

## Tema 6

### El sistema de Entrada/Salida

Curso 2007-2008

## Objetivos

- Comprender la diferencia entre periférico e interfaz
- Comprender el direccionamiento de los interfaces
- Conocer ejemplos típicos de interfaces
- Ser capaz de programar Entrada/Salida por muestreo
- Comprender el concepto de interrupción y su implementación en la Unidad de Control.
- Ser capaz de programar Entrada/Salida por interrupciones.

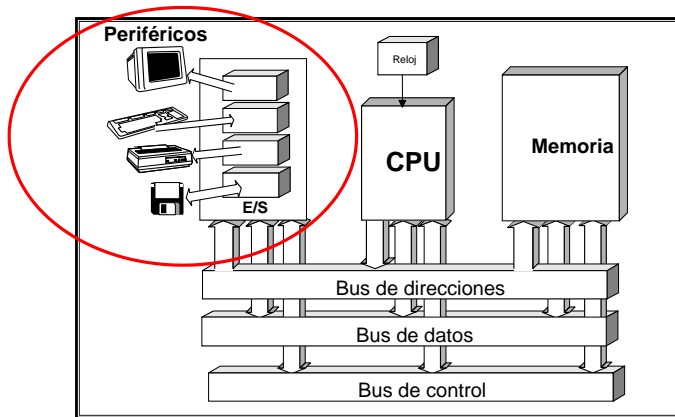
# Esquema

- 1 Conceptos preliminares
- 2 Interfaces
- 3 Ejemplos: Interfaces para la CPU elemental
- 4 Programación de la Entrada/Salida por muestreo
- 5 Interrupciones

# Esquema del apartado 1

- 1 Conceptos preliminares
- 2 Interfaces
- 3 Ejemplos: Interfaces para la CPU elemental
- 4 Programación de la Entrada/Salida por muestreo
- 5 Interrupciones

## E/S en la arquitectura Von Neumann



# Periféricos e interfaces

## Periférico

El periférico es un dispositivo que interacciona con el “mundo exterior” (personas, otros ordenadores, almacén permanente).

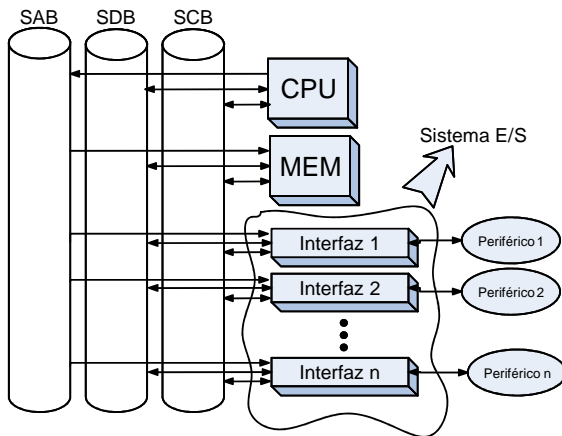
## Interfaz

El interfaz es el dispositivo que permite conectar un periférico a los buses de sistema.

## Ejemplos y velocidades

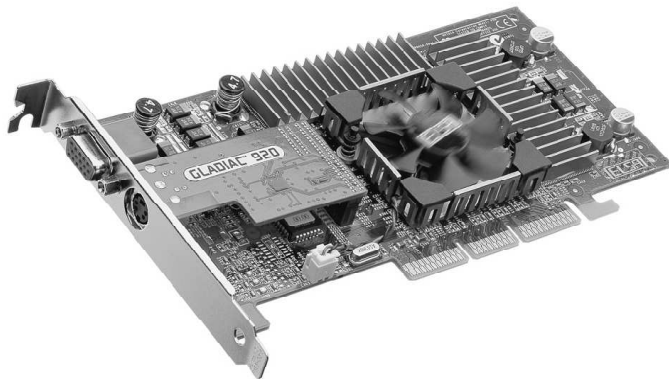
Dispositivo	Comportamiento	Interlocutor	Velocidad (KB/seg)
Teclado	entrada	humano	0,01
Ratón	entrada	humano	0,02
Entrada de voz	entrada	humano	0,02
Scanner	entrada	humano	400,00
Salida de voz	salida	humano	0,60
Impresora de líneas	salida	humano	1,00
Impresora láser	salida	humano	200,00
Pantalla gráfica	salida	humano	60.000,00
Modem	entrada o salida	máquina	2,00-8,00
Red/LAN	entrada o salida	máquina	500,00-6.000,00
Disco flexible	almacenamiento	máquina	100,00
Disco óptico	almacenamiento	máquina	1.000,00
Cinta magnética	almacenamiento	máquina	2.000,00
Disco magnético	almacenamiento	máquina	2.000,00-10.000,00

