微机原理与接口

第七章

中断系统与Cortex-M 微处理器的异常

中断系统与Cortex-M 微处理器的异常

* 第一节 中断技术概述
* 第二节 Cortex-M 微处理器的异常
* 第三节 中断控制器 NVIC 的寄存器
* 第四节 用于系统异常的 SCB寄存器
* 第五节 外部中断/事件控制器 EXTI

第一节 中断技术概述

* 为了解决“快速CPU”与“慢速外设”之间速度不匹配问题，一方面要提高外设的工作速度，另一方面，发展了中断的概念。
* 最初的中断技术是作为处理器与外部设备交换信息的一种控制方式而提出的。所以中断最初的概念全部是对外部设备而言的，称为外部中断或硬件中断。
* 随着计算机技术的发展，中断的含义已不再局限于处理器与外设交换信息的一种方式，其应用范围也变得非常之广。

第一节 中断技术概述

* **中断**：是指计算机的CPU暂时中止它正在执行的主程序，转去执行请求中断的那个外设或事件的中断服务（处理）程序，待处理完后，又返回到被中止了的程序。
* **中断系统**：计算机所具有的上述功能，称为中断功能。为了实现中断功能而设置的各种硬件和软件统称为中断系统。
* **中断源**：引起中断的原因或发出中断申请的来源，称为中断源。一般，计算机系统都是多中断源系统。

第一节 中断技术概述

* 中断的作用：
  * 实现CPU和I/O的并行工作
  * 处理故障
  * 实现多道程序和分时操作
  * 实时控制
  * 实现人机联系
  * 实现多处理器联系

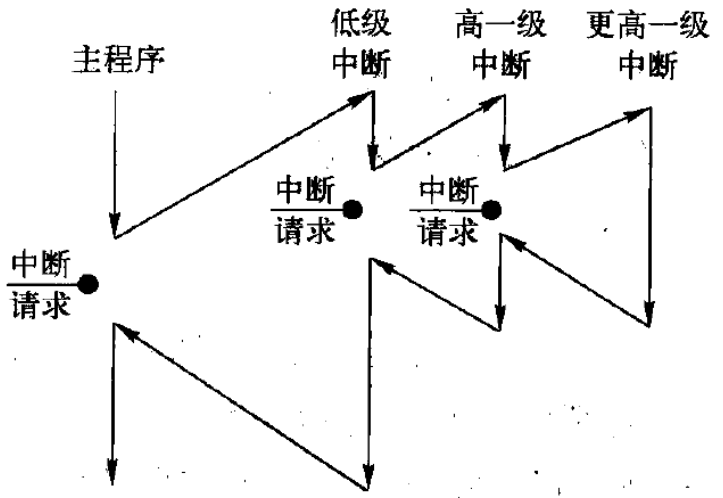
第一节 中断技术概述

中断优先级与中断嵌套

* 一个处理器总会有若干中断源，但在同一瞬间，处理器只能响应若干个中断源中的一个中断请求。
* 处理器为了避免在同一瞬间因响应若干个中断源的中断请求而带来的混乱，就必须给每个中断源的中断请求分配一个优先级，称为**中断优先级**。
* 优先级管理，分为软件方式和硬件方式两种。

