# *Tema 4*

# *Sistema de Excepciones en Cortex M3*

# Indice

# 4.1. Introducción

- Como sabemos, el Procesador puede ejecutar las acciones que se le soliciten a traves de un programa.
- Es habitual pedirle al Procesador que tambien sea capaz de reaccionar ante distintos eventos que se produzcan en su entorno.
- La forma de definir lo que tiene que hacer el procesador como respuesta/reaccion a cada evento, es ejecutando una rutina para cada uno de los eventos posibles, donde se situen un conjunto de instrucciones que codifican dicha respuesta.
- Se define un mecanismo para dotar al Procesador de la posibilidad de responder a esos eventos (excepciones o interrupciones)

# 4.1. Introducción

- **Mecanismo de Atencion a la Interrupcion**
  - Paso 1: El procesador reconoce que se ha producido el evento
  - Paso 2: El procesador comprueba si en su configuracion actual debe responder al evento o ignorarlo→Habilitacion de las Interrupciones.
  - Paso 3: El procesador compara la "importancia" (prioridad) del evento (respecto al resto de eventos que en ese momento se hayan producido). Si su importancia es superior, pasa inmediatamente al paso 4. En caso contrario, espera su turno.
  - Paso 4. El procesador busca la direccion de la memoria de programa donde se encuentra la Rutina que contiene las instrucciones programadas para reaccionar al evento (Manejador de Interrupcion ó Rutina de atencion a la Interrupcion, Interrupt Serving Routine, ISR)

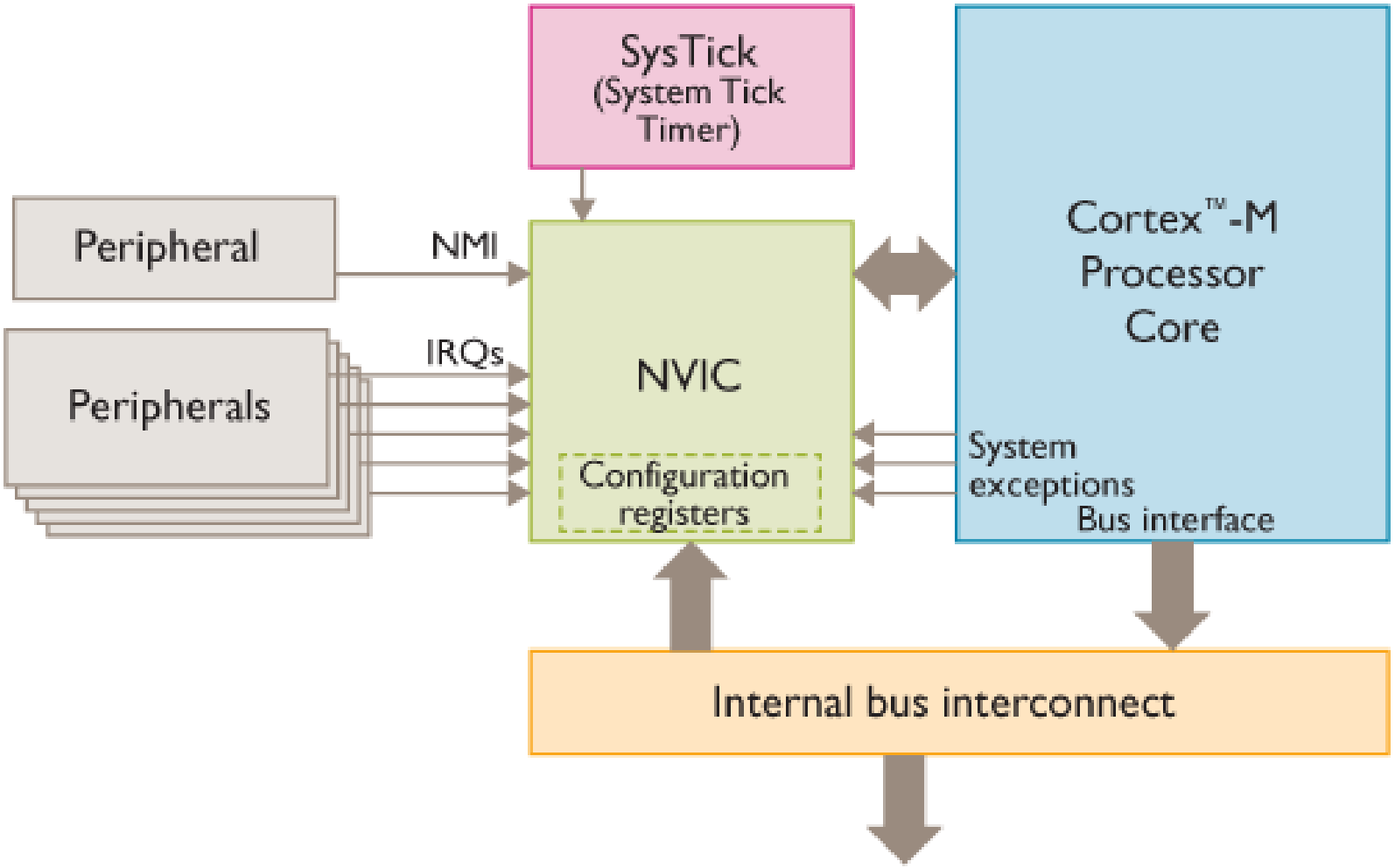
# 4.1. Introducción

- **Mecanismo de Atencion a la Interrupcion**
  - **Paso 5. Se guarda en la pila el "contexto" de la ejecución que se este llevando a cabo (contexto es todo aquello que define la ejecucion)**
  - **Paso 6. Se copia en el CP el valor obtenido en el paso 4. Se comienza a ejecutar la ISR adecuada.**
  - **Paso 7. Una vez finalizada la ejecución de todas las instrucciones de la ISR, se restablede desde la pila el contexto guardado en el paso 5.**
  - **Paso 8. Se continua con la ejecución de la aplicacion en las mismas condiciones anteriores al evento.**

# 4.2. Cortex-M3: NVIC

- Excepcion es cualquier circunstancia que para la ejecucion actual del procesador y le conmuta a realizar otra atencion diferente (mediante la ejecucion de otro codigo)
- En Cortex-M3, toda la gestion de las excepciones del procesador se realiza desde un bloque: El Nested Vectored Interrupt Controller (NVIC)
- Para eso tiene en su interior una serie de registros que permiten configurar el comportamiento del uP ante la llegada de excepciones
- Cortex-M3 distingue entre dos tipos de excepciones:
  - Excepciones de sistema: Aquella generadas dentro del uP (del numero 1 al 15)
  - Interrupciones: Aquellas generadas fuera del uP (excepcion 16 en adelante). Dos tipos:
    - Generadas dentro del chip, por algun elemento incluido en el uControlador
    - Generadas desde fuera del chip→Interrupciones externas.

# 4.2. Cortex-M3: NVIC



# 4.2. Cortex-M3: NVIC

| Exception Number | Exception Type | Priority | Description |
|---|---|---|---|
| 1 | Reset | –3 (Highest) | Reset |
| 2 | NMI | –2 | Nonmaskable interrupt (external NMI input) |
| 3 | Hard fault | –1 | All fault conditions if the corresponding fault handler is not enabled |
| 4 | MemManage fault | Programmable | Memory management fault; Memory Protection Unit (MPU) violation or access to illegal locations |
| 5 | Bus fault | Programmable | Bus error; occurs when Advanced High-Performance Bus (AHB) interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access) |
| 6 | Usage fault | Programmable | Exceptions resulting from program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor) |
| 7–10 | Reserved | NA | — |
| 11 | SVC | Programmable | Supervisor Call |
| 12 | Debug monitor | Programmable | Debug monitor (breakpoints, watchpoints, or external debug requests) |
| 13 | Reserved | NA | — |
| 14 | PendSV | Programmable | Pendable Service Call |
| 15 | SYSTICK | Programmable | System Tick Timer |

# 4.2. Cortex-M3: NVIC

Table 50. Connection of interrupt sources to the Vectored Interrupt Controller

| Interrupt ID | Exception Number | Vector Offset | Function | Flag(s) |
|---|---|---|---|---|
| 14 | 30 | 0x78 | SSP0 | Tx FIFO half empty of SSP0<br>Rx FIFO half full of SSP0<br>Rx Timeout of SSP0<br>Rx Overrun of SSP0 |
| 15 | 31 | 0x7C | SSP 1 | Tx FIFO half empty<br>Rx FIFO half full<br>Rx Timeout<br>Rx Overrun |
| 16 | 32 | 0x80 | PLL0 (Main PLL) | PLL0 Lock (PLOCK0) |
| 17 | 33 | 0x84 | RTC | Counter Increment (RTCCIF)<br>Alarm (RTCALF) |
| 18 | 34 | 0x88 | External Interrupt | External Interrupt 0 (EINT0) |
| 19 | 35 | 0x8C | External Interrupt | External Interrupt 1 (EINT1) |
| 20 | 36 | 0x90 | External Interrupt | External Interrupt 2 (EINT2) |
| 21 | 37 | 0x94 | External Interrupt | External Interrupt 3 (EINT3).<br>**Note:** EINT3 channel is shared with GPIO interrupts |
| 22 | 38 | 0x98 | ADC | A/D Converter end of conversion |
| 23 | 39 | 0x9C | BOD | Brown Out detect |
| 24 | 40 | 0xA0 | USB | USB_INT_REQ_LP, USB_INT_REQ_HP, USB_INT_REQ_DMA |
| 25 | 41 | 0xA4 | CAN | CAN Common, CAN 0 Tx, CAN 0 Rx, CAN 1 Tx, CAN 1 Rx |
| 26 | 42 | 0xA8 | GPDMA | IntStatus of DMA channel 0, IntStatus of DMA channel 1 |
| 27 | 43 | 0xAC | $I^2S$ | irq, dmareq1, dmareq2 |
| 28 | 44 | 0xB0 | Ethernet | WakeupInt, SoftInt, TxDoneInt, TxFinishedInt, TxErrorInt, |