## Conexión de interfaces a los buses

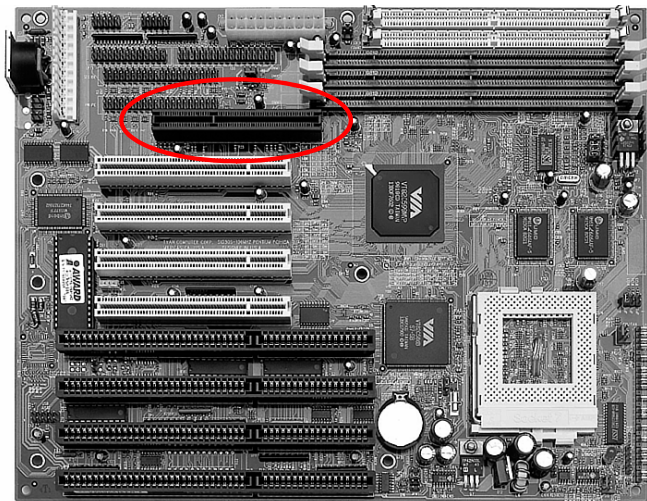




## Imagen real: interfaz de vídeo



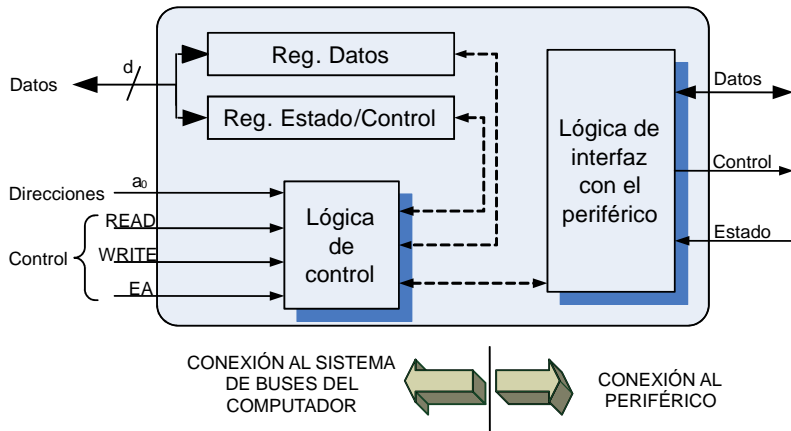
## Imagen real: conexión a los buses



## Esquema del apartado 2

- 1 Conceptos preliminares
- 2 Interfaces**
  - Estructura interna
  - Direccionamiento
- 3 Ejemplos: Interfaces para la CPU elemental
- 4 Programación de la Entrada/Salida por muestreo
- 5 Interrupciones

# Diagrama genérico



## Registros básicos

### Registro de datos

- Contiene temporalmente los datos transferidos entre CPU y periférico.
- Lo que la CPU escribe, se envía al periférico.
- Cuando la CPU lee, se recibe información del periférico.

### Registro de control/estado

- Cuando la CPU escribe, se le da una orden de **control** al interfaz o periférico.
- Cuando la CPU lee, obtiene información sobre el **estado** del interfaz o periférico.

## Concepto de direccionamiento del interfaz

Cuando la CPU quiera enviar o leer un dato de un registro (datos o control) de un interfaz, debe especificar:

- Con qué interfaz quiere hablar
- Con qué registro dentro de ese interfaz.

### Algunos interfaces tienen memoria dentro

Y en ese caso la CPU debe especificar en qué dirección de memoria dentro del interfaz quiere leer o escribir.

### Solución

- Cada registro y cada posición de memoria de cada interfaz recibe una **dirección** única.
- La CPU pone en el bus de direcciones la dirección del elemento con que quiere dialogar.

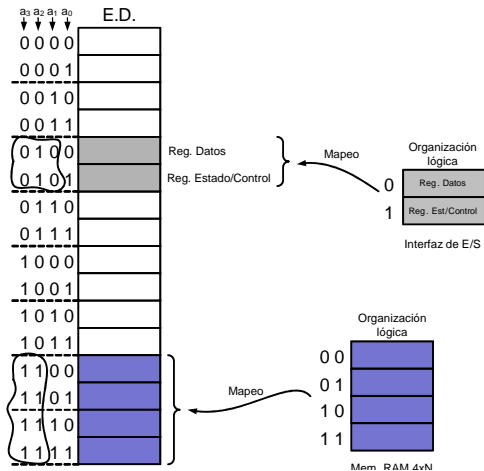
## La E/S en el espacio de direcciones

Si la CPU utiliza el bus de direcciones tanto para “hablar” con la memoria como para “hablar” con los interfaces, los interfaces en el fondo se “ven” desde la CPU como un módulo de memoria más, en el mapa de direcciones.

