Sistemas Empotrados

Tema 4:
Excepciones e interrupciones

Lección 11:
Interrupciones

Jesús González Peñalver

Contenidos

Tema 4: Excepciones e interrupciones

Interrupciones

Motivación

Permiten interrumpir la tarea actual para poder ejecutar otra:

- Permiten implementar sistemas multitarea
- Permiten una gestión eficiente de los dispositivos

Latencia de la interrupción

Tiempo transcurrido desde que se se activa la línea de petición de interrupción hasta que se ejecuta la primera instrucción de su ISR → Debe ser mínimo

Empeora cuando hay múltiples fuentes de interrupción:

- Se debe identificar la fuente de la interrupción para llamar a la ISR correspondiente
- Cuando el procesador atiende la petición de interrupción deshabilita las interrupciones, así que no se podrán atender nuevas peticiones hasta que se termine de ejecutar la ISR

La realidad: Hay muchas fuentes de interrupción

- Timers, DMA, controlador de red, pantalla táctil, controlador de sonido, controlador de vídeo, codecs multimedia, …

Solución: Interrupciones anidadas

Interrupciones anidadas

Motivación

Se permite que las ISR puedan ser interrumpidas por otras peticiones de interrupción

Funcionamiento

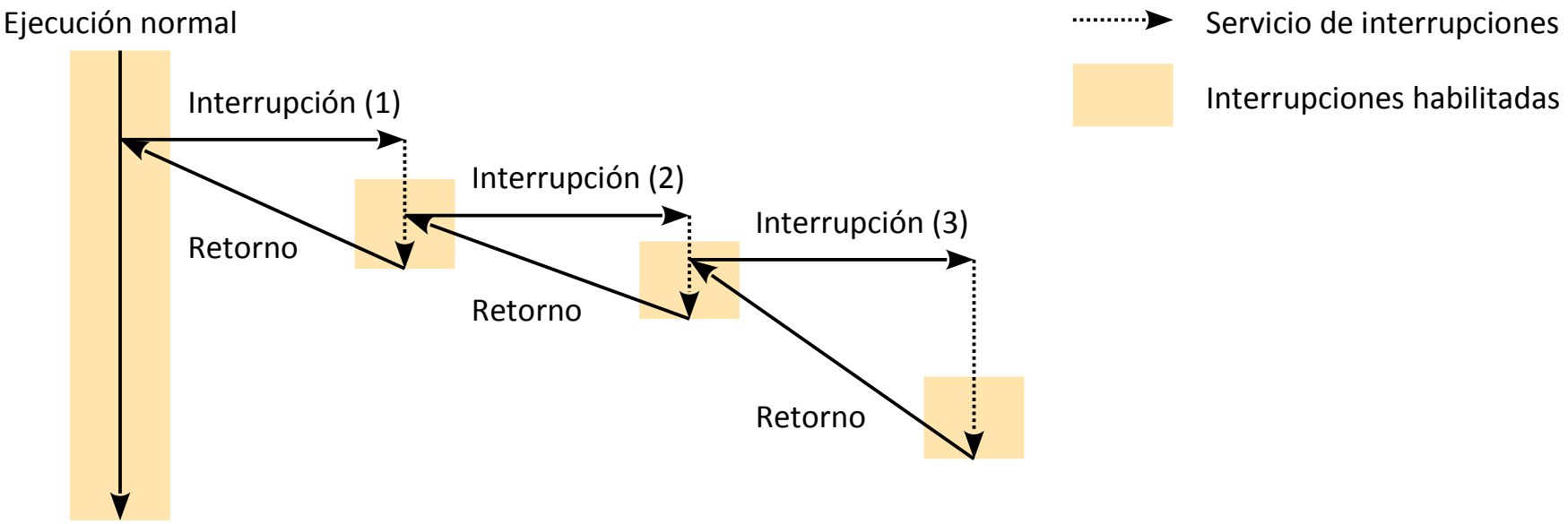
Se habilitan las interrupciones antes de terminar de servir la interrupción

Problema

Peticiones de interrupción poco importantes podrían interrumpir a ISR más importantes

Solución: Establecer prioridades

Una ISR sólo podrá ser interrumpida por una petición de más prioridad



Priorización de interrupciones

Motivación

Minimizar la latencia global del sistema

Las interrupciones críticas se atenderán antes que el resto

Problema

Para cada petición de interrupción, se debe...

identificar su fuente y su prioridad

... para determinar si debe interrumpir a la tarea que se está ejecutando

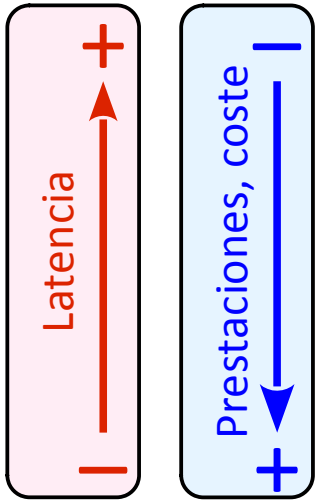
La implementación de estos dos pasos afecta seriamente a la latencia de las interrupciones

Posibilidades

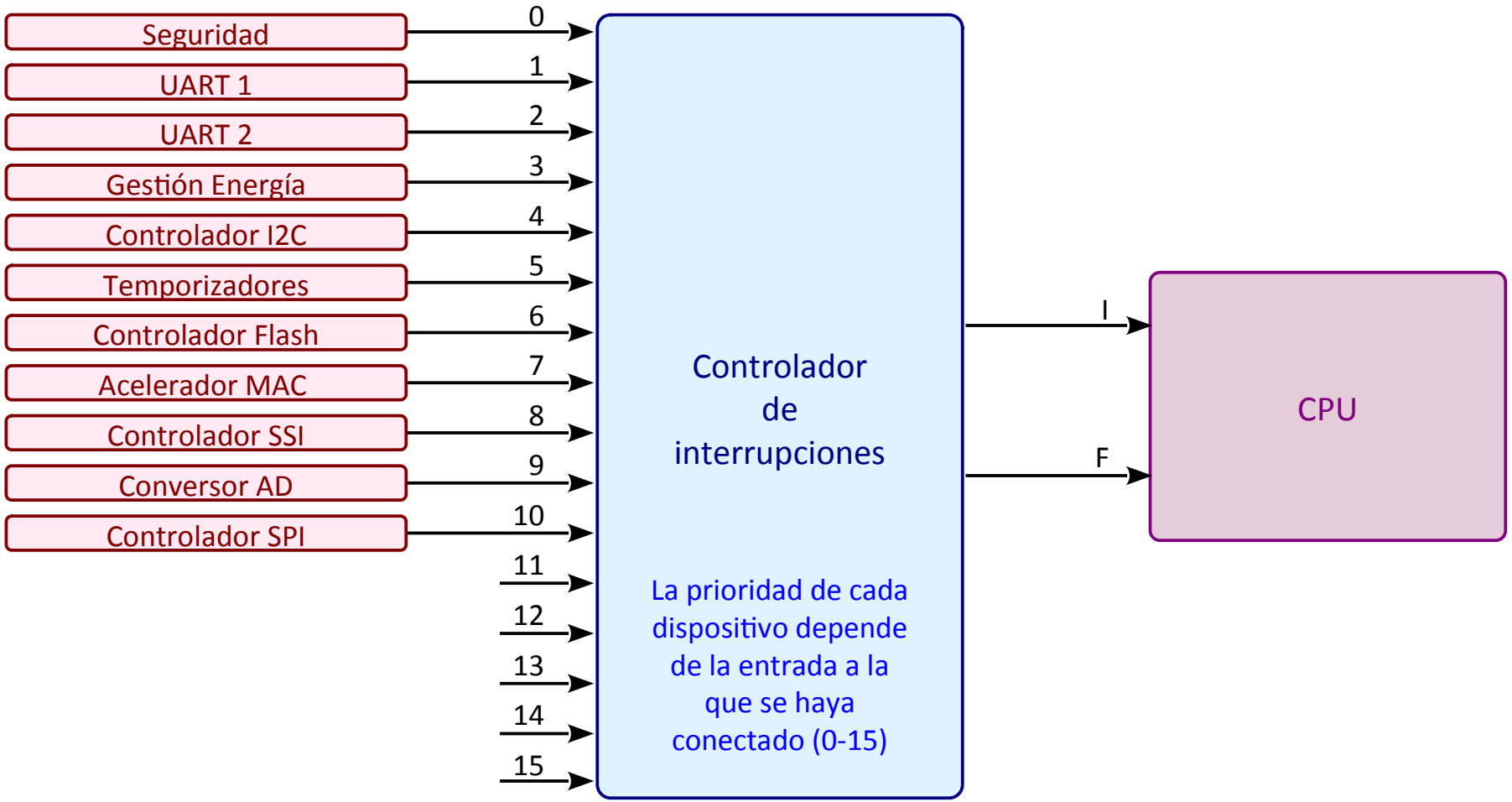
Sólo SW: Un manejador se encarga de la detección de la fuente, la priorización, y la determinación de la dirección de la ISR

HW+SW: El controlador de interrupciones interrumpe al procesador sólo si ocurre una petición de interrupción de más prioridad. El manejador sólo debe identificar la dirección de la ISR

Sólo HW: Si ocurre una petición de interrupción de más prioridad, el controlador de interrupciones vectorizado facilita al procesador la dirección de la ISR de la interrupción



Niveles de gestión de interrupciones



En cada dispositivo

Se configura para que pueda o no generar interrupciones, así como los casos en los que generarlas

En el controlador de interrupciones

Se asignan las fuentes (0-15) a las patillas I o F y se indica qué fuentes pueden pedir una interrupción

En el procesador

Se habilitan las peticiones por la entrada I y/o F

Interrupciones en la arquitectura ARM

FIQ (Fast Interrupt reQuest)

