Módulo 2. Entrada/salida
Tema 2.1. Sistema de Entrada/salida

Indice

- Introducción
  - Estructura y funciones del sistema de E/S
  - Módulo de E/S
  - Operación de E/S
- E/S programada
- E/S mediante interrupciones
  - Interrupción y excepción. E/S por interrupción
  - Gestión de interrupciones. Anidamiento.
  - Controlador de interrupciones

- Bibliografia:

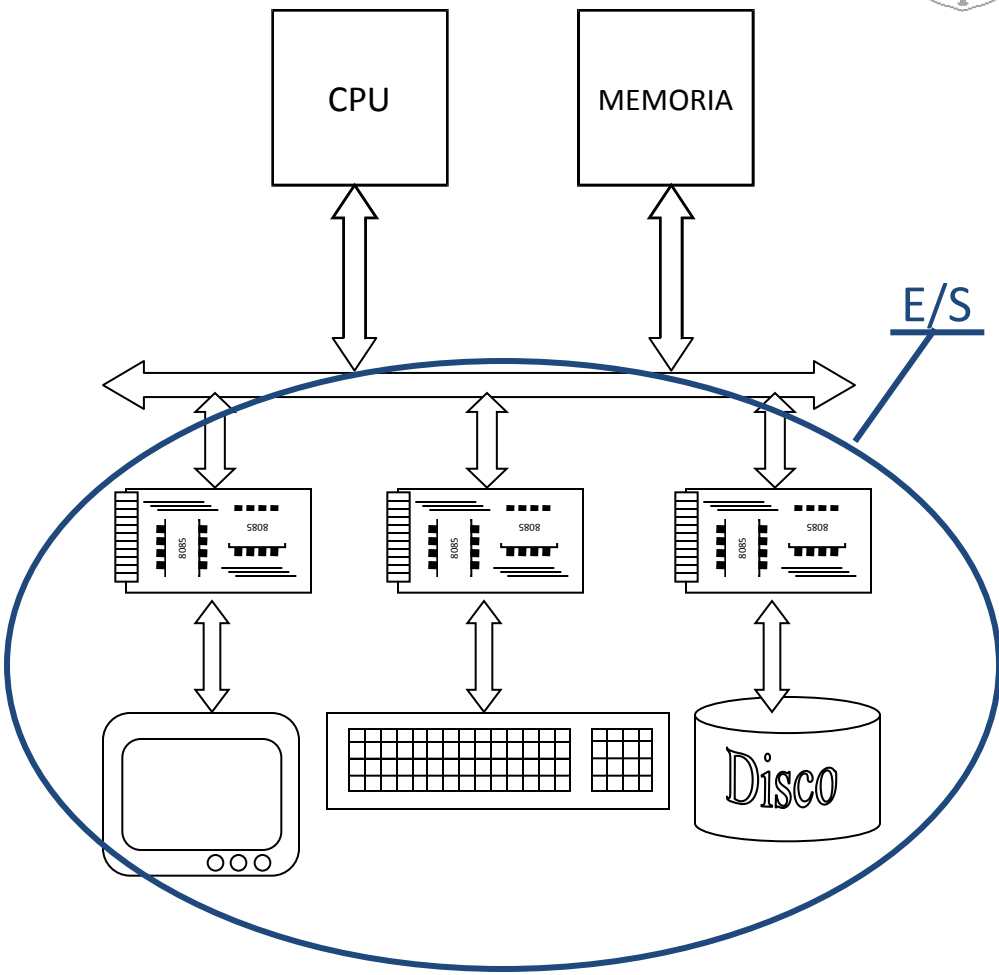
- W. Stallings; Computer Organization and Architecture, 9ed. Prentice Hall 2013.
- D. A. Patterson & J. L. Hennessy; Estructura y diseño de computadores. La interfaz hardware/software, 4ed. Reverté 2011.
- B. Parhami; Arquitectura de Computadoras. De los microprocesadores a las supercomputadoras. McGrawHill, 2007
- S. Furber; ARM System-on-Chip architecture, 2ed. Addison-Wesley 2000.

Tema 2.1. Sistema de Entrada/salida

Introducción

Necesidad del sistema de E/S

- ¿De dónde proceden los datos que llegan a memoria? ¿A dónde van?
- ¿Cómo interacciona la CPU con el mundo exterior?
- Este tema presenta los aspectos más relevantes de la E/S
  - En muchas ocasiones, el rendimiento del sistema de E/S determina el rendimiento global del sistema



Ejemplos de periféricos

Tipos de periféricos

- Dispositivos de **presentación de datos**.
  - Interaccionan con los usuarios, transportando datos entre éstos y la máquina
  - Ratón, teclado, pantalla, impresora, etc.
- Dispositivos de **comunicación** con otros procesadores.
  - Permiten la comunicación con procesadores remotos a través de redes
  - Tarjeta de red, módem...
- Dispositivos de **adquisición de datos**.
  - Permiten la comunicación con sensores y actuadores que operan de forma autónoma.
  - Se utilizan en sistemas de control automático de procesos por computador
  - Suelen incorporar conversores de señales A/D y D/A.

Comunicación

- Dispositivos de **almacenamiento de datos**.
  - Forman parte de la jerarquía de memoria: interactúan de forma autónoma con la máquina
  - Discos magnéticos y cintas magnéticas…

Tipos de periféricos (2)

Input type	Prime examples	Other examples	Data rate (b/s)	Main uses
Symbol	Keyboard, keypad	Music note, OCR	**10s**	Ubiquitous
Position	Mouse, touchpad	Stick, wheel, glove	**100s**	Ubiquitous
Identity	Barcode reader	Badge, fingerprint	**100s**	Sales, security
Sensory	Touch, motion, light	Scent, brain signal	**100s**	Control, security
Audio	Microphone	Phone, radio, tape	**1000s**	Ubiquitous
Image	Scanner, camera	Graphic tablet	**1000s-10^6s**	Photos, publishing
Video	Camcorder, DVD	VCR, TV cable	**1000s-10^9s**	Entertainment
Output type	**Prime examples**	**Other examples**	**Data rate (b/s)**	**Main uses**
Symbol	LCD line segments	LED, status light	10s	Ubiquitous
Position	Stepper motor	Robotic motion	100s	Ubiquitous
Warning	Buzzer, bell, siren	Flashing light	A few	Safety, security
Sensory	Braille text	Scent, brain stimulus	100s	Personal assistance
Audio	Speaker, audiotape	Voice synthesizer	1000s	Ubiquitous
Image	Monitor, printer	Plotter, microfilm	1000s	Ubiquitous
Video	Monitor, TV screen	Film/video recorder	1000s-10^9s	Entertainment
Two-way I/O	**Prime examples**	**Other examples**	**Data rate (b/s)**	**Main uses**
Mass storage	Hard/floppy disk	CD, tape, archive	10^6s	Ubiquitous
Network	Modem, fax, LAN	Cable, DSL, ATM	1000s-10^9s	Ubiquitous

Tomada de las transparencias de clase de (de B. Parhami, 2011)

Tipos de periféricos (3)

- Son muy distintos en cuanto a:
  - Tipo de datos que se transmiten
  - Tasa de transferencia (determina la interacción con el sistema)
    - Ancho de banda: cantidad de datos que se pueden transferir por unidad de tiempo (medido en Mbit/s, MB/s…)
      - Los dispositivos de almacenamiento (discos SATA) y red (Gigabit Ethernet) proporcionan un gran ancho de banda
    - Latencia: tiempo requerido para obtener el primer dato (medido en segundos)
      - La latencia de la mayoría de los dispositivos de E/S es muy alta en comparación con la velocidad del procesador

Conclusiones:

Necesitamos un interfaz que sea distinto para cada uno: módulo de E/S

Necesitamos distintas formas de interaccionar con ellos: técnicas de E/S

Estructura del sistema de E/S

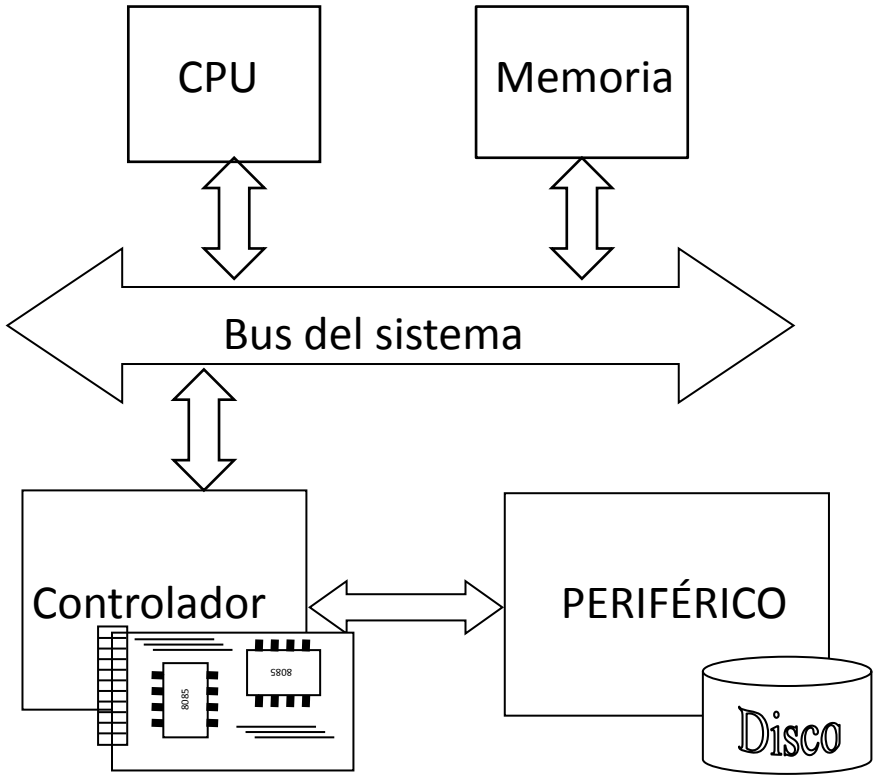
- Componentes:
  - Dispositivo periférico
  - Módulo de E/S, controlador (hardware)
    - Tarjeta
  - Controlador software
    - Driver

S.O.