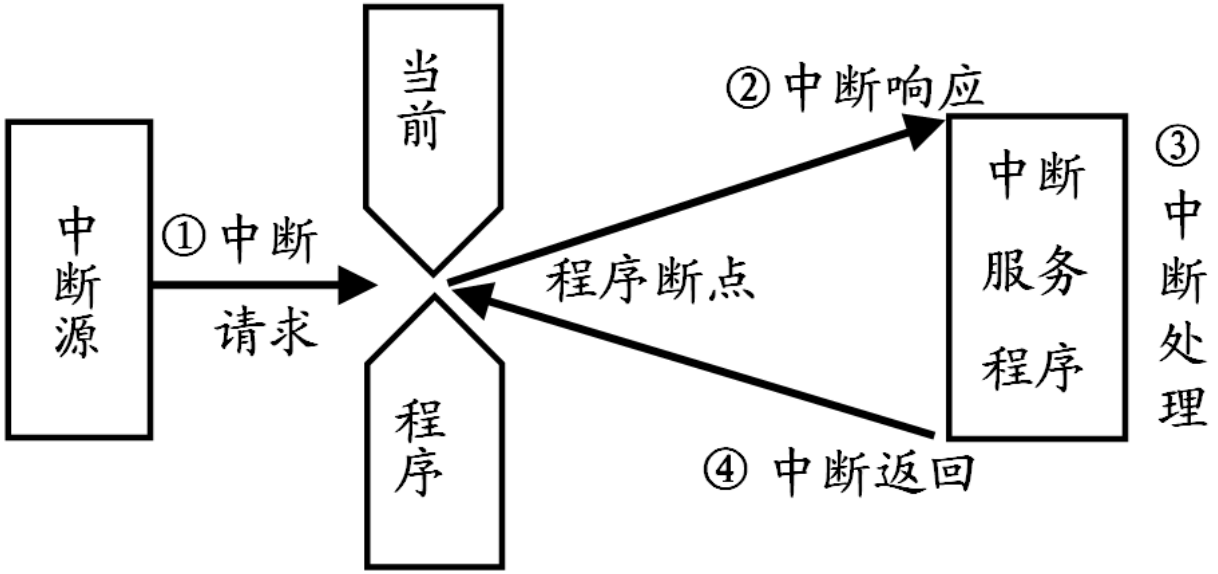
第一节 中断技术概述

* 优先级有两个用途：
  * 同时有多个中断请求，先响应优先级高的
  * 正在处理一个中断，有更高优先级的中断请求到达
* 处理器正在为较低优先级的中断源服务时，若出现中断优先级更高的中断请求，处理器暂停目前正在进行的中断服务，响应新的中断源的中断请求，服务完毕，再返回继续完成被中断的较低优先级的中断服务，直至最后返回主程序。这就是所谓的中断嵌套。



第一节 中断技术概述

中断的处理过程



* 第一步：中断请求

请求触发器

第一节 中断技术概述

中断的处理过程

* 第二步：中断响应
  * 响应的条件：
    * 有请求并保持
    * 没有被禁止
    * 是当前优先级最高的请求
  * 响应后的处理：
    * 关闭中断
    * 保存PC
    * 找到中断服务程序的入口地址

屏蔽寄存器

优先级判别

断点寄存器

第一节 中断技术概述

中断的处理过程

* 第三步：中断处理
  * 保护现场
  * 开中断
  * 中断服务
  * 关中断
  * 恢复现场

* 第四步：中断返回

中断系统与Cortex-M 微处理器的异常

* 第一节 中断技术概述
* 第二节 Cortex-M 微处理器的异常
* 第三节 中断控制器 NVIC 的寄存器
* 第四节 用于系统异常的 SCB寄存器
* 第五节 外部中断/事件控制器 EXTI

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的异常源

嵌套向量中断控制器NVIC接收多个中断源产生的中断请求，这些中断源包括外部中断请求IRQ、不可屏蔽中断NMI、SysTick系统节拍中断以及内核系统异常。多数IRQ由定时器、I/O 端口和通信接口（例如UART、SPI和I2C)等外设产生，NMI由看门狗定时器或者掉电检测产生，SysTick来自处理器内核。

