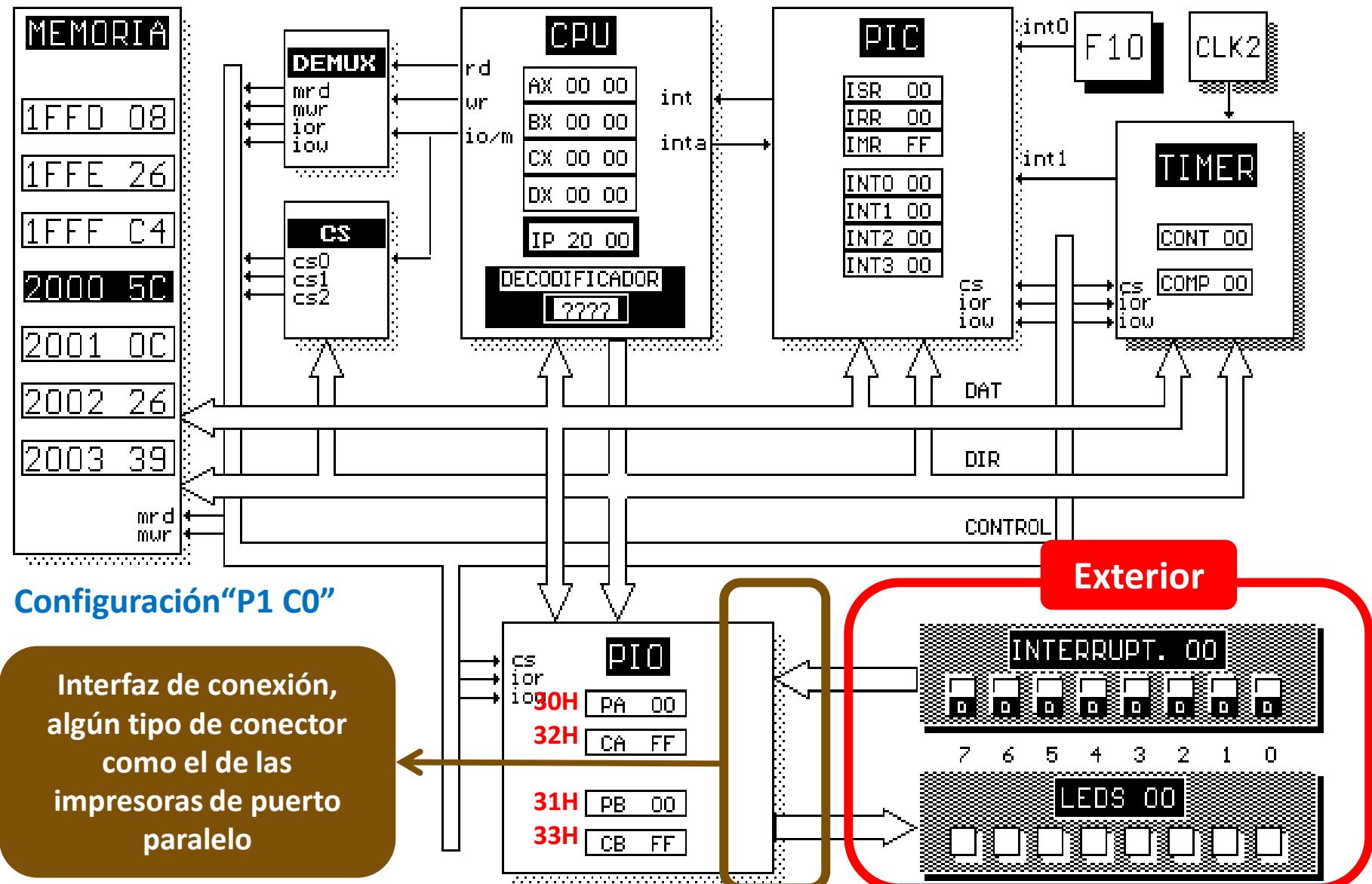
- ✓ Leer un 1 en bit 7 indica interruptor presionado
- ✓ Leer un 0 en bit 7 indica interruptor sin presionar

Conector  
Externo 1 de 8

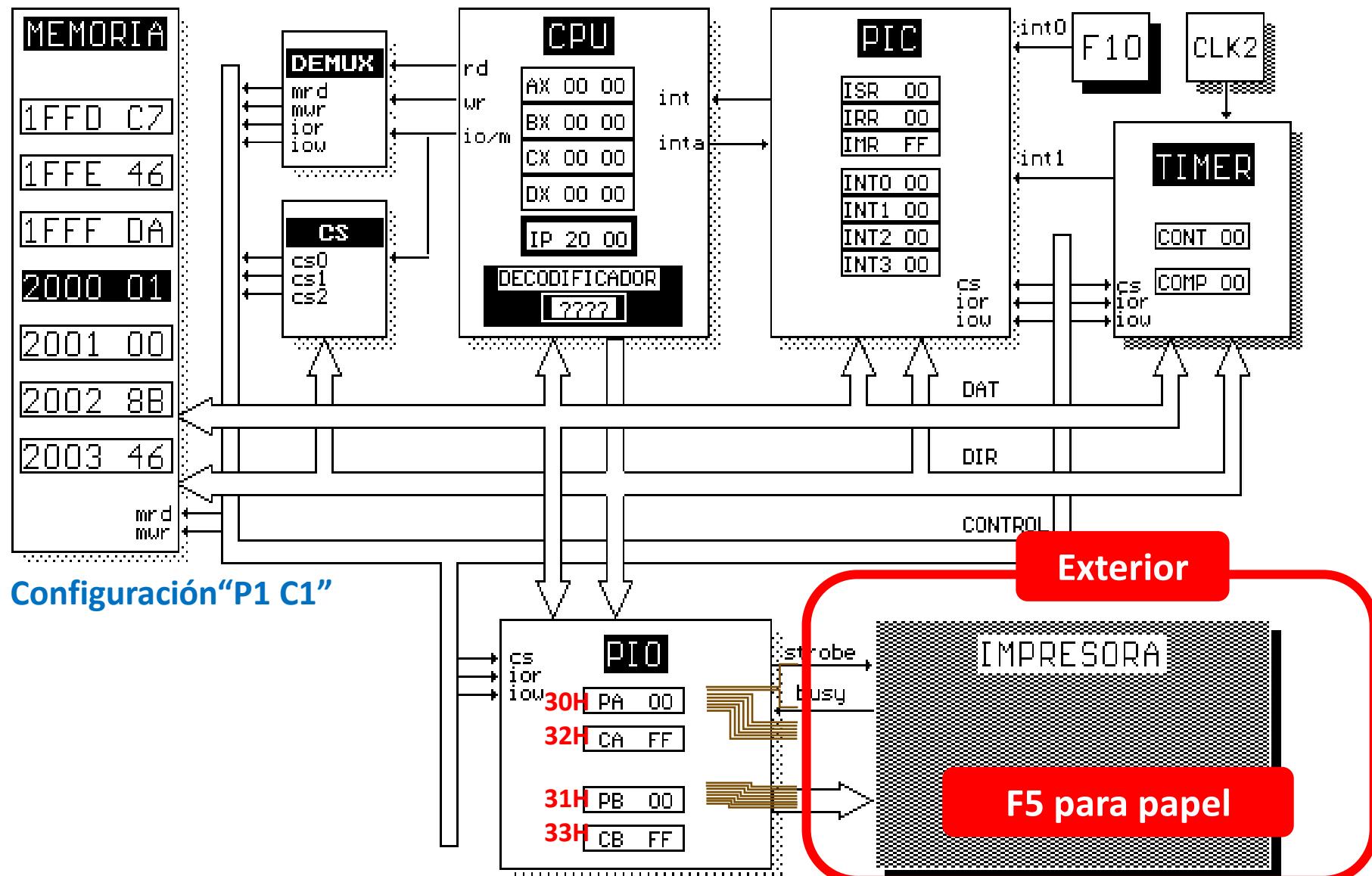


El bit 7 de CA habilita la electrónica necesaria para leer desde exterior un 1 o 0 a través del bit 7 del PA

# PIO – Conexión con Leds e Interruptores



# PIO – Conexión con Impresora





# Ejercicios con PIO

# Ejercicios de Práctica

## Ejercicio 1 – Controlar leds con interruptores

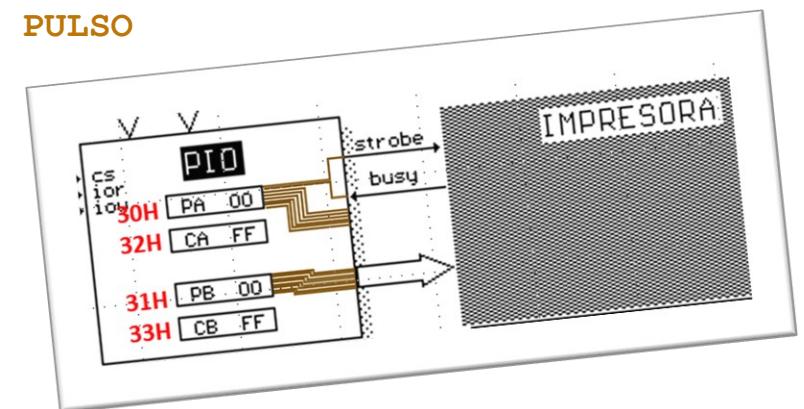
```
1. PA EQU 30H
2. PB EQU 31H
3. CA EQU 32H
4. CB EQU 33H
5.
6. ORG 2000H
7.     MOV AL, OFFH ; PA entradas (Microconmutadores)
8.     OUT CA, AL
9.     MOV AL, 0      ; PB salidas (Luces)
10.    OUT CB, AL
11. POLL: IN AL, PA ; Refleja el valor de interruptores
12.        OUT PB, AL ; en los leds
13.        JMP POLL
14.        END
```

RECORDAR:  
Bit en 1 en puerto de configuración significa que hay una señal de entrada conectada al bit correspondiente del puerto de datos

# Ejercicios de Práctica

## Ejercicio 4 – Imprimir texto con impresora y PIO

```
PIO EQU 30H ;PIO =PA PIO+1=PB  
;PIO+2=CA PIO+3=CB  
1. ORG 1000H  
2. MSJ DB "ARQUITECTURA DE "  
3. DB "COMPUTADORAS"  
4. FIN DB ?  
5.  
6. ORG 2000H  
7. MOV AL, 0FDH ; INIC. PIO IMPRESORA  
8. OUT PIO+2, AL ; CONF. STROBE Y BUSY  
9. MOV AL, 0  
10. OUT PIO+3, AL ; CONF. DATOS  
11. IN AL, PIO ; LEE STROBE Y BUSY  
12. AND AL, 0FDH ; 253 o 11111101b  
13. OUT PIO, AL ; FIN INICIALIZACION  
  
14. MOV BX, OFFSET MSJ  
15. MOV CL, OFFSET FIN-OFFSET MSJ  
16. POLL: IN AL, PIO ; LEE STROBE Y BUSY  
17. AND AL, 1 ; SOLO DEJA BUSY  
18. JNZ POLL  
19. MOV AL, [BX] ; RECUPERA CARACTER  
20. OUT PIO+1, AL ; ENVIA A DATA(PB)  
IN AL, PIO ; PULSO 'STROBE'  
OR AL, 02H ; PONE 1 en bit 2  
OUT PIO, AL  
IN AL, PIO  
AND AL, 0FDH ; PONE 0 en bit 2  
OUT PIO, AL ; FIN PULSO  
INC BX  
DEC CL  
JNZ POLL  
INT 0  
END
```