  - Canal de comunicación
    - P. ej. bus

TEMA SIGUIENTE

- Organización sencilla del sistema de E/S



Componentes del sistema de E/S

- **Dispositivo periférico**
  - Dispositivo en sí, de carácter mecánico, magnético…. (teclado, platos del disco..)
- **Controlador del dispositivo (hardware)**
  - Interfaz lógica que ofrece el dispositivo al sistema
  - Se *entiende* con el bus y conoce las características físicas del dispositivo
- **Controlador software (*driver*)**
  - Código encargado de programar el controlador HW del dispositivo concreto
  - Forma parte del sistema operativo (suele proporcionarlo el fabricante)
- **Canal de comunicación (p. ej. bus)**
  - Interconexión entre la CPU, memoria y el controlador HW del dispositivo
  - Puede ser compartido por varios dispositivos
    - Necesidad de arbitraje

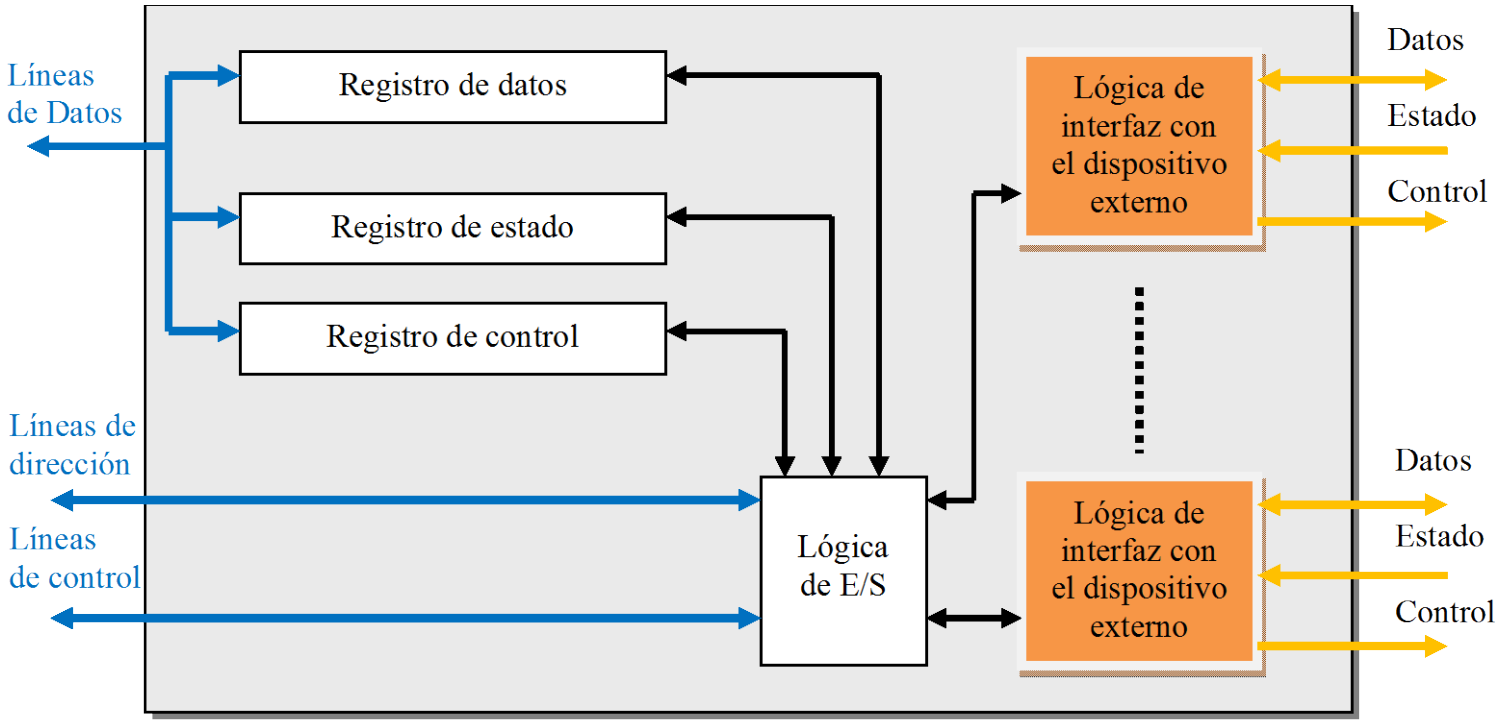
Módulo de entrada/salida

- Interfaz entre el dispositivo y el procesador, que oculta las particularidades de los dispositivos y adapta la velocidad de transferencia.
- Funciones:
  - Control y temporización
  - Comunicación con el procesador o la memoria
  - Comunicación con el periférico
  - Buffering o almacenamiento intermedio
  - Detección de errores

Tiene dos interfaces

Estructura de un módulo de E/S

- Traduce peticiones de la CPU (genéricas) al periférico específico
  - La CPU lee y/o escribe en sus registros: datos, estado y control (por ejemplo para solicitar el bloque 33).
  - **El controlador pide al disco que mueva el brazo** a la pista 12 y lea el sector 313.



Registros de un módulo de E/S

- **Registro de datos** de entrada/salida
  - Usados para el intercambio de datos entre CPU y periférico
- **Registro de estado**
  - Contiene el estado del dispositivo: preparado, situación de error, ocupado…
- **Registro de control**
  - Codifica la orden que la CPU da al dispositivo (imprime carácter, lee un bloque…)
- CUIDADO: NO son registros de la CPU. ¡¡¡Están (generalmente) en otro chip!!!

Características de los módulos de E/S

- Unidad de transferencia
  - Bloque: cintas, discos
  - Carácter: terminales, impresoras

Depende del periférico

- Direccionamiento:
  - Espacio de direcciones separado: E/S aislada
  - En memoria

Depende de la arquitectura

- Interacción computador-controlador:
  - E/S programada
  - E/S mediante interrupciones
  - E/S mediante DMA

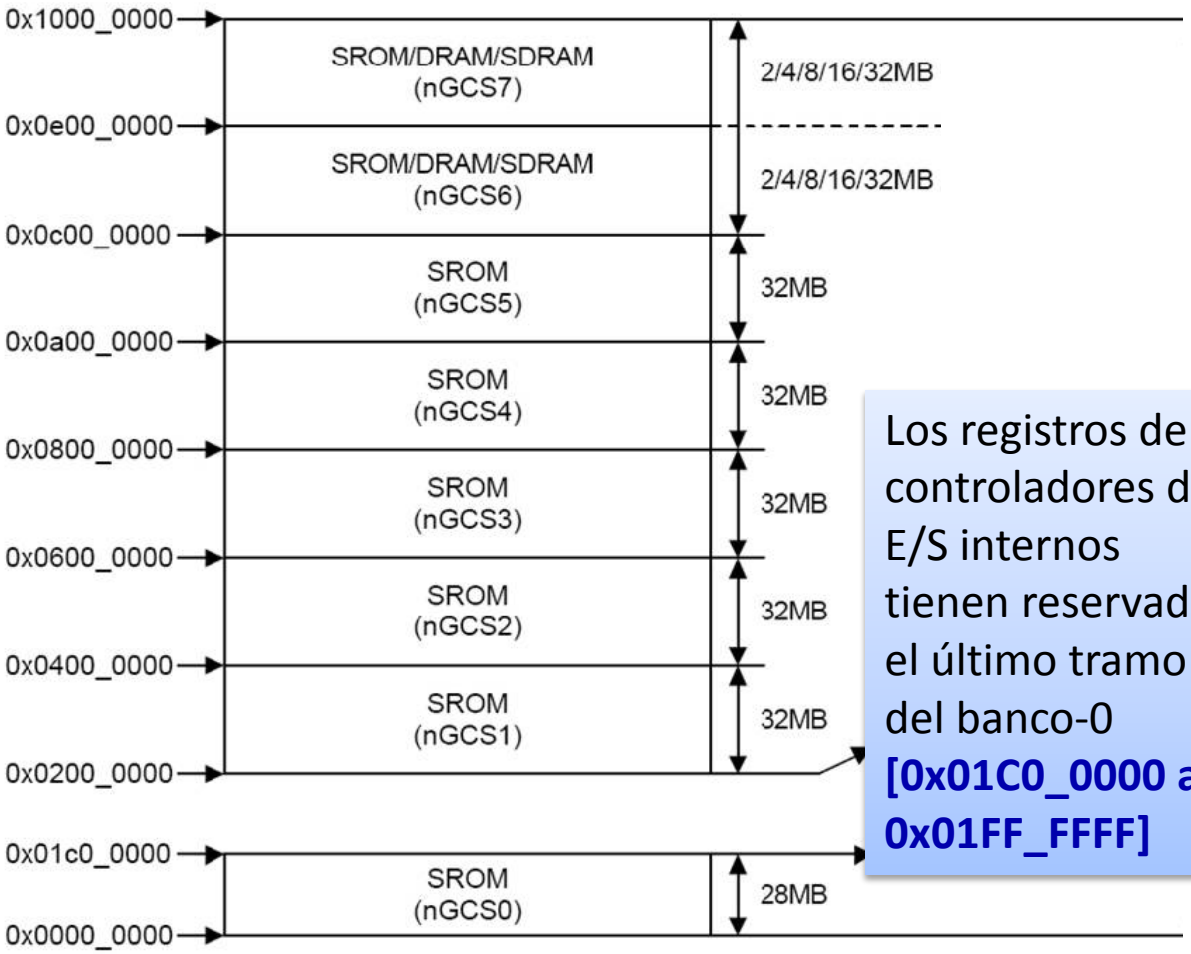
Depende de la tasa de transferencia. Puede necesitar un controlador especial.

