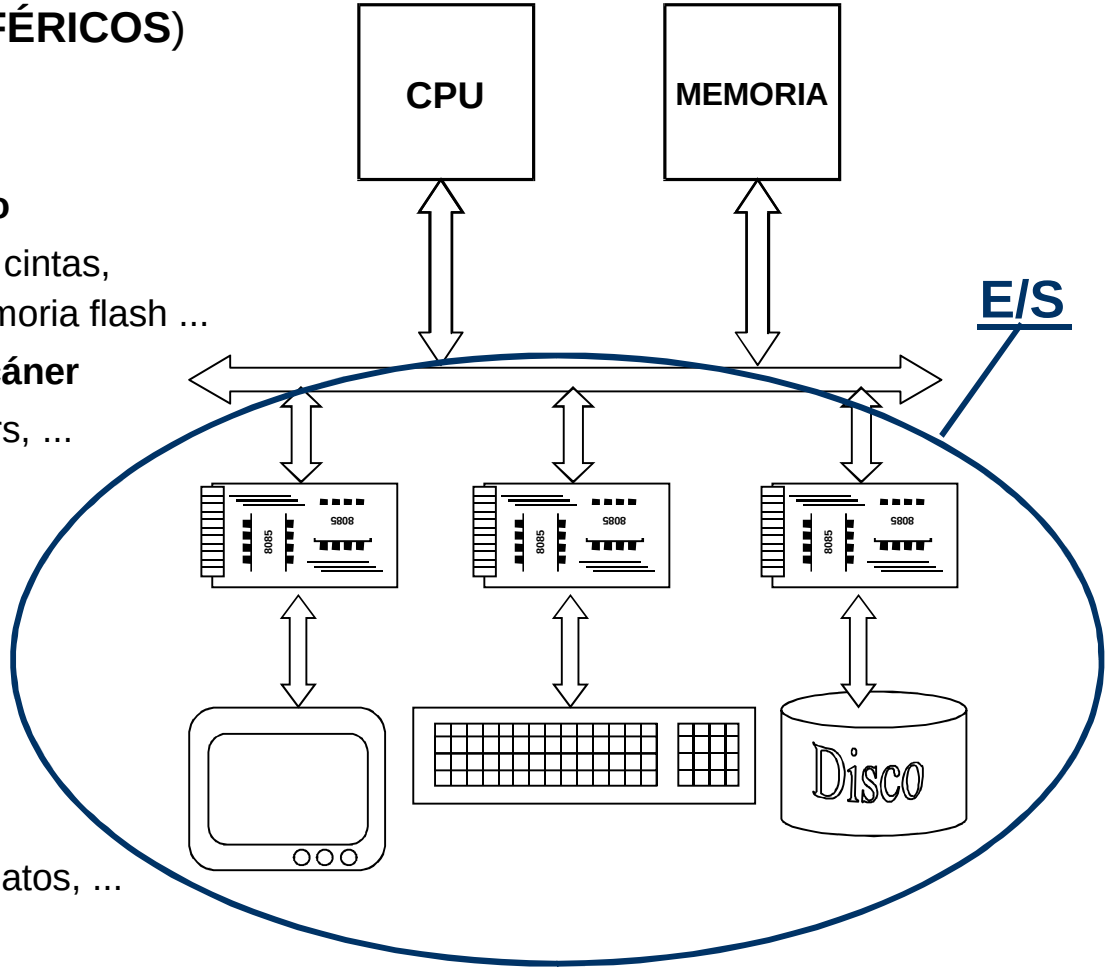
Organización del sistema de entrada/salida

1. Organización del subsistema de E/S
2. Mecanismos básicos de E/S
3. Gestión de interrupciones
4. DMA

La E/S permite al computador interactuar con el “mundo exterior”

- Dispositivos típicos de E/S (**PERIFÉRICOS**)
  - **Dispositivos de E/S básica**
    - teclado, ratón, pantalla
  - **Dispositivos de almacenamiento**
    - discos, disquetes, CD-ROM, cintas, discos magneto-ópticos, memoria flash ...
  - **Dispositivos de impresión y escáner**
    - impresoras, plotters, scanners, ...
  - **Dispositivos de comunicación**
    - redes, módems, ...
  - **Dispositivos multimedia**
    - audio, video, ...
  - **Dispositivos de automatización y control**
    - sensores, alarmas, sistemas de adquisición de datos, ...

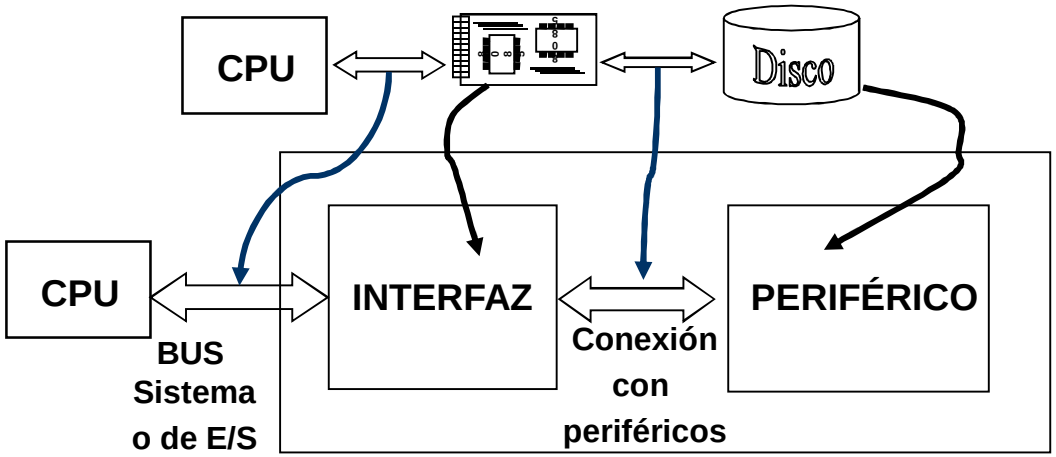


Funciones básicas del subsistema de E/S

- **Direccionamiento**
  - Necesario seleccionar el dispositivo de E/S con el que se realiza la transferencia
- **Transferencia de datos** entre el computador y el periférico
  - Tipos de transferencia
    - Lectura: computador ← periférico
    - Escritura: computador → periférico
  - Puede requerir ciertas conversiones de formato de los datos
    - Conversión de niveles eléctricos
      - TTL: 1 → V > 2,0 Volt; 0 → V < 0,8 Volt
      - RS-232-C: 1 → V < -3.0 Volt; 0 → V > +3.0 Volt
    - Conversión del tipo de codificación
      - Caracteres (ASCII, EBCDIC)
      - Enteros (magnitud y signo, C'1, C'2, ...)
      - Reales (punto fijo punto flotante, simple precisión, doble precisión, ...)
    - Conversión serie-paralelo / paralelo-serie
    - Conversión digital-analógico / analógico-digital
- **Sincronización y control de la transferencia**
  - Necesario un mecanismo de sincronización de la transferencia
    - El computador debe conocer
      - Si el periférico está preparado para enviar o recibir datos
      - Si el periférico ha terminado de realizar una transferencia y puede iniciar una nueva
  - No confundir con la sincronización elemental a nivel de transferencias de palabras a través del bus

El interfaz de E/S

- Conecta a los dispositivos con el computador
  - Interfaz = Controlador = Adaptador = Tarjeta de E/S



Ejemplo

Ordenes CPU → Interfaz

Leer N bytes a partir de
Superficie S
Cilindro C
Sector T

Ordenes Interfaz → periférico

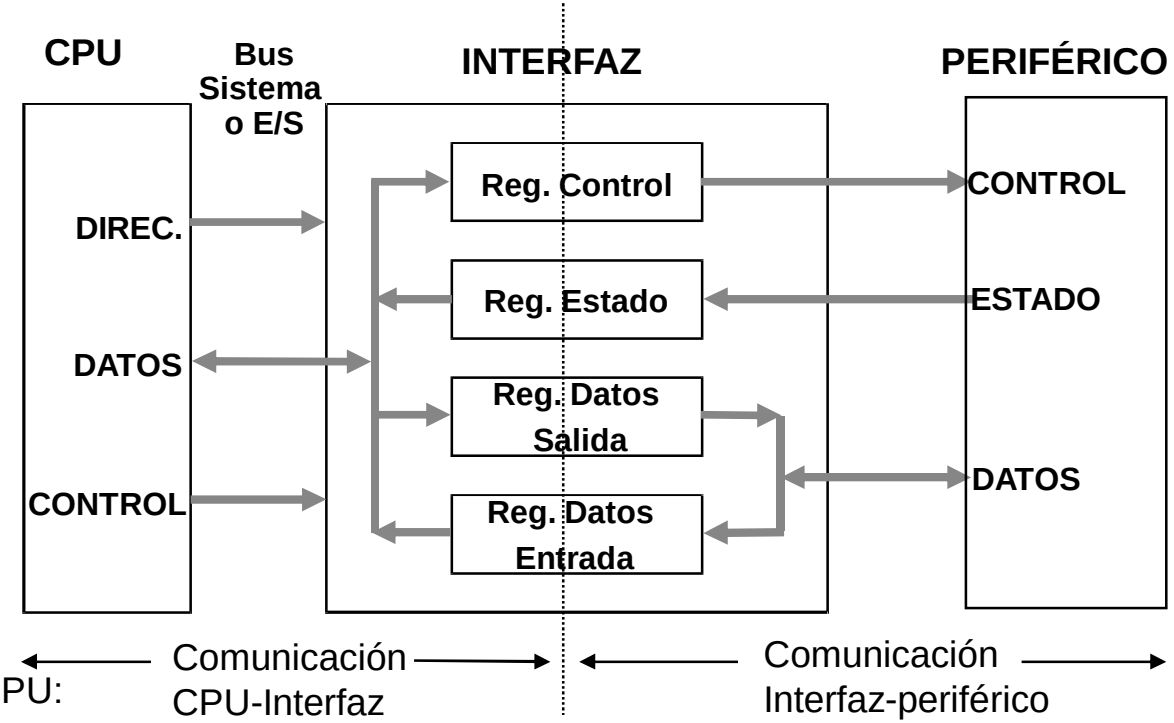
Posicionar cabezales en cilindro C
Posicionar cabezales en sector T
Seleccionar cabezal de superficie S
Leer N bytes
Retirar cabezales

Funciones del interfaz de E/S

- Interpretar las órdenes que recibe de la CPU y transmitirlas al periférico
- Controlar la transferencia de datos entre la CPU y el periférico
  - Conversión de formatos
  - Adaptar la diferencia de velocidades entre CPU y periférico (mediante buffers de almacenamiento)
- Informar a la CPU del estado del periférico

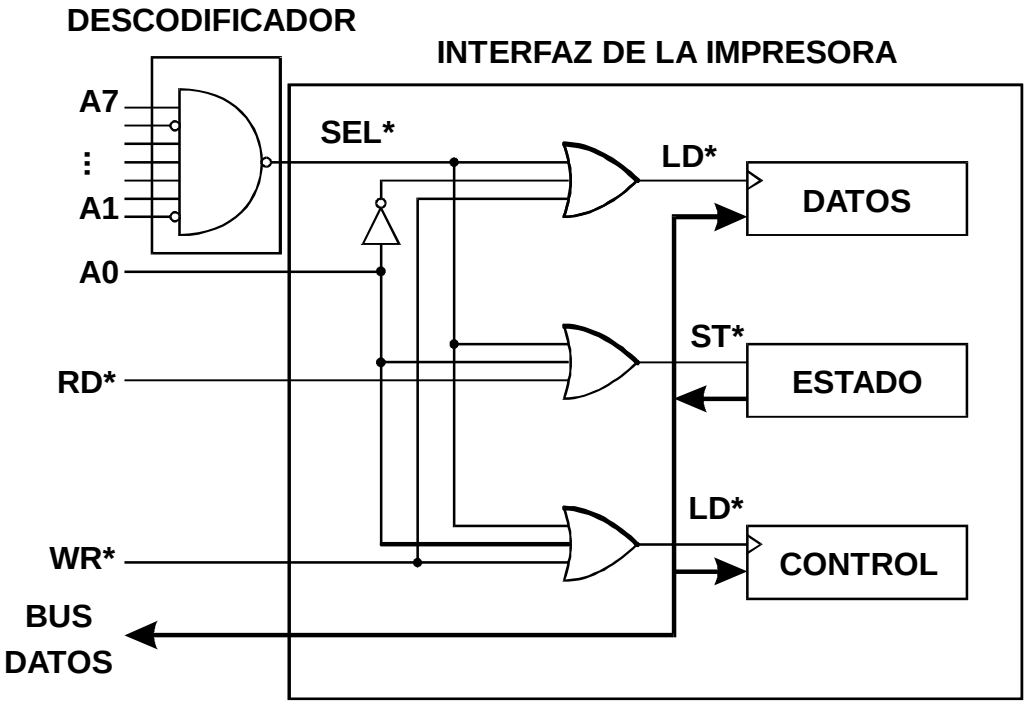
Estructura del interfaz de E/S

- La comunicación se realiza a través de los registros del interfaz:



- **Registro de datos de salida**
  - Almacena los datos enviados al periférico
- **Registro de datos de entrada**
  - Almacena los datos enviados por el periférico
- **Registro de estado**
  - Proporciona información a la CPU:
    - Dispositivo preparado/no preparado
    - Reg. datos lleno/vacío
    - Transferencia finalizada/no finalizada, etc.
- **Registro de control**
  - Almacena las órdenes de la CPU
    - Leer/escribir N bytes en cilindro C, pista P, sector S (para discos)
    - Rebobinar / avanzar / leer N bytes (para cintas)
    - Imprimir carácter / saltar de línea / saltar de página (para impresoras), etc.