### Ejemplo para una CPU imaginaria

- El Espacio de Direcciones tiene sólo 16 posiciones
- El Bus de direcciones es de 4 bits
- Se tiene un módulo de memoria que ocupa 4 posiciones, en el rango Ch-Fh
- Un interfaz de E/S con dos registros se mapeará en el rango 8h-9h

# Mapa de memoria del ejemplo anterior

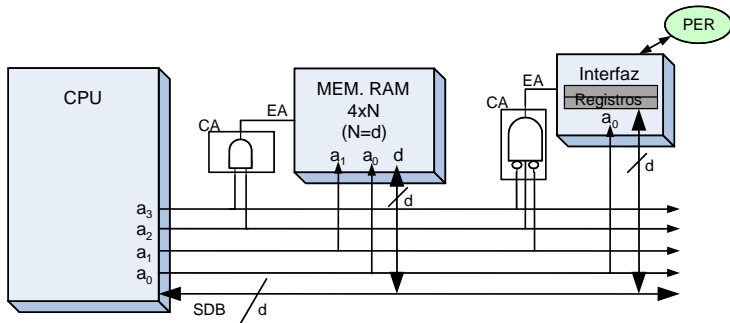




## Circuitos de activación del ejemplo anterior

Se trata por tanto de asignar direcciones diferentes a los módulos de memoria y a los interfaces de E/S.

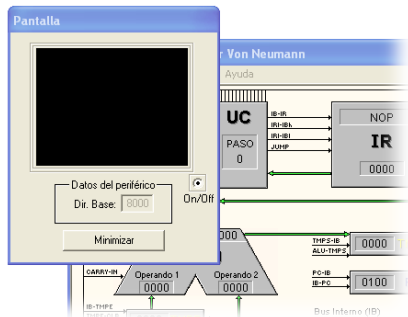
Esto se logra con los circuitos de activación apropiados.



## Esquema del apartado 3

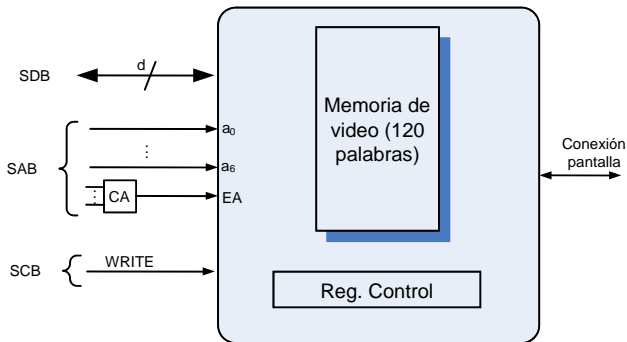
- 1 Conceptos preliminares
- 2 Interfaces
- 3 Ejemplos: Interfaces para la CPU elemental**
  - Interfaz de salida: Vídeo
  - Interfaz de entrada: Teclado
  - Interfaz de entrada/salida: Luces
- 4 Programación de la Entrada/Salida por muestreo
- 5 Interrupciones

- Al igual que la propia CPU elemental, su pantalla es simulada.
- Es una “pantalla” muy pequeña.
- Sólo puede mostrar texto (15 letras por fila, 8 filas)



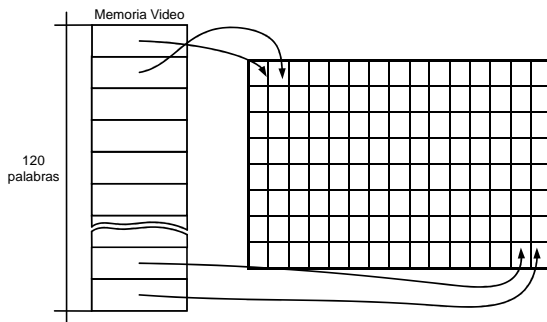
## El interfaz de Video de la CPU elemental

Para controlar la pantalla anterior, tenemos un interfaz con la siguiente estructura interna:



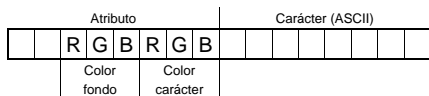
## Memoria de vídeo

- Tiene 120 posiciones.
- Cada posición almacena la información que debe mostrarse en un lugar de la pantalla.
- La información consiste en: código (ASCII) de letra + colores



# Significado de cada dato en la memoria de vídeo

- Cada dato es de 16 bits
- Los dos más altos no se usan
- Los 8 inferiores contienen el código ASCII del carácter a mostrar
- Los restantes 6 contienen información RGB del color de la letra y del fondo.

