Módulo 2. Entrada/salida

Excepciones/Interrupciones del ARM

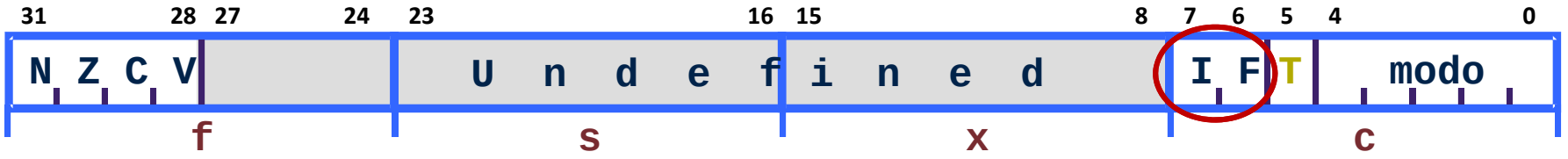
ec

Tipos de Excepciones, Interrupciones

Reset	**cuando se activa la señal externa de reset del sistema**
Undef	**cuando se intenta ejecutar una instrucción no definida**
SWI	**Cuando se ejecuta la instrucción swi (interrupción software)**
IRQ	**Cuando se activa la línea de interrupciones externas IRQ**
FIQ	**Cuando se activa la línea de interrupciones externas rápidas FIQ**
Prefetch Abort	**Cuando se realiza la búsqueda (fetch) de una instrucción en una dirección no válida**
Data Abort	**Cuando se intenta acceder a memoria en una posición no válida, para lectura o escritura de datos**

Habilitación de Interrupciones

- **El procesador puede habilitar o enmascarar las interrupciones mediante su Registro de estado**
  - Dos líneas externas de interrupción FIQ y IRQ
  - Enmascaradas mediante los bit F e I del registro de estado (CPSR)
    - **F=1 →FIQ deshabilitada/enmascarada**
    - **I=1 → IRQ deshabilitada/enmascarada**



Excepciones, Interrupciones

- Cuando se produce una excepción el procesador
  - Salta automáticamente a estas direcciones (**vector de la excepción**)
    - **Son autovectorizadas**
    - **IRQy IFQ pueden ser vectorizadas o no vectorizadas**
  - **Cambia de modo de ejecución**

Prioridad	Excepción	Modo	Vector
1	Reset	SVC	0x00
2	Data Abort	Abort	0x10
3	FIQ	FIQ	0x1C
4	IRQ	IRQ	0x18
6	Prefetch Abort	Abort	0x0C
7	Instruccion no definida	Undef	0x04
8	SWI	SVC	0x08

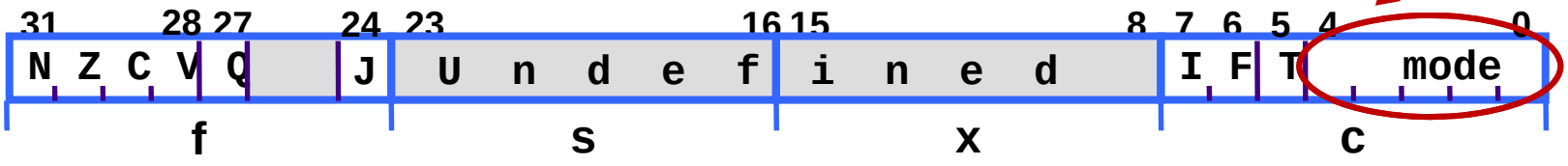
Modos de ejecución

- **Sólo los modos privilegiados permiten escribir en registro de estado** (CPSR)
- Para cambiar del modo usr a cualquier otro modo tiene que ocurrir una excepción o usar la instrucción swi

Modo del procesador	Codigo	Uso
usr	10000	Ejecución de código de usuario
fiq	10001	Servicio de int. rápidas
irq	10010	Servicio de int. lentas
svc	10011	Modo protegido para sistema operativo (int. sw)
abt	10111	Procesado de fallos de acceso a mem
und	11011	Manejo de instrucc. indefinidas
sys	11111	Ejecución de tareas del SO

Modos de excepción

Modos privilegiados



Acceso al Registro de Estado

- Instrucciones de manejo del Registro de estado:

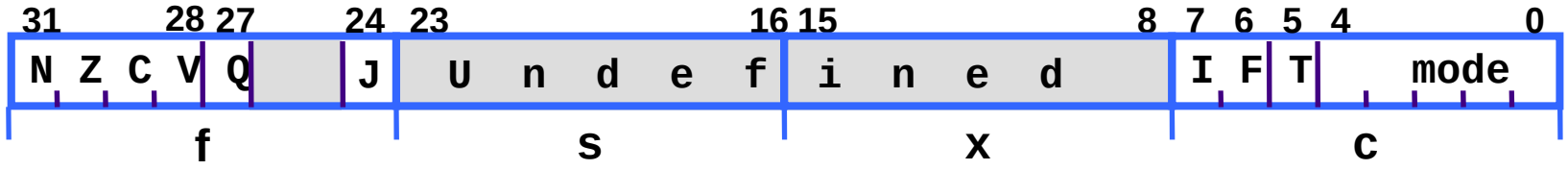
MRS Rd, <psr> ; Rd = <psr>

MSR <psr[_campos]>, Rm ; <psr[_campos]> = Rm

MSR <psr[_campos]>, #N ; <psr[_campos]> = N

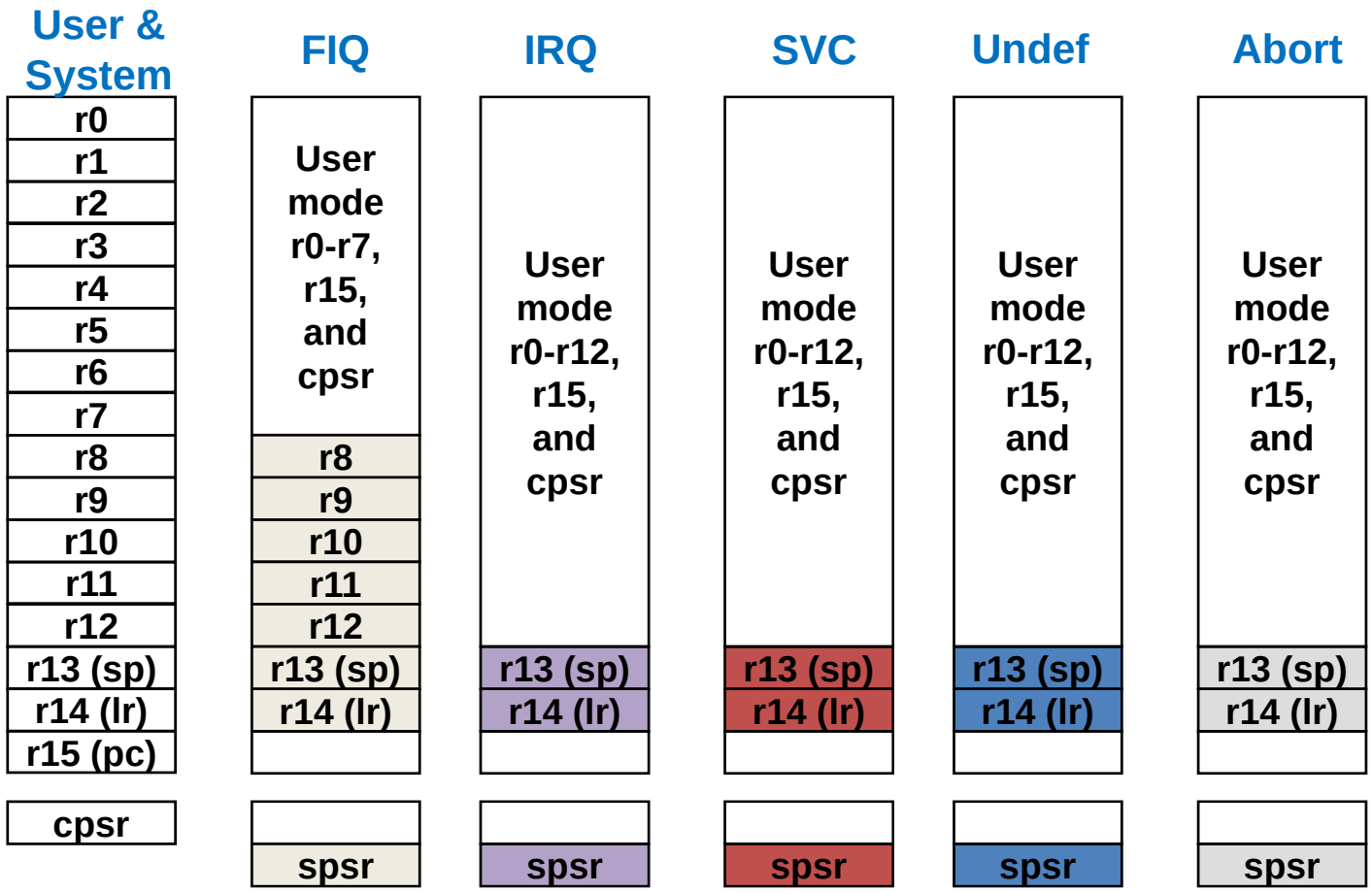
  - Dónde: <psr> = CPSR o SPSR ; [_campos] = combinación de '**fsxc**'

- En **modo "user"**, todos los bits se pueden leer, pero **sólo los flags de condición (_f) se pueden escribir**



Modos de ejecución

- **Cada Modo tiene:**
  - Su propio puntero de pila SP (R13), permite utilizar distintas zonas de memoria para la pila
  - Su propio registro LR (R14)
  - Su propio registro sombra del registro de estado SPSR (Excepto el de usuario)



Gestión de excepciones

- Cuando ocurre excepción el **procesador realiza automáticamente** los siguientes pasos:
  - Guarda la dirección de retorno en LR_<modo>

    R14_<modo_de_excepción> = dirección de retorno

  - Copia el CPSR en SPSR_<modo>

    SPSR_<modo_de_excepción> = CPSR

  - Modifican los bits adecuados del CPSR
    - Cambio a estado ARM → **CPSR[5] = 0**
    - Cambio al modo de excepción → **CPSR[4:0] = código del modo de excepción**
    - Deshabilitar interrupciones (si procede)