Ejemplo de conexión de un interfaz de E/S al bus



- Enviar un carácter almacenado en el registro R1 a la impresora
  - MOVE R1, $BD
  - OUT R1, $BD
- Enviar una orden almacenada en el registro R2 a la impresora
  - MOVE R2, $BC
  - OUT R2,$BC
- Leer estado de la impresora y almacenarlo en el registro R3
  - MOVE $BC, R3
  - IN $BC, R3

Dirección Registro Estado/Control: 10111100 ($BC)
Dirección Registro Datos: 10111101 ($BD)

Nota: los registros del interfaz de E/S también se llaman ***puertos de E/S***

Ejemplo Placa ARM: temporizadores

- Son contadores descendentes con cierta lógica adicional
- Incluyen **5 registros de control** mapeados en memoria que permiten al procesador:
  - Configurar la velocidad a la que cuentan
  - Especificar dónde comienza la cuenta
  - Si se utilizan para generar una onda cuadrada definir cuando deben generar salida 0 y cuando 1
  - Indicar si deben contar una vez o repetir la cuenta indefinidamente
  - Indicar si deben avisar al procesador cada vez que lleguen a cero
  - Parar la cuenta
  - Reiniciar la cuenta
  - Indicar si deben avisar al DMA
- **Un registro de salida** que permite ver el valor actual de la cuenta

Alternativas de diseño: E/S aislada o localizada en memoria

E/S aislada

- ***La E/S y la memoria utilizan un espacio de direcciones distinto***
  - El conjunto de direcciones de que utiliza la memoria y el que utiliza la E/S son independientes
- ***Existen instrucciones específicas de E/S***
  - IN dir_E/S, Ri (CPU ← Periférico)
  - OUT Ri, dir_E/S (Periférico ← CPU)
- ***El bus dispone de líneas de control específicas*** (MEM/IO*) para indicar si se trata de una operación con memoria o una operación de E/S
  - Si MEM/IO* = 1
    - Operación con memoria (MOVE, LOAD, STORE) ⇒ La dirección del bus corresponde a una posición de memoria
  - Si MEM/IO* = 0
    - Operación de E/S (IN, OUT) ⇒ La dirección del bus corresponde a un puerto de E/S
  - ***Ejemplos***
    - i8086 y demás computadores de la familia intel x86

E/S localizada en memoria

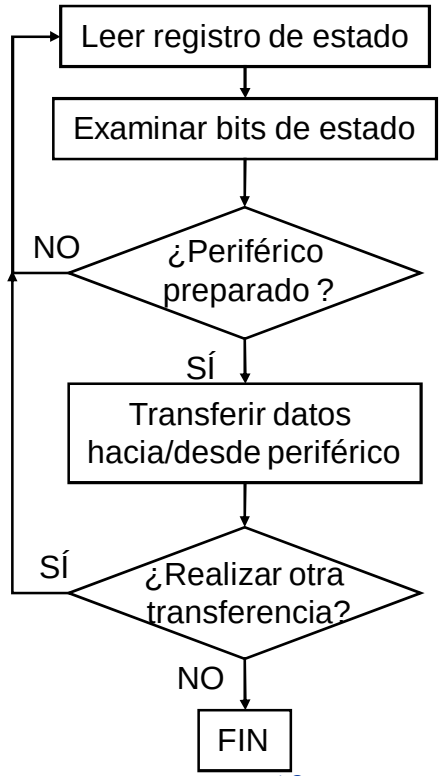
- ***La E/S y la memoria comparten el mismo espacio de direcciones***
- ***No se requieren instrucciones específicas de E/S***
  - Las mismas instrucciones que se utilizan para movimiento de datos con memoria (MOVE) pueden utilizarse para realizar operaciones de E/S
    - MOVE dir_E/S, Ri (CPU ← Periférico)
    - MOVE Ri, dir_E/S (Periférico ← CPU)
- ***En el bus no existe una línea especial*** para distinguir operaciones con memoria de operaciones de E/S
  - Un puerto de E/S no puede tener asignada la misma dirección que una posición de memoria válida
  - Normalmente se asigna a los dispositivos de E/S una **porción contigua del espacio de direcciones** que no se utiliza para la memoria
- ***Ventajas de la E/S localizada en memoria***
  - Es más flexible que la E/S aislada ya que permite realizar distintos tipos de operaciones sobre los puertos de E/S (aritméticas, lógicas, manipulación de bits, etc.) y no sólo de movimiento de datos
- ***Ejemplo***
  - **ARM**

Sincronización de la E/S

- Cuando la CPU quiere enviar/recibir datos a/desde un periférico tiene asegurarse de que el dispositivo está preparado para realizar la transferencia
- Existen dos mecanismos básicos de sincronización de la E/S
  - E/S programada con espera de respuesta
  - E/S por interrupciones

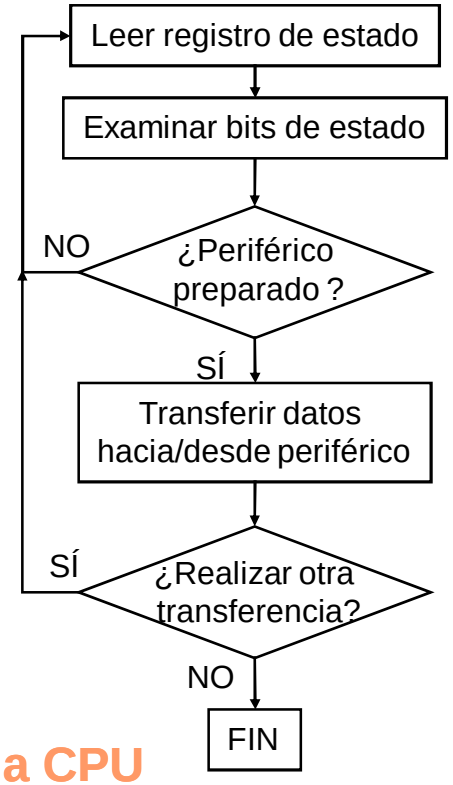
E/S programada con espera de respuesta

- Cada vez que la CPU quiere realizar una transferencia entra en un bucle hasta que éste está preparado para realizar la transferencia
- **Problema: ¡Es muy ineficiente!**



Problemas de la E/S con espera de respuesta

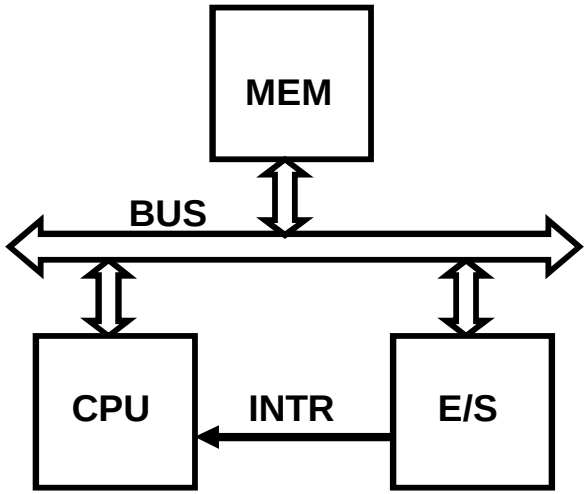
- La CPU no hace trabajo útil durante el bucle de espera
  - Con dispositivos lentos el bucle podría repetirse miles/millones de veces
- La dinámica del programa se detiene durante la operación de E/S
  - Ejemplo: en un vídeo-juego no se puede detener la dinámica del juego a espera que el usuario puse una tecla o mueva el *jostick*
- Dificultades para atender a varios periféricos
  - Mientras se espera a que un periférico esté listo para transmitir, no se puede atender a otro



Solución: Los periféricos deben poder comunicarse con la CPU

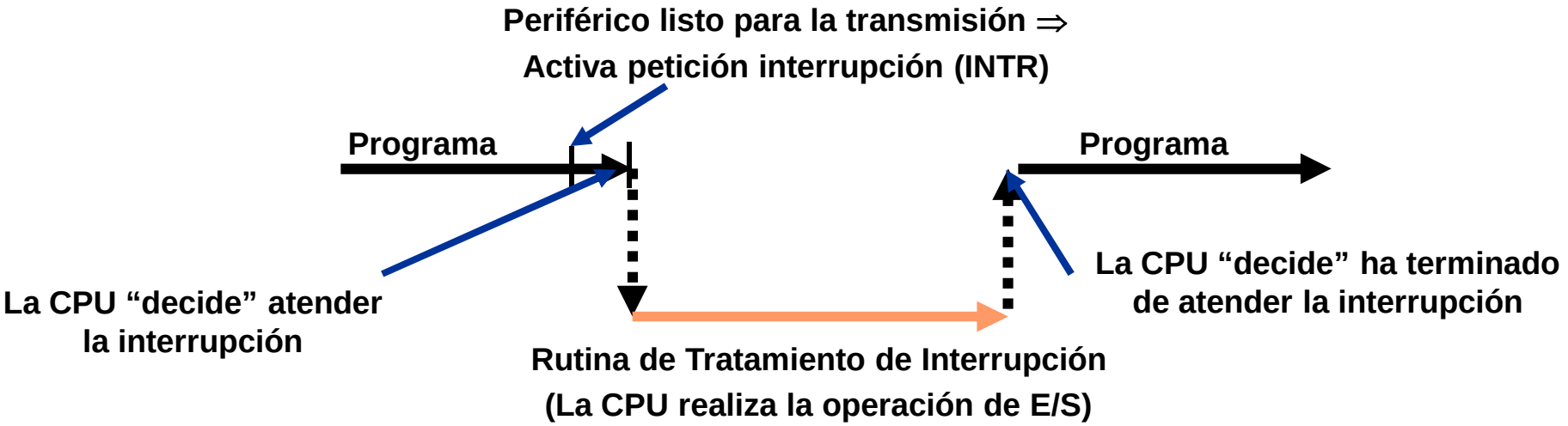
E/S por interrupciones

- No existe bucle de espera
- Cuándo la CPU da una orden puede realizar otras tareas mientras espera
- Cuando un periférico está listo para transmitir se lo indica a la CPU activando una línea especial del bus de control denominada **LÍNEA DE PETICIÓN INTERRUPCIÓN**
  - La CPU decide qué periféricos tienen capacidad para interrumpir