Se usa sólo para la fuente de interrupción más crítica del sistema

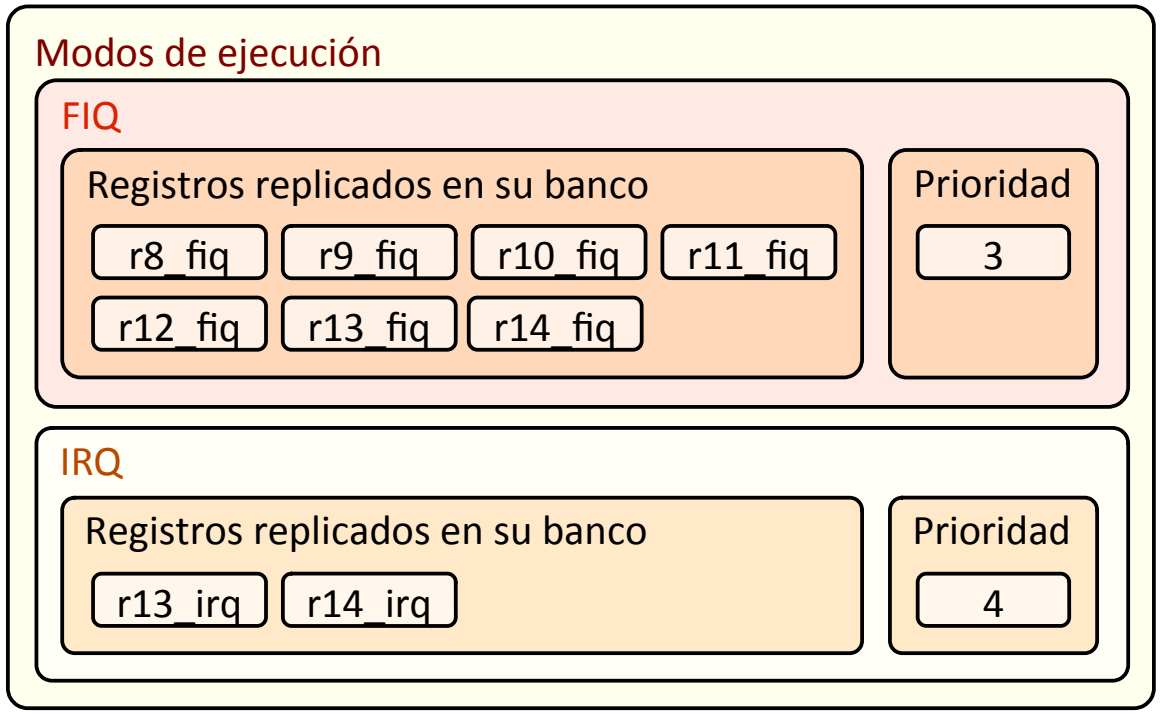
Su latencia es mínima

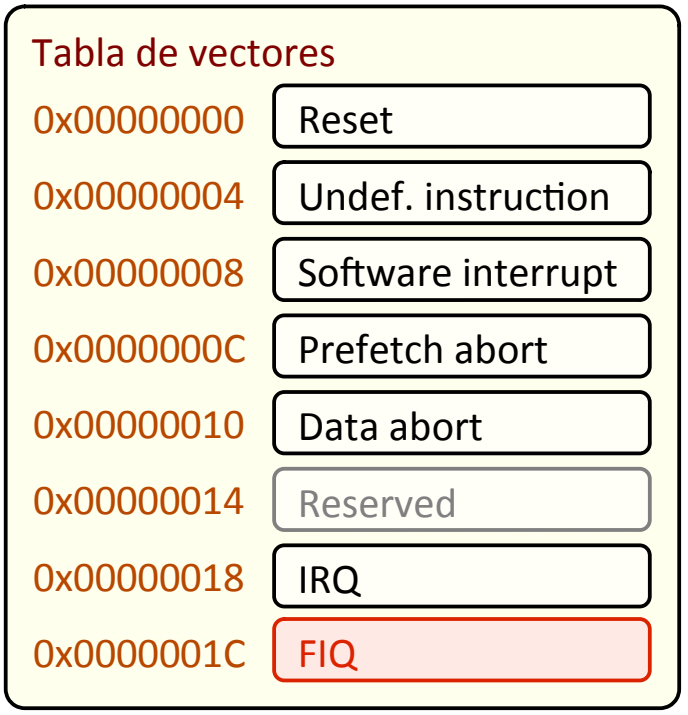
Puede interrumpir a cualquier otra interrupción (tiene más prioridad)

El cambio de contexto es mínimo (tiene más registros replicados)

No hay que identificar la fuente (es única)

El manejador puede empezar en la misma tabla de vectores (ahorramos el salto)





Interrupciones en la arquitectura ARM

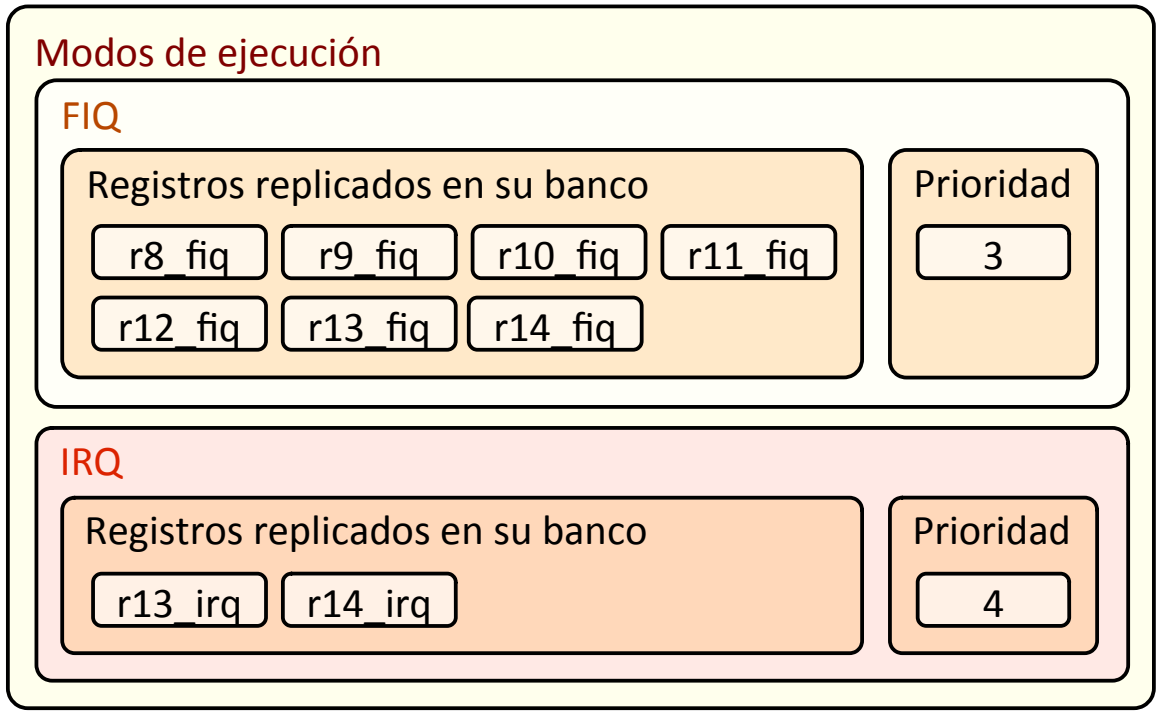
IRQ (Interrupt ReQuest)

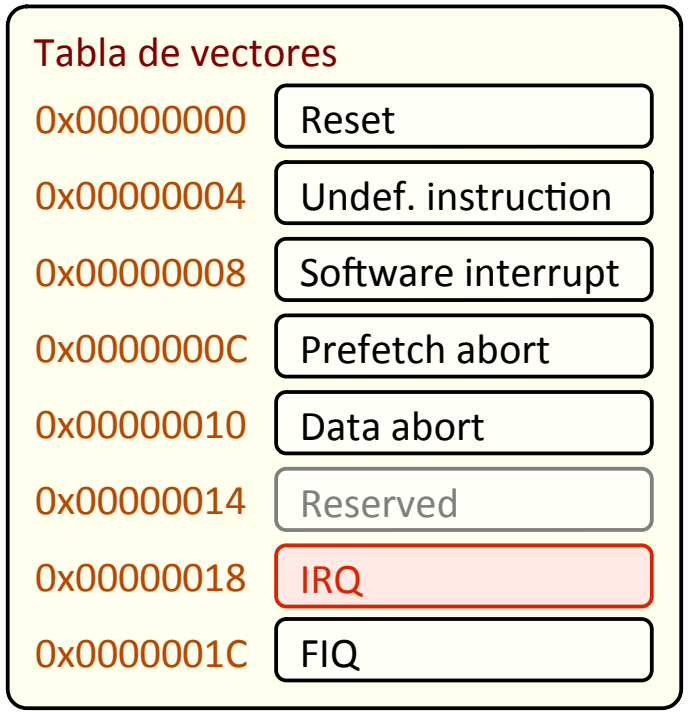
Se usa para el resto de fuentes de interrupción

Es necesario identificar la fuente y priorizar las peticiones de interrupción

Un manejador sólo SW es claramente ineficiente. La mayoría de los sistemas incorporan un controlador HW de interrupciones (vectorizado o no)

El diseño del manejador dependerá del tipo de controlador de interrupciones presente en el sistema





Contenidos

Tema 4: Excepciones e interrupciones

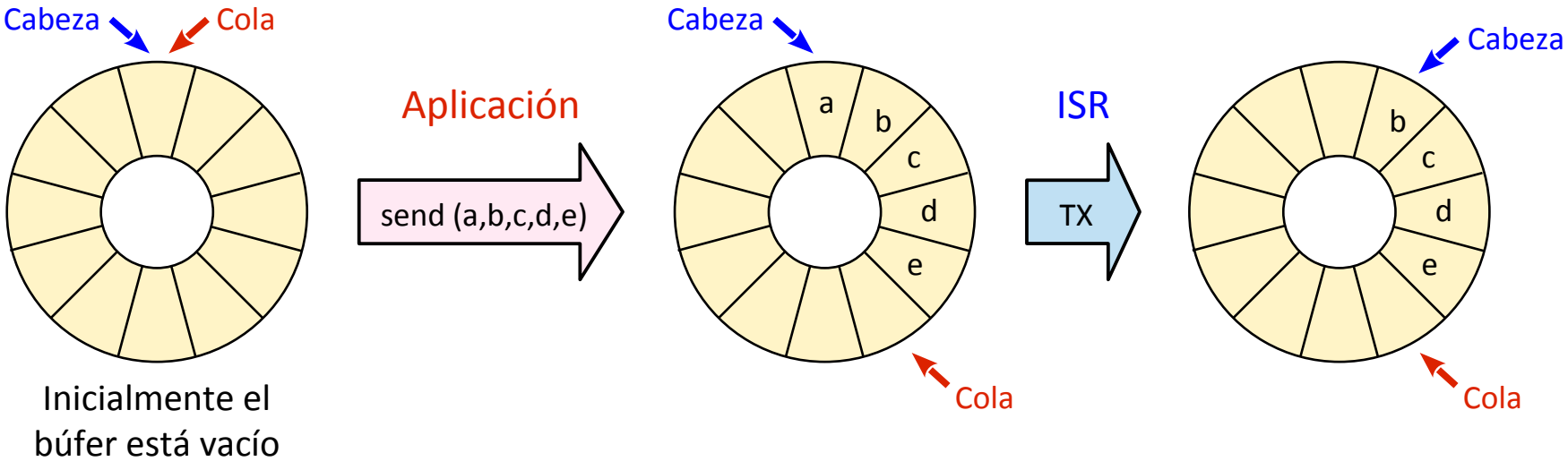
Condiciones de carrera

Definición

Cuando dos o más tareas acceden a un recurso compartido sin control y el resultado final depende del orden en el que se hayan realizado los accesos

Ejemplo

Comunicación entre una aplicación y el driver de un dispositivo mediante un búfer circular