      CPSR[7] = 1 /* Deshabilitar interrupciones normales (IRQ) */

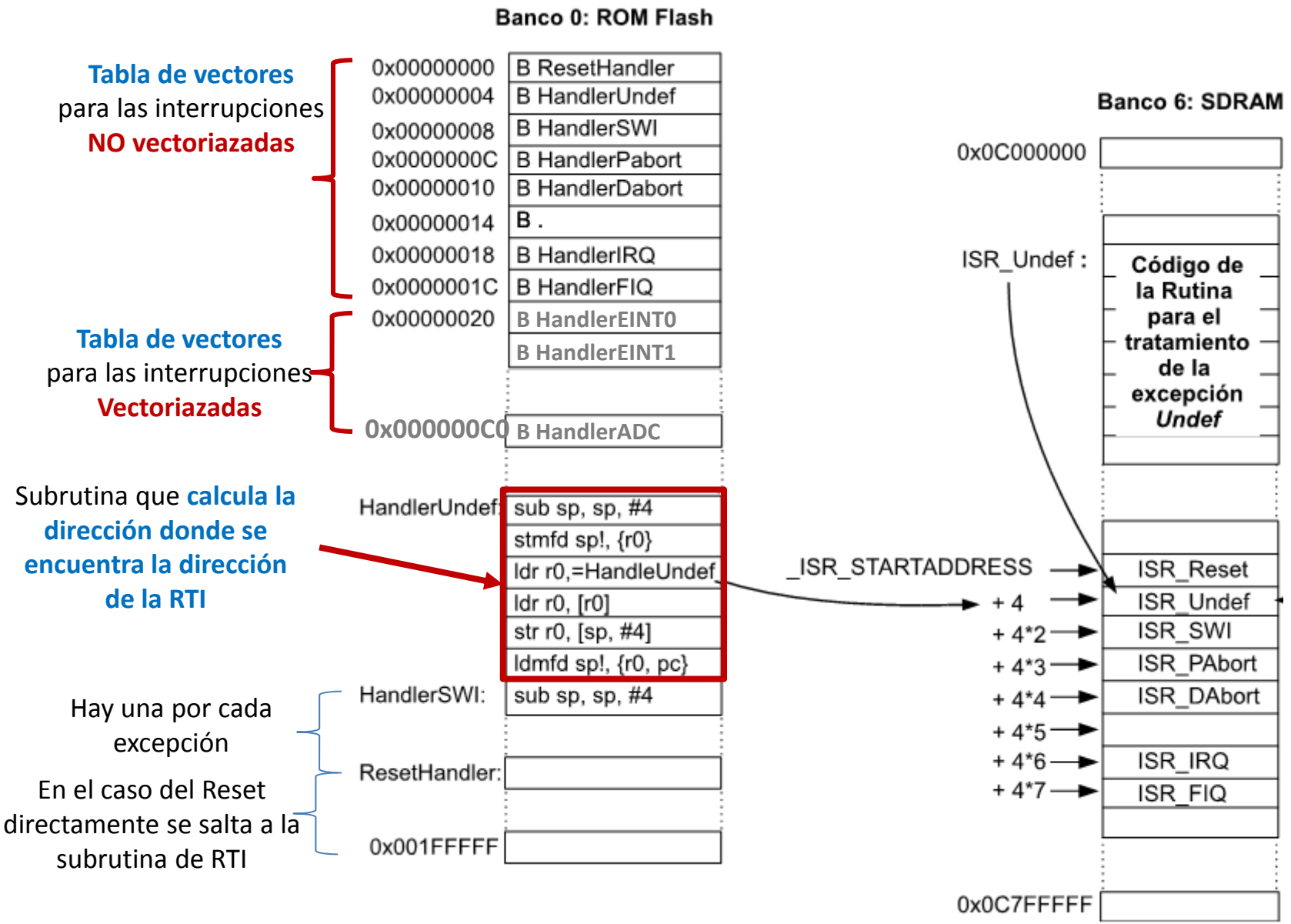
      if <modo_de_excepción> == Reset or FIQ then

      CPSR[6] = 1 /* Deshabilitar interrupciones rápidas */

  - Actualiza el PC con el vector correspondiente

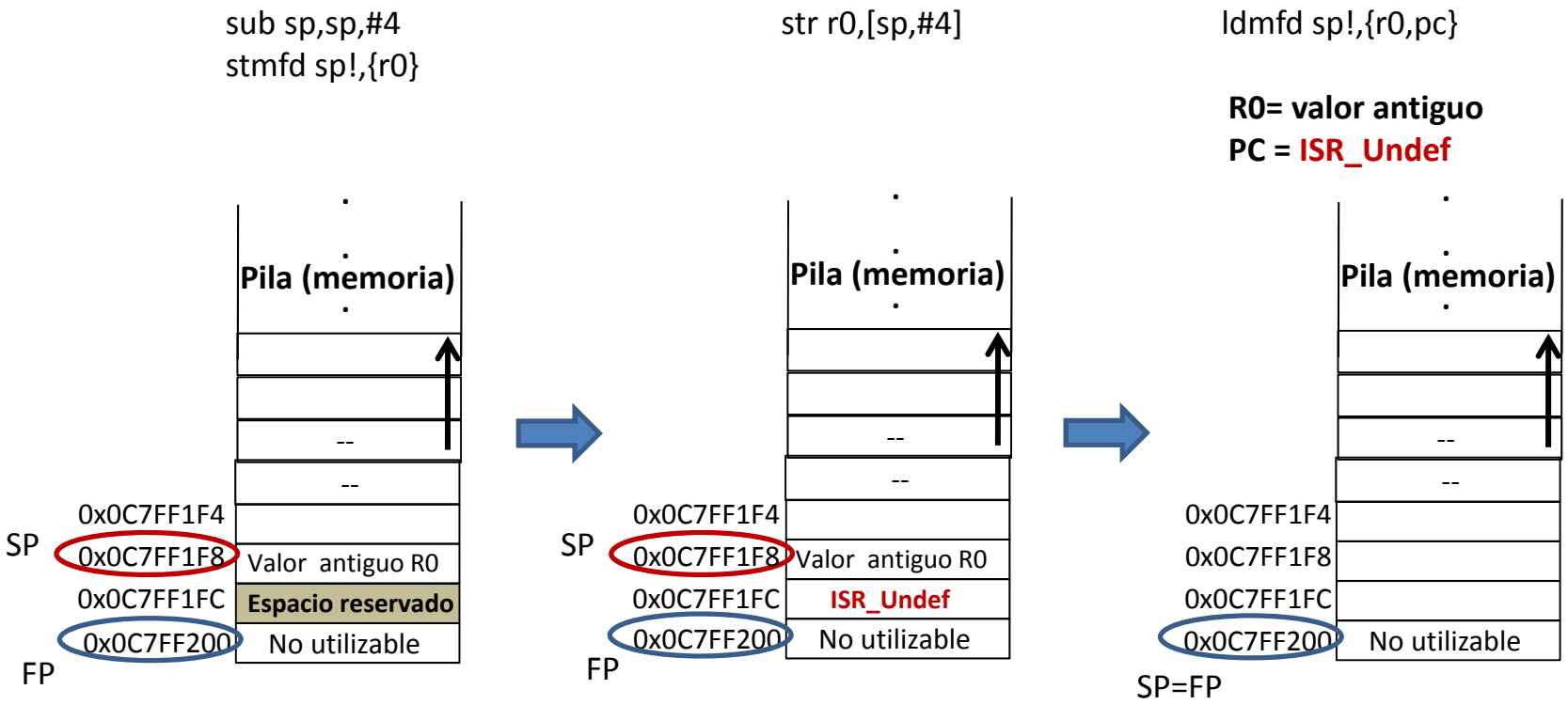
    PC = dirección del vector de excepción

