
# Práctica 2

Modos de ejecución, gestión de excepciones y entrada/salida mediante interrupciones

ec

# Excepciones


- La entrada salida mediante interrupciones es un tipo particular de excepción:
  - Una excepción es cualquier evento que, en lugar de continuar con la ejecución del programa en curso, para su ejecución y ejecuta otro programa (E*xceptionHandler*)
    - Si la excepción es causada internamente, la denominaremos **excepción** (ej. División por cero)
    - Si la excepción es causada externamente, lo denominaremos **interrupción** (ej. Se ha pulsado una tecla)

# Qué ocurre cuando se produce una excepción

**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 |

Cuando se produce una excepción el procesador salta automáticamente a estas direcciones

- Cuando se produce una excepción la CPU salta automáticamente a una dirección de memoria (vector) prefijada

# Excepciones vs Interrupciones

| Exception | Description |
|---|---|
| **Reset** | Occurs when the processor reset pin is asserted. This exception is only expected to occur for signalling power-up, or for resetting as if the processor has just powered up. A soft reset can be done by branching to the reset vector (0x0000). |
| **Undefined Instruction** | Occurs if neither the processor, or any attached coprocessor, recognizes the currently executing instruction. |
| **Software Interrupt (SWI)** | This is a user-defined synchronous interrupt instruction.It allows a program running in User mode, for example, to request privileged operations that run in Supervisor mode, such as an RTOS function. |
| **Prefetch Abort** | Occurs when the processor attempts to execute an instruction that was not fetched, because the address was illegal[1]. |
| **Data Abort** | Occurs when a data transfer instruction attempts to load or store data at an illegal address[a]. |
| **IRQ** | Occurs when the processor external interrupt request pin is asserted (LOW) and the I bit in the CPSR is clear. |
| **FIQ** | Occurs when the processor external fast interrupt request pin is asserted (LOW) and the F bit in the CPSR is clear. |

# Qué ocurre cuando se produce una excepción

| Dirección | Instrucción |
|---|---|
| 0x00000000 | b ResetHandler |
| 0x00000004 | b HandlerUndef |
| 0x00000008 | b HandlerSWI |
| 0x0000000C | b HandlerPabort |
| 0x00000010 | b HandlerDabort |
| 0x00000014 | b . |
| 0x00000018 | b HandlerIRQ |
| 0x0000001C | b HandlerFIQ |

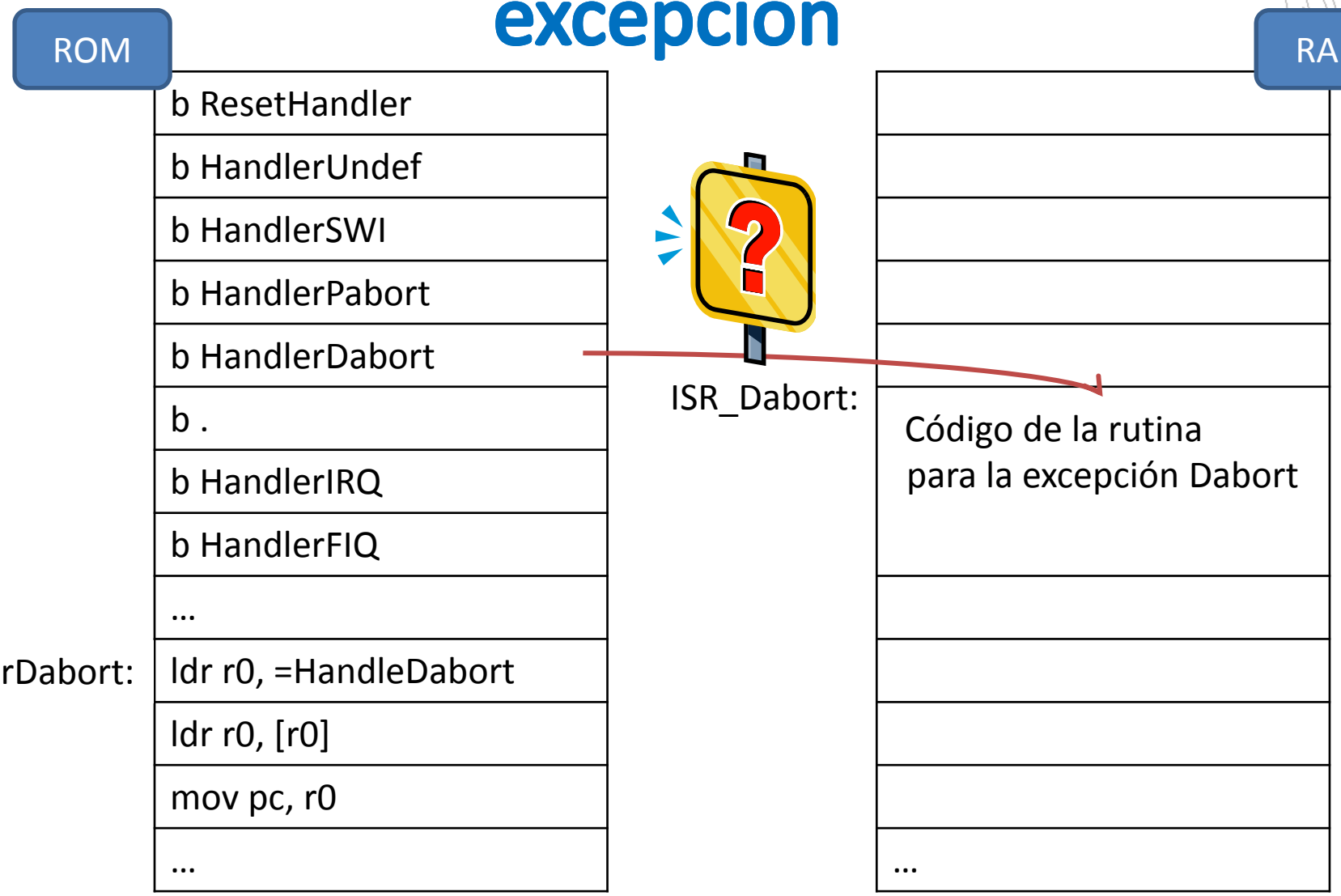
La rutina *HandlerDabort* calcula dónde está la subrutina *Dabort* (*ISR_Dabort*) mediante la lectura de una tabla de punteros a subrutinas.

Lo único que hay que hacer es almacenar en esa tabla el puntero a la subrutina

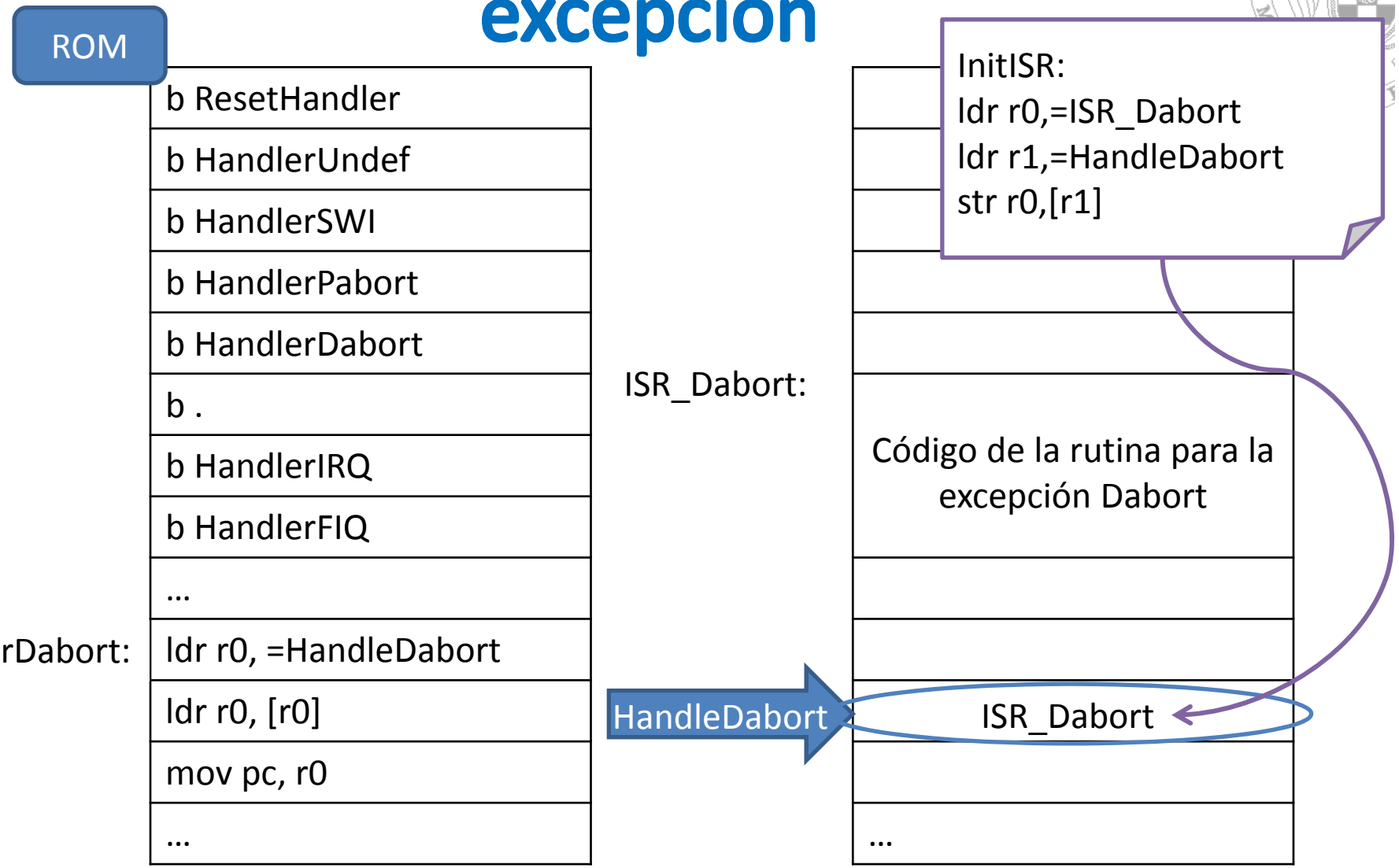
IntISR:

```
ldr r0,=ISR_Dabort
ldr r1,=HandleDabort
str r0,[r1]
```