# 4.2. Cortex-M3: NVIC

- El número de exception 16 y siguientes son consideradas interrupciones

| Exception Number | Exception Type | Priority |
|---|---|---|
| 16 | External Interrupt #0 | Programmable |
| 17 | External Interrupt #1 | Programmable |
| … | … | … |
| 255 | External Interrupt #239 | Programmable |

# 4.2. Cortex-M3: NVIC

- NVIC es una parte integrada del Cortex™-M3
  - Gestiona tanto las interrupciones como las excepciones de sistema
  - Para eso, utiliza registros accesibles por el procesador mapeados en memoria (es decir, se puede acceder a ellos como si fueran posiciones de memoria)
  - NVIC tambien contiene registro para configuracion del temporizador de sistema (SYSTICK), y para depuracion
  - NVIC soporta 1–240 interrupciones [IRQ] (más 15 excepciones de sistema). En LPC17XX se implementan unicamente 35 excepciones
  - NVIC tambien permite la Interrupcion No Enmascarable (Nonmaskable Interrupt, NMI)
  - La mayoria de las excepciones tienen una prioridad configurable, salvo 3:
    - Reset
    - HardFault
    - NMI

# 4.2. Cortex-M3: NVIC

- El valor de la excepcion que se encuentra ejecutando en cada momento en el uP se puede obtener:
  - En el Registro de Estado de Interrupcion (IPSR)
  - Y en el campo VECTACTIVE del Registro de Estado de Control de Interrupcion (Interrupt Control State Register, ICSR)
- Para la gestion de interrupciones el NVIC utiliza una serie de registros, que son
  - Registros de Habilitación/Deshabilitacion de Intrrupciones (Interrupt Set/ Clear Enable Registers, ISER and ICER)
  - Registros de Estableciminto/Borrado de Interrupcion Pendiente (Interrupt Set-Pending and Interrupt Clear-Pending Registers, ISPR and ICPR)
  - Registros de Prioridad de Interrupcion (Interrupt Priority Registers, IPRs)
  - Registros de Bit Interrupcion Activa (Interrupt Active Bit Registers, IABR)

# 4.2. Cortex-M3: NVIC

- Todos están a su vez controlados por los registros de contro general del NVIC:
  - Registro de Control de Reset y Aplicacion de Interrupcion (App. Interrupt and Reset Control Reg, AIRCR)→ Importante su campo Priority group.
  - Registro de Interrupcion de Trigger Software (Software Trigger Interrupt Register,STIR)
- Además, otros registros en el µP fuera del NVIC puede afectar a la ejecucion de las excepciones:
  - Registro de Mascara de Excepciones (PRIMASK, FAULTMASK, and BASEPRI)
  - Registro de Offset de la Tabla de Vectores (VTOR)

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 1. Reconocimiento de la interrupcion**:
  - En Cortex-M3, cuando se produce una excepción, automáticamente pasa al estado de pendiente.
  - Existen en NVIC los registros ISPR e ICPR relacionados con esto.
- Interrupt Set-Pending and Clear-Pending Registers
  - En LPC17XX son dos registros ISPR (ISPR0, ISPR1) y dos ICPR (ICPR0, ICPR1)
  - Registros de 32 bits con un bit asociado a cada interrupcion
  - Si se produce una excepcion, pasará a estar pendiente hasta que se comience a ejecutar su ISR asociadael it will be pended
  - El estado de pendiente de una excepcion se puede modificar mediante software, escribiendo el bit asociado del registro ICPR o ISPR:
    - Registo ICPRs para cancelar una interrupcion que se encuentre en el estado de pendiente
    - Registro ISPRs para generar una interrupcion por software.
- El registro Interrupt Control and State Register (en el campo VECTPENDING field) tambien recogerá el valor de la excepcion pendiente

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 1. Reconocimiento de la interrupcion**:
- El mismo efecto de escritura en el registro ISPR se puede conseguir con el registro Software Trigger Interrupt Register (STIR)
  - Registro de 8 bit que almacena la interrupcion que se quiere activar por software
  - No permite la generacion de excepciones de sistema

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 2. Habilitacion de la interrupcion:**
  - En Cortex-M3, cuando una interrupcion se encuentra pendiente, a continuacion se mira si esta permitido que esa interrupcion sea tratada realizando el resto de pasos (esta habilitada) o no (deshabilitada).
    - Si se encuentra habilitada, se pasará al paso 3
    - Si no se encuentra habilitada, se quedará pendiente en espera de que se habilite para que continue con el resto de pasos.
  - Existen en NVIC los registros ISER e ICER relacionados con esto.
- Interrupt Set and Clear Enable Registers
  - En LPC17XX son dos registros ISER (ISER0, ISER1) y dos ICER (ICER0, ICER1)
  - Registros de 32 bits con un bit asociado a cada interrupción
  - El programador podra configurar en cada instante qué interrupciones quiere permitir que se traten o cuales no

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 3. Prioridad de la interrupcion:**
  - En Cortex-M3, si existen un conjunto de interrupciones pendientes y habilitadas, se permite clasificar cual de ellas será la que se trate a continuacion (sigua con los pasos 4,5,…) y cuales esperarán hasta que finalice la elegida
  - Para eso, a cada fuente de interrupcion se le puede asignar un número (prioridad) que indica la urgencia de dicha interrupcion en su tratamiento.
  - Cortex-M3 implementa un sistema de interrupciones con desalojo→ una interrupcion con mayor prioridad que la que se encuentre en ejecucion, pausará la ejecución de ésta inmediantamente para que el uP la atienda por tener mayor urgencia.
  - Con Cortex-M3, el número de prioridad que se puede asignar a cada interrupcion va desde 0 hasta 255. →Mayor prioridad(urgencia), menor numero
  - Existen excepciones que tienen una prioridad fija no configurable.
    - Reset, NMI, HardFault (-3,-2 y -1 respectivamente). Mas prioritarias que el resto.
  - Existen en NVIC los registros IPR relacionados con esto, además del registro AIRCR.

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 3. Prioridad de la interrupcion:**
- Interrupt Priority Registers (IPRs) en Cortex-M3
  - Registros dedicados para asignar la prioridad para cada posible interrupcion.
  - Cada interrupcion tiene asignados 8 bits para configurar su prioridad.
    - Hasta 256 números de prioridad distintos.
  - IPRs son registros de 32 bits, por lo que cada uno de ellos alberga la prioridad de 4 interrupciones.
  - Cortex-M3 permite que los uControladores que incluyan su procesador no utilicen los 8 bits que ofrece para configurar la prioridad→ Entre 3 y 8 bits.
  - En caso de que un uC no quiera utilizar los 8 bits que ofrece Cortex-M3, reducirá comenzando por los bits menos significativos (LSB)

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Implemented | | | Not implemented | | | | |

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Implemented | | | | Not implemented | | | |

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 3. Prioridad de la interrupcion:**
- Interrupt Priority Registers (IPRs) en Cortex-M3
  - En Cortex-M3, cada prioridad asignada a una interrupcion consta de dos campos:
    - Prioridad de desalojo (the preempt priority)
    - Subprioridad
  - El número de bits asignados a cada campo es configurable.
    - Se configura en el campo PRIGROUP del registro Application Interrupt and Reset Control Register (AIRCR) del NVIC.

**Table 7.4** Application Interrupt and Reset Control Register (Address 0xE000ED0C)

| Bits | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 31:16 | VECTKEY | R/W | — | Access key; 0x05FA must be written to this field to write to this register, otherwise the write will be ignored; the read-back value of the upper half word is 0xFA05 |
| 15 | ENDIANNESS | R | — | Indicates endianness for data: 1 for big endian (BE8) and 0 for little endian; this can only change after a reset |
| 10:8 | PRIGROUP | R/W | 0 | Priority group |
| 2 | SYSRESETREQ | W | — | Requests chip control logic to generate a reset |
| 1 | VECTCLRACTIVE | W | — | Clears all active state information for exceptions; typically used in debug or OS to allow system to recover from system error (Reset is safer) |
| 0 | VECTRESET | W | — | Resets the Cortex-M3 processor (except debug logic), but this will not reset circuits outside the processor |

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 3. Prioridad de la interrupcion:**
- Como se definen los campos en funcion del Priority Group?

**Table 7.5** Definition of Preempt Priority Field and Subpriority Field in a Priority Level Register in Different Priority Group Settings

| Priority Group | Preempt Priority Field | Subpriority Field |
|---|---|---|
| 0 | Bit [7:1] | Bit [0] |
| 1 | Bit [7:2] | Bit [1:0] |
| 2 | Bit [7:3] | Bit [2:0] |
| 3 | Bit [7:4] | Bit [3:0] |
| 4 | Bit [7:5] | Bit [4:0] |
| 5 | Bit [7:6] | Bit [5:0] |
| 6 | Bit [7] | Bit [6:0] |
| 7 | None | Bit [7:0] |

**Priority Group** define el bit MAS ALTO que se usa para la Subprioridad en los IPRs

- Prioridad de desajolo (Preempt Priority) define si una excepcion puede ejecutar cuando el procesador ya se encuentra ejecutando otro manejador de interrucion (ISR)
- Subprioridad (Subpriority) define que excepcion ejecutará antes, si varias excepciones con la misma prioridad de desalojo se encuentran pendientes al mismo tiempo

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 3. Prioridad de la interrupcion:**

Table 660. Priority grouping

| | Interrupt priority level value, PRI_N[7:0] | | | Number of | |
|---|---|---|---|---|---|
| PRIGROUP | Binary point[1] | Group priority bits | Subpriority bits | Group priorities | Subpriorities |
| b010 | bxxxxx.000 | [7:3] | none | 32 | 1 |
| b011 | bxxxx.y000 | [7:4] | [3] | 16 | 2 |
| b100 | bxxx.yy000 | [7:5] | [4:3] | 8 | 4 |
| b101 | bxx.yyy000 | [7:6] | [5:3] | 4 | 8 |
| b110 | bx.yyyy000 | [7] | [6:3] | 2 | 16 |
| b111 | b.yyyyy000 | None | [7:3] | 1 | 32 |

[1] PRI_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit. Bits [2:0] are not used in LPC17xx devices.