Cómo se calcula la dirección de la RTI

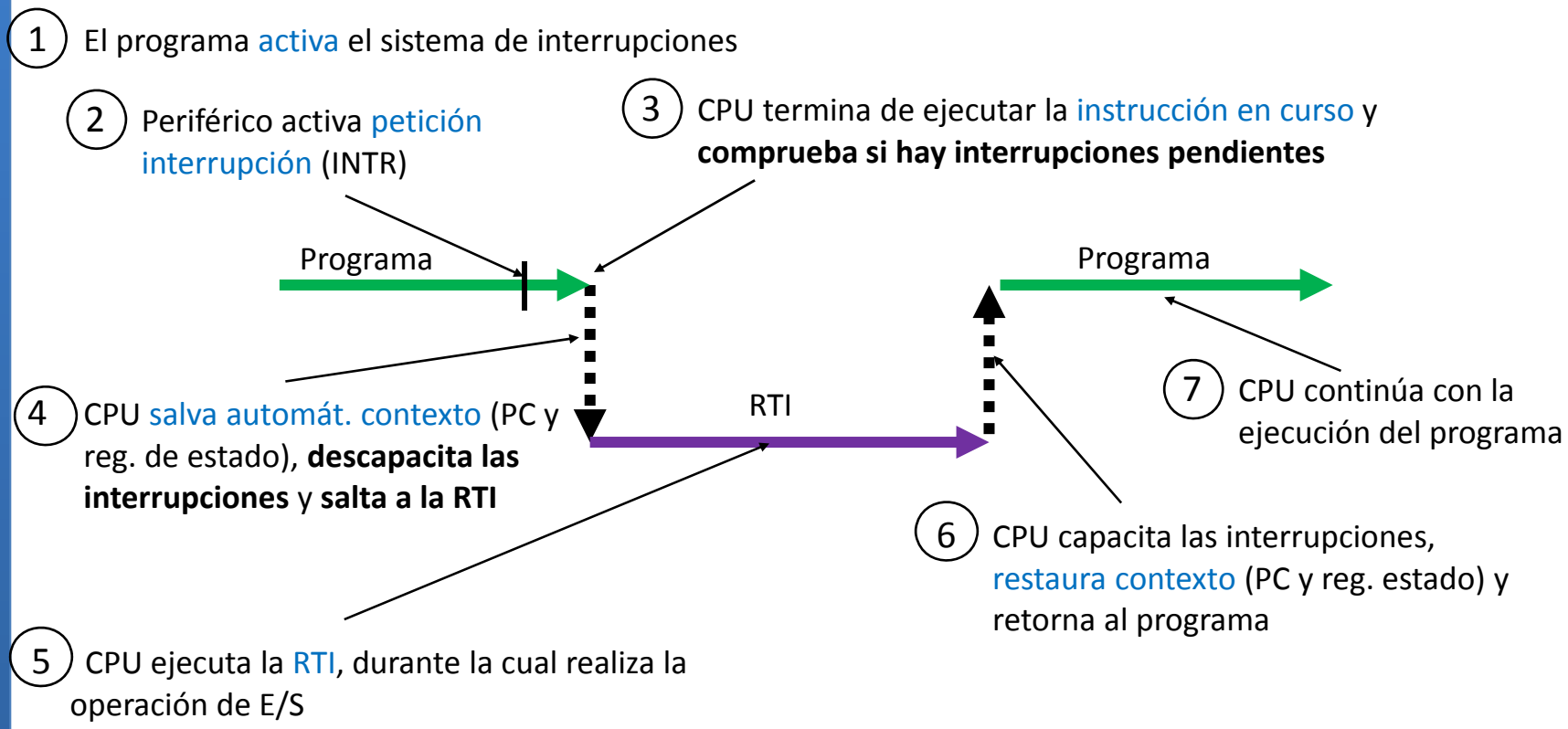


Ejecución de la subrutina HandlerUndef

```
sub sp,sp,#4          /* Reserva una palabra en la pila*/
stmfd sp!,{r0}        /* Salva R0 en la pila */
ldr r0,=HandleUndef   /* Carga en R0 la dirección que tiene la etiqueta HandleUndef = _ISR_STARTADDRESS+4 */
ldr r0,[r0]           /* Carga en R0 el contenido de esta dirección, que será la dirección de comienzo de
                         la rutina de tratamiento de la excepción, en este caso R0 = Dir RTI = ISR_Undef */
str r0,[sp,#4]        /* Almacenamos esta dirección en la posición de la pila que reservamos al comienzo */
ldmfd sp!,{r0,pc}     /* Salta a la RTI*/
```