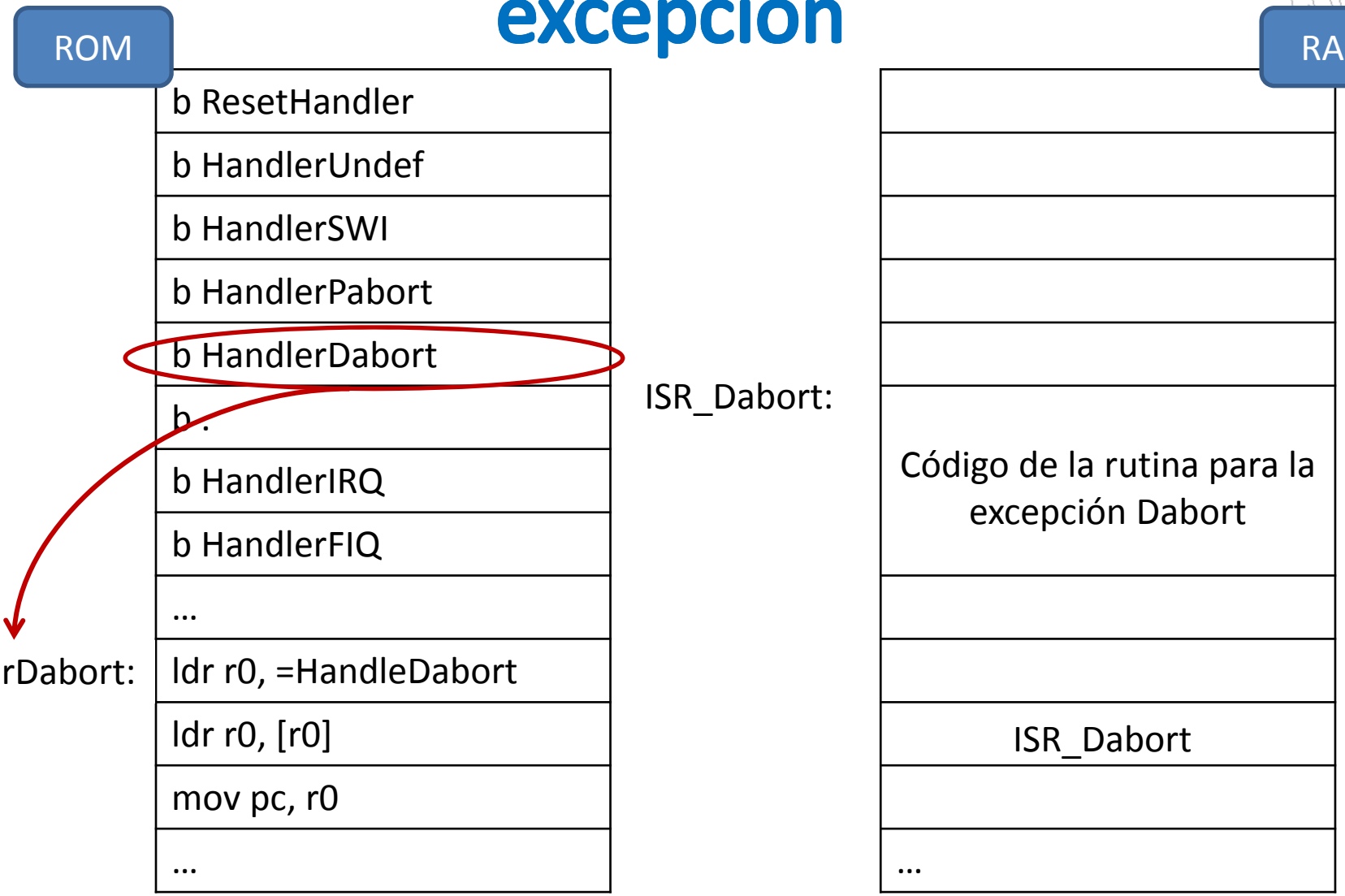
# Qué ocurre cuando se produce una excepción



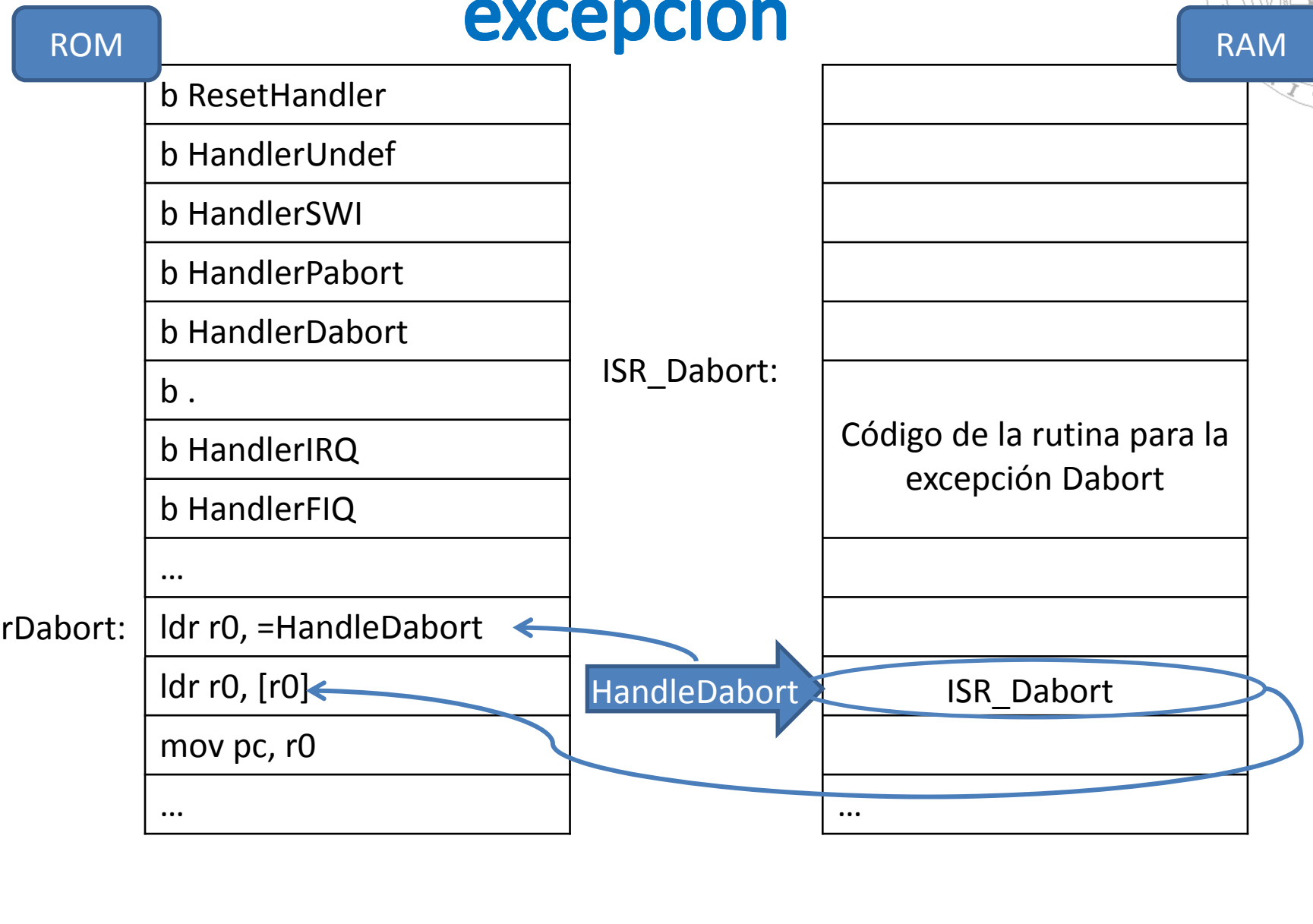
# Qué ocurre cuando se produce una excepción