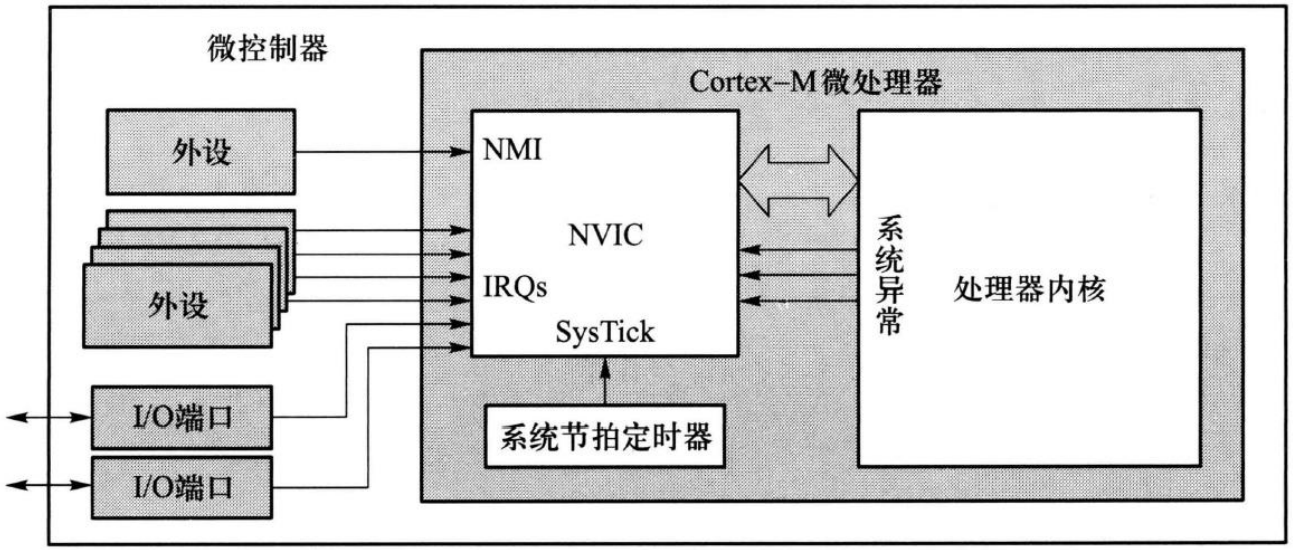


图 7-2-1　Cortex-M 微处理器的各种异常源

第二节 Cortex-M

ARM Cortex-M4 微处理器共支持240个外部中断，外部中断输入＃0～＃239对应NVIC的中断输入

Cortex-M 微处理器的异常源

异常编号	异常类型	优先级	CMSIS编号	向量地址偏移	描　述
1	复位（RESET）	-3（最高）	NA	0x04	复位异常
2	非屏蔽中断（NMI）	-2	-14	0x08	不可屏蔽中断（外部 NMI 输入）
3	硬件错误（HardFault）	-1	-13	0x0C	所有错误都可触发，但前提是相应的错误处理未使能
4	存储器管理错误（MemManage）	可编程	-12	0x10	存储器管理单元（MPU）错误或者访问非法位置时产生，可编程管理。若被屏蔽，会升级为 HardFault
5	总线错误（BusFault）	可编程	-11	0x14	当 AHB 总线接口收到总线错误时响应，该错误一般由除 MemManage 异常之外的存储器保护错误产生
6	使用错误（UsageFault）	可编程	-10	0x18	非内存错误导致的程序执行失败或者试图访问协处理器导致的错误
7~10	保留	NA	NA	0x1C~0x2B	—
11	请求管理调用（SVCall）	可编程	-5	0x2C	处理 SVC 指令引起的异常，一般用于 OS 环境且允许应用任务访问系统服务
12	调试监控（DebugMonitor）	可编程	-4	0x30	在基于软件方式的调试时，处理断点和监视点等异步调试事件造成的异常
13	保留	NA	NA	0x34	—
14	可挂起的服务调用（PendSV）	可编程	-2	0x38	OS 一般用该异常进行上、下文切换
15	系统节拍定时器（SysTick）	可编程	-1	0x3C	用于 OS 或简单的定时器外设
16	外部中断 #0	可编程	0	0x40	可由片上外设或者外设中断源产生
17	外部中断 #1	可编程	1	0x44	
……	……	……	……	……	
255	外部中断 #239	可编程	2~239	0x48~0x3FF	

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断管理

Cortex-M微处理器内核中有中断屏蔽寄存器PRIMASK、FAULTMASK和BASEPRI，用于异常或者中断的屏蔽管理。

PRIMASK 寄存器只能在特权等级下访问，可以禁止除了NMI和HardFault 外的所有异常，常应用于需要暂时禁止所有中断以执行一些实时和关键的任务。

（1）CMSIS-Core 提供的 C 语言函数：

```
void     _enable_irq( void );                    // 清除 PRIMASK 使能中断
void     _disable_irq( void );                   // 设置 PRIMASK 禁止所有中断
void     _set_PRIMASK( uint32_t primask );       // 设置 PRIMASK 为特定值
uint32_t_get_PRIMASK( void );                    // 读取 PRIMASK 的数值
```

（2）汇编语言提供的方法：

```
MOVS     R0 ,   #0
MSR      PRIMASK ,   R0          ;将 0 写入 PRIMASK 使能所有中断
MOVS     R0 ,   #1
MSR      PRIMASK ,   R0          ;将 1 写入 PRIMASK 禁止所有中断
```

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断管理

FAULTMASK 寄存器只能在特权等级下访问，FAULTMASK置位后，只允许 NMI，其他所有中断 / 异常都被屏蔽。

（1）CMSIS-Core 提供的 C 语言函数：

```
void     _enable_fault_irq( void );                  // 清除 FAULTMASK 使能中断
void     _disable_fault_irq( void );                 // 设置 FAULTMASK 禁止所有中断
void     _set_FAULTMASK( uint32_t primask );         // 设置 FAULTMASK 为特定值
uint32_t_get_FAULTMASK( void );                      // 读取 FAULTMASK 的数值
```

（2）汇编语言提供的方法：

```
MOVS        R0 ,  #0
MSR         FAULTMASK , R0        ;将 0 写入 FAULTMASK 使能所有中断
MOVS        R0 ,  #1
MSR         FAULTMASK , R0        ;将 1 写入 FAULTMASK 禁止所有中断
```

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断管理

如果只想禁止某特定优先等级及其以下等级的中断，就可以使用BASEPRI寄存器。用户只需要将所需屏蔽的优先级写入BASEPRI 寄存器，例如将0x40 写入BASEPRI 寄存器，则会屏蔽优先级小于等于0x40（即在0x40-0xFF之间）的中断。

（1）CMSIS-Core 提供的 C 语言函数：

```
_set_BASEPRI( 0x40 );          // 设置 BASEPRI,禁止优先级在 0x40~0xFF 间的中断
uint32_t      x = _get_BASEPRI( void );   // 读取 BASEPRI 的数值
```

（2）汇编语言提供的方法：

```
MOVS      R0 ,   #0x40
MSR       BASEPRI ,     R0          ; 0x40 写入 BASEPRI,禁止优先级在 0x40~0xFF 间的中断
MRS       R0 ,   BASEPRI            ; 读取 BASEPRI 的数值到 R0
```

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断管理

CMSIS-Core 还提供了多个访问函数用于中断的使能、优先级设置以及优先级分组管理：

```
void    NVIC_EnableIRQ ( IRQn_Type IRQn );                          // 使能外部中断
void    NVIC_DisableIRQ ( IRQn_Type IRQn );                         // 禁止外部中断
void    NVIC_SetPriority ( IRQn_Type IRQn , uint32_t priority );    // 设置中断的优先级
void    NVIC_SetPriorityGrouping ( uint32_t PriorityGroup );        // 优先级分组配置
```

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断优先级

Cortex-M4 微处理器的8位优先级配置寄存器可以得到最大128个分组（抢占）等级，是因为这8位寄存器进一步划分为两个部分：分组优先级（也称抢占优先级，pre-emption priority) 和子优先级(subpriority)。分组优先级决定了是否能嵌套，高的组优先级（数值低）中断可以抢占低的组优先级（数值高）中断，如果组优先级是一样的，即使子优先级比正在执行的中断的子优先级高也是不能抢占的。在组优先级一致的情况下，多个中断请求同时发生，这样的情况下子优先级高的可以先执行，而子优先级低的则只能暂时挂起(pending)。

