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

ec

Indice

- Introducción
- Estructura y funciones del sistema de E/S
- E/S Programada
- E/S por interrupciones
- Bibliografïa:
  - 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.
  - S. Furber; ARM System-on-Chip architecture, 2ed. Addison-Wesley 2000.
  - Manuales del S3C44B0X y de la placa S3CEV40

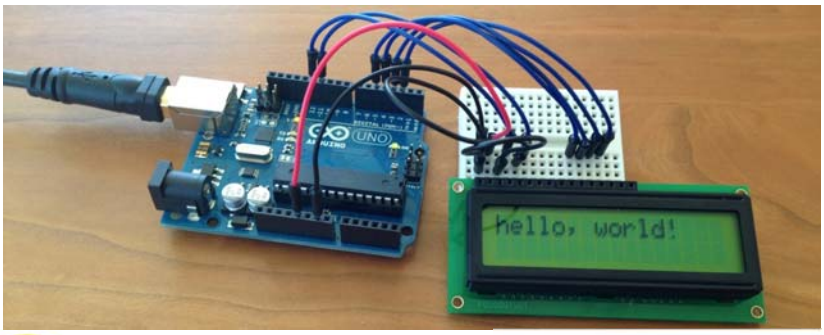
ec

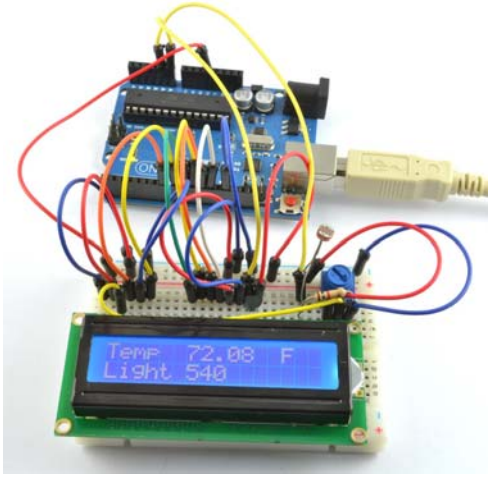
Indice

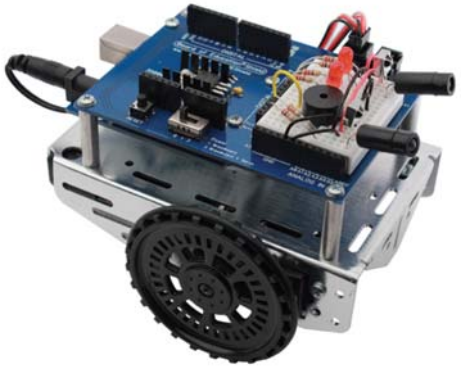
- **Introducción**
- Estructura y funciones del sistema de E/S
- E/S Programada
- Concepto de Interrupción
- E/S por interrupciones

ec

Conexión CPU <-> mundo exterior



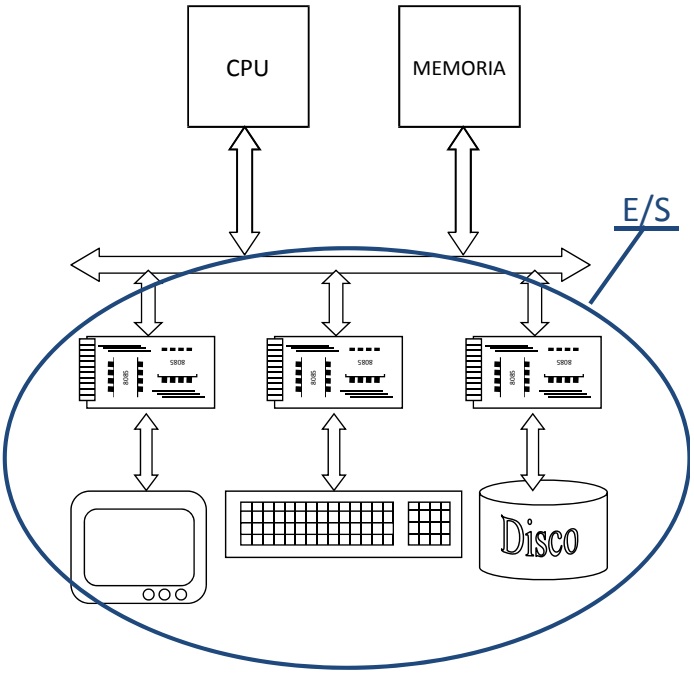




ec

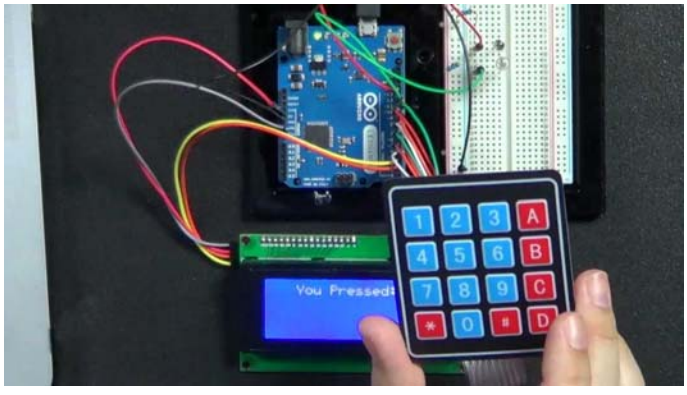
Funciones del Sistema de E/S

- Interacción CPU <-> mundo
- Entrada de datos
  - Desde dispositivo a CPU
  - Desde dispositivo a Memoria
- Salida de datos
  - Desde CPU a dispositivo
  - Desde Memoria a dispositivo
- En muchas ocasiones, el rendimiento del sistema de E/S determina el rendimiento global del sistema



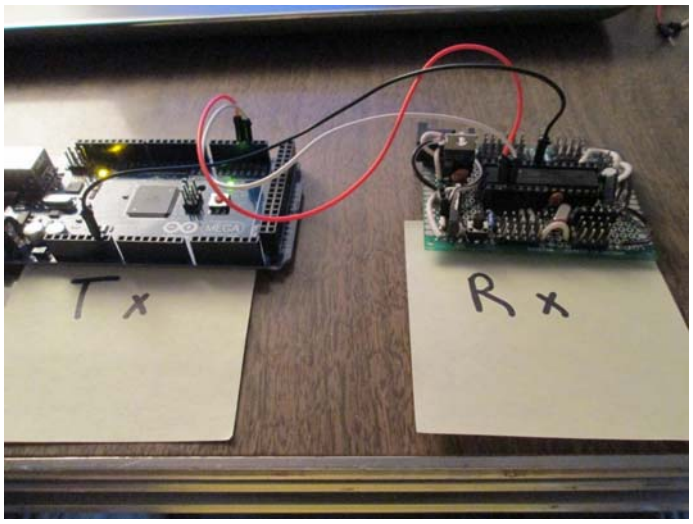
Tipos de periféricos

- Dispositivos de **presentación de datos**
  - Interaccionan con los usuarios, transportando datos entre éstos y la máquina
  - Ratón, teclado, pantalla, impresora, etc