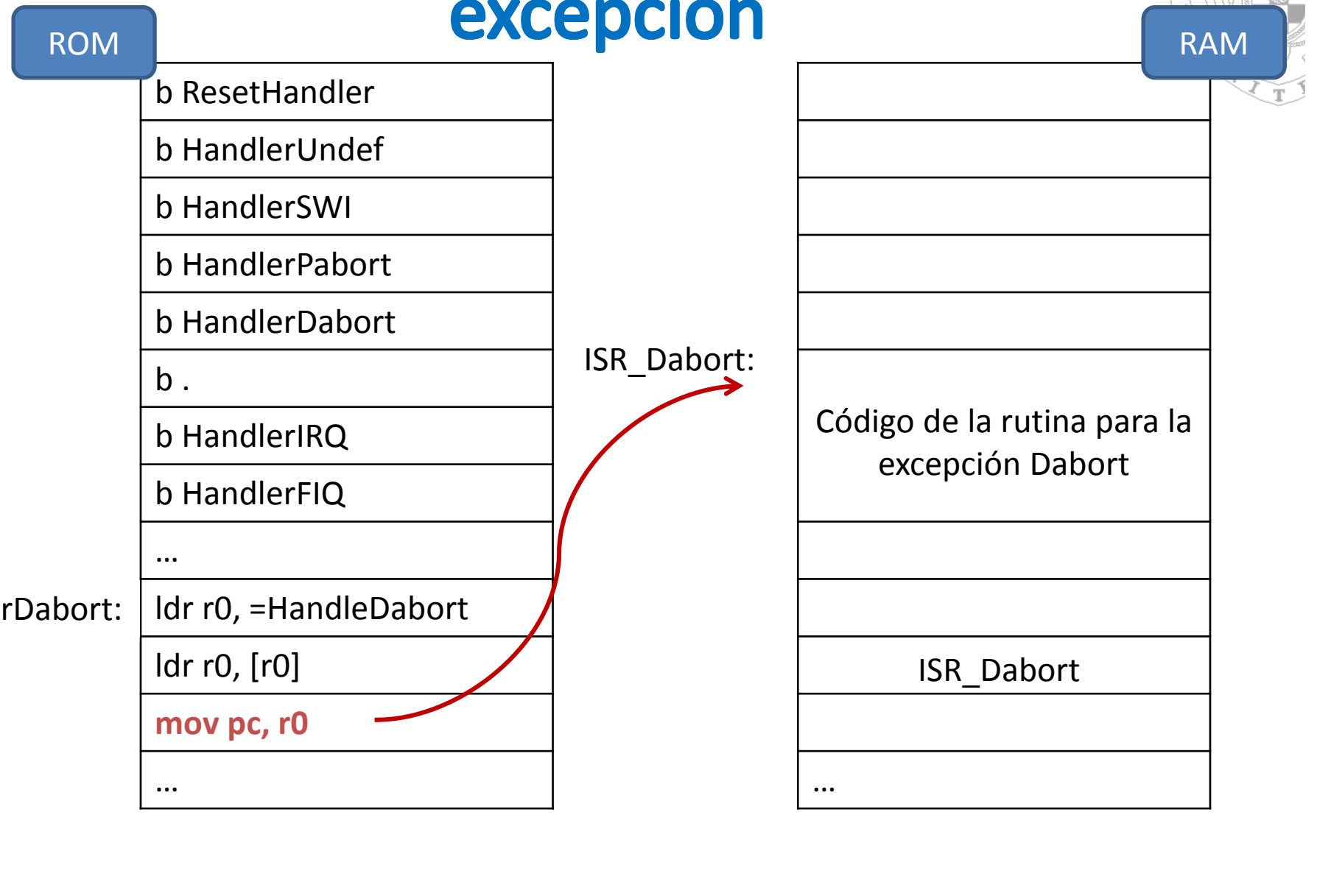
# Qué ocurre cuando se produce una excepción



# Qué ocurre cuando se produce una excepción



# Qué ocurre cuando se produce una excepción



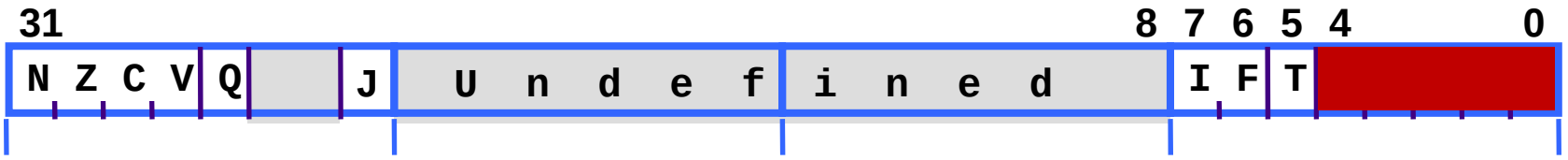
# Qué ocurre cuando se produce una excepción

- Los valores de HandlerExcepction (ROM) y HandleException (RAM) son fijos:

```
.equ    HandleReset, _ISR_STARTADDRESS
.equ    HandleUndef, _ISR_STARTADDRESS+4
.equ    HandleSWI, _ISR_STARTADDRESS+4*2
.equ    HandlePabort, _ISR_STARTADDRESS+4*3
.equ    HandleDabort, _ISR_STARTADDRESS+4*4
.equ    HandleReserved, _ISR_STARTADDRESS+4*5
.equ    HandleIRQ, _ISR_STARTADDRESS+4*6
.equ    HandleFIQ, _ISR_STARTADDRESS+4*7
```

# Qué ocurre cuando se produce uan excepción

- Cada excepción tiene un modo específico de trabajo (modo privilegiado):
  - Para saber en qué modo nos encontramos se leen los cinco últimos bits del registro de estado




  - Lectura: mrs ri, cprs
  - Escritura: msr cprs, ri

# Qué ocurre cuando se produce una excepción

- Modos de ejecución

```
.equ        USERMODE,     0x10
.equ        FIQMODE,      0x11
.equ        IRQMODE,      0x12
.equ        SVCMODE,      0x13
.equ        ABORTMODE,    0x17
.equ        UNDEFMODE,    0x1b
/-----------------------------------------------/
.equ        MODEMASK,     0x1f
```

- Secuencia de instrucciones que modifican el modo de ejecución (el registro de estado)
  - **MRS Ri, CPRS**