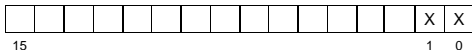


Color resultante según las componentes RGB activadas:

RGB	Color
000	Negro
001	Azul (B)
010	Verde (G)
011	Cian
100	Rojo (R)
101	Morado
110	Amarillo
111	Blanco

## Registro de control

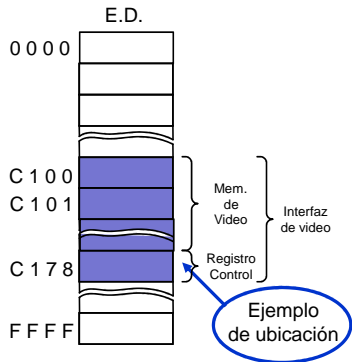
- El registro de control tiene 16 bits, pero sólo se usan dos.
- Al escribir en ese registro, se realiza una acción sobre la pantalla:
  - Bit 0: Si 1, borra la memoria de vídeo. Si 0, no hace nada.
  - Bit 1: Si 1, enciende la pantalla. Si 0, la apaga.



- No se puede leer de este registro.

## Mapecto en el espacio de direcciones

- El dispositivo usa un rango de 128 direcciones consecutivas. .
- Se puede ubicar en direcciones múltiplo de 128.
- El simulador nos pregunta la “dirección base” al conectarlo.
- En esa dirección base comienza la memoria de vídeo.
- A continuación de la memoria de vídeo aparece el registro de control.





## Programación de la interfaz de vídeo

### Idea clave

“Imprimir” en pantalla no es más que escribir datos en la memoria de vídeo

### Instrucción clave

La instrucción para escribir datos en el Espacio de Direcciones es:

`MOV [Ri], Rs`

`Rs` es el registro que contiene el dato a escribir

`Ri` es el registro que contiene la dirección donde escribir

# Ejemplo

*; Este programa muestra el texto "Hola" en la segunda línea de la pantalla  
; con letras rojas sobre fondo amarillo (asumir interfaz de vídeo mapeado en C100h)*

ORIGEN 0100h

INICIO *ini*

.DATOS

*cadena* VALOR "HOLA",0

.CODIGO

XOR R3, R3, R3 ; *terminador de cadena*

MOVL R5, BYTEBAJO DIRECCION *cadena*

MOVH R5, BYTEALTO DIRECCION *cadena*

MOVL R6, 0Fh

MOVH R6, 0C1h ; *R6 apunta a la segunda línea*

*bucle*: MOV R0, [R5] ; *Carga en R0 el ASCII*

COMP R0, R3

BRZ *salir*

MOVH R0, 34h ; *Carga en R0 el atributo*

MOV [R6], R0 ; *"Imprime"*

INC R5 ; *a por otra letra*

INC R6 ; *en otra posición de pantalla*

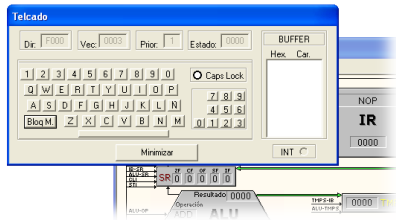
JMP *bucle*

*salir*:

FIN

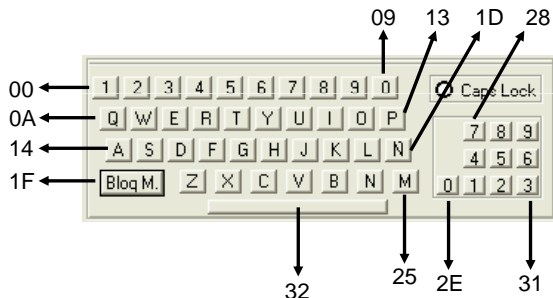
## El Teclado de la CPU elemental

- Se trata también de un teclado simulado.
- Contiene las teclas habituales en un teclado.
- No obstante algunas están ausentes

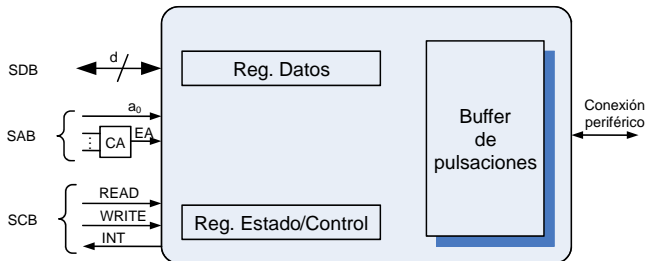


## Funcionamiento del teclado

- Cada tecla tiene asignado un código (SCAN), de 8 bits.
- Este código identifica la posición de la tecla en el teclado, no el carácter que lleva escrito.
- Cuando una tecla es pulsada, el teclado envía al interfaz el código SCAN.



## El Interfaz de teclado



## Elementos en el interfaz de teclado

### Buffer de pulsaciones

- Cada vez que el teclado envía un SCAN, se almacena en este buffer, hasta que la CPU lo lea. Además, le añade el código ASCII.
- Sólo tiene espacio para 16 pulsaciones. Si se llena, se ignoran las pulsaciones adicionales.