Tipos de periféricos

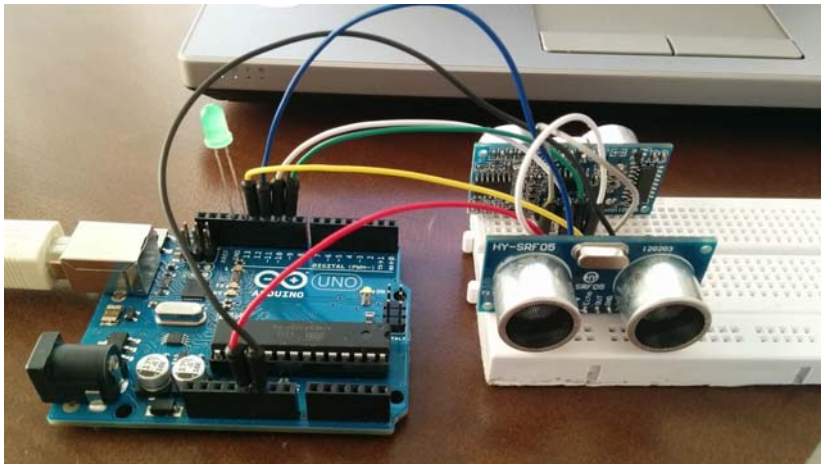
- Dispositivos de **comunicación** con otros procesadores
  - Permiten la comunicación con procesadores remotos a través de redes
  - Tarjeta de red, módem...



ec

Tipos de periféricos

- 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.



ec

Tipos de periféricos

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

ec

Distintas Características

- **Tipo de datos** que se transmiten
  - Bytes, bloques, tramas, etc
- **Tasa de transferencia**
  - **Ancho de banda:** cantidad de datos que se pueden transferir por unidad de tiempo (Mbit/s, MB/s…)
    - Dispositivos de almacenamiento/red -> gran ancho de banda
  - **Latencia:** tiempo requerido para obtener el primer dato (s, ms, us, ns)
    - Generalmente alta en comparación con velocidad de la CPU

ec

Distintas Características

Ancho de banda y latencia de algunos dispositivos periféricos

➢ Los dispositivos tradicionales de transporte y presentación de datos representan una carga relativamente baja de trabajo para el procesador :

Dispositivos	Ancho de banda
Sensores	1 Bps – 1 KBps
Teclado	10 Bps
Línea de comunicaciones	30 Bps – 200 KBps
Pantalla (CRT)	2 KBps
Impresora de línea	1 – 5 KBps
Cinta (cartridge)	0.5 – 2 MBps
Disco	4.5 MBps
Cinta	3-6 MBps

➢ Los dispositivos de E/S multimedia demandan mayores exigencias de velocidad:

Medio	Ancho de banda	Latencia max. permitida
Gráficos	1 MBps	1 - 5 segundos
Vídeo	100 MBps	20 milisegundos
Voz	64 KBps	50 - 300 milisegundos

ec

Conclusiones

- Necesitamos
  - Un interfaz distinto para periférico
    - Controlador de E/S del dispositivo
  - Distintas formas de interaccionar con ellos
    - Técnicas de E/S

ec

Indice

- Introducción
- **Estructura y funciones del sistema de E/S**
- E/S Programada
- E/S por interrupciones

ec

Estructura del sistema de E/S

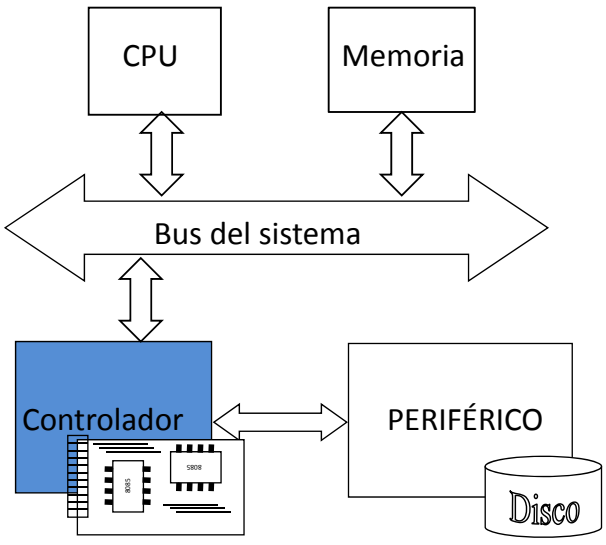
- Componentes:
  - Dispositivo periférico
  - Módulo de E/S controlador (hardware)
    - Tarjeta
  - Controlador software
    - Driver

S.O.

  - Canal de comunicación
    - P. ej. bus

TEMA SIGUIENTE

- Organización sencilla del sistema de E/S



ec

Componentes del sistema de E/S

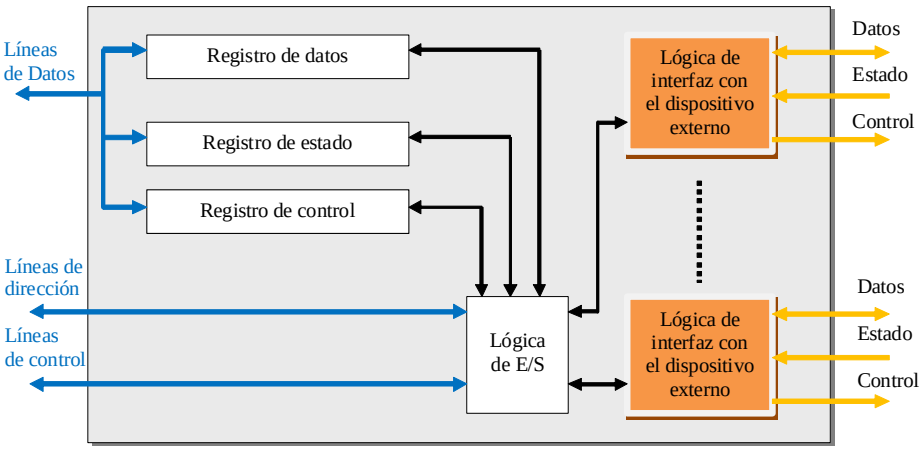
- **Dispositivo periférico**
  - Dispositivo en sí, de carácter mecánico, magnético…. (teclado, platos del disco..)
- **Controlador del dispositivo (hardware)**
  - Interfaz lógica que ofrece el dispositivo al sistema
  - Se *entiende* con el bus y conoce las características físicas del dispositivo
- **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