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 3. Prioridad de la interrupcion:**
- Interrupt Priority Registers (IPRs) en LPC17xx
  - En LPC17XX:
    - Hay 9 registros IPR de 32 bits (IPR0-IPR8), para configurar la prioridad de las 35 interrupciones con prioridad probramable.
    - El numero de bits que fija para configurar la prioridad de una interrupcion son 5 bits

# 4.3. Mecanismo de atencion a la excepcion

## Ejemplos En LPC17XX

- If PRIGROUP = 3, then 5 bits used to format peripheral interrupt level will be defined as XXXX : Y where XXXX – groups (can have total16 groups), and Y – sub-priority within each group – (can have total 2 sub-priorities within each group)
- If PRIGROUP = 5, then 5 bits used to format peripheral interrupt level will be defined as XX : YYY where XX – groups (can have total 4 groups), and YYY – sub-priority within each group – (can have total 8 sub-priorities within each group)
- If PRIGROUP = 2, then 5 bits used to format peripheral interrupt level will be defined XXXXX where XXXXX – groups (can have 32 groups), via software - cannot set sub-priority within group.

| PRIGROUP (3 bits) | Binary Point (Group: sub) | | Pre-empting Priority (Group Priority) | | Sub-Priority (Sub-Level) | |
|---|---|---|---|---|---|---|
| | | | Bits | Levels | Bits | Levels |
| 2 | 5:0 | 5 bits for groups, 0 bit for sub-priority | 5 | 32 | 0 | 0 |
| 3 | 4:1 | 4 bits for groups, 1 bit for sub-priority | 4 | 16 | 1 | 2 |
| 4 | 3:2 | 3 bits for groups, 2 bits for sub-priority | 3 | 8 | 2 | 4 |
| 5 | 2:3 | 2 bits for groups, 3 bits for sub-priority | 2 | 4 | 3 | 8 |
| 6 | 1:4 | 1 bit for group, 4 bits for sub-priority | 1 | 2 | 4 | 16 |
| 7 | 0:5 | 0 bit for group, 5 bits for sub-priority | 0 | 0 | 5 | 32 |

# 4.3. Mecanismo de atencion a la excepcion

## Paso 3. Prioridad de la interrupcion:

- **Example:** We have two interrupts in our system - EINT0 and GPIO Interrupt and we want to assign both interrupts under the same group with different sub-priorities where GPIO Interrupt has a higher sub-priority than EINT0
  - We can set PRIGROUP to 4. We can have 8 groups total and 4 sub-levels within each group.
  - We set priority field (bit 23:19) of EINT0 to 2 so it will 00010 which means EINT0 is from group 0 and has sub-priority of 2 within group 0
  - We set priority field (bit 15:11) of GPIO Interrupt (EINT3) to 1 so it will be 00001 which means GPIO Interrupt (EINT3) is from group 0 and has sub-priority of 1 within group 0
  - So if both interrupts take place at the same time, GPIO Interrupt will be serviced first and then EINT0 Interrupt (example of Tail-Chaining)

| PRIGROUP (3 bits) | Binary Point (Group: sub) | | Pre-empting Priority (Group Priority) | | Sub-Priority (Sub-Level) | |
|---|---|---|---|---|---|---|
| | | | Bits | Levels | Bits | Levels |
| 2 | 5:0 | 5 bits for groups, 0 bit for sub-priority | 5 | 32 | 0 | 0 |
| 3 | 4:1 | 4 bits for groups, 1 bit for sub-priority | 4 | 16 | 1 | 2 |
| 4 | 3:2 | 3 bits for groups, 2 bits for sub-priority | 3 | 8 | 2 | 4 |
| 5 | 2:3 | 2 bits for groups, 3 bits for sub-priority | 2 | 4 | 3 | 8 |
| 6 | 1:4 | 1 bit for group, 4 bits for sub-priority | 1 | 2 | 4 | 16 |
| 7 | 0:5 | 0 bit for group, 5 bits for sub-priority | 0 | 0 | 5 | 32 |

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 3. Prioridad de la interrupcion:**
- Registros de Mascara de Interrupcion: **PRIMASK, FAULTMASK, BASEPRI**

| Register Name | Description |
|---|---|
| PRIMASK | A 1-bit register. When this is set, it allows NMI and the hard fault exception; all other interrupts and exceptions are masked. The default value is 0, which means that no masking is set. |
| FAULTMASK | A 1-bit register. When this is set, it allows only the NMI, and all interrupts and fault handling exceptions are disabled. The default value is 0, which means that no masking is set. |
| BASEPRI | A register of up to 9 bits (depending on the bit width implemented for priority level). It defines the masking priority level. When this is set, it disables all interrupts of the same or lower level (larger priority value). Higher-priority interrupts can still be allowed. If this is set to 0, the masking function is disabled (this is the default). |

  - PRIMASK, FAULTMASK, BASEPRI son registros internos del procesador
  - Fijan en cada momento el nivel de prioridad a partir del que ninguna interrupcion de prioridad inferior pasará a tratarse.

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 4. Búsqueda de la direccion de memoria donde comienza la ISR:**
- La interrupcion pendiente, habilitada y con mayor prioridad de las que se encuentran en el sistema (y mayor prioridad que el nivel fijado en los registros de máscara de interrupcion) pasará a continuación a ser tratada.
- Su tratamiento consiste en la ejecucion de la ISR (manejador o handler) asociada.
- Para ejecutarla, el uP necesita conocer la direccion de la memoria de programa donde se encuentra la primera instruccion de la ISR
- Para eso, Cortex-M3 utiliza el sistema de Tabla de Vectores.
- Se define (en una direccion fija de la memoria de programa) una Tabla con una fila por Interrupcion
  - **Dicha fila contedrá la direccion donde comienza la ISR asociada a la interrupcion**
- Esta Tabla se guarda en memoria no volatil

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 4. Búsqueda de la direccion de memoria donde comienza la ISR:**
  - Por defecto, la Tabla de Vectores comienza en la direccion 0 de la memoria de programa.
    - Cortex-M3 permite realojar la Tabla a otras direcciones de memoria (Vector Table Base Address) que cumplan una serie de requisitos, escribiendo esta nueva direccion de comienzo en el registro Vector Table Offset Register (VTOR) en el NVIC.

**Table 7.6** Exception Vector Table After Power Up

| Address | Exception Number | Value (Word Size) |
|---|---|---|
| 0x00000000 | — | MSP initial value |
| 0x00000004 | 1 | Reset vector (program counter initial value) |
| 0x00000008 | 2 | NMI handler starting address |
| 0x0000000C | 3 | Hard fault handler starting address |
| … | … | Other handler starting address |