# Ejercicios de Práctica

## Ejercicio 4 – Imprimir texto con impresora y PIO

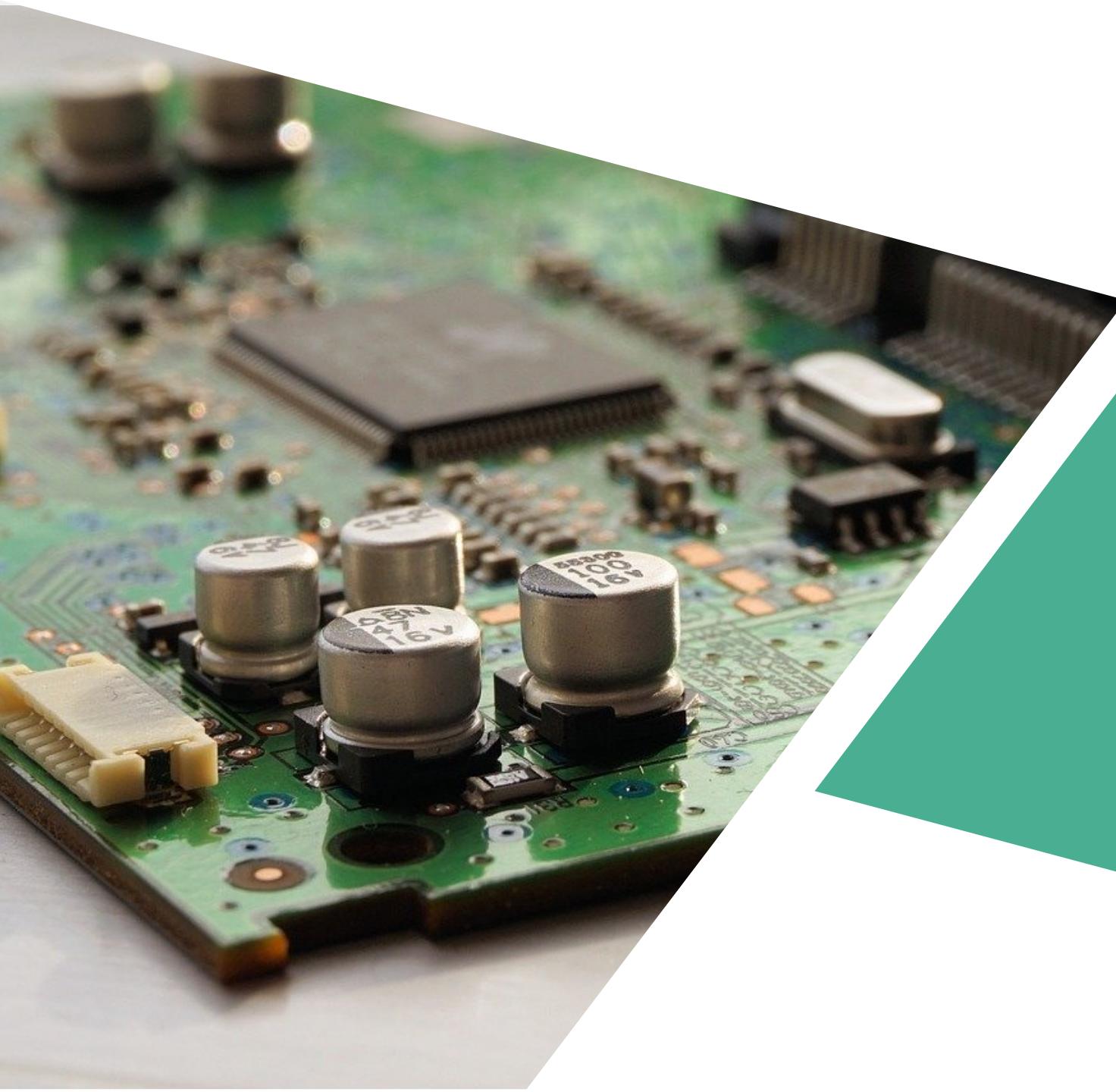
```
14.     MOV BX, OFFSET MSJ
15.     MOV CL, OFFSET FIN-OFFSET MSJ
16. POLL: IN AL, PIO ; LEE STROBE Y BUSY
17.     AND AL, 1 ; SOLO DEJA BUSY
18.     JNZ POLL
19.     MOV AL, [BX] ; RECUPERA CARACTER
20.     OUT PIO+1,AL ; ENVIA A DATA (PB)
21.     IN AL, PIO ; PULSO 'STROBE'
22.     OR AL, 02H ; PONE 1 en bit 2
23.     OUT PIO,AL
24.     IN AL,PIO
25.     AND AL, 0FDH ; PONE 0 en bit 2
26.     OUT PIO, AL ; FIN PULSO
27.     INC BX
28.     DEC CL
29.     JNZ POLL
30.     INT 0
31.     END
```

Notar que durante la ejecución de las líneas 16 a 18 la CPU hace "polling"

Se denomina "polling" a una secuencia de instrucciones que continuamente se ejecutan en espera a que cambie la condición en un dispositivo

¿Cuál es el problema y cómo lo evitamos?

El problema es el tiempo perdido en la ejecución de instrucciones que verifican un cambio de condición que sucede cada tanto. Esto puede evitarse si el dispositivo genera una interrupción cuando la condición que esperamos cambia

A close-up photograph of a green printed circuit board (PCB). The board is populated with several electronic components, including four cylindrical surface-mount capacitors in the foreground, a large grey microchip in the upper left, and a black integrated circuit package in the center. The PCB has a dense grid of gold-colored vias and traces.

# Entrada/Salida con Handshake

# Handshake - Características

Dispositivo especializado para comunicarse con impresoras

Ventajas:

- Genera el pulso en la línea de **strobe** de forma automática
- Puede generar interrupciones:
  - Puede generar una interrupción cuando la impresora puede recibir un carácter. Con este mecanismo podemos escribir un carácter sin necesidad de **esperar** o perder tiempo consultando que este lista (evita **polling**)

➤ Desventajas:

- Solo permite la comunicación con la impresora, mientras que la PIO puede comunicarse con cualquier dispositivo



# Handshake - Conexión

- Se conecta al MSX88 a partir de la dirección 40H
- Tiene solo 2 registros
- Conectado a PIC a través de la línea Int2



# Periféricos – Registros

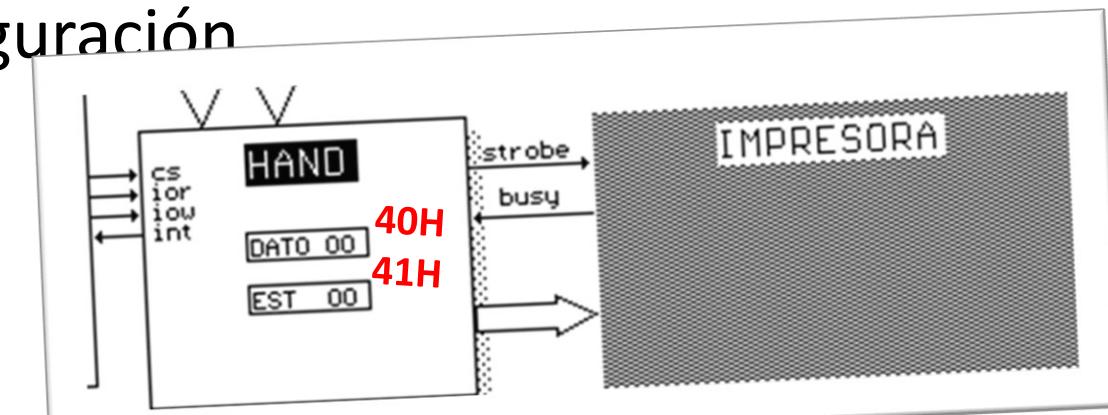
## Registro de datos (data):

- Ubicado en la dirección 40H:
  - 8 bits para almacenar un código ASCII
- Cada vez que se escribe un valor, el handshake espera que la impresora este libre (línea busy=0) y luego genera el pulso strobe
- Si estamos usando interrupciones es seguro escribir el carácter para que lo envíe a la impresora
- Si NO estamos usando interrupciones hay que verificar que el bit busy este en 0. La escritura de un nuevo carácter hace que se pierda el anterior