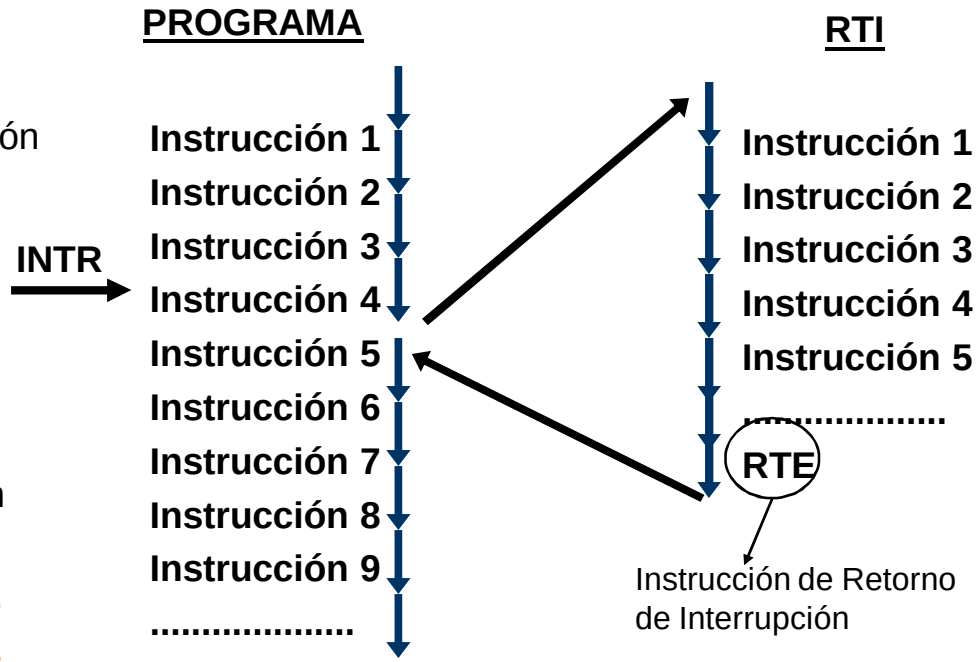
E/S por interrupciones

- Cuándo la CPU recibe una interrupción puede elegir si quiere atenderla o no en ese instante
- Para atender una interrupción la CPU ejecuta su **rutina de tratamiento de interrupciones** (RTI)
  - Cuando la CPU recibe una señal de petición de interrupción salta a ejecutar una **RTI**
  - La RTI se encarga de atender al periférico que interrumpió y realizar la operación de E/S



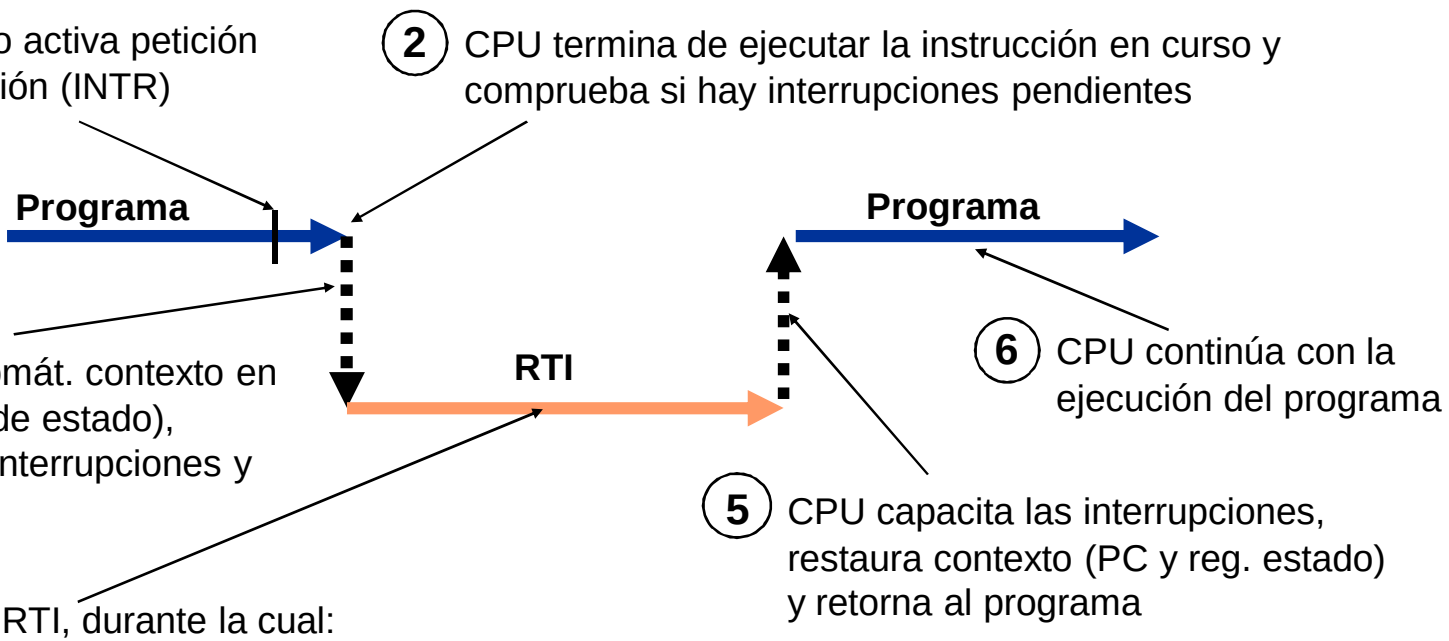
E/S por interrupciones

- **Analogías entre una subrutina y una RTI**
  - Se rompe la secuencia normal de ejecución
  - Cuando terminan de ejecutarse se debe retornar al punto de ruptura
    - Debemos **guardar el PC** en ambos casos
- **Diferencias entre una subrutina y una RTI**
  - En una subrutina el programador sabe en qué punto exacto se rompe la secuencia
  - **Una RTI puede ejecutarse en cualquier momento, sin control del programador**



  - Necesario **guardar el registro de estado** y restaurarlo al retornar de la RTI
    - Normalmente se realiza automáticamente
  - Necesario **guardar los registros** que utiliza la RTI en la pila y restaurarlos al retornar de la RTI
    - Normalmente hay que realizarlo de forma manual

Secuencia de eventos en el tratamiento de una interrupción



4 CPU ejecuta la RTI, durante la cual:

- Informa al periférico que se ha reconocido su interrupción (por software o por hardware)
  - ⇒ El periférico desactiva INTR
- Salva en pila todos los registros de datos y/o direcciones utilizados por la RTI (manual)
- Realiza la operación de E/S con el periférico
- Restaura los registros de datos/direcciones
- Ejecuta la instrucción de retorno de interrupción (RTE)

Cuestiones planteadas

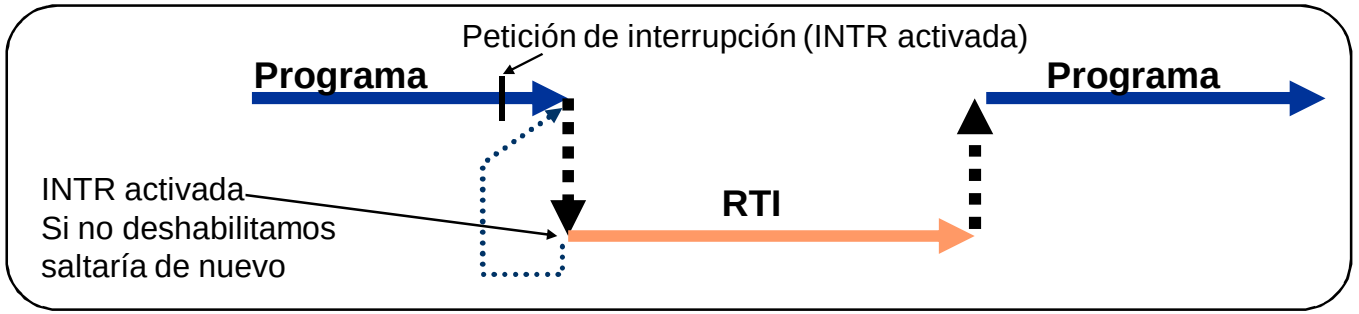
- ¿Cuándo comprueba la CPU si hay interrupciones pendientes?
- ¿Por qué es necesario deshabilitar las interrupciones?
- ¿Cómo se informa al periférico que se ha reconocido su interrupción?
  - ⇒ *Identificación de la fuente de interrupción*
- ¿Qué ocurre si se produce una segunda interrupción durante la ejecución de la RTI?
  - ⇒ *Interrupciones multinivel y anidamiento de interrupciones*

Comprobación de peticiones de interrupción pendientes

- La CPU comprueba si hay interrupciones pendientes (línea INTR activada) al final de la ejecución de cada instrucción
  - **Motivo:**
    - Sólo es necesario guardar el PC, el reg. de estado y los registros accesibles por programa (registros de datos y/o direcciones)
    - Si se interrumpiese una instrucción en mitad de la ejecución sería necesario guardar el valor de todos los registros internos de la CPU
      - Reg. de instrucción, registros de dirección de datos, registros de datos de memoria, etc.
  - **¿Qué ocurre con los procesadores segmentados?**

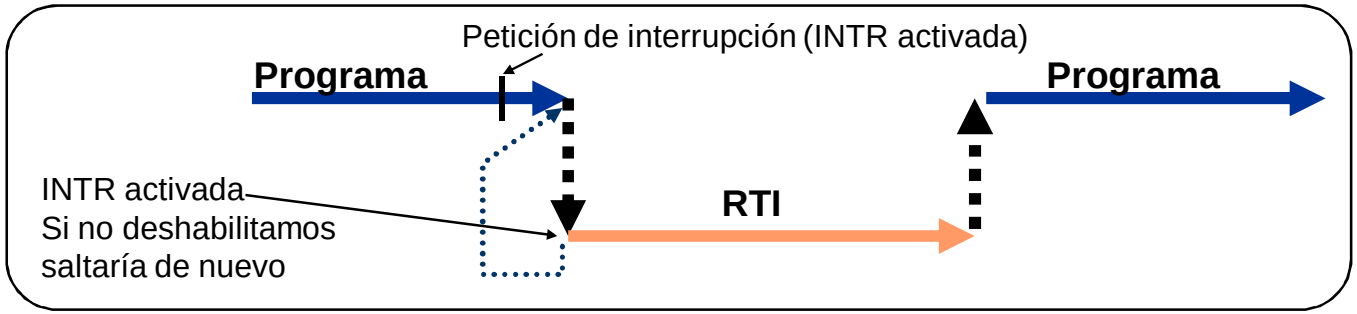
Inhibición o deshabilitación de las interrupciones

- Antes de saltar a la RTI es necesario inhibir o deshabilitar las interrupciones
  - **Motivo:** Si no se inhiben la CPU puede entrar en un bucle infinito
    - Cuando se entra en la RTI el periférico todavía no ha desactivado su petición
    - Si las interrupciones están capacitadas ⇒ la CPU detecta una interrupción pendiente y vuelve saltar a la RTI una y otra vez
    - Antes de finalizar la RTI hay que asegurarse de que el periférico ha desactivado la línea de petición INTR
      - **Por software:** accediendo al registro de estado o de datos del interfaz
      - **Por hardware:** activando una señal de reconocimiento de interrupción (INTA)



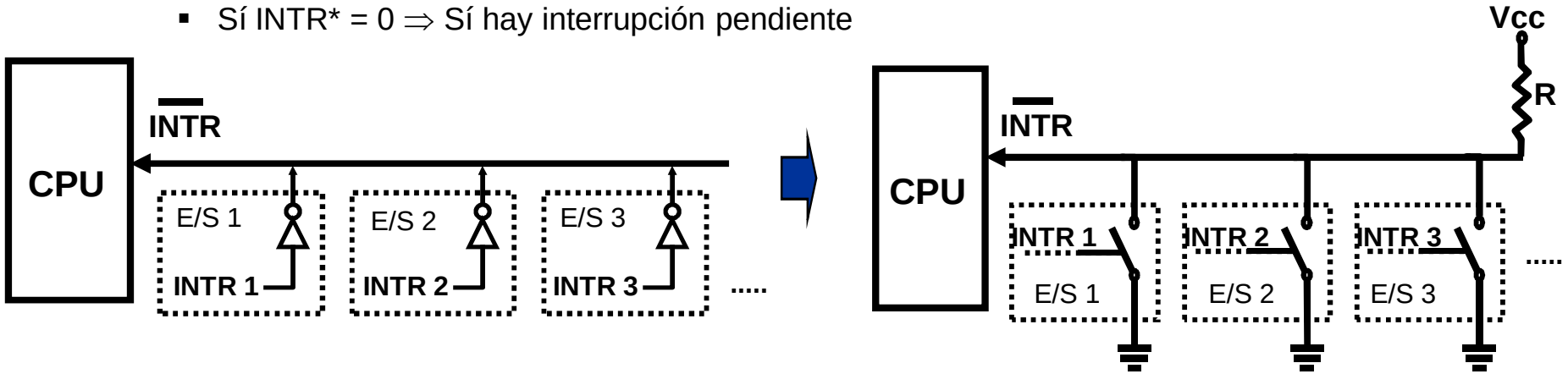
Inhibición o deshabilitación de las interrupciones

- Alternativas
  - **deshabilitación global**
    - Se inhiben todas las interrupciones ⇒ ningún otro periférico podrá interrumpir durante la ejecución de la RTI
  - **deshabilitación o enmascaramiento selectivo**
    - Cuando hay varios niveles de interrupción se pueden deshabilitar las interrupciones por el nivel que interrumpe, pero no necesariamente por el resto de niveles
    - Véase interrupciones multinivel y anidamiento de interrupciones