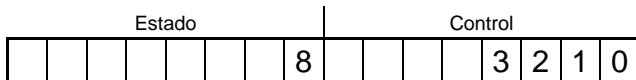
### Registro de datos

Cuando la CPU lee de este registro, se le envía la pulsación más antigua en el buffer, y se elimina del mismo.

### Registro de estado/control

- Cuando la CPU lo lee, averigua si hay teclas esperando en el buffer.
- Cuando la CPU lo escribe, se ejecutan ciertos comandos de control.

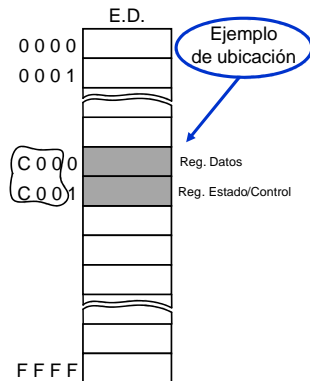
## Registro de estado/control



- En la lectura (estado):
  - Bit 8: 1 si hay teclas en el buffer, 0 si no.
- En la escritura (control):
  - Bit 3: activar/desactivar interrupciones
  - Bit 2: borrar todos los caracteres del buffer
  - Bit 1: borrar último carácter del buffer
  - Bit 0: borrar primer carácter del buffer

## Mapecto del interfaz de teclado

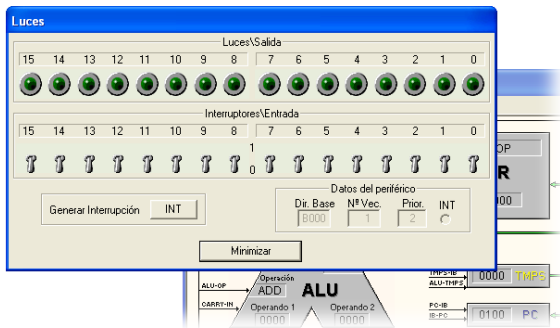
- El buffer del teclado no es direccionable
- Sólo tiene dos elementos direccionables
- Por tanto sólo ocupa dos posiciones en el E.D.
- Basta 1 bit para direccionar uno de sus dos elementos:
  - 0: direcciona el registro de datos
  - 1: direcciona el registro de control/estado
- Los restantes 15 van al circuito de activación





## El dispositivo luces

Se estudia en las sesiones prácticas.



## Esquema del apartado 4

- 1 Conceptos preliminares
- 2 Interfaces
- 3 Ejemplos: Interfaces para la CPU elemental
- 4 Programación de la Entrada/Salida por muestreo**
  - El problema de la sincronización
  - La solución del muestreo
- 5 Interrupciones

## ¿Cuándo sincronizar?

### No es necesario sincronizar si...

- El interfaz siempre está listo para recibir o enviar
- Debido a que es lo suficientemente rápido
- Ejemplo: interfaz de vídeo

### Es necesario sincronizar si...

- El interfaz es más lento que la CPU
- La CPU debe esperar a que el interfaz esté listo o tenga datos
- Ejemplo: teclado

## En qué consiste el muestreo

- El interfaz dispone de un **bit de estado** que indica cuándo está listo.
- La CPU lee ese bit
  - Si está listo, realiza la transferencia (MOV).
  - Si no, vuelve a leer el bit

### Problema

La CPU está malgastando su tiempo (y energía) leyendo el bit una y otra vez.

## ¿Cómo leer el bit de estado?

- El bit de estado del interfaz suele formar parte de un **registro de estado**.
- Es posible leer el registro de estado con MOV, pero ¿cómo leer un solo bit?
- ¿Cómo saber si un bit dentro de un registro es 1 ó 0?

### Máscaras de bits

- La solución es aplicar una **máscara**
- La máscara es un dato que tiene todos ceros, excepto en el bit que queremos comprobar.
- Se realiza la operación AND entre el registro con el dato y la máscara.
  - Si sale cero, el bit en cuestión era 0
  - Si sale distinto de cero, era 1

## Ejemplo: muestreo del teclado

Hacer un programa que lea tres pulsaciones de tecla (sincronizando por muestreo) y deje los códigos leídos en un array de tres posiciones.

Asumir teclado mapeado en C000h

### Listado

ORIGEN 100h

INICIO ini

.PILA 10h

.DATOS

teclas\_leidas VALOR 0,0,0 ; *Aquí hay que dejar los códigos*

.CODIGO

???

FIN

## Bucle de sincronización

- Para esperar hasta que haya una tecla disponible, es necesario programar un **bucle de sincronización**
- En cada iteración, se lee el registro de estado, se le aplica la máscara, y si sale cero, se repite el bucle.
- Cuando se salga del bucle, ya se puede leer la tecla.

### La máscara

- En el registro de control de teclado, el bit de estado es el 8
- Una máscara para detectar ese bit tendrá el valor 0100h

# Programación del bucle de sincronización

Usar:

- R2 para contener la dirección en el E.D. del registro de estado del teclado.
- R4 para recibir una copia del registro de estado del teclado.
- R6 para contener la máscara.