  //Aplicar máscara
  - **MSR CPSR, Ri**

# Qué ocurre cuando se produce una excepción

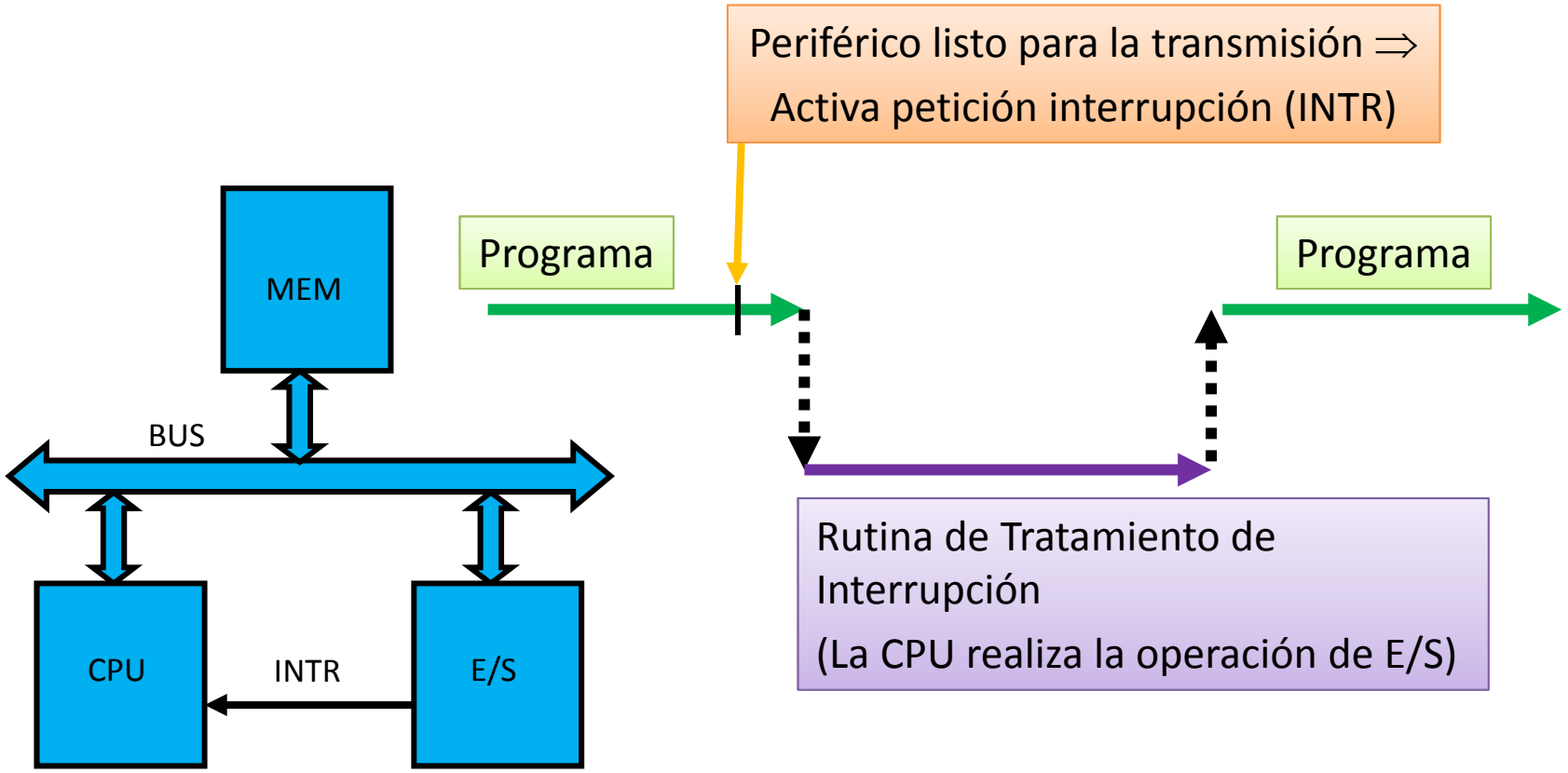
- Antes de ejecutar cualquier programa hay que incializar todas las pilas, por si durante la ejecución del programa se produce una excepción
- Inicialización de cada una de las pilas:

```
mrs     r0, cpsr
bic     r0,r0,#MODEMASK   @R0 = R0 and (NOT #MODEMASK)
orr     r1,r0,#Mode
msr     cpsr,r1
ldr     sp,=ModeStack
```

**InitStacks.asm**

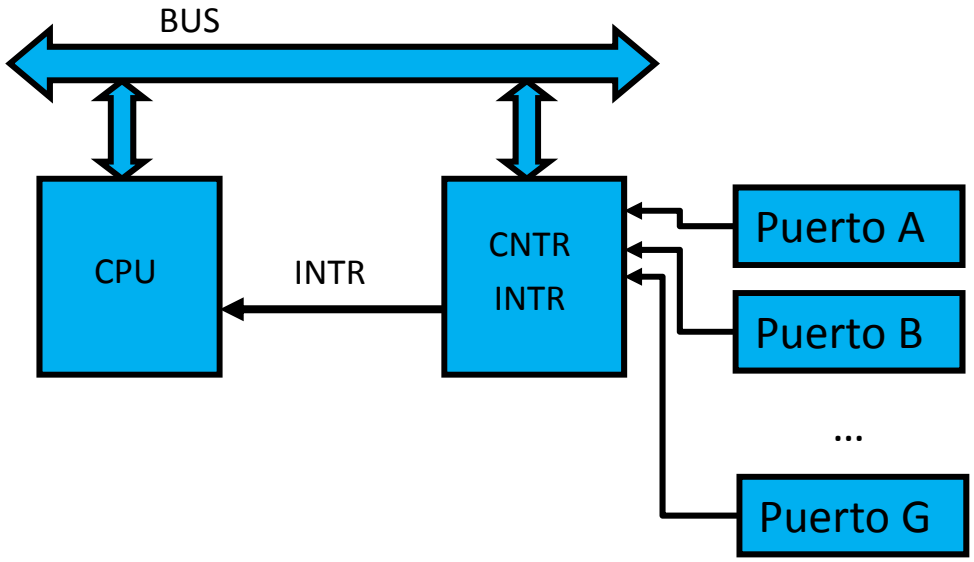
# E/S mediante interrupción

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

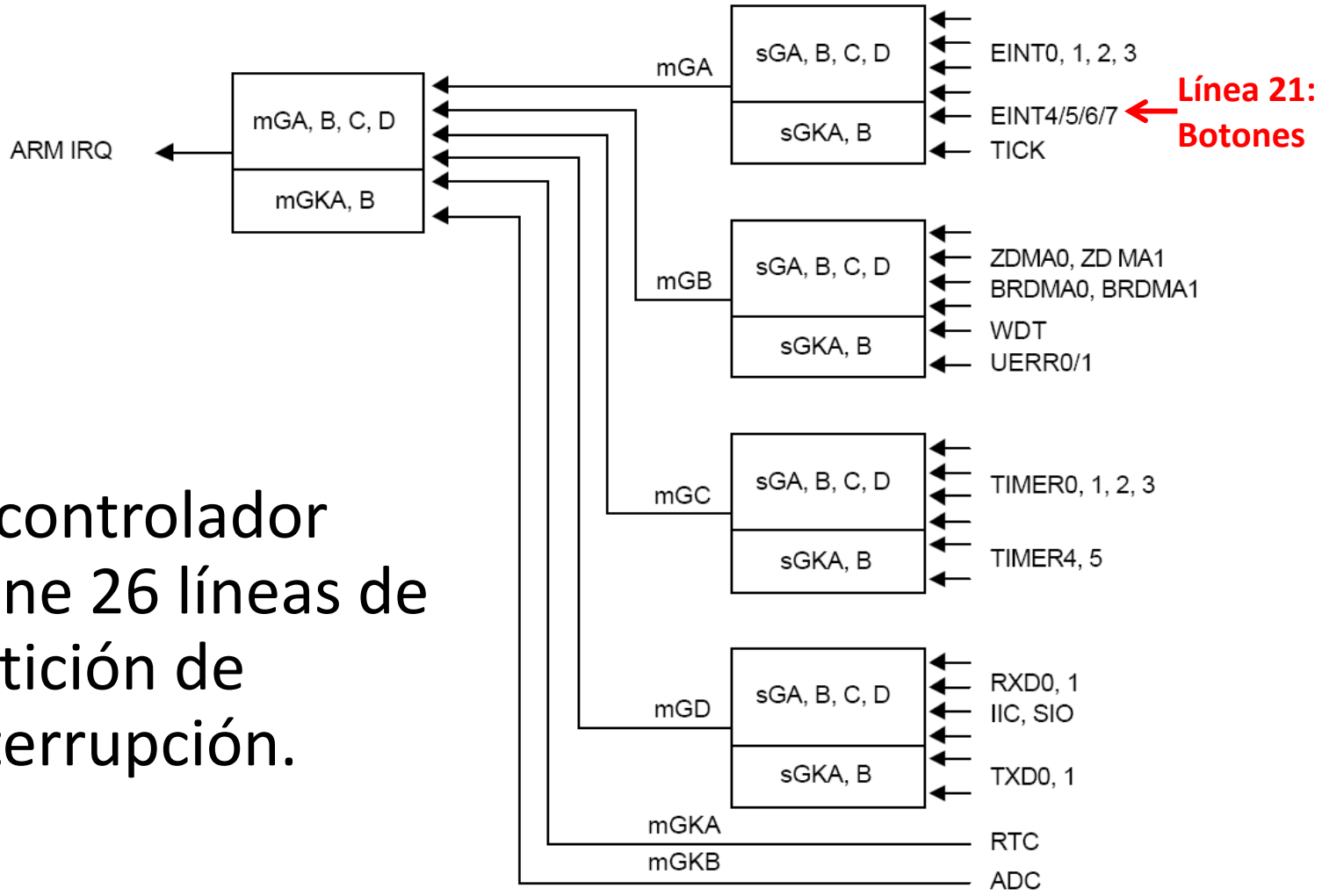


# Controlador de interrupciones

- Como hay muchos periféricos que pueden interrumpir, suele existir un controlador de interrupciones que se encarga de la gestión