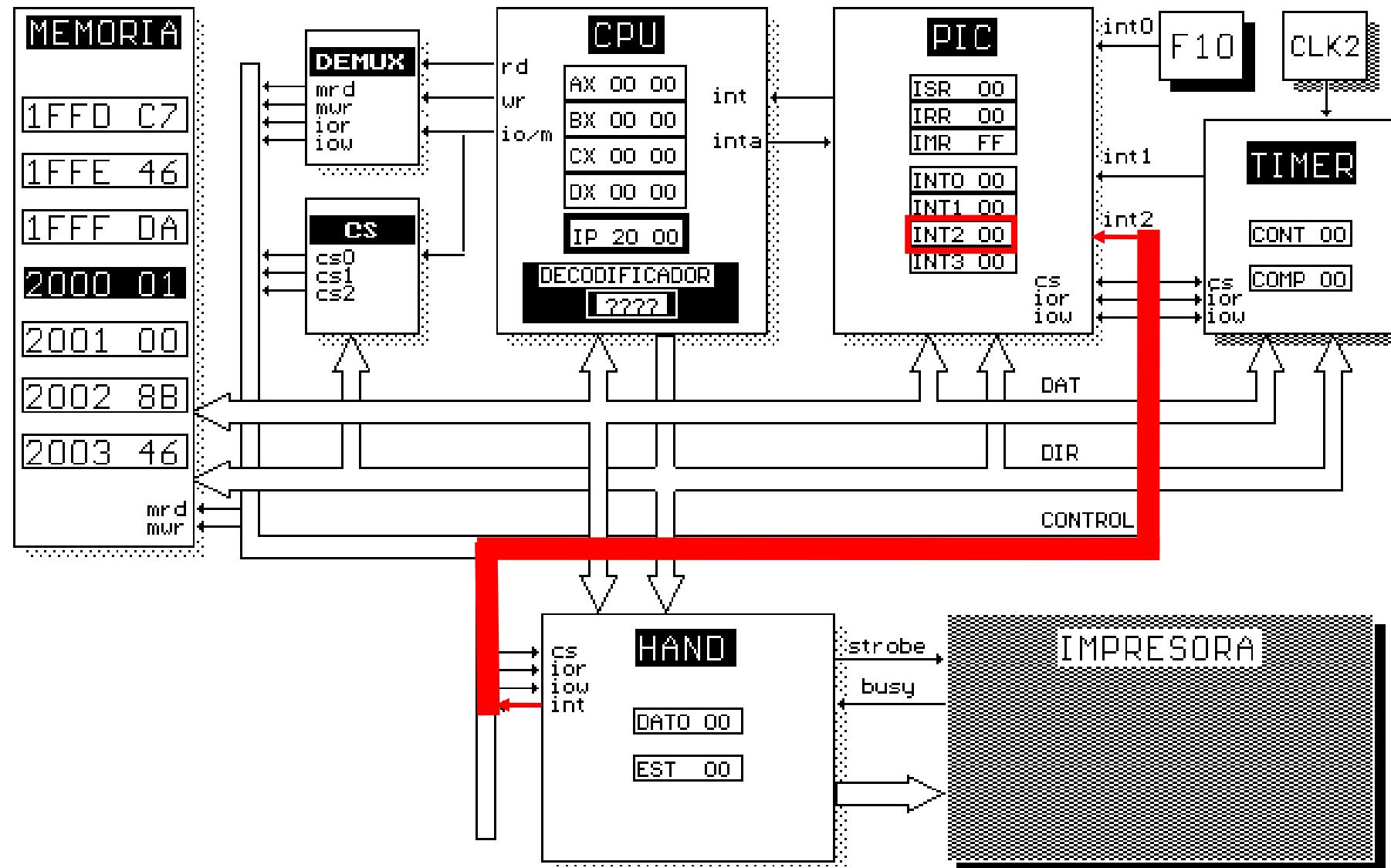


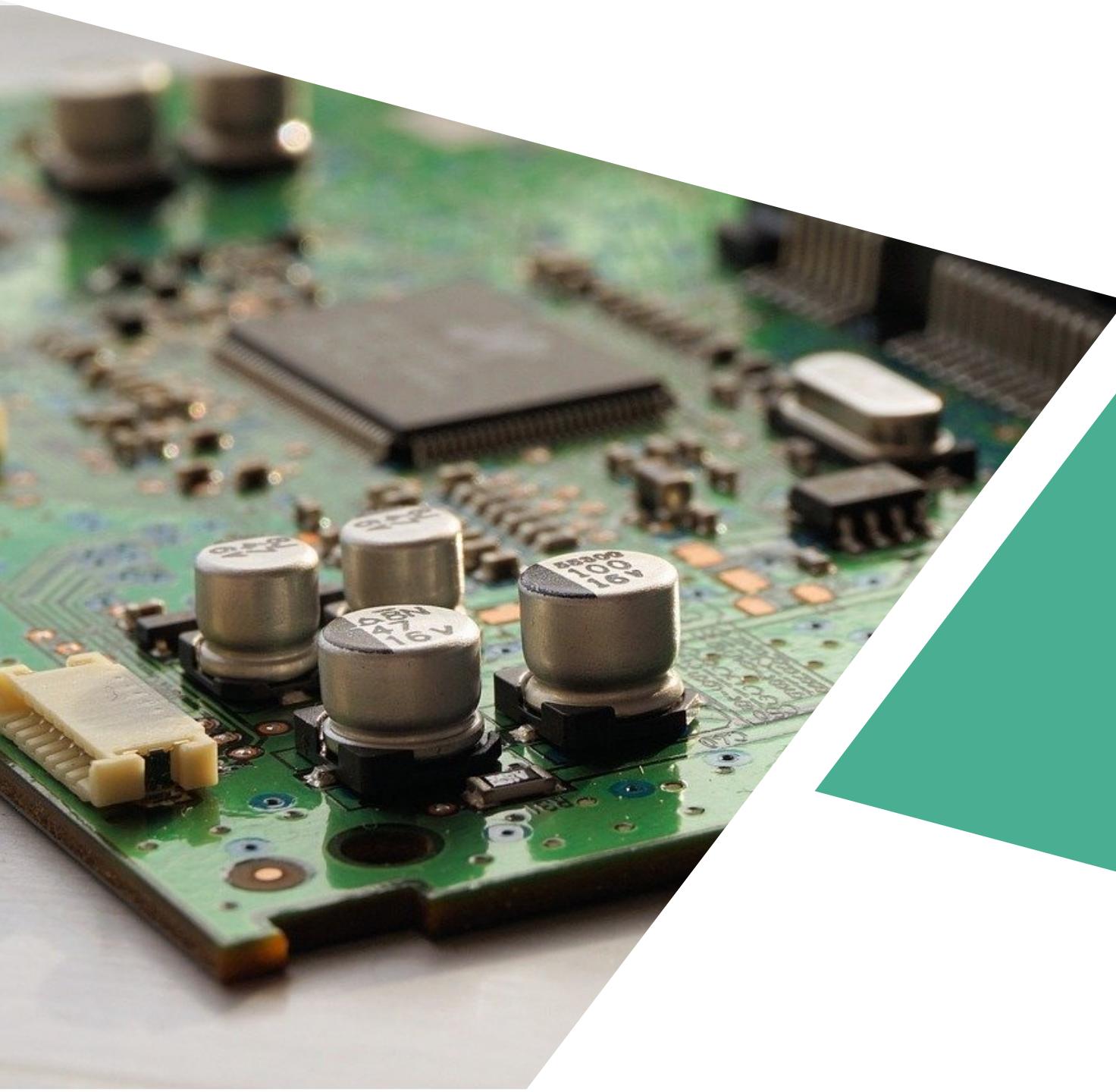
# Periféricos – Registros

- Registro de estado y control(status):
  - Ubicado en la dirección 41H
  - Funciones de los bits:
    - Bit 0: conectado a línea **busy** de la impresora (solo útil para polling)
    - Bit 1: conectado a línea **strobe** de la impresora
    - Bit 2-6: sin uso
    - Bit 7: Control de interrupción:
      - Valor en 1: habilita interrupciones
      - Valor en 0: deshabilita interrupciones
- El comando “P1 C2” muestra esta configuración del Handshake con la impresora



# Handshake – Conexión con Impresora



A close-up photograph of a green printed circuit board (PCB). The board is populated with several electronic components, including four cylindrical surface-mount capacitors in the foreground, a large grey microchip in the upper left, and other smaller components like resistors and connectors. The board has a dense grid of gold-colored vias and traces.

# Ejercicios con Handshake

# Handshake – Ejemplo con Impresora

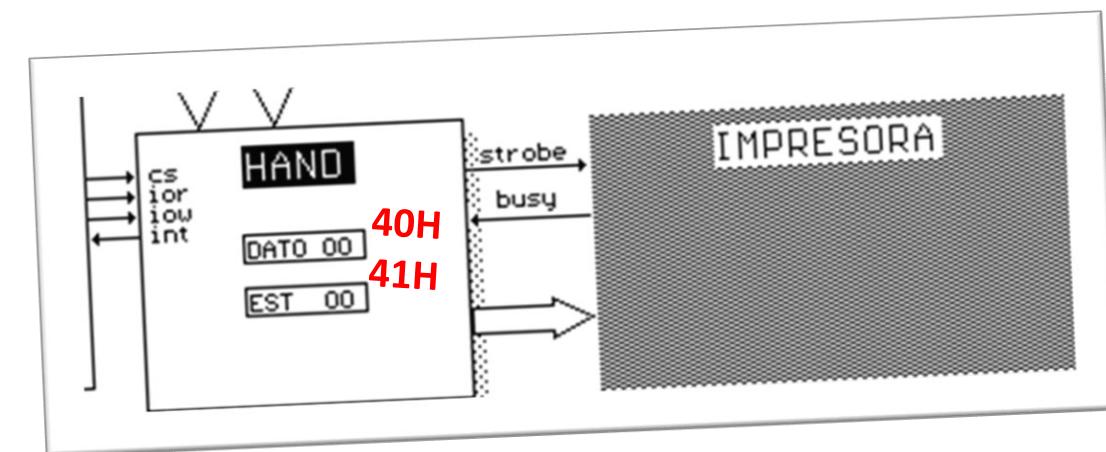
## Ejercicio 7 - Imprimir con Handshake sin interrupciones

```
HAND EQU 40H
```

```
ORG 1000H
MSJ DB "FACULTAD DE "
      DB "INFORMATICA"
FIN DB ?
```

```
ORG 2000H
1.   IN AL, HAND+1          ; LEE STATUS DE HAND
2.   AND AL, 7FH             ; 7FH = 127 = 01111111B PONE A 0 BIT 7
3.   OUT HAND+1, AL         ; ESCRIBE STATUS DE HAND
4.   MOV BX, OFFSET MSJ     ; PUNTERO A TEXTO
5.   MOV CL, OFFSET FIN-OFFSET MSJ
6. POLL: IN AL, HAND+1      ; LEE STATUS DE HAND
7.   AND AL, 1                ; DEJA VALOR DE BIT 0
8.   JNZ POLL
9.   MOV AL, [BX]              ; RECUPERA CARACTER
10.  OUT HAND, AL            ; ESCRIBE STATUS DE HAND
11.  INC BX
12.  DEC CL                  ; FALTA 1 MENOS
13.  JNZ POLL
14.  INT 0
15. END
```

| Status | 7   | 6 | 5 | 4 | 3 | 2 | 1   | 0    |
|--------|-----|---|---|---|---|---|-----|------|
| 41H    | int | X | X | X | X | X | str | busy |



# Handshake – Ejemplo con Impresora

## Ejercicio 8 - Imprimir con Handshake por interrupciones

```
PIC EQU 20H  
HAND EQU 40H  
N_HND EQU 10
```

| Status | 7   | 6 | 5 | 4 | 3 | 2 | 1   | 0    |
|--------|-----|---|---|---|---|---|-----|------|
| 41H    | int | X | X | X | X | X | str | busy |

```
ORG 40 ORG  
IP_HND DW RUT_HND
```

```
ORG 1000H  
MSJ DB "UNIVERSIDAD "  
DB "NACIONAL DE LA PLATA"  
FIN DB ?
```

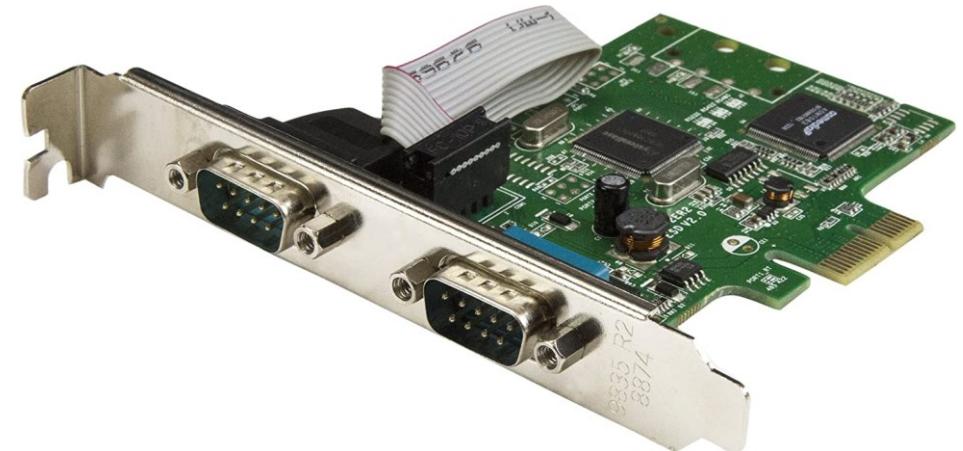
```
ORG 3000H ; RUTINA DE INTERRUPCION  
1. RUT_HND: PUSH AX  
2. MOV AL, [BX]  
3. OUT HAND, AL ; CARÁCTER A DATOS  
4. INC BX ; PROXIMO  
5. DEC CL ; DESCUENTA  
6. MOV AL, 20H ; AVISA A PIC  
7. OUT PIC, AL  
8. POP AX  
9. IRET
```

```
ORG 2000H  
1. MOV BX, OFFSET MSJ  
2. MOV CL, OFFSET FIN-OFFSET MSJ  
3. CLI ; BLOQUEA INTERRUPCIONES  
4. MOV AL, OFBH; FBH = 251 = 11111011B  
5. OUT PIC+1, AL ; ESCRIBE IMR  
6. MOV AL, N_HND ; POS. 10 DE VECT INT  
7. OUT PIC+6, AL ; REGISTRO INT2 DE PIC  
8. MOV AL, 80H ; 80H = 128 = 10000000B  
9. OUT HAND+1, AL; REG ESTADO (CON INT)  
10. STI ; HABILITA INTERRUPCIONES  
11. LAZO: CMP CL, 0 ; CUANDO TERMINA?  
12. JNZ LAZO  
13. IN AL, HAND+1 ; LEE ESTADO HAND  
14. AND AL, 7FH ; 7FH = 127= 01111111B  
15. OUT HAND+1, AL ; REG ESTADO(SIN INT)  
16. INT 0  
END
```