## Solución

*; Inicializaciones*

MOVL R2, 01h

MOVH R2, 0C0h

MOVL R6, 00h

MOVH R6, 01h

*; Bucle de sincronizacion*

esperar:

MOV R4, [R2] *; leer el registro de estado*

AND R4, R4, R2 *; aplicar la máscara*

BRZ esperar *; si bit "listo" es cero, repetir*



# Programa completo

Al salir del bucle de sincronización, ya se puede leer la tecla y dejarla en el array. Esto se repetirá tres veces.

```
ORIGEN 100h
INICIO ini
.PILA 10h
.DATOS
teclas_leidas VALOR 0,0,0 ; Aqui hay que dejar los códigos
.CODIGO
    ; Inicializaciones
    ; R1=C000h apunta al reg. dat. de teclado
    ; R0=direccion de la variable "teclas_leidas"
    ; R5=3, número de repeticiones del bucle principal
    ; R2=C001h apunta al reg. estado. de teclado
    ; R6=0100h, máscara para el bit "listo"
    MOVL R1, 00h
    MOVH R1, 0C0h
    MOVL R0, BYTEBAJO DIRECCION teclas_leidas
    MOVH R0, BYTEALTO DIRECCION teclas_leidas
```

*sigue >>*

# Programa completo

➤➤ *sigue*

```
MOVL R5, 3
MOVH R5, 0
MOVL R2, 01h
MOVH R2, 0C0h
MOVL R6, 00h
MOVH R6, 01h
; Bucle principal
```

*repetir:*

*esperar:*

```
MOV R4, [R2] ; leer el registro de estado
AND R4, R4, R2 ; aplicar la máscara
BRZ esperar ; si bit "listo" es cero, repetir
; Ya podemos leer la tecla, usando R3 como destino
MOV R3, [R1]
; y transferirla al array
MOV [R0], R3
; Vamos a por la siguiente tecla
INC R0
DEC R5
BRNZ repetir
```

FIN

## Esquema del apartado 5

- 1 Conceptos preliminares
- 2 Interfaces
- 3 Ejemplos: Interfaces para la CPU elemental
- 4 Programación de la Entrada/Salida por muestreo
- 5 Interrupciones**
  - Concepto
  - Vectorización
  - Priorización
  - Implementación en la micromáquina
  - Programación de rutinas de servicio

# Concepto

## Idea clave

En lugar de hacer que la CPU consulte continuamente si el interfaz está listo, que sea el interfaz el que “avise” a la CPU.

Esta idea requiere soporte en el *hardware*, en forma de nuevas líneas de control y nuevos pasos en la unidad de control.

# Concepto

## Elaboración de la idea

- La CPU está ejecutando un programa “normalmente”
- Cuando un dispositivo tiene datos listos, **genera una interrupción**
- La CPU abandona temporalmente lo que estaba haciendo
- Pasa a ejecutar un sub-programa específico para atender al dispositivo que interrumpió
- Cuando finaliza con él, vuelve con lo que estaba haciendo

## Observación

Las tres últimas líneas tienen la misma estructura que un CALL/RET.

## Aspectos a resolver

### Identificación

- ¿Quién ha generado la interrupción?
- Según quién haya sido, el sub-programa específico a ejecutar deberá ser diferente.

### Priorización

- Si dos interfaces interrumpen a la vez ¿cuál se atiende primero?
- Si la CPU estaba ejecutando el sub-programa de atención a un dispositivo y recibe una nueva interrupción de otro ¿a quién atiende ahora?

# Soluciones

- Existen muchas soluciones a los problemas de la identificación y priorización:
  - Múltiples líneas de interrupción
  - Consulta software
  - Vectorización
  - etc.

La CPU elemental usará las siguientes soluciones:

## Identificación

Vectorización con una única línea de interrupción para todos los interfaces.

## Priorización

Conexión en *daisy-chain*.

## Concepto de vectorización

- Cada interfaz tiene un **número de interrupción**. No puede haber dos interfaces con el mismo número.
- Cuando el interfaz interrumpe y la CPU acepta la interrupción, el interfaz envía su número de interrupción a la CPU.
- La CPU recibe el número de interrupción y lo usa para determinar qué sub-programa es el que hay que ejecutar.

El número de interrupción también se denomina **número de vector**.