Direccionamiento

- E/S aislada
  - Espacio de direcciones diferente al de la memoria principal (existen dos mapas de direcciones)
  - Existen instrucciones específicas de E/S
    - IN dir_E/S, Ri (CPU ← Periférico)
    - OUT Ri, dir_E/S (Periférico ← CPU)
  - Ejemplo: Intel x86
    - Cada registro de un módulo de E/S es un puerto.
- E/S localizada en memoria
  - La memoria y los dispositivos de E/S comparten el espacio de direcciones
  - Podemos usar instrucciones tipo load/store (lw/sw)
    - LOAD Ri, dir_E/S (CPU ← Periférico)
    - STORE Ri, dir_E/S (Periférico ← CPU)
  - Ejemplo: ARM
    - Cada registro de un módulo de E/S es una dirección de memoria dentro de un rango reservado

Direccionamiento: ejemplo FDI

- Mapa de memoria del sistema de los laboratorios



Los registros de los controladores de E/S internos tienen reservado el último tramo del banco-0 **[0x01C0_0000 a 0x01FF_FFFF]**

Operación de E/S

- Una operación de entrada/salida se divide en:
  - Petición: enviar el comando (y dirección).
    - El maestro (Master) es quien inicia la transacción.
    - La CPU indica al dispositivo la operación que quiere hacer escribiendo en su *registro de control* .
      - ¿Cómo indicar el dispositivo de E/S deseado? Depende de que sea E/S aislada o mapeada en memoria.
  - Envío: transferencia de los datos
    - El esclavo (Slave) responde a la petición
    - La CPU leerá/escribirá de/al *registro de datos* del controlador del periférico deseado
      - ¿Cómo saber cuándo? SINCRONIZACIÓN
- El código que interacciona a tan bajo nivel con el dispositivo es el conocido como ***controlador SW (driver)***
  - Es parte del sistema operativo
  - En el laboratorio trabajaremos con el ARM directamente (bare metal), sin sistema operativo.

Ejemplo: controlador de los leds del laboratorio

- El controlador de los leds del laboratorio (puerto B) tiene las siguientes direcciones de memoria asignadas a sus registros:
  - Registro de control PCONB: 0x01D20008
  - Registro de datos PDATB: 0x01D2000C
- Configurar los pines del controlador como salidas para poder escribir en los leds:
  - Escribir un '0' en PCONB

```
mov r0,#0
ldr r1,=0x01D20008
str  r0,[r1]
```

- Encender los leds:
  - Escribir en PDATB:
    - 0 enciende el led,
    - 1 apaga el led.

```
mov r0,#0
ldr r1,=0x01D2000C
str  r0,[r1]
```

Sincronización

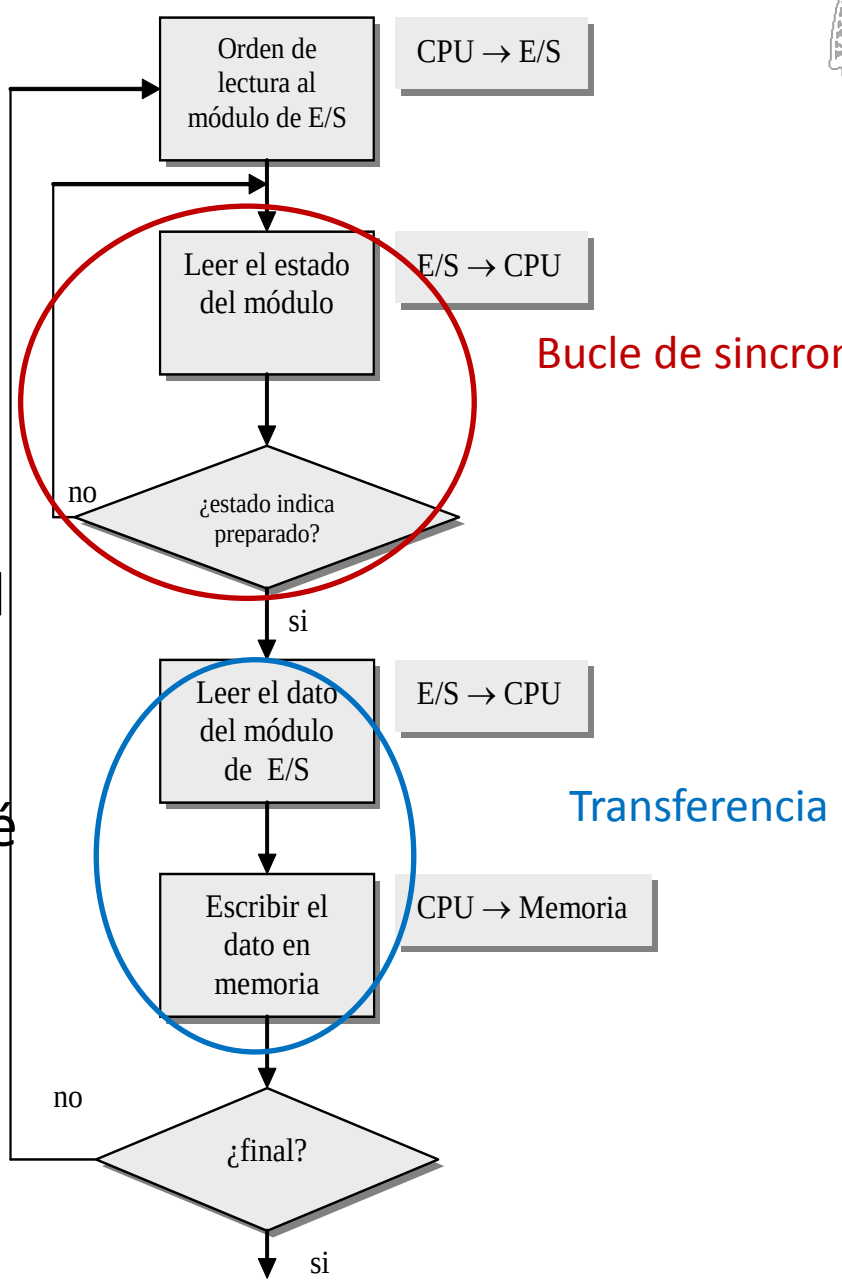
- Exigida por la diferencia de velocidad entre la CPU y los dispositivos de E/S.
- Antes de enviar/recibir datos a/desde un periférico hay que asegurarse de que el dispositivo está preparado para realizar la transferencia.
  - ¿Cómo sabe la CPU cuándo está lista/ha terminado la transferencia?
  - ¿Cómo sabe si todo ha ido bien?
- Existen dos mecanismos básicos de sincronización de la E/S
  - E/S programada con espera de respuesta (polling)
  - E/S por interrupciones

Tema 2.1. Sistema de Entrada/salida

E/S programada

¿En qué consiste?

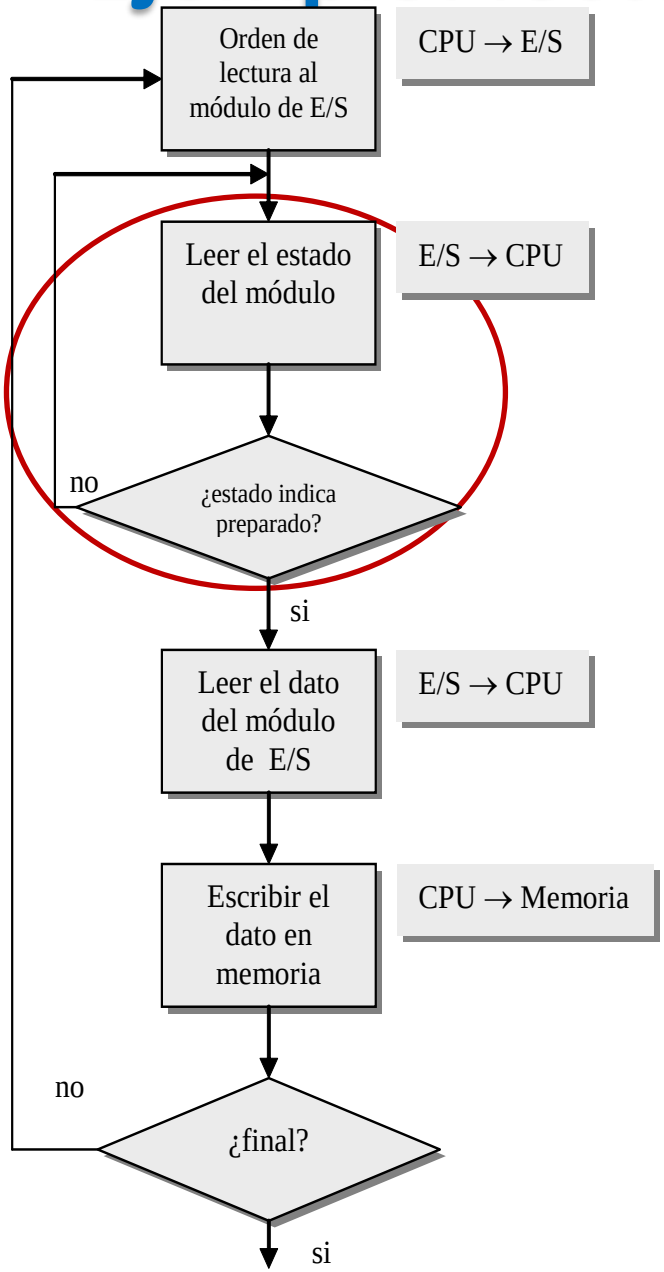
- Cada vez que la CPU quiere realizar una transferencia:
  - entra en un **bucle** en el que lee una y otra vez el **registro de estado** del periférico (encuesta o "polling") hasta que esté preparado para realizar la transferencia
  - realiza la transferencia



E/S programada (polling)

- Problemas
  - 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 pulse una tecla
  - Dificultades para atender a varios periféricos
    - Mientras se espera a que un periférico esté listo para transmitir, no se puede atender a otro