# 4.3. Mecanismo de atencion a la excepcion

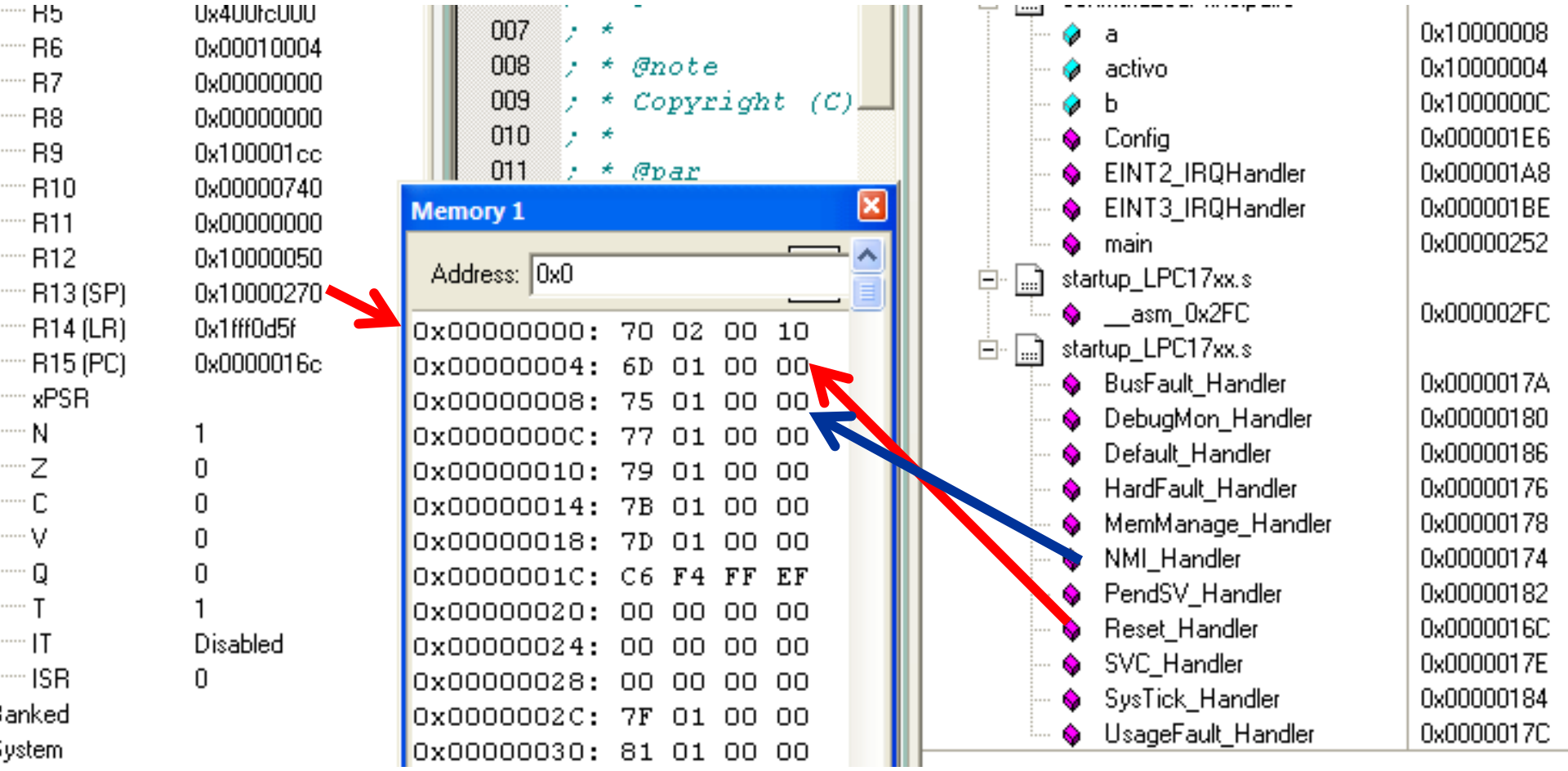
```
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors

__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     MemManage_Handler         ; MPU Fault Handler
                DCD     BusFault_Handler          ; Bus Fault Handler
                DCD     UsageFault_Handler        ; Usage Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     DebugMon_Handler          ; Debug Monitor Handler
                DCD     0                         ; Reserved
                DCD     PendSV_Handler            ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler

                ; External Interrupts
                DCD     WDT_IRQHandler            ; 16: Watchdog Timer
                DCD     TIMER0_IRQHandler         ; 17: Timer0
                DCD     TIMER1_IRQHandler         ; 18: Timer1
                DCD     TIMER2_IRQHandler         ; 19: Timer2
                DCD     TIMER3_IRQHandler         ; 20: Timer3
                DCD     UART0_IRQHandler          ; 21: UART0
                DCD     UART1_IRQHandler          ; 22: UART1
                DCD     UART2_IRQHandler          ; 23: UART2
                DCD     UART3_IRQHandler          ; 24: UART3
                DCD     PWM1_IRQHandler           ; 25: PWM1
                DCD     I2C0_IRQHandler           ; 26: I2C0
                DCD     I2C1_IRQHandler           ; 27: I2C1
                DCD     I2C2_IRQHandler           ; 28: I2C2
```

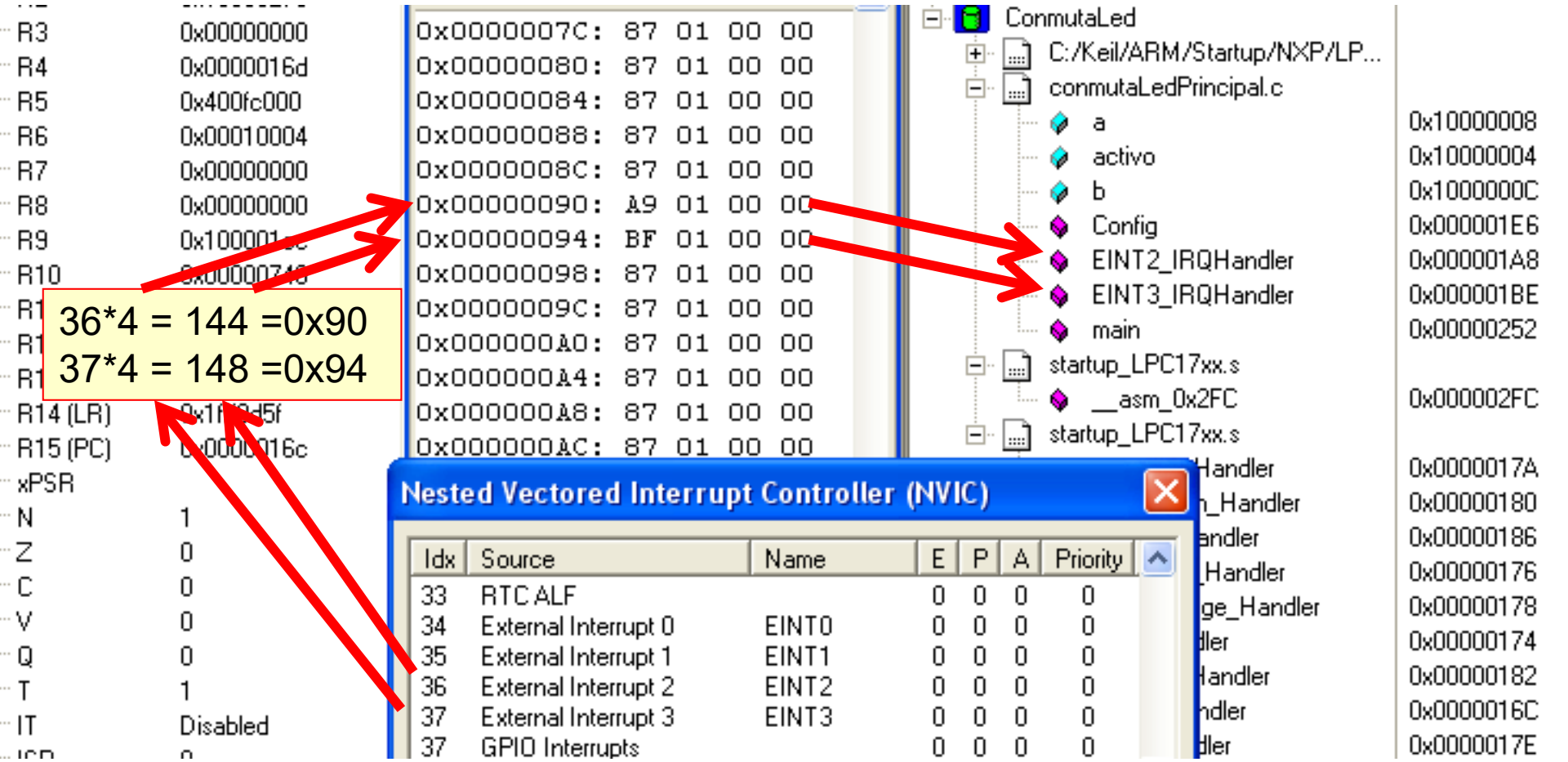
# 4.3. Mecanismo de atencion a la excepcion