# Controlador de interrupciones



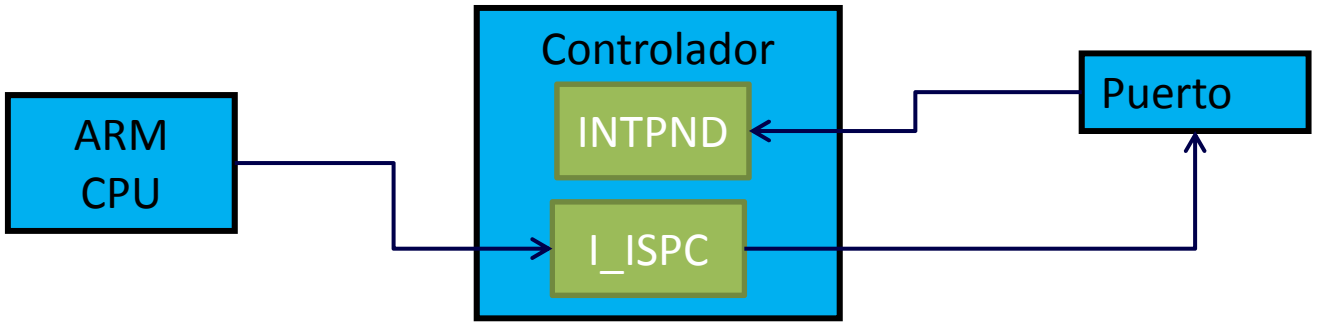

- El controlador tiene 26 líneas de petición de interrupción.

# Controlador de interrupciones

- Registros de configuración
  - INTCON (Interrupt Control Register), 4 bits
    - V: (INTCON(2)) = 0, habilita las interrupciones vectorizadas
    - I: (INTCON(1)) = 0, habilita la línea IRQ
    - F: (INTCON(0)) = 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

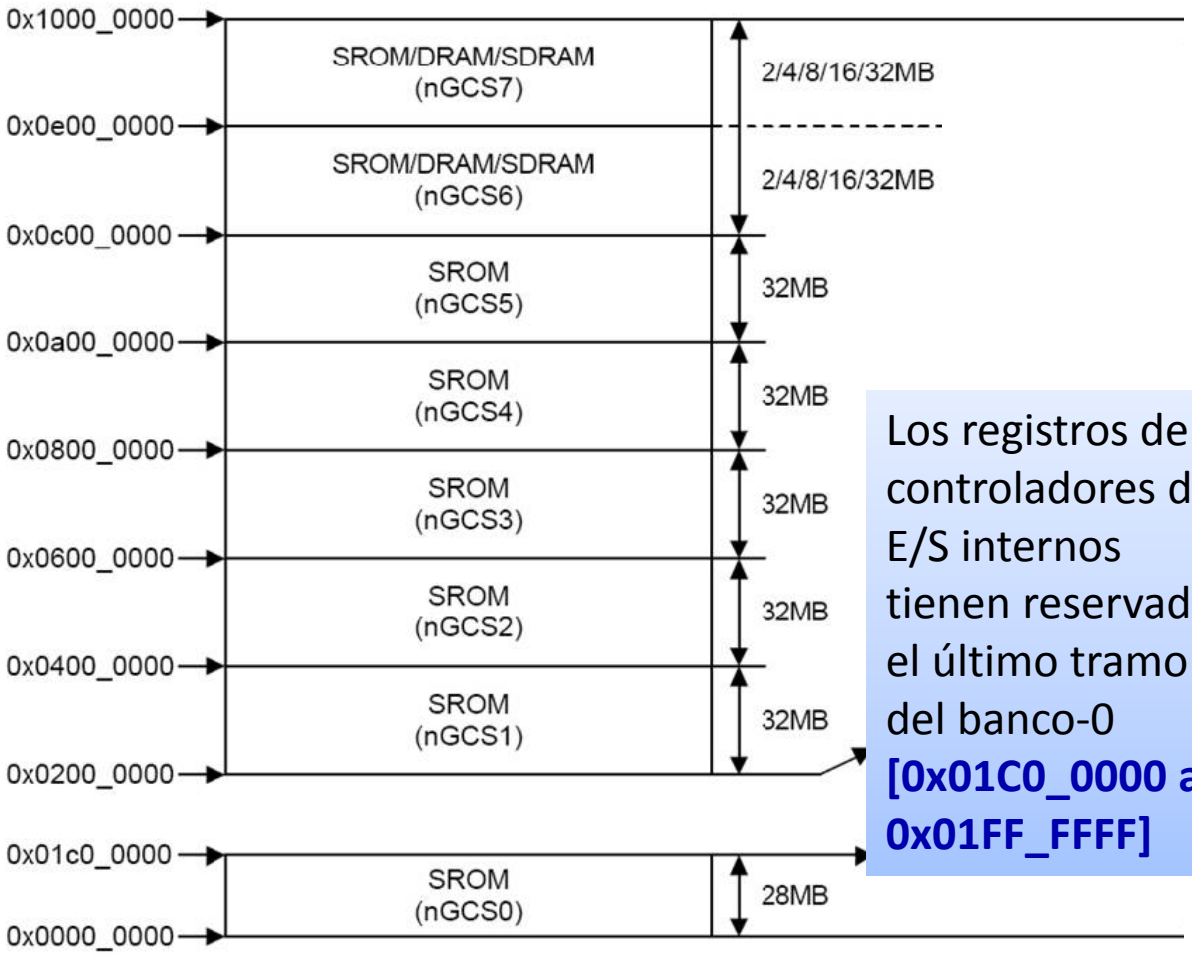


# Controlador de interrupciones


- Cómo se lee o se escribe sobre un registro del controlador de interrupciones
  - El ARM tiene la E-S mapeada en memoria, por lo que es lo mismo que realizar una operación de entrada salida con memoria (LDR y STR)
  - Ocurre lo mismo con los registros del interfaz HW de cada uno de los elementos de entrada salida

# 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]**

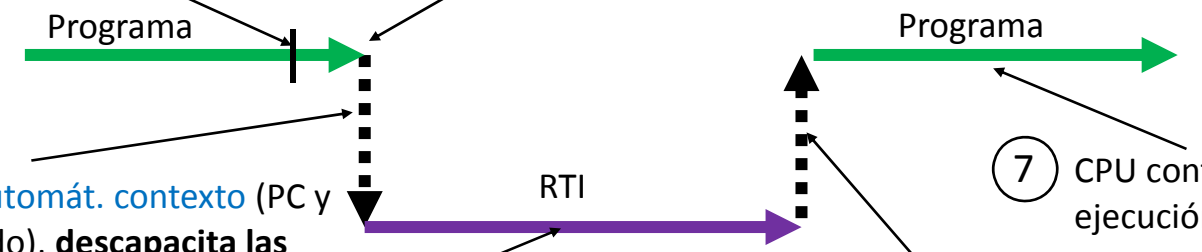
# Controlador de Interrupciones


## Rellenar por el estudiante

| Registro del controlador | Dirección de memoria |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |

# Secuencia de eventos en el tratamiento de una interrupción

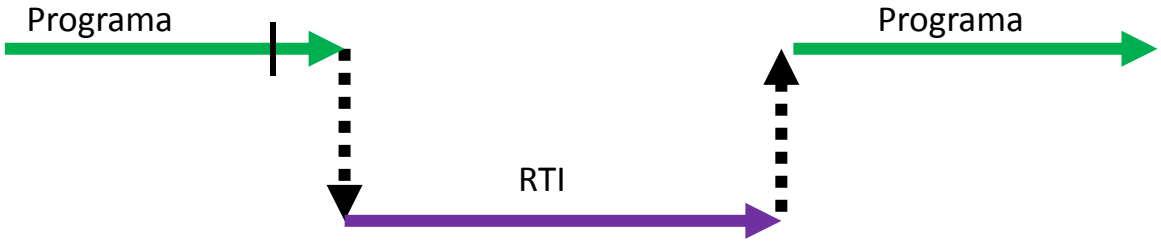
1. El programa activa el sistema de interrupciones
2. Periférico activa petición interrupción (INTR)
3. CPU termina de ejecutar la instrucción en curso y **comprueba si hay interrupciones pendientes**




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

# 1. El programa activa el sistema de interrupciones

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.