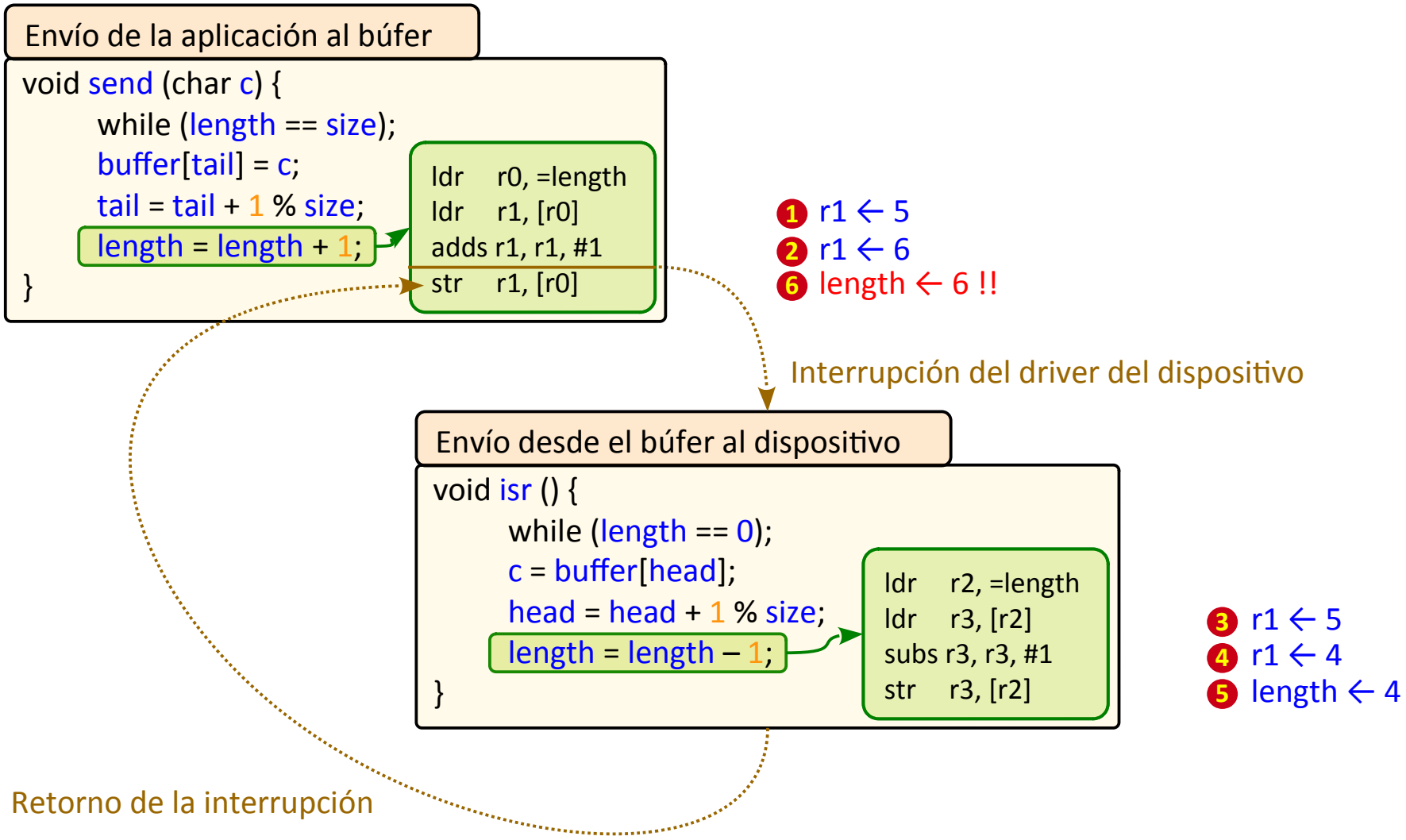


Inicialmente el búfer está vacío

Condiciones de carrera

La aplicación envía un carácter al búfer mientras la ISR saca otro

Partimos de un búfer en el que ya hay 5 caracteres



Regiones críticas

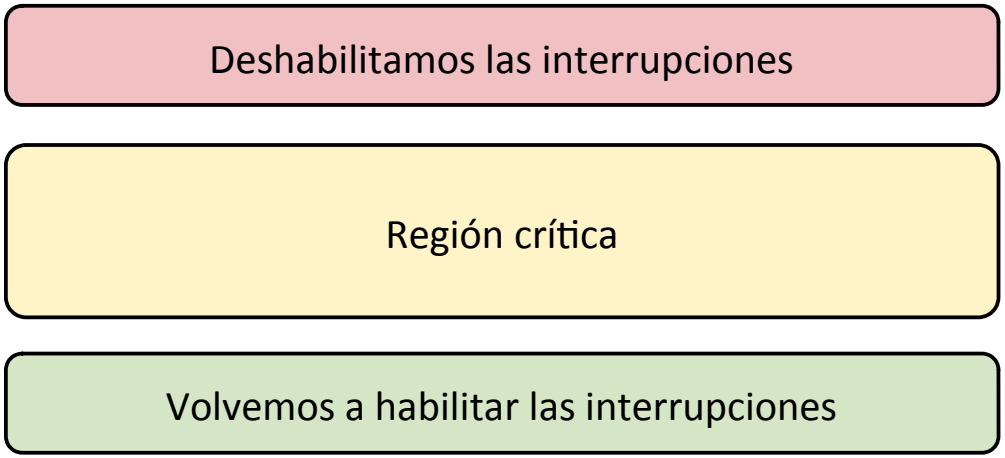
Motivación

Permiten compartir recursos entre diferentes tareas evitando las condiciones de carrera

Funcionamiento

Los cambios de tarea se provocan siempre mediante una interrupción. Si deshabilitamos las interrupciones la tarea actual se ejecutará indefinidamente

Implementación sencilla

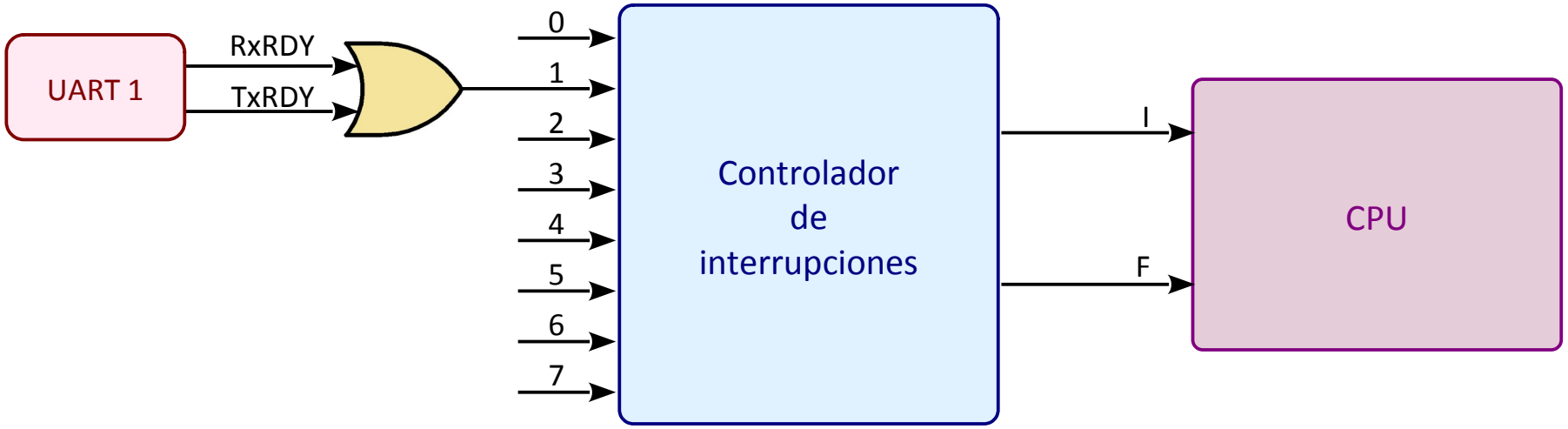


Consideraciones

Deben ser cortas para no penalizar en exceso el funcionamiento del sistema

Niveles de creación de una sección crítica

Hay que conseguir un acceso exclusivo al recurso compartido inhibiendo el menor número de fuentes de interrupción que sea posible



Sólo para un tipo de petición de interrupción de un dispositivo

Anulamos la generación de interrupción sólo para transmisión o recepción

Ej. En el acceso al búfer de envío entre la función send y la ISR de un driver

Sólo para uno o varios dispositivos

Inhibimos determinadas entradas en el controlador de interrupciones

Ej. Si algunas funciones de uno o varios drivers comparten un recurso

Para todas las interrupciones

Inhibiendo las peticiones por la entrada I y/o F

En el caso general para acceso a recursos compartidos por tareas de la aplicación

— Penalización +

Regiones críticas en la arquitectura ARM

Motivación

La arquitectura ARM tiene 2 líneas de petición de interrupción, nFIQ y nIRQ

Estas líneas se habilitan/deshabilitan mediante los bits F e I del registro CPSR

La modificación de estos bits sólo se puede hacer en ensamblador (instrucciones mrs y msr)

Funcionamiento

Antes de deshabilitar las interrupciones, debemos guardar el valor de los bits I y F en alguna variable

A la salida de la región crítica simplemente tendremos que restaurar el valor de estos bits en el registro de estado

Implementación

Es interesante disponer de funciones que se puedan llamar desde cualquier parte de nuestro código, con una interfaz en C → Inline assembler

Regiones críticas en la arquitectura ARM

Deshabilitar I y F

```
inline uint32_t excep_disable_ints () {
        uint32_t if_bits;

        asm volatile(   "mrs %[bits], cpsr\n\t"          /* bits <- cpsr */
                        "orr r12, %[bits], #0xC0\n\t"    /* I,F <- 1 */
                        "msr cpsr_c, r12"
                    :   [bits] "=r" (if_bits)            /* Parámetros de salida */
                    :                                    /* Parámetros de entrada */
                    :   "r12", "cc");                    /* Preservar */
        return (if_bits >> 6) & 3;
}
```

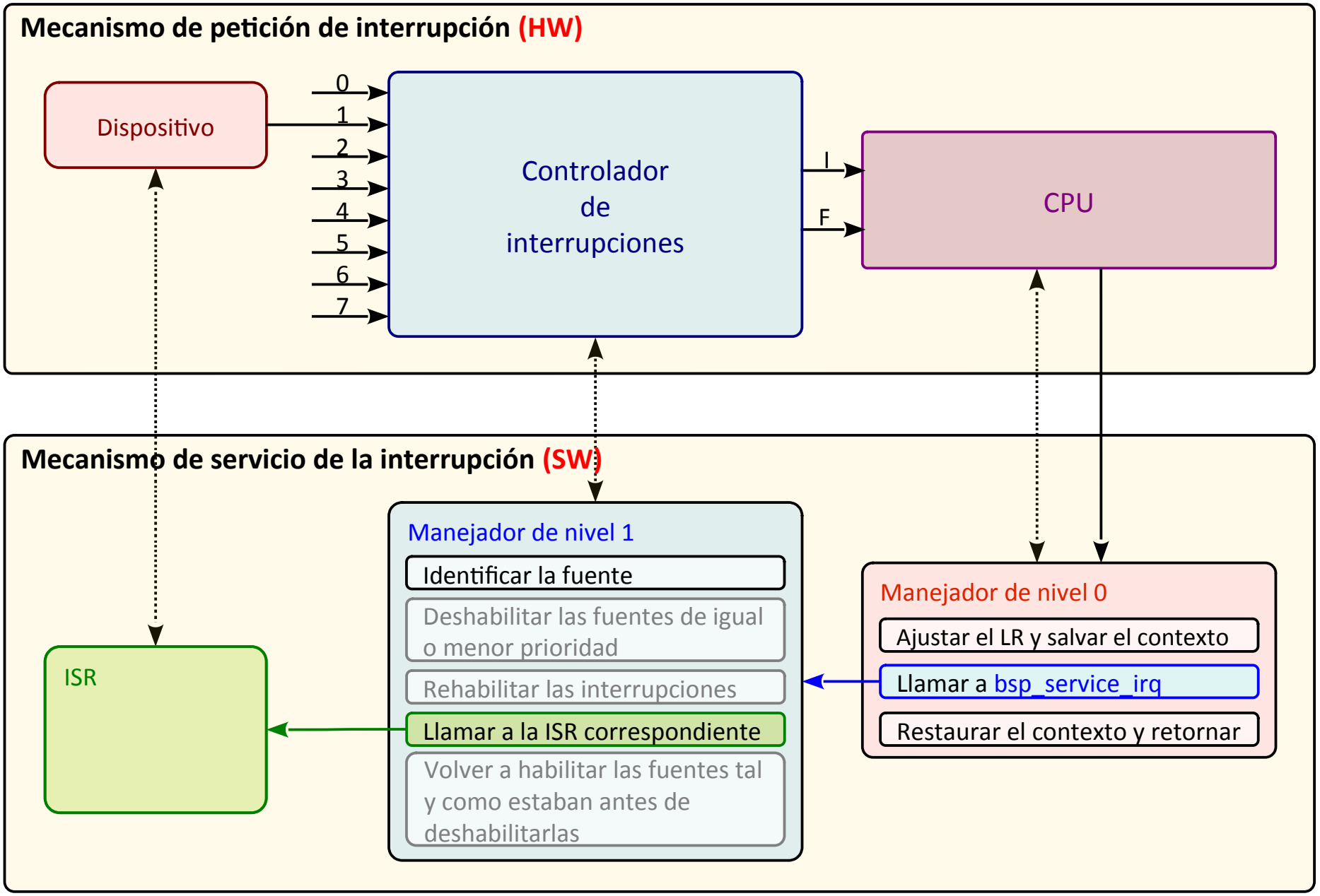
Detalle importante:
MSR no puede cambiar los bits 0:23 de un PSR en modo USER
Estas funciones no funcionan en modo USER!