# USART – Características



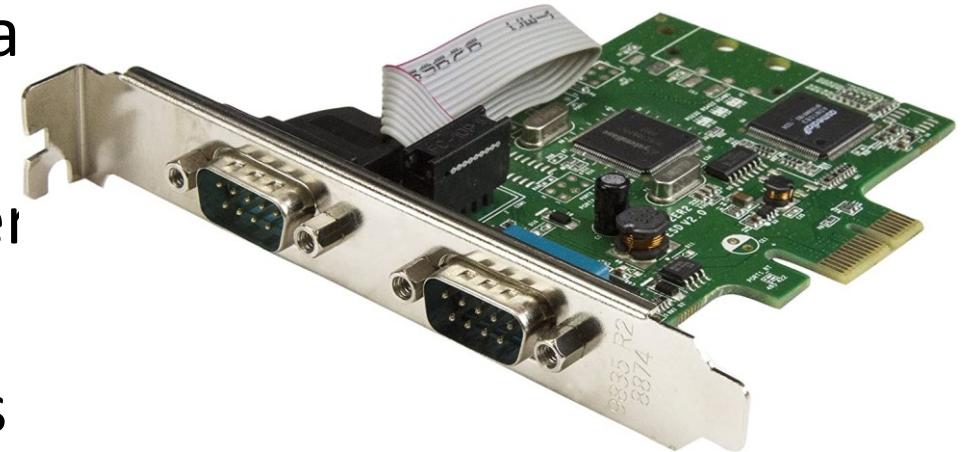
- Universal Synchronous/Asynchronous Receiver/Transmitter o Receptor/Transmisor Síncrono/Asíncrono Universal
- Transmite información de forma serie, es decir que el dato a transmitir va por una única línea bit a bit.
- Modos de Transmisión:
  - Asíncrona (Solo vemos esta en la práctica)
  - Síncrona-Maestro
  - Síncrona-Esclavo
- Protocolos de Comunicación
  - XON/XOFF (por software)
  - DTR (por hardware)
- Capacidad para funcionar con interrupciones para transmitir y recibir



# USART – Comunicación Asincrónica



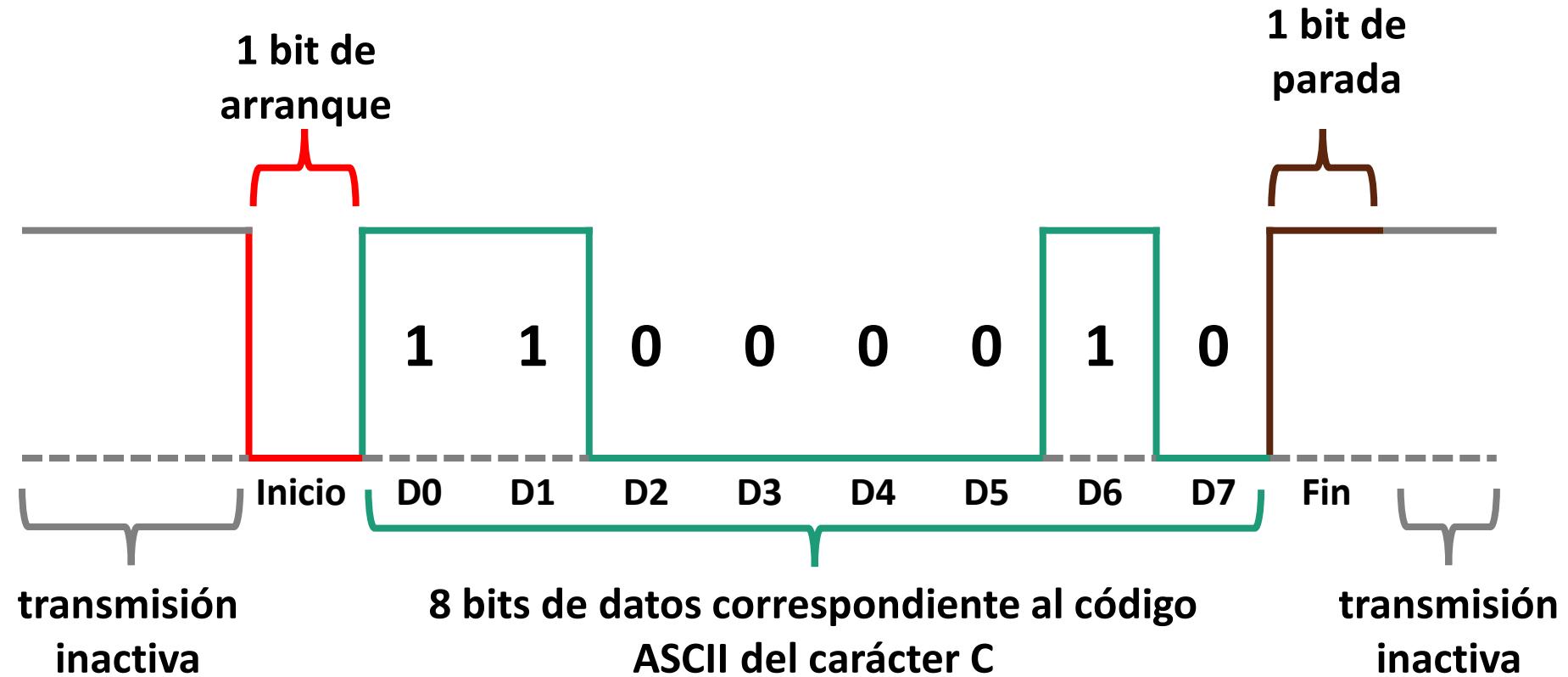
- No hay señal de reloj para sincronizar. La sincronización se realiza a través del bit de arranque o inicio y de parada o fin (start/stop)
- El receptor y transmisor deben estar de acuerdo en la velocidad de transmisión.
- Cuando el receptor detecta el bit de arranque, lee los demás bits utilizando su propio reloj asumiendo que la velocidad es la misma
- Transmite 8 bits de datos por vez. El primer bit es el más bajo.
- Por cada 8 bits se envían 2 bits adicionales de inicio y uno al final



# USART – Comunicación Asincrónica



Carácter “C” → 43H → 67 → 01000011<sub>2</sub>



# USART – Control de Flujo



- Control de Flujo se utiliza cuando alguno de los dispositivos conectados con la USART no puede recibir los datos: indica cuando la transmisión debe detenerse o reanudarse
- Teniendo en cuenta que la CPU es mucho mas rápida que la impresora, es necesario un mecanismo para “pausar” el envío de los datos
- El control de Flujo hacerse de 2 maneras:
  - Por software: enviando un código especial para detener o reiniciar la transmisión
  - Por hardware: utilizando una señal adicional entre el transmisor y el receptor (similar a la señal BUSY en la transmisión de la impresora paralela)

# USART – Control de Flujo XON/XOFF



- Es un control de flujo por software
- Se utilizan caracteres especiales que envía el receptor para controlar el flujo de datos que envía el transmisor
  - XON habilita la transmisión (código ASCII 17 o 11H)
  - XOFF deshabilita la transmisión (código ASCII 19 o 13H)
- Usa una única señal tanto para transmitir datos como para hacer el control (ventaja)
- Cuando el transmisor está enviando datos y el receptor necesita detenerlo debe enviar el carácter de control con suficiente tiempo (recordar que el transmisor debe recibir todos los bits y luego interpretarlo), sino se pueden perder datos (desventaja)

# USART – Control de Flujo XON/XOFF



- Esquema de comunicación con impresora serie por control de flujo por software XON/XOFF



- Desde la USART se envía por la señal TXD los caracteres a imprimir y por la señal RXD se recibe el carácter XOFF para pausar el envío (cuando el buffer de la impresora se llena) y XON para reanudarlo

**Notar que las señales están invertidas. Lo que de un lado se ve como TX del otro se ve como RX y viceversa**

# USART - Control de Flujo DTR/DSR



- Es un control de flujo por hardware
- Receptor y Transmisor comparten una línea de control que alterna entre 1 y 0 para indicar cuando puede recibir o no un dato
- Cuando el transmisor está enviando datos y el receptor necesita detenerlo desactiva la señal. Como el transmisor verifica esta señal antes de enviar un dato no hay riesgo de perdida de datos (ventaja)

# USART – Control de Flujo XON/XOFF



- Esquema de comunicación con impresora serie por control de flujo por hardware DTR/DSR



- Desde la USART se envían por la señal TXD los caracteres a imprimir y por la señal DTR/DSR se recibe el estado de la impresora:
  - 1 cuando puede recibir datos
  - 0 cuando no puede recibir datos.
- No se reciben datos de la impresora a través de la señal RXD

# USART MSX88 - Características



- Dispositivo especializado para comunicarse de forma serie con cualquier dispositivo (solo con impresora en MSX88)
- Ventajas:
  - Pocas líneas para realizar la comunicación
  - Puede generar interrupciones:
    - Luego de recibir un carácter de la impresora
    - Luego de enviar un carácter a la impresora
    - Este mecanismo permite evitar la pérdida de tiempo consultando para enviar o recibir un dato (evita polling)
- Desventajas:
  - Al ser un dispositivo serie, la velocidad de transmisión es menor que la de dispositivos paralelos como HANDSHAKE y PIO

# USART MSX88 – Conexión



- Esta conectado a partir de la dirección 60H
- Tiene 3 registros
- Conectado a PIC a través de la línea Int2 para enviar datos e int 3 para recibir datos
- El comando del simulador “P1 C4” muestra esta configuración USART+ impresora serie
- El comando “VI” configura la velocidad de comunicación de la impresora. “VI 6” establece una velocidad de 6 baudios (valor por defecto)
- El comando “PI” configura el protocolo de control de flujo de la impresora. “PI X” configura XON/XOFF y “PI D” configura DTR/DSR

# USART MSX88 – Registros



## Registro de datos de entrada DIN:

- Ubicado en la dirección 60H:
- 8 bits para almacenar byte (para impresora un código ASCII)
- Transforma el dato serie recibido por la señal de entrada RXD en un dato paralelo para que pueda ser leído por la CPU

## Registro de datos de salida DOUT:

- Ubicado en la dirección 61H:
- 8 bits para almacenar un byte (para impresora un código ASCII)
- Transforma el dato paralelo recibido por la CPU en un dato serie que se transmite por la señal de salida TXD a la impresora

# USART MSX88 – Registros



Registro de datos control/estado CTRL :

- Ubicado en la dirección 62H:
- Los bits cambian de función dependiendo si se leen o se escriben
- 8 bits para función de control (escritura)
- 7 bits para función de estado (lectura)

# USART – Registro de Control (escritura)



| 7     | 6  | 5   | 4   | 3    | 2    | 1     | 0     |
|-------|----|-----|-----|------|------|-------|-------|
| Synch | ER | RTS | DTR | RxEn | TxEn | VBAUD | Sy/As |

- **Sy/As** (Synchronous/Asynchronous, modo de transferencia de datos):  
Bit=0 → sincrónico                      Bit=1 → asincrónico
  - **Vbaud** (Baudios, velocidad de transferencia de datos):  
Bit=0 → 6 baudios (6 bits/segundo)  
Bit=1 → 18 baudios (18 bits/segundo)
  - **TxEn** (Transmit Enabled, habilita interrupción para transmitir un dato):  
Bit=0 → Deshabilitada                      Bit=1 → Habilitada
  - RxEn** (Receive Enabled, habilita interrupción para recibir un dato):  
Bit=0 → Deshabilitada                      Bit=1 → Habilitada
  - **DTR** (Data Terminal Ready, indica que el MSX88 está listo para iniciar la comunicación):  
Bit=0 → No iniciar comunicación        Bit=1 → Iniciar comunicación
  - **RTS** (Request To Send, activa la señal RTS del transmisor para avisar al receptor que quiere transmitir):  
Bit=0 → RTS Desactivada                      Bit=1 → RTS Activada
  - **ER** (Error Reset, Pone en 0 los flags de error):  
Bit=0 → no hace nada                      Bit=1 → Limpia los flags de error
  - **Synch** (Inserta y busca el carácter de sincronización 16H, válido para modo sincrónico):  
Bit=0 → Desactivado                      Bit=1 → Activado

# USART – Registro de Control (lectura)



| 7   | 6     | 5   | 4 | 3  | 2  | 1     | 0     |
|-----|-------|-----|---|----|----|-------|-------|
| DSR | SynDt | CTS | - | FE | OE | RxRdy | TxRdy |

- **DSR (Data Set Ready, estado de señal de entrada DSR conectada a DTR de impresora):**  
Bit=0 → impresora no esta lista      Bit=1 → impresora lista
- **SynDet (Sync Detect, recepción del carácter SYNC, válida para modo sincrónico):**  
Bit=0 → no hay detección de SYNC      Bit=1 → hay detección de SYNC
- **CTS (Clear To Send, indica el estado de la señal de entrada CTS):**  
Bit=0 → No puede recibir datos      Bit=1 → Puede recibir datos
- **FE- (Frame Error, válido en modo asincrónico, indica que el dato recibido no tiene la cantidad de bits correcta):**  
Bit=0 → No hay error      Bit=1 → Hay error
- **OE (Overrun Error, indica que se recibió un dato nuevo sin haber leído el dato anterior):**  
Bit=0 → No hay error      Bit=1 → Hay error
- **RxRdy (Receiver Ready, indica recepción de un dato por la señal RxD y esta en registro DIN):**  
Bit=0 → No hay dato      Bit=1 → hay un dato nuevo  
**Se pone automáticamente en 0 cuando se lee el registro DIN y en 1 cuando se recibe un dato por RxD**
- **TxRdy (Transmitter Ready, Indica que el dato de DOUT se envió por la señal TxD y que puede escribir uno nuevo):**  
Bit=0 → no puede escribir DOUT      Bit=1 → puede escribir DOUT  
**Se pone automáticamente en 0 cuando se escribe DOUT y en 1 cuando el dato en DOUT se transmitió**

# USART MSX88 – impresora con DTR/DSR



Programa que envía datos a la impresora a través de la USART usando el protocolo DTR. La comunicación es por consulta de estado. Ejecutar en configuración P1 C4 y utilizar el comando PI que corresponda.