Identificación de la fuente de interrupción

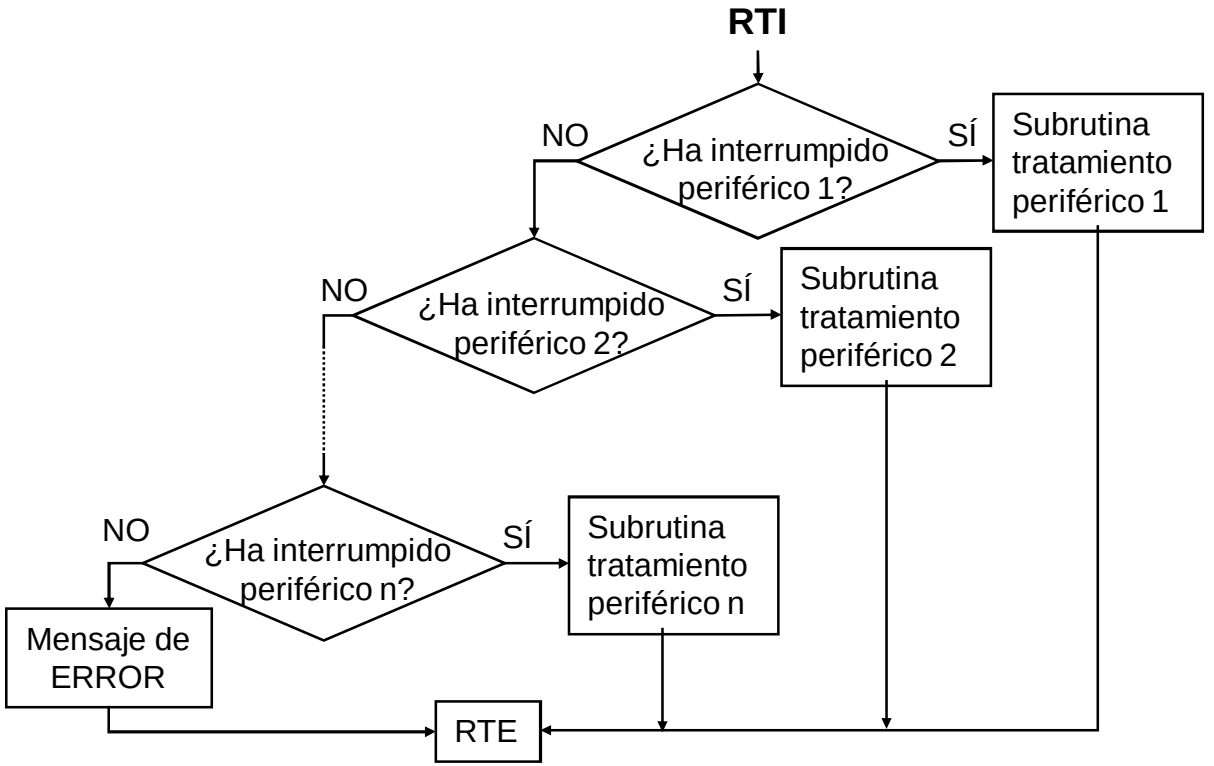
- Normalmente hay más periféricos que líneas de interrupción
- En una misma línea de interrupción es posible conectar varios periféricos
  - Normalmente se utiliza lógica negativa (INTR*) y cableada (en colector abierto, *"open collector"*)
    - Sí INTR* = 1 ⇒ No hay interrupción pendiente
    - Sí INTR* = 0 ⇒ Sí hay interrupción pendiente



- Cuando existen varias fuentes de interrupción es necesario un mecanismo para identificar al periférico que interrumpió y ejecutar la RTI adecuada para atender a ese periférico particular
  - **Identificación software:** por encuesta (polling)
  - **Identificación hardware:** por vectores

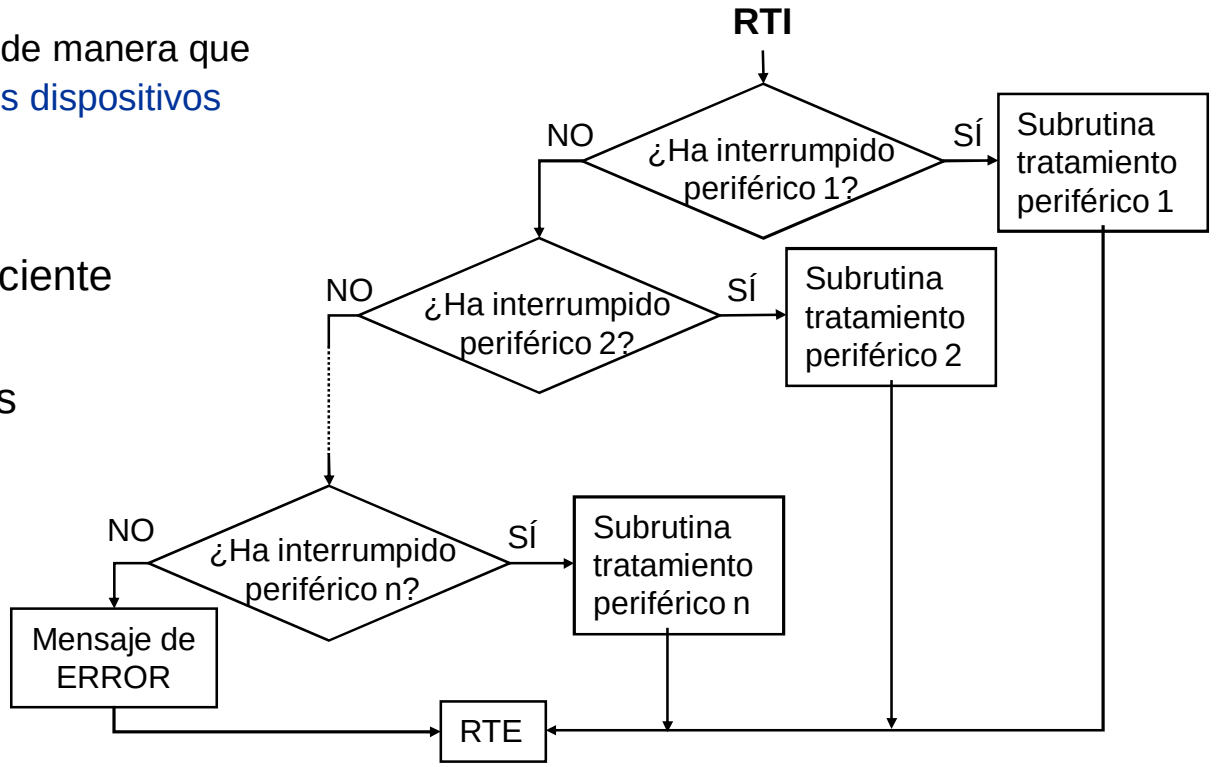
Identificación software por encuesta (polling)

- La RTI examina los registros de estado de cada periférico hasta hallar el que tiene activado su bit de petición de interrupción
  - Una vez detectado el periférico que interrumpió se ejecuta una subrutina particular del periférico en cuestión
  - Durante la ejecución de esa rutina se debe desactivar el bit de petición de interrupción del periférico



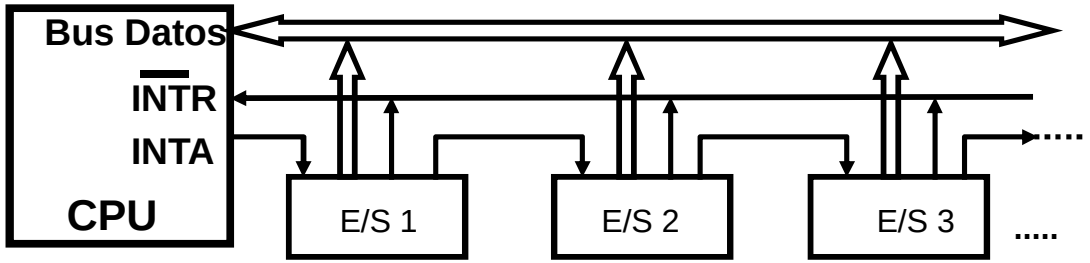
Identificación software por encuesta (polling)

- **Asignación de prioridades**
  - El método de encuesta introduce un mecanismo de prioridades software
    - En caso de peticiones simultáneas se atiende por orden de encuesta
    - La RTI se suele diseñar de manera que se pregunta primero a los dispositivos más prioritarios
- **Rendimiento**
  - No es un sistema muy eficiente
  - Se desperdicia tiempo consultando a dispositivos que no han solicitado servicio



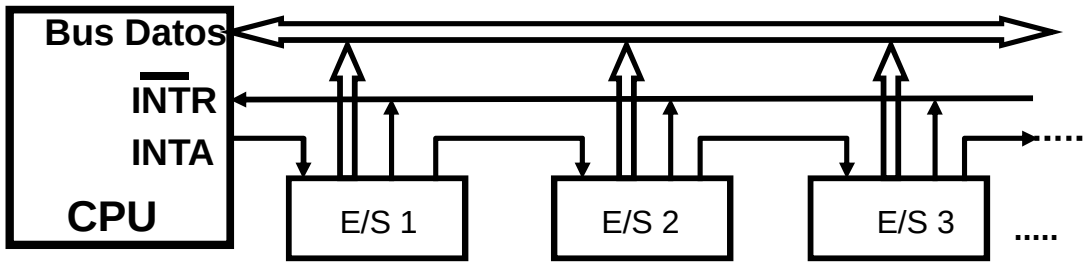
Identificación hardware: interrupciones vectorizadas

- Los periféricos son capaces de identificarse ellos mismos enviando un código (nº de vector) a la CPU.
  - Cuando el periférico recibe una señal de reconocimiento de interrupción **INTA** (*"Interruption Ack."*) envía el nº de vector a través del bus de datos
  - A partir del **nº de vector** la CPU calcula la dirección de comienzo de la RTI



Identificación hardware: interrupciones vectorizadas

- Secuencia de eventos en el tratamiento de una interrupción vectorizada
  1. El periférico activa la señal de interrupción **(INTR*=0)**
  2. La CPU activa la señal de confirmación de interrupción **(INTA=1)** que se conecta a los dispositivos de forma encadenada (daisy-chain)
  3. Un periférico que no ha interrumpido, cuando recibe la señal INTA, la propaga al siguiente
  4. Cuando el periférico que interrumpió recibe la señal INTA vuelca su número de vector sobre el bus de datos y desactiva la señal de petición de interrupción. Este periférico no propaga INTA
  5. La CPU calcula la dirección de comienzo de la RTI a partir del nº de vector
  6. La CPU salva el contexto en pila (CPU y reg. de estado) y salta a la RTI
  7. Se guardan los registros accesibles por programa, se ejecuta la operación de E/S y se retorna de la interrupción al programa principal restaurando previamente todo el contexto

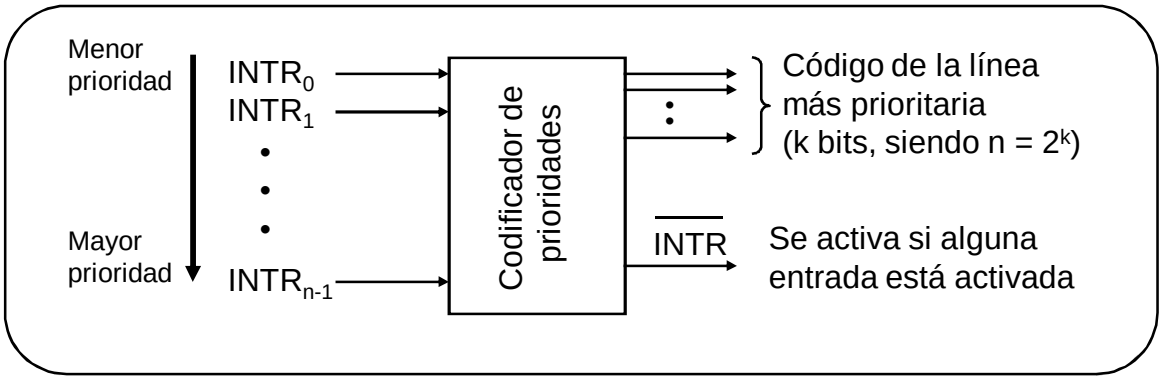


Identificación hardware: interrupciones vectorizadas

- **Ventajas**
  - La transmisión de INTA es totalmente hardware ⇒ es mucho más rápido que el método de encuesta
- **Desventajas**
  - El nº de dispositivos que se pueden identificar con este método depende del nº de bits que utilicemos para el nº vector
    - Ejemplo: con un nº de vector de 4 bits podemos identificar 16 dispositivos
- ***Solución:*** pueden utilizarse una estrategia mixta
  - Un mismo nº de vector puede utilizarse para identificar a un grupo de varios dispositivos
  - Cuando la CPU recibe un nº de vector de grupo, la RTI debe identificar al dispositivo particular de ese grupo mediante encuesta