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断优先级

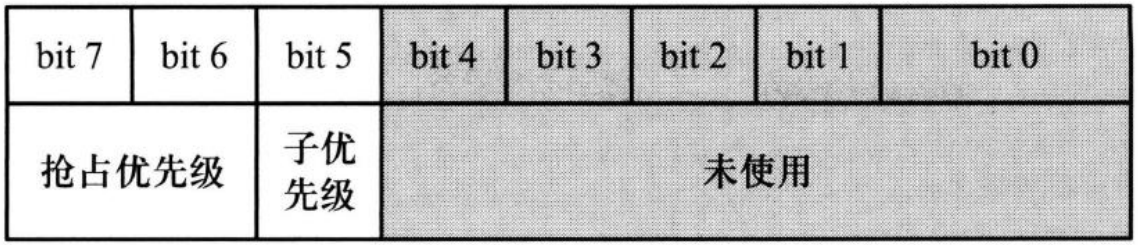
在Cortex-M4 微处理器的系统控制块SCB中有一个优先级分组的配置寄存器SCB_AIRCR（应用中断和复位控制寄存器），其中PRIGROUP域的值决定了8 种不同的优先级分组方式。

优先级分组	抢占优先级域	子优先级域
0（默认）	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	无	bit[7:0]

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断优先级

在确定实际的分组优先级和子优先级时，还需要考虑实际可用的配置寄存器的宽度，例如配置寄存器的宽度为3 （第7~5 位可用），并且AIRCR 设置的优先级分组为5，则会有4 个分组优先级（第7~6 位），每个分组优先级下具有2个子优先级。