Tener en cuenta en la configuración de transmisión:

- modo asincrónico
- consulta de estado (polling) ➔ no usar interrupción
- misma velocidad de transmisión que la impresora
- reiniciar los errores al iniciar la comunicación
- poner el bit DTR de la USART en 1
- ejecutar comando PI D para modo DTR de impresora

# USART MSX88 – impresora con DTR/DSR

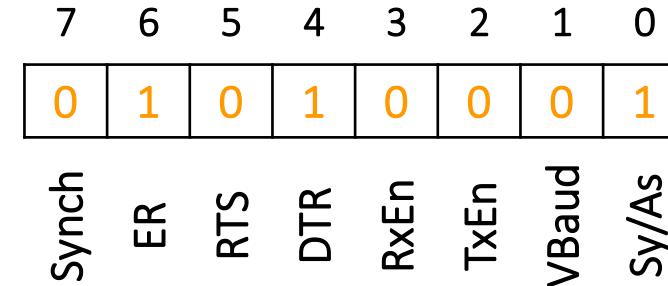


```

1.  USART EQU 60H
2.  ORG 1000H
3.  SACADOS DW 0
4.  TABLA DB "Comunicacion serie a..."
5.  FIN DB ? ; programa principal
6.  ORG 2000H
7.  INICIO: MOV BX, OFFSET
8.  MOV SACADOS, 0
9.  ; programo la USART
10. MOV AL, 51H ; binario=01010001
11. OUT USART+2, AL
12. TEST: IN AL, USART+2
13. AND AL, 81H ; binario=10000001
14. CMP AL, 81H
15. JNZ TEST
16. MOV AL, [BX]
17. OUT USART+1, AL
18. INC BX
19. INC SACADOS
20. CMP SACADOS, OFFSET FIN-OFFSET TABLA
21. JNZ TEST
22. INT 0
23. END

```

Registro CTRL en escritura



Configura USART en modo asincrónico, sin interrupciones a 6 baudios

|       |  |
|-------|--|
| Sy/As | 1 → Modo Asincrónico                             |
| VBAud | 0 → 6 baudios, según impresora                   |
| TxEn  | 0 → sin interrupción para transmitir             |
| RxEn  | 0 → sin interrupción para recibir                |
| DTR   | 1 → Lista para transmitir                        |
| RTS   | 0 → Listo para recibir                           |
| ER    | 1 → reiniciar flags de error                     |
| Synch | X → no importa, solo válido para modo sincrónico |

# USART MSX88 – impresora con DTR/DSR



```

1. 1. USART EQU 60H
2. 2. ORG 1000H
3. 3. SACADOS DW 0
4. 4. TABLA DB "Comunicacion serie a..."
5. 5. FIN DB ? ; programa principal
6. 6. ORG 2000H
7. 7. INICIO: MOV BX, OFFSET
8. 8. MOV SACADOS, 0
9. 9. ; programo la USART
10. 10. MOV AL, 51H ; binario=01010001
11. 11. OUT USART+2, AL
12. 12. TEST: IN AL, USART+2
13. 13. AND AL, 81H ; binario=10000001
14. 14. CMP AL, 81H
15. 15. JNZ TEST
16. 16. MOV AL, [BX]
17. 17. OUT USART+1, AL
18. 18. INC BX
19. 19. INC SACADOS
20. 20. CMP SACADOS, OFFSET FIN-OFFSET TABLA
21. 21. JNZ TEST
22. 22. INT 0
23. 23. END

```

Registro CTRL en lectura

|  | 7   | 6      | 5   | 4 | 3  | 2  | 1     | 0     |
|--|-----|--------|-----|---|----|----|-------|-------|
|  | DSR | SynDet | CTS | . | FE | OE | RxRdy | TxRdy |
|  | 1   | 0      | 0   | 0 | 0  | 0  | 0     | 1     |

Configura USART en modo asincrónico, sin interrupciones a 6 baudios  
 Comprueba TxRdy y DSR para verificar que la impresora pueda recibir datos y este disponible DOUT para escribir un carácter

|        |  |
|--------|--|
| TxRdy  | 1 → Puede escribir DOUT                          |
| RxRdy  | 1 → Puede leer DIN (dato recibido)               |
| OE     | 1 → Recibió dato en DIN sin leer ant.            |
| FE     | 1 → Error en el frame recibido                   |
| -      | 1 → sin función, no importa                      |
| CTS    | 0 → Listo para recibir                           |
| SynDet | x → no importa, solo válido para modo sincrónico |
| DSR    | 1 → DTR de impresora, puede recibir              |

# USART MSX88 – impresora con DTR/DSR



```
1. USART EQU 60H
2. ORG 1000H
3. SACADOS DW 0
4. TABLA DB "Comunicacion serie a..."
5. FIN DB ? ; programa principal
6. ORG 2000H
7. INICIO: MOV BX, OFFSET
8. MOV SACADOS, 0
9. ; programo la USART
10. MOV AL, 51H ; binario=01010001 } Configura USART en modo asincrónico, sin
11. OUT USART+2, AL } interrupciones a 6 baudios
12. TEST: IN AL, USART+2
13. AND AL, 81H ; binario=10000001 } Comprueba TxRdy y DSR para verificar que la impresora
14. CMP AL, 81H } pueda recibir datos y este disponible DOUT para escribir un
15. JNZ TEST } carácter
16. MOV AL, [BX] } Envía carácter a impresora, es seguro escribir
17. OUT USART+1, AL } DOUT
18. INC BX
19. INC SACADOS
20. CMP SACADOS, OFFSET FIN-OFFSET TABLA } Cuenta caracteres que faltan imprimir y
21. JNZ TEST } verifica si se envió el último
22. INT 0
23. END
24.
```

# USART MSX88 – impresora con XON/XOFF



Programa que envía datos a la impresora a través de la USART usando el protocolo XON/XOFF realizando la comunicación entre CPU y USART por consulta de estado. Ejecutar en configuración P1 C4 y utilizar el comando PI que corresponda.

Tener en cuenta en la configuración de transmisión:

- modo asincrónico
- consulta de estado (polling), no usar interrupción
- misma velocidad de transmisión que la impresora
- reiniciar los errores al iniciar la comunicación
- poner el bit DTR de la USART en 1
- ejecutar comando PI X para modo XON/XOFF de impresora

# USART MSX88 – impresora con XON/XOFF



```
1. 1. USART EQU 60H
2. 2. XON EQU 11H
3. 3. XOFF EQU 13H
4. 4. ; definición de datos
5. 5. ORG 1000H
6. 6. caracteres DW 0
7. 7. TABLA DB "Comunicacion serie..."
8. 8. FIN DB ?
9. 9. ; PROGRAMA PRINCIPAL
10. 10. ORG 2000H
11. 11. INICIO: MOV BX, OFFSET TABLA
12. 12. ; programo la USART
13. 13. MOV AL, 51H ;binario=01010001 } Configura USART en modo asincrónico, sin interrupciones
14. 14. OUT USART+2, AL } a 6 baudios, igual que con DTR
15. 15. TEST: IN AL, USART+2 } Verifica TxRdy para determinar que este
16. 16. AND AL, 01H } disponible DOUT para escribir, NO verifica que la
17. 17. CMP AL, 01H } impresora pueda recibir datos
18. 18. JNZ TEST
19. 19. MOV AL, [BX]
20. 20. OUT USART+1, AL } Envía carácter a impresora, es seguro escribir
21. 21. INC BX } DOUT
22. 22. INC caracteres
23. 23. CMP caracteres, (OFFSET FIN) - (OFFSET TABLA)
24. 24. JZ FINAL
```

# USART MSX88 – impresora con XON/XOFF



```
25. IN AL, USART+2
26. AND AL, 02H
27. CMP AL, 02H
28. JZ RXON
29. JMP TEST
30. ; espera recibir XON
RECIBIR: IN AL, USART+2
32. AND AL, 02H
33. CMP AL, 02H
34. JNZ RECIBIR
RXON: IN AL, USART
36. MOV AH, AL
37. CMP AL, XON
38. JZ TEST
39. CMP AH, XOFF
40. JZ RECIBIR
FINAL: INT 0
42. END
```

- } Verifica RxRdy para determinar si la impresora envió un dato (XON o XOFF). Si no recibió un dato salta a TEST para enviar un nuevo carácter a imprimir. Si recibió un dato salta para verificar cual es
- } Cuenta caracteres que faltan imprimir y verifica si se envió el último
- } Verifica si el carácter enviado por la impresora es XON. Si es XON salta a TEST y continúa para enviar un nuevo carácter
- } Si el carácter enviado es XOFF salta a RECIBIR hasta que llegue el carácter XON
- } Si no es XON o XOFF finaliza el programa

# DMA - ¿Por que?

Supongamos que queremos transferir 200 bytes de una dirección de memoria a otra. Tenemos:

- AX= dirección origen
- DX=dirección destino
- CL=200

|            |   | Accesos |
|------------|---|---------|
| Otro_byte: | MOV BX, AX ;Dirección actual de origen    | 2       |
|            | MOV CH, [BX] ;Recupera byte               | 2+1     |
|            | INC AX ;Apunta a próximo byte             | 2       |
|            | MOV BX, DX ;Dirección actual de destino   | 2       |
|            | MOV [BX], CH ;transfiere 1 byte a destino | 2+1     |
|            | INC DX ;Apunta a próximo byte             | 2       |
|            | DEC CL ;Decrementa bytes faltantes        | 2       |
|            | JNZ otro_byte                             | 2       |

¿Cuántos accesos a memoria serían necesarios para transferir 1 byte?