Secuencia de eventos en el tratamiento de una interrupción



1. El programa activa el sistema de interrupciones

- Para activar el sistema de interrupciones hay que hacer los siguientes pasos:
  - 1.1 **Pasar a modo supervisor**
  - 1.2 Desde el modo supervisor **inicializar la pila (SP) de todos los modos** de ejecución privilegiados
  - 1.3 En el controlador de interrupciones **deshabilitar todas las interrupciones**
  - 1.4 **Habilitar** las líneas **IRQ y FIQ** del procesador
  - 1.5 **Inicializar la tabla de las direcciones** de cada una de las subrutinas de tratamiento de interrupción
  - 1.6 **Eliminamos** posibles **interrupciones pendientes**
  - 1.7 **Configurar el controlador de interrupciones**
  - 1.8 **Configurar periféricos** que generan interrupciones

1. El programa activa el sistema de interrupciones

1.1 Pasar a modo supervisor

- Aprovechamos también para **inicializar su pila**
- El procesador arranca en modo system (**modo privilegiado)** por lo que **permite usar instrucciones para acceder al CPRS**

Código para pasar a modo supervisor

```
.equ MODEMASK, 0x1f      /* 011111 mascara para  trabajar con los bits de modo de CPSR*/
.equ SVCMODE, 0x13       /* Código de modo supervisor */
.equ SVCStack,   c7ff100 /*dirección de comienzo de la pila del modo supervisor*/


mrs r0, cpsr                  /* R0=CPSR*/
bic r0, r0,#MODEMASK     /* R0 and (NOT 0x13) Pone a 0 los bit0 -4 de r0 (bits de modo)*/
orr r1 ,r0,#SVCFMODE     /* R1=R0  or  (código del modo Undef) */
msr cpsr_c, r1                /* Escribimos el resultado en el registro de estado,
                              cambiando de éste los bits del campo control */
ldr sp,=SVCStack          /* Guardamos en SP la dirección de comienzo de la pila de ese  modo*/
```

Nota: recordad que cada modo tiene su registro SP

1. El programa activa el sistema de interrupciones

- **1.2** Desde el modo supervisor **inicializar la pila (SP) de todos los modos de ejecución privilegiados**
  - Crear una subrutina llamada InitStacks
    - Primero hay que cambiar de modo
      - Es igual que el código para cambiar al modo supervisor
    - Una vez en el modo se inicializa el registro SP con la dirección de comienzo de la pila de ese modo

Direcciones donde empieza la pila de cada modo

```
.equ UserStack,     0x0c7ff000
.equ SVCStack,      0x0c7ff100    /* 0x0c7ff000+256 */
.equ UndefStack,    0x0c7ff200    /*0x0c7ff000 +256*2 */
.equ AbortStack,    0x0c7ff300    /*0x0c7ff000 +256*3 */
.equ IRQStack,      0x0c7ff400    /*0x0c7ff000 +256*4 */
.equ FIQStack,      0x0c7ff500    /*0x0c7ff000 +256*5 */
```

Cada zona de pila es de 256Byte

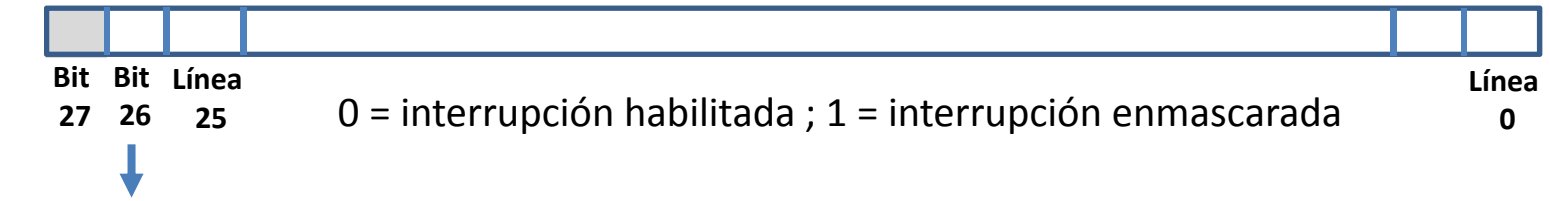
1. El programa activa el sistema de interrupciones

- **1.3 En el controlador de interrupciones deshabilitar todas las interrupciones**
  - Para que mientras estemos en modo supervisor inicializando todo no se atiendan interrupciones
  - En el **registro de mascara del controlador** (INTMSK) **poner un 1 en el bit 26**
    - También se pueden poner a 1 los bits de todas las líneas de interrupción

```
.equ rINTMSK,    0x1e0000c    /*dirección del  Registro de máscara de interrupción */

ldr r0, =rINTMSK              /*R0= dirección del  Registro de máscara de interrupción */
ldr r1, =0x1fffffff           /*R1= pone 1 del bit 0-27,   nota: son 32 bits porque R1 es de 32 bits */
str r1, [r0]
```