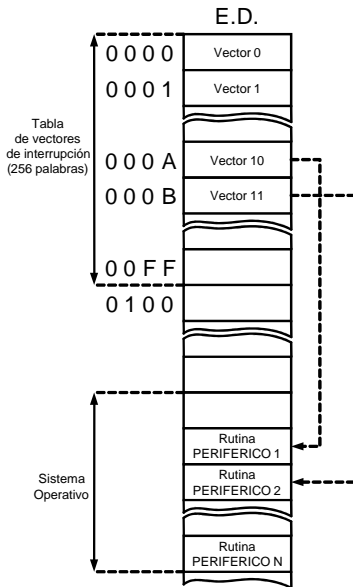
## Detalles

### ¿Cómo determina la CPU qué sub-programa ejecutar?

- Hay una serie de sub-programas preparados en la memoria, denominados **rutinas de servicio a las interrupciones**.
- Cada una estará en una posición de memoria diferente.
- En la memoria se almacena una tabla, que contiene las direcciones donde están las rutinas.
- Estas direcciones se denominan **vectores de interrupción**. La tabla se denomina **tabla de vectores de interrupción** (TVI).
- La CPU usa el número de interrupción como un índice dentro de la tabla, para encontrar la rutina.

# La TVI en el E.D. de la CPU elemental

La CPU elemental usa una TVI de 256 elementos, almacenada en las direcciones 0000h a 00FFh.



## Pasos de atención a una interrupción

### Fase A: **Aceptación de la interrupción**

- ① El periférico activa la línea INT.
- ② La CPU, en el último paso de ejecución de la instrucción, detecta que INT está activa.
- ③ La CPU almacena (en la pila) su estado actual (registro de estado + contador de programa)
- ④ La CPU “responde” activando la línea INTA
- ⑤ El periférico “responde” poniendo su número de vector en el bus de datos
- ⑥ La CPU recoge el número de vector y con él averigua la dirección de la rutina a ejecutar, consultándolo en la TVI.
- ⑦ La CPU carga esa dirección en PC.
- ⑧ La CPU desactiva el bit IF.
- ⑨ La ejecución continúa “normalmente”.

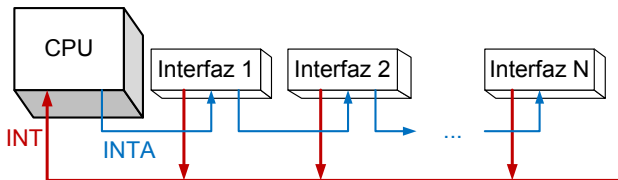
# Pasos de atención a una interrupción

## Fase B: **Ejecución de la rutina de servicio**

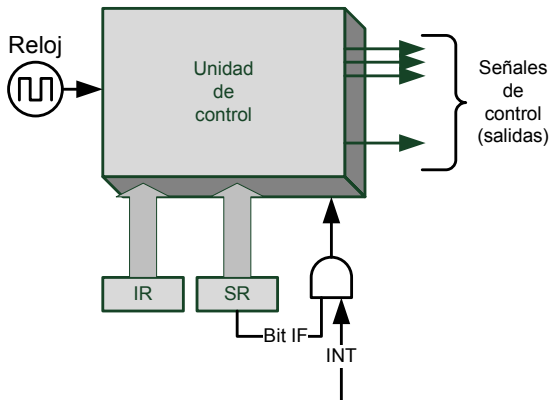
- ① La CPU, al continuar ejecutando “normalmente”, está ya ejecutando la rutina de servicio apropiada.
- ② En algún momento, la CPU debe alcanzar una instrucción llamada IRET
- ③ La instrucción IRET recupera (de la pila) el estado almacenado en el paso 3 de la fase de aceptación.
- ④ Como consecuencia de esa recuperación, la ejecución continuará donde fue interrumpida.

## Priorización por *daisy-chain*

- Sólo existe una línea INT para todos los interfaces. Cualquiera de ellos puede activarla.
- Sólo sale una línea INTA de la CPU, y llega a un interfaz.
- Este interfaz se la “pasa” al siguiente y así sucesivamente.
- El orden de conexión a INTA nos da la prioridad.



## Modificación a la Unidad de Control



Observar el papel del bit IF del registro de estado.

## Detección de la interrupción

- Cuando la UC encuentra la señal FIN, es cuando mira el estado de la línea INT.
- Observar que si el bit IF es 0, la UC siempre “verá” 0
- Aunque la interrupción puede tener lugar en cualquier momento, no se detecta hasta que se encuentra la señal FIN.

### Nuevo funcionamiento de la UC al alcanzar FIN

- Si  $IF=0$ , se va de nuevo al paso 1 (buscar siguiente instrucción).
- Si  $IF=1$  e  $INT=1$ , se va a una nueva serie de pasos especiales.

La misión de los nuevos pasos es generar todas las señales necesarias para almacenar el estado, solicitar el número de vector al periférico, consultar la TVI y saltar a la rutina apropiada.