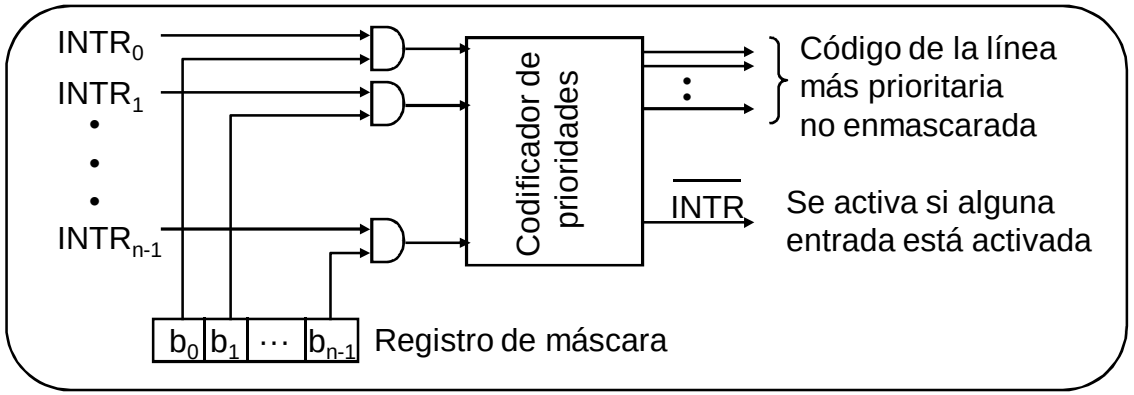
Sistemas con varios niveles de interrupciones

- **Interrupciones multinivel**
  - Existen varias líneas o niveles de petición de interrupción
  - Cada nivel tiene asignado una prioridad distinta
  - A cada línea de interrupción se pueden conectar uno o varios dispositivos
- **Resolución de conflictos:**
  - Peticiones simultáneas por la misma línea
    - Se resuelve con alguno de los mecanismos estudiados anteriormente
      - Mediante encuesta (software)
      - Mediante vectores (hardware)
  - Peticiones simultáneas por líneas distintas
    - Se suele resolver mediante un **codificador de prioridades** ⇒ Se atiende a la línea más prioritaria



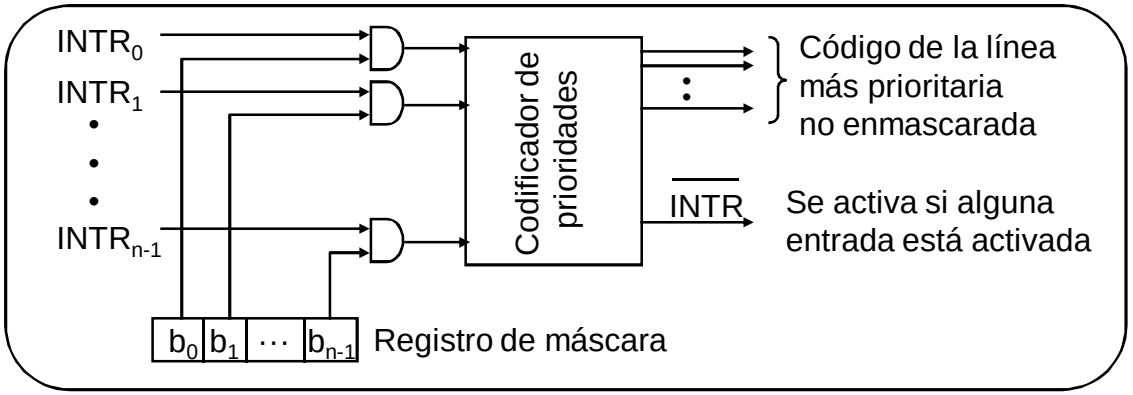
Sistemas con varios niveles de interrupciones

- **Enmascaramiento selectivo:**
  - Los sistemas de interrupciones multinivel permiten enmascarar o deshabilitar selectivamente las interrupciones por determinados niveles
  - Para ello se utiliza un ***registro de máscara***
    - 1 bit de máscara b_k por nivel
      - Si $b_k = 1 \rightarrow$ Nivel $INTR_k$ habilitado
      - Si $b_k = 0 \rightarrow$ Nivel $INTR_k$ deshabilitado o enmascarado



Sistemas con varios niveles de interrupciones

- **Anidamiento de interrupciones**
  - En general, en los sistemas de interrupciones multinivel se permite el anidamiento de interrupciones
    - Mientras se ejecuta la RTI de un determinado nivel se inhiben las interrupciones por el mismo nivel o inferiores, pero se pueden atender petición de interrupción de mayor nivel
  - El anidamiento se controla mediante el registro de máscara
    - Cuando se produce una interrupción de prioridad P_k se enmascaran todas las interrupciones de prioridad $P \leq P_k$



3. Ejemplo de anidamiento de interrupciones

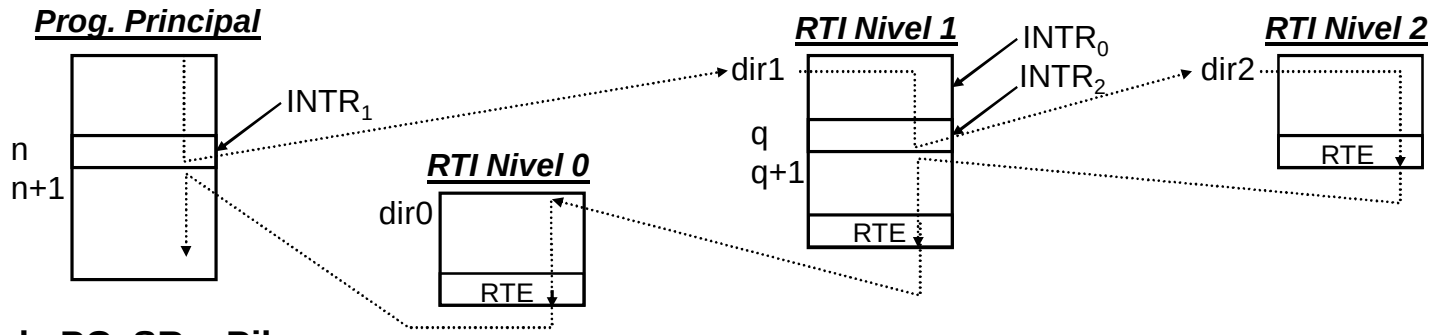
Sistema con 3 niveles de interrupción: $INTR_0$, $INTR_1$, $INTR_2$ $(INTR_0 < INTR_1 < INTR_2)$

Registro de estado

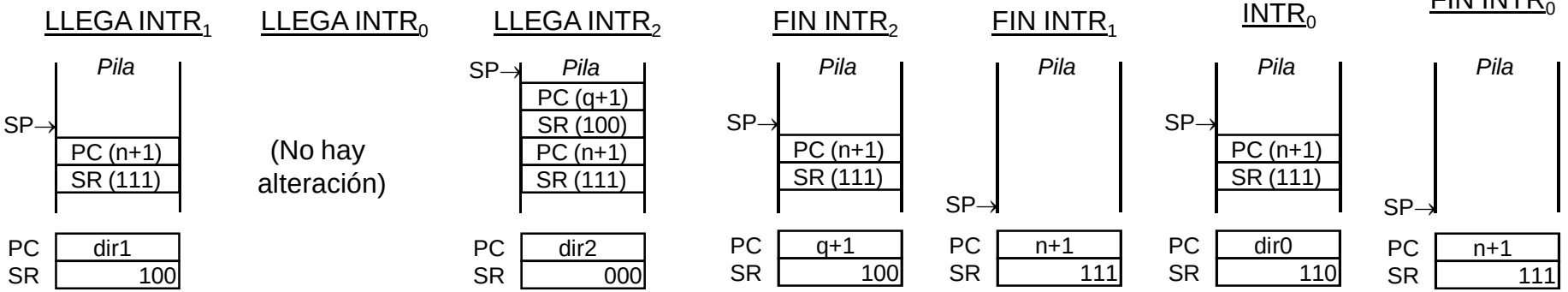
SR: [ ] [b_2] [b_1] [b_0] → Bits de máscara

- Si $b_k = 1 \rightarrow$ Nivel $INTR_k$ capacitado
- Si $b_k = 0 \rightarrow$ Nivel $INTR_k$ enmascarado

Supongamos que se producen 3 peticiones de interrupción en el orden $INTR_1$ - $INTR_0$ - $INTR_2$



Evolución de PC, SR y Pila



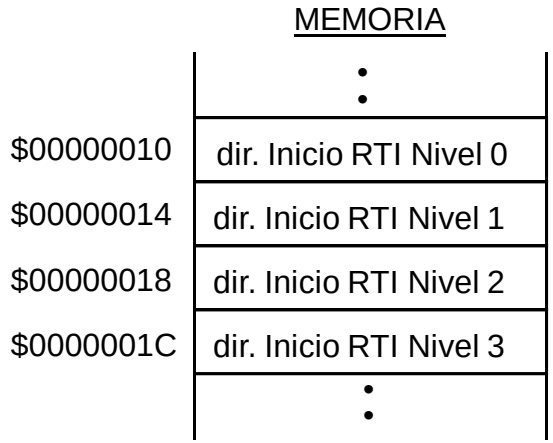
Interrupciones autovectorizadas

- La mayoría de sistemas de interrupciones multinivel asignan un vector de interrupción por defecto a cada nivel de interrupción denominada **autovector**
  - El autovector almacena la **dirección de comienzo de la RTI asignada por defecto** a ese nivel
  - Los autovectores se utilizan para periféricos que no son capaces de generar su propio nº de vector
  - El autovector activara una RTI por defecto
  - La RTI por defecto utilizará un **mecanismo de encuesta** para identificar al periférico que interrumpió

Ejemplo

Computador con 4 niveles de interrupción: (direcciones de 32 bits)

- $INTR_0 \rightarrow$ autovector $00000010
- $INTR_1 \rightarrow$ autovector $00000014
- $INTR_2 \rightarrow$ autovector $00000018
- $INTR_3 \rightarrow$ autovector $0000001C