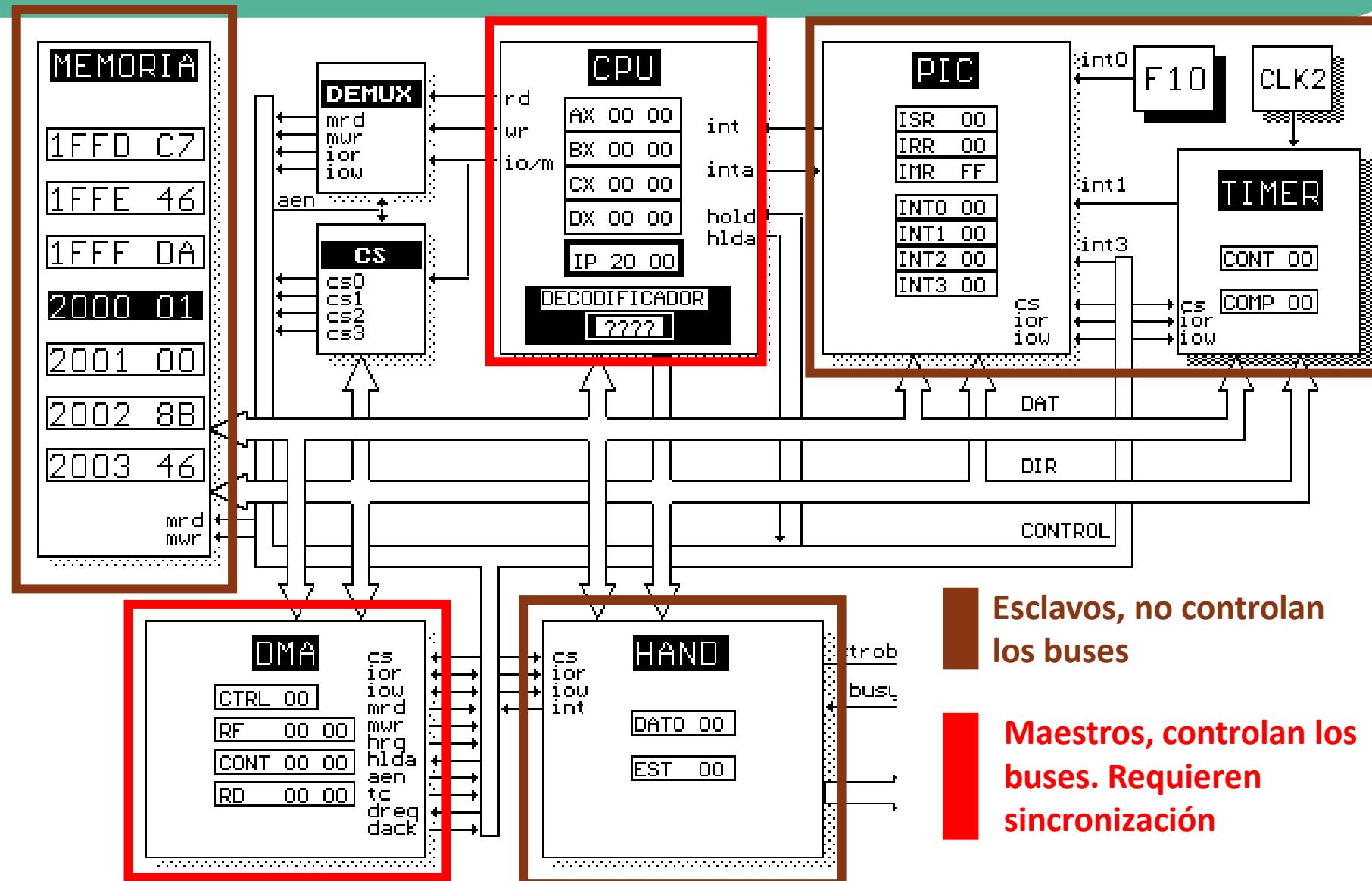
→ 2

18

# DMA - Características

- Es un dispositivo que permite transferir de manera eficiente bloques de datos a memoria y a dispositivos
- Realiza accesos a memoria y a otros dispositivos como la CPU, es “una CPU especializada en transferencia”
- Comparte los buses con la CPU, por lo que se necesitan señales especiales para que sincronicen los accesos. Los pasos serían:
  - DMAC pide a la CPU acceso a los buses
  - La CPU acepta y se desconecta de los buses (no ejecuta mas instrucciones)
  - El DMAC toma el control de los buses y realiza la transferencia
  - El DMAC indica a la CPU que termino la transferencia
  - La CPU toma el control (reinicia la ejecución de instrucciones)

# DMA - Características



# DMA - Características

Tipos de transferencias:

- Memoria → Memoria
- Memoria → Periférico
- Periférico → Memoria

Modos de Transferencias:

- Por bloque o ráfagas (burst): acuerda con la CPU para generar una transferencia completa, sin detenerse
- Por demanda o robo de ciclos: transfiere el bloque de a un byte por vez. Se alterna con la CPU entre byte y byte ( La CPU ejecuta instrucción, DMA transfiere byte, CPU ejecuta instrucción, DMA transfiere byte, etc.
- Al transferir el último byte genera una interrupción para que el programador detenga o reprograme otra transferencia

# DMA - Características

Para las transferencias Memoria-Memoria requiere:

- Una dirección origen
- Una dirección destino
- Una cantidad de bytes a transferir
- Modo en el que se va a transferir (Bloque o Demanda)

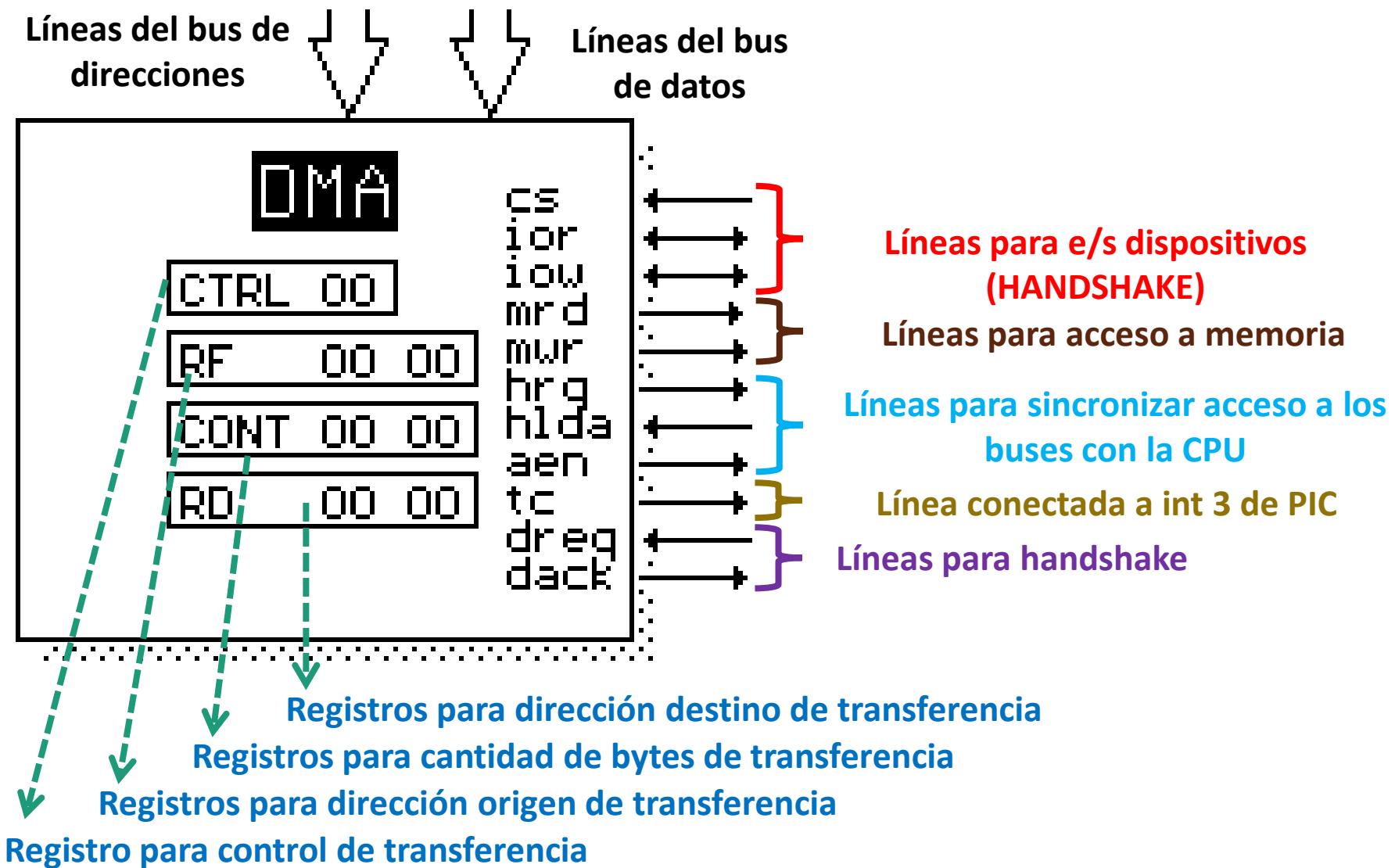
Para las transferencia Memoria-Periférico solo admite el Handshake (impresora) porque esta conectado (cableado) físicamente. Requiere:

- Una dirección origen
- Una cantidad de bytes a transferir
- Modo en el que se va a transferir (Bloque o Demanda)

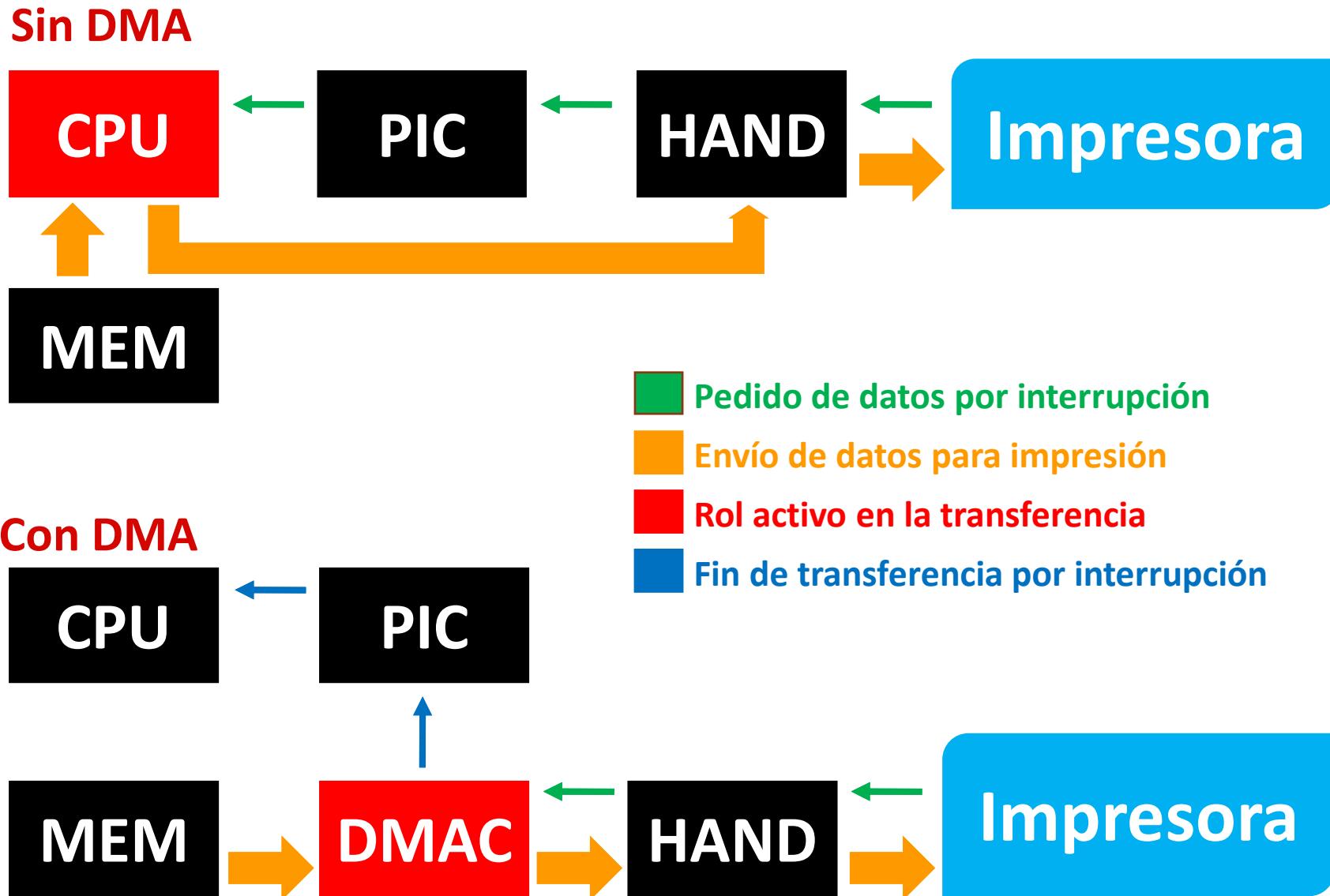
Transferencias Periférico-Memoria no admite. ¿Por qué?