Controlador del dispositivo

- Interfaz entre el dispositivo y el procesador
- Funciones:
  - Control y temporización
    - Adapta la velocidad de transferencia
  - Comunicación con el procesador
  - Comunicación con el periférico
  - Buffering o almacenamiento intermedio
  - Detección de errores

Tiene dos interfaces

Estructura de un controlador E/S



- **Registro de datos**: intercambio de datos entre CPU y periférico
- **Registro de estado:** estado del dispositivo, ej. preparado, error, ocupado…
- **Registro de control:** actuación sobre el dispositivo (imprime carácter, lee un bloque…)

CUIDADO: NO son registros de la CPU. ¡¡¡Pueden estar en otro chip!!!

ec

Operación de E/S

- Una operación de entrada/salida se divide en:
  - **Petición**: enviar el comando (y dirección)
    - La CPU indica al dispositivo la operación que quiere hacer escribiendo en su *registro de control*
      - **¿Cómo seleccionar el dispositivo de E/S deseado?** Depende de que sea E/S aislada o mapeada en memoria
  - **Envío**: transferencia de los datos
    - La CPU leerá/escribirá de/al *registro de datos* del controlador del periférico deseado
      - **¿Cómo saber cuándo? SINCRONIZACIÓN**

ec

Direccionamiento de controladores

- **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

Ejemplo (1)

- Controlador GPIO, gestiona pines multifunción del chip
- Para encender un led conectado a uno de los pines
  - Conocer las direcciones de memoria asignadas a sus registros (Puerto B):
    - Registro de control PCONB: 0x01D20008
    - Registro de datos PDATB: 0x01D2000C
  - Configurar los pines del controlador como salidas para poder escribir en los leds :
    - Escribir en el registro de control (PCONB) un '0'

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

  - Encender los leds :
    - Escribir en un registro de datos (PDATB)del GPIO el valor de la salida
      - » 0 enciende el led,
      - » 1 apaga el led

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

Ejemplo (2)

- **El GPIO** tiene un circuito electrónico que:
  - Permite dar al pin la tensión Vcc o GND en función de lo que se haya escrito en el registro de datos, sólo si en el registro de control se ha configurado como pin de salida
  - Permite leer el valor del pin

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**

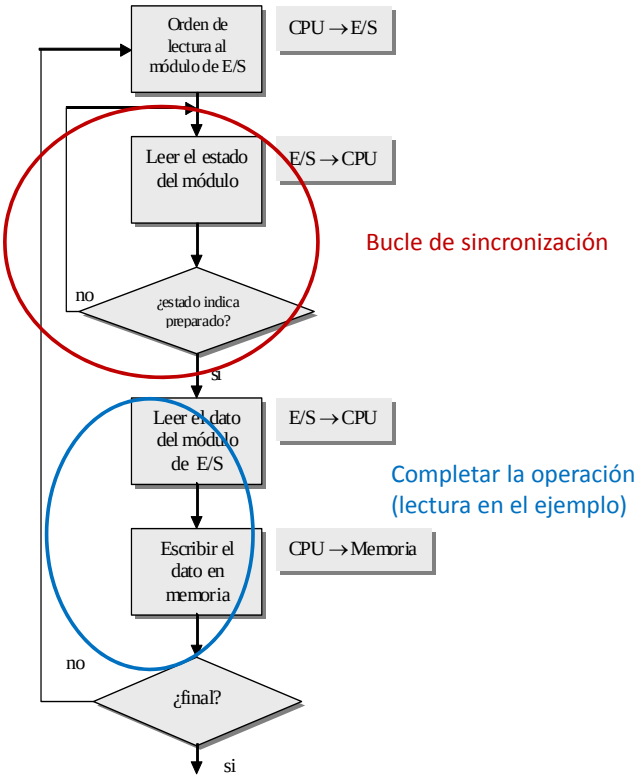
Indice

- Introducción
- Estructura y funciones del sistema de E/S
- **E/S Programada**
- E/S por interrupciones

ec

E/S Programada

- Cada vez que la CPU quiere realizar una transferencia:
  - 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

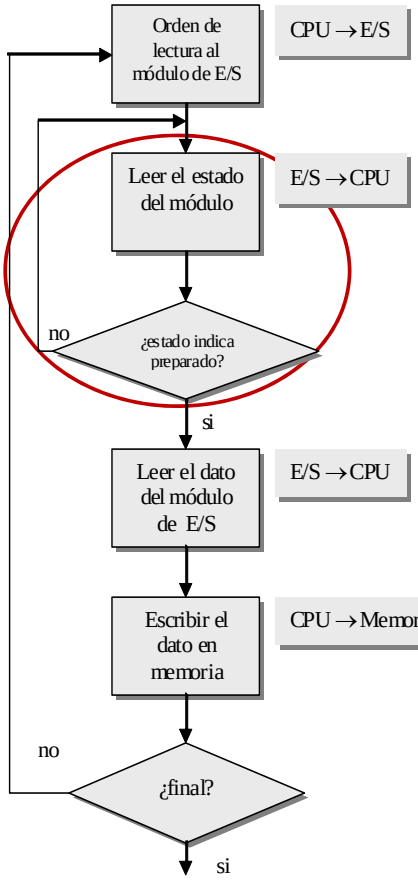


ec

Problemas de la E/S programada

- **La CPU no hace trabajo útil en el bucle de espera activa**
  - Con dispositivos lentos el bucle podría repetirse miles/millones de veces
  - La dinámica del programa se detiene durante la operación de E/S
    - Ejemplo: imaginar que en un vídeo-juego se detuviese la dinámica del juego a la espera de 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