第二节 Cortex-M 微处理器的异常

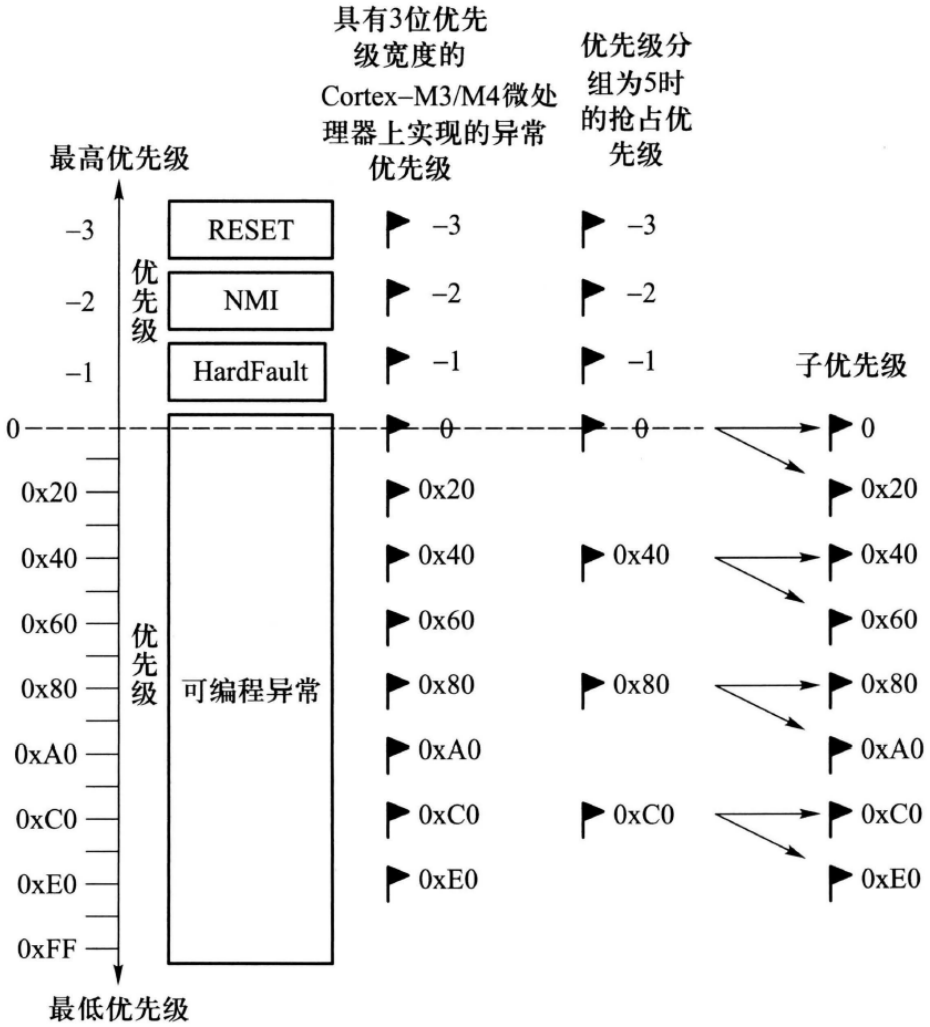


图 7-2-4　3 位优先级寄存器中优先级分组为 5 的可用优先级

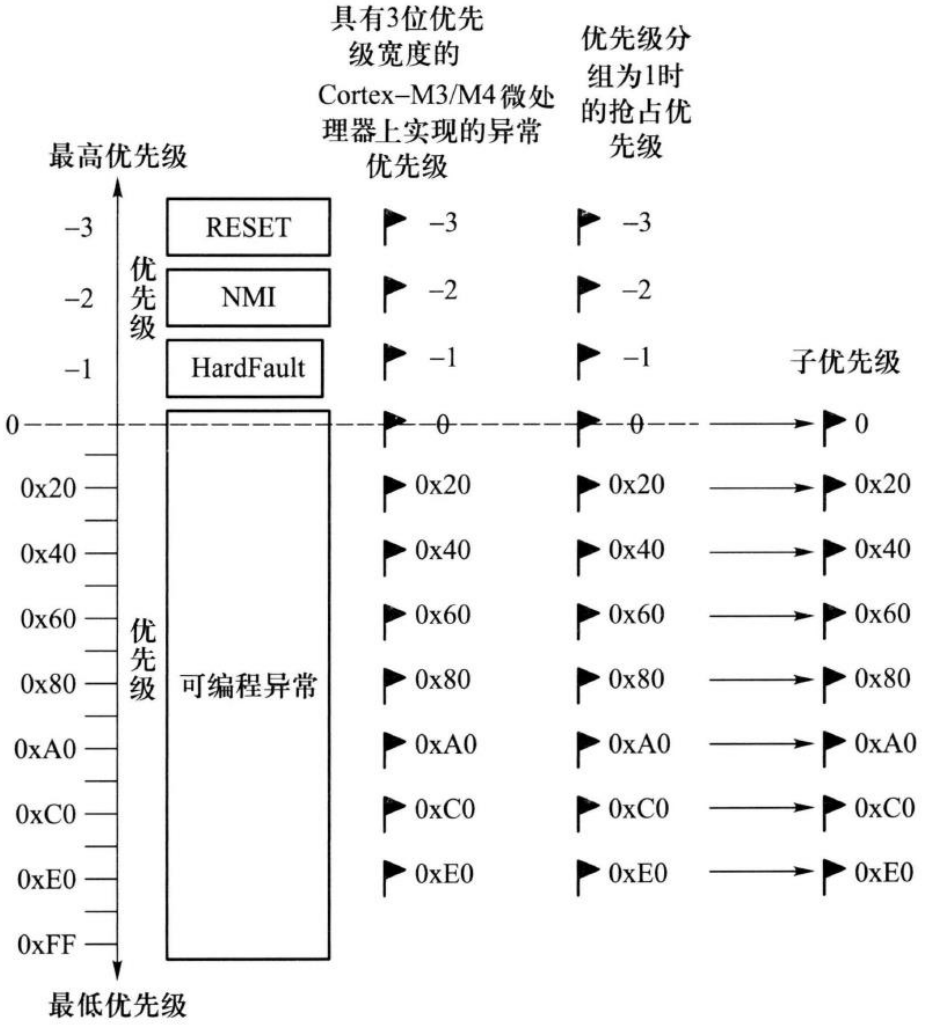


图 7-2-5　3 位优先级寄存器中优先级分组为 1 的可用等级

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断过程

1、初始化过程

Cortex-M微处理器在CPU 复位后，所有中断处于禁止状态，且默认的优先级为0 。在使用任何一种中断之前，需要进行如下操作：

(1) 设置所需中断的优先级（该步骤是可选的）；

(2) 使能外设中的可以触发中断的中断产生控制；

(3) 使能NVIC 中的中断。

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断过程

2、中断申请过程

若满足下列的中断申请条件， Cortex-M 微处理器会接受中断申请：

(1) 处理器正在运行，未被暂停或处于复位状态；

(2) 中断处于使能状态， NMI 和HardFault 总是处于使能；

(3) 中断的优先级高于当前的等级；

(4) 中断未被屏蔽寄存器PRIMASK 等屏蔽。

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断过程

2、中断申请过程

中断挂起是一种等待处理器处理的状态，当处理器正在执行另外一个高优先级或者同等优先级的中断，或者中断被中断屏蔽寄存器给屏蔽了，即使中断源外设取消了请求信号，已产生的中断挂起状态会一直保持到其他中断处理结束或者中断屏蔽被清除。