Ejemplo: lectura de los pulsadores




```
            ldr r0,=PDATG
...
bucleDet: ldr r1,[r0]
            mvn r1,r1            @complemento el valor
            and r1,r1,#0x0040    @ me quedo con el bit 6
            cmp r1,#0
            beq bucleDet         @ si no hubo pulsacion
@boton 1 pulsado
```

El bit 6 de PDATG es:
0 si el botón 1 ha sido pulsado y
1 si no ha sido pulsado

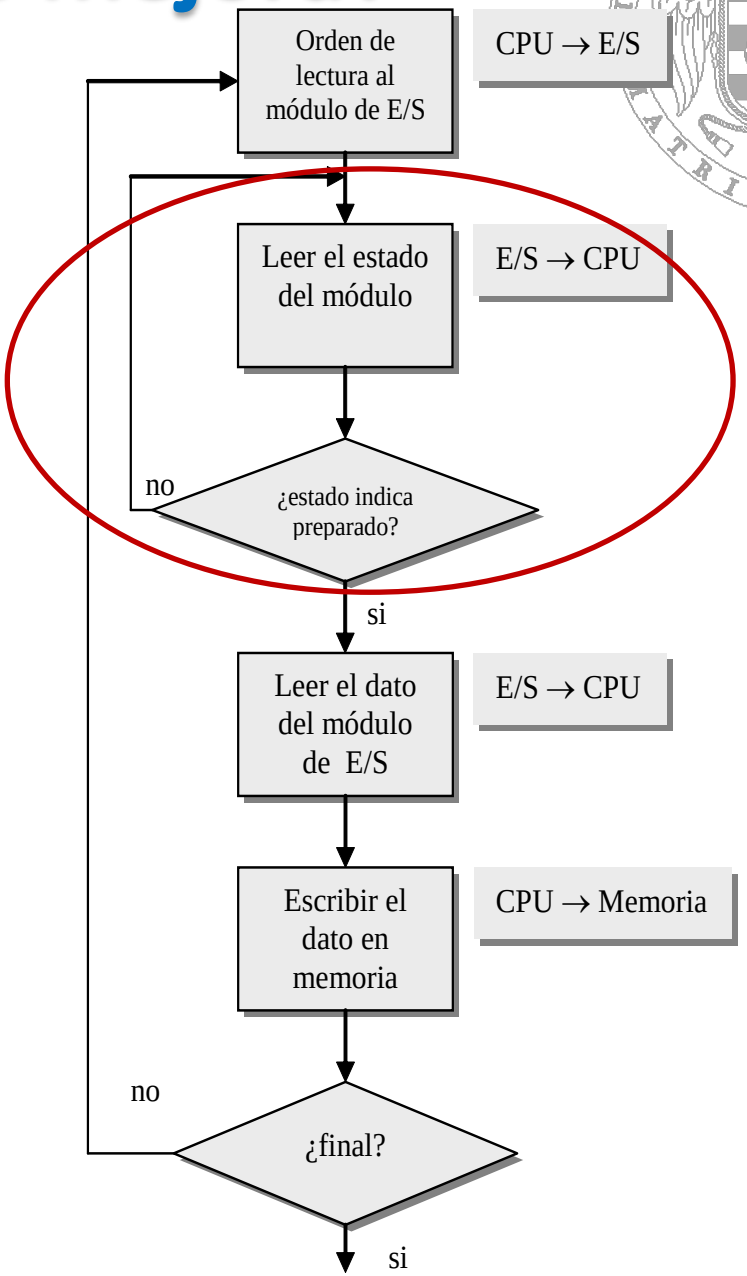
Evaluación del ejemplo

```
            ldr r0,=PDATG
...
bucleDet: ldr r1,[r0]
            mvn r1,r1           @complemento el valor
            and r1,r1,#0x0040   @ me quedo con el bit 6
            cmp r1,#0
            beq bucleDet        @ si no hubo pulsación
@boton 1 pulsado
```

Ejercicio: ¿cuántas veces se ejecuta el bucle suponiendo que soy capaz de pulsar el botón 1 vez por segundo, que el procesador tiene una frecuencia de reloj de 40MHz y cada instrucción tarda 3 ciclos en ejecutarse?

¿Hay margen de mejora?

- E/S programada:
  - El bucle de SINCRONIZACIÓN ejecuta instrucciones inútiles.
- ¿Y **si el dispositivo avisase a la CPU cuando haya terminado** su operación?
  - La CPU podría hacer trabajo útil mientras el periférico hace la operación solicitada
  - Sólo tendría que retomar el control de la operación cuando el periférico hubiese terminado
- Este mecanismo se denomina E/S por interrupciones



Tema 2.1. Sistema de Entrada/salida

E/S mediante interrupciones

Interrupciones y excepciones

- Excepción:
  - Evento inesperado que provoca un cambio en el flujo de control normal del programa.
- Tipos de excepciones:
  - Excepciones hardware
    - Internas: producidas por la CPU (división por cero, desbordamiento, instrucción ilegal, dirección ilegal, raíz cuadrada de negativos, etc.)
    - Externas: producidas por los dispositivos de E/S (Interrupciones)
  - Excepciones software (trap): producidas por la ejecución de instrucciones de la CPU.
- Interrupción:
  - Señal externa al procesador que provoca la detención del programa en curso para que la CPU realice otra actividad.
    - Es una excepción hardware externa. Pero la terminología es confusa: Intel llama interrupciones a las excepciones.

Excepciones en el ARM

Prioridad	Excepción	Vector
1	Reset	0x00
2	Data Abort	0x10
3	FIQ	0x1C
4	IRQ	0x18
5	Prefetch Abort	0x0C
6	Instr. no definida	0x04
7	SWI	0x08

- Excepciones externas
  - 1 línea externa de RESET,
  - 2 líneas externas de interrupción (IRQ y FIQ)
- Excepciones internas:
  - error al acceder a memoria para leer una instrucción (prefetch) o un dato (data);
  - error al decodificar una instrucción (Undef).
- Excepción software (trap):
  - para llamar al sistema operativo.

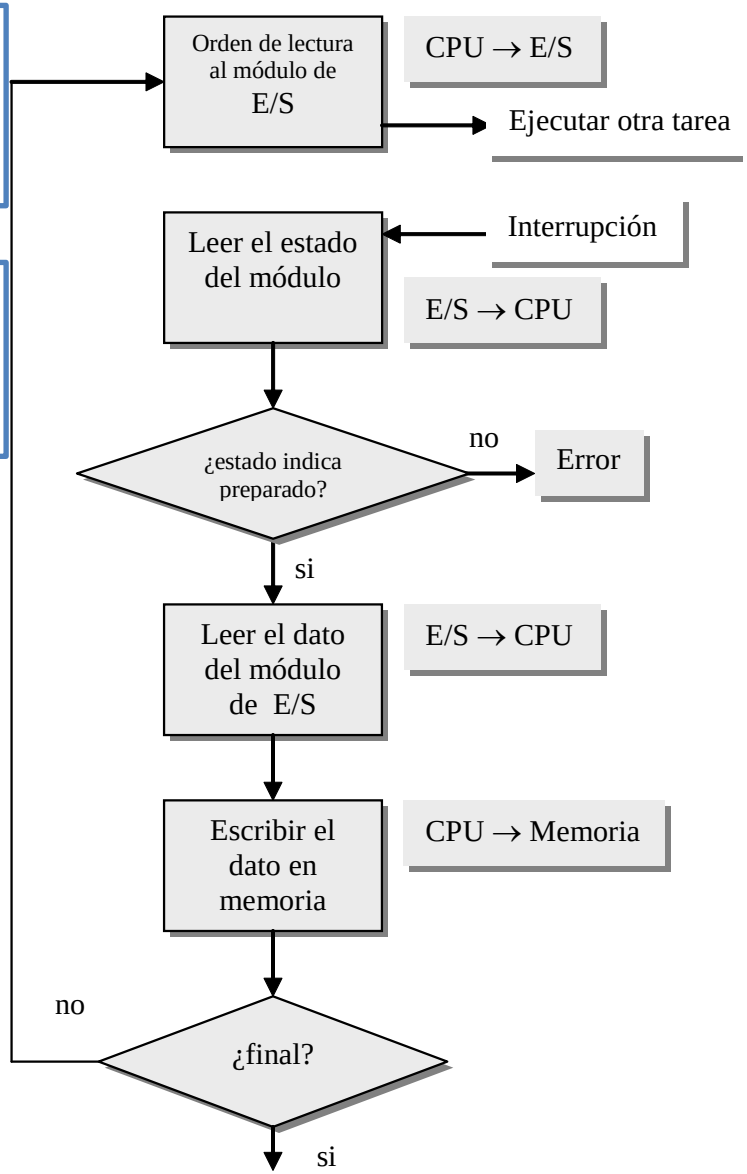
E/S mediante interrupción

La CPU hace la petición de operación de E/S y pasa a ejecutar otros programas.