INTMSK

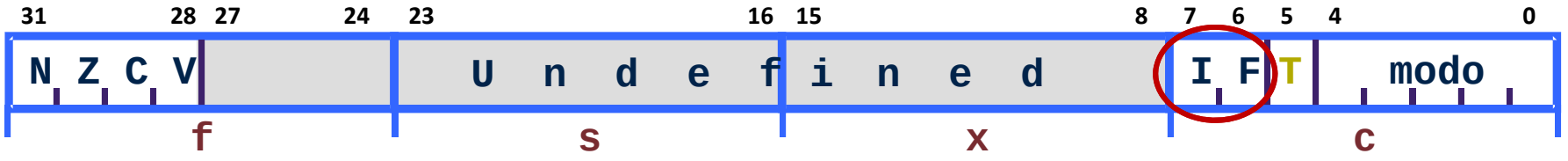


1. El programa activa el sistema de interrupciones

1.4 Habilitar las líneas IRQ y FIQ del procesador

- Hay que poner un "**0**" en los **bits I y F** del registro CPSR
- Se debe hacer al principio del programa

```
mrs r0,cpsr
bic   r0, r0, #0xC0     @R0 = R0 and (NOT 0xC0)
msr   cpsr, r0
```



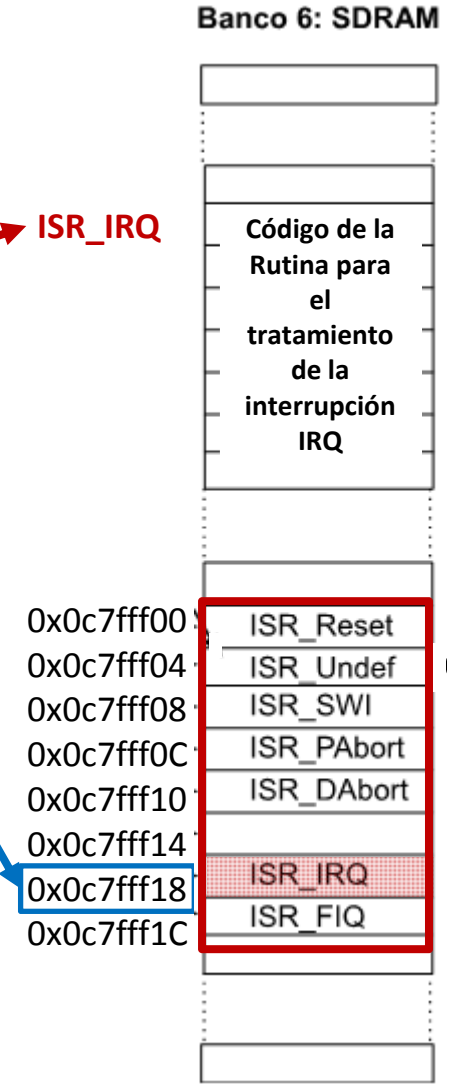
1. El programa activa el sistema de interrupciones

- **1.5** Inicializar **la tabla de las direcciones** de cada una de las subrutinas de tratamiento de interrupción

```
InitISR:                    /*Subrutina para inicializar las direcciones*/

ldr r0,=ISR_IRQ    /*R0= dirección de la RTI para tratar la interrupción IRQ */
ldr r1,=HandleIRQ  /*R1= dirección donde se va a guardar
                          la dirección de la RTI para la interrupción IRQ */
str r0,[r1]
...
...
```

```
.equ   _ISR_STARTADDRESS,   0xc7fff00   /* Banco 6 del mapa de memoria */
.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
```



1. El programa activa el sistema de interrupciones

- 1.6 Eliminamos posibles **interrupciones pendientes**

En nuestro caso los pulsadores del puerto G de la GPIO

- **EXTINTPND**
  - Como estamos inicializando el controlador. **Poner todo a "1" para eliminar las interrupciones pendientes** por si se ha producido alguna interrupción indeseada

- **I_ISPC y F_ISPC**
  - Como estamos inicializando el controlador. **Poner todo a "1" para eliminar las interrupciones pendientes** por si se ha producido alguna interrupción indeseada

En nuestra práctica 2b
lo hacemos en ensamblador (fichero init.S)

1. El programa activa el sistema de interrupciones

- 1.7 **Configurar el controlador de interrupciones**
  - **INTCON** (Interrupt Control Register)
    - Para configurar
      - Si se habilitan las interrupciones vectorizadas o no
      - Si se habilita la línea IRQ o no
      - Si se habilita la línea FIQ o no
  - **INTMOD** (Interrupt Mode Register)
    - Para activar las líneas de interrupción por IRQ, por FIQ o por ambas
  - **INTMSK** (Interrupt Mask Register)
    - Para habilitar las líneas de interrupción que nos interesen