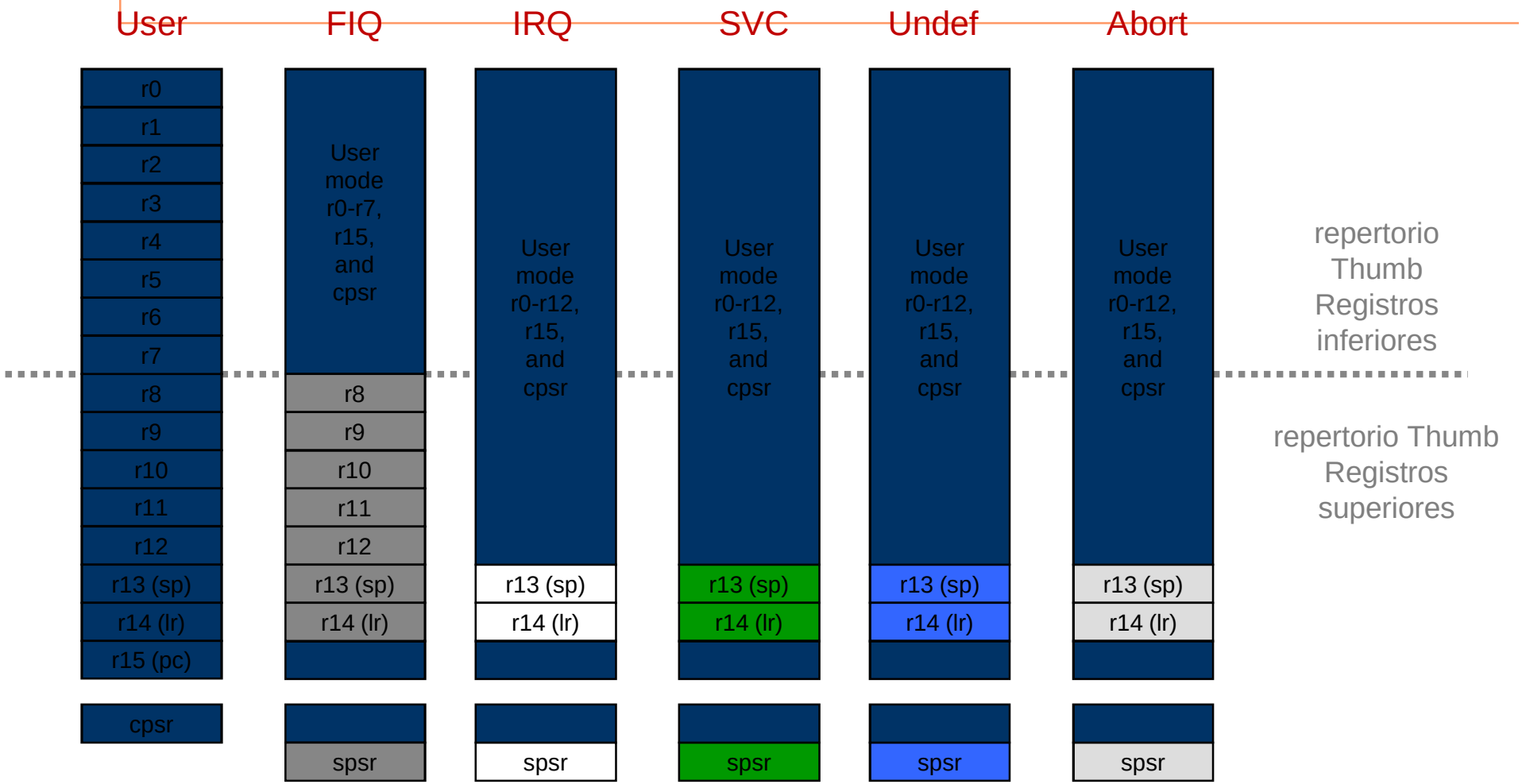
Recordatorio: excepciones en ARM

- Los procesadores ARM tienen 7 modos de operación:
  - **User** (usr): estado normal de ejecución
  - **FIQ** : manejo de interrupciones rápidas para transferencias de datos
  - **IRQ** : manejo de interrupciones de propósito general o lentas
  - **Supervisor** (svc): modo protegido para el sistema operativo
  - **Abort** (abt): usado para gestionar las fallos de acceso a memoria (prefetching o accesos convencionales)
  - **Undef** (und): manejo de excepciones por códigos de operación no definidas
  - **System** : modo privilegiado para el sistema operativo, usando los mismos registros que en el modo usuario

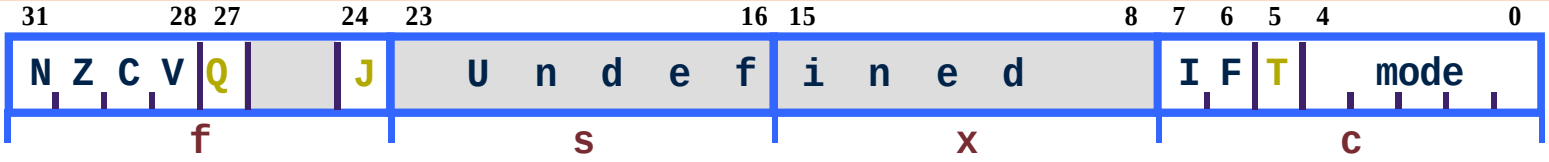
Registros y modos

- El modo en el que se encuentra el procesador determina los registros accesibles
- En cada modo se puede acceder a:
  - Un conjunto particular de registros generales (r0-r12)
  - Registros de puntero de pila (r13) y enlace (r14) particulares
  - El contador de programa (r15)
  - El registro de estado actual del programa (cpsr)
- En los modos privilegiados (excepto system) se puede acceder también a:
  - Un registro especial que almacena el estado del programa (spsr): **permite guardar el estado sin tener que apilarlo**

Registros y modos



Registro de estado



- Códigos de condición
  - N = resultado de la ALU negativo (C2)
  - Z = resultado de la ALU cero
  - C = operación de ALU produce acarreo
  - V = operación de ALU desborda (C2)
  - Sólo para la arquitectura 5TE/J
- Bits de habilitación de interrupciones
  - I = 1: Deshabilita IRQ.
  - F = 1: Deshabilita FIQ.
  - Sólo para arquitecturas -T
  - T = 0: repertorio ARM
  - T = 1: repertorio Thumb
- Bits de modo
  - Especifican el modo del procesador

Gestión de excepciones

- Cuando se produce una excepción **de forma automática**:
  - Se copia el CPSR en SPSR_<modo>
  - Se modifican los bits adecuados del CPSR
    - Cambio a repertorio ARM
    - Actualizar el registro de estado con la excepción correspondiente
    - Deshabilitar interrupciones (si procede)
  - Se guarda la dirección de retorno en LR_<modo>
  - Se actualiza el PC con el vector correspondiente

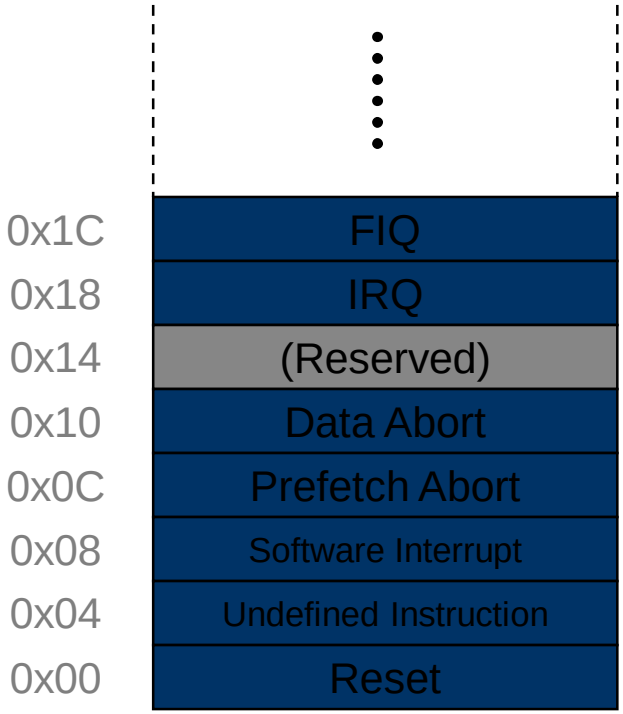


Tabla de Vectores

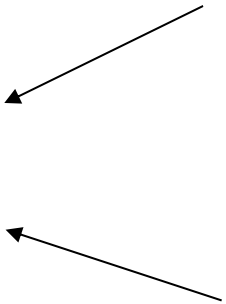
Gestión de excepciones

- Además se debe actualizar la dirección de retorno:
  - Restar 4 a LR en el caso de las interrupciones
- Guardar en pila los registros compartidos que se vayan a modificar.
  - ¿hay que guardar LR en pila?
- Para volver, la rutina de tratamiento de excepción tiene que:
  - Restaurar los registros
  - Restaurar el CPSR a partir del SPSR_<modo>
  - Restaurar el PC a partir del LR_<modo>
  - Se puede hacer con una única instrucción de store múltiple

Marco de pila de una rutina

- Estructura de una rutina:

Código de entrada (prólogo)
Cuerpo de la rutina
Código de salida (epílogo)



Construye el marco

Destruye el marco y hace el retorno

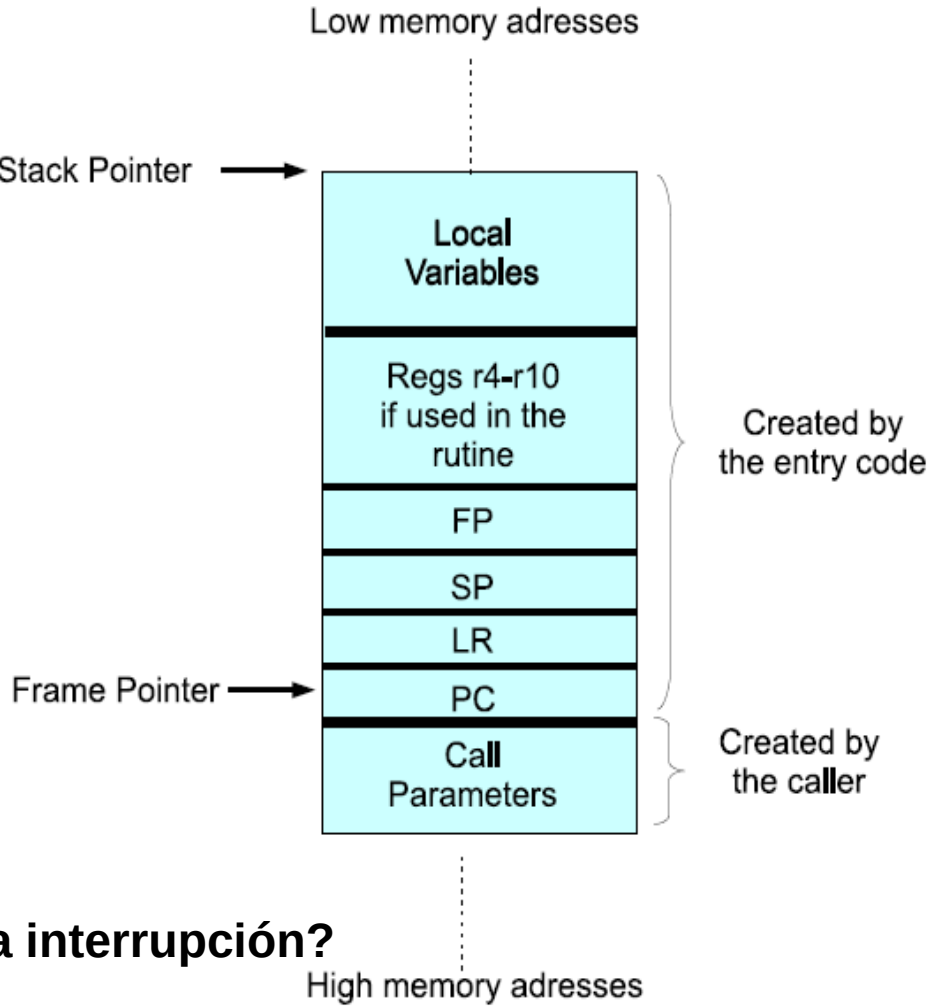
Marco de pila de una rutina

- Estructura de una rutina:
  - Prólogo

```
MOV     IP, SP
STMDB   SP!, {r4-r10,FP,IP,LR,PC}
SUB     FP, IP, #4
SUB     SP, #SpaceForLocalVaribles
```