## Los nuevos pasos

Paso	Señales activas
I-1	R7-IB, TMPE-SET, ADD, ALU-TMPS
I-2	SR-IB, IB-MDR
I-3	TMPS-IB, IB-R7, IB-MAR, WRITE
I-4	R7-IB, TMPE-SET, ADD, ALU-TMPS
I-5	PC-IB, IB-MDR
I-6	TMPS-IB, IB-R7, IB-MAR, WRITE, INTA
I-7	— Espera —
I-8	MDR-IB, IB-MAR, LEER
I-9	— Espera —
I-10	MDR-IB, IB-PC, FIN



## Tareas a realizar

### Escribir la rutina de servicio

- Se escribe como un procedimiento.
- Usa IRET en lugar de RET.
- No debe modificar **ningún** registro.

### Instalar la rutina de servicio

El “programa principal” debe:

- Modificar la TVI para que el elemento adecuado apunte a nuestra rutina.
- Permitir interrupciones (activar bit IF)

El resto es automático. La rutina de servicio se llama “sola” cuando ocurre la interrupción.

# Escritura de la rutina de servicio

## Esqueleto básico

```
.CODIGO
```

```
; —
```

```
; —
```

```
PROCEDIMIENTO rutina_servicio
```

```
    PUSH R0 ; Guardar TODOS los registros que vaya a usar
```

```
    PUSH R1
```

```
    . . .
```

```
; ———
```

```
; Instrucciones de la rutina
```

```
; ———
```

```
    . . .
```

```
    POP R1 ; Recuperar los registros guardados
```

```
    POP R0
```

```
    IRET
```

```
FINP
```

```
; —
```

## Instalación de la rutina de servicio

Hay que:

- Saber el número de interrupción al que atiende la rutina ( $N$ )
- Averiguar en qué dirección de memoria está almacenada la rutina ( $D$ )
- Modificar la Tabla de Vectores de Interrupción:
  - Desactivar bit IF, por si acaso (instrucción CLI)
  - Escribir en el elemento  $N$  de la TVI, el valor  $D$
  - Activar bit IF (instrucción STI)

Esto puede hacerse fácilmente desde un programa si conocemos  $N$ .  
Por ejemplo, sea  $N = 3$ .

# Instalación de la rutina de servicio

## Código de instalación

```
; Pongo en R0 el valor de N (número de interrupción)  
MOVH R0, 0  
MOVL R0, 3  
; y en R1 el valor de D (dirección de la rutina)  
MOVH R1, BYTEALTO DIRECCION rutina_servicio  
MOVL R1, BYTEBAJO DIRECCION rutina_servicio  
; Vamos a modificar la TVI  
; Desactivar bit IF  
CLI  
; Escribir en la TVI  
MOV [R0], R1  
; Reactivar bit IF  
STI
```

## Ejemplo completo

Asumiendo que hay un interfaz de vídeo mapeado a partir de FC00h, escribir una rutina de servicio que:

- Lee el primer carácter de la memoria de vídeo.
- Cambia su color a rojo sobre blanco.
- Vuelve a ponerlo en la memoria de vídeo.

Y un programa principal que:

- Instala dicha rutina en el vector número 2.
- Entra en un bucle infinito en el que imprime los números del 9 al 1 en la esquina superior izquierda de la pantalla, en blanco sobre negro.

Puede probarse el funcionamiento con el dispositivo “Luces”.

# Solución

```
ORIGEN 200h
INICIO ini
.PILA 30h
.DATOS
; No hay
.CODIGO
PROCEDIMIENTO cambia_fondo
    PUSH R0
    PUSH R1
    MOVH R0, 0FCh
    MOVL R0, 00h
    MOV R1, [R0] ; Leer letra
    MOVH R1, 3Ch ; cambiar color
    MOV [R0], R1 ; escribirla otra vez
    POP R1
    POP R2
    IRET
FINP
```

*sigue ►►*

# Solución

➤➤ *sigue*

*ini:*

*; Instalar la rutina*

MOVL R0, 2

MOVH R0, 0

MOVL R1, BYTEBAJO DIRECCION *cambia\_fondo*

MOVH R1, BYTEALTO DIRECCION *cambia\_fondo*

CLI

MOV [R0], R1 *; Escribir TVI*

STI

*; Tras la instalación, empieza el programa*

*sigue ➤➤*

# Solución

➤➤ *sigue*

*; Inicializamos variables:*

MOVL R2, 00h

MOVH R2, 0FCh ; R2 apunta a la esquina de la pantalla

MOVL R1, '0' ; ASCII del 0, para saber cuándo acabamos

MOVL R1, 7 ; color blanco sobre negro

otra\_vez:

MOVL R0, '9' ; ASCII del 9

MOVH R0, 7 ; color blanco sobre negro

bucle:

MOV [R2], R0 ; Escribirlo en pantalla

DEC R0 ; pasar al siguiente dígito en la cuenta atrás

COMP R0, R1 ; ¿llegamos al cero?

BRZ otra\_vez ; en ese caso volver a ponerlo a '9'

JMP bucle ; si no, repetir bucle

FIN