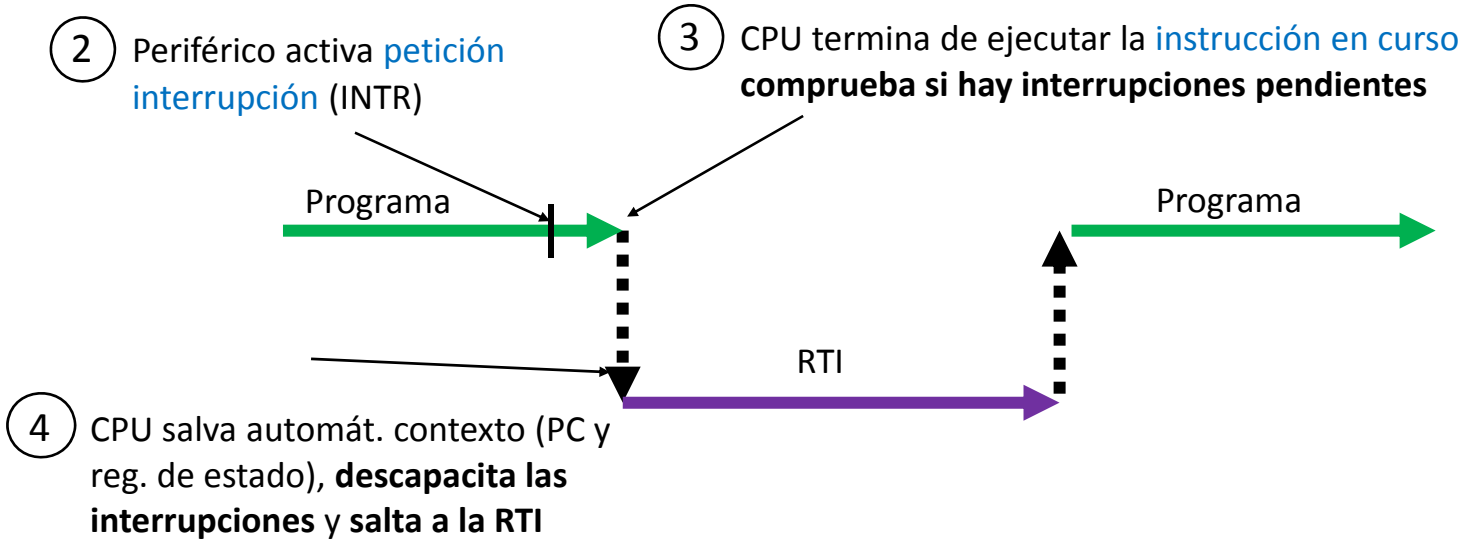
En nuestra práctica 2b
Configuramos el controlador de interrupciones usando las funciones del fichero intcontroller.c
Lo hacemos en el main.c, en la función setup()

1. El programa activa el sistema de interrupciones

- 1.8 **Configurar periféricos** que generan interrupciones
  - En nuestro caso los **pulsadores del puerto G de la GPIO**
    - **PCONG**
      - Configurar los botones para que a**ctivar una petición de interrupción** por las líneas **EINT*** del controlador de interrupciones
    - **EXINT**
      - Elegir el modo con el que la int. externa se dispara (flanco de subida, bajada …)
  - Configurar **el timer**

En la práctica 2b
Configuramos el **puerto G** usando las funciones del fichero **gpio.c**
Configuramos el **timer** usando las funciones del fichero **timer.c**
Lo hacemos en el **main.c**, en la función **setup()**

Secuencia de eventos en el tratamiento de una interrupción



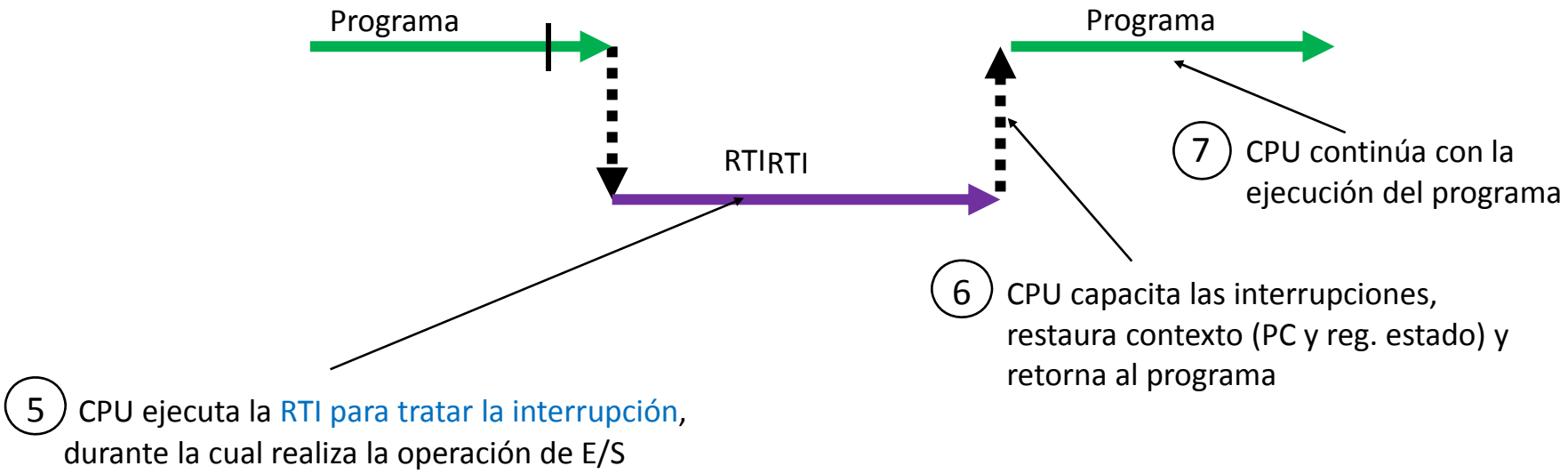
(2)

Cuando se produce una interrupción **se pone un 1** en **el bit correspondiente** del registro **I_ISPR** del controlador de interrupciones y **se activa la línea IRQ o FIQ** según esté configurado el controlador

(3) (4)

Estos pasos los hace automáticamente el procesador, ver la transparencia **"Gestión de excepciones"**