  - Epílogo:

```
LDMDB   FP, {r4-r10,FP,SP,PC}
```

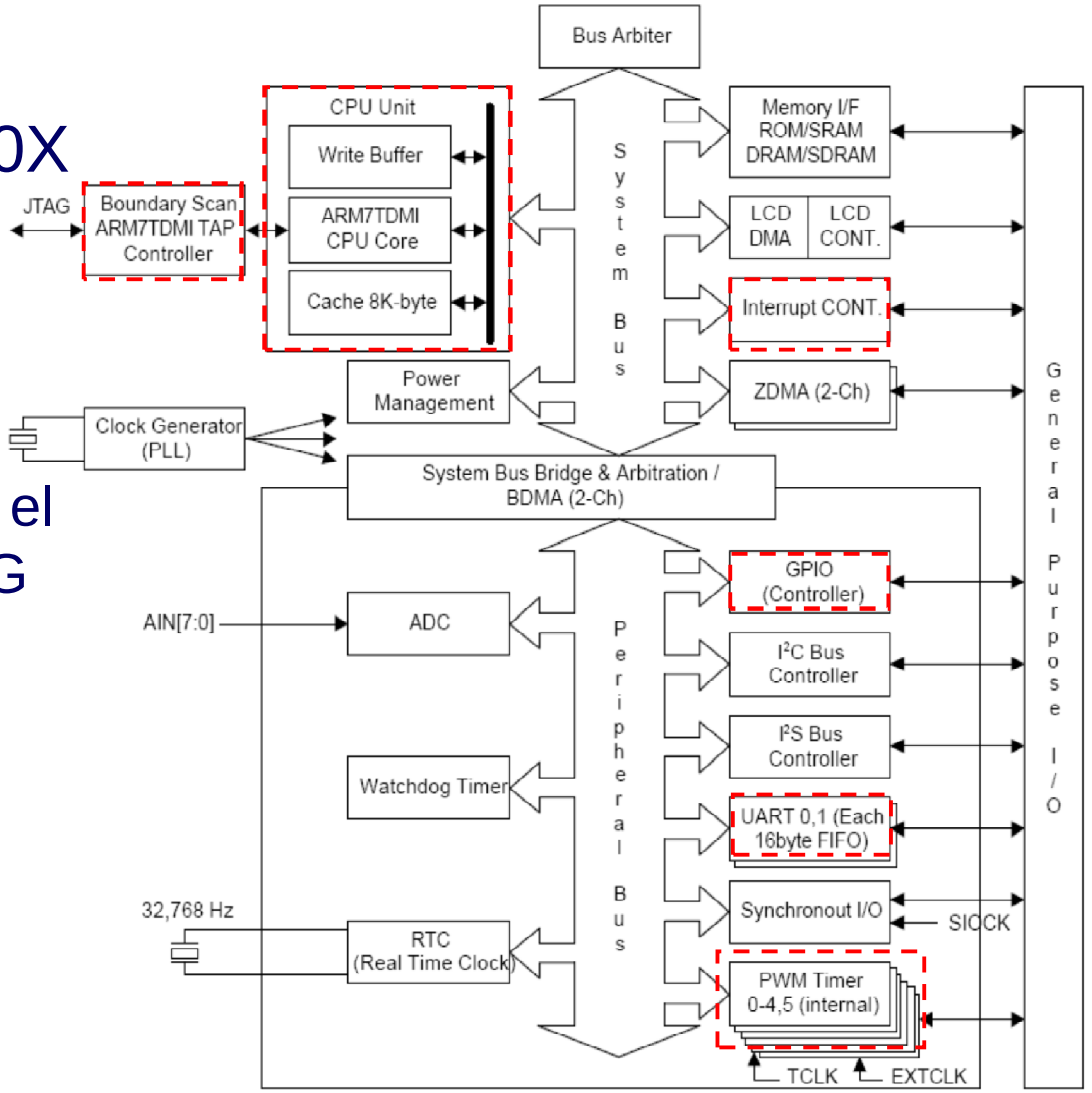


¿Qué hay que añadir en el caso de una interrupción?

Gestión de las interrupciones en nuestra placa de laboratorio:

System-on-Chip S3C44B0X

- Procesador: ARM7TDMI
- Controladores E/S
- Buses
- Comunicación directa con el PC a través del puerto JTAG
- Temporizadores
- **Controlador de interrupciones**
- y muchas otras cosas…

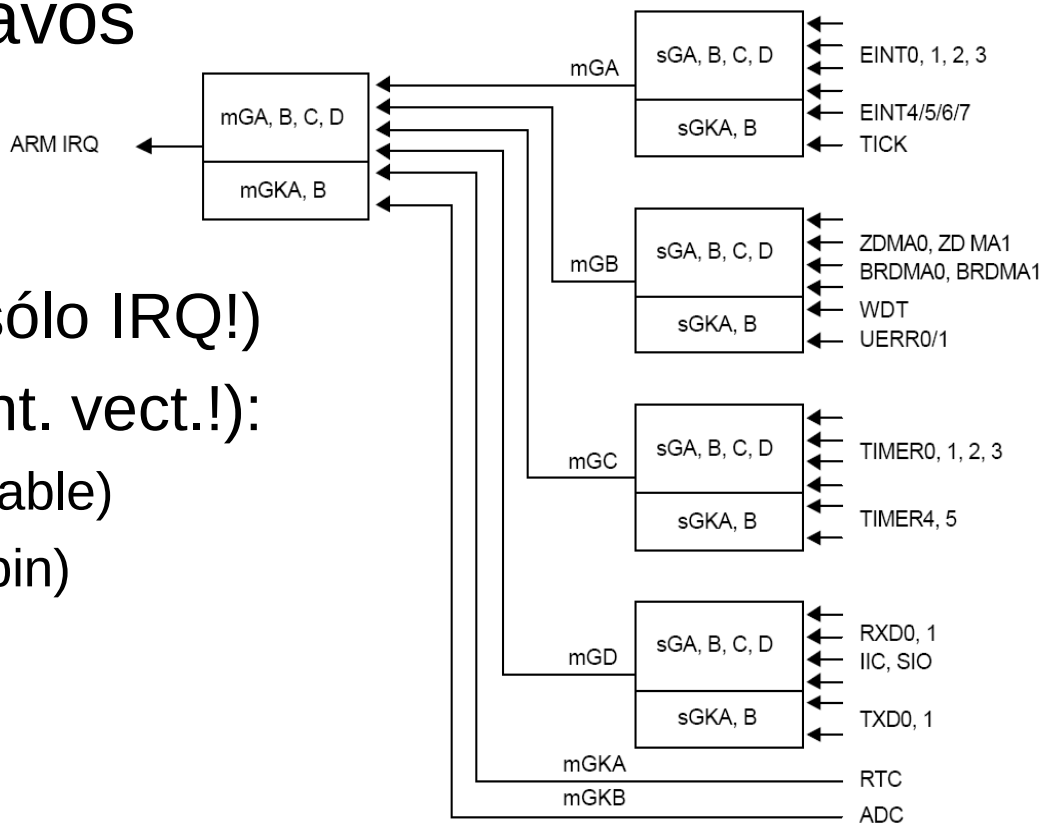


Controlador de interrupciones del S3C44B0X

- Permite ampliar el nº de líneas de petición de interrupción del ARM7TDMI
- 30 posibles fuentes de interrupción usando 26 líneas (algunas fuentes comparten línea)
- Añade soporte de interrupciones vectorizadas (¡Solo para aquellas fuentes que usan IRQ!)
- Implementado mediante 5 controladores encadenados (1 master + 4 slaves)

Controlador de interrupciones

- 26 líneas de interrupción
- 1 maestro + 4 esclavos
- Configurable:
  - Modo IRQ/FIQ
  - Int. vectorizadas (¡sólo IRQ!)
  - Prioridades (¡sólo int. vect.!):
    - Orden (fijo/programable)
    - Modo (fijo/round-robin)



Funcionamiento interrupciones vectorizadas:

- Cuando el ARM intenta leer la instrucción en la dir. 0x18 (vector IRQ)
- El controlador actúa sobre el bus de datos e inserta una instrucción de salto al vector correspondiente a la línea más prioritaria activa
  - EINT0 (0x20)
  - EINT1 (0x24)
  - …
  - EINT4/5/6/7 (0x30)
  - …
  - INT_ADC (0xc0)

Controlador de interrupciones

- Código de ejemplo para int. vectorizadas:

```
…
b HandlerIRQ                ; 0x18
b HandlerFIQ                ; 0x1c
ldr pc,=HandlerEINT0        ; 0x20
ldr pc,=HandlerEINT1        ; 0x24
ldr pc,=HandlerEINT2        ; 0x28
ldr pc,=HandlerEINT3        ; 0x2c
ldr pc,=HandlerEINT4567     ; 0x30
…
```

Controlador de interrupciones

- Registros de configuración
  - INTCON (Interrupt Control Register), 3 bits
    - V (bit [2]) = 0, habilita las interrupciones vectorizadas
    - I (bit [1]) = 0, habilita la línea IRQ
    - F (bit [1]) = 0, habilita la línea FIQ
  - INTMOD (Interrupt Mode Register), 1 bit por línea
    - 0 = modo IRQ; 1 = modo FIQ

Controlador de interrupciones

- Registros de gestión
  - INTPND (Interrupt Pending Register), 1 bit por línea
    - 0 = no hay solicitud; 1 = hay una solicitud
  - INTMSK (Interrupt Mask Register), 1 bit por línea
    - 0 = int. disponible; 1 = int. enmascarada
  - I_ISPC (IRQ Int. Service Pending Clear register), 1 bit por línea
    - 1 = borra el bit correspondiente del INTPND e indica al controlador el final de la rutina de servicio **(¡FIN RUTINA SERVICIO!)**
  - F_ISPC (FIQ Int. Service pending Clear register), 1 bit por línea
    - Idem

Controlador de interrupciones

- Registros de gestión para int. vectorizadas:
  - I_ISPR (IRQ Int. Service Pending Register), 1 bit por línea
    - Indica la interrupción que se está sirviendo actualmente
    - Sólo el bit de la línea más prioritaria puede estar a 1 (≠INTPND)

4. DMA: Necesidad de acceso directo a memoria

- La E/S con espera de respuesta o por interrupciones resulta inadecuada para periféricos de alta velocidad, sobre todo si hay que transferir una gran cantidad de datos
- **Ejemplo periférico lento**
  - Procesador a 200 MHz (tiempo ciclo = 5 ns.; Ciclo medio por instrucción: CPI = 2 ciclos
    - Una instrucción tarda en promedio 2 x 5 ns = 10 ns ⇒ el computador puede ejecutar ~100 MIPS
  - Queremos imprimir un fichero de 10 Kbytes en una impresora láser de 20 páginas por minuto
  - 1 página ≅ 3.000 caracteres (1 carácter = 1 byte)
    - La impresora imprime 60.000 caracteres por minuto = 1 Kbyte/s

  a) E/S con espera de respuesta
  - La CPU entra en un bucle y envía un nuevo byte cada vez que la impresora está preparado para recibirlo
    - La impresora tarda 10 s en imprimir 10 Kbyte
    - **La CPU está ocupada con la operación de E/S durante 10 s**
      (en ese tiempo la CPU podría haber ejecutado 1000 millones de instrucciones)

  b) E/S por interrupciones
  - La impresora genera una interrupción cada vez que está preparada para recibir un nuevo byte
    - Suponemos que la RTI tiene 10 instrucciones (salvar contexto, comprobar estado, transferir byte, restaurar contexto, RTE)
    - Para transferir 10 Kbyte tenemos que ejecutar 10.000 veces la RTI
      ⇒ hay ejecutar 100.000 instrucciones para atender al periférico ⇒ la CPU tarda 0,001 s
    - **La CPU está ocupada con la operación de E/S durante 0,001 s**

  CONCLUSIÓN
  - La E/S por interrupciones reduce en 10.000 veces el tiempo que la CPU está ocupada gestionando la impresora

Necesidad de DMA

⌦ **Ejemplo periférico rápido**

- ➢ Procesador a 200 MHz (tiempo ciclo = 5 ns.; Ciclo medio por instrucción: CPI = 2 ciclos
  - ⇨ Una instrucción tarda en promedio 2 x 5 ns = 10 ns ⇒ el computador puede ejecutar ~100 MIPS
- ➢ Disco con velocidad de transferencia de 10 Mbytes/s (1 byte cada $2*10^{-7}$ seg)
- ➢ Queremos transferir un fichero de memoria a disco de 10 Mbytes

a) E/S con espera de respuesta

- ➢ La CPU entra en un bucle y envía un nuevo byte cada vez que el disco está preparado para recibirlo
  - ⇨ El disco tarda 1 seg en recibir un fichero de 10 Mbyte
  - ⇨ **La CPU está ocupada con la operación de E/S durante 1 s**
    (en ese tiempo la CPU podría haber ejecutado 200 millones de instrucciones)

b) E/S por interrupciones

- ➢ El disco genera una interrupción cada vez que está preparado para recibir un nuevo byte
  - ⇨ Suponemos que la RTI tiene 10 instrucciones (salvar contexto, comprobar estado, transferir byte, restaurar contexto, RTE)
  - ⇨ Para transferir 10 Mbytes tenemos que ejecutar 10^7 veces la RTI
    ⇒ hay ejecutar 100 millones de instrucciones para atender al periférico ⇒ la CPU tarda 1 s
  - ⇨ **La CPU está ocupada con la operación de E/S durante 1 s**

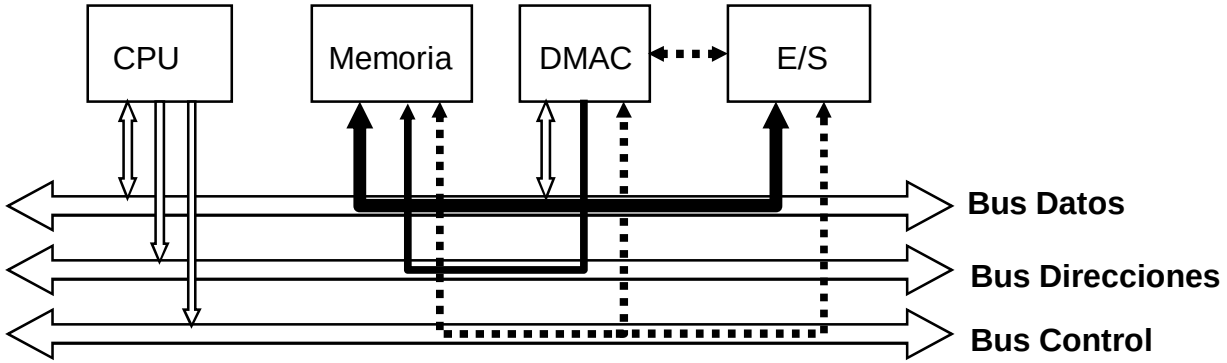
CONCLUSIÓN

- ➢ La E/S por interrupciones no mejora el tiempo que la CPU está ocupada en atender al periférico