Restaurar I y F

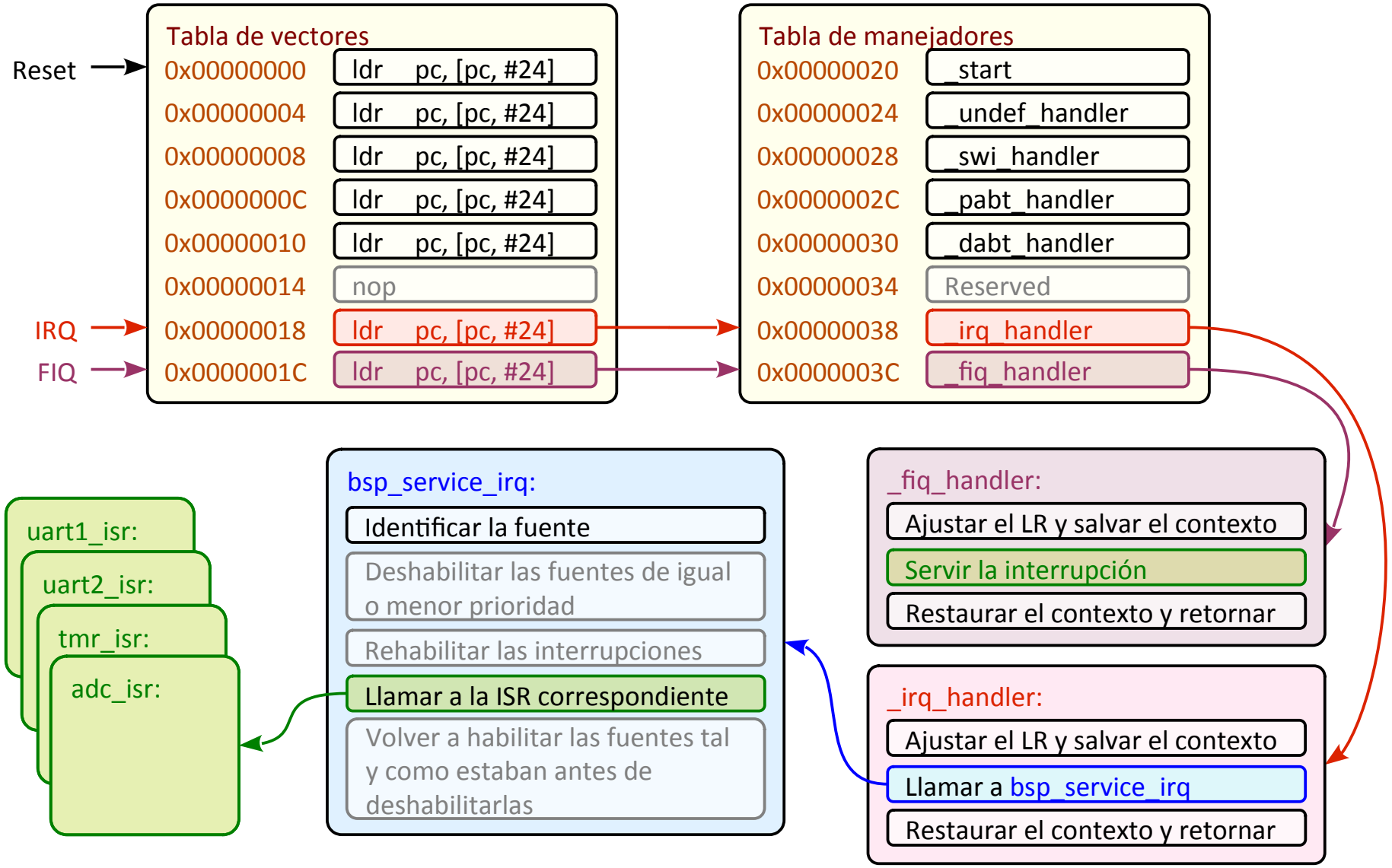
```
inline void excep_restore_ints (uint32_t if_bits) {
        asm volatile(   "mrs r12, cpsr\n\t"              /* r12 <- cpsr */
                        "bic r12, r12, #0xC0\n\t"        /* Limpiamos los bits I,F */
                        "orr r12, r12, %[bits], LSL #6\n\t"  /* Restauramos los bits */
                        "msr cpsr_c, r12"
                    :                                    /* Parámetros de salida */
                    :   [bits] "r" (if_bits & 3)         /* Parámetros de entrada */
                    :   "r12", "cc");                    /* Preservar */
}
```

Contenidos

Resumen del mecanismo de petición/servicio de interrupciones



Esquema de gestión de interrupciones para plataformas ARM



Manejador de interrupciones no anidadas

Funcionamiento

Una petición de interrupción no podrá interrumpir el servicio de otra interrupción

Ventajas

Fácil de implementar y depurar. Se puede escribir en C usando el atributo “interrupt”

Inconvenientes

Si hay varias fuentes de interrupción, se aumenta la latencia de las interrupciones

Manejador de IRQ de nivel 0 en C

```
__attribute__ ((interrupt ("IRQ")))
void nonnested_irq_handler ()
{
  Llamada al manejador de nivel 1
}
```

Mejor un manejador de nivel 0 en ensamblador que permita interrupciones anidadas para minimizarla latencia

Manejador de FIQ en C

```
__attribute__ ((interrupt ("FIQ")))
void nonnested_fiq_handler ()
{
  Código de la ISR
}
```

Se puede escribir en C. Como sólo hay una fuente FIQ, no necesitamos interrupciones anidadas

El código del manejador de nivel 1 dependerá del controlador de interrupciones y de las fuentes de interrupción del sistema, gestionados desde el BSP

Manejador nivel 0 (igual para todas las plataformas ARM)

Las interrupciones permanecen deshabilitadas mientras se sirve la interrupción

Identifica a la fuente y llama a la ISR correspondiente

El procesador

$spsr_{IRQ} \leftarrow cpsr$ — Cambia de modo

$cpsr.mode \leftarrow IRQ$

$cpsr.I \leftarrow 1$

$cpsr.F \leftarrow 1$ — Deshabilita interrupciones

$lr_{IRQ} \leftarrow PC$

$pc \leftarrow$ vector de IRQ — Salta al vector

En ensamblador

```
nonnested_irq_handler:

    @ Entrada del manejador
    @ Ajustamos el registro de enlace
    sub     lr, lr, #4

    @ Salvamos los registros corruptibles
    @ según AAPCS
    stmfd   sp!, {a1-a4, fp, ip, lr}

    @ Pasamos al manejador de nivel 1
    ldr     ip, =bsp_service_irq
    mov     lr, pc
    bx      ip

    @ Salida del manejador. Recuperamos
    @ los registros y retornamos
    ldmfd   sp!, {a1-a4, fp, ip, pc}^
```

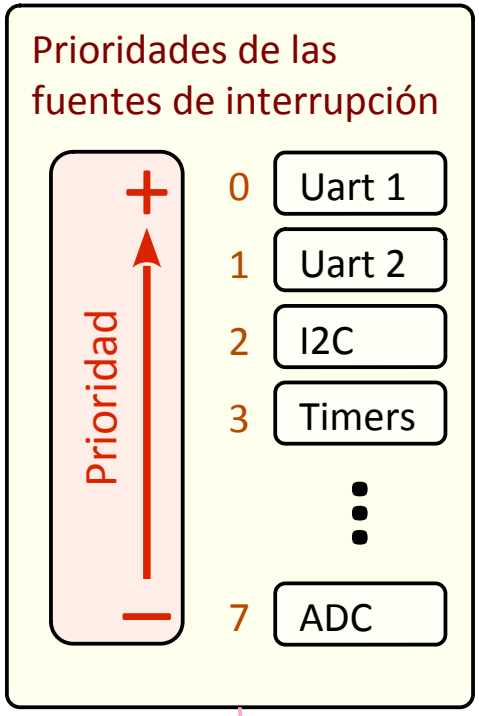
Manejador de IRQ de nivel 0 en C

```
__attribute__ ((interrupt ("IRQ")))
void nonnested_irq_handler ()
{
  bsp_service_irq ();
}
```

$pc \leftarrow lr - 4$ — Retorna de la interrupción

$cpsr \leftarrow spsr_{IRQ}$ — Retorna al modo original, restaurando las interrupciones

Manejador nivel 1 (según cada plataforma)



Prototipo para las ISR

```
typedef void (* isr_t) (void);
```

Tabla de ISRs

```
static isr_t _isrs[int_max];
```

Fijar una ISR

```
inline void bsp_set_isr (int_t source, isr_t isr) {
        _isrs[source] = isr;
}
```

Obtener una ISR

```
inline isr_t bsp_get_isr (int_t source) {
        return _isrs[source];
}
```

Dependiente del controlador de interrupciones de la plataforma

Manejador de IRQ de nivel 1

```
void bsp_service_irq () {
        int vector;
        vector = ict_get_vector ();  /* Identificamos la fuente de más prioridad */
        _isrs[vector]();             /* Llamamos a su ISR */
}
```

Contenidos

Tema 4: Excepciones e interrupciones

Manejador de interrupciones anidadas

Funcionamiento

Una petición de interrupción podrá interrumpir el servicio de otra interrupción

Ventajas

Permite gestionar múltiples fuentes de interrupción con prioridades y baja latencia media

Inconvenientes

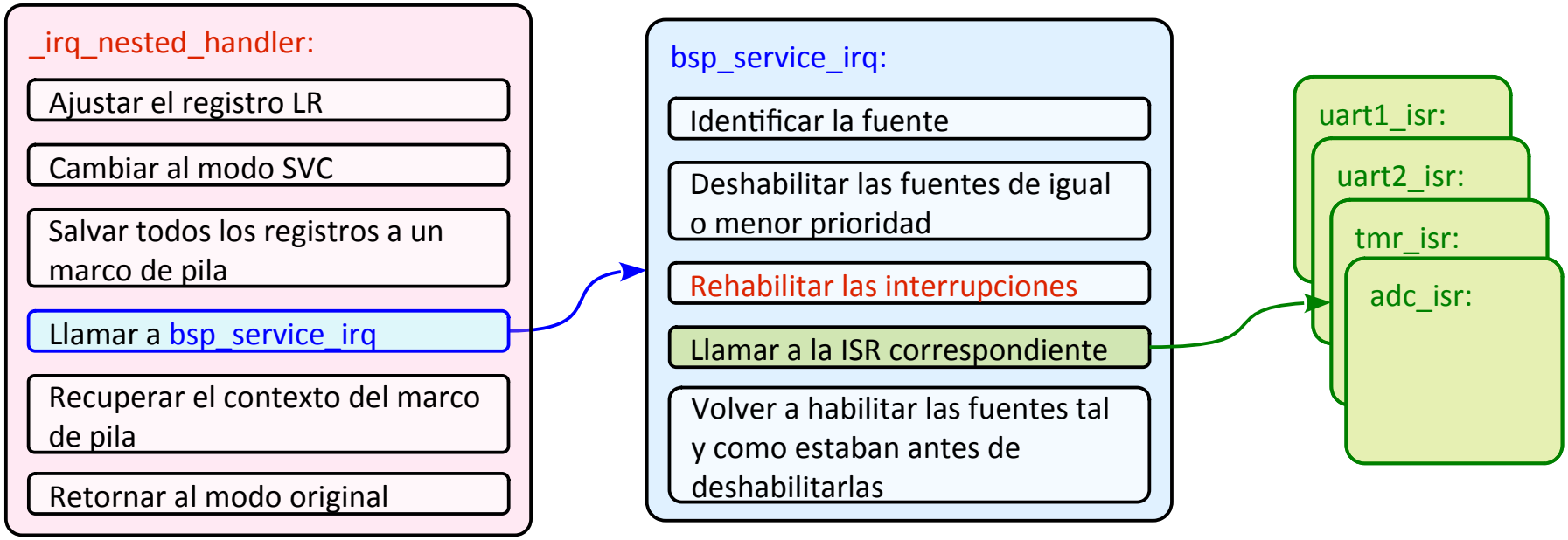
El atributo “interrupt” de C no guarda el registro SPSR en la pila al salvar el contexto en el manejador
→ Hay que codificar el nivel 0 del manejador en ensamblador

El manejador o la ISR podrán ser interrumpidos → Necesitamos gestionar un marco de pila para salvar el contexto completamente (todos los registros del procesador) antes de volver a habilitar las interrupciones

Debemos proteger la pila IRQ ante desbordamientos por causa de múltiples interrupciones anidadas
→ El manejador de nivel 0 cambia a modo SVC y guarda el contexto en la pila SVC

Si no tenemos un controlador HW que gestione las prioridades, la implementación del manejador de nivel 1 puede volverse muy complicada