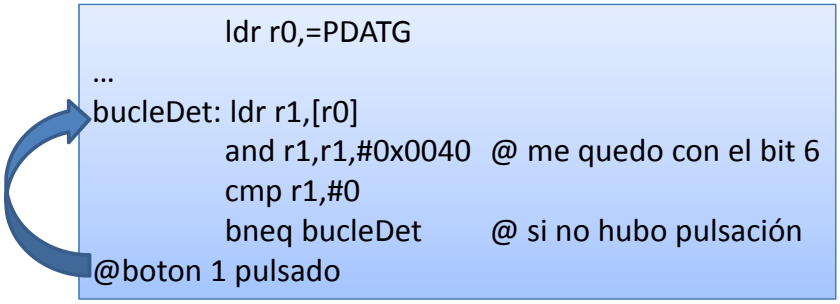
El bit 6 de PDATG es:
0 si el botón 1 ha sido pulsado y
1 si no ha sido pulsado

pseudocodigo

```
int  reg
reg = rPDATG
If ( bit 6 de reg) == 0)
        pulsado=SI
else
        pulsado=NO
```

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

Evaluación del ejemplo



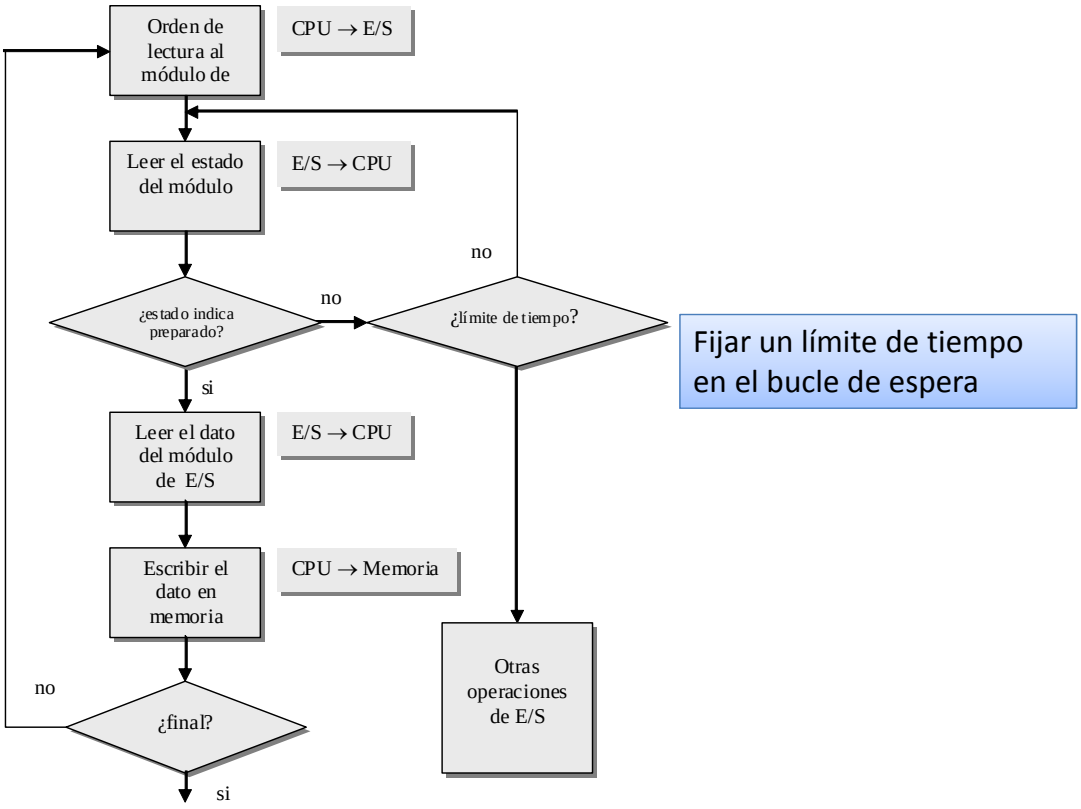
```
            ldr r0,=PDATG
...
bucleDet: ldr r1,[r0]
            and r1,r1,#0x0040   @ me quedo con el bit 6
            cmp r1,#0
            bneq bucleDet       @ si no hubo pulsación
@boton 1 pulsado
```

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?

40 x 1000000 ciclos/s / 12 ciclos/iteración = 3 millones de iteraciones/s

ec

Solución parcial



Fijar un límite de tiempo en el bucle de espera

ec

¿Se puede mejorar?

- **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**

Indice

Excepción

- **Evento inesperado** que **provoca un cambio en el flujo de control normal del programa**

1) La CPU **interrumpe la ejecución** de instrucciones

2) **Guarda información** para poder retomar la ejecución correctamente más tarde

3) Cambia a un ***estado especial*** para tratar la excepción

4) Pasa a ejecutar una **Rutina de Tratamiento de Interrupción** (RTI o *ISR*)

5) El retorno de la RTI **retomaría la ejecución del programa original**

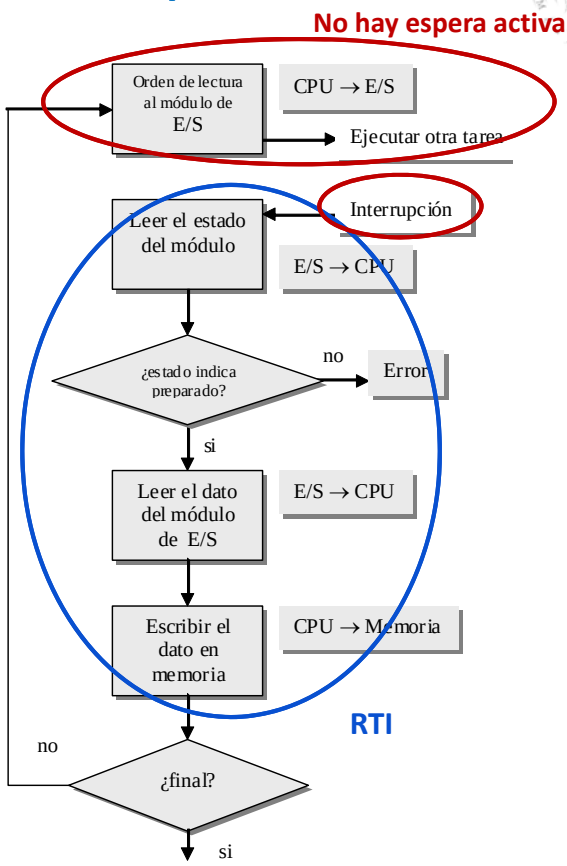
ec

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

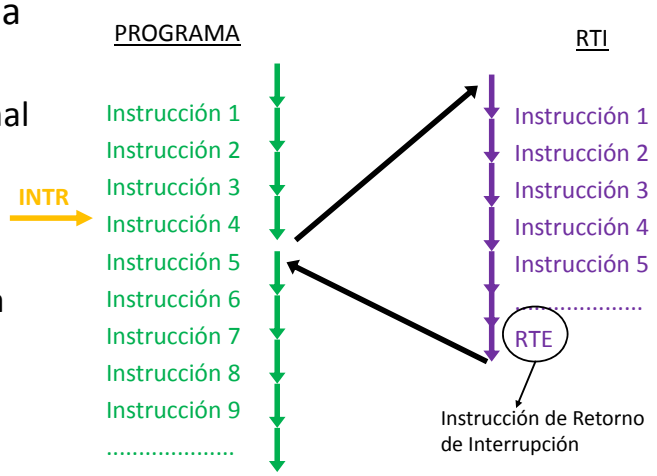
Cuando la CPU recibe una señal de petición de interrupción salta a una RUTINA DE TRATAMIENTO DE INTERRUPCIONES (RTI), que se encarga de atender al periférico que interrumpió y realizar la operación de E/S