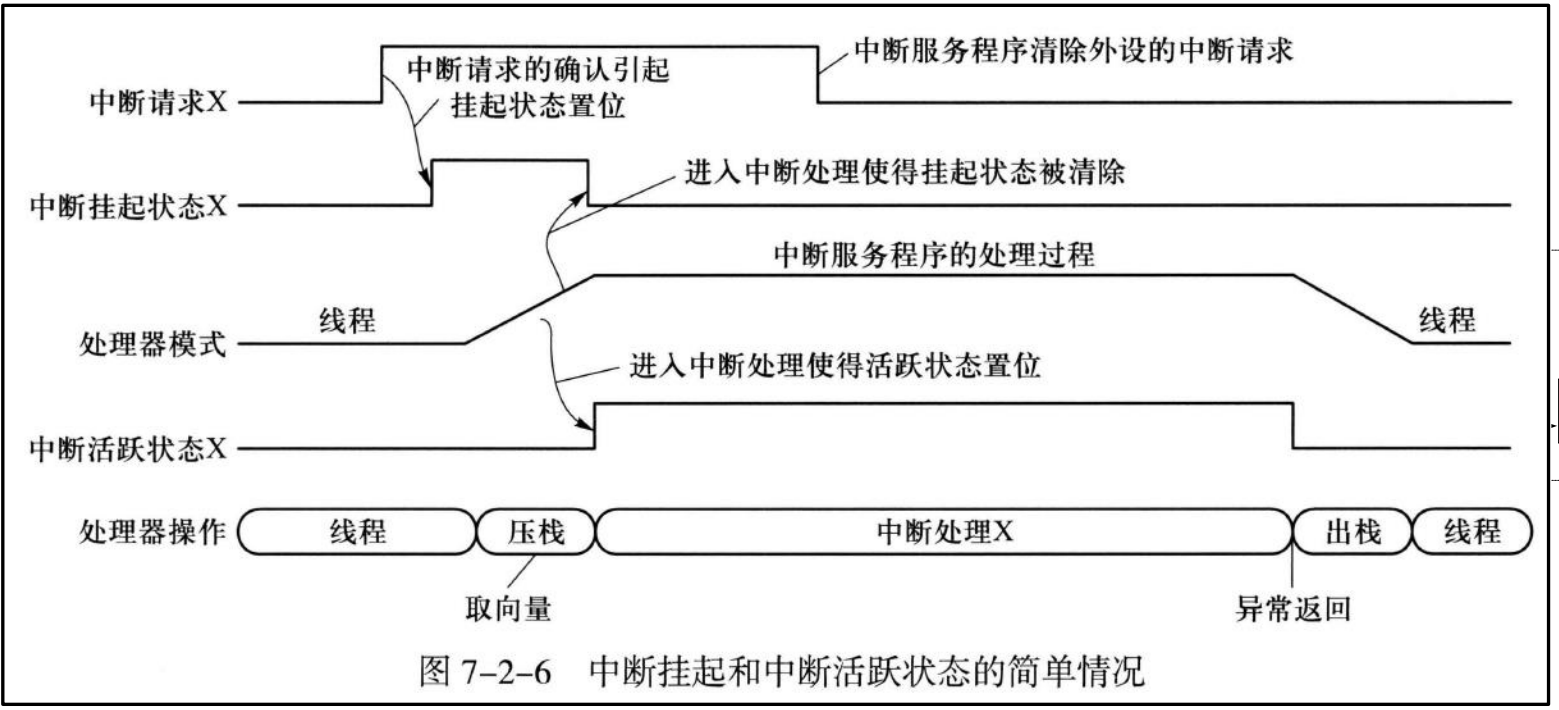
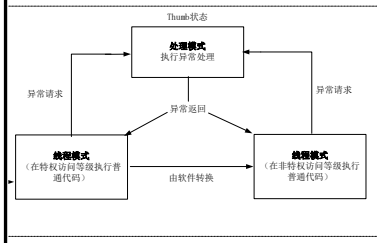


图 7–2–6　中断挂起和中断活跃状态的简单情况



第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断过程

3、中断响应过程

Cortex-M 微处理器在中断响应过程中，系统自动完成以下的操作：

(1) 多个寄存器和返回地址被压入当前正在使用的栈。

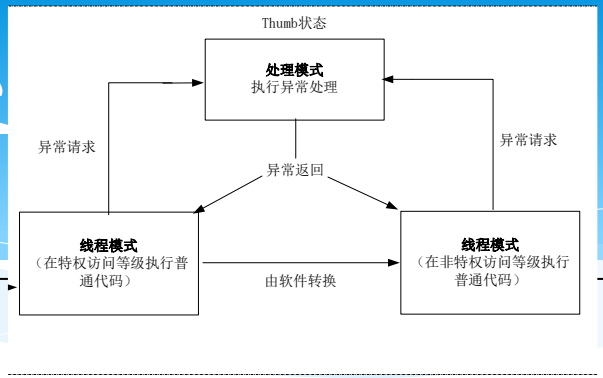
若正在使用PSP进程栈指针，则压栈发生在PSP指向的栈区域，如果是MSP栈指针，则压栈在MSP 指向的栈区域。

(2) 取出中断向量，转入中断服务处理程序执行指令。

(3) 同时更新多个NVIC 寄存器和内核寄存器的值。

异常处理开始前，会根据压栈时使用的栈，栈指针PSP 或者MSP 自动调整为压栈完成后的值。程序计数器PC更新为异常处理的起始地址，链接寄存器LR更新为EXC_RETURN的值，这个EXC_RETURN用来保存异常处理流程的状态信息，包括压栈使用的是哪个栈指针。