Gestión de interrupciones anidadas con prioridades



Manejador de nivel 0
Dependiente del procesador
Necesariamente en ensamblador

Manejador de nivel 1
Dependiente del controlador de interrupciones
y de las fuentes de interrupción de la plataforma
Normalmente en C

ISR
Sirve la interrupción

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
    .set  _FRAME_A1,    0
    .set  _FRAME_A2,    _FRAME_A1  + 4
    .set  _FRAME_A3,    _FRAME_A2  + 4
    .set  _FRAME_A4,    _FRAME_A3  + 4
    .set  _FRAME_V1,    _FRAME_A4  + 4
    .set  _FRAME_V2,    _FRAME_V1  + 4
    .set  _FRAME_V3,    _FRAME_V2  + 4
    .set  _FRAME_V4,    _FRAME_V3  + 4
    .set  _FRAME_V5,    _FRAME_V4  + 4
    .set  _FRAME_V6,    _FRAME_V5  + 4
    .set  _FRAME_V7,    _FRAME_V6  + 4
    .set  _FRAME_FP,    _FRAME_V7  + 4
    .set  _FRAME_IP,    _FRAME_FP  + 4
    .set  _FRAME_PSR,   _FRAME_IP  + 4
    .set  _FRAME_LR,    _FRAME_PSR + 4
    .set  _FRAME_PC,    _FRAME_LR  + 4
    .set  _FRAME_SIZE,  _FRAME_PC  + 4

nested_irq_handler:

@ Ajustar lr y salvar los registros
@ corruptibles en la pila IRQ
    sub   lr, lr, #4
    stmfd sp!, {a1-a4, fp, ip, lr}

@ Guardar spsr_irq y sp_irq. Vaciar la pila IRQ
    mrs   a1, spsr
    mov   a2, sp
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
    .set  _FRAME_A1,    0
    .set  _FRAME_A2,    _FRAME_A1  + 4
    .set  _FRAME_A3,    _FRAME_A2  + 4
    .set  _FRAME_A4,    _FRAME_A3  + 4
    .set  _FRAME_V1,    _FRAME_A4  + 4
    .set  _FRAME_V2,    _FRAME_V1  + 4
    .set  _FRAME_V3,    _FRAME_V2  + 4
    .set  _FRAME_V4,    _FRAME_V3  + 4
    .set  _FRAME_V5,    _FRAME_V4  + 4
    .set  _FRAME_V6,    _FRAME_V5  + 4
    .set  _FRAME_V7,    _FRAME_V6  + 4
    .set  _FRAME_FP,    _FRAME_V7  + 4
    .set  _FRAME_IP,    _FRAME_FP  + 4
    .set  _FRAME_PSR,   _FRAME_IP  + 4
    .set  _FRAME_LR,    _FRAME_PSR + 4
    .set  _FRAME_PC,    _FRAME_LR  + 4
    .set  _FRAME_SIZE,  _FRAME_PC  + 4

nested_irq_handler:

@ Ajustar lr y salvar los registros
@ corruptibles en la pila IRQ
    sub   lr, lr, #4
    stmfd sp!, {a1-a4, fp, ip, lr}

@ Guardar spsr_irq y sp_irq. Vaciar la pila IRQ
    mrs   a1, spsr
    mov   a2, sp
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@
@
@
@
    msr   spsr_cxsf, lr
    ldmfd sp!,{lr,pc}^
```

Al reconocer la IRQ, el procesador...

$spsr_{IRQ} \leftarrow cpsr$

$cpsr.mode \leftarrow IRQ$

$cpsr.I \leftarrow 1$

$cpsr.F \leftarrow 1$

$lr_{IRQ} \leftarrow PC$

$pc \leftarrow$ vector de IRQ

Cambia de modo

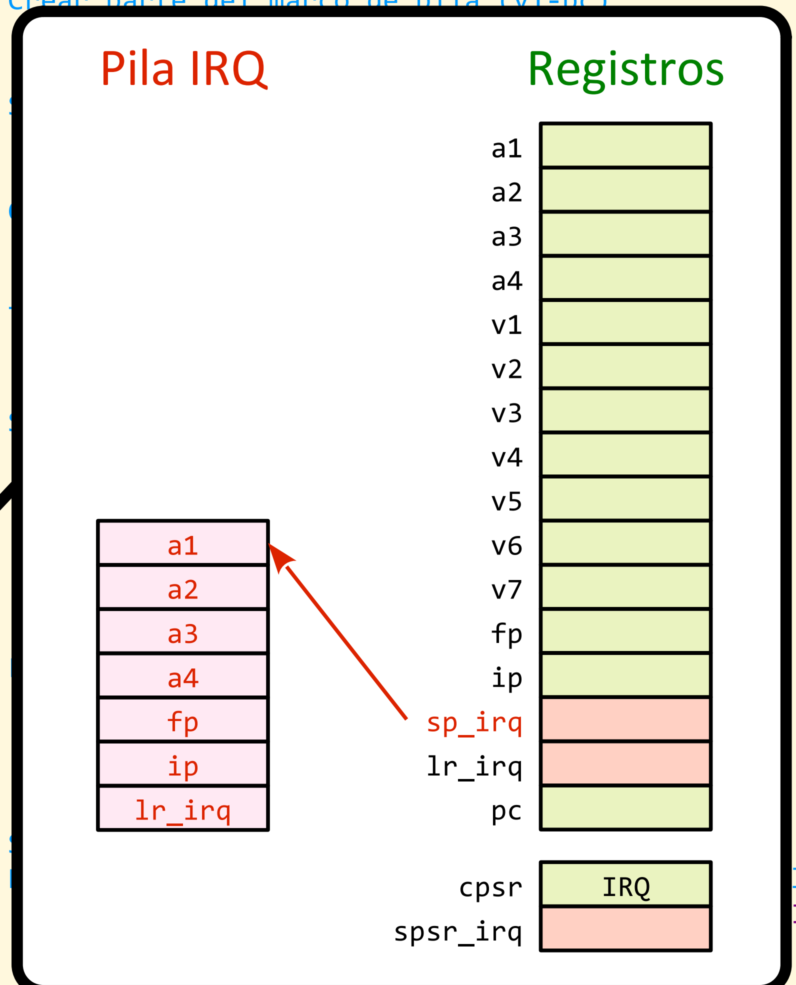
Deshabilita interrupciones

Salta al vector

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set   _IRQ_DISABLE, 0x80
    .set   _FIQ_DISABLE, 0x40
    .set   _SVC_MODE,    0x13
    .set   _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
    .set   _FRAME_A1,    0
    .set   _FRAME_A2,    _FRAME_A1  + 4
    .set   _FRAME_A3,    _FRAME_A2  + 4
    .set   _FRAME_A4,    _FRAME_A3  + 4
    .set   _FRAME_V1,    _FRAME_A4  + 4
    .set   _FRAME_V2,    _FRAME_V1  + 4
    .set   _FRAME_V3,    _FRAME_V2  + 4
    .set   _FRAME_V4,    _FRAME_V3  + 4
    .set   _FRAME_V5,    _FRAME_V4  + 4
    .set   _FRAME_V6,    _FRAME_V5  + 4
    .set   _FRAME_V7,    _FRAME_V6  + 4
    .set   _FRAME_FP,    _FRAME_V7  + 4
    .set   _FRAME_IP,    _FRAME_FP  + 4
    .set   _FRAME_PSR,   _FRAME_IP  + 4
    .set   _FRAME_LR,    _FRAME_PSR + 4
    .set   _FRAME_PC,    _FRAME_LR  + 4
    .set   _FRAME_SIZE,  _FRAME_PC  + 4

nested_irq_handler:

@ Ajustar lr y salvar los registros
@ corruptibles en la pila IRQ
    sub   lr, lr, #4
    stmfd sp!, {a1-a4, fp, ip, lr}

@ Guardar spsr_irq y sp_irq. Vaciar la pila IRQ
    mrs   a1, spsr
    mov   a2, sp
    add   sp, sp, #7*4
```

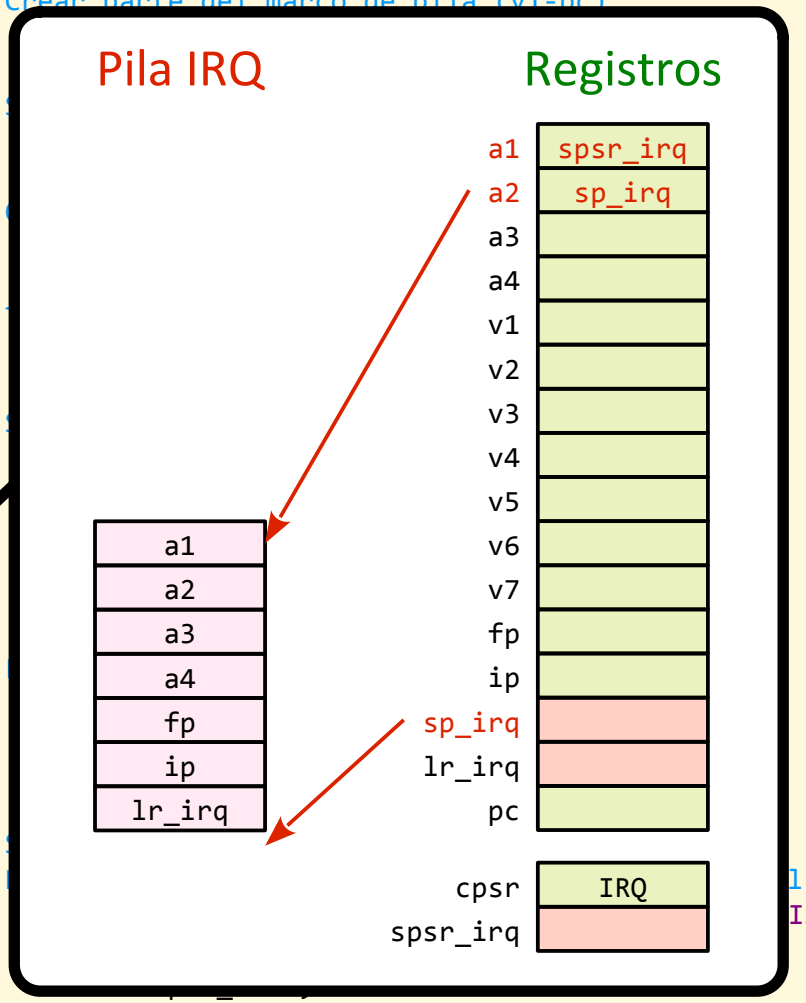
```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)

    ldmfd  sp!,{lr,pc}^
```



Manejador nivel 0 (igual para todas las plataformas ARM)