ec

Subrutinas vs RTI

- **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
    - Por eso el HW guarda automáticamente 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

ec

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, swi):
  - Producidas por la ejecución de instrucciones especiales
  - Usadas por el SO para ofrecer servicios

ec

Excepción vs Interrupción

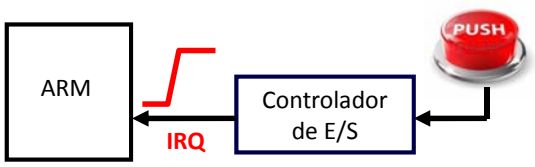
- **Interrupción**: excepción creada por una señal externa al procesador
  - Petición de algún dispositivo para ser atendido
  - Puede suceder en cualquier ciclo
  - El tratamiento a bajo nivel es más complejo que el de una excepción que se sabe en qué etapa del procesador va a ser detectada

Excepción vs Interrupción

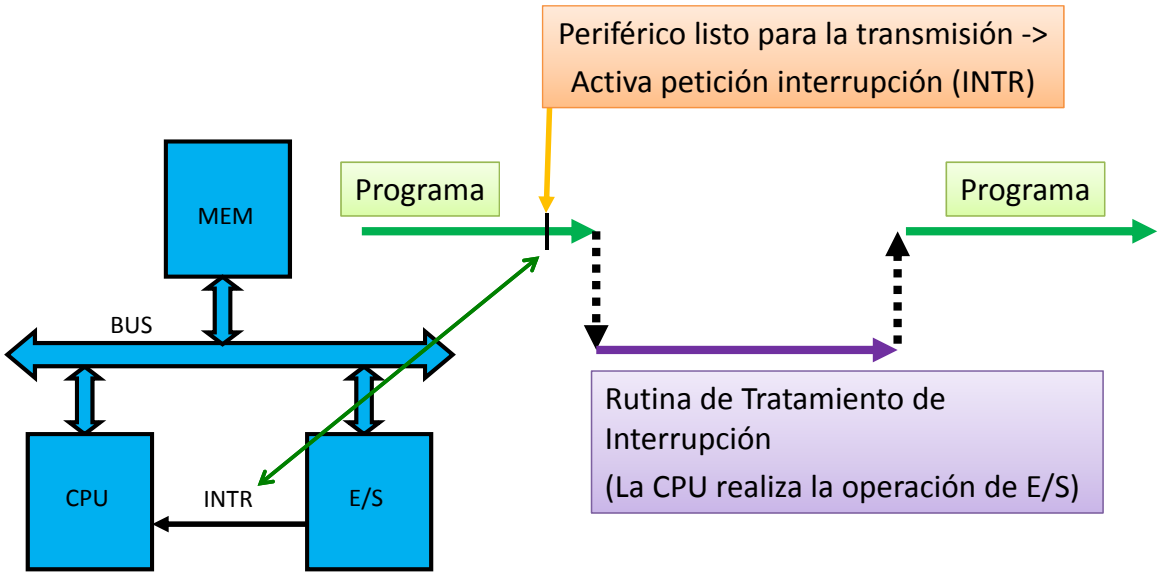
- Una **excepción interna** se produce debido a la ***ejecución (incorrecta) de instrucciones del programa***
  - Siempre que se ejecute esa instrucción se producirá la excepción
  - Ejemplo del ARM:
    - Si se intenta acceder a un dato de tamaño palabra usando una dirección que no es múltiplo de 4 se produce un DATA ABORT.
    - El procesador bifurca a la subrutina asociada a esa excepción ISR_Dabort .

```
ldr r0,=0x0a333333
str r1,[r0]  @ genera Data Abort
```

- Una **interrupción** se produce debido a una ***señal externa al procesador***
  - Es un evento asíncrono, que avisa al procesador de que necesita su atención
  - Ejemplo de interrupción debida a una operación de entrada/salida:
    - En el ejemplo de lectura de pulsadores se podría programar el controlador para que cada vez que se pulse el botón genere una interrupción por IRQ.