- Porque el handshake no recibe datos de la impresora

# DMA - Conexión



# DMA - Conexión



# DMA - Conexión

- Conectado a PIC a través de la línea **Int3**
- El comando del simulador “P1 C3” muestra esta configuración DMA + Handshake + impresora
- Ubicado en la dirección 50H
- Tiene 8 Registros

# DMA - Conexión

Dirección fuente (origen), 16 bits (8+8):

- RFL (50H)
- RFH (51H)

Cantidad a transferir, 16 bits (8+8):

- CONTL (52H)
- CONTH (53H)

Dirección destino, 16 bits (8+8):

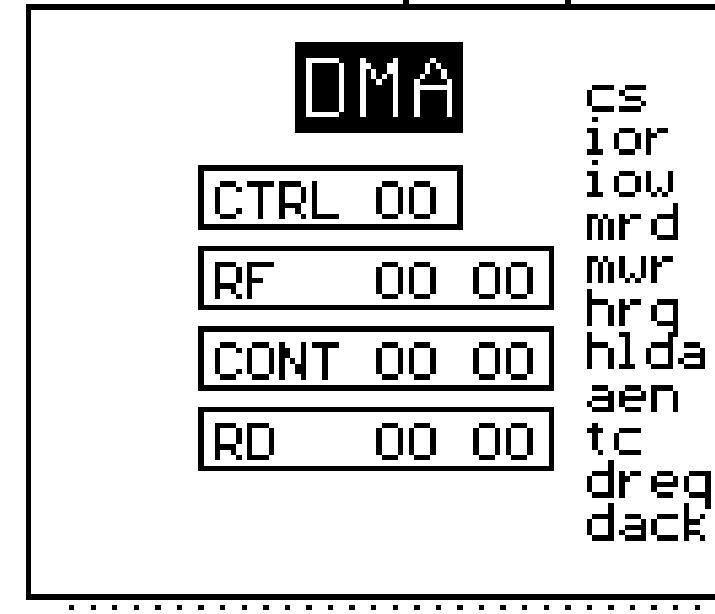
- RDL (54H)
- RDH (55H)

Control, establece como se hace la transferencia, 8 bits

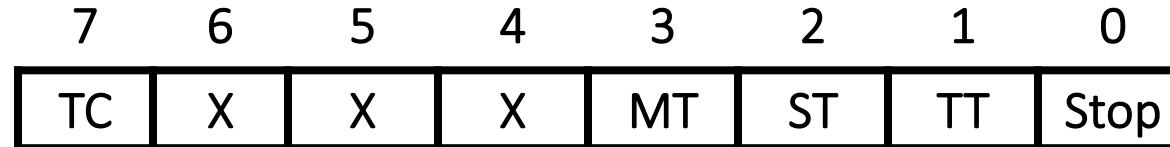
- CONTROL (56H)

Inicio de transferencia, 8 bits:

- Arranque (57H): solo se necesita leer o escribir el registro con cualquier valor



# DMA - Registros



- Stop (detenido):  
Bit=1 → Transferencia detenida      Bit=0 → Transferencia en curso  
escribir un 1 detiene la transferencia, escribir 0 no tiene efecto. Para reiniciar la transferencia es necesario leer o escribir el registro de arranque
- TT(Tipo de Transferencia):  
Bit=0 → Periférico-Memoria o Memoria-Periférico      Bit=1 → Memoria-Memoria
- ST (Sentido de transferencia, restringido a Tipo cuando es 0):  
Bit=0 → Periférico-Memoria      Bit=1 → Memoria-Periférico
- MT (Modo de transferencia):  
Bit=0 → Bajo Demanda      Bit=1 → Por Bloque
- Bits 4 a 6 no se usan
- TC(Transferencia finalizada, solo lectura, genera interrupción si esta habilitada):  
Bit=0 → Transferencia no finalizada      Bit=1 → Transferencia finalizada

# DMA – Ejercicio Memoria-Memoria

DMA. Transferencia de datos memoria-memoria.

Escribir un programa que copie una cadena de caracteres almacenada a partir de la dirección 1000H en otra parte de la memoria, utilizando el DMAC en modo de transferencia por bloque (ráfaga).

La cadena original se debe mostrar en la pantalla de comandos antes de la transferencia. Una vez finalizada, se debe visualizar en la pantalla la cadena copiada para verificar el resultado de la operación.

Ejecutar el programa en la configuración P1 C3.

# DMA - Ejercicio Memoria-Memoria

```
PIC EQU 20H
DMA EQU 50H
N_DMA EQU 20
ORG 80
IP_DMA DW RUT_DMA.
```

|                      |                                |
|----------------------|--------------------------------|
| <b>ORG 1000H</b>     |                                |
| MSJ DB "FACULTAD DE" | Cantidad de bytes a transferir |
| DB " INFORMATICA"    |                                |
| FIN DB ?             |                                |
| NCHAR DB ?           |                                |
| <b>ORG 1500H</b>     |                                |
| COPIA DB ?           | Dir destino del bloque         |

; rutina atencion interrupcion del CDMA

|                  |                      |                             |                 |
|------------------|----------------------|-----------------------------|-----------------|
| <b>ORG 3000H</b> |                      |                             |                 |
| RUT_DMA:         | MOV AL, OFFH         | Muestra mensaje transferido |                 |
|                  | OUT PIC+1, AL        |                             | IMR = 1111 1111 |
|                  | MOV BX, OFFSET COPIA |                             |                 |
|                  | MOV AL, NCHAR        |                             |                 |
|                  | INT 7                |                             |                 |
|                  | MOV AL, 20H          |                             |                 |
|                  | OUT PIC, AL          |                             |                 |
|                  | IRET                 |                             |                 |

```
ORG 2000H
CLI
MOV AL, N_DMA
OUT PIC+7, AL
MOV AX, OFFSET MSJ
OUT DMA, AL
MOV AL, AH
OUT DMA+1, AL
MOV AX, OFFSET FIN-OFFSET MSJ
OUT DMA+2, AL
MOV AL, AH
OUT DMA+3, AL
MOV AX, OFFSET COPIA
OUT DMA+4, AL
MOV AL, AH ;
OUT DMA+5, AL
MOV AL, OAH
OUT DMA+6, AL
MOV AL, 0F7H
OUT PIC+1, AL
STI
MOV BX, OFFSET MSJ
MOV AL, OFFSET FIN-OFFSET MSJ
MOV NCHAR, AL
INT 7
MOV AL, 7H
OUT DMA+7, AL
INT 0
END
```

Configura INT3 del PIC

Dir origen del bloque

Transferencia mem a mem por bloque/ráfaga

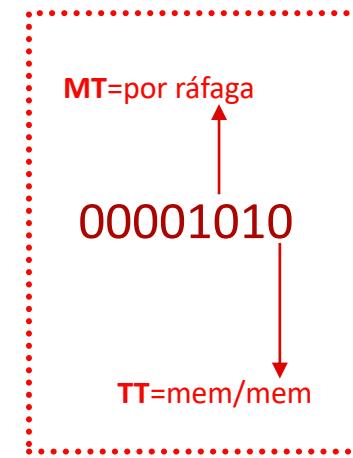
IMR = 1111 0111

Inicia transferencia

Inmediatamente luego de ejecutar la instrucción OUT, la CPU accede al pedido del DMA

Muestra mensaje original

| DMAC |                    |
|------|--------------------|
| 50H  | RFL 00H            |
| 51H  | RFH 10H            |
| 52H  | ContL 00H          |
| 53H  | ContH 00H          |
| 54H  | RDL 00H            |
| 55H  | RDH 15H            |
| 56H  | Control 1000 1010  |
| 57H  | Arranque 0000 0111 |



# DMA – Ejercicio Memoria-Periférico

DMA. Transferencia de datos memoria-periférico.

Escribir un programa que transfiera datos desde la memoria hacia la impresora sin intervención de la CPU, utilizando el DMAC en modo de transferencia bajo demanda (robo de ciclo).

# DMA - Ejercicio Memoria-Periférico

|       |     |     |
|-------|-----|-----|
| PIC   | EQU | 20H |
| HAND  | EQU | 40H |
| DMA   | EQU | 50H |
| N_DMA | EQU | 20  |

**ORG 80**

IP\_DMA DW RUT\_DMA

**ORG 1000H**

|      |    |                |
|------|----|----------------|
| MSJ  | DB | " INFORMATICA" |
| FIN  | DB | ?              |
| FLAG | DB | 0              |

*; rutina atención interrupción del CDMA*

**ORG 3000H**

RUT\_DMA: MOV AL, 0  
OUT HAND+1, AL } Deshabilita interrupción del HAND  
MOV FLAG, 1 Indica fin de lazo  
MOV AL, 0FFH } IMR = 1111 1111  
OUT PIC+1, AL  
MOV AL, 20H  
OUT PIC, AL  
IRET

**ORG 2000H**

CLI  
MOV AL, N\_DMA } Configura INT3 del PIC  
OUT PIC+7, AL  
MOV AX, OFFSET MSJ } Dir origen del bloque  
OUT DMA, AL  
MOV AL, AH  
OUT DMA+1, AL  
MOV AX, OFFSET FIN-OFFSET MSJ } Contidad de bytes a transferir  
OUT DMA+2, AL  
MOV AL, AH  
OUT DMA+3, AL  
MOV AL, 4 } CTRL = 0000 0100  
OUT DMA+6, AL  
MOV AL, 0F7H } IMR = 1111 0111  
OUT PIC+1, AL  
OUT DMA+7, AL } Inicia transferencia  
MOV AL, 80H  
OUT HAND+1, AL } HAND por interrupción  
STI