➡ No existe bucle de espera

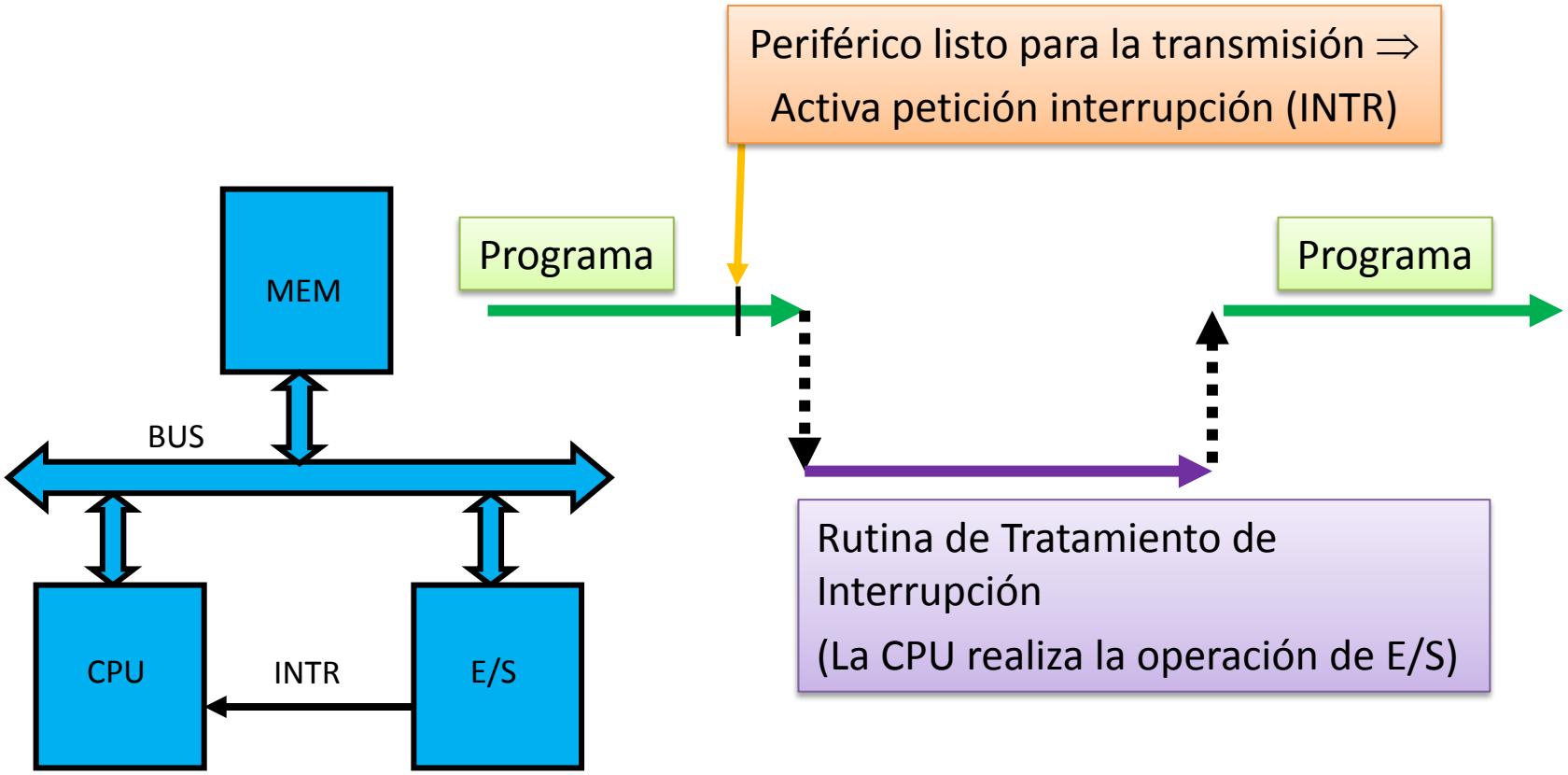
Cuando un periférico está listo para transmitir se lo indica a la CPU activando una LÍNEA DE PETICIÓN INTERRUPCIÓN

Cuando la CPU recibe una señal de petición de interrupción salta a una RUTINA DE TRATAMIENTO DE INTERRUPCIONES (RTI), que se encarga de atender al periférico que interrumpió y realizar la operación de E/S



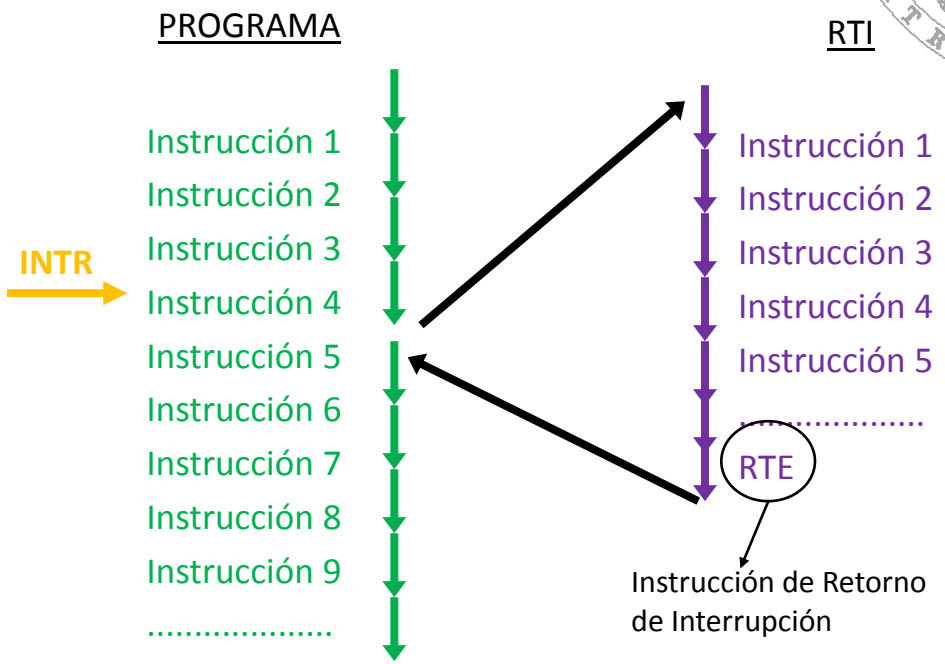
E/S mediante interrupción

- Flujo de ejecución de instrucciones cuando se produce una interrupción:



Rutina de tratamiento de interrupción

- 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



- 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 todos los registros que usa la RTI y restaurarlos al retornar de la RTI

Secuencia de eventos en el tratamiento de una interrupción

1. El programa activa el sistema de interrupciones
2. Periférico activa petición interrupción (INTR)
3. CPU termina de ejecutar la instrucción en curso y **comprueba si hay interrupciones pendientes**

4. CPU salva automát. contexto (PC y reg. de estado), **descapacita las interrupciones** y **salta a la RTI**
5. CPU ejecuta la RTI, durante la cual realiza la operación de E/S
6. CPU capacita las interrupciones, restaura contexto (PC y reg. estado) y retorna al programa
7. CPU continúa con la ejecución del programa

Secuencia de eventos en el tratamiento de una interrupción

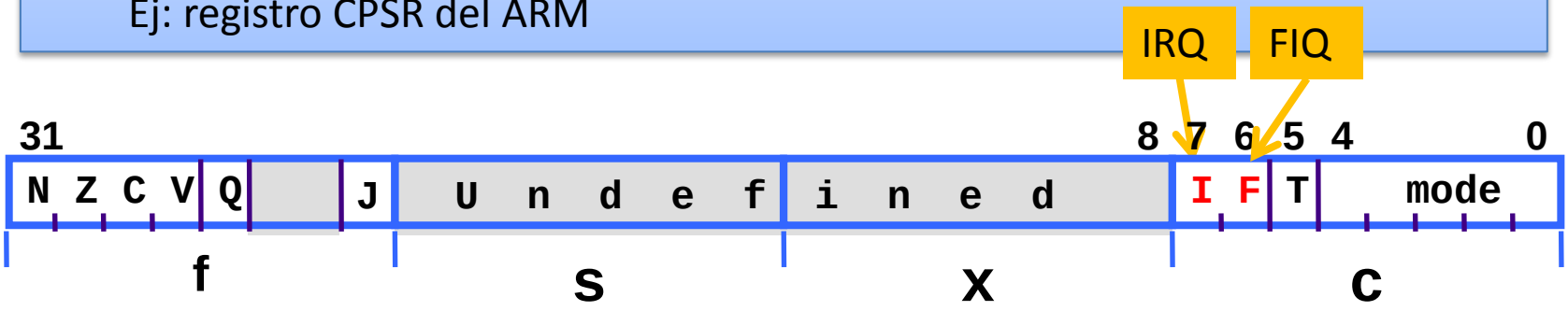
1 El programa activa el sistema de interrupciones

Para que pueda producirse una interrupción la CPU debe permitir que un dispositivo le interrumpa, **capacitando las interrupciones**:

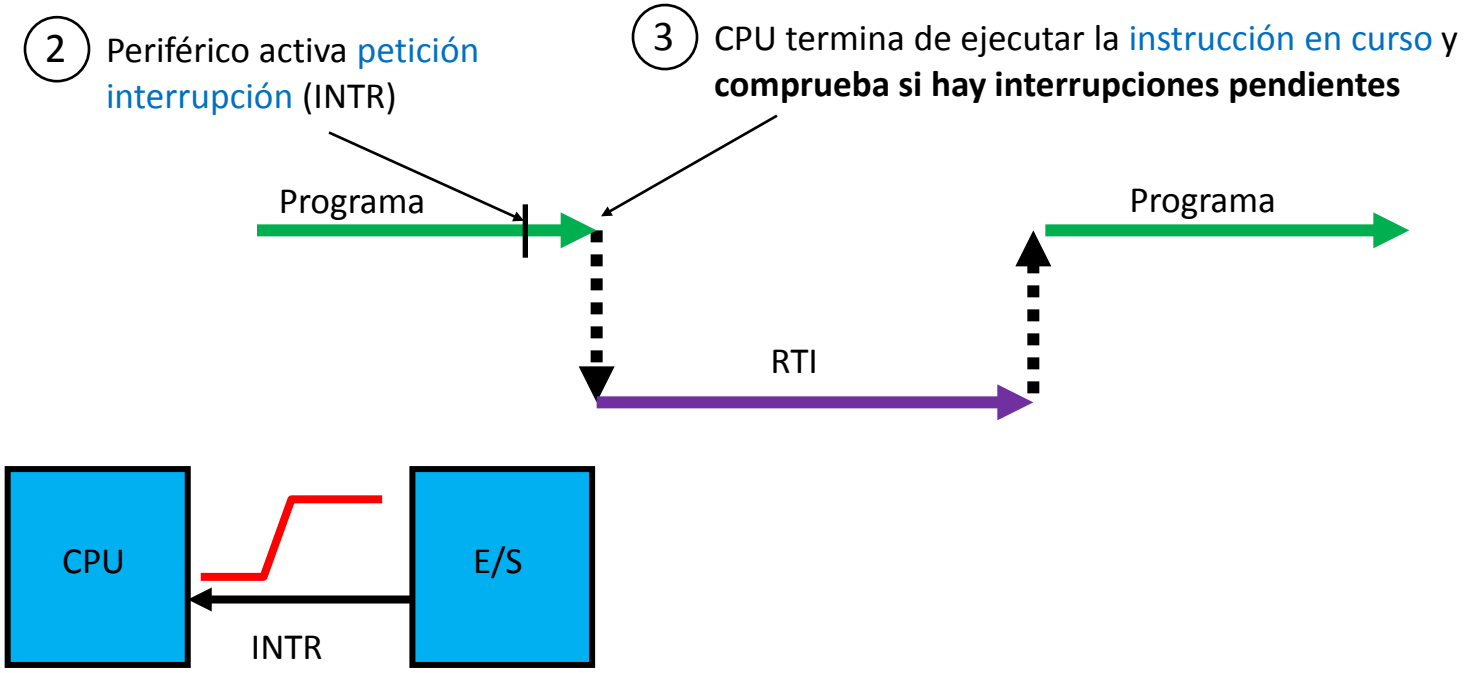
Localmente: se indica al controlador del dispositivo que puede generar una señal de interrupción.