第二节 Cortex-M 微处



Cortex-M 微处理器的中断过程

4、中断服务

（1）Cortex-M 微处理器在执行中断处理服务程序时，微处理器处于处理模式，处于特权访问等级，使用主栈指针MSP 。

（2）若更高优先级的中断在这个阶段产生， Cortex-M微处理器会接受新的中断请求，而正在执行的处理会被挂起(pending) 而被更高优先级的中断抢占。

（3）中断处理服务的结尾，程序代码执行的返回会引EXC_RETURN 数值被加载到程序计数器PC 中，触发中断返回过程。

（4）每个外部中断都有一个活跃状态位，当中断处理开始执行时该位会被置1 ，中断返回后会被清零。活跃状态位保存在中断状态寄存器NVIC_IABR 中。例如NVIC_IABR[0] 用于保存外部中断＃0~#31 的活跃状态。由于存在中断嵌套，所以可能有多个被置位的活跃状态位。

第二节 Cortex-M 微处理器的异常

Cortex-M 微处理器的中断过程

5、中断返回

Cortex-M 微处理器的异常返回是由一个特殊的值EXC_RETURN 来触发。当异常申请被接受后，EXC_RETURN值被产生且写入到链接寄存器LR中，该数值由异常返回指令写入PC 时，就会触发异常返回过程。

异常返回可由以下指令产生：

(1) **BX LR**

若EXC_RETURN 数值仍在LR 中，则在异常处理结束时可以使用BX LR 指令执行中断返回。

(2) **POP {PC} 或POP{ …… ,PC}**

在进入异常处理后， LR 的值通常会被压入栈中，可以使用以上出栈到PC 寄存器的POP 指令，将LR 中存储的EXC_RETURN 出栈到程序计数器PC 中，实现异常返回。

(3) **加载(LDR) 或多寄存器加载(LDM)**

加载指令以PC 为目的寄存器可以产生中断返回。

中断系统与Cortex-M 微处理器的异常

* 第一节 中断技术概述
* 第二节 Cortex-M 微处理器的异常
* 第三节 中断控制器 NVIC 的寄存器
* 第四节 用于系统异常的 SCB寄存器
* 第五节 外部中断/事件控制器 EXTI

第三节 中断控制器 NVIC 的寄存器

中断控制的NVIC 寄存器简介

NVIC 中有多个对应的中断控制寄存器，主要是设置使能(set enable) 寄存器、清除使能(clear enable) 寄存器、设置挂起(set pending) 寄存器、清除挂起(clear pending) 寄存器和活跃位(active bit) 寄存器。

以上寄存器各有连续的8个32 位寄存器0 、寄存器l 、寄存器2 、… 、寄存器7，分别控制第0~31 、第32~63 、第64~95 、…… 、第224~255 号中断IRQ。

地址范围	寄存器	CMSIS-Core 符号	功能
0xE000E100 ~ 0xE000E11C	中断设置使能寄存器	NVIC_ISER[0]~ NVIC_ISER[7]	写 1 设置使能
0xE000E180 ~ 0xE000E19C	中断清除使能寄存器	NVIC_ICER[0]~ NVIC_ICER[7]	写 1 清除使能
0xE000E200 ~ 0xE000E21C	中断设置挂起寄存器	NVIC_ISPR[0]~ NVIC_ISPR[7]	写 1 设置挂起状态
0xE000E280 ~ 0xE000E29C	中断清除挂起寄存器	NVIC_ICPR[0]~ NVIC_ICPR[7]	写 1 清除挂起状态
0xE000E300 ~ 0xE000E31C	中断活跃位寄存器	NVIC_IABR[0]~ NVIC_IABR[7]	活跃状态位，只读
0xE000E400 ~ 0xE000E4EF	中断优先级寄存器	NVIC_IP[0]~ NVIC_IP[239]	每个中断的中断优先级（8 位宽）
0xE000EF00	软件触发中断寄存器	NVIC_STIR	写中断编号设置相应中断的挂起状态

Accessing the Cortex®-M7 NVIC registers using CMSIS

CMSIS functions enable the software portability between different Cortex®-M profile processors. To access the NVIC registers when using CMSIS, use the following functions:

Table 41. CMSIS access NVIC functions

CMSIS function	Description
`void NVIC_EnableIRQ(IRQn_Type IRQn)`(1)	Enables an interrupt or exception.
`void NVIC_DisableIRQ(IRQn_Type IRQn)`(1)	Disables an interrupt or exception.
`void NVIC_SetPendingIRQ(IRQn_Type IRQn)`(1)	Sets the pending status of interrupt or exception to 1.
`void NVIC_ClearPendingIRQ(IRQn_Type IRQn)`(1)	Clears the pending status of interrupt or exception to 0.
`uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)`(1)	Reads the pending status of interrupt or exception. This function returns non-zero value if the pending status is set to 1.
`void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)`(1)	Sets the priority of an interrupt or exception with configurable priority level to 1.
`uint32_t NVIC_GetPriority(IRQn_Type IRQn)`(1)	Reads the priority of an interrupt or exception with configurable priority level. This function return the current priority level.