**Registro CPSR del ARM**

# 1. El programa activa el sistema de interrupciones


- Activación del sistema de interrupciones:
  - Se debe hacer al principio del programa
  - Para poder activar el sistema de interrupciones hay que pasar a un modo privilegiado que nos permite realizar el cambio

# 1. El programa activa el sistema de interrupciones

- Para activar la IRQ hay que poner un **cero** en el **bit I** del registro de estado

¿Con esta mascará estamos habilitando IRQ e IFQ?

```
@Activamos las IRQ
mrs r0,cpsr
bic   r0,r0,#0xDF          @R0 = R0 and (NOT 0xDF)
orr   r1,r0,#USERMODE
msr   cpsr, r1
ldr sp,=UserStack
```

# 1. El programa activa el sistema de interrupciones

- Inicializar la tabla de excepciones:
  - Indicar dónde está la rutina de tratamiento de cada una de las excepciones

InitISR:

```
ldr r0,=ISR_Tipo
ldr r1,=HandleTipo
str r0,[r1]
…
```

El alumno obligatoriamente debe “etiquetar” a la rutina de tratamiento de interupciones IRQ como ISR_IRQ

# 1. El programa activa el sistema de interrupciones

- Configurar las interrupciones => Registros de configuración

*Contr. de Interrupciones*

1. INTCON (Interrupt Control Register), 3 bits
   - I (bit [1]) = 0, habilita la línea IRQ

   ⇒ STR Valor, [INTCON_ADDR]
2. INTMOD (Interrupt Mode Register), 1 bit por línea
   - 0 = modo IRQ

   ⇒ STR Valor, [INTMOD_ADDR]
3. INTMSK (Interrupt Mask Register), 1 bit por línea
   - 0 = int. disponible; 1 = int. Enmascarada

   ⇒ STR Valor, [INTMSK_ADDR]

   ?

*Puerto G*

4. EXINT Activamos el modo en que la int. externa se dispara (flanco de subida, bajda …):
   - Hay que modificar EXTINT del puerto G (0x1d20050)

   STR **0x22222222**, [EXTINT_ADDR]

   ?

InitINTR

# 1. El programa activa el sistema de interrupciones

- Después de configurar las interrupciones es conveniente borrarlas por si se ha producido alguna interrupción indeseada.

*Cntr de INTR*

  - I_ISPC: Registro de 26 bits.
    - Si escribimos un "1" decimos que todas las interrupciones pendientes han sido atendidas.
    - I_ISPC Address?

*Puerto G*

  - EXTINTPND Nos dice cuál de las EINT4/5/6/7 está pendiente (0)
    - Poner que ninguna de ellas está pendiente (1)
    - STR **Valor**, [EXTINT_ADDR]
      **?**

InitINTR

# 1. El programa activa el sistema de interrupciones

- Cómo se programa en C
  - Cómo accedemos a una dirección de memoria fija
    - Definimos una variable que apunta a esa dirección

44b.h

```
#define rEXTINTPND  (*(volatile unsigned *)0x1d20054)
```

  - Cómo manipulamos bits

InitINTR

```
rEXTINTPND = 0xf;
```

# 1. El programa activa el sistema de interrupciones


- Se han definido en 44b.h todos los registros del sistema de entrada salida con el que vamos a trabajar en el laboratorio:

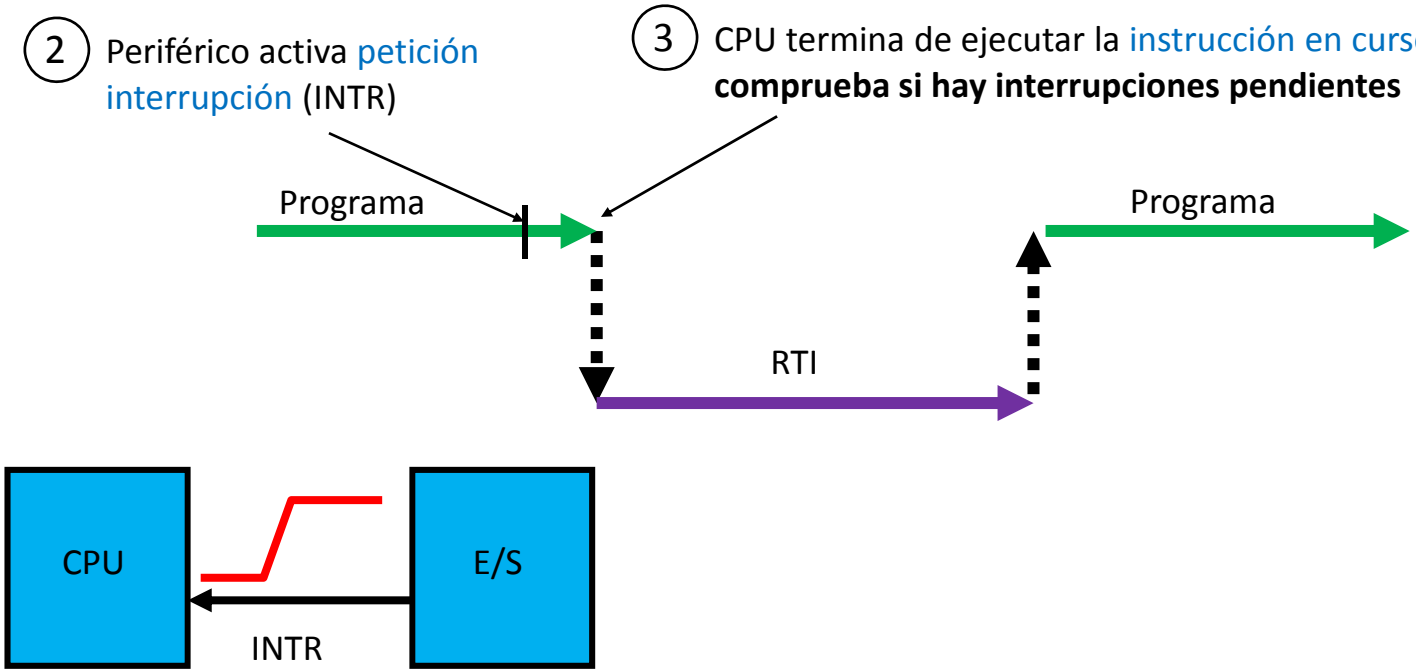
  **r**NOMBREDELREGISTRO

  – Se les ha asociado un tamaño y una posición fija de memoria

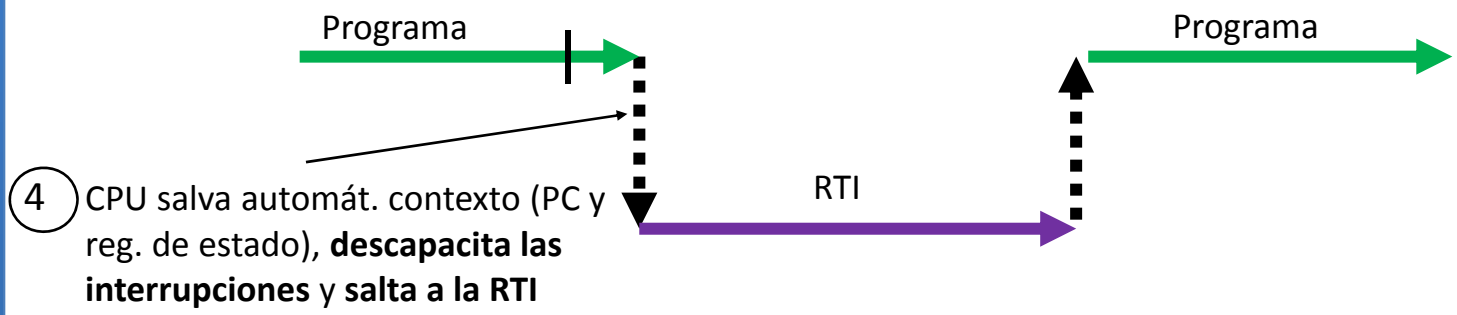
  – Se puede leer o escribir en cualquiera de esos registros utilizando el operador =

*Tarea: Reescribir el código de las transparencias 28 y 29 para C*

# Secuencia de eventos en el tratamiento de una interrupción (pasos 2 y 3)



# 4. CPU salva automát. contexto, descapacita las interrupciones y salta a la RTI




Salvar el estado: CPSR => SPSR y PC => LR

- 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.