Globalmente: en el registro de estado de la CPU se activa el bit correspondiente a la línea de interrupción.

Ej: registro CPSR del ARM



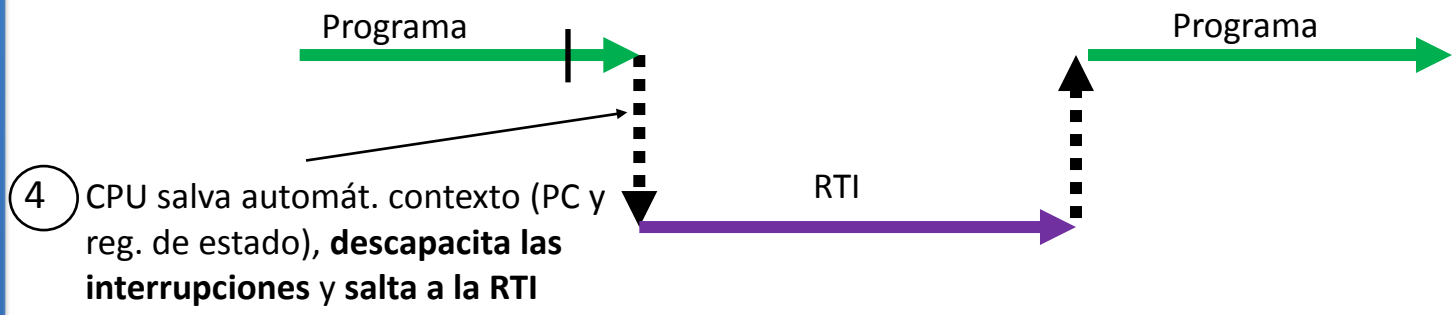
Secuencia de eventos en el tratamiento de una interrupción



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.
  - Salvo para:
    - Instrucciones de larga duración
      - Por ejemplo, en instrucción de movimiento múltiple (STM), se comprueba si hay interrupciones pendientes después de mover cada una de las palabras
    - Interrupciones muy prioritarias
      - Por ejemplo, una interrupción por fallo de página, en la que hay que acceder a disco para traer los operandos en memoria de la instrucción

Secuencia de eventos en el tratamiento de una interrupción



Salvar el estado:

El procesador guarda **automáticamente** el contexto del programa en ejecución (PC y registro de estado) en una zona de la pila distinta de la del programa o en registros especiales.

Descapacitar las interrupciones:

En el registro de estado se inhabilitan **automáticamente** las interrupciones por la línea INTR.

¿Por qué?

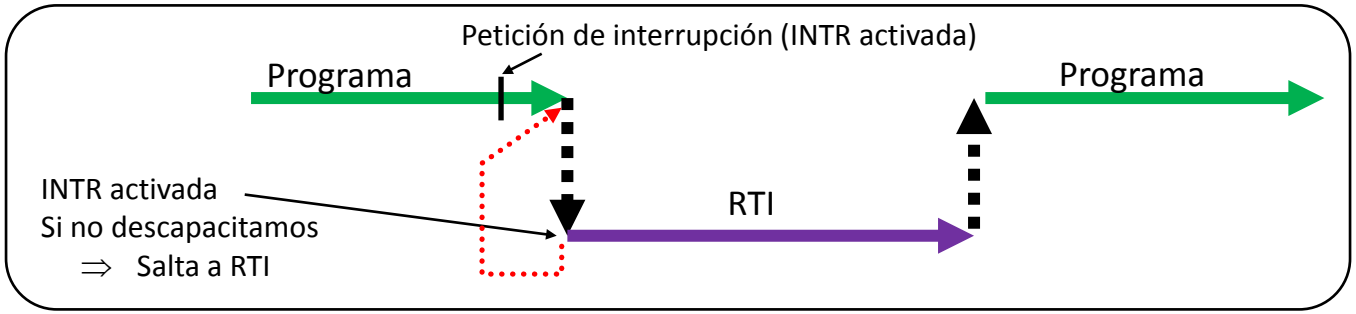
Saltar a RTI:

Se obtiene la dirección de la RTI que corresponda al periférico que ha interrumpido.

¿Cómo se identifica al periférico? Lo veremos en el siguiente punto

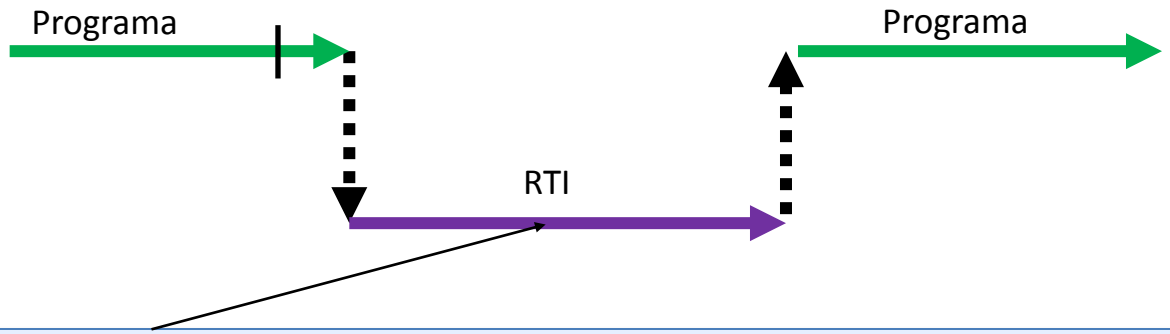
Inhibición o descapacitación de las interrupciones

- Antes de saltar a la RTI es necesario inhibir o descapacitar 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



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

Secuencia de eventos en el tratamiento de una interrupción



5 CPU ejecuta la RTI, durante la cual:

- Salva en pila todos los registros que utiliza (manual)
- Informa al periférico que se ha reconocido su interrupción

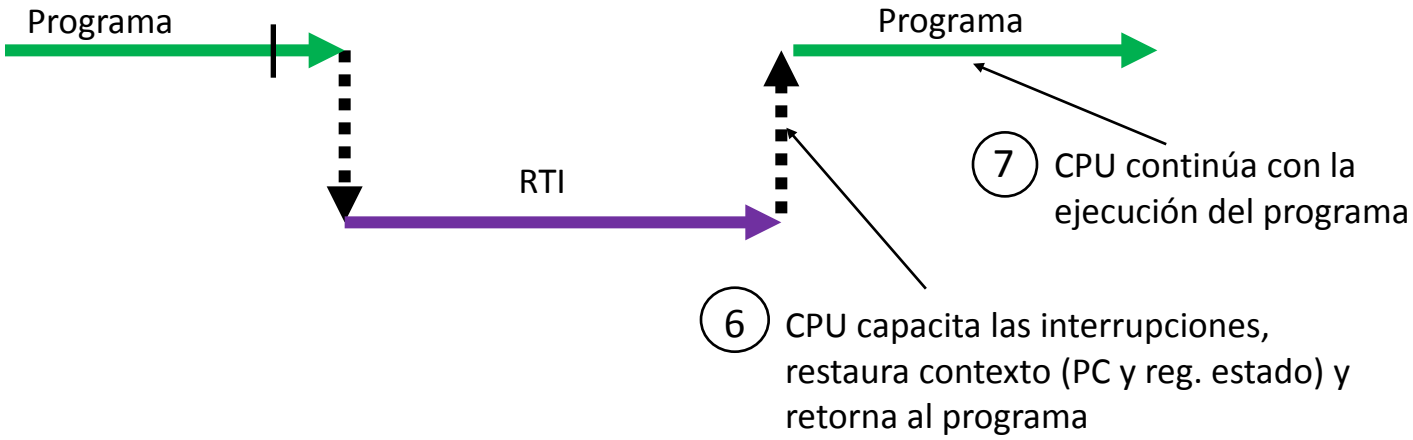
¿Cómo?