- **Paso 4. Búsqueda de la direccion de memoria donde comienza la ISR:**



# 4.3. Mecanismo de atencion a la excepcion

- **Paso 4. Búsqueda de la direccion de memoria donde comienza la ISR:**

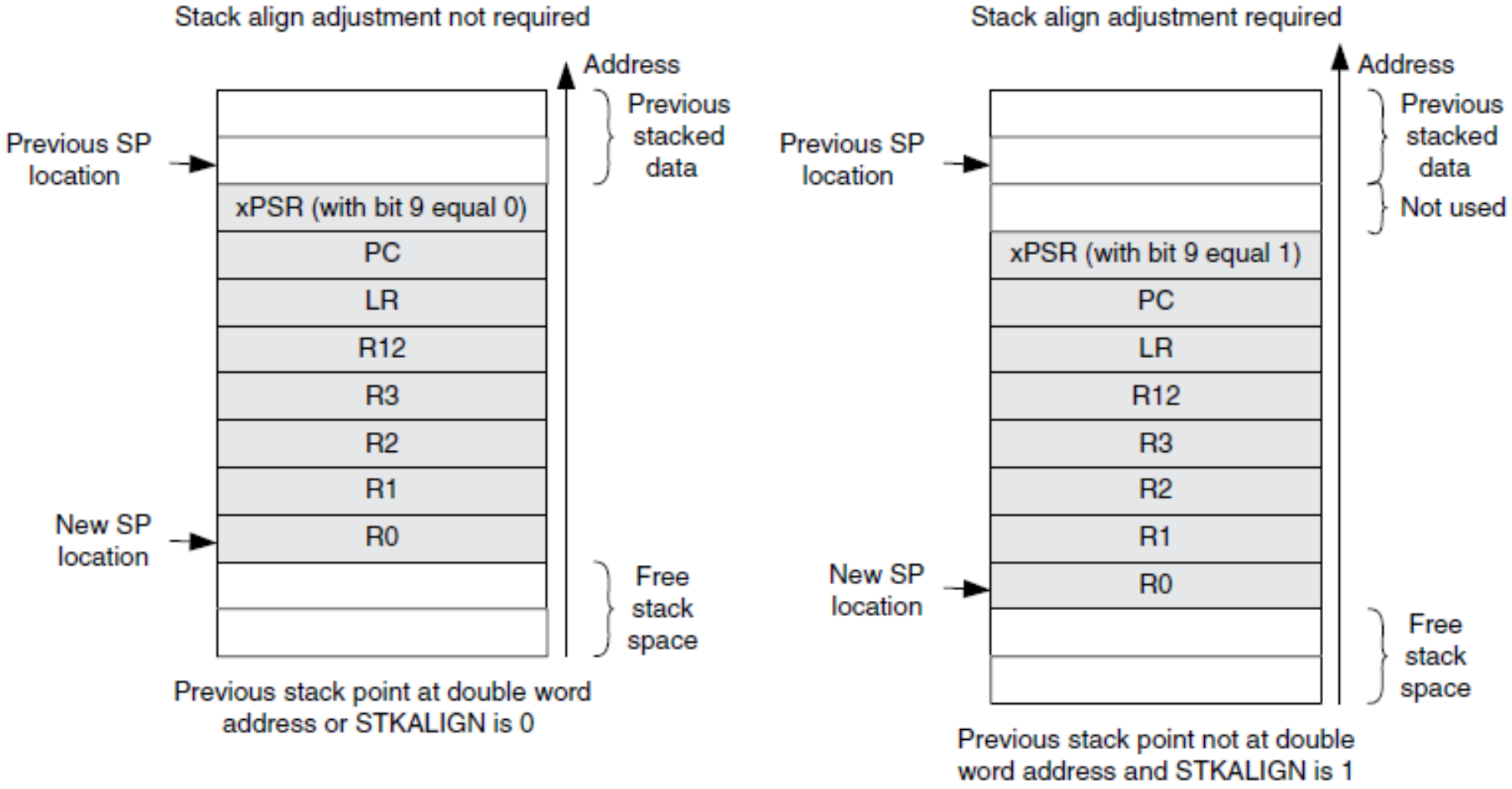


# 4.3. Mecanismo de atencion a la excepcion

- **Paso 5. Guardar Estado en la Pila:**
- Se guardan en la pila mediante una operacion PUSH el valor de los registros que definen el estado de la ejecucion
  - **Automáticamente en Cortex-M3 los registros** R0–R3, R12, LR, PC y PSR
  - El resto de registros
    - si se programa en ensamblador, se guardan manualmente
    - Si se programa en lenguaje de alto nivel, lo hace el compilador.
- Tras la operacion PUSH, varios registros resultan actualizados
    - **SP:** El registro SP se actualiza con el nuevo valor teniendo en cuenta las operaciones PUSH. Durante la ejecucion de la ISR (paso 6) se usa como puntero de pila el MSP
    - **PSR:** El registro IPSR se actualizara para guardar el nuevo numero de excepcion que se trata
    - **LR:** Se actualiza a un valor llamado EXC_RETURN, que conduce al retorno de la excepcion

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 5. Guardar Estado en la Pila:**



# 4.3. Mecanismo de atencion a la excepcion

- **Paso 6. Ejecucion de la ISR:**
- Se carga en el registro PC el valor obtenido de la tabla de vectores en el paso 4 (Direccion donde comienza la ISR).
- Consecuentemente, se comienza a continuacion a ejecutar la primera instruccion de la rutina ISR.
- En ese momento, la interrupcion pasa del estado de pendiente al estado de activa→Registro IABR.
- Dicha ISR contendrá el procesado que se quiere dar para responder al evento.
- Se continua hasta el final de las instrucciones de la ISR.

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 6. Ejecucion de la ISR:**
- Registro Interrupt Active Bit Registers IABR
  - Registro de 32 bit con un bit asociado a cada interrupcion→ muestra si una interrupcion está siendo atendida o no
  - En LPC17XX se definen dos registros IABR (IABR0, IABR1))
  - Cuando un procesador comienza la ejecucion de la ISR, su bit asociado en el registro IABR se pone a “1” y se borra cuando la interrupcion termina la ejecucion de su ultima instrucción
  - Si la interrupcion es desalojada, aunque el procesador esté a tendiendo a una interrupcion de mayor prioridad, el bit de la interrupcion desalojada continuará a “1” indicando que todavia permanece activa

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 7 Restablecer Estado desde la Pila**
  - Al finalizar la ejecución de la ISR, se restablecen desde la pila los registros guardados en el paso 5 (que, como recordamos, contenia el estado (situacion en que se encontraba el procesador cuando se comenzó a tratar la interrupcion))
  - De esta forma, una vez resturado los registros, el sistema se encontrará en la misma situacion en que estaba cuando se comenzó a tratar la IRQ
  - In the Cortex-M3, todo este proceso de restauracion de registros se lleva a cabo cuando dentro de la ISR se ejecuta la sentencia return, de tal forma que toda la ISR se puede implementar como una rutina de C.
  - Adicionalmente, el bit asociado en el registro IABR se pone automáticamente a “0”

# 4.3. Mecanismo de atencion a la excepcion

- **Paso 8 Continua ejecucion previa a la IRQ**
  - Una vez restablecido el Estado en el paso 7, el procesador recupera la situacion que tenia antes de que llegara la interrupcion, y continua con la ejecucion de la instruccion que en aquel momento estuviera ejecutando

# 4.4. Excepciones anidadas

- Ocurre cuando en mitad de la ejecución de la ISR de una IRQ, se genera una IRQ habilitada y de mayor prioridad.
- En este caso, se pausa la ejecucion de la ISR y se desencadena el mecanismo de atención a la interrupcion de la interrupcion recien generada, ya que es lo más prioritario
  - En el paso 5 de este mecanismo recien lanzado se guardara el estado de lo que se esta ejecutando (ISR menos prioritara)
- Cuando finalice el mecanismo de atencion de esta IRQ más prioritaria, se continuará con la ejecución de la ISR que se tenia anteriormente
  - En el paso 7 del mecanismo de atencion de la ISR mas prioritaria se restablecera el estado previo (ejecución ISR menos prioritara)
- Se ha analizado la situacion de dos excepciones anidadas, pero en un momento pueden estar anidadas mas excepciones
  - El mecanismo definido permite que se atiendan perfectamente
  - Se debe tener en cuenta el tamaño de la pila.

# 4.5 Principales Registros del NVIC

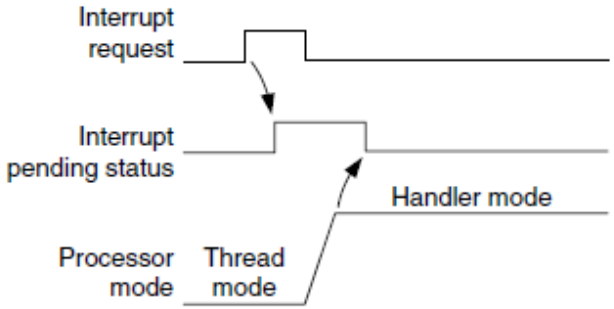
| Name | Description | Access | Reset value |
|---|---|---|---|
| ISER0 to ISER1 | Interrupt Set-Enable Registers. These 2 registers allow enabling interrupts and reading back the interrupt enables for specific peripheral functions. | RW | 0 |
| ICER0 to ICER1 | Interrupt Clear-Enable Registers. These 2 registers allow disabling interrupts and reading back the interrupt enables for specific peripheral functions. | RW | 0 |
| ISPR0 to ISPR1 | Interrupt Set-Pending Registers. These 2 registers allow changing the interrupt state to pending and reading back the interrupt pending state for specific peripheral functions. | RW | 0 |
| ICPR0 to ICPR1 | Interrupt Clear-Pending Registers. These 2 registers allow changing the interrupt state to not pending and reading back the interrupt pending state for specific peripheral functions. | RW | 0 |
| IABR0 to IABR1 | Interrupt Active Bit Registers. These 2 registers allow reading the current interrupt active state for specific peripheral functions. | RO | 0 |
| IPR0 to IPR8 | Interrupt Priority Registers. These 9 registers allow assigning a priority to each interrupt. Each register contains the 5-bit priority fields for 4 interrupts. | RW | 0 |

# 4.6. Creando aplicaciones con interrupciones

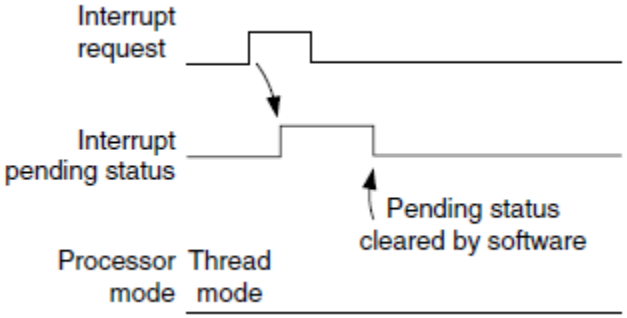
- Cuando se crea una aplicacion que va a manejar Interrupciones, se deben realizar:
- En primer lugar, se debe crear las rutinas que contengan las instrucciones de respuesta a cada interrupcion de la aplicacion.
  - En Cortex-M3, por defecto, estas rutinas acaban en …Handler()
- Se establece un tamaño de Pila que permita anidar las interrupciones de la aplicacion
- A continuacion, en la aplicacion principal se debe configurar
  - La direccion en la que se alojará Tabla de Vectores→Registro VTOR (por defecto, 0)
  - Establecer la direccion inicial de la pila.
  - Prioridad. Se clasifican las excepciones en funcion de su prioridad, y se escribe en los registros.
  - Finalmenet, se habilitan las excepciones a las que se permita interrumpir