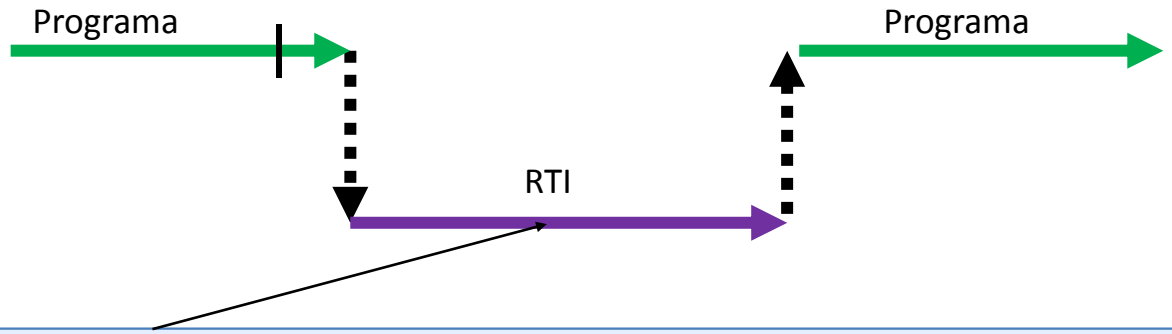
Descapacitar las interrupciones:

- En el registro de estado se inhabilitan **automáticamente** las interrupciones por la línea INTR (IRQ para el ARM)

Saltar a RTI => b HandlerIRQ => ISR_IRQ => PC <= La dirección de ISR_IRQ

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

# 5. CPU ejecuta la RTI




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)

# 5. CPU ejecuta la RTI

- Qué debe hacer la rutina de tratamiento de interrupciones ISR:
  - Epílogo/Prologo como si fuera una subrutina normal , pero asociado a su SP particular (SP_irq).
  - Descubrir quién ha sido el periférico que ha interrumpido
    - Método de encuesta
  - Llamar a la subrutina de tratamiento de ese periférico
  - Recuperar el valor de PC del flujo normal del programa
    - Para IRQ: SUBS PC, LR, #4

Recupera registro de estado
SPSR => CPSR

# 5. CPU ejecuta la RTI

```
/* prólogo */
push {r0-r10,fp}            @ Basta con apilar los registros modificados
                            @ INCLUYENDO r0-r3 si se modifican
add fp,sp,#(4*NumRegistrosApilados-4)

/* cuerpo de la rutina */

/* epílogo */
sub sp,fp, #(4*NumRegistrosApilados-4)
pop {r0-r10, fp}            @ restauramos contexto y retornamos
subs pc,lr,#4               @ La constante a restar depende de la excepción
```

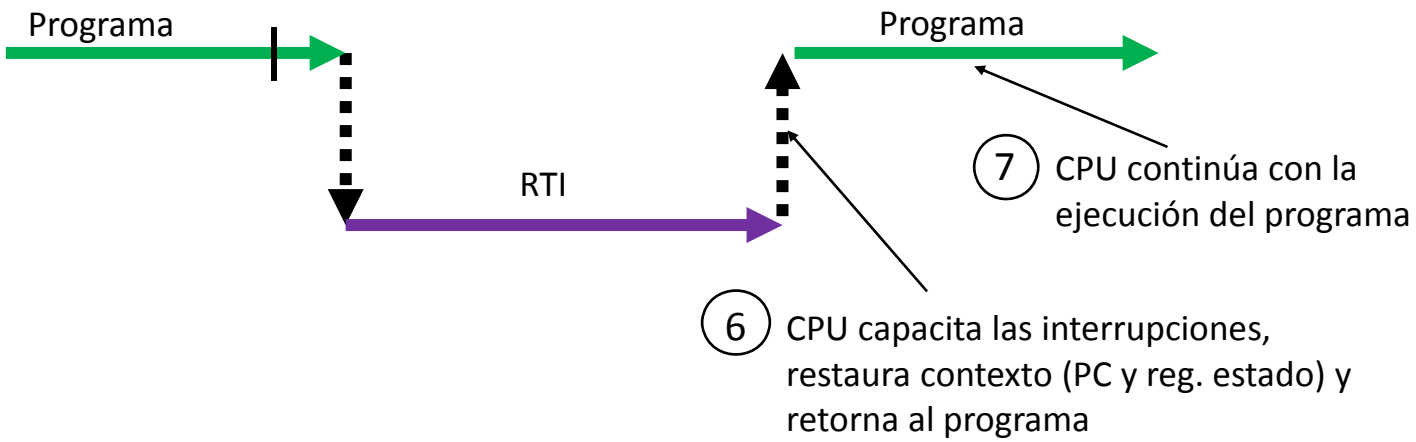
Aplicar el método de encuesta y saltar a la rutina de tratamiento del periférico específico que haya interrumpido

ISR_IRQ

# 5. CPU ejecuta la RTI

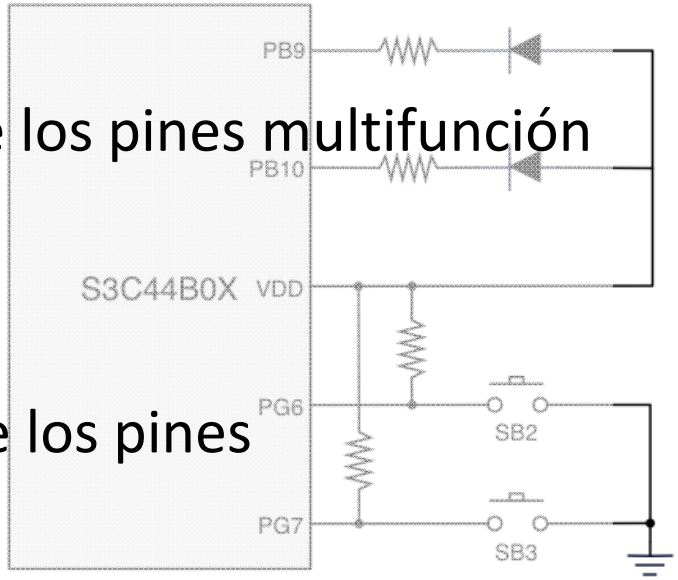

- Qué tiene que mirar el programa para saber quién ha interrumpido
  1. Comprobar el I_SPR
     - Indica si hay alguna interrupción pendiente
  2. El registro EXTINTPND
     - Indica qué interrupción está pendiente
- Cómo se mira:
  - Si hay alguna interrupción pendiente los bits de EXTINPND nos dirán qué periférico ha interrumpido y por lo tanto qué rutina hay que ejecutar
  - Los botones de la placa interrumpen por el bit 2 y 3
    - Saltar a DoDetecta
- Una vez ejecutado *DoDetecta*, cómo limpiamos las interrupciones
  - Borrar los bits en EXTINTPND
  - Borra el bit pendiente en INTPND