Secuencia de eventos en el tratamiento de una interrupción



5. CPU ejecuta la RTI

- **Interrupciones NO vectorizadas**
  - **Existe sólo una RTI** que se llamaremos irq_ISR o fiq_ISR
  - Esta RTI se encargará de:
    - **Descubrir que línea** de las que entra en el controlador de interrupciones **ha interrumpido**
      - Hará una encuesta accediendo al registro **I_ISPR** del controlador de interrupciones para saber que línea ha sido
    - **Llamar a la subrutina de tratamiento del periférico** asociado a esa línea
      - **Esta subrutina no es propiamente una rutina de tratamiento de interrupción**, ya que es invocada desde irq_ISR**, por lo que se programa como una subrutina normal**
      - **Si es la línea 21** esta subrutina tiene que **leer** el registro **EXTINTPND** del puerto G del la GPIO para saber quién de los 4 dispositivos que tiene esa línea ha sido

5. CPU ejecuta la RTI

- **Interruciones Vectorizadas**
  - Existe **una RTI para cada línea de interrupción** que entra en el controlador de interrupciones
  - Cada una de estas subrutinas trata específicamente la interrupción del periférico asociado a esa línea
    - **Si es la línea 21** tiene que **leer** el registro **EXTINTPND** del puerto G del la GPIO para saber quién de los 4 dispositivos que tiene esa línea ha sido

5. Estructura de una RTI

Esta estructura es sólo para una RTI

/* prólogo */

push {r0-r10,fp, lr}

- **Importante ahora se guardan los registros desde R0**
  - Sólo guardar los registros que se usen
- Guardar LR sólo si es NO hoja (llama a otra subrutina)
- Guardar FP sólo si se quiere trabajar con el marco de pila

add fp,sp,#(4*NumRegistrosApilados-4) @sólo si se va a usar el marco de pila

/*cuerpo de la rutina */

…

…

-Si va a llamar a otra subrutina hay que guardar r0-r3 si los va a usar a la vuelta

Borrar el flag de interrupción (bit correspondiente a la línea que acaba de atenderse del **registro I_ISPC del controlador de interrupciones)**

Si ha sido **la línea 21, borrar el bit** correspondiente del **registro EXTINPND** antes de borrar el bit de ISPC

/* epílogo */

sub sp,fp, #(4*NumRegistrosApilados-4)

pop {r0-r10, fp, lr} @ restauramos contexto y retornamos

Inst retorno de subrutina → la vuelta de las RTI es un poco especial, ver siguiente transparencia

5. CPU ejecuta la RTI

- Para regresar al punto donde se había interrumpido la ejecución del programa, **la rutina de tratamiento de excepción tiene que hacer simultáneamente (de lo contrario el retorno no sería correcto) dos cosas**:
  - **Restaurar el valor del CPSR a partir del valor guardado en el SPSR_<modo>**
  - **Restaurar el PC a partir del LR de cada modo siguiendo las indicaciones de la tabla**

Excepción	Inst. Retorno
Reset	NA
Data Abort	SUBS PC, R14_abt, #8
FIQ	SUBS PC, R14_fiq, #4
IRQ	SUBS PC, R14_irq, #4
Prefetch Abort	SUBS PC, R14_abt, #4
Undef	MOVS PC, R14_und
SWI	MOVS PC, R14_svc

IMPORTANTE
El emsamblador no reconoce el nombre R14_modo.
Hay que poner R4 o LR

Cuando se utiliza el PC como destino en una instrucción que modifica el registro de estado, el hardware automáticamente restaura el valor de CPSR a partir del valor de SPSR

Si la RTI se programa en C en vez de en ensamblador

- Para implementar las RTI como funciones de C
  - Hay que **informar al compilador que no es una función normal sino que la función se utilizará para el tratamiento de una interrupción**
    - En gcc esto se consigue del siguiente modo:

TYPE puede ser "**IRQ", "FIQ",**

Tipo nombreRTI(params) __attribute__((interrupt (TYPE)));

  - Hay que Inicializar su dirección **en la tabla de direcciones de RTIs**

pIRS_TYPE = (unsigned) **nombreRTI**;

Donde pIRS_TYPE está previamente definido

```
#define _ISR_STARTADDRESS 0xc7fff00
#define pISR_RESET        (*(unsigned *)(_ISR_STARTADDRESS+0x0))
#define pISR_UNDEF        (*(unsigned *)(_ISR_STARTADDRESS+0x4))
#define pISR_SWI          (*(unsigned *)(_ISR_STARTADDRESS+0x8))
#define pISR_PABORT       (*(unsigned *)(_ISR_STARTADDRESS+0xc))
#define pISR_DABORT       (*(unsigned *)(_ISR_STARTADDRESS+0x10))
#define pISR_RESERVED     (*(unsigned *)(_ISR_STARTADDRESS+0x14))
#define pISR_IRQ          (*(unsigned *)(_ISR_STARTADDRESS+0x18))
#define pISR_FIQ          (*(unsigned *)(_ISR_STARTADDRESS+0x1c))
…
#define pISR_TIMER0       (*(unsigned *)(_ISR_STARTADDRESS+0x54))
#define pISR_EINT4567     (*(unsigned *)(_ISR_STARTADDRESS+0x74))
..
```

Si la RTI se programa en C en vez de en ensamblador

- Por ejemplo para la rutina de tratamiento de interrupción irq_ISR
  - Declarar que es una RTI

```
void  irq_ISR( params ) __attribute__((interrupt ( "IRQ" )));
```

  - Inicializar su dirección **en la tabla de direcciones de RTIs**

```
#define pISR_IRQ    (*(unsigned *)(_ISR_STARTADDRESS+0x18))

pIRS_IRQ = (unsigned) irq_ISR;
```