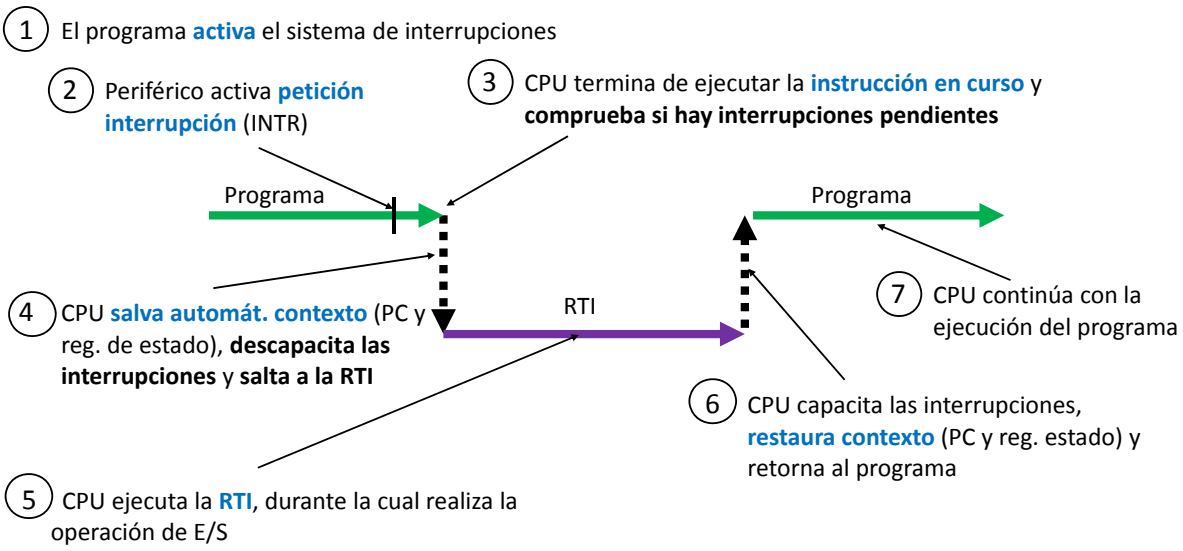


Interrupción: Flujo de ejecución



ec

Secuencia de eventos para la gestión de interrupciones

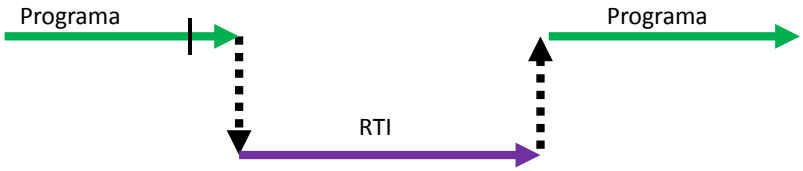


ec

Habilitación de interrupciones

Eventos en el tratamiento de una interrupción

① El programa **activa** el sistema de interrupciones

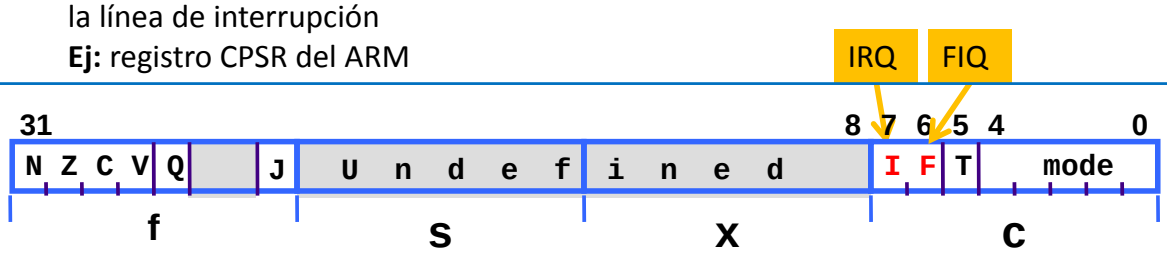


Para que pueda producirse una interrupción la CPU debe permitir que un dispositivo le interrumpa, **habilitando las interrupciones**:

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



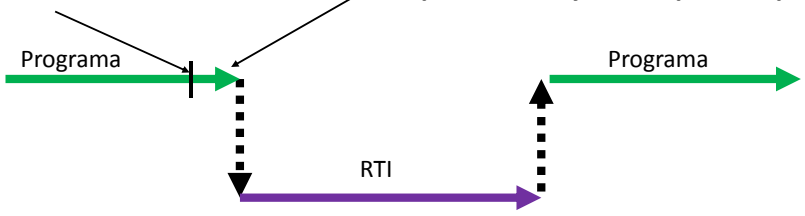
ec

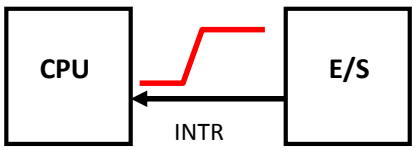
Comprobación de peticiones de interrupciones

Eventos en el tratamiento de una interrupción

② Periférico activa **petición interrupción** (INTR)

③ CPU termina de ejecutar la **instrucción en curso** y **comprueba si hay interrupciones pendientes**





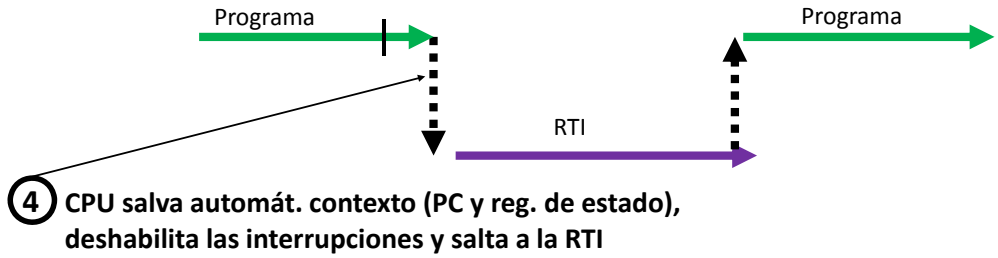
ec

Comprobación de peticiones de interrupciones

- 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

Salto a la RTI

Eventos en el tratamiento de una interrupción



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

Deshabilitar (enmascarar) las interrupciones:
En el registro de estado se enmascaran (deshabilitan) **automáticamente** las interrupciones por la línea INTR

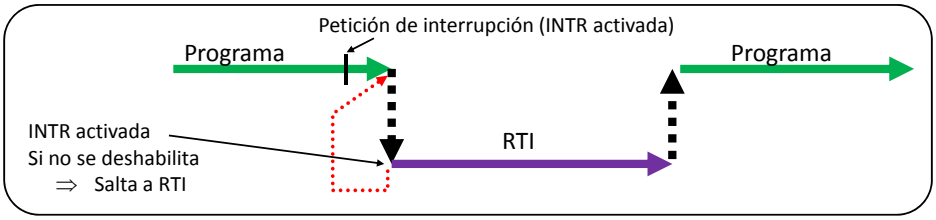
¿Por qué?

Saltar a RTI:
Se obtiene la dirección de la RTI que corresponda al periférico que ha interrumpido

¿Cómo se identifica al periférico?

Deshabilitar (enmascarar) las interrupciones

- **Antes de saltar a la RTI es necesario enmascarar las interrupciones**
  - **Motivo: Si no se enmascaran 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 habilitadas ⇒ 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

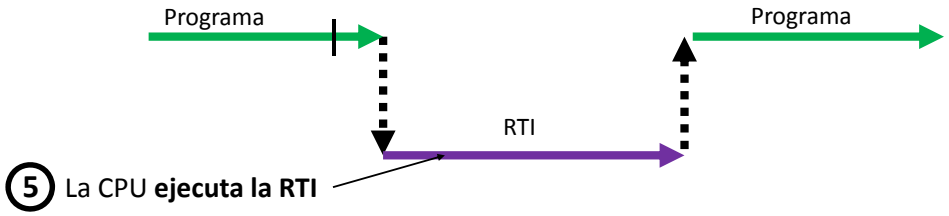


  - **Alternativas**
    - **Enmascaramiento global**
      - Se deshabilitan todas las interrupciones ⇒ ningún otro periférico podrá interrumpir durante la ejecución de la RTI
    - **Enmascaramiento selectivo**
      - Cuando hay varios niveles de interrupción se pueden deshabilitar las interrupciones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