Por software: accediendo al registro de estado o de datos del controlador
Por hardware: activando una señal de reconocimiento de interrupción (INTA)

⇒ El periférico desactiva INTR

- Realiza la operación de E/S con el periférico
- Restaura los registros de datos/direcciones
- Ejecuta la instrucción de retorno de interrupción (RTE)

Secuencia de eventos en el tratamiento de una interrupción

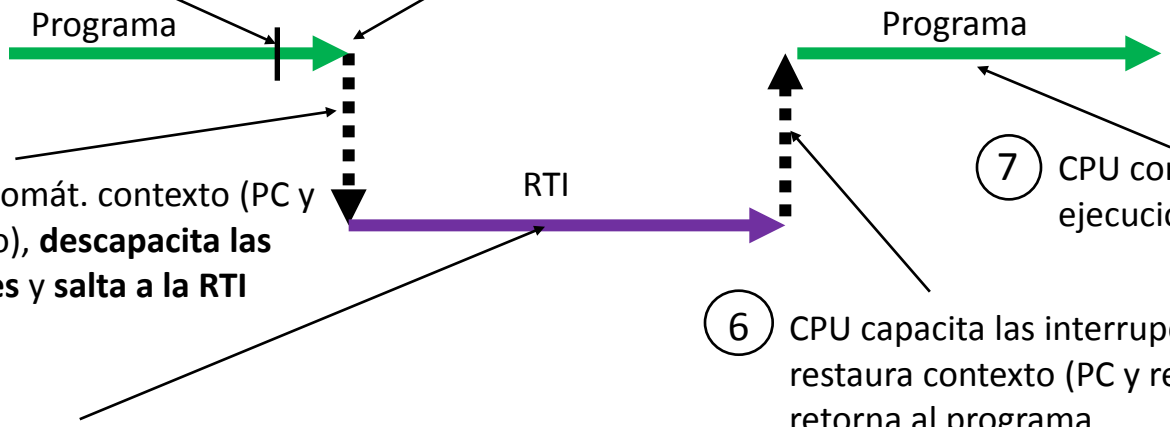


Cuestiones planteadas

- ¿Cómo se informa al periférico que se ha reconocido su interrupción?
- ¿Cómo se identifica la fuente de interrupción y, por tanto, la RTI que hay que ejecutar?
- ¿Qué ocurre si se produce una segunda interrupción durante la ejecución de la RTI?
  - ⇒ *Interrupciones multinivel y anidamiento de interrupciones*

Secuencia de eventos en el tratamiento de una interrupción

1. El programa activa el sistema de interrupciones
2. Periférico activa petición interrupción (INTR)
3. CPU termina de ejecutar la instrucción en curso y **comprueba si hay interrupciones pendientes**



4. CPU salva automát. contexto (PC y reg. de estado), **descapacita las interrupciones** y **salta a la RTI**
5. 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)
6. CPU capacita las interrupciones, restaura contexto (PC y reg. estado) y retorna al programa
7. CPU continúa con la ejecución del programa

Cuestiones planteadas

- ¿Cómo se informa al periférico que se ha reconocido su interrupción?
- ¿Cómo se identifica la fuente de interrupción y, por tanto, la RTI que hay que ejecutar?
- ¿Qué ocurre si se produce una segunda interrupción durante la ejecución de la RTI?
  ⇒ *Interrupciones multinivel y anidamiento de interrupciones*

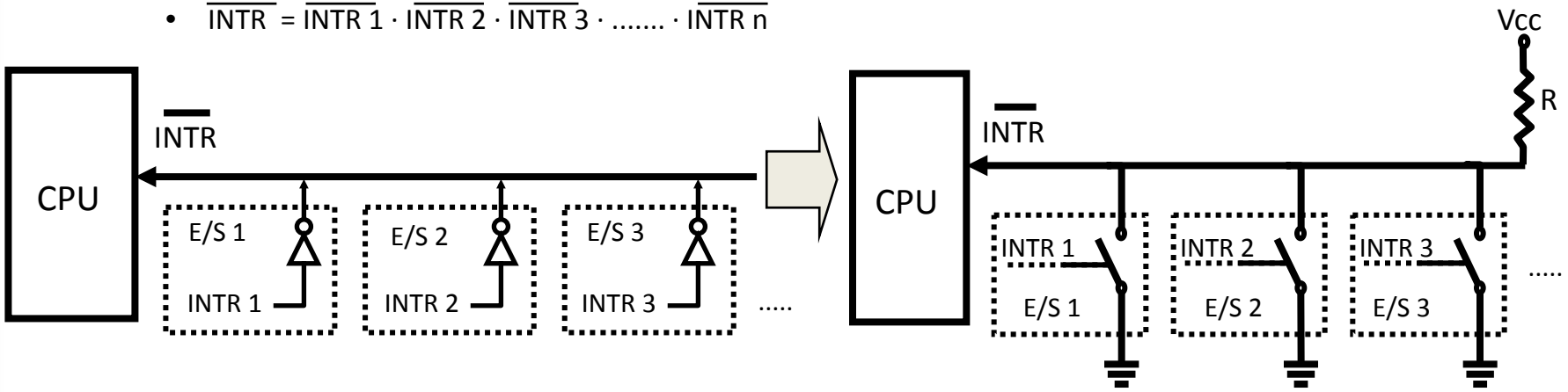
Identificación de la fuente de interrupción

- La CPU dispondrá de una serie de líneas de interrupción
  - Pueden existir prioridades entre las diferentes líneas
  - Ej: en el ARM Reset es la más prioritaria y FIQ es más prioritaria que IRQ.
- La activación de una de esas líneas, supone un salto a una dirección de memoria (habitualmente, prefijada)
  - No vectorizadas → polling
  - Interrupciones vectorizadas
- Ej: en el ARM del laboratorio
  - Si se activa IRQ el procesador bifurca a 0x18

	Memoria
0x1C	salto a FIQ
0x18	salto a IRQ
0x14	(Reserved)
0x10	salto a Data Abort
0x0C	salto Prefetch Abort
0x08	salto Software Interrupt
0x04	salto Undef Instruction
0x00	salto a Reset

Identificación de la fuente de interrupción (2)

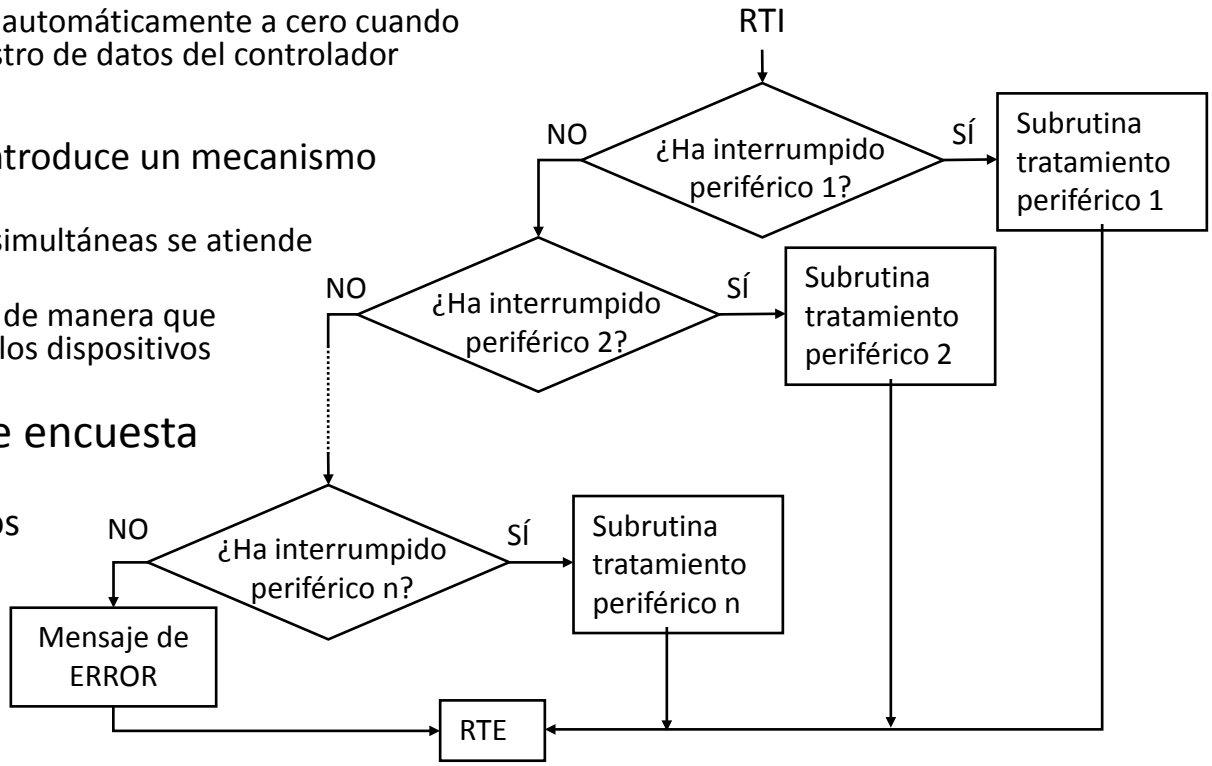
- A una misma línea de interrupción es posible conectar varios periféricos
  - Normalmente se utiliza lógica negativa (INTR*) y cableada (en colector abierto, “open collector”)
    - Si INTR* = 1 ⇒ No hay interrupción pendiente
    - Si INTR* = 0 ⇒ Sí hay interrupción pendiente
  - La señal INTR* se calcula como la Y lógica cableada de cada una de las líneas de petición de interrupción individuales:
    - $\overline{INTR} = \overline{INTR\ 1} \cdot \overline{INTR\ 2} \cdot \overline{INTR\ 3} \cdot \ldots\ldots \cdot \overline{INTR\ n}$



- 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

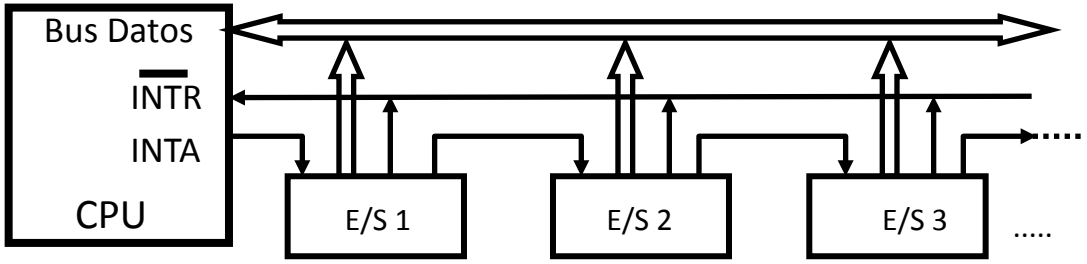
Identificación por encuesta (software)