LAZO: CMP FLAG, 1  
JNZ LAZO  
INT 0  
END

MT=por demanda o robo de ciclo

00000100

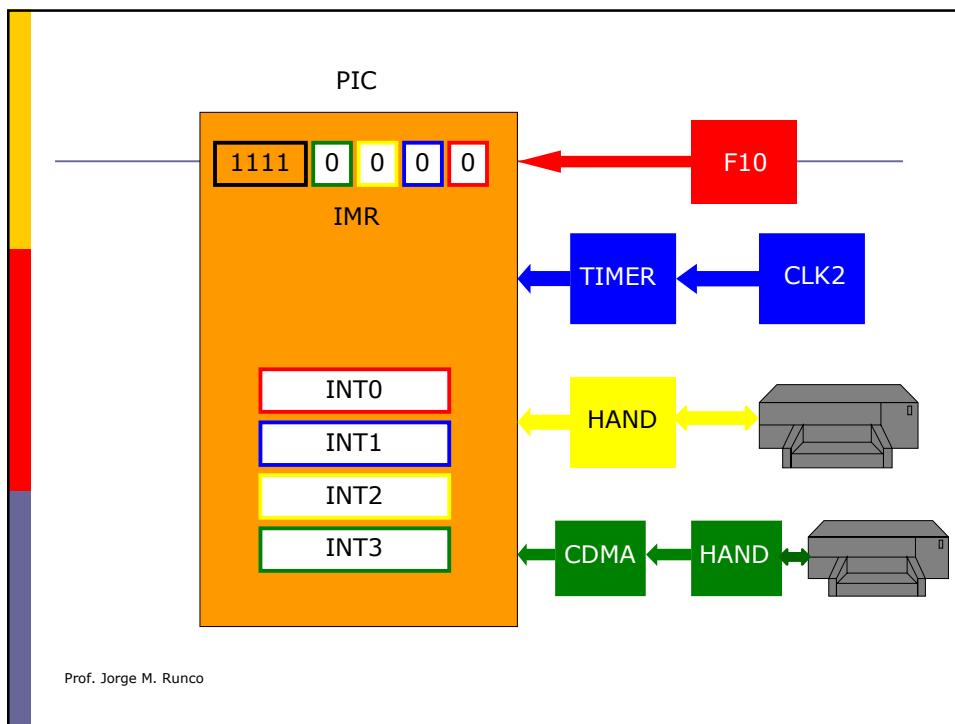
TT=disp o mem

ST=mem → disp

# ARQUITECTURA DE COMPUTADORAS

Curso 2020  
Prof. Jorge Runco  
Comunicación CPU-Impresora

1



2

1

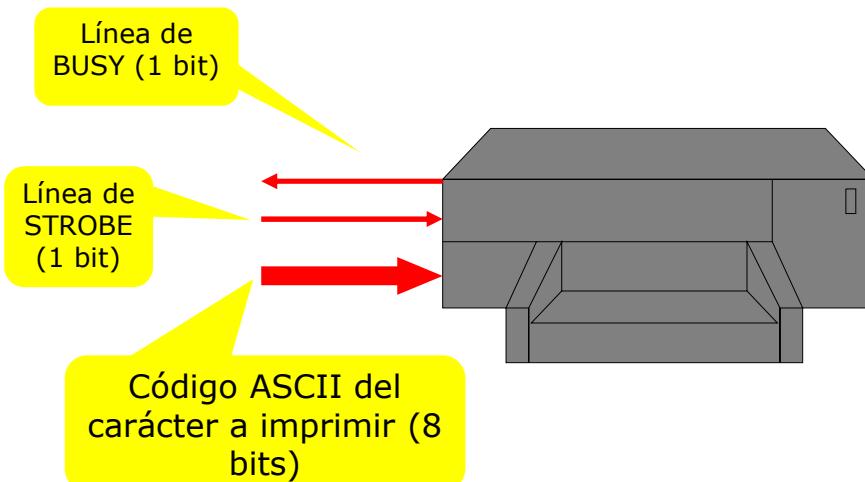
## Comunicación cpu - impresora

- Tres alternativas de conexionado :
- CPU – PIO – IMPRESORA
- CPU – HAND – IMPRESORA
- CPU – CDMA – HAND – IMPRESORA

Prof. Jorge M. Runco

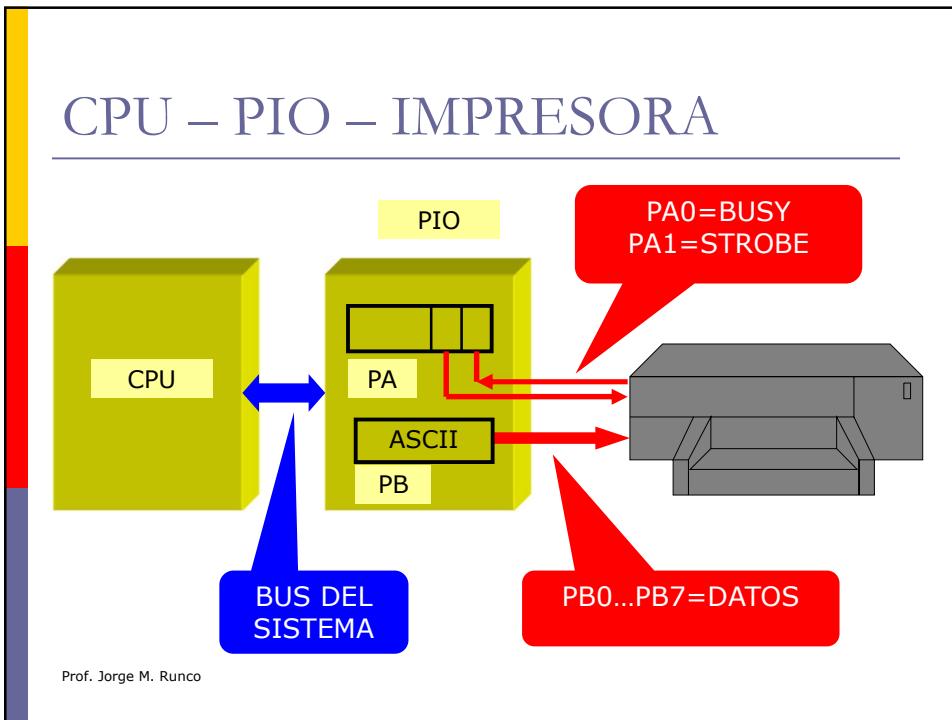
3

## Comunicación cpu - impresora

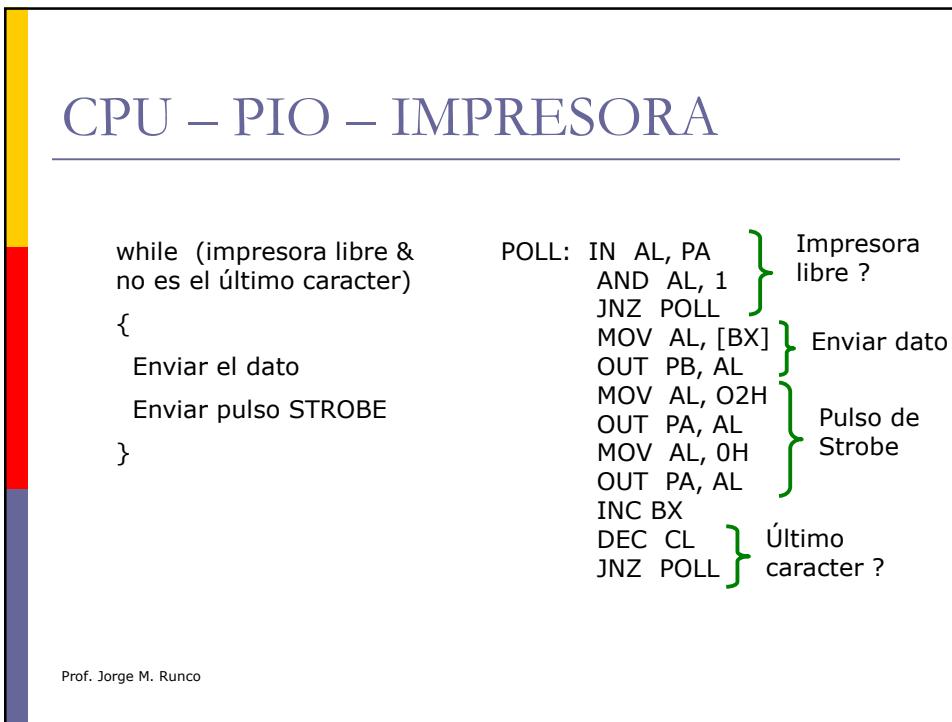


Prof. Jorge M. Runco

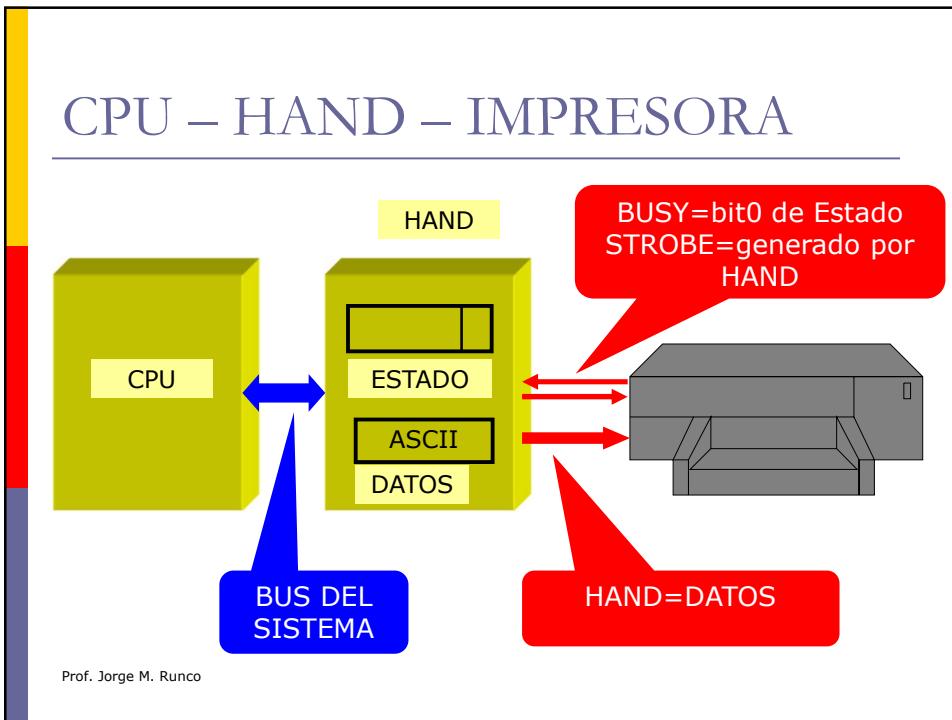
4



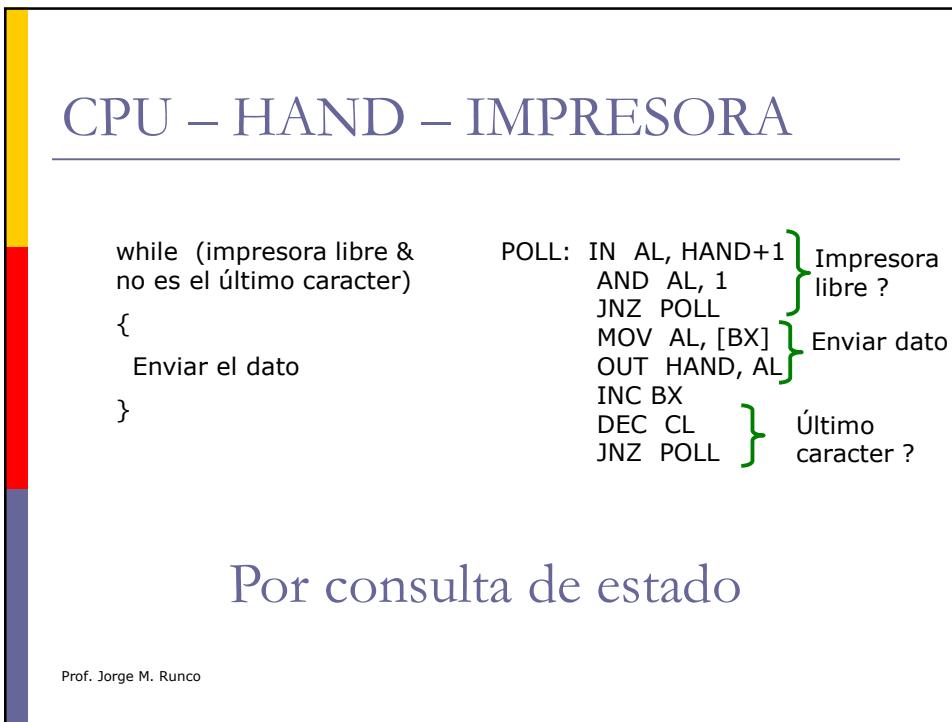
5



6



7



8

## CPU – HAND – IMPRESORA

```
while (no es el último carácter)
{
    .....
}
```

```
POLL: CMP CL, 0
      JNZ POLL
```

```
RUT_HND: PUSH AX
          MOV AL, [BX]
          OUT HAND, AL
          INC BX
          DEC CL
          MOV AL, 20H
          OUT PIC, AL
          POP AX
          IRET
```

### Interrupción generada por el HAND

Prof. Jorge M. Runco