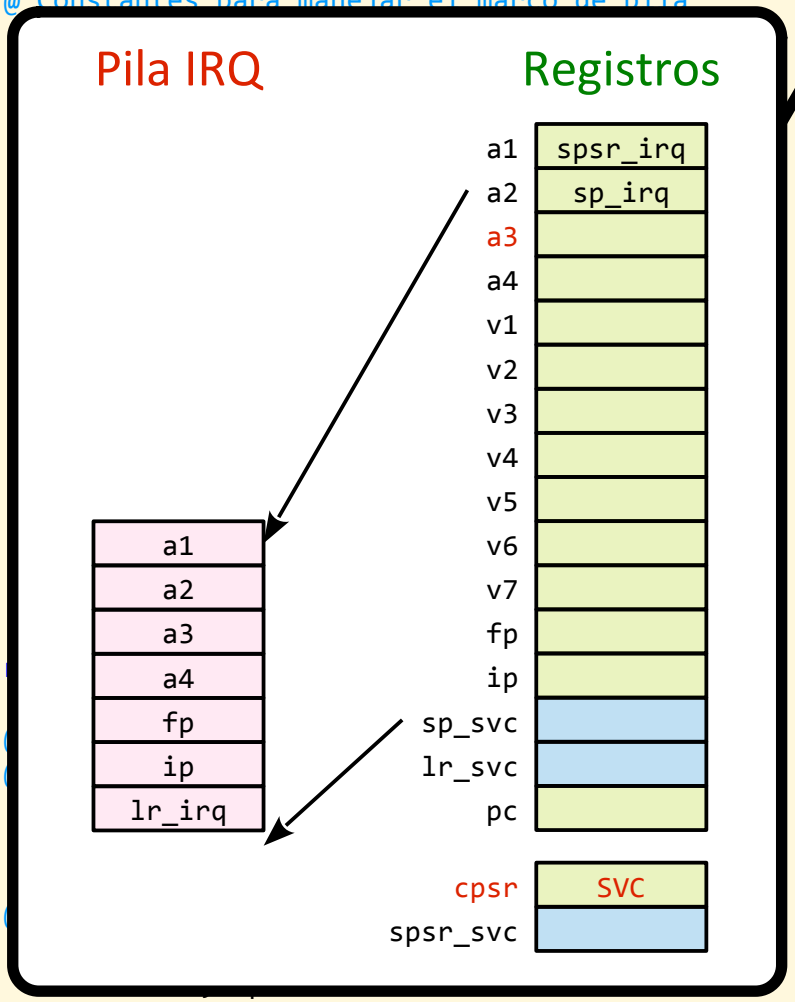
```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
    .set  _FRAME_A1,    0
    .set  _FRAME_A2,    _FRAME_A1  + 4
    .set  _FRAME_A3,    _FRAME_A2  + 4
    .set  _FRAME_A4,    _FRAME_A3  + 4
    .set  _FRAME_V1,    _FRAME_A4  + 4
    .set  _FRAME_V2,    _FRAME_V1  + 4
    .set  _FRAME_V3,    _FRAME_V2  + 4
    .set  _FRAME_V4,    _FRAME_V3  + 4
    .set  _FRAME_V5,    _FRAME_V4  + 4
    .set  _FRAME_V6,    _FRAME_V5  + 4
    .set  _FRAME_V7,    _FRAME_V6  + 4
    .set  _FRAME_FP,    _FRAME_V7  + 4
    .set  _FRAME_IP,    _FRAME_FP  + 4
    .set  _FRAME_PSR,   _FRAME_IP  + 4
    .set  _FRAME_LR,    _FRAME_PSR + 4
    .set  _FRAME_PC,    _FRAME_LR  + 4
    .set  _FRAME_SIZE,  _FRAME_PC  + 4

nested_irq_handler:

@ Ajustar lr y salvar los registros
@ corruptibles en la pila IRQ
    sub   lr, lr, #4
    stmfd sp!, {a1-a4, fp, ip, lr}

@ Guardar spsr_irq y sp_irq. Vaciar la pila IRQ
    mrs   a1, spsr
    mov   a2, sp
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
@ S
@ (
@ T
@ S
@ (
@ S
@ (                                          l
                                             ISABLE)
    ldmfd  sp!,{lr,pc}^
```



Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```

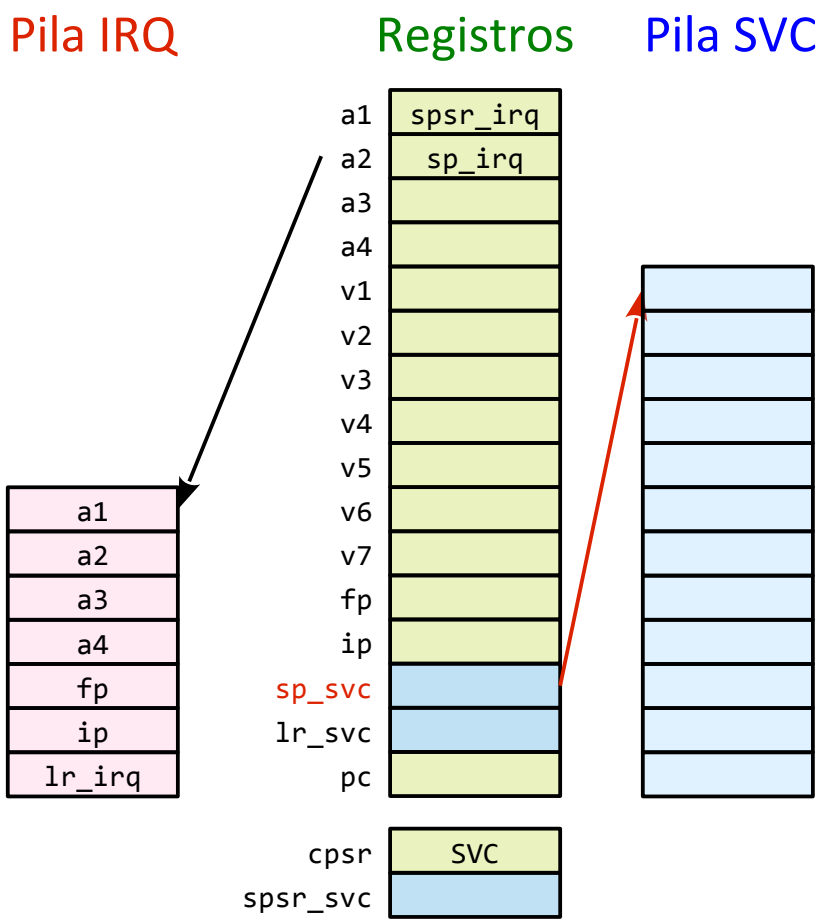



```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```

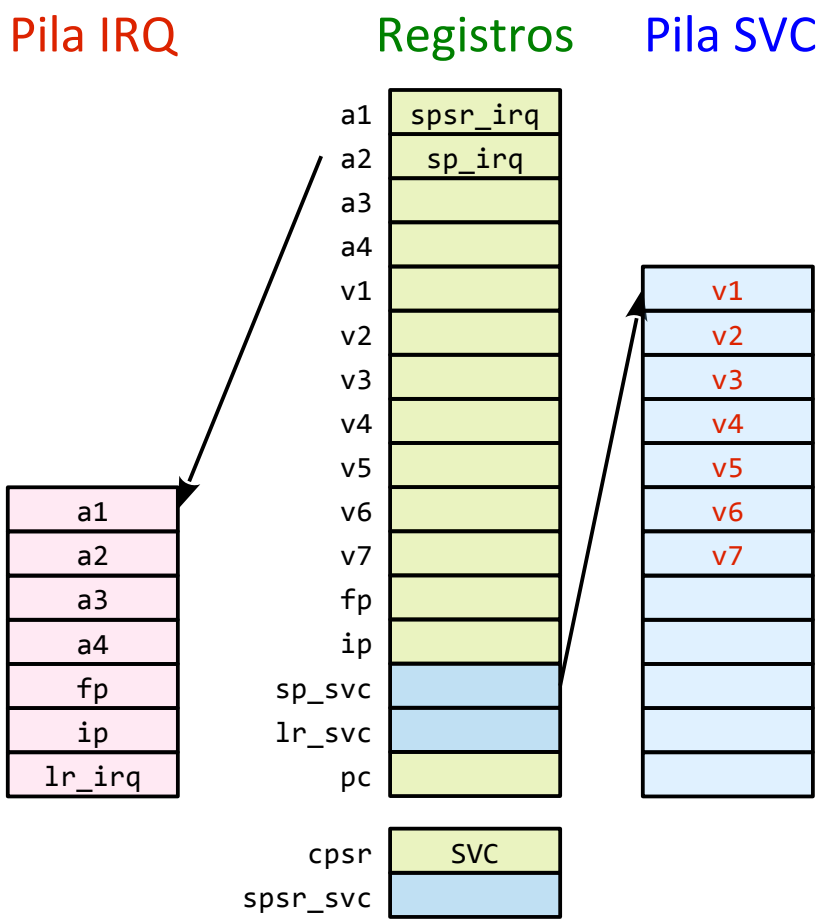



```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```

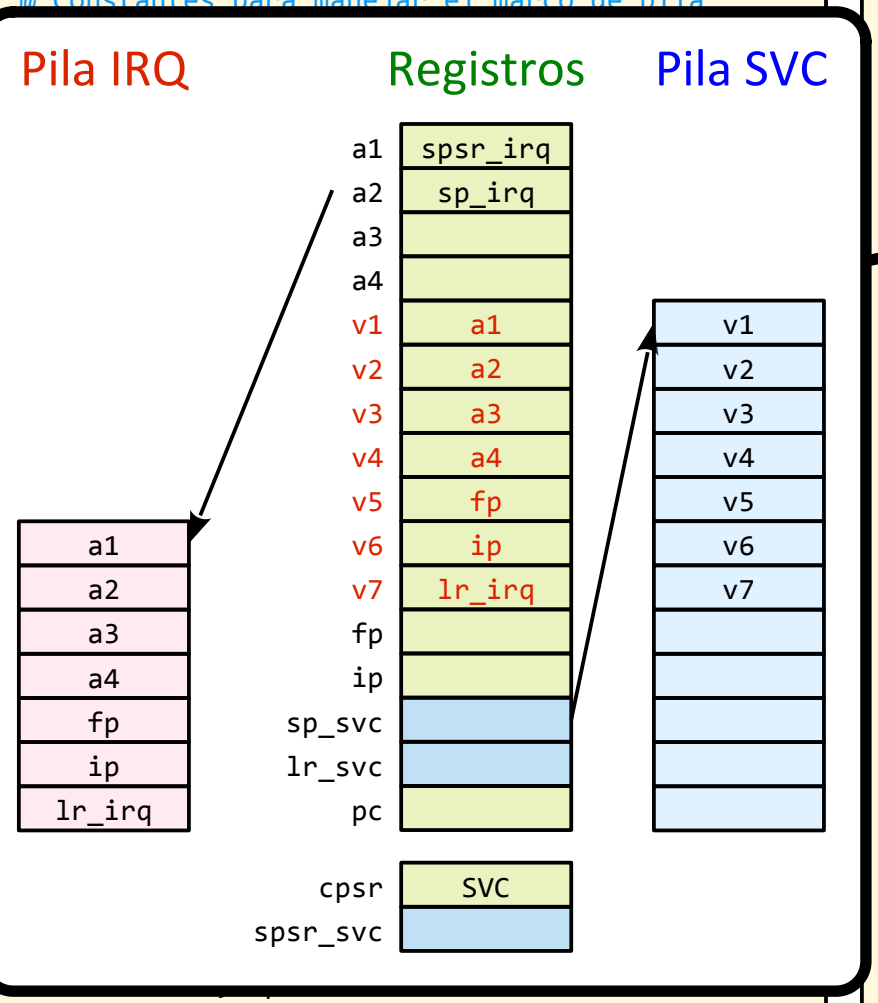



```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```

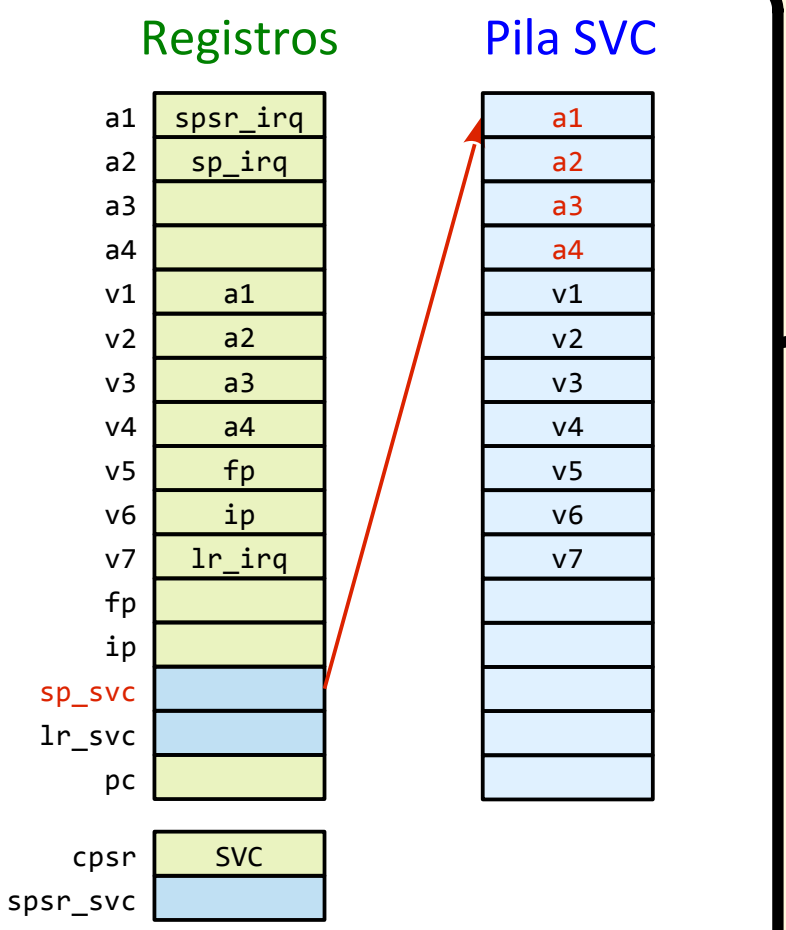