ec

Ejecución de la RTI

Eventos en el tratamiento de una interrupción



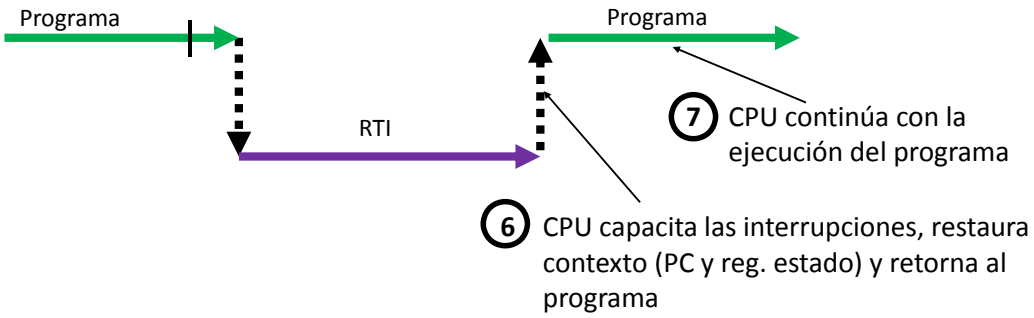
Responsabilidades de la RTI:

- Salvar en pila todos los registros que utiliza (manual)
  - Si va a invocar subrutinas, tiene que cumplir el estándar de subrutinas
- Informar al periférico que se ha reconocido su interrupción
  - **¿Cómo?**
    - ⇒Por **software**: borrar flag de petición en un registro de control
    - ⇒Por **hardware:** activando una señal de reconocimiento de interrupción (INTA)
  - **Una vez informado el periférico desactiva INTR**
- Realizar la operación de E/S con el periférico
- Restaura el contexto arquitectónico salvado
- Realizar el retorno de interrupción correcto

ec

Cuando acaba la ejecución de la RTI

Eventos en el tratamiento de una interrupción



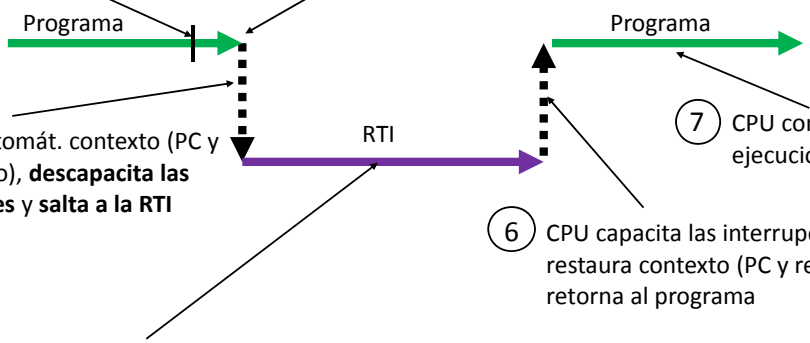
ec

Resumen de eventos para la gestión de interrupciones

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 el contexto arquitectónico salvado
- Realizar el retorno de interrupción correcto

ec

Cuestiones pendientes

- ¿Cómo se identifica la fuente de interrupción?
- ¿Cómo sabe la CPU qué 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

Indice

- Introducción
- Estructura y funciones del sistema de E/S
- E/S Programada
- **E/S por interrupciones**
  - Identificación de la fuente de interrupción y la RTI que hay que ejecutar
  - Interrupciones multinivel y anidamiento

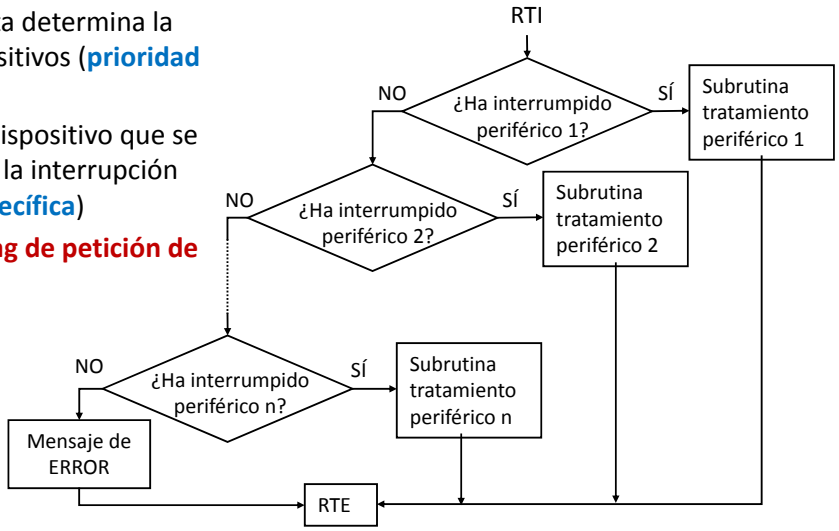
Identificación de la fuente de interrupción

- Muchos procesadores (ej ARM del lab) tienen **una única línea de petición de interrupciones (IRQ)**, por lo que a una misma línea tienen que conectarse 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:
    - INTR* = INTR 1* · INTR 2* · INTR 3* · ..... · INTR n*
- Es necesario un **mecanismo para identificar** al dispositivo que interrumpió
  - **Identificación software:** por encuesta (polling)
  - Identificación hardware: por vectores

ec

Identificación software: encuesta

- Ante cualquier petición se carga la instrucción de una posición fija **única para todos los dispositivos (ejecuta una RTI general)**
  - La RTI debe identificar por software que dispositivo solicitó la interrupción
    - Consulta en un orden los registros de control de los controladores HW de los dispositivos (**Encuesta**)
    - El orden de la encuesta determina la prioridad de los dispositivos (**prioridad software**)
    - Se atiende al primer dispositivo que se encuentre solicitando la interrupción (**se ejecuta su RTI específica**)
    - La RTI sólo **borra el flag de petición de ese dispositivo**



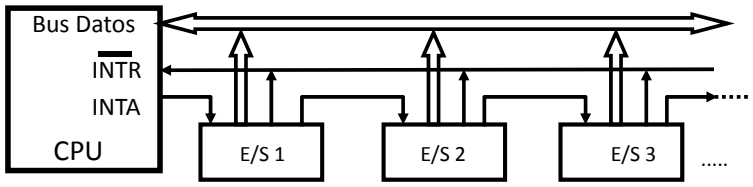
ec

Identificación software: encuesta

- **Ventajas**
  - Mecanismo sencillo
  - Fácil de extender
- **Desventajas**
  - RTI tiene que invertir tiempo en identificar la fuente
    - Puede ralentizar mucho el tratamiento de la interrupción (normalmente esto es crítico)
    - El problema se incrementa con el número de dispositivos conectados