La técnica de DMA permite la transferencia de datos entre un periférico y la memoria **sin intervención de la CPU** (salvo en la fase de inicialización de los parámetros de la transferencia)

El controlador de DMA (DMAC)

- El controlador de DMA es un dispositivo capaz de controlar una transferencia de datos entre un periférico y memoria sin intervención de la CPU



- El DMAC debe actuar como máster del bus durante la transferencia DMA y debe ser capaz de
  - Solicitar el uso del bus mediante las señales y la lógica de arbitraje necesarias
  - Especificar la dirección de memoria sobre la que se realiza la transferencia
  - Generar las señales de control del bus
    - Tipo de operación (lectura/escritura)
    - Señales de sincronización de la transferencia

Etapas de una transferencia DMA

- **Inicialización de la transferencia**
  - La CPU debe enviar al interfaz del periférico y al DMAC los parámetros de la transferencia

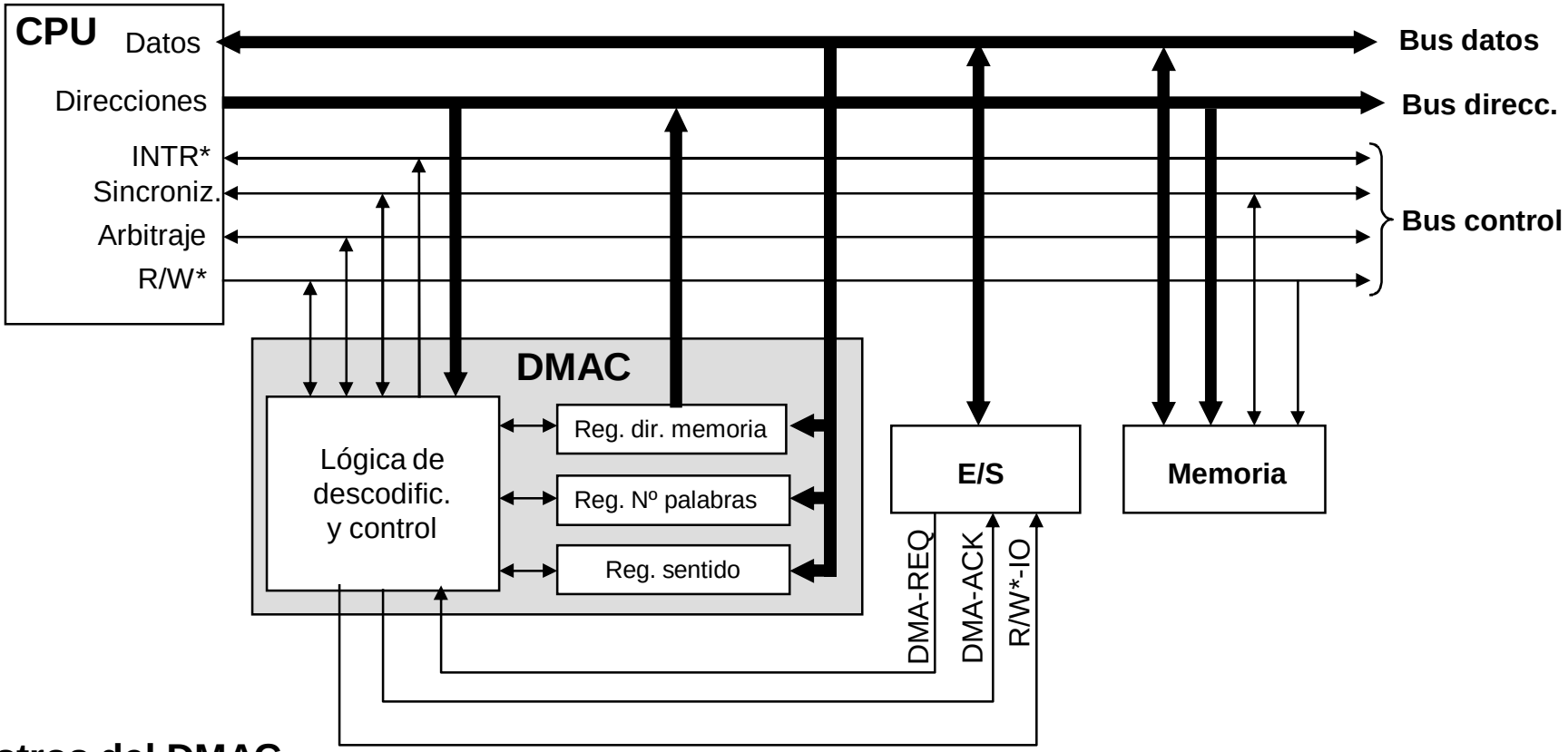
  Inicialización del interfaz (Bus master: CPU - Bus slave: Interfaz)
    - Nº de bytes a transferir
    - Tipo de transferencia (lectura/escritura)
    - Otra información de control (pista, sector, etc.)

  Inicialización del controlador DMA (Bus master: CPU - Bus slave: DMAC)
    - Nº de bytes o palabras a transferir
    - Tipo de transferencia (lectura/escritura)
    - Dirección de memoria inicial para la transferencia
    - Nº de canal (para DMAs con varios canales)
  - Después de la inicialización la CPU retorna a sus tareas y ya no se preocupa más de la evolución de la transferencia
- **Realización de la transferencia**
  - Cuando el periférico está listo para realizar la transferencia se lo indica al DMAC
  - El DMAC pide el control del bus y se realiza la transferencia entre el periférico y la memoria
    - **Bus máster:** DMAC + Periférico - **Bus slave:** Memoria
    - Después de la transferencia de cada palabra se actualizan los registros del DMA
      - ✓ Nº de bytes o palabras a transferir
      - ✓ Dirección de memoria
- **Finalización de la transferencia**
  - El DMAC libera el bus y devuelve el control a la CPU
  - El DMAC suele activar una señal de interrupción para indicar a la CPU la finalización de la operación de E/S solicitada

Transferencias DMA

- **Transferencia DMA modo ráfaga**
  - El DMAC solicita el control del bus a la CPU
  - Cuando la CPU concede el bus el DMAC no lo libera hasta haber finalizado la transferencia de todo el bloque de datos completo
  - **VENTAJAS:**
    - La transferencia se realiza de forma rápida
  - **DESVENTAJAS:**
    - Durante el tiempo que dura la transferencia la CPU no puede utilizar el bus con memoria, lo que puede degradar el rendimiento del sistema
- **Transferencia DMA modo robo de ciclo**
  - El DMAC solicita el control del bus a la CPU
  - Cuando la CPU concede el bus al DMAC, se realiza la transferencia de una única palabra y después el DMAC libera el bus
  - El DMAC vuelve a solicitar el control del bus tantas veces como sea necesario hasta haber finalizado la transferencia del bloque completo
  - La CPU cede el control del bus durante los ciclos que hace uso del mismo
  - **VENTAJAS:**
    - No se degrada el rendimiento del sistema
  - **DESVENTAJAS:**
    - La transferencia tarda más tiempo en llevarse a cabo

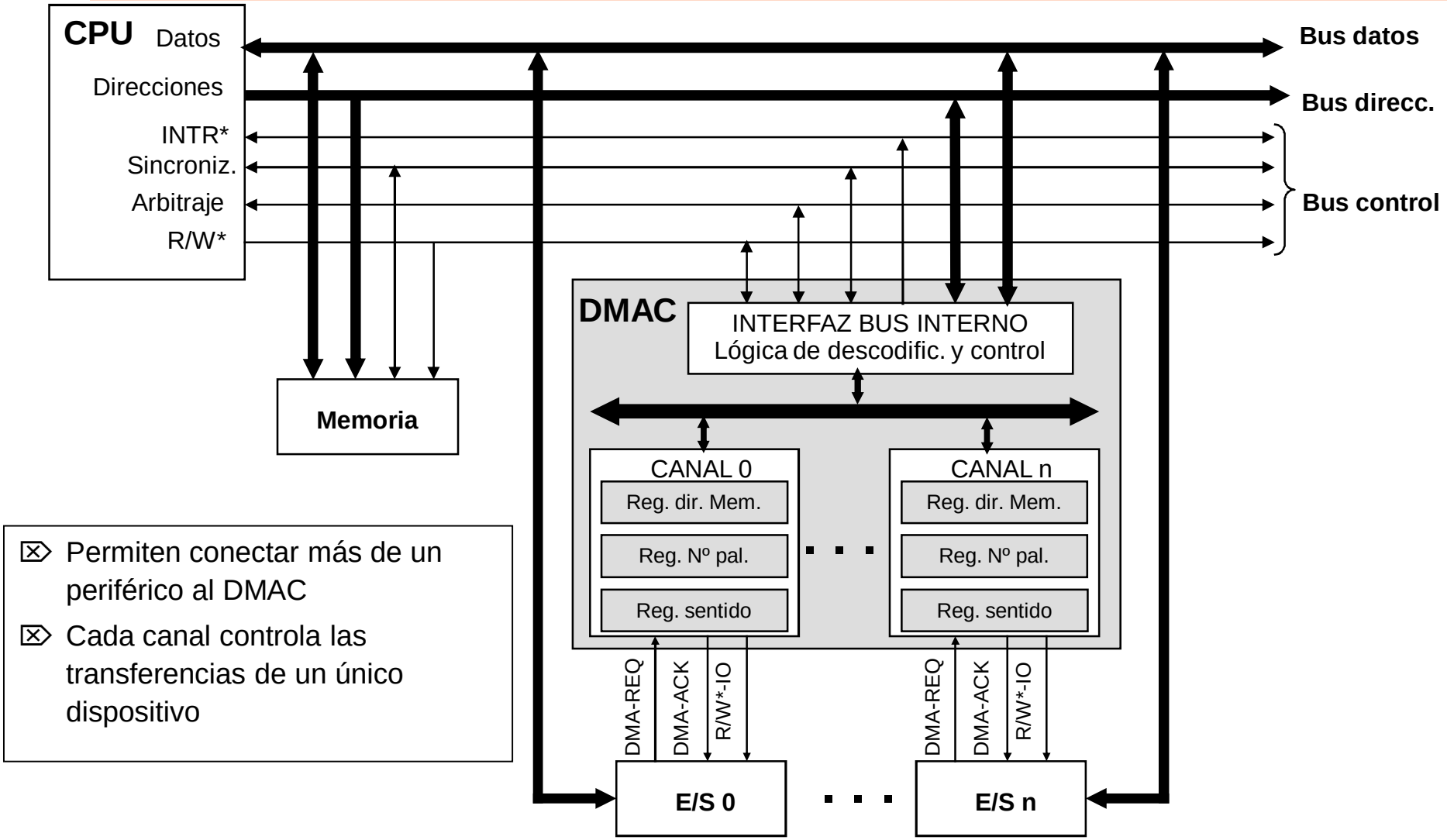
Estructura hardware de un DMAC



Registros del DMAC

- **Reg. dir. memoria:** almacena la dir. inicial de memoria y se incrementa/decrementa después de transferir cada palabra
- **Reg. Nº palabras:** almacena el número de palabras a transferir y se decrementa después de transferir cada palabra
- **Reg. sentido:** almacena el sentido de la transferencia (lectura o escritura)

Controlador DMA de varios canales



- Permiten conectar más de un periférico al DMAC
- Cada canal controla las transferencias de un único dispositivo

Conclusiones

- Los procesadores deben interactuar con todo tipo de dispositivos
- Estos dispositivos suelen incluir un interfaz para facilitar las comunicaciones
- Las interrupciones son la clave para una E/S eficiente
- Debido al gran número de fuentes de interrupción muchos sistemas incluyen un controlador de interrupciones
- Las transferencias de datos son claves para el rendimiento de un sistema:
  - Los DMAs permiten descargar al procesador