```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```

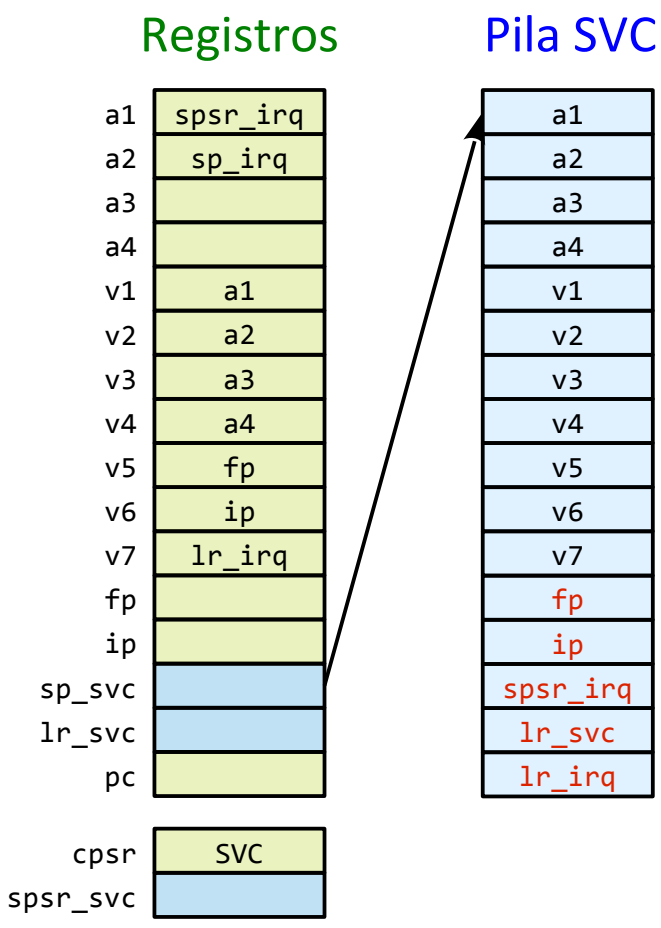



```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```




```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
    .set  _FRAME_A1,    0
```

El código del manejador de nivel 1 es dependiente de la plataforma

Los pasos básicos serían los siguientes:

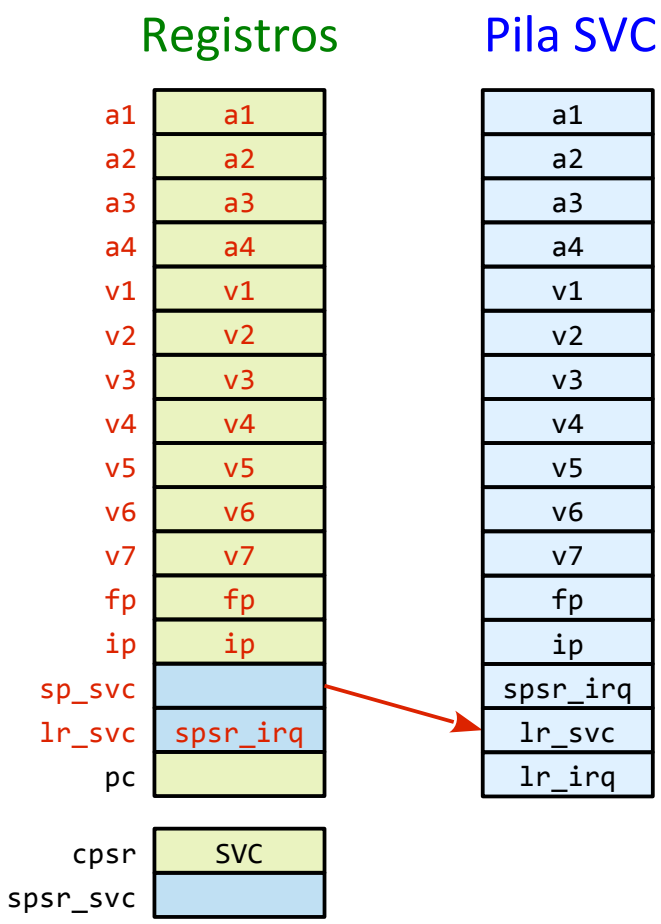
- Identificar la fuente
- Deshabilitar las fuentes de igual o menor prioridad
- Rehabilitar las interrupciones
- Llamar a la ISR correspondiente
- Volver a habilitar las fuentes tal y como estaban antes de deshabilitarlas

```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```

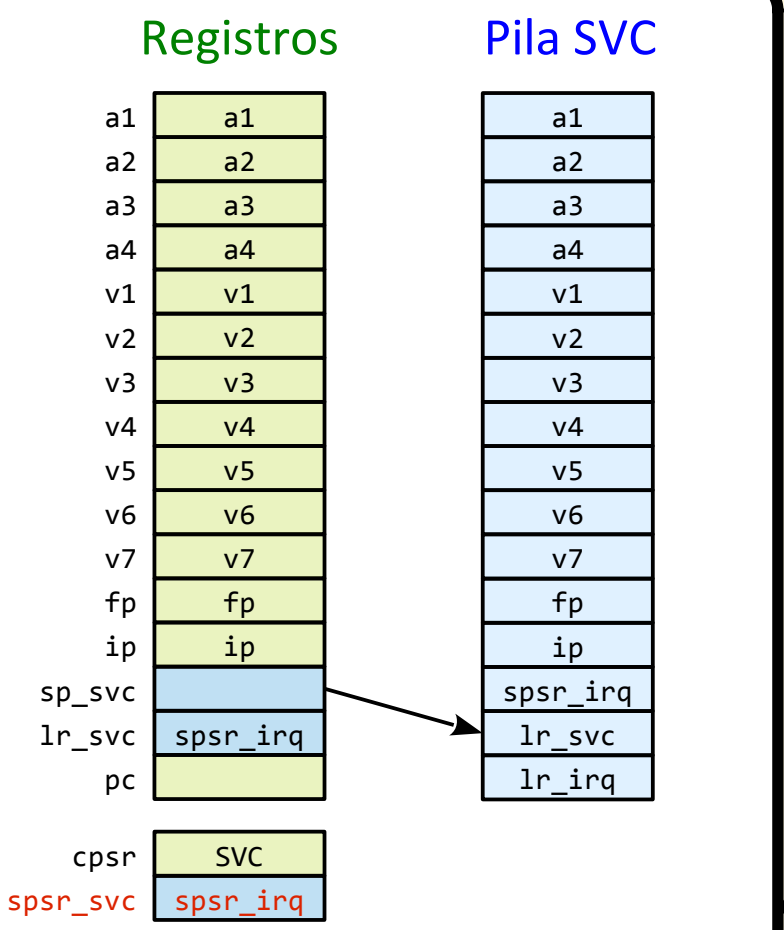



```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```



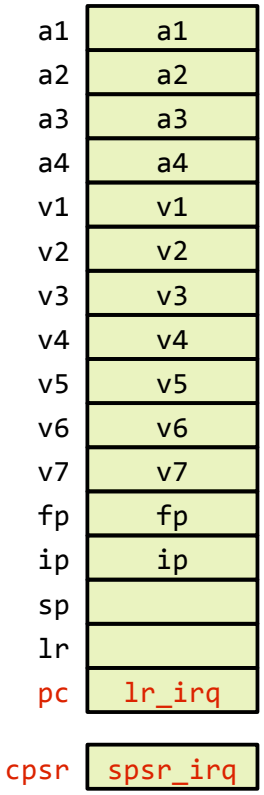

```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

Manejador nivel 0 (igual para todas las plataformas ARM)

```
@ Constantes para manejar el registro de estado
    .set  _IRQ_DISABLE, 0x80
    .set  _FIQ_DISABLE, 0x40
    .set  _SVC_MODE,    0x13
    .set  _MSK_MODE,    0x1F

@ Constantes para manejar el marco de pila
```

Registros

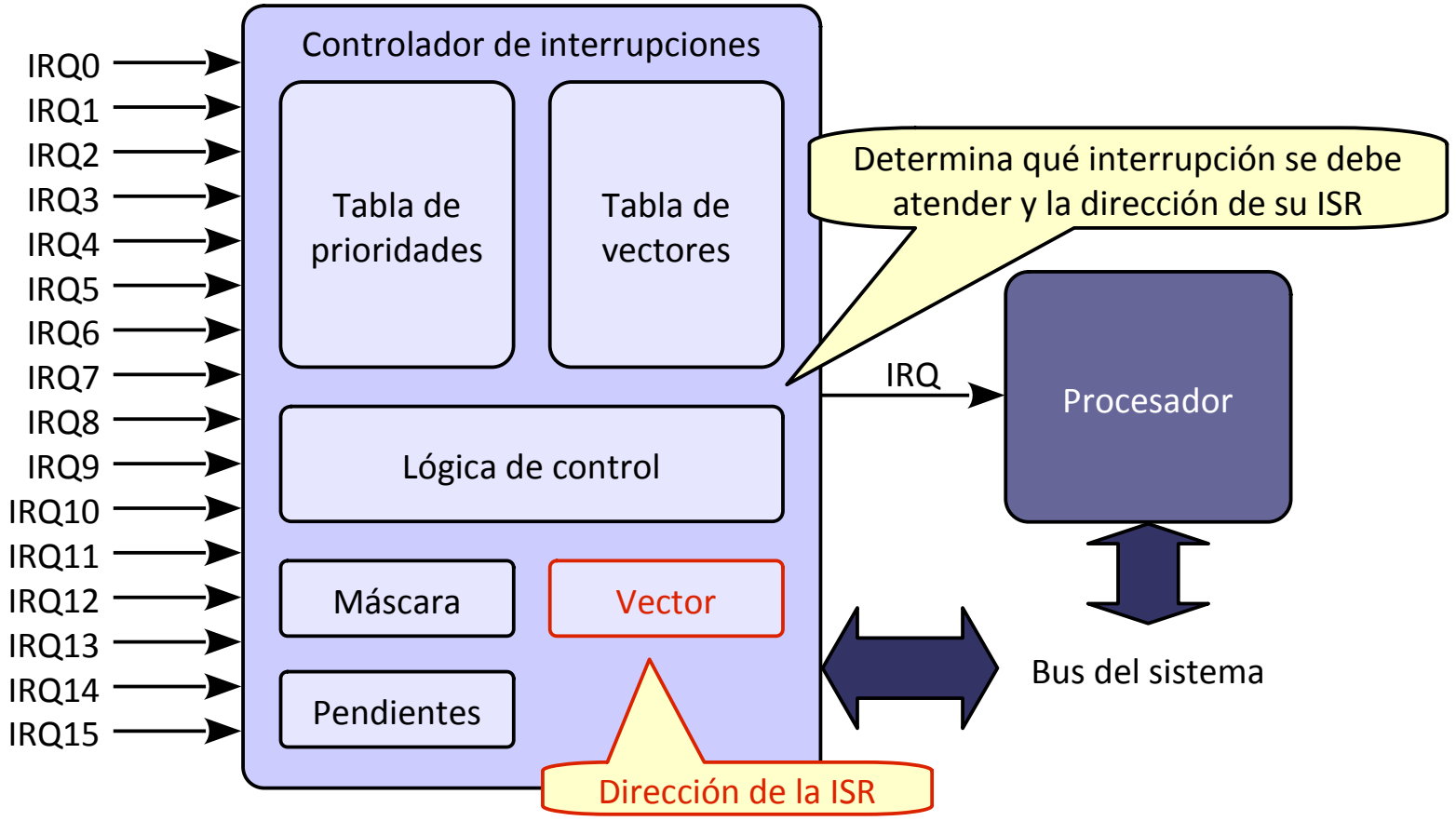



```
    add   sp, sp, #7*4
```

```
@ Cambiar a modo SVC
    mrs   a3, cpsr
    bic   a3, a3, #_MSK_MODE
    orr   a3, a3, #_SVC_MODE
    msr   cpsr_c, a3

@ Crear parte del marco de pila (v1-pc)
    sub   sp, sp, #_FRAME_SIZE-_FRAME_V1

@ Salvar v1-v7 en el marcho de pila
    stmia sp,{v1-v7}

@ Cargar a1-a4, fp, ip, lr de la pila IRQ en v1-v7
    ldmia a2,{v1-v7}

@ Terminar el marco de pila y salvar a1-a4
    stmfd sp!,{v1-v4}

@ Salvar el resto de registros en el marco de pila
    str   v5,[sp, #_FRAME_FP]
    str   v6,[sp, #_FRAME_IP]
    str   v7,[sp, #_FRAME_PC]
    str   a1,[sp, #_FRAME_PSR]
    str   lr,[sp, #_FRAME_LR]

@ Llamar al manejador de nivel 1
    ldr    ip, =bsp_service_irq
    mov    lr, pc
    bx     ip

@ Sección crítica para restaurar el contexto.
@ Restaurar el contexto y retornar al modo original
    msr    cpsr_c, #(_SVC_MODE|_IRQ_DISABLE|_FIQ_DISABLE)
    ldmfd  sp!, {a1-a4,v1-v7,fp,ip,lr}
    msr    spsr_cxsf, lr
    ldmfd  sp!,{lr,pc}^
```