- La RTI examina uno a uno los bits 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 para atender a ese periférico en cuestión
  - Durante la ejecución de esa rutina se debe desactivar el bit de petición de interrupción del periférico
    - Normalmente se pone automáticamente a cero cuando se lee o escribe el registro de datos del controlador
- 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
- Problemas del método de encuesta
  - Se desperdicia tiempo consultando a dispositivos que no han solicitado servicio



Interrupciones vectorizadas (hardware)

- El periférico que ha interrumpido envía un código o número de vector a la CPU a partir del cual se puede calcular la dirección de comienzo de la RTI de ese periférico particular
  - Cuando el periférico recibe una señal de confirmación o reconocimiento de interrupción INTA (“Interruption Ack.”) envía el nº de vector a través del bus de datos
  - A partir del nº de vector se calcula una dirección de memoria (vector) donde está almacenada la dirección de comienzo de la RTI



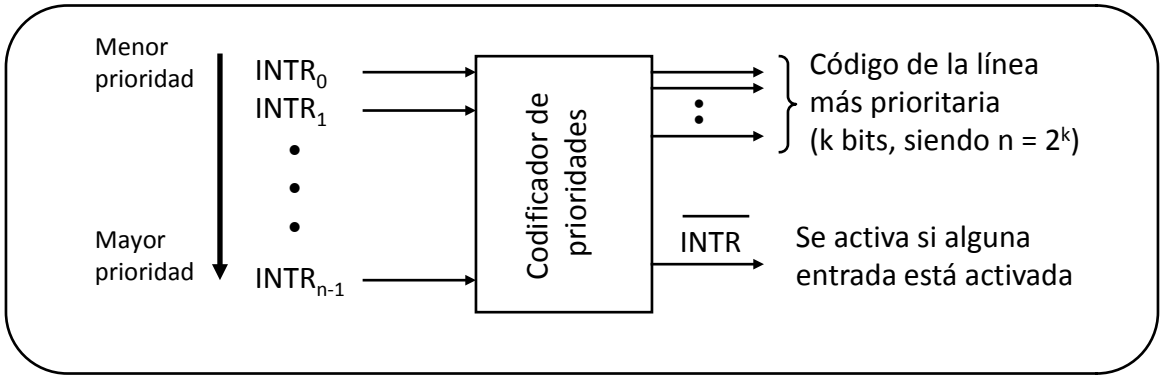
- Secuencia de eventos en el tratamiento de una interrupción vectorizada
  1. El periférico activa la señal de interrupción (INTR*=0)
  2. La CPU activa la señal de confirmación de interrupción (INTA=1) que se conecta a los dispositivos de forma encadenada (daisy-chain)
  3. Un periférico que no ha interrumpido, 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 (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

Interrupciones vectorizadas (hardware)

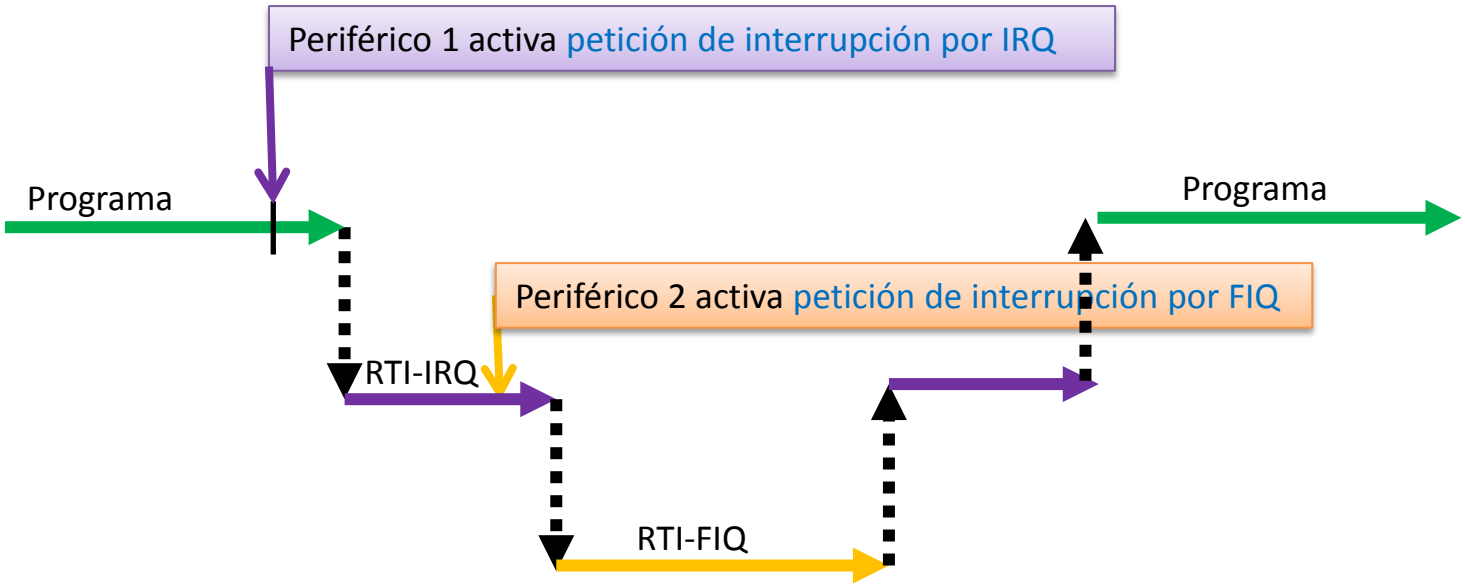
- Ventajas
  - La transmisión de INTA es totalmente hardware ⇒ es mucho más rápido que el método de encuesta
- Desventajas
  - El nº de dispositivos que se pueden identificar con este método depende del nº de bits que utilicemos para el nº vector
    - Ejemplo: con un nº de vector de 4 bits podemos identificar 16 dispositivos
  - Solución: pueden utilizarse códigos de grupo
    - 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

Interrupciones multinivel y anidamiento de interrupciones

- Interrupciones multinivel
  - Existen varias líneas o niveles de petición de interrupción
  - Cada nivel tiene asignada una prioridad distinta
  - A cada línea de interrupción se pueden conectar uno o varios dispositivos
- Resolución de conflictos de peticiones de interrupción simultáneas
  - 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

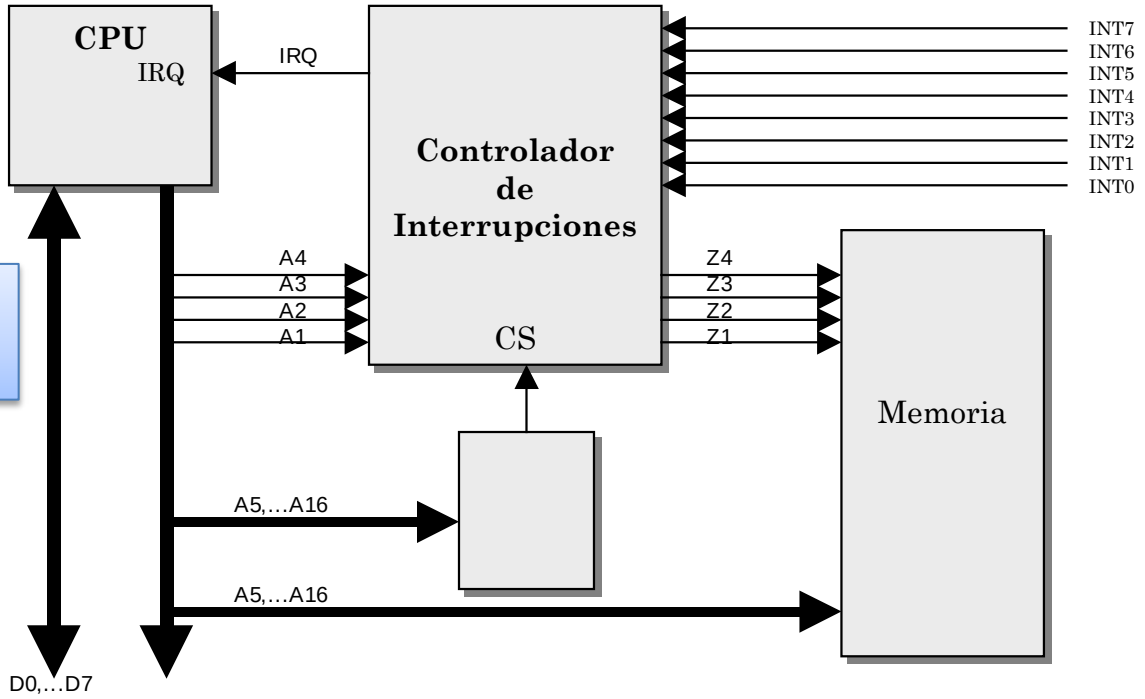


Ejemplo de anidamiento de interrupciones



Controlador de interrupciones

- Permite transformar una sola línea de interrupción (p. e., IRQ) en varias
  - Debe *interceptar* la dirección que se envía a memoria (por ejemplo…)
  - Si se ha producido una interrupción y la dirección que se presenta en el bus es la correspondiente al vector de IRQ, cambiar los bits necesarios en función de la interrupción recibida



Lo veremos en el laboratorio

Evaluación de la E/S mediante interrupciones

1. Ejemplo de periférico lento

- Procesador a 200 MHz (tiempo ciclo = 5 ns.); Ciclos medios 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 preparada para recibirlo
  - La impresora tarda 10 s en imprimir 10 Kbytes
  - **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 instruc. (salvar contexto, comprobar estado, transferir byte, restaurar contexto, RTE)
  - Para transferir 10 Kbytes tenemos que ejecutar 10.000 veces la RTI
    ⇒ hay que 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

Evaluación de la E/S mediante interrupciones (2)

2. Ejemplo de periférico rápido

- Procesador a 200 MHz (tiempo ciclo = 5 ns.); Ciclos medios 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 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 Mbytes
  - **La CPU está ocupada con la operación de E/S durante 1 s** (en ese tiempo la CPU podría haber ejecutado 100 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 que 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