# 4.7. Ejemplos de Excepciones

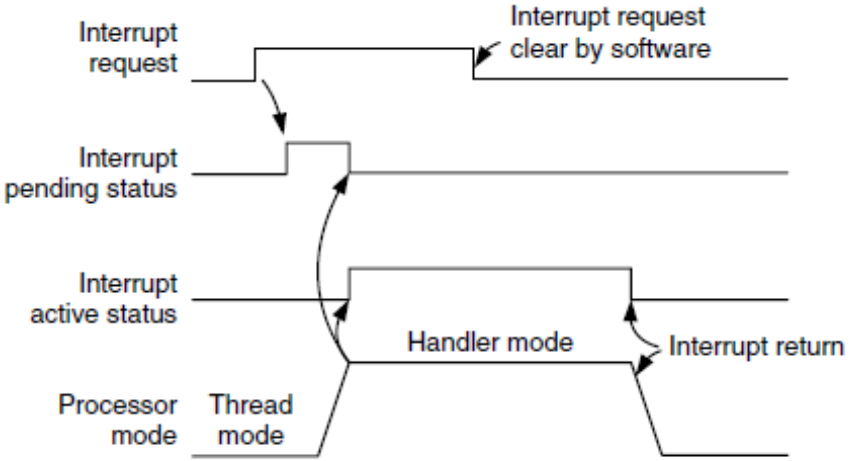
- Ejemplos:
  - Temporizacion



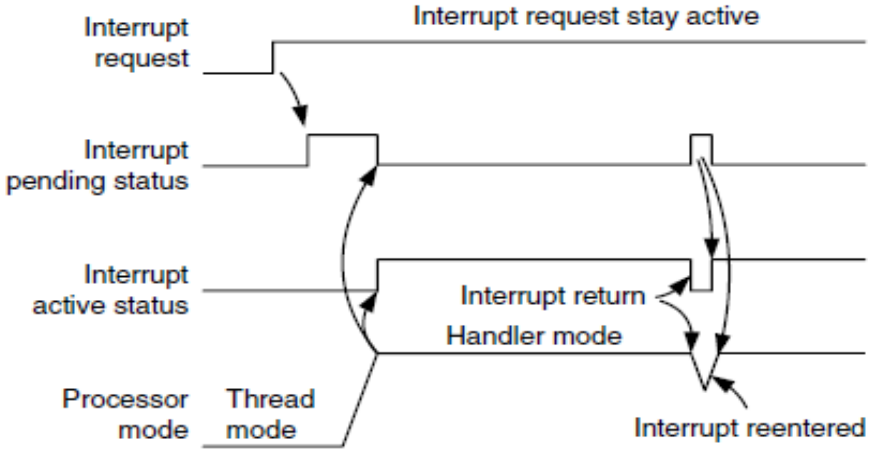

  - Cancelacion por software




  - Comportamiento del bit asociado en el registro AIBR he ISR execution

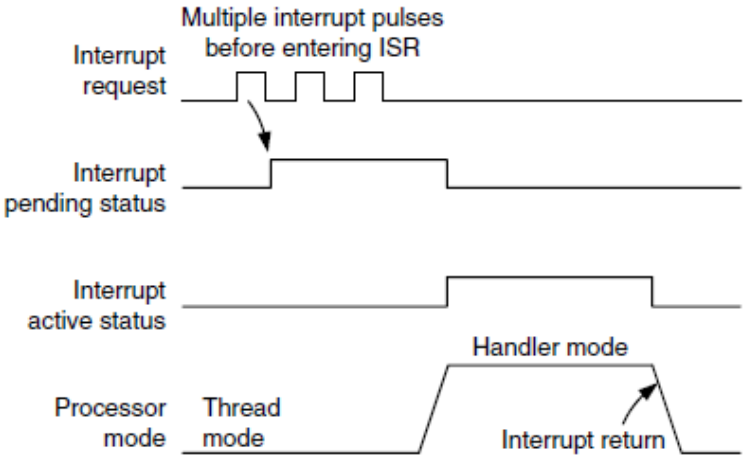



  - Solicitud continua de interrupcion

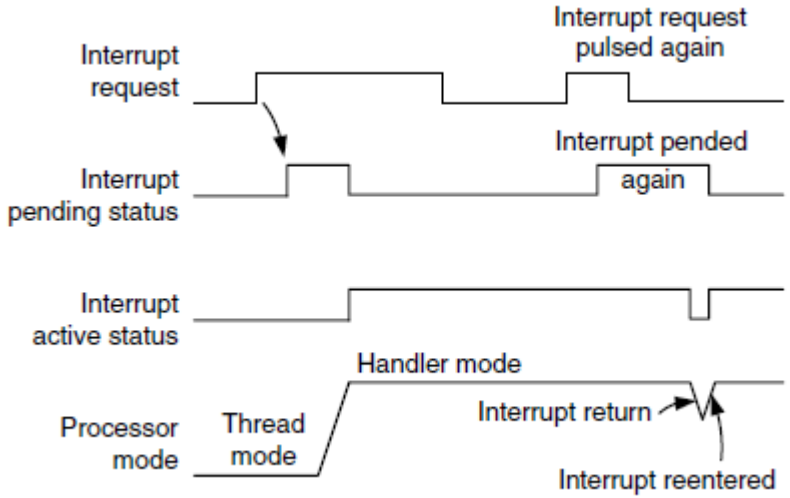


# 4.7. Ejemplos de Excepciones

- Ejemplos
  - Solicitud multiple **antes** del comienzo de la ISR
    - La ISR asociada se ejecuta una única vez




  - Solicitud multiple **despues** del comienzo de la ISR
    - La ISR asociada se vuelve a ejecutar



# 4.8 Latencia de interrupcion

- **Latencia de interrupcion (o latencia de excepcion)**
  - Tiempo que se produce desde el reconocimiento de que se produjo una excepcion hasta el comienzo de la ejecucion de su ISR asociada
  - Se busca que ese tiempo sea el menor posible, ya que se desea que su atencion sea inmediata.
  - Depende de:
    - Duracion de una instruccion (no se puede dejar una instruccion a medio ejecutar)
    - Tiempo que esta la aplicacion con las interrupciones deshabilitadas
    - Duracion de las excepciones de prioridad superior.
  - En Cortex-M3 , la latencia puede ser de sólo 12 ciclos de reloj

# 4.9 Niveles de privilegio/Modos de funcionamiento

- **Niveles de privilegios en un procesador**
  - Limitan las acciones y el acceso a recursos en la ejecucion de instrucciones
  - Se usan en la implementacion de applicaciones que ejecutan sobre un sistema operativo(OS)
  - Como mínimo dos niveles:
    - **Nivel usuario**: Nivel con privilegios limitados
      - No pueden acceder al hardware ni ejecutar cierto tipo de instrucciones
      - Si necesitan servicios, se los pide al sistema operativol ( mediante un TRAP denominado Llamada a Supervisor.
      - Es habitual que utilicen este nivel las aplicaciones de usuario.
    - **Nivel Privilegiado**: Nivel con todos los privilegios
      - Pueden realizar cualquier accion sobre cualquier recurso del sistema
      - Siempre que se ejecute una ISR se ejecutará en este nivel
      - Es habitual que utilicen este nivel los Sistemas Operativos y las ISRs

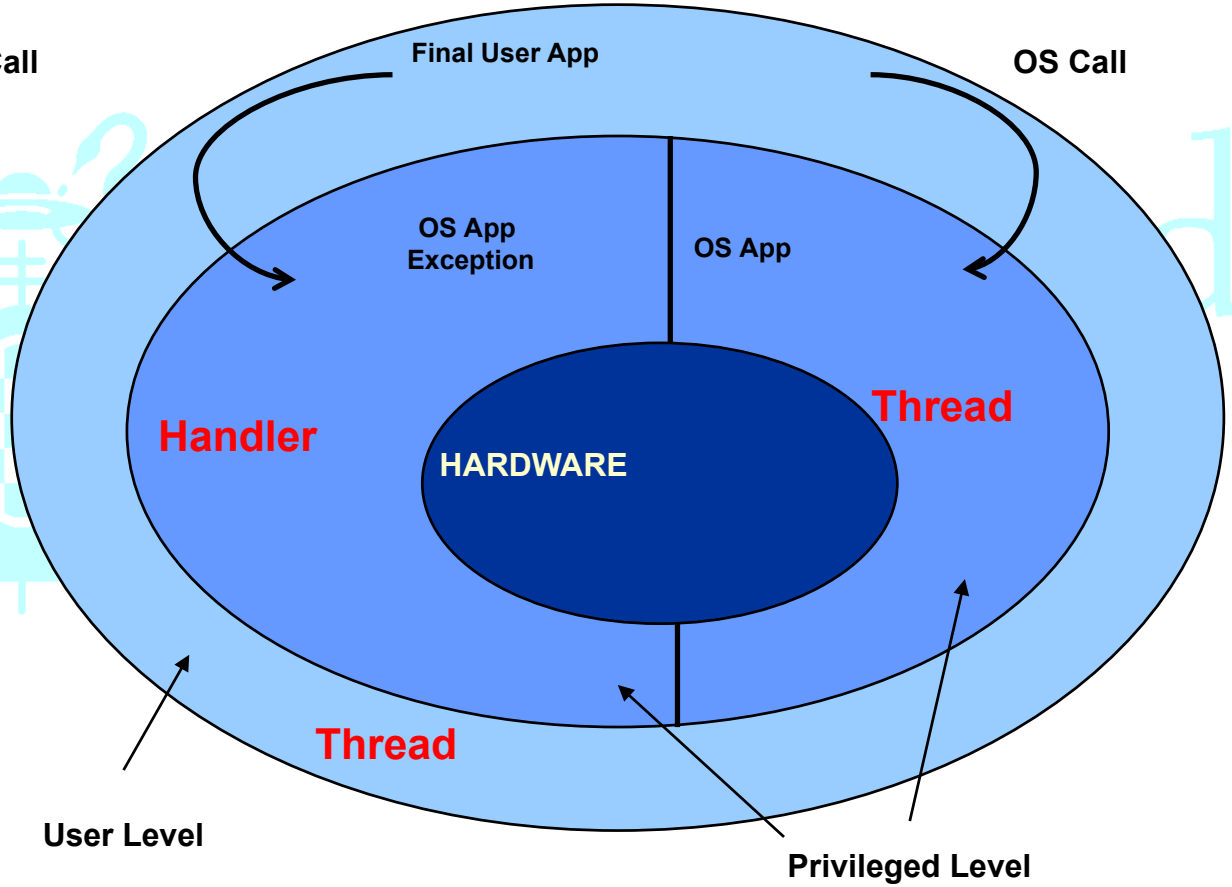
# 4.9 Niveles de privilegio/Modos de funcionamiento