第三节 中断控制器 NVIC 的寄存器

中断使能寄存器

中断使能寄存器ISER/ICER 都是32 位宽，每位代表一个中断输入的使能控制。ISER/ICER 的数量取决于微处理器可以处理的外部中断的数量， Cortex-M4 微处理器可能存在32个以上的外部中断，因此ISER/ICER 会有多个，可能的寄存器序号依次为0~7 。

表 7-3-2 中断使能设置（NVIC_ISERx）和清除寄存器（NVIC_ICERx）

地址	名称	类型	复位值	说 明
0xE000E100	NVIC_ISER[0]	R/W	0	设置外部中断 #0 ~ #31 的使能 bit[0]用于中断 #0（异常 #16），bit[1]用于中断 #1（异常 #17），……，bit[31]用于中断 #31（异常 #47） 写 1 将位置 1，写 0 无作用，读出值表示当前使能状态
0xE000E104	NVIC_ISER[1]	R/W	0	设置外部中断 #32 ~ #63 的使能 写 1 将位置 1，写 0 无作用，读出值表示当前使能状态
0xE000E108	NVIC_ISER[2]	R/W	0	设置外部中断 #64 ~ #95 的使能 写 1 将位置 1，写 0 无作用，读出值表示当前使能状态
……	……	……	……	……
0xE000E180	NVIC_ICER[0]	R/W	0	清零外部中断 #0 ~ #31 的使能 bit[0]用于中断 #0（异常 #16），bit[1]用于中断 #1（异常 #17），……，bit[31]用于中断 #31（异常 #47） 写 1 将位置 0，写 0 无作用，读出值表示当前使能状态
0xE000E184	NVIC_ICER[1]	R/W	0	清零外部中断 #32 ~ #63 的使能 写 1 将位置 0，写 0 无作用，读出值表示当前使能状态
0xE000E188	NVIC_ICER[2]	R/W	0	清零外部中断 #64 ~ #95 的使能 写 1 将位置 0，写 0 无作用，读出值表示当前使能状态
……	……	……	……	……

第三节 中断控制器 NVIC 的寄存器

中断挂起寄存器

若中断产生但没有被立即执行， 则会被挂起。中断挂起状态可以通过中断设置挂起寄存器(NVIC_ISPRx) 和中断清除挂起寄存器(NVIC_ICPRx) 来访问。寄存器中的每一位代表一个中断输入的挂起状态。

表 7-3-3 中断挂起设置(NVIC_ISPRx)和清除寄存器(NVIC_ICPRx)

地址	名称	类型	复位值	说 明
0xE000E200	NVIC_ISPR[0]	R/W	0	设置外部中断 #0 ~ #31 的挂起 bit[0]用于中断 #0(异常 #16), bit[1]用于中断 #1(异常 #17), ……, bit[31]用于中断 #31(异常 #47) 写 1 将位置 1,写 0 无作用,读出值表示当前状态
0xE000E204	NVIC_ISPR[1]	R/W	0	设置外部中断 #32 ~ #63 的挂起 写 1 将位置 1,写 0 无作用,读出值表示当前状态
0xE000E208	NVIC_ISPR[2]	R/W	0	设置外部中断 #64 ~ #95 的挂起 写 1 将位置 1,写 0 无作用,读出值表示当前状态
……	……	……	……	……
0xE000E280	NVIC_ICPR[0]	R/W	0	清零外部中断 #0 ~ #31 的挂起 bit[0]用于中断 #0(异常 #16), bit[1]用于中断 #1(异常 #17), ……, bit[31]用于中断 #31(异常 #47) 写 1 将位置 0,写 0 无作用,读出值表示当前挂起状态
0xE000E284	NVIC_ICPR[1]	R/W	0	清零外部中断 #32 ~ #63 的挂起 写 1 将位置 0,写 0 无作用,读出值表示当前挂起状态
0xE000E288	NVIC_ICPR[2]	R/W	0	清零外部中断 #64 ~ #95 的挂起 写 1 将位置 0,写 0 无作用,读出值表示当前挂起状态
……	……	……	……	……

第三节 中断控制器 NVIC 的寄存器

活跃状态位寄存器

当微处理器开始执行外部中断的处理程序时，对应的活跃状态位会被置1 。活跃状态位寄存器(NVIC_IABRx) 用来表示当前正在执行的异常服务。如果有异常嵌套，置位的活跃状态位不止一个。

表 7-3-4　中断活跃状态寄存器(NVIC_IABRx)

地址	名称	类型	复位值	说　　明
0xE000E300	NVIC_IABR[0]	R	0	外部中断 #0 ~ #31 的活跃状态 bit[0] 用于中断 #0, bit[1] 用于中断 #1, ……, bit[31]用于中断 #31
0xE000E304	NVIC_IABR[1]	R	0	外部中断 #32 ~ #63 的活跃状态
……	—	—	—	—

第三节中断控制器NVIC的寄存器

中断优先级寄存器

Cortex-M 微处理器必须为中断选择一个可用的优先级配置PRI_xx ，存放在对应的中断优先级寄存器NVIC_IPRx 内。Cortex-M3/M4微处理器的中断需要配置一个包括分组 / 抢占优先级和子优先级在内的优先等级值，该值的最大宽度为8 位，最小为3 位

表 7–3–5　中断优先级寄存器（NVIC_IPRx）

地址	名称	类型	复位值	说　　明
0xE000E400	NVIC_IP[0]	R/W	0（8 位）	外部中断 #0 的优先级
0xE000E401	NVIC_IP[1]	R/W	0（8 位）	外部中断 #1 的优先级
……	—	—	—	—
0xE000E41F	NVIC_IP[31]	R/W	0（8 位）	外部中断 #31 的优先级
……	—	—	—	—