# Secuencia de eventos en el tratamiento de una interrupción

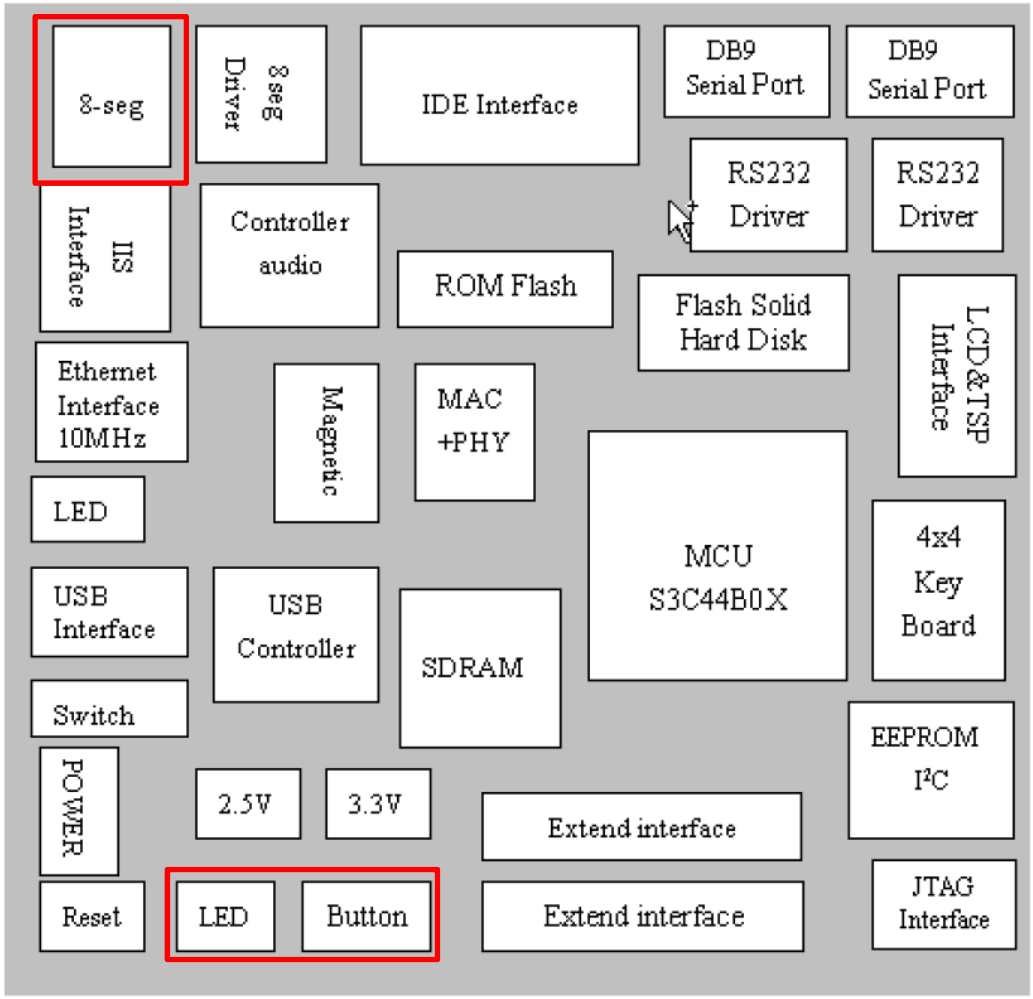


# ¿Qué periféricos usaremos?

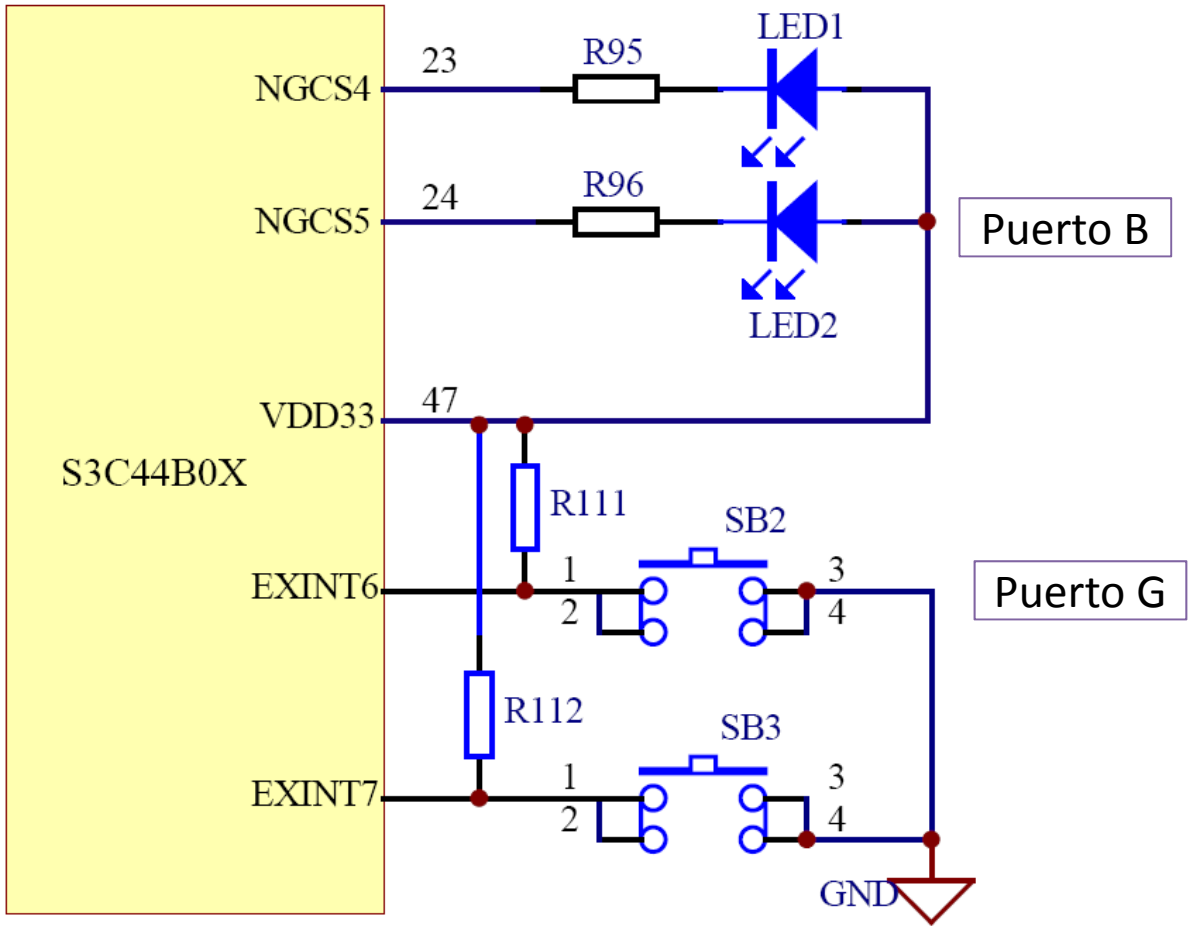
- LEDs
  - Existen 2 LEDs
  - Accesibles a través del puerto B de los pines multifunción de E/S (GPIO)
- Pulsadores
  - Existen 2 botones (pulsadores)
  - Accesibles a través del puerto G de los pines multifunción de E/S (GPIO)
- Display 8-segmentos
  - 7 LEDs para formar cualquier dígito hexadecimal
  - 1 LED para el punto decimal
  - Si el LED está a '0' → luz encendida



# ¿Dónde están los periféricos?



# Pulsadores y leds



# Conexión a los leds

- Puerto B (11 pines)
  - Registro de control de 11 bits: PCONB
  - Registro de datos de 11 bits: PDATB

| Registro | Dirección | R/W | Descripción | Valor por defecto |
|---|---|---|---|---|
| PCONB | 0x01D20008 | R/W | Configuración pines | 0x7ff |
| PDATB | 0x01D2000C | R/W | Datos | Undef |

  - Con la configuración por defecto
    - PB10 = nGCS5
    - PB9 = nGCS4

# Uso de LEDs


- Operación:
  1. Configurar pines 9 y 10 como salida (una vez)
     - Escribir un '0' en los bits 9 y 10 de PCONB

       **rPCONB = 0x1CF**
  2. Escribir en PDATB para encender/apagar. Ejemplo:
     - Si bit 9 de PDATB =0 → LED 1 encendido
     - Si bit 10 de PDATB=1 → LED 2 apagado

# Conexión de los pulsadores

- Puerto G (8 pines)

| Registro | Dirección | R/W | Descripción | Valor por defecto |
|---|---|---|---|---|
| PCONG | 0x01D20040 | R/W | Configuración pines | 0x0 |
| PDATG | 0x01D20044 | R/W | Datos | Undef |
| PUPG | 0x01D20048 | R/W | Deshabilitar pull-up | 0x0 |

  - Con la configuración por PCONG=0xffff
    - PG7 = EINT7
    - PG6 = EINT6
    - …
    - **Permitimos que los pulsadores interrumpan por esas líneas**

    PG3 = EINT3
    PG2 = EINT2
    …
  - Para que funcionen es necesario PUPG=0x0