## Modos de Funcionamiento del procesador

- Cortex-M3 define dos modos de funcionamiento a la hora de ejecutar instrucciones
  - **Modo Handler**:
    - Modo de ejecucion del procesador cuando ejecuta una ISR
    - Se entre en este modo como resultado del tratamiento de una excepcion.
    - Cuando el procesador se encuentra en este modo, se ejecuta en nivel privilegiado
  - **Modo thread**:
    - Se entra en este modo cuando se produce un reset o cuando se termina la ejecucion de una ISR (en otras palabras, el procesador se encontrará en modod thread siempre que no se este ejecutando instruccioens de una ISR).
    - En este modo se puede ejecutar codigo con privilegios y sin privilegios

# 4.9 Niveles de privilegio/Modos de funcionamiento

- Niveles de privilegios / Modos de Funcionamiento

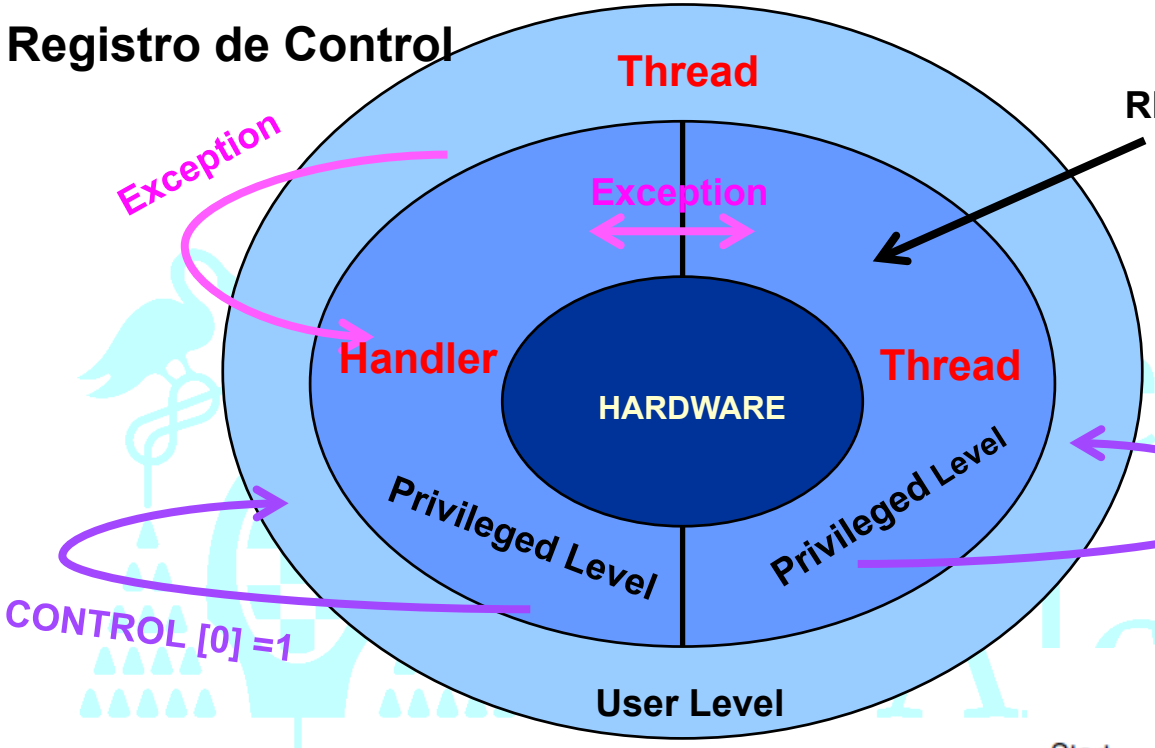


# 4.9 Niveles de privilegio/Modos de funcionamiento

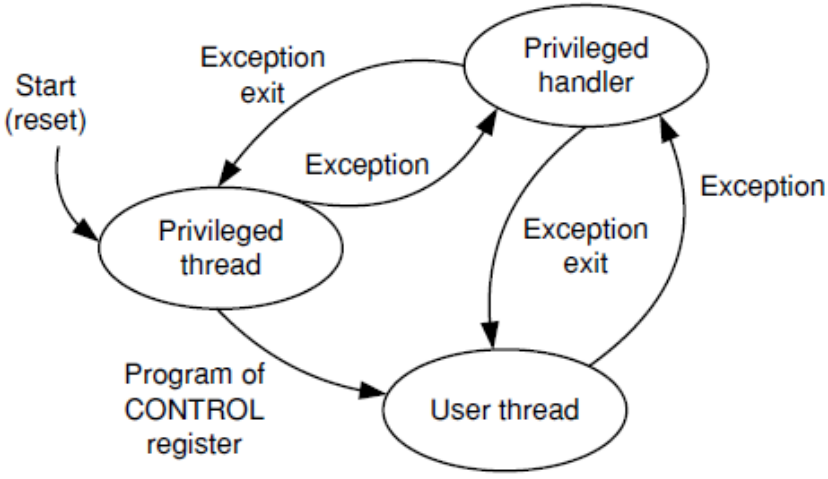
- **Niveles de privilegios / Modos de Funcionamiento**
- **El registro de Control CONTROL**
  - Registro interno del procesador de 32 bits donde solo 2 son validos
  - Utilizado para definir el nivel de privilegio y la pila que se usa en cada momento
    - **CONTROL[0]:** indica el nivel de privilegio en que cada momento ejecuta el procesador
      - Se puede escribir por software (solo en nivel **privilegiado)**
      - Una vez que se entra en modo usuario, la unica forma para volver a modo provilegiado es mediante una excepcion y camibiar el modo es la ISR asociada
    - **CONTROL[1]:** indica el registro SP que se esta usando
      - Siempre vale 0 en **modo Handler** (en modo **Thread**, puede ser 0 (privilegiado) o 1 (usuario)
      - Se puede escribir por software (a 1) solo en **Thread + Privileged**
      - En nivel usuario o en modo Handler, la escritura en este bit no está permitido

| Bit | Function |
|---|---|
| CONTROL[1] | Stack status:<br>1 = Alternate stack is used<br>0 = Default stack (MSP) is used<br>If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode. |
| CONTROL[0] | 0 = Privileged in Thread mode<br>1 = User state in Thread mode<br>If in handler mode (not Thread mode), the processor operates in privileged mode. |

# 4.9 Niveles de privilegio/Modos de funcionamiento

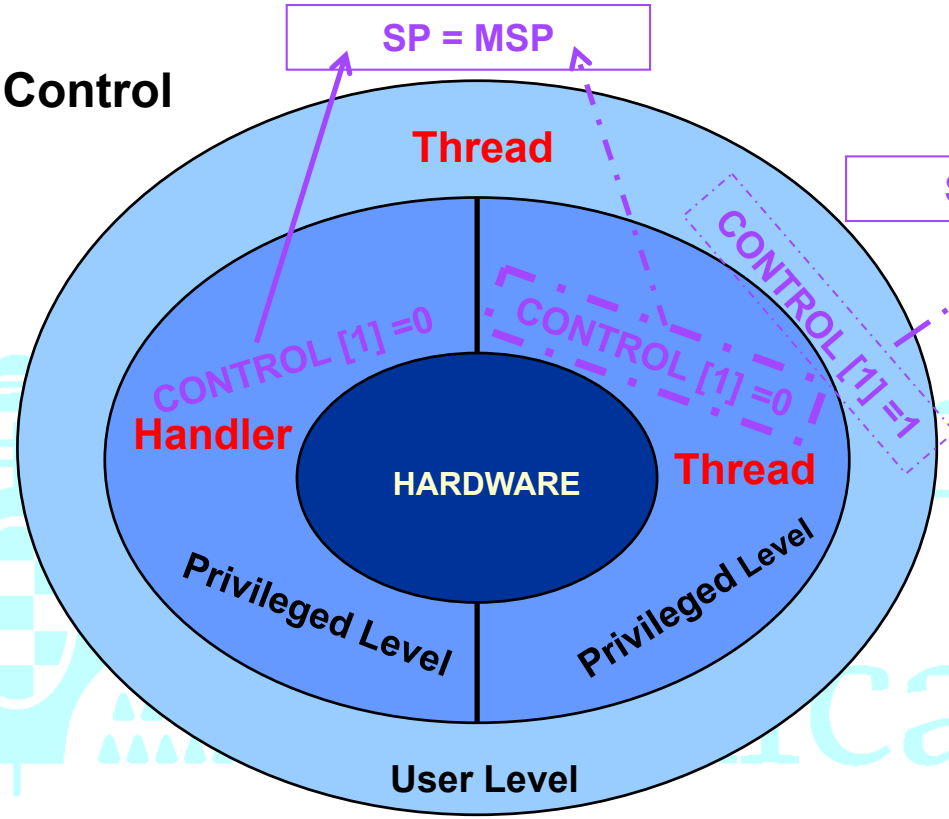



| Bits | Name | Function |
|---|---|---|
| [31:2] | - | Reserved |
| [1] | Active stack pointer | Defines the current stack:<br>0 = MSP is the current stack pointer<br>1 = PSP is the current stack pointer.<br>In Handler mode this bit reads as zero and ignores writes. |
| [0] | Thread mode privilege level | Defines the Thread mode privilege level:<br>0 = privileged<br>1 = unprivileged. |



# 4.9 Niveles de privilegio/Modos de funcionamiento

**Registro de Control**



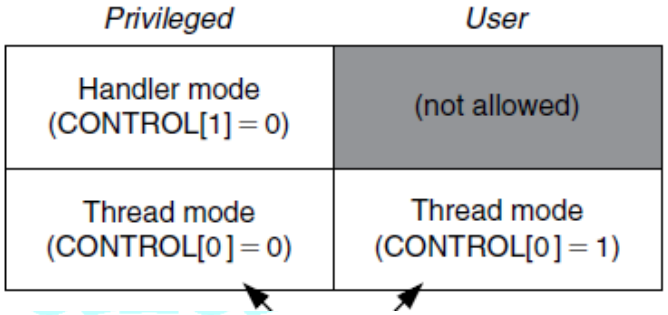

| Bits | Name | Function |
|---|---|---|
| [31:2] | - | Reserved |
| [1] | Active stack pointer | Defines the current stack:<br>0 = MSP is the current stack pointer<br>1 = PSP is the current stack pointer.<br>In Handler mode this bit reads as zero and ignores writes. |
| [0] | Thread mode privilege level | Defines the Thread mode privilege level:<br>0 = privileged<br>1 = unprivileged. |