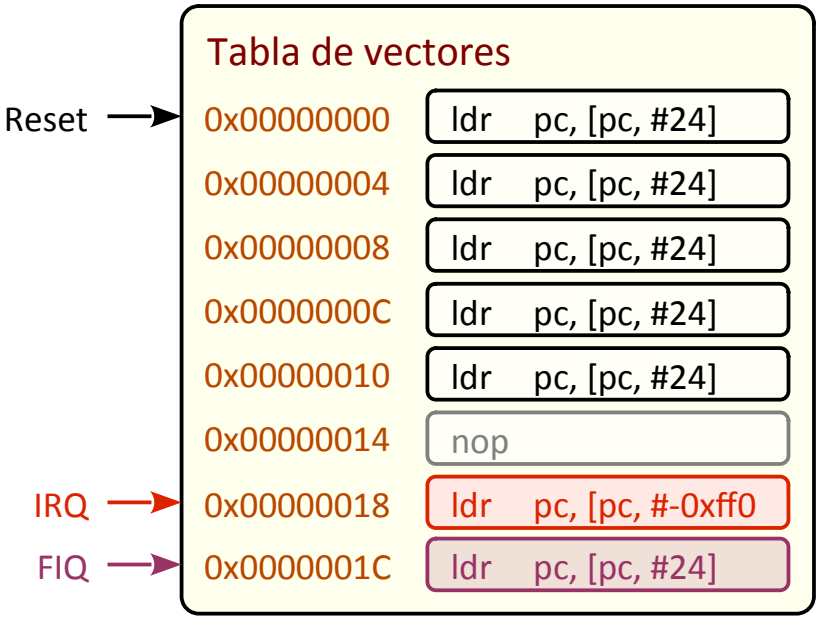
Contenidos

Controlador de interrupciones vectorizado



No necesitamos manejadores, sólo las ISR. La identificación de las fuentes y la gestión de prioridades están implementadas en el VIC

Gestión de interrupciones mediante un VIC



El registro del VIC que contiene el vector de la ISR está mapeado a la dirección `0xffffff030`

PC = 0x00000018 + 8
(por la segmentación del cauce)

0x00000020 – 0x00000ff0 = 0xffffff030

```
    .set  _IRQ_DISABLE,     0x80
    .set  _IRQ_MODE,        0x40
    .set  _SVC_MODE,        0x13
    .set  _VIC_BASE_ADDR,   0xffffff000
    .set  _VIC_VECTOR_ADDR, _VIC_BASE_ADDR + 0x30

vic_ISR:

@ Ajustar lr y salvar los registros  corruptibles
    sub   lr, lr, #4
    stmfd sp!, {a1-a4, fp, ip, lr}

@ Guardar spsr_irq en la pila IRQ
    mrs   ip, spsr
    stmfd sp!, {ip}

<Limpiar la fuente de interrupción>

@ Cambiar a modo SVC, habilitando IRQ y FIQ
    msr   cpsr_c, #_SVC_MODE

<Servicio de la interrupción>

@ Volver a modo IRQ, deshabilitando IRQs
    msr   cpsr_c, #(_IRQ_MODE | _IRQ_DISABLE)

@ Restaurar spsr_irq
    ldmfd sp!, {ip}
    msr   spsr_cxsf, ip

@ Limpiar la petición de IRQ en el VIC
    ldr   a2, =_VIC_VECTOR_ADDR
    str   a1, [a2]

@ Retornar de la ISR
    ldmfd sp!, {a1-a4, fp, ip, pc}^
```

Contenidos

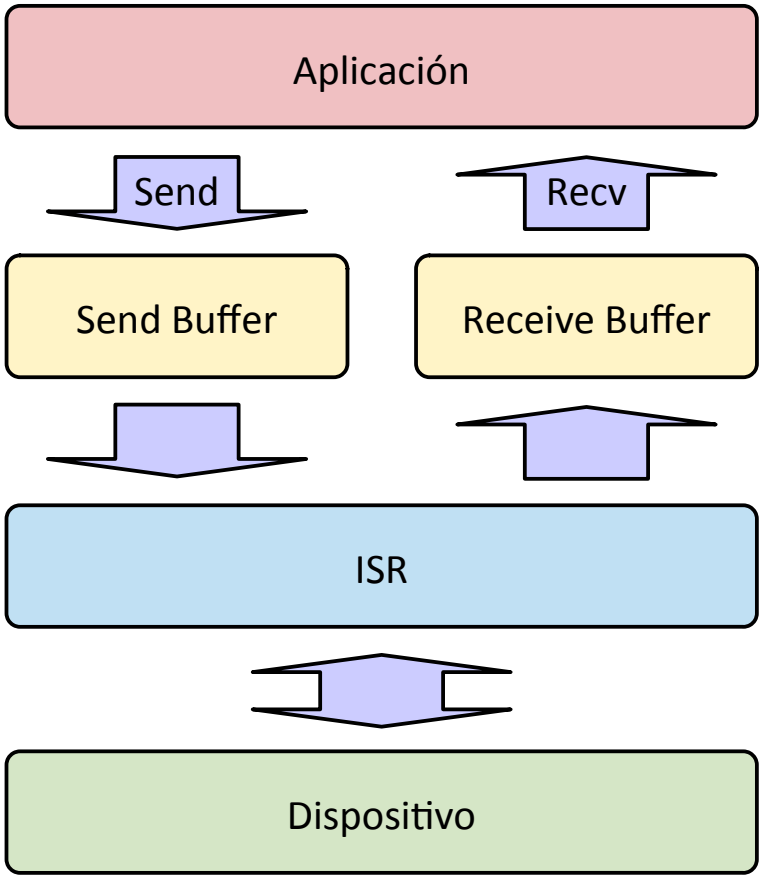
E/S eficiente, flexible y portable

Motivación

Proporciona un API abstracta que aísla a la aplicación de cambios en el hardware

Las llamadas a las funciones de E/S no son bloqueantes

Proporciona interfaces basadas en búferes en vez de en bytes



Para la aplicación, las operaciones de E/S terminan en cuanto se actualiza el búfer correspondiente

No se preocupa de si el dispositivo está disponible, de latencias, anchos de banda, etc.

En cuanto el dispositivo está preparado, interrumpe a la aplicación

La ISR realiza las transferencias entre el dispositivo y los búferes y retorna el control a la aplicación

Arquitecturas Foreground/Background

Motivación

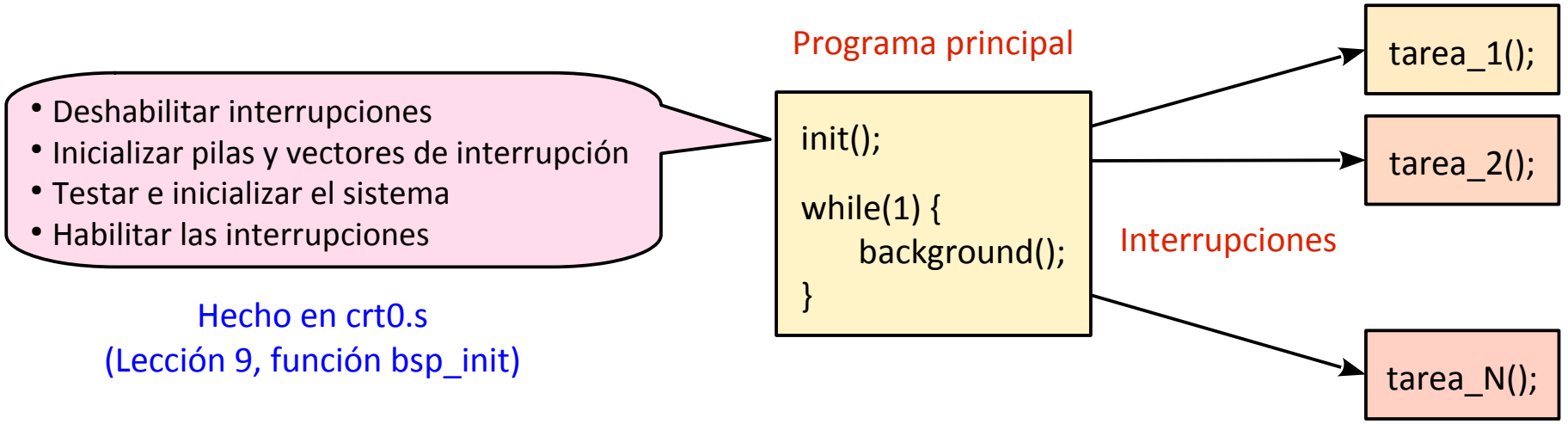
Permiten implementar un Sistema de Tiempo Real sencillo

Es la arquitectura más común en los sistemas empotrados

Funcionamiento

La tarea de mínima prioridad (el programa principal) se ejecuta en *background*, mientras que el resto de tareas (*foreground*) se ejecutan mediante la petición de interrupciones de dispositivos externos

La tarea *background* se puede interrumpir en cualquier momento por cualquier tarea *foreground*. Es común usarla para supervisar la correcta operación del resto de tareas



Lecturas recomendadas

Sistemas Foreground/Background:

P. A. Laplante. *Real-Time Systems Desing and Analysis*. Wiley-IEEE Press, 3ª edición, 2004. Capítulo 3

Ensamblador en línea:

H. Kipp. *ARM GCC Inline Assembler Cookbook*, 2012.
http://www.ethernut.de/en/documents/arm-inline-asm.html

Gestión de interrupciones:

A. N. Sloss, D. Symes, C. Wright. *ARM System Developer's Guide*. Morgan Kaufmann, 2004. Capítulo 9

W. Hohl. *ARM Assembly Language. Fundamentals and Techniques*. CRC Press, 2009. Capítulo 11

M. Samek. *Building Bare-Metal ARM Systems with GNU: Part 6 General Description of Interrupt Handling*. Embedded.com, 2007.

M. Samek. *Building Bare-Metal ARM Systems with GNU: Part 7 Interrupt Locking and Unlocking*. Embedded.com, 2007.

M. Samek. *Building Bare-Metal ARM Systems with GNU: Part 8 Low Level Interrupt Wrapper Functions*. Embedded.com, 2007.

M. Samek. *Building Bare-Metal ARM Systems with GNU: Part 9 C-level ISRs and other ARM Exceptions*. Embedded.com, 2007.