Identificación HW

- **Ejemplo: Daisy Chain con INTA**



- El orden de conexión determina la prioridad
  - Los periféricos transmiten la señal INTA al siguiente si ellos no han solicitado interrupción
  - Cuando el periférico recibe INTA 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

Identificación HW

- **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

ec

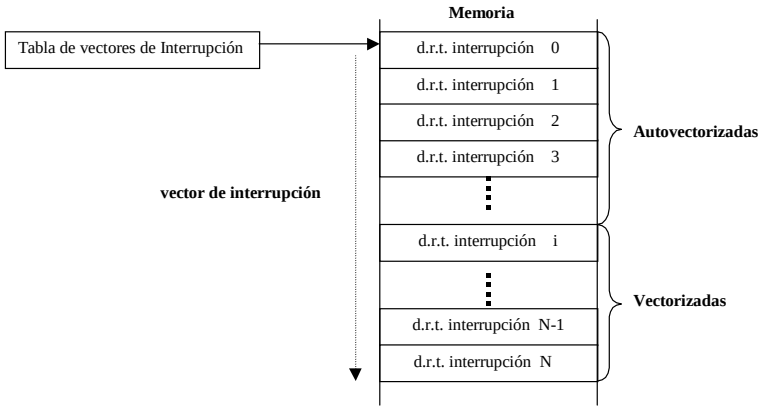
Dónde buscar la RTI

- **No vectorizadas :**
  - Varias líneas de petición de interrupción
  - Cada línea tiene su propio vector de interrupción (fijo para la línea)
- **Vectorizadas**
  - Una o más líneas de interrupción
  - Cada dispositivo vuelca en el bus su propio vector de interrupción
    - Puede ser también un controlador de interrupciones intermediario
  - Si hay varios dispositivos en una línea hay un mecanismo para decidir cuál puede volcar el vector en el bus

ec

Dónde buscar la RTI

- Cada computador dispone de una tabla en memoria con las direcciones de las rutinas de tratamiento asociadas a cada interrupción, denominada **tabla de vectores de interrupción**



- En función de la forma de obtener el vector de interrupción existen dos tipos de líneas de interrupción:
  - **Interrupciones no vectorizadas**: el vector de interrupción es fijo, una posición de memoria asociada a la línea de interrupción
  - **Interrupciones vectorizadas**: el vector de interrupción o parte de él lo suministra el propio periférico cuando se le reconoce la interrupción

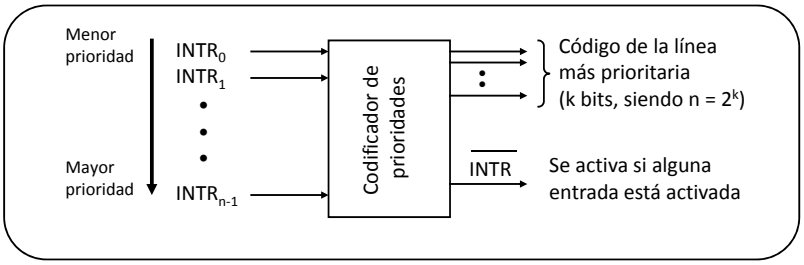
ec

Indice

- Introducción
- Estructura y funciones del sistema de E/S
- E/S Programada
- **E/S por interrupciones**
  - Identificación de la fuente de interrupción
  - Interrupciones multinivel y anidamiento

ec

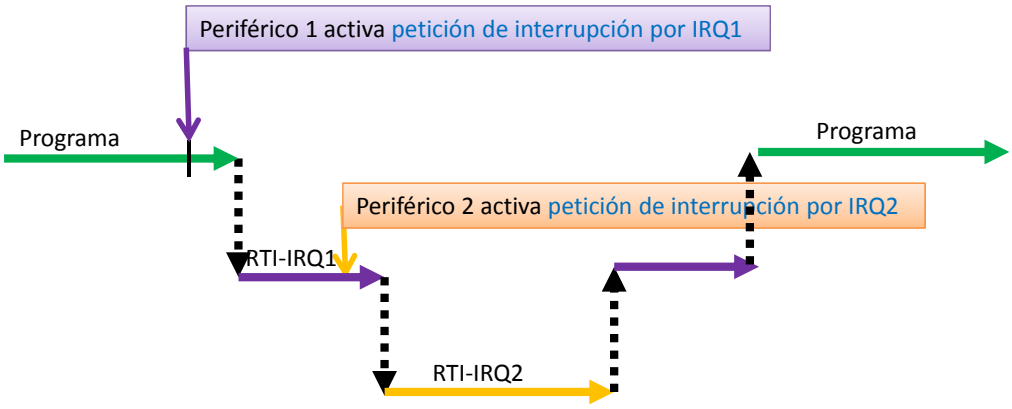
Interrupciones multinivel



- 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
    - Encuesta (software)
    - Vectores (hardware)
  - Peticiones simultáneas por líneas distintas
    - Codificador de prioridades
    - Se atiende a la línea más prioritaria

ec

Anidamiento de Interrupciones

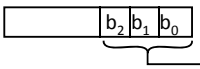


ec

Anidamiento de interrupciones

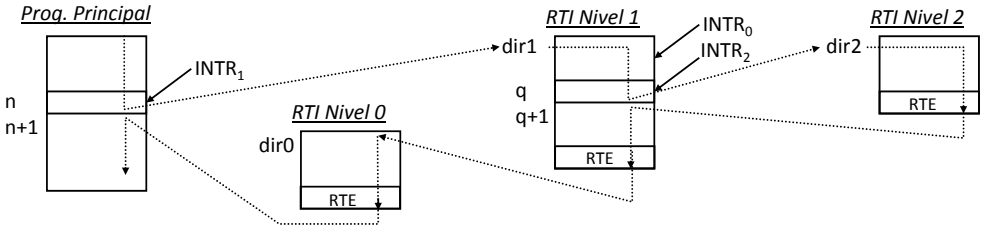
Sistema con 3 niveles de interrupción $\begin{cases} INTR_0 \\ INTR_1 \\ INTR_2 \end{cases}$ $(INTR_0 < INTR_1 < INTR_2)$

Registro de estado

SR: 

→ Bits de máscara $\begin{cases} \text{Si } b_k = 1 \rightarrow \text{Nivel } INTR_k \text{ capacitado} \\ \text{Si } b_k = 0 \rightarrow \text{Nivel } INTR_k \text{ enmascarado} \end{cases}$

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



Evolución de PC, SR y Pila