→ **Recommended by ARM**

# 4.9 Niveles de privilegio/Modos de funcionamiento

- **Niveles de privilegios / Modos de Funcionamiento**

| How are apps performed? | *Privileged Level* | *User Level* |
|---|---|---|
| ***When running an exception*** | Handler Mode | |
| ***When running main program*** | Thread Mode | Thread Mode |

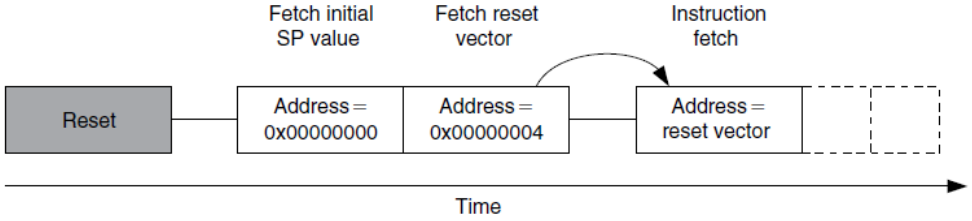


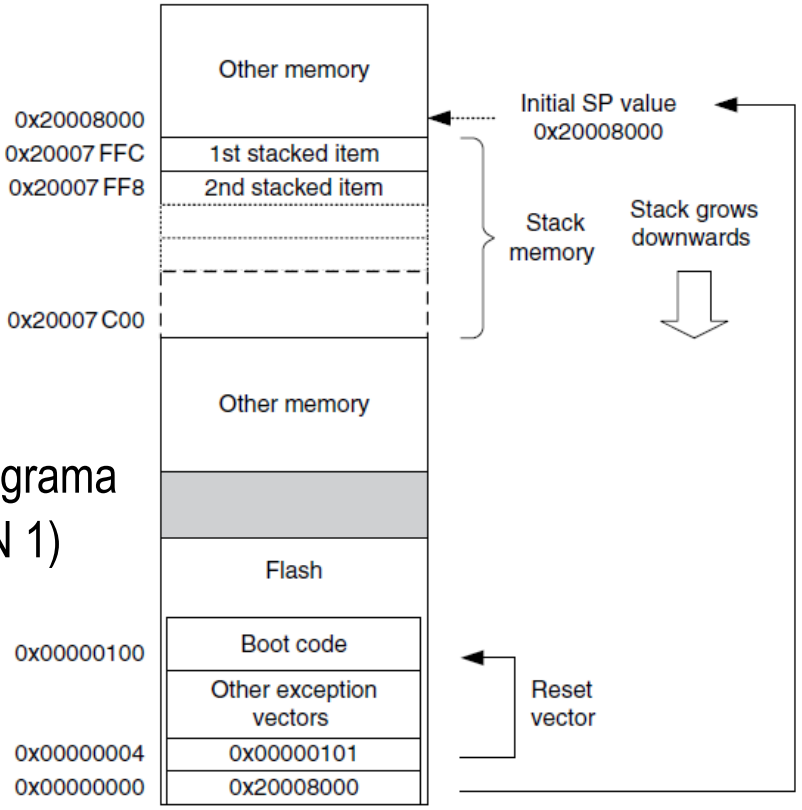
# 4.10. Excepciones de Sistema

- Existen varias excepciones de sistema, cuya funcionalidad se puede definir como:
  - Reset
  - Faults,
    - Memory Management Fault
    - Bus Fault
    - Usage Fault
    - HardFault
  - Llamada a supervisor (Supervisor Call,SVC) y Pendable Service Call (PendSC)
  - SYSTICK

# 4.10. Excepciones de Sistema

- **Reset:**
  - Es la excepcion mas urgente (con más prioridad)en cualquier µP
- **Secuencia de Reset:** el µP leerá 2 palabras de la Tabla de Vectores:
  - Direccion 0x00000000: Valor de comienzo de la pila R13 (the MSP)
  - Direccion 0x00000004: Vector de Reset (VN 1), valor inicial del registro R15 (the PC)

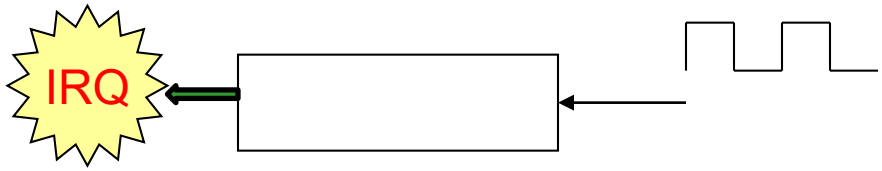



- Tras la captura del Vector de Reset:
  - Cortex-M3 comienza la ejecucion nomral del programa en la direccion que indica el Vector de Reset (VN 1)
  - Es necesario tener inicializado el SP desde ese momento, ya que alguna excepcion (como NMI) pueden generarse justo despues del reset

# 4.11 SYSTICK

- **SYSTICK** es un temporizador hardware integrado en el NVIC
  - Todos los chips Cortex-M3 tiene en mismi timer, incrementando su portabilidad software
  - Es un timer descendente de 24-bits, sincronizado con:
    - La señal de reloj interna…
    - …o una referencia de reloj externa
- SYSTICK puede can generar una exception por tiempo (VN 15), muy util en RTOS:
  - En la mayoria de OS, se usa un temporizador hardware para generar excepciones que realize la gestion de tareas
  - Este timer debe ser protegido de aplicaciones de usuario para que estas no cambien su comportamiento



# 4.11 SYSTICK

- SYSTICK se controla por medio de 4 registros
  - SysTick->LOAD

**Table 8.10** SYSTICK Reload Value Register (0xE000E014)

| Bits | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 23:0 | RELOAD | R/W | 0 | Reload value when timer reaches 0 |

  - SysTick->VAL

**Table 8.11** SYSTICK Current Value Register (0xE000E018)

| Bits | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 23:0 | CURRENT | R/Wc | 0 | Read to return current value of the timer. Write to clear counter to 0. Clearing of current value also clears COUNTFLAG in SYSTICK Control and Status register |

  - SysTick->CALIB

**Table 8.12** SYSTICK Calibration Value Register (0xE000E01C)

| Bits | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 31 | NOREF | R | — | 1 = No external reference clock (STCLK not available)<br>0 = External reference clock available |
| 30 | SKEW | R | — | 1 = Calibration value is not exactly 10 ms<br>0 = Calibration value is accurate |
| 23:0 | TENMS | R/W | 0 | Calibration value for 10 ms; chip designer should provide this value through Cortex-M3 input signals. If this value is read as 0, calibration value is not available |

# 4.11 SYSTICK

- SYSTICK se controla por medio de 4 registros
  - SysTick->CTRL

**Table 8.9** SYSTICK Control and Status Register (0xE000E010)

| Bits | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 16 | COUNTFLAG | R | 0 | Read as 1 if counter reaches 0 since last time this register is read; clear to 0 automatically when read or when current counter value is cleared |
| 2 | CLKSOURCE | R/W | 0 | 0 = External reference clock (STCLK)<br>1 = Use processor free running clock |
| 1 | TICKINT | R/W | 0 | 1 = Enable SYSTICK interrupt generation when SYSTICK Timer reaches 0<br>0 = Do not generate interrupt |
| 0 | ENABLE | R/W | 0 | SYSTICK Timer enable |

- Funciones relacionadas con Systick
  - SysTick_Config(uint32_t ticks)

# 4.12. CMSIS

- CMSIS (Cortex Microcontroller Software Interface Standard,CMSIS) es un conjunto de funciones que entrega CMSIS para que se usen en sus aplicaciones.
- Son compatibles entre varios Procesadores Cortex→ Aseguran portabilidad
  - Puesta a punto de la Tabla de Vectores

```
void SetVector(unsigned int ExcpType, unsigned int VectorAddress)
{ // Calculate vector location = VTOR + (Exception_Type * 4)

*((volatile unsigned int *) (SCB->VTOR + (ExcpType << 2))) =
VectorAddress | 0x1;

// LSB of vector set to 1 to indicate Thumb

return;
}
```

# 4.12. CMSIS

- Establecimiento de Prioridad de Excepcion
  - Establece la prioridad de la (IRQ) #4 a 0xC0:

```
NVIC_SetPriority(IRQ4_IRQn,0xC);
// This function automatically shifts the priority value to
// implemented bits in the interrupt priority registers
```

  - Otro método:

```
NVIC_SetPriority(IRQ4_IRQn, NVIC_EncodePriority(PriorityGroup,
PreemptPriority,SubPriority));
```

- Habilitar/Deshabilitar la excepcion

```
NVIC_EnableIRQ(UART1_IRQn); // Enable UART#1 interrupt
// UART1_IRQn is MCU specific and is defined
// in the device driver library
NVIC_DisableIRQ(UART1_IRQn); // Disable UART#1 interrupt
```

# 4.12. CMSIS

- Codificacion de una ISR en lenguaje C usando CMSIS (Keil version) (__irq es opcional):

```
__irq void UART1_Handler(void) {
... // process IRQ request for the peripheral
... // Deassert IRQ request in peripheral
return;
}
```

- Se puede generar excepciones por software de 2 formas:
  - Usando registros ISPR con la funcion CMSIS, escribiendo el siguiente codigo en C:

```
NVIC_SetPendingIRQ(3);
```

  - Usando el registro STIR:

```
NVIC->STIR = 3; /* NVIC->STIR is defined